ABSTRACT

Title of dissertation: A GROUP-BASED RING OSCILLATOR PHYSICAL UNCLONABLE FUNCTION

Chi-En Yin, Doctor of Philosophy, 2012

Dissertation directed by: Professor Gang Qu
Department of Electrical and
Computer Engineering

Silicon Physical Unclonable Function (PUF) is a physical structure of the chip which has functional characteristics that are hard to predict before fabrication but are expected to be unique after fabrication. This is caused by the random fabrication variations. The secret characteristics can only be extracted through physical measurement and will vanish immediately when the chip is powered down. PUF promises a securer means for cryptographic key generation and storage among many other security applications. However, there are still many practical challenges to cost effectively build secure and reliable PUF secrecy. This dissertation proposes new architectures for ring oscillator (RO) PUFs to answer these challenges. First, our temperature-aware cooperative (TAC) RO PUF can utilize certain ROs that were otherwise discarded due to their instability. Second, our novel group-based algorithm can generate secrecy higher than the theoretical upper bound of the conventional pairwise comparisons approach. Third, we build the first regression-based entropy distiller that can turn the PUF secrecy statistically random and robust, meeting

the NIST standards. Fourth, we develop a unique Kendall syndrome coding (KSC) that makes the PUF secrecy error resilient against potential environmental fluctuations. Each of these methods can improve the hardware efficiency of the RO PUF implementation by 1.5X to 8X while improving the security and reliability of the PUF secrecy.

A GROUP-BASED RING OSCILLATOR PHYSICAL UNCLONABLE FUNCTION

by

Chi-En Yin

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2012

Advisory Committee:
Professor Gang Qu, Chair/Advisor
Professor Charles Silio
Professor Ankur Srivastava
Professor Amand Makowski
Professor Jeff Foster

Dedication

To my parents Shih-Jiun and Yung-Mei, my brother Holden and his family, Charlie and Hwan-Hwan; to Hsieh Ge, Bi-Lin Je, Wei-Wu, Ya-Jane, Fu-Tsi and dear brothers and sisters in Shin-Yo-Ton; to Alex, Jim and dear BSF leaders; to Supajohn, Tracer, Fandy, Amy, Ernie "ST. FADE"; to Yen-Ping95, NTUEE99, Omnicom99, DCNSLab01, GoldKey05; to Keep-it-Real and the Grace; to Roger, Kevin, Mark and fellow crossfit fighters; to those ever touched my life and cared about me; to those always on my mind but I choose to keep in the bottom of my heart; to myself, to the mankind, to the nature, to God.

Acknowledgments

First and foremost, I deeply thank my advisor Prof. Qu for his selfless support, the freedom and inspiration he lavished upon me. I also thank my defense committee for their service, the ECE faculty for their profound knowledge I received in class, Tracy and Melanie for their administrative support, the ECE department for seven terms of teaching assistantship, the UMD graduate school for two years of fellowship, and the Wireless Sensor Lab for the equipment to conduct my experiments. I am also in debt to my home country for two years of the Taiwan Merit Scholarship (NSC-095-SAF-I-564-056-TMS). Also, my deep gratitude to Prof. Schaumont and his team in Virginia Tech for generously sharing their experimental results with us. My gratitude extends to the entire research community; without their intellectual asset this dissertation is not possible.

Publications

- M, Taylor, C.E. Yin, M. Wu, G. Qu, "A hardware-assisted data hiding based approach in building high performance secure execution systems," Proceedings of 1st IEEE International Workshop on Hardware Oriented Security and Trust (HOST), pp. 93-96, Jun. 2008

- C.E. Yin, G. Qu, "Temperature-aware cooperative ring oscillator puf," Proceedings of 2nd IEEE International Workshop on Hardware Oriented Security and Trust (HOST), pp. 36-42, Jul. 2009

- C.E. Yin, G. Qu, "Lisa: Maximizing ro puf's secret extraction," Proceedings of 3rd IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 100-105, Jun. 2010

- C.E. Yin, G. Qu, "A regression-based entropy distiller for ro pufs," Institute for Systems Research (ISR) Technical Reports (TR 2011-12), UMCP (DAC WIP 2012)

- C.E. Yin, G. Qu, "Kendall syndrome coding (ksc) for group-based ring-oscillator physical unclonable functions (ro pufs)," Institute for Systems Research (ISR) Technical Reports (TR 2011-13), UMCP (DAC WIP 2012)

Table of Contents

List of Tables viii

List of Figures xi

1 Introduction 1
1.1 Physical Attacks . . . 2
1.2 Physical Unclonable Functions . . . 3
1.2.1 Physical Structures . . . 5
1.2.1.1 Arbiter PUFs . . . 5
1.2.1.2 Ring Oscillator PUFs . . . 6
1.2.1.3 SRAM PUFs . . . 7
1.2.2 The Typical Workflow of a Weak RO PUF . . . 8
1.2.2.1 Fabrication Variation Extraction . . . 9
1.2.2.2 Secrecy Selection . . . 9
1.2.2.3 Error Correction . . . 11
1.2.2.4 Secrecy Amplification . . . 12
1.2.2.5 Tests for Randomness and Reliability . . . 13
1.2.3 Statement of Problems . . . 13
1.2.3.1 On Hardware Efficiency . . . 13
1.2.3.2 On Security . . . 14
1.2.4 On Reliability . . . 16
1.3 Contribution . . . 17
1.4 Summary . . . 17

2 Temperature-Aware Cooperative (TAC) RO PUF 19
2.1 Preliminary . . . 20
2.1.1 The Impact of Temperature on the Frequency of ROs . . . 20
2.1.2 Temperature-Aware Bit Generation Rules . . . 20
2.2 System Overview . . . 22
2.2.1 An Example of Temperature-Aware Cooperation . . . 24
2.3 Discussion . . . 25
2.3.1 The Undecided Temperature Gap . . . 25
2.3.2 Security of the Generated Bitstring . . . 26
2.4 Validation . . . 28
2.4.1 Experiment Setup . . . 28
2.4.2 Experimental Results . . . 32
2.5 Summary . . . 32

3 LISA: Longest Increasing Subsequence Algorithm 37
3.1 Maximal Entropy of a Comparison-based RO PUF . . . 37
3.2 RO PUF Secrecy Extraction Algorithms . . . 39
3.2.1 Sequential Pairing Algorithm (SPA) . . . 39
3.2.2 Longest Increasing Subsequence Algorithm (LISA) . . . 40

3.3 Hardware Cost and Delay Analysis . . . 46
3.4 Validation . . . 48
3.4.1 Experiment Setup . . . 48
3.4.2 Experimental Results . . . 49
3.5 Summary . . . 51

4 Regression-Based Entropy Distiller 54
4.1 Security Analysis . . . 56
4.1.1 Failure Cause 1: Chain Dependency . . . 56
4.1.2 Failure Cause 2: Spatial Correlation . . . 57
4.1.3 A Cautious Note on Spatial Correlation . . . 60
4.2 Systematic Variation Elimination . . . 61
4.2.1 Modeling Fabrication Variations . . . 61
4.2.2 The Causes of Process Variation . . . 63
4.2.3 Polynomial Regression . . . 63
4.2.4 Model Selection . . . 66
4.2.4.1 Random Sequence Generation . . . 66
4.2.4.2 Tests for Randomness . . . 68
4.3 Experimental Results . . . 69
4.3.1 Random Variation Distillation . . . 69
4.3.2 Model Selection . . . 71
4.3.2.1 Random Sequences $\boldsymbol{S}$ and $\boldsymbol{T}$ . . . 71
4.3.2.2 1-out-of-8 Coding . . . 74
4.3.2.3 Neighbor Coding . . . 93
4.4 Summary . . . 94

5 Kendall Syndrome Coding (KSC) 95
5.1 Problems of the Previous Design . . . 96
5.2 Kendall Syndrome Coding (KSC) . . . 99
5.3 The Proposed Group-Based RO PUF . . . 101
5.3.1 Frequency Characterization . . . 102
5.3.2 Randomness Distillation . . . 102
5.3.3 Grouping Algorithm . . . 104
5.3.4 Kendall Syndrome Coding . . . 105
5.3.5 Conventional ECC . . . 105
5.3.6 Entropy Packing . . . 107
5.3.7 Test for Randomness and Stability . . . 107
5.4 Experimental Results . . . 109
5.4.1 Test for Randomness . . . 110
5.4.1.1 Virginia Tech Dataset . . . 110
5.4.1.2 In-House Dataset . . . 111
5.4.2 Test for Stability . . . 112
5.4.2.1 Virginia Tech Dataset . . . 113
5.4.2.2 In-House Dataset . . . 114
5.4.3 In Comparison with IBS-Based RO PUF . . . 117

5.5 Summary 120

6 Conclusion 134

Bibliography 136

List of Tables

2.1 Information stored for contributing pairs in Figure 2.2 26
2.2 Hardware cost of the logic elements in the design 31

3.1 6 Boolean equations expressed in terms b_{AB}, b_{BC} and b_{AC} 38
3.2 Values applied to the cost and delay metrics 51
3.3 Cost models of two architectures 51

4.1 NIST test results with respect to the random sequences generated by **1-out-of-8 Coding**, **Chain-like Neighbor Coding** and **Chain-like Neighbor Coding**. For **1-out-of-8 Coding**: $M = 32$ for Block Frequency Test, $m = 1$ for Approximate Entropy Test and $m = 4$ for Serial Test. For **Chain-like Neighbor Coding** and **Decoupled Neighbor Coding**: $M = 32$ for Block Frequency Test, $m = 2$ for Approximate Entropy Test and $m = 5$ for Serial Test. '*' marks a failure. 55
4.2 The results of NIST 'P-VAL. (P-VALUE of P-VALUES)' and 'PROP. (PROPORTION)' analyses with respect to random sequences generated by $\boldsymbol{S}$, $\boldsymbol{T}$, 1-out-of-8 Coding, Chain-like Neighbor Coding and Decoupled Neighbor Coding accompanied by 0^{th}- to 1^{st}-order distillers, where '*' marks a failure. 75
4.3 Cont'd with the results using $2^{nd} - 4^{th}$ order models 76
4.4 Cont'd with the results using $5^{th} - 6^{th}$ order models 77
4.5 NIST test results with respect to random sequence $\boldsymbol{S}$ using 0^{th} and 1^{st} distillers, where $M = 32$ for Block Frequency Test, $m = 2$ for Approximate Entropy Test and $m = 5$ for Serial Test and '*' marks a failure. 78
4.6 Cont'd with the results using $2^{nd} - 4^{th}$ order models 79
4.7 Cont'd with the results using $5^{th} - 6^{th}$ order models 80
4.8 NIST test results with respect to random sequence $\boldsymbol{T}$ using 0^{th} and 1^{st} distillers , where $M = 32$ for Block Frequency Test, $m = 2$ for Approximate Entropy Test and $m = 5$ for Serial Test and '*' marks a failure. 81
4.9 Cont'd with the results using $2^{nd} - 4^{th}$ order models 82
4.10 Cont'd with the results using $5^{th} - 6^{th}$ order models 83
4.11 NIST test results with respect to the random sequence generated by **1-out-of-8 Coding** using 0^{th} and 1^{st} distillers, where $M = 32$ for Block Frequency Test, $m = 1$ for Approximate Entropy Test and $m = 4$ for Serial Test and '*' marks a failure. 84
4.12 Cont'd with the results using $2^{nd} - 4^{th}$ order models 85
4.13 Cont'd with the results using $5^{th} - 6^{th}$ order models 86

4.14 NIST test results with respect to the random sequence generated by the **original Neighbor Coding** using using $0^{th} - 1^{st}$ order models, where $M = 32$ for Block Frequency Test, $m = 2$ for Approximate Entropy Test and $m = 5$ for Serial Test and '*' marks a failure. . . . 87
4.15 Cont'd with the results using $2^{nd} - 4^{th}$ order models 88
4.16 Cont'd with the results using $5^{th} - 6^{th}$ order models 89
4.17 NIST test results with respect to the random sequence generated by the **decoupled Neighbor Coding** using $0^{th} - 1^{st}$ order models, where $M = 32$ for Block Frequency Test, $m = 2$ for Approximate Entropy Test and $m = 5$ for Serial Test and '*' marks a failure. . . . 90
4.18 Cont'd with the results using $2^{nd} - 4^{th}$ order models 91
4.19 Cont'd with the results using $5^{th} - 6^{th}$ order models 92

5.1 The codebook of the rank permutations of 4 ROs using Compact Syndrome Coding (CSC), where $\{ABCD\}$ is a shorthand of the frequency relation $RO_A < RO_B < RO_C < RO_D$ 96
5.2 The estimate of H_∞^{IBS} and H_∞^{KSC} in various ECC scenarios with Virginia Tech's dataset on the left and our in-house dataset on the right 118
5.3 NIST test results derived from **the first half of the dataset** [31] using $0^{th} - 1^{st}$ order distillers, where the length of one bitstring is 400 (except 3x for FFT test), the block length for Frequency Test 32, the block length for Approximate Entropy Test 2 and the block length for Serial Test 5. '*' marks a failure. 121
5.4 Cont'd with the results using $2^{nd} - 4^{th}$ order models 122
5.5 Cont'd with the results using $5^{nd} - 6^{th}$ order models 123
5.6 NIST test results derived from **the first half of the in-house dataset** using $0^{th} - 1^{st}$ order distillers, where the length of one bitstring is 400 (except 3x for FFT test), the block length for Frequency Test 32, the block length for Approximate Entropy Test 2 and the block length for Serial Test 5. '*' marks a failure. 124
5.7 Cont'd with the results using $2^{nd} - 4^{th}$ order models 125
5.8 Cont'd with the results using $5^{nd} - 6^{th}$ order models 126
5.9 NIST test results derived from **the second half of the Virginia Teach dataset** [31] with 1^{st}-order polynomial regression model applied in Step 2. All tests are passed. 127
5.10 NIST test results derived from **the second half of our own dataset** with 1^{st}-order polynomial regression model applied in Step 2. All tests are passed 127
5.11 NIST test results derived from **the entire Virginia Tech's dataset** [31] using $0^{th} - 1^{st}$ order distillers, where the length of one bitstring is 400 (except 3x for FFT test), the block length for Frequency Test 32, the block length for Approximate Entropy Test 2 and the block length for Serial Test 5. '*' marks a failure. 128
5.12 Cont'd with the results using $2^{nd} - 4^{th}$ order models 129
5.13 Cont'd with the results using $5^{th} - 6^{th}$ order models 130

5.14 NIST test results derived from **the entire in-house dataset** using $0^{th} - 1^{st}$ order distillers, where the length of one bitstring is 400 (except 3x for FFT test), the block length for Frequency Test 32, the block length for Approximate Entropy Test 2 and the block length for Serial Test 5. '*' marks a failure. 131
5.15 Cont'd with the results using $2^{nd} - 4^{th}$ order models 132
5.16 Cont'd with the results using $5^{nd} - 6^{th}$ order models 133

List of Figures

1.1 The physical structure of a Arbiter PUF [47] 5
1.2 The physical structure of a RO PUF [47] 6
1.3 The physical structure of a SRAM PUF 8
1.4 The typical workflow of a Weak PUF 9
1.5 512 ROs placed in 16 columns by 32 rows, where $z_{x,y}$ denotes the running frequency of the RO at site $RO_{x,y}$ 10
1.6 The typical frequency-temperature relation of two ROs: (a) the result of the frequency comparison of two ROs may flip; (b) shows the case where a secret bit can be extracted reliably when the frequency gap is substantial [47]. 11
1.7 The hardware structure of the 1-out-of-8 RO PUF [47] 15
1.8 The across-die frequency topology of a 2D RO array. The roughness of the surface represents the random variations while the slope represents the systematic [44] 16

2.1 The linear relation on frequency vs. temperature for three ROs [13] . 21
2.2 Contributing pairs in cooperation to generate reliable bits: V: 1^{st} RO in the pair is sufficiently faster than 2^{nd}; X: 2^{nd} RO in the pair is sufficiently faster than 1^{st}; ?: none of the RO is sufficiently faster than the other. 25
2.3 Schematics: The ROs (above row) and ripple counters (center) are realized through LUTs and flip-flops, primitive elements on Xilinx FPGAs. 29
2.4 Symmetric placement: The ROs are placed symmetrically in 32 columns divided in two rows. 30
2.5 Symmetric layout: The symmetry of two target ROs placed at two CLB columns C12 and C14 is crafted manually in Xilinx FPGA Editor. 34
2.6 Cell classification vs. reliability threshold: Data are collected from 144 pairs generated by our 1-out-of-2 scheme, number of cells identified in each classification. 35
2.7 Cell cooperation vs. reliability threshold: It shows the number of reliable bits (left axis) generated by the contributing pairs (right axis) as δ (x-axis) changes. 35
2.8 Cost performance vs. reliability threshold: performance gain from our 1-out-of-2 scheme compared with the 1-out-of-8 benchmark. . . . 36

3.1 The hybrid architecture with hybrid coefficient $h = 4$. 47
3.2 Identical logic layout by manual place-and-route of two ROs 49
3.3 The total number of reliable unclonable bits k discovered out of 288 target ROs on 9 Xilinx XC4010XL FPGAs 50
3.4 $h - c$ characteristic 52
3.5 $h - d$ characteristic 52
3.6 The cost-performance gains of SPA and LISA vs. the benchmark . . . 53

4.1 The placement of 512 ROs as a 16 (columns) by 32 (rows) array; for site $RO_{x,y}$, its running frequency is labeled $z_{x,y}$ 57
4.2 The across-die frequency topology of a RO array. The roughness of the surface represents the random variation while the slope represents the systematic [44] 58
4.3 Illustration of the impact from systematic variation even after pairs are decoupled. 59
4.4 The typical workflow of a Weak RO PUF augmented with a new entropy distillation step marked in dashes 61
4.5 The fabrication variation of Chip No. 1 [31] with the z-axis indicating the average value of the 100 frequency readings of each RO site. . . . 70
4.6 The modeled systematic variation after applying 0^{th} through 6^{th}-order polynomial regression to the dataset of Chip No. 1. 70
4.7 The distilled random variation after applying 0^{th} through 6^{th}-order polynomial regression to the dataset of Chip No. 1. Notably, we see the 'bull's eye', i.e., the radial pattern close to the center, vanishing in the cases of 2^{nd} order model and beyond. 71
4.8 The histogram of the distilled random variation after applying 0^{th} through 6^{th}-order polynomial regression to the dataset of Chip No. 1. It is difficult to judge simply from the chart which model is the best fit without running NIST tests for all appears normal but with a diminishing variance as the order increases 72

5.1 The architecture of the proposed group-based RO PUF 101
5.2 The estimated H_∞^{CSC} and H_∞^{KSC} derived with BCH(31, k, t) based on the Virginia Tech dataset 114
5.3 The estimated H_∞^{CSC} and H_∞^{KSC} derived with BCH(63, k, t) based on the Virginia Tech dataset 115
5.4 The estimated H_∞^{CSC} and H_∞^{KSC} derived with BCH(127, k, t) based on the Virginia Tech dataset 115
5.5 The estimated H_∞^{CSC} and H_∞^{KSC} derived with BCH(31, k, t) based on the in-house dataset 116
5.6 The estimated H_∞^{CSC} and H_∞^{KSC} derived with BCH(63, k, t) based on the in-house dataset 117
5.7 The estimated H_∞^{CSC} and H_∞^{KSC} derived with BCH(127, k, t) based on the in-house dataset 117
5.8 The estimated H_∞^{IBS} derived from the Virginia Tech dataset 119
5.9 The estimated H_∞^{IBS} derived from the in-house dataset 119

Chapter 1

Introduction

"He [the beast] also forced everyone, small and great, rich and poor, free and slave, to receive a mark on his right hand or on his forehead, so that no one can buy or sell unless he had the mark. (Revelation 13:16)"

In the era of cloud computing, numerous apps are at fingertips anytime anywhere. But to have the peace of mind of using mission critical apps like online banking, sensitive data such as credit card and bank account information, billing address, and social security number must be in good hands. Before granting access to such information, the cloud server has to know which client device, like the iPhone, it talks to; likewise, the client device has to ensure from which source the information comes from before it displays to the end user. Modern cryptography allows us to do this by, for instance, signing and encrypting the information with a shared key or a public-private key pair. Due to the importance and security concerns of these cryptographic keys, they are often made very long, too long for a human being to remember. An alternative is to store these keys in integrated circuits (ICs). When the user wants to engage in a security-sensitive transaction like paying a bill, she can prove her identity by entering her password to the device; once authorized, the ICs inside the device will perform the requested functionality on her behalf.

The importance of keeping the keys safe and security has long been recognized and summarized in the Kerckhoffs' principle: "The security of a cryptosystem must not depend on keeping secret the crypto algorithms. The security depends only on keeping secret the key (translated from La cryptographie militaire (1883))."[1]

In the past decade, Silicon Physical Unclonable Function (PUF), which is a physical structure of the chip which has functional characteristics that are hard to predict before fabrication but are expected to be unique after fabrication, has been developed as a securer means for cryptographic key generation and storage among many other security applications. This dissertation proposes new architectures and practical techniques to cost effectively build secure and reliable "PUF secret."

1.1 Physical Attacks

One may think that the keys stored in the physical devices are safe. However, it is still possible for an attacker to crack crypto keys out of physical devices. In fact, not all hardware devices are designed to serve as a security safe for crypto keys. Attackers may manipulate the supply voltage while repeatedly writing to the security bit. Chances are that over-voltage may clear the check bit but keep rest of the memory intact. Or with under-voltage, they can trigger irregular output like all 0's out of a random number generator. Attackers can also manipulate the clock signal to skip certain instructions for security checks or inject faults to RAM cells by exposing them to a magnified flashlight. Ultimately, the adversary can de-

[1] another translation: "A cryptosystem should be secure even if everything about the system, except the key, is public knowledge. (1883)"

package the IC, remove the passivation layer, drill into the inner layers of the circuit if necessary, so as to probe directly on signal lines with the help from advanced semiconductor mechineries such as Scanning Electron Microscope (SEM) and/or Focused Ion Beam (FIB).

To sustain physical attacks, modern crypto modules are often equipped with intricate security measures such as metallic mesh shields to detect physical drilling, capacitive sensors to ensure the passivation layer is intact, optical sensors to detect the existence of persistent opaque coating and other sensors to guarantee the regularity of power supply, clock frequency and ambient temperature, etc. Devices can self-destruct the key whenever it is assumed to be in danger. These hardened devices are called tamper-resistant devices, examples of which are ubiquitous, such as chip cards for banking, metro, cellphone, building access, just to name a few. More sophisticated examples include the Trusted Platform Modules (TPM) for modern PCs and servers, and Hardware Security Modules (HSM) at the backend datacenter to handle billions of transactions. Nevertheless, in the tussle on physical security, there is no definitive winner between the attacker and defender [3, 2, 34, 4, 45, 46, 28]. Next we will introduce Physical Unclonable Function (PUF) as a novel security primitive for safekeeping of our crypto keys.

1.2 Physical Unclonable Functions

Even though the notion of *tamper-proof* appears theoretically unattainable, one may not be too concerned about storing her key in a *tamper-resistant* device

as long as the expected cost to crack a key out of the device is exorbitantly high enough. This is how we have intricate security devices and expensive certification processes such as FIPS, VISA and Common Criteria to establish the trust from the society. For security researchers, however, the war never stops and we have to keep looking out for new security measures before successful attacks ever come close. As we see, it requires system power to enforce all sorts of security measures and to erase the key immediately whenever necessary. But for devices without persistent power like smartcards, they store keys in non-volatile memory when the power goes away. Since most security measures are inactive when the power is off, tamper-resistant devices without persistent power are generally considered less secure than those with persistent power. This motivated the study of Physical Unclonable Function (PUF) as a new security primitive, which, as opposed to storing keys in non-volatile memory, stores keys in intrinsic electrical disorder such as asymmetry of two wires of identical design or mismatch in the threshold voltages of two identical transistors caused by uncontrollable fabrication variations like the focus shift of photolithography, the gradient of thermal annealing, random dopant profiles, and so on [21, 6, 12]. To state it formally, *PUF is a function that is embodied in a physical structure and is easy to evaluate but hard to predict* and can be regarded as the fingerprint or DNA of the IC. The underlying security assumption is that *it is practically infeasible to physically tamper a PUF without damaging its nano-scale disorders, i.e., the keys.* As such, PUF promises a higher level of security especially when keys cannot be protected with persistent power.

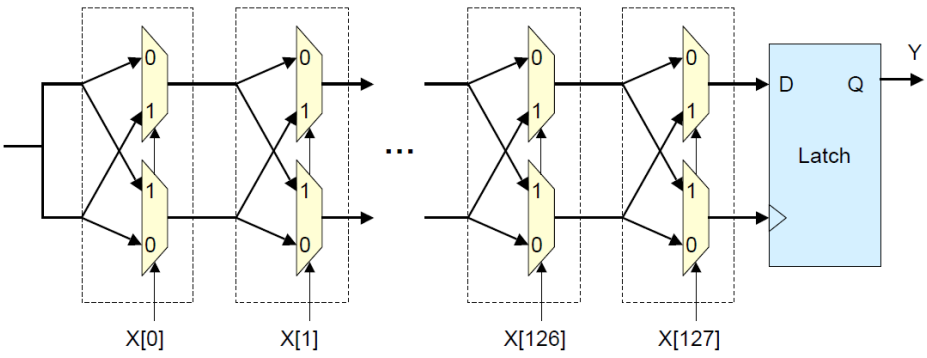

Figure 1.1: The physical structure of a Arbiter PUF [47]

1.2.1 Physical Structures

Since its inception in 2001 [36], PUF has been a very active research topic and various physical structures have been discovered over the past decade, including Optical PUFs [36, 38], Arbiter PUFs [8, 22, 19, 23, 40, 32, 31], Ring Oscillator (RO) PUFs [8, 47, 32, 56, 31], Coating PUFs [9, 37, 10], SRAM PUFs [24, 25, 17, 39], Butterfly PUFs [41], RST PUFs [1], and so on. For silicon PUFs, the most notable are Arbiter PUFs, RO PUFs and SRAM PUFs and we will introduce how each operates.

1.2.1.1 Arbiter PUFs

Arbiter PUF is the first silicon PUF realized in electrical circuitry. Figure 1.1 shows its physical architecture with 128 stages of multiplexers and one D flip-flop at the end. The 128-bit challenges X [0 ... 127] configures the 128 pairs of multiplexers that sets up two signal lines of the same length from the same source, one reaching the D input of the flip-flop in the end and the other reaching the CLK

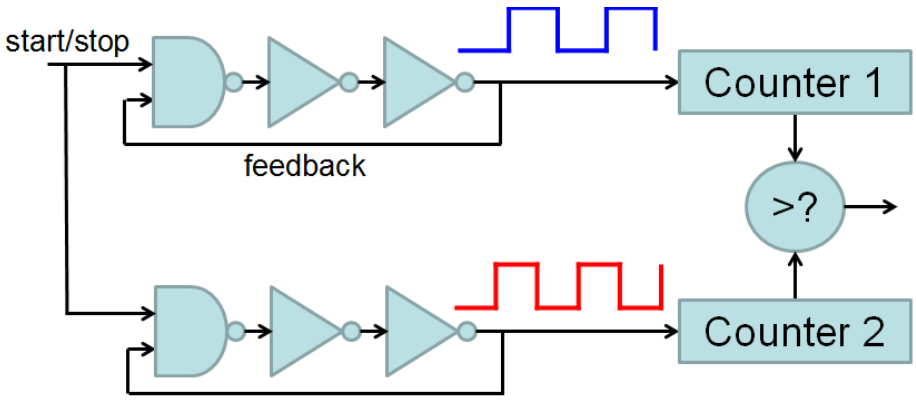

Figure 1.2: The physical structure of a RO PUF [47]

input. Due to uncontrollable fabrication variations, when logic one is asserted at the very beginning of the signal lines, one of the flip-flop input will be asserted faster than the other, yielding the output Q '1' if the D is asserted faster than the CLK, otherwise '0' if the other way around. To generate a reliable output, the timing difference between the two assertions has to satisfy the setup time and hold time of the flip flip. Arbiter PUF requires symmetric routing and therefore suitable for ASIC design; due to its small hardware footprint and its large number of challenge and response pairs (CRPs), Arbiter PUF has found a great application in low-cost authentication for RFIDs [40]. However, without due protection [7], it has to shown that Arbiter PUF is susceptible to modelling attacks [49].

1.2.1.2 Ring Oscillator PUFs

Another physical structure to realize the notion of PUF is through comparing the frequency readings of a pair of ring oscillators, see Figure 1.2. Similar to Arbiter PUFs, it is a race between two signals but the competition magnifies the speed/delay

difference by feedback loops. The winner goes to the one recording more circles in the counter as the referee stops the watch. The binary output encodes the result of the match, say, '1' when the top RO wins, else '0'. For a N-stage RO loop, the total propagation delay d_T can be modelled as the sum of the delays of each stage (an inverter or a buffer); it is typically assumed that each stage has the same nominal delay A and the stochastic fabrication variation for each stage is normally distributed with zero mean and variance σ^2. Thus, in [44], the total delay and the relative standard deviation of the stochastic variation are expressed as

$$d_T = NA + N\sigma^2 \tag{1.1}$$

$$\frac{\sqrt{N}\sigma}{NA} = \frac{1}{\sqrt{N}}\frac{\sigma}{A} \tag{1.2}$$

From Eqn. 1.2, we see it is more desirable for RO PUFs to have a smaller number of oscillator stages. Compared with Arbiter PUF, RO PUF is easier to implement on FPGAs and more reliable but slower, larger and consume more power [47]. Recently, researchers also proposed a hybrid architecture to incorporate the merits of both [56].

1.2.1.3 SRAM PUFs

SRAM PUFs emerged later than the previous two. They harvest the initial start-up values of SRAM cells to form their intrinsic secrecy. As shown in Figure 1.3, each SRAM cell consists of 6 transistors, where (M_1, M_2) and (M_3, M_4) are two

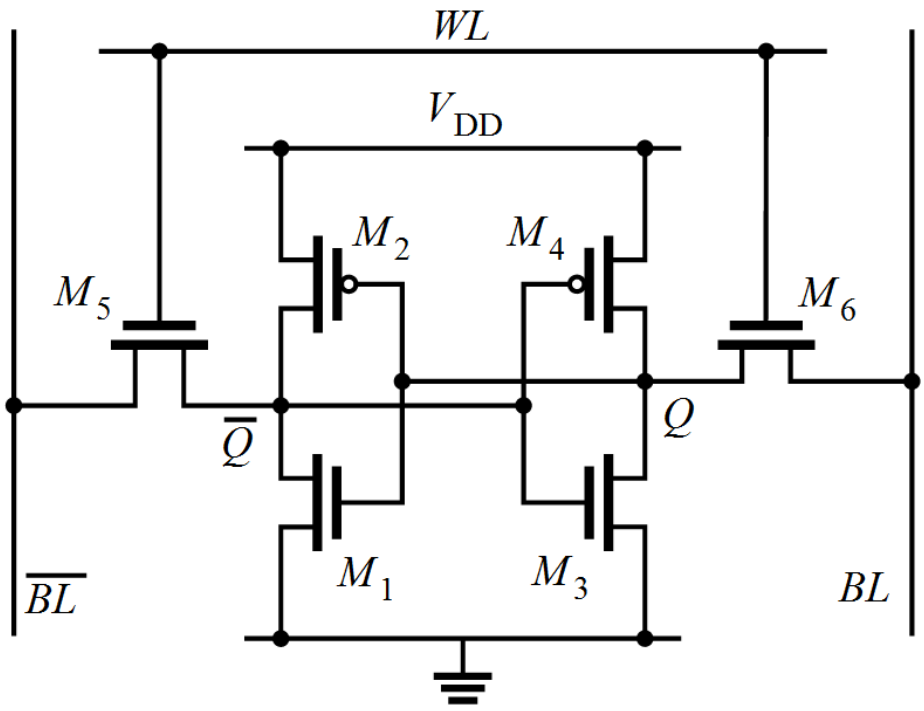

Figure 1.3: The physical structure of a SRAM PUF

coupled inverters to reinforce the value in storage. To access (read and write) the value, WL shorts Q and BL as well as $\overline{Q}$ and $\overline{BL}$ through M_5 and M_6. For SRAM PUFs, only the read operation is involved. Compared with RO PUFs, SRAM PUFs are faster, consume less power yet require a stronger error correcting scheme to produce reliable secrecy. Related works can be found in [18, 25, 24, 17, 39].

1.2.2 The Typical Workflow of a Weak RO PUF

Some researchers further classify PUFs as 'Strong' and 'Weak', though the words *Strong* and *Weak* are irrelevant to the security strength [49] but only refer to the difference in the number of challenge-response pairs (CRPs) a PUF can generate. In fact, the distinction can be insignificant since a large number of CRPs can

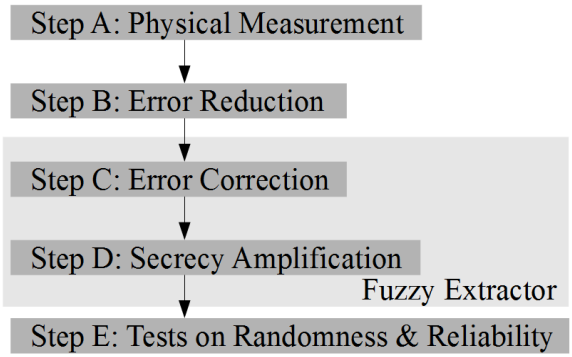

Figure 1.4: The typical workflow of a Weak PUF

be generated by a crypto function keyed with a 'Weak' PUF. For simplicity and clarity, however, we use 'Weak' PUFs as the context of our discussion, though our methodologies are expected to work for 'Strong' PUFs as well with certain modifications, e.g., an Arbiter and RO hybrid. Next, as outlined in Figure 1.4, we walk through the steps typically involved in a 'Weak' RO PUF.

1.2.2.1 Fabrication Variation Extraction

The first task is to measure the unique characteristic determined by the uncontrollable fabrication process. For a typical RO PUF, this step corresponds to the frequency characterization of a 2-dimension RO array as illustrated in Figure 4.1, where 512 ROs are placed as a 16 columns by 32 rows grid. One of the dataset we use derives such characterizations from 125 FPGA devices [31].

1.2.2.2 Secrecy Selection

The goal of the step is to select secure and reliable information out of the frequency profile measured in the previous step. While the maximal extractable

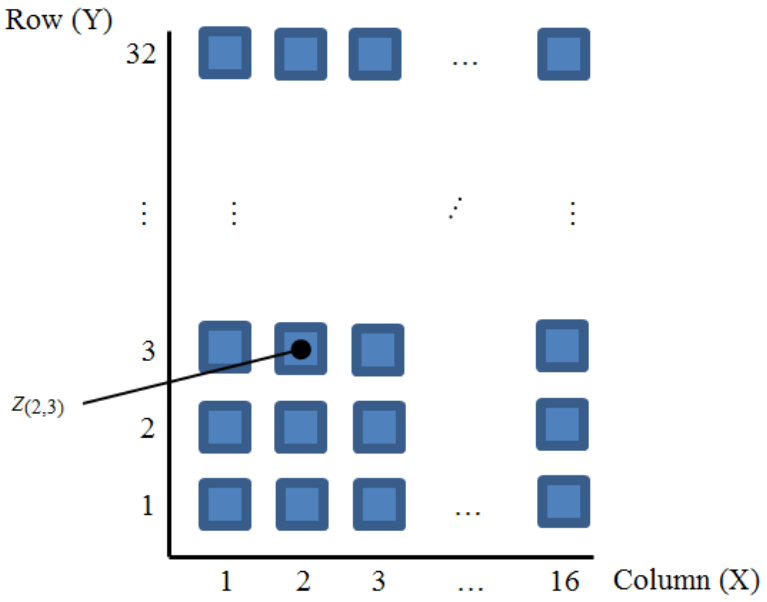

Figure 1.5: 512 ROs placed in 16 columns by 32 rows, where $z_{x,y}$ denotes the running frequency of the RO at site $RO_{x,y}$

entropy out of n ROs is $\log_2 n!$ when pairwise comparisons are assumed [47], the usable entropy is less than the bound when we have stringent constraints on output randomness and resilience against ambient variations. For instance, the most prominent reliability issue of RO PUFs is that the running frequency of ROs decreases as the ambient temperature increases but the decreasing rate varies from RO cell to RO cell. As Figure 1.6 (a) illustrates, the RO marked blue runs faster than the RO marked green at low temperature but the relation flips at high temperature. To avoid such a cross-over, we may select those pairs with a significant frequency gap like the case of Figure 1.6 (b). The most notable secrecy selection schemes are the classic 1-out-of-8 coding [47], which selects the most reliable pair out of 8 pairs, its successor index-based syndrome (IBS) coding [56] and the chain-like neighbor coding [32, 31, 20].

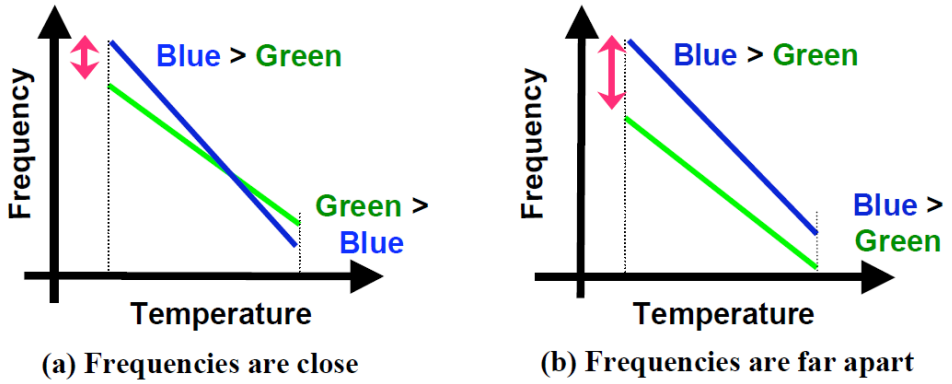

Figure 1.6: The typical frequency-temperature relation of two ROs: (a) the result of the frequency comparison of two ROs may flip; (b) shows the case where a secret bit can be extracted reliably when the frequency gap is substantial [47].

1.2.2.3 Error Correction

In real-life deployment, environmental conditions can vary as large as 25% [56], whereas the runtime error rate target is as low as 10^{-6}. To attain such a low error rate, solely relying on error reduction schemes in the previous step may not be efficient. Alternatively, one may also apply error correcting codes (ECC) to contain random errors. The first ECC considered in PUF literature is 2D Hamming code [8] but later on researchers acknowledged the the noisy nature of PUFs and considered stronger ECCs such as BCH codes [47, 24]. The basic operation is the following: To enroll the output from the previous step as a secret, it is fed to an ECC encoder as information bits to generate parity bits as public helper data, which is then stored publicly in non-volatile memory and will be used to facilitate secrecy regeneration, or called secrecy recovery or secrecy reconciliation. To regenerate the secrecy enrolled in the first place, we perform ECC decoding with the stored public helper data and

new fuzzy input from the previous secrecy selection step. As long as the new fuzzy input is close enough to the secrecy (input) enrolled in the first place, the enrolled secrecy can be recovered correctly. While SRAM PUFs made most of their advance in finding the most efficient ECC implementation, RO PUFs had their breakthrough in devising their syndrome coding schemes, a technique to represent the secrecy in a more error-resilient form. It has been reported that the error rate of a RO PUF can be lower than 1 ppm (part per million) using IBS syndrome coding followed by a 3× repetition code and a BCH(63) code even under severe environmental changes (-55°C, 1.1V to 125°C, 0.9V) [56].

1.2.2.4 Secrecy Amplification

Since real world entropy sources always demonstrate some 'imperfection' in forms of bias and correlation, further entropy distillation, or often called secrecy amplification in PUF-related works, is necessary to make the output appear to be in uniform distribution. Uniform distribution is regarded 'ideal' for it yields the maximal entropy of a sequence of random variables. Given a non-uniform bitstring containing certain min-entropy, a universal hash function (UHF) can transform it into another bitstring in uniform distribution but of shorter length. Roughly speaking, Fuzzy Extractor is a security primitive consisting of the functionality of error correction, e.g. ECC, and the functionality of secrecy amplification, e.g., UHF [53].

1.2.2.5 Tests for Randomness and Reliability

The security of a random number generator (RNG) can be judged by the statistical characteristic of random sequences it produces. The NIST test suite [5] is designed to serve this purpose and is widely regarded as the standard for the security industry. Reliability, on the other hand, can be gauged by subjecting PUF devices to extreme operating conditions in terms of temperature and supply voltage. In general, the target failure rate is below 1 ppm [56].

1.2.3 Statement of Problems

This dissertation will improve upon the typical workflow we have just described. In particular, we will provide original methodologies to better address the issues of reliability, security and hardware efficiency. Although the three issues are sort of related, for clarity we state the problems separately and point to our solutions in later chapters.

1.2.3.1 On Hardware Efficiency

To deal with the crossover issue in Figure 1.6 (a), 1-out-of-8 Coding [47] and index-based syndrome (IBS) Coding [56] aim at generating one reliable bit out of 8 hardware-wired RO pairs as shown in Figure 1.7. Only the pair with the largest frequency difference will be selected for use and the rest will be discarded. There is good chance that the selected pair will yield a reliable result in terms of frequency comparison across all environmental conditions, see Figure 1.6 (b). The information

about which pair is selected can be disclosed to the public without hindering the security, assuming the probability of one RO runs faster than the other is 1/2. As we see, the enhancement on reliability is at the cost of hardware efficiency since the scheme in the best case can only generate $n/8$ bits given n pairs of ROs.

Another popular coding scheme is the Chain-Like Neighbor Coding [32], which pairs a RO with its neighbors like a chain, i.e., (RO_1, RO_2), $(RO_2, RO_3) \ldots (RO_{n-1}, RO_n)$. For reconfigurable devices like FPGA, Chain-Like Neighbor Coding may search through different combinations of look-up tables (LUTs) and select the most reliable configuration for secret regeneration. Given n ROs, the best case scenario improves from $n/16$ bits using the 1-out-of-8 Coding to $n - 1$ bits; however, it is still falls short of the theoretical upper bound $\log_2 n!$. To further improve on hardware efficiency, we provides two original designs. Firstly, Chapter 2 describes a new Temperature-Aware Cooperative (TAC) RO PUF that takes advantage of the linear frequency response of a RO as the temperature varies. Secondly, in Chapter 3, the proposed Longest Increasing Subsequence Algorithm (LISA) approaches the $\log_2 n!$ theoretical bound by forming groups made of two or more ROs.

1.2.3.2 On Security

While main objective of PUFs is to produce an unpredictable but reliable bitstring from uncontrollable fabrication variations. However, the entropy source of natural phenomena may also contain certain systematic bias that forbids direct cryptographic uses. As Figure 4.2 shows, the roughness of the surface (pure ran-

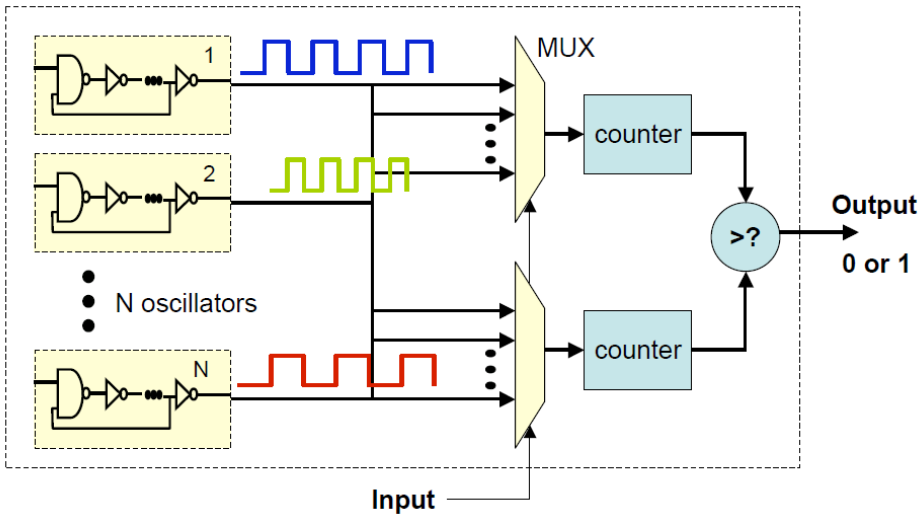

Figure 1.7: The hardware structure of the 1-out-of-8 RO PUF [47]

dom variations) is superimposed upon a spatial trend (systematic variations). The systematic component can significantly reduce the min-entropy of the extracted secrecy such that one may not be able to generate 1-bit worth secrecy out of one RO pair. To see this, in Figure 4.2, if one generates bit A as '0' if $A1 < A2$, else '1', bit B '0' if $B1 < B2$, else '1', the strong spatial correlation would render $p(A = B) >> p(A \neq B)$. In light of the security threat, the Chain-Like Neighbor Pairing suggests that 1) place ROs as close as possible and 2) pair only ROs located adjacent to each other, such that the systematic factor would cancel out in comparison [32]. Unfortunately, the solution causes another security problem in the chain, rendering its secret bits not independent and identically distributed (IID) uniform. These observations lead to the development of our solution in Chapter 4: the Regression-Based Entropy Distiller.

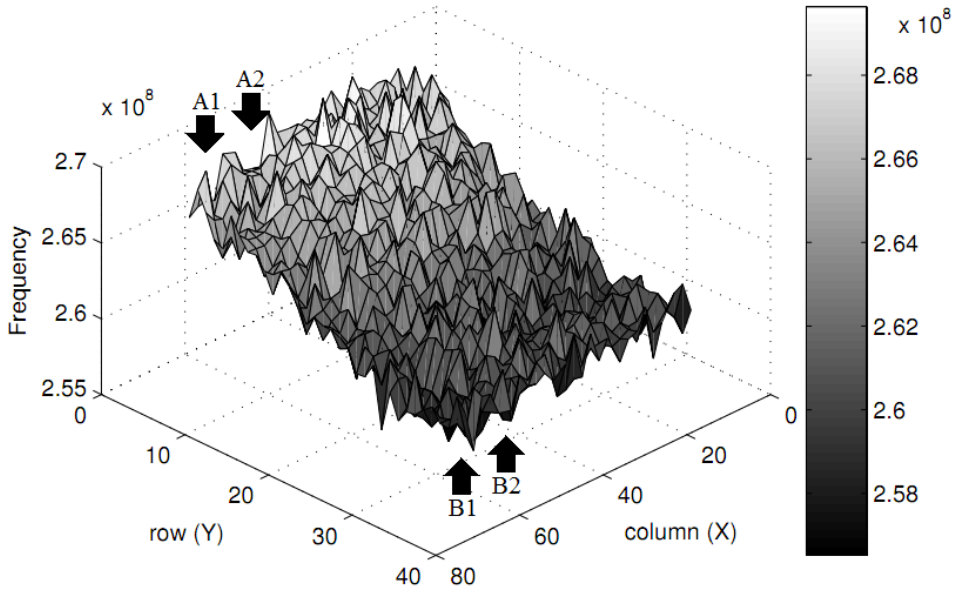

Figure 1.8: The across-die frequency topology of a 2D RO array. The roughness of the surface represents the random variations while the slope represents the systematic [44]

1.2.4 On Reliability

For a $(n,\ k,\ t)$ linear error correcting code (ECC), the more the correctable errors t, the more the min-entropy loss $n-k$, where k is the number of information bits in a n-bit code block. In other words, the increase in ECC error correction capability will either increase the complexity of ECC by using a larger code block n or increase the min-entropy loss t per n-bit block or both. Instead of devising the most hardware efficient ECC, we develop a more error-resistant syndrome coding scheme based on Kendall tau distance to further reduce the hardware cost of our group-based RO PUF. When various BCH codes are assumed, the proposed Kendall Syndrome Coding (KSC) generally can achieve a lower error rate when compared

with a more intuitive approach based on a compact codebook, see Chapter 5.

1.3 Contribution

This dissertation contributes several original methodologies to improve RO PUFs in terms of security, reliability and hardware efficiency. First, we devise the temperature-aware cooperative (TAC) RO PUF to recycle the ROs that were discarded previously due to crossover's. Second, we design the algorithmic solver LISA to harvest the maximal entropy extractable in comparison-based RO PUFs. Third, we demonstrate certain weaknesses in the statistical characteristics of existing RO PUFs and provide a cure through the proposed regression-based entropy distiller. Further, we identify the coding inefficiency if one were to encode the results of comparisons of RO frequencies compactly; instead, we propose a more error-resistant coding scheme based on Kendall tau distance to reduce the complexity of the error correcting code (ECC) in the company. Lastly, we put all together and implemented the proposed group-based RO PUF as an embedded system on modern FPGAs; extensive experiments have also been conducted and detailed test results are reported to support our claims.

1.4 Summary

In this chapter we introduced the background of hardware security and how PUFs can provide a higher level of security when applied to current tamper-resistant devices. We also mentioned three most remarkable PUF constructions that can be

realized on integrated circuitry and walked through the typical steps in order to produce secure and reliable PUF output. Based on that, we stated the problems that will be addressed by the remaining of the dissertation, which is thus organized: Chapter 2 describes a method that recycles those unusable ring oscillators to enhance the hardware efficiency; Chapter 3 presents LISA, an algorithmic approach that derives the secrecy from groups of ring oscillators. In such a group-based setting, Chapter 4 tackles the security problems caused by spatial correlations. Chapter 5 further reduces the complexity of ECC with Kendall syndrome coding (KSC); moreover, we put together the concepts in previous chapters to construct the grand design of our group-based RO PUF. Finally, we conclude in Chapter 6.

Chapter 2

Temperature-Aware Cooperative (TAC) RO PUF

Previously, we mentioned that RO PUFs can generate a reliable unclonable secret through comparing two amplified delays determined by uncontrollable fabrication variations. Given RO1 and RO2, we can define that the pair generates bit '1' if RO1 is faster than RO2, otherwise '0'. We also said that not all pairs of ROs can generate one reliable unclonable bit as the ambient condition changes. In particular, the temperature has a significant impact on frequency such that one RO faster at low temperature may become slower than its peer at high temperature. Previous wisdom suggested to pick only those pairs with frequency differences large enough to avoid frequency crossovers throughout the temperature range of concern. As we see, the solution has a high hardware price tag on it. Rather than trading hardware efficiency for reliability, this chapter describes a more cost-effective approach. Through cooperation, we can generate a reliable bit from two pairs of ROs previously discarded for lack of unreliable. Empirical results showed that the proposed mechanism can improve the hardware efficiency by 80% when compared with the classic approach.

2.1 Preliminary

2.1.1 The Impact of Temperature on the Frequency of ROs

Literature [13] showed that every 15°C temperature increase will roughly increase the delay of a RO by 10%–15%. As such, temperature is critical factor that affects the reliability of RO PUFs. Figure 2.1 shows the experimental results of three RO-based temperature sensors, each of them centered at a different frequency by being assigned with a different type of FPGA interconnect. As we see, all ROs run faster in low temperature and slow down in high temperature. An important observation is that the frequency of ROs respond linearly as temperature changes. The linear dynamic between the running frequency of ROs and their operating temperature has also been reported in other literature like [42]. For RO PUFs, as long as two running frequencies never cross each other throughout the temperature range of concern, we can use the pair to generate a reliable bit with a frequency comparator. But if the speed of one RO drops much faster than its peer like s1 and s2 in Figure 2.1, a crossover may occur and result in an unreliable secret bit.

2.1.2 Temperature-Aware Bit Generation Rules

Due to random measurement errors, when two running speeds of a RO pair get very close, say, less than certain threshold f_{th}, we may not be able to derive consistent results regarding which RO runs faster than the other. The unreliable region, or undecided temperature gap, is denoted by $[k_{low},\ k_{high}]$. When the temperature at the time of measurement falls outside the region of $[k_{low},\ k_{high}]$, we

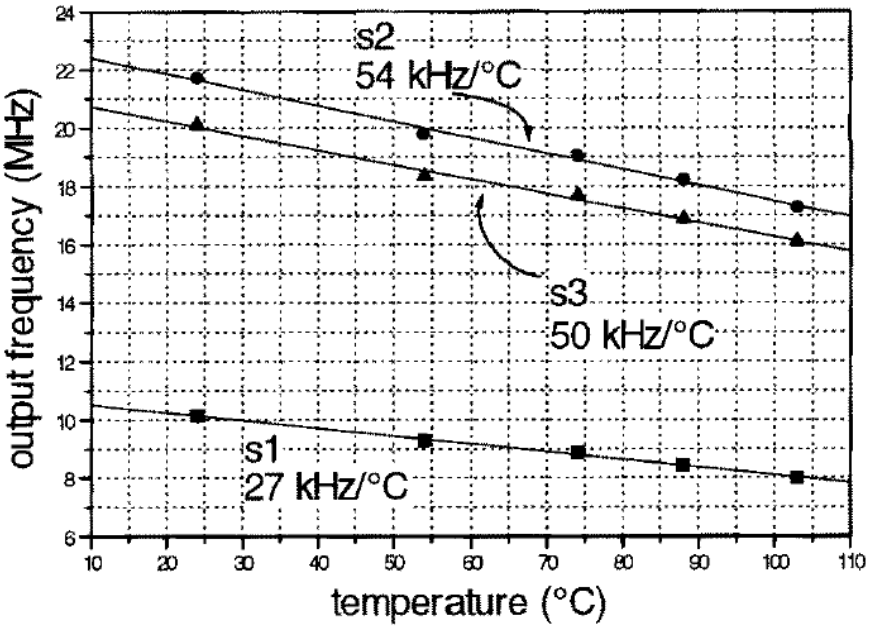

Figure 2.1: The linear relation on frequency vs. temperature for three ROs [13]

may generate a reliable bit by obeying the temperature-aware rules R1 and R2 as below, assuming the chip can tell the temperature at the time of measurement via a built-in temperature sensor. In case the temperature at the time of measurement falls into the the region $[k_{low},\ k_{high}]$ of certain pair, say, RO1 and RO2, the pair can not generate a reliable bit; when this happens, we turn to some other pair, say, RO3 and RO4, to supply the secret bit that is supposed to be generated by RO1 and RO2. In other words, we use two or more pairs to work out one reliable bit in a cooperative way.

- **R1.** If the operating temperature k°C is lower than k_{low}, i.e., k°C$< k_{low}$, the bit generated is '0' if RO1 is *faster* than RO2, otherwise '1'.

- **R2.** If the operating temperature k°C is higher than k_{high}, i.e., $k_{high} < k$°C,

the bit generated is '0' if RO1 is *slower* than RO2, otherwise '1'.

2.2 System Overview

- **Deployment** In the logic design phase, suppose there are n pairs of ROs and each RO is directly connected to a counter; each pair is connected to a software or hardware comparator. Manual placement and route is performed when necessary to ensure the symmetry of the RO pair such that the uncontrollable fabrication variations dominate the running speed of RO pairs.

- **Measurement** In the wafer testing phase, we build the linear frequency-temperature model for all ROs through the readings in the frequency counters sampled at two temperature points. For better resolution, more samples can be acquired as needed. As modern processors can heat themselves up quickly, acquiring frequency samples at two different temperature points is not expected to introduce notable delay to the testing process.

- **Enrollment** In the wafer testing phase, we also classify each RO pair, say, RO1 and RO2, as:

  - **A good pair:** If RO1 (or RO2) is sufficiently faster than RO2 (or RO1) throughout the entire range of the concerned operating temperature, we define this to be a bit '1' (or '0'), respectively. This is a reliable bit.

  - **A bad pair:** If the delay of RO1 and RO2 are not sufficiently apart to define a reliable bit at anywhere in temperature of concern, this pair of

RO is totally unusable and will be disconnected from the power source to save power.

  - **A contributing pair:** If the delay difference of RO1 and RO2 is large enough to define a bit only in a certain interval of the operating temperature, we use the bit generation rules R1 and R2 to generate a reliable bit in this reliable interval and rely on cooperation with other RO pairs to generate the same bit when the temperature is beyond. Although cooperation involving multiple good, bad, and contributing pairs are possible, for simplicity our discussion only considers the case between two contributing pairs.

- **Secrecy Generation** Still in the wafer testing phase, assuming we have a number g of good pairs, b bad pairs and c contributing pairs, where $g+b+c = n$, for those good pairs, we can derive g reliable bits b_i where $i = 0, 1, \ldots, g-1$; for bad pairs, no bit can be defined; for contributing pairs, we can come up with at best $c/2$ bits by forming pairwise cooperation among the contributing pairs. The secret bits can be treated as a cryptographic key per se or as a seed to a random number generator (RNG) for future key generation. To further enhance on reliability and secrecy, error correcting codes (ECC) or a Fuzzy Extractor can be applied accordingly [47, 25, 41].

2.2.1 An Example of Temperature-Aware Cooperation

While contributing pairs cannot generate a reliable bit by itself, they can work together to produce reliable bits. As the example illustrated in Figure 2.2, we have 6 pairs of ROs and each has its own undecided temperature gap $[k_{low},\ k_{high}]$, where the delay difference of that pair is not enough to tell consistently which RO is faster than the other. The operating temperature we are concerned is between k_{min} and k_{max}. If the temperature at the time of measurement is k_1, since k_1 is lower than all the k_{low}'s, we can consistently observe the speed difference of each pair and derive a 6-bitstring '010001' by applying the temperature-aware bit generation rule R1. Similarly, when the temperature at the time of measurement is k_2, since k_2 does not fall into any undecided gap $[k_{low},\ k_{high}]$, we can obtain the same bitstring by applying R1 to Pair 4 and R2 to the rest 5 pairs. But if the temperature at the time of measurement is k, the bitstring we get would be '0?00?1', where two ?'s indicate Pair 2 and Pair 5 cannot make a reliable decision because the operating temperature k falls into their undecided temperature gap. Note that since the undecided temperature gap of Pair 2 does not overlap with the gap of Pair 4, we know the operating temperature can never fall into both undecided temperature gaps at the same time and therefore, the two pairs can work together to generate a reliable bit throughout the entire temperature range we are concerned. That is, when the temperature at the time of measurement falls into Pair 2's undecided temperature gap, we know that Pair 4 can produce the reliable bit '1' on behalf of Pair 2 after complementing its output '0'. Likewise, we can pair up Pair 1 and 6, Pair 3 and 5 for cooperation. Table

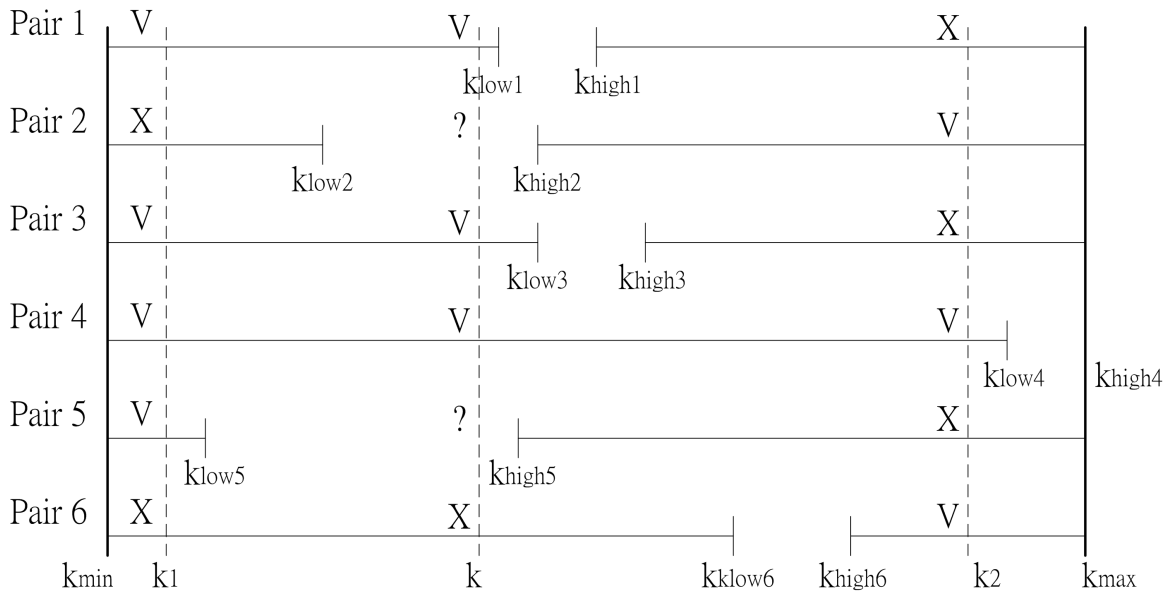

Figure 2.2: Contributing pairs in cooperation to generate reliable bits: V: 1^{st} RO in the pair is sufficiently faster than 2^{nd}; X: 2^{nd} RO in the pair is sufficiently faster than 1^{st}; ?: none of the RO is sufficiently faster than the other.

2.2.1 lists the information stored in the non-volatile memory that enables reliable regeneration of the 6-bit secret '010001' as long as the operating temperature is within the predefined range k_{min} and k_{max}. The disclose of the helper data does not reveal the hidden secret '010001' as long as the probability for one RO to run faster than its peer is 1/2.

2.3 Discussion

2.3.1 The Undecided Temperature Gap

Now let's quantitatively define the term 'sufficiently large' we have used so far in describing the delay difference between two ROs of a pair. Consider RO1 and RO2

Pair Number	**Gap Low**	**Gap High**	**Backup Number**	**Complement**
1	k_{low1}	k_{high1}	6	Yes
2	k_{low2}	k_{high2}	4	Yes
3	k_{low3}	k_{high3}	5	No
4	k_{low4}	k_{high4}	2	Yes
5	k_{low5}	k_{high5}	3	No
6	k_{low6}	k_{high6}	1	Yes

Table 2.1: Information stored for contributing pairs in Figure 2.2

and their counter readings N1 and N2. The two ROs start and stop simultaneously, controlled by a reference counter with reading 0 (start) to N (stop). We say the delay difference of RO1 and RO2 is sufficiently large if and only if $|N1 - N2| \geq \delta N$, where δ is a reliability threshold factor, a predefined constant. Clearly, the larger the δ, the more reliable the extracted bit. Due to the delay-temperature linearity of ROs, we write the counter readings of RO1, RO2 and the reference RO as $a_1k + b_1, a_2k + b_2$, and $ak + b$, respectively. With $(a_1 - a_2)k + (b_1 - b_2) = \pm\delta N$ and two boundary conditions $k_{min} \leq k_{low}$ and $k_{high} \leq k_{max}$, we can solve for k_{low} and k_{high} of the given RO pair.

2.3.2 Security of the Generated Bitstring

From Table 2.2.1, one may argue that the bits generated by the proposed TAC RO PUF are reliable but are not secure. For example, suppose the adversary reveals the table information stored in the non-volatile memory, and then he will learn that the bits generated by Pair 1 and Pair 6 are complement. This does release certain locality information and the security concern can be addressed, but not completely

resolved, with help from good pairs. For instance, assuming that the first 6 bits generated by good pairs are '110101', to ensure the six contributing pairs can generate a reliable bitstring '010001', we first generate their bit-wise exclusive-or, which is '100100' in this case and then for the i-th bit b_i in '100100', we find a contributing pair that satisfies the following two conditions: (1) the pair's undecided temperature gap does not overlap with the i-th pair; (2) the pair produces a reliable bit that equals to b_i. Consequently, we can replace the last two columns in Table 2.2.1 with the indices to the corresponding peer pairs.[1] In our case, the last column after replacement can be Pair 6, Pair 4, Pair 5, Pair 2, Pair 4, and Pair 5, corresponding to the exclusive-or result '100100'. Now we show how to generate reliable bits from contributing pairs after the hardening. Recall that in Figure 2.2, at temperature k, we can only get '0?00?1'. For the undecided Pair 2 and Pair 5, the modified table indicates that we should look at the outputs of Pair 4, which is a bit '0'. Now we extract bits 2 and 5, '1' and '0' respectively from the bitstring '110101...' generated by the good pairs. A new table look-up tells us to exclusive-or both the bits with the output '0' from Pair 4, yielding bits '1' and '0' respectively to replace the two ?'s and the enrolled 6-bit secret '010001'. Note that in this modified version, the adversary only learns from the table which two contributing pairs are cooperating but does not gain explicit information on whether they generate the same bit value or its complement. Nevertheless, this security enhancement does not help when the

[1] If there is no such contributing pair, we may simply discard the singleton or pair it with a good pair; however, it must be done with caution because the increase of correlation reduces the strength of security.

adversary uses brute force to search for the key. After accounting for the 3-bit entropy loss in Table 2.2.1, the min-entropy of the 6-bit secret '010001' is 3. Moreover, there is no guarantee on the output of TAC RO PUF to be in uniform distribution; interested readers can jump to Chapter 4 for our treatment.

2.4 Validation

2.4.1 Experiment Setup

Our experiment was conducted on 9 Xilinx XC4010XL FPGA boards. For each FPGA, we implemented 32 target ROs on each to simulate the 1-out-of-8 scheme and our 1-out-of-2 temperature-aware cooperation scheme. To double the secrecy generation power, here the 1-out-of-k selects the most reliable pair out of k ROs instead of k pairs of ROs.

Our design tool is Xilinx Foundation 2.1i. In order to gain full control on the number and the placement of logic gates, the design is conducted in schematics. The ROM16×1 in the design library is the primitive logic block in our design. By setting up the initial value of ROM16×1, primitive logic gates such as NAND can be realized as shown in Figure 2.3. The physical location of each ROM16×1 can be specified in the user constraint file, $*$.ucf that guides the design tool in placement and route. As indicated in Figure 2.4, the 32 target ROs are placed in two rows and labeled sequentially. We are able to create 4 disjoint units: (1–8), (9–16), (17–24), (25–32) for the 1-out-of-8 scheme and 16 disjoint pairs for our 1-out-of-2 scheme. The default routes determined by the tool for all the ROs and counters are not

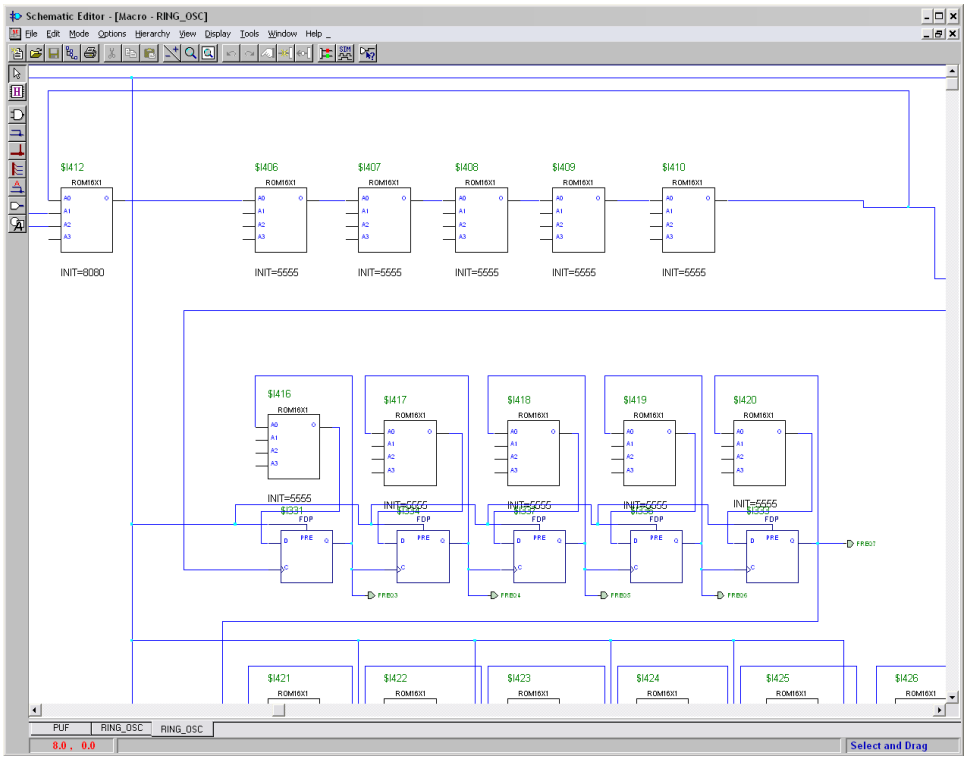

Figure 2.3: Schematics: The ROs (above row) and ripple counters (center) are realized through LUTs and flip-flops, primitive elements on Xilinx FPGAs.

symmetric; however, through manual routing the asymmetry can be fixed in Xilinx FPGA Editor as shows Figure 2.5.

The compiled design is uploaded to Xilinx XC4010XL via a parallel port. Each measurement is triggered by a start signal sent from the PC software to the board and the on-board LED indicates the relative frequencies recorded in the target counter and the reference counter. For each target RO, 10 measurements are taken at 0°C and the average is used as the frequency. The same methodology is applied to obtain the frequency of each RO at 100°C. The temperature is modulated by placing the boards in the freezer or the oven until the thermal meter attached on

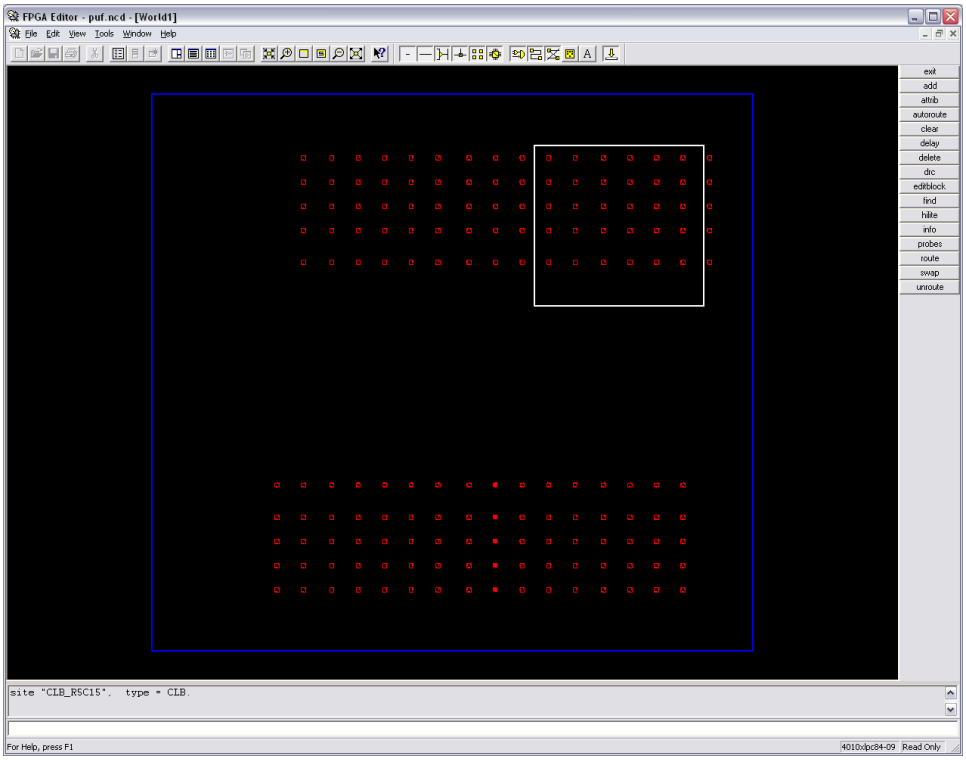

Figure 2.4: Symmetric placement: The ROs are placed symmetrically in 32 columns divided in two rows.

the FPGA indicates that the target temperature is reached.[2] For our 9 FPGAs, each with 32 ROs, we collected two sets of 288 frequencies, one at 0°C and one at 100°C. Based on these, we derived the linear delay-temperature model for each of the 288 ROs.

Considering the total number of reliable bits generated by the two schemes are different, we use the number of reliable bits per NAND gate to compare the

[2]Since the sensor is external to the ICs, we do not rule out the possibility that there are minor differences in temperature among the ROs across the chip. In later chapters when we will conduct our experiment on the latest FPGAs where we can derive temperature readings from on-chip sensors for better accuracy.

Logic Element	**Cost in number of NAND gates**
A. 5-stage RO	5
B. 8-to-1 multiplexer	21
C. 7-stage ripple counter	49

Table 2.2: Hardware cost of the logic elements in the design

hardware cost of the two schemes.[3] Table 2.2 breaks down the cost for each hardware component.[4] To conduct a fair comparison between the two schemes, the same delay constraint must be specified, which means there is no sharing of MUXes among reliable bits in both schemes, otherwise all measurements would then have to be taken sequentially. To accommodate the maximum frequency difference between the reference and target ROs without overflow, the counter size at least has to be 7 stages.

Consequently, the total cost to implement 144 pairs of a 1-out-of-2 TAC RO PUF and 36 units of a 1-out-of-8 RO PUF is $288 \times (5 + 49) = 15552$ and $36 \times (8 \times 5 + 2 \times 21 + 2 \times 49) = 6480$, respectively. Note that since the calculation does not take into account like the cost of the non-volatile memory to store helper data, the gain can be smaller.

[3]Although (look-up table) LUT is a handy high-level logic unit on FPGA, we choose NAND as the unit for we envision ASIC implementations in the future.

[4]No hardware comparator is assumed as we do it in software.

2.4.2 Experimental Results

Figure 2.6 shows the number of good pairs, bad pairs and contributing pairs we discovered from the 144 RO pairs in the 1-out-of-2 scheme. We can see a clear trend that as δ increases, that is, raising the bar to qualifying a good pair, we see the number of good pairs decreases and the number of bad pairs increases. Searching all combinations, Figure 2.7 shows the number of reliable bits that can be generated from the contributing pairs. We see the number of reliable bits generated from contributing pairs is less than $c/2$ for perfect cooperation seldom exists. In total, the number of extracted secret bits is the sum of the number of good cells plus the number of the reliable contributing bits. Figure 2.8 draws the comparison between the secrecy generation power of two schemes. As it shows, for a large δ, that is, for a high requirement on reliability, our TAC RO PUF can be 1.8× more hardware efficient than the conventional 1-out-of-8 approach.

2.5 Summary

This chapter describe a new mechanism to improve the hardware utilization of RO PUFs. By exploiting the linearity of ROs with respect to their speed versus temperature and forming cooperation among RO pairs, we can utilize those pairs previously deemed unreliable and discarded. Our empirical data and cost analysis suggest 80% hardware savings from the new design. Having that said, the maximal entropy that can be extracted from a TAC RO PUF is still far less than the theoretical bound $\log_2 n!$, where n is the total number of ROs. In the next chapter, we

will continue to improve the hardware efficiency and achieve the bound through an algorithmic solution.

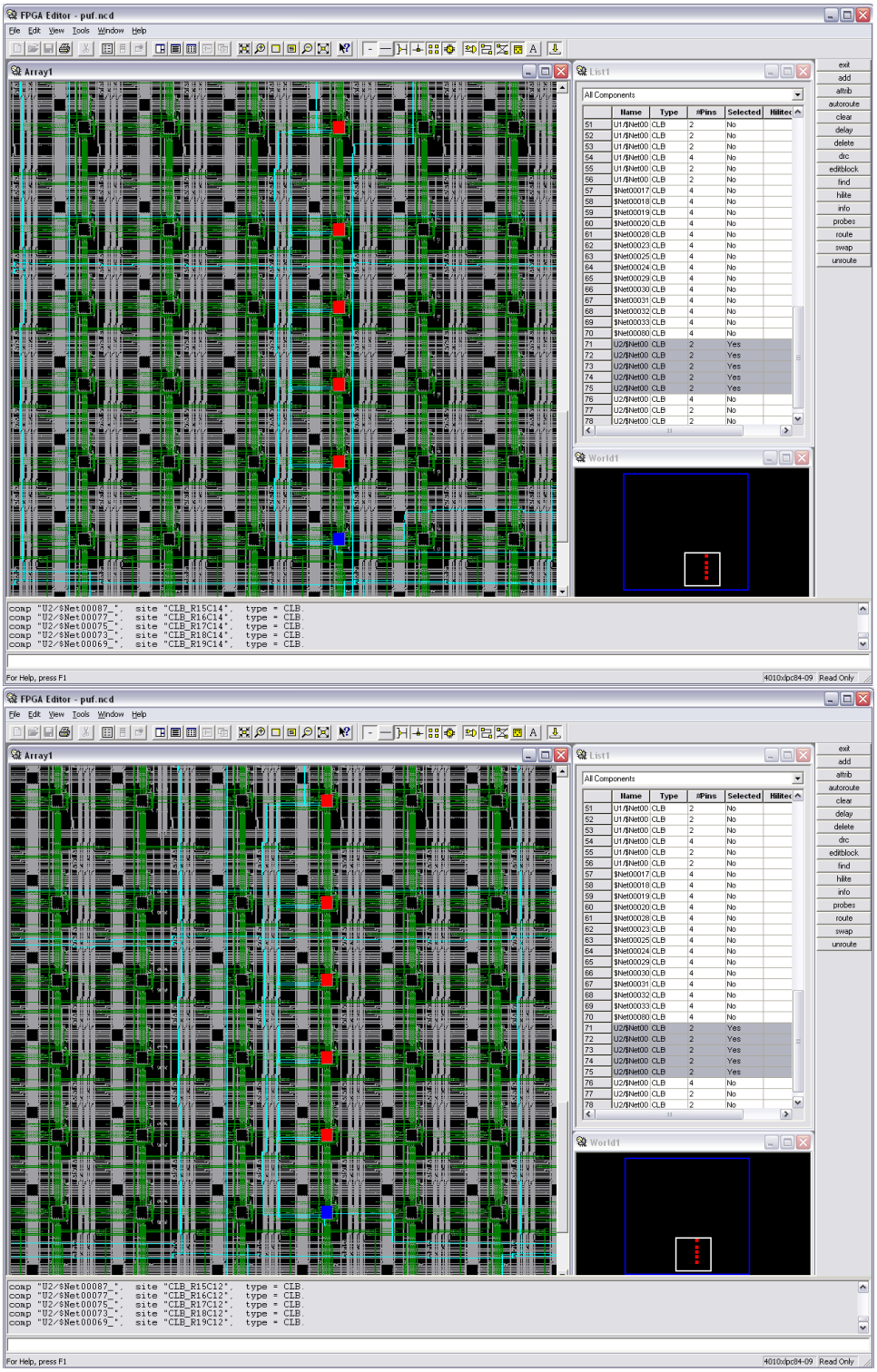

Figure 2.5: Symmetric layout: The symmetry of two target ROs placed at two CLB columns C12 and C14 is crafted manually in Xilinx FPGA Editor.

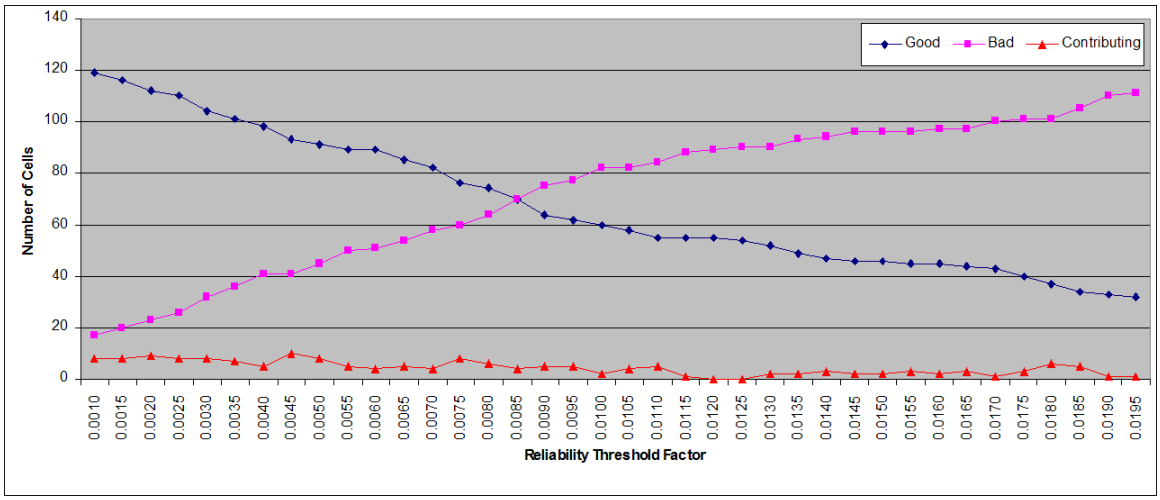

Figure 2.6: Cell classification vs. reliability threshold: Data are collected from 144 pairs generated by our 1-out-of-2 scheme, number of cells identified in each classification.

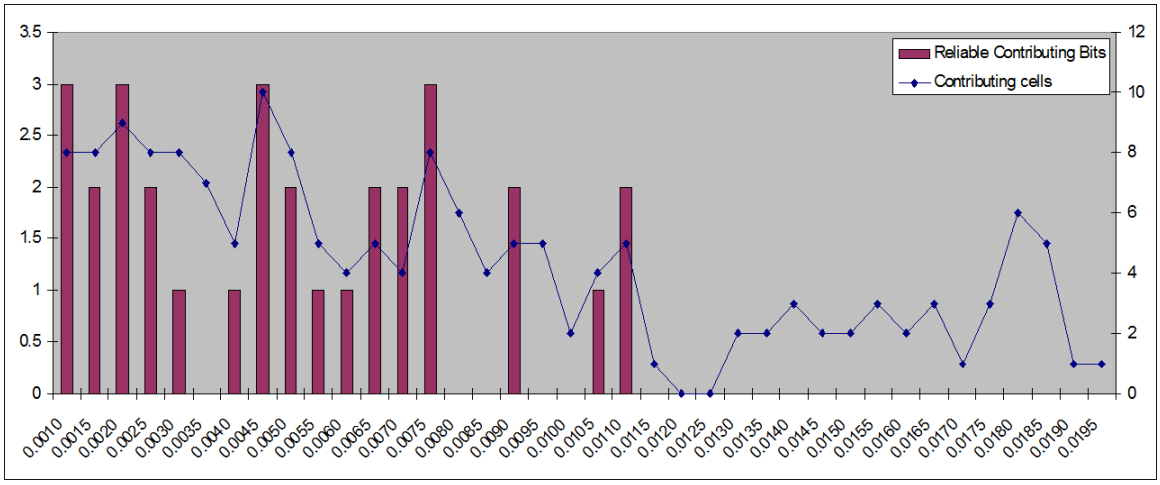

Figure 2.7: Cell cooperation vs. reliability threshold: It shows the number of reliable bits (left axis) generated by the contributing pairs (right axis) as δ (x-axis) changes.

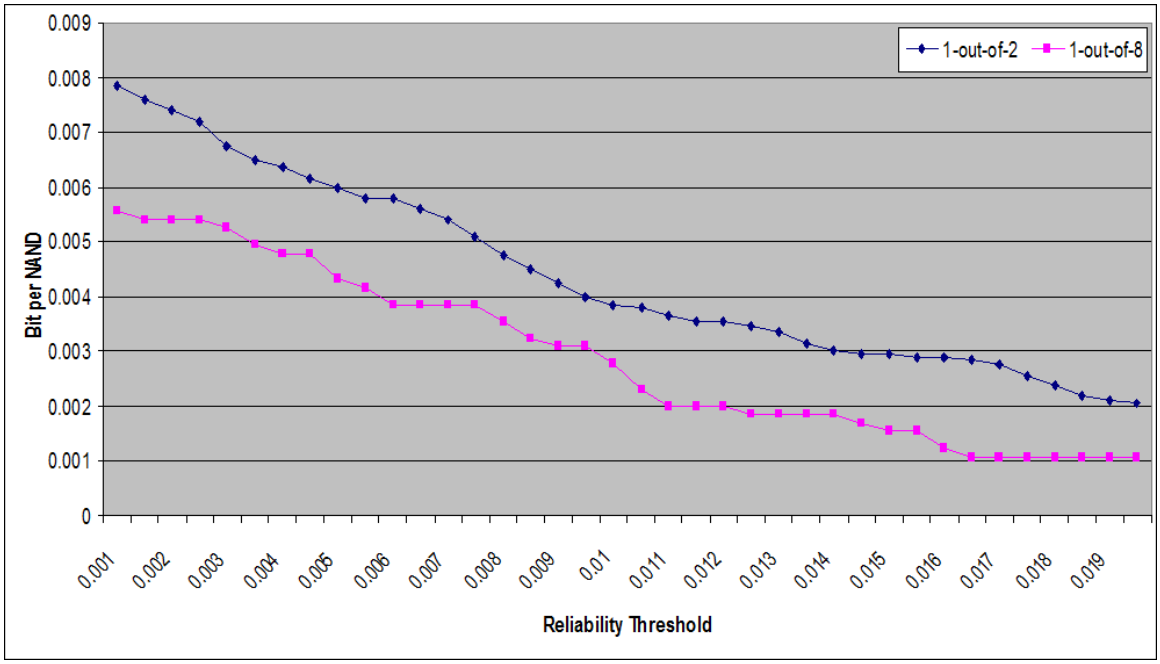

Figure 2.8: Cost performance vs. reliability threshold: performance gain from our 1-out-of-2 scheme compared with the 1-out-of-8 benchmark.

Chapter 3

LISA: Longest Increasing Subsequence Algorithm

In the previous chapter, we discussed how to improve the secrecy generation power of a RO PUF from $n/8$ to $n/2$ given n ROs. In this chapter, we will introduce LISA to further improve the hardware utilization, aiming for the theoretical bound $\log_2 n!$. We will analyze the performance of LISA based on a hybrid architecture and formulate its cost and delay metrics. Meanwhile, RO PUF designers can use a hybrid coefficient h to quickly determine the optimal hardware configuration. As with the TAC RO PUF, our claims are validated on FPGAs.

3.1 Maximal Entropy of a Comparison-based RO PUF

As we know, RO PUFs can generate one secret bit out of a pair of ROs through a simple comparison of two frequencies. Given n ROs, the best case for a TAC RO PUF is that all the $n/2$ pairs are good and thus produce $n/2$ reliable bits. However, what is the maximal number of secret bits one can generate from a RO PUF by frequency comparison? Say we have 3 ROs A, B, and C, can we extract 3-bit worth entropy from comparing frequencies of $\{A, B\}$, $\{B, C\}$, and $\{A, C\}$? Let us denote the three bits as b_{AB}, b_{BC}, b_{AC} and b_{xy} is '1' if RO x is faster than y, otherwise '0'. The answer is negative due to the existence of logic dependency. To illustrate, the three bits must satisfy one of the following 6 Boolean equations listed in Table

$$b_{AB} = b_{AC} \cdot \overline{b_{BC}} \quad b_{BC} = \overline{b_{AB}} \cdot b_{AC} \quad b_{AC} = b_{AB} \cdot b_{BC}$$

$$\overline{b_{AB}} = b_{BC} \cdot \overline{b_{AC}} \quad \overline{b_{BC}} = \overline{b_{AC}} \cdot b_{AB} \quad \overline{b_{AC}} = \overline{b_{BC}} \cdot \overline{b_{AB}}$$

Table 3.1: 6 Boolean equations expressed in terms b_{AB}, b_{BC} and b_{AC}

3.1, where operators $\cdot$ and $\overline{\ }$ are the logic AND and NOT, respectively. This means that the value of one bit can always be deduced from the other two. Take the first Boolean equation $b_{AB} = b_{AC} \cdot \overline{b_{BC}}$ for example: It means if A is faster than C ($b_{AC} = 1$) and B is slower than C ($b_{BC} = 0$), then A must be faster than B ($b_{AB} = 1$). In other words, the outcomes of b_{AB}, b_{BC}, b_{AC} = 0,0,1 and 1,1,0 are impossible to occur. As these three bits cannot be assigned values independently, the maximal entropy of this case is actually less than 3.

To have another example: one may try to generate $O(n^2)$ independent bits by dividing the n ROs into two groups, say, left and right and each of size $n/2$. By pairing up one from each group and defining the outcome as '0' if the left faster than the right, otherwise '1', one may generate $(n/2)^2$ bits. It seems that no bit can be deducted from other pairs so that one may conclude that the $(n/2)^2$ bits are independent. In fact, dependency does exist among the $n^2/4$ bits and this can be shown by listing all $n!$ permutations lexicographically in terms of the running speed. That is, since $O(n^2) > O(n \log n)$, thus $n^2/4 > \log_2 n!$ asymptotically; therefore, dependency exists. The argument also shows that the maximal amount of secret bits extractable from a comparison-based RO PUF is less or equal to $\log_2 n!$) [47]. In the following, we define two properties for the generated secret bits:

Definition 1: *k bits are independent if the probability of any bit is not affected*

by the knowledge of the rest $k-1$ bits.

Furthermore, when pairing up two ROs to generate one bit, we want the bit value to be consistent regardless of ambient conditions such as across 0°C to 100°C. As such, we say

Definition 2: *a bit is reliable if its value remains constant regardless of environmental changes, such as temperature, supply voltage, humidity, aging, etc.*

In this work, the reliability of RO PUFs is ensured by requiring the speed difference between two ROs larger than certain threshold, herein, defined as f_{th}. The higher the threshold, the more reliable the extracted bits. At the moment, our efforts in this chapter is to tolerate temperature fluctuations while assuming other environmental factors can be accommodated by a careful choice of f_{th}. Later on, all impacts will be treated equally with the introduction of error correcting codes (ECC).

3.2 RO PUF Secrecy Extraction Algorithms

In this section, we provide two algorithms SPA and LISA to maximize the secrecy extraction power of RO PUFs.

3.2.1 Sequential Pairing Algorithm (SPA)

To our best knowledge, all RO PUFs extract their secrecy by forming RO pairs. To avoid logic dependency, we may restrict every RO be paired no more than once. As such, in the best case we can extract $\lfloor n/2 \rfloor$ bits. To be more resilient to

environmental changes, the bound can be much lower, like $\lfloor n/16 \rfloor$ for the 1-out-of-8 masking scheme [47]. Even though the chain-like neighbor pairing can generate as much as $(n-1)$ bits, all ROs except the first and the last are paired twice, violating the one-pairing policy and having logic dependency among the secret bits. Furthermore, neighbor pairing has a smaller operating range between 25°C and 65°C [32]. As the pseudo code shows, SPA has two greedy characteristics in nature. Firstly, SPA starts with pairing the fastest RO in f_{min} where $(i = 0)$ with the RO right in the middle of f_{min} where $(j = n/2)$ such that SPA can generate $n/2$ bits in the best case. Secondly, SPA always tries to form a pair with an unpaired RO, indexed by i, that is fastest in f_{min}. The complexity of SPA is dominated by the sorting algorithm used in Line 1, while Line 2–12 take $O(n)$ time because j increases by 1 for each iteration of the single loop.

3.2.2 Longest Increasing Subsequence Algorithm (LISA)

In order to beat the upper bound $O(n)$ of pairing algorithms, we consider two or more ROs at a time. Suppose we have 4 ROs $\{A, B, C, D\}$, there are $4! = 24$ different ways to list their running speed in an increasing order. To list them all lexicographically, we have $\{ABCD\}$, $\{ABDC\}$, $\{ACBD\}$, $\{ACDB\}$ ... $\{DCBA\}$. Each permutation is unique and equally likely when we assume RO X faster than RO Y is 1/2 and vice versa. The 24 permutations can be encoded in five bits, say, a one-on-one mapping to '00000', '00001', '00010', '00011',...,'10111'. The 5-bit bitstring represents the extracted random unclonable secret in a binary form. For instance,

Algorithm 1 SPA

Input: (1) RO link list $phy[n]$ in physical order on the silicon. Each element of $phy[n]$ contains the information of the lowest frequency $phy[i].f_{min}$ at the highest temperature as well as the highest frequency $phy[i].f_{max}$ at the lowest temperature; (2) the link list size n. For simplicity, n is an even number; (3) the frequency threshold f_{th} for reliability.

Output: 1) The link list of discovered pairs $p[]$; (2) the size k of the link list $p[]$. k is also the number of independent reliable bits.

1: sort $phy[n]$ by f_{min} in the decreasing order and store the result in link list $sorted[n]$

2: $i \leftarrow 0$, $k \leftarrow 0$, $p \leftarrow \emptyset$

3: **for** $j \leftarrow n/2$ to $(n-1)$ **do**

4: **if** $((sorted[i].f_{min} - sorted[j].f_{min} \geq f_{th})$ && $(sorted[i].f_{max} - sorted[j].f_{max} \geq f_{th}))$ **then**

5: $p.append(i, j)$

6: $i{+}{+}$, $k{+}{+}$

7: **if** $(i \geq n/2)$ **then**

8: break

9: **end if**

10: **end if**

11: **end for**

12: **return** p, k

if the 4 ROs ordered from slow to fast is $\{ACBD\}$, the physically unclonable secret is '00010'. In this way, we can have 24 possible kinds of secret information. Note that the p.m.f of the most significant bits b_5 is not uniform but has a bias to take '0' with probability 2/3 and a probability 1/3 to take '1'. Furthermore, b_5 and b_4 are not independent because the knowledge on b_5 affects the p.m.f of b_4. In fact, $b_5 = 1$ implies $b_4 = 0$. Nevertheless, the least significant 3 bits b_3, b_2 and b_1 are independent and have an equal probability to take '0' and '1'. Roughly speaking, we say we can generate $\lfloor \log_2 24 \rfloor = 4$ secret bits from 4 ROs of identical design. To study such a group-based approach, we formulate the following optimization problem:

Given n ROs $\{1, 2, \ldots, n\}$, a threshold f_{th} to tolerate measurement noise, and the highest and lowest running frequencies f_{max} and f_{min} throughout interested environmental conditions for each RO, we want to find a partition $s = \{s_1, s_2, \ldots, s_m\}$ where for any set of ROs s_i Constraints 1)–4) are satisfied and $\sum_{i=1,\ldots,m} \log_2 |s_i|!$ is maximized, where m is the number of sets in partition s and $|s_i|$ is the number of ROs in set s_i.

1. $s_i \cap s_j = \emptyset$, where $1 = i, j = m$
2. $s_1 \cup s_2 \cup s_3 \cup \ldots \cup s_m = 1 \ldots n$
3. $\forall x, y \in s_i$, if $f_{min}(x) < f_{min}(y)$, then $f_{max}(x) < f_{max}(y)$
4. $\forall x, y \in s_i$, $|f_{min}(x) - f_{min}(y)| \geq f_{th}$ && $|f_{max}(x) - f_{max}(y)| \geq f_{th}$

Constraints 1) and 2) ensure that $s = s_1, s_2, \ldots, s_m$ is a partition; Constraint 3) protects against potential bit flips under different operating temperature; Con-

straint 4) ensures that the frequency difference between any two ROs in the same set s_i is large enough to accommodate measurement errors. The objective function $\sum_{i=1,\ldots,m} \log_2 |s_i|!$ measures the number of independent and reliable bits generated by partition s. While SPA can be viewed as a special case of this partitioning problem, where each set s_i has either a single or a pair of ROs, finding sets in larger sizes yields more bits in total. The challenge boils down to finding large s_i'S while satisfying Constraints 3)–4). To address, LISA first sorts ROs increasingly with respect to $phy[n].f_{min}$ and denotes the sorted result $sorted[n]$. Then, it iteratively searches for longest increasing subsequences (LIS) that satisfy Constraints 3) and 4). For each iteration, LISA discovers one group (set) of ROs; all ROs in that group are removed from $sorted[n]$ (Line 4–5) before the next iteration. For each LIS s_i, Constraint 3) is satisfied in that all f_{min}'s and all f_{max}'s are in an increasing order. In other words, no crossover as mentioned in Chapter 1 is possible. Constraint 4) is satisfied in function $findReliableLIS()$.

The $findReliableLIS()$ procedure is based on the optimal LIS algorithm in [33]. For each RO j, if it can be included in the current longest list, a new stack ST_h will be created to extend the length the subsequence (Lines 6–8); otherwise it forms a sequence with length $(p+1)$ whenever it is feasible (Lines 10–13). When all the n ROs are pushed to one of the stacks $ST_1 \ldots ST_h$, a LIS is discovered backward by starting from the last element of the LIS on top of ST_h and tracing back the link pointers of PRE until ST_1 (Line 18). The tests on Line 5 and 10 enforce Constraints 3) and 4). The complexity of LISA is $O(mn \log_2 n)$ where m is the number of groups (sets) in the partition s returned from the algorithm. In the worst

Algorithm 2 LISA

Input: (1) A link list $phy[n]$ of ROs in their physical order on chip where each element of $phy[n]$ contains the information of the lowest frequency $phy[i].f_{min}$ at the highest temperature and the highest frequency $phy[i].f_{max}$ at the lowest temperature; (2) The number of ROs n. For simplicity, n is an even number and $n \geq 2$; (3) The reliability threshold f_{th}.

Output: (1) The partition of ROs.

1: sort $phy[n]$ by f_{min} in the increasing order and store the result in link list $sorted[n]$

2: $i \leftarrow 0$

3: **while** there are ungrouped ROs **do**

4: $s_i \leftarrow$ **findReliableLIS(**$sorted[n]$**)**

5: mark ROs in s_i grouped, remove them from $sorted[n]$ and update n

6: i++

7: **end while**

8: **return** $s_0, \ldots, s_{i-1}$

Procedure 3 findReliableLIS()

1: create a stack ST_1 and push the first RO $sorted[1]$ to it

2: $h \leftarrow 1$

3: **for** $j \leftarrow 2$ to n **do**

4: $top \leftarrow$ the top RO on stack ST_h

5: **if** $((sorted[j].f_{min} - top.f_{min} \geq f_{th})$ && $(sorted[j].f_{max} - top.f_{max} \geq f_{th}))$ **then**

6: h++

7: new a stack ST_h, push $sorted[j]$ to it

8: $sorted[j].PRE \leftarrow top$

9: **else**

10: find the stack ST_p with the largest index p that has its top element's $((f_{max} < sorted[j].f_{max} - f_{th})$ && $(f_{min} < sorted[j].f_{min} - f_{th}))$

11: **if** $p \neq \emptyset$ **then**

12: push $sorted[j]$ to ST_{p+1}

13: $sorted[j].PRE \leftarrow$ top element of ST_p

14: **end if**

15: **end if**

16: **end for**

17: **return** sequence $(sorted[j_1], sorted[j_2], \ldots, sorted[j_h])$, where $sorted[j_h]$ is the top element of ST_h and $sorted[j_{q-1}] = sorted[j_q].PRE$, q from 2 to h

case, this is bounded by $O(n^2 \log_2 n)$. Two points are worth noting: First, with a minor change, LISA can run in $O(n \log_2 n)$ by reusing ST_1 to ST_h created at the first round; however, the optimization in speed will diminish its power in secret generation. Second, LISA also works with the case as only one frequency reading, say, at room temperature 25°C, is available for each RO. However, in such a case, the linear frequency-temperature characteristic of RO is not available and a more conservation f_{th} and/or a stronger error correcting code or fuzzy extractor are likely required to achieve the same level of reliability.

3.3 Hardware Cost and Delay Analysis

Hardware utilization and execution speed are often a trade-off. To facilitate the cost and delay analysis of RO PUFs and to evaluate the secret extraction power of LISA, we construct a hybrid architecture. Through solving a hybrid parameter h, a designer can determine the most efficient structure that meets diverse design specifications. The proposed hybrid hardware architecture is meant to be general and independent of secret extraction algorithms such as SPA and LISA. The hybrid coefficient h is defined as the number of target ROs sharing the same target counter and the value of h is between 1, completely parallel and n, fully sequential. Take n as 8 and h as 4 for example, 2 target ROs can be measured simultaneously and the values of the target counters can then be transmitted to CPU via a pipeline as shown in Figure 3.1. In this case, it takes 4 iterations to conduct a full frequency characterization.

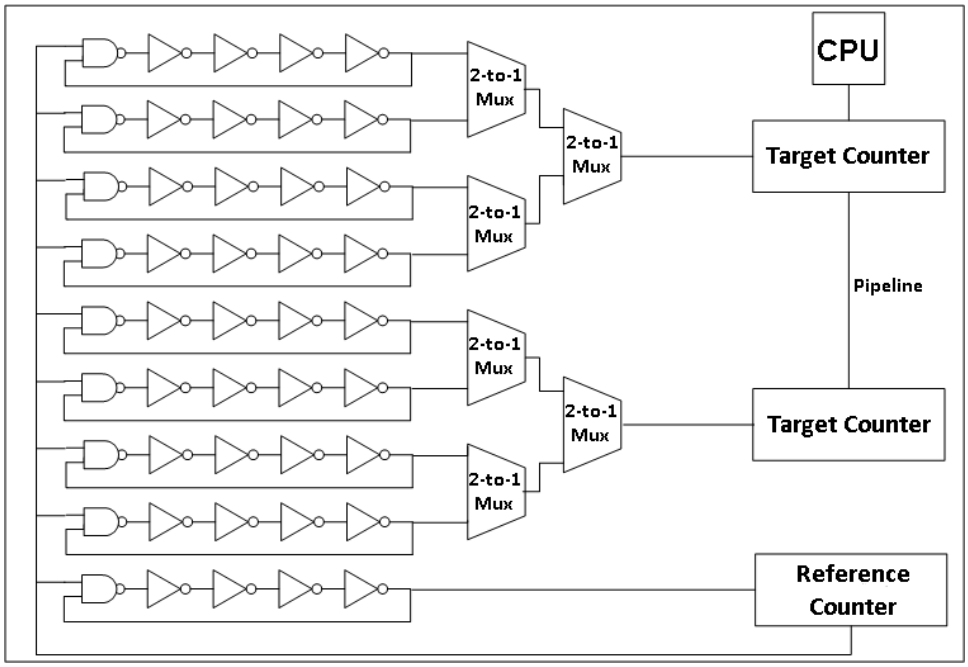

Figure 3.1: The hybrid architecture with hybrid coefficient $h = 4$.

$$c = (5(n+1) + 3(n/h) \cdot (h-1) + 7q(n/h+1))/k \tag{3.1}$$

$$d = 2b - 1h/f \tag{3.2}$$

For k physical unclonable bits extracted, Equations (3.1) and (3.2) normalize the cost and delay metrics c and d as NAND gates and micro-seconds per bit. For the cost metric, a RO composed of 5 stages costs 5 NANDs gates, assuming the overhead of control logic is negligible; a x-to-1 multiplexer costs $3(x-1)$ NANDs as it can be implemented as $(x-1)$ 2-to-1 multiplexers, 3 NANDs each; a b-stage reference counter and a q-stage target counter cost $7b$ and $7q$ NANDs as one stage of ripple counter costs one NOT gate (1 NAND) plus one flip-flop (6 NANDs). For the delay metric c, we only account for the time to conduct frequency measurement;

other factors such as memory access time and CPU time are excluded for simplicity. Each RO is targeted to run at f MHz.

3.4 Validation

3.4.1 Experiment Setup

As in the previous chapter, the experiments were conducted on the same 9 Xilinx XC4010XL FPGAs, 32 target ROs each. 576 frequency readings were collected in total from the 288 ROs as the input of SPA and LISA respectively for secrecy extraction. The logic design is accomplished in the Schematic Editor of Xilinx Foundation 2.1i and the user constraint file (.ucf) allows us to specify CLBs to implement the logic. Logic gates such as NAND and NOT are realized by configuring the initial value of ROM16×1, a Xilinx logic primitive in the design library. Figure 3.2 demonstrates the symmetry of the place-and-route of two ROs located at CLB R15–19C12 and R15–19C14. For each target RO, we take 10 measurements once at 0°C and once at 100°C. The average of the 10 measurements is one frequency reading of the input to SPA and LISA. Each algorithm outputs the number of extracted bits k as well as related information to facilitate secrecy regeneration. The temperature was modulated by placing the board in freezer/oven until the thermal meter attached to the FPGA indicates the target temperature.

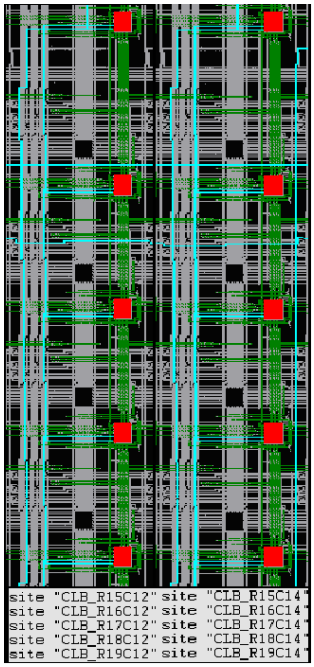

Figure 3.2: Identical logic layout by manual place-and-route of two ROs

3.4.2 Experimental Results

From the total 288 linear frequency-temperature relations, Figure 3.3 shows the number of reliable unclonable bits k identified by SPA, LISA and the benchmark 1-out-of-8 scheme. The x-axis is the reliability threshold R_{th} defined as the ratio of f_{th} over the capacity of the reference counter. We see that relaxing R_{th} does not help much on the secrecy extraction power of the 1-out-of-8 pairing scheme, suggesting that the 1-out-of-8 scheme is very robust in reliability. On the other hand, SPA and LISA are much more sensitive on R_{th}. Take R_{th} as 0.01 for example, the gain in k from SPA and LISA is 5.66 and 6.94 times respectively when compared with the 1-out-of-8 benchmark.

Next we take the hardware cost and delay into account. Table 3.2 lists the parameters we plugged into the calculation. As mentioned earlier, both the hardware

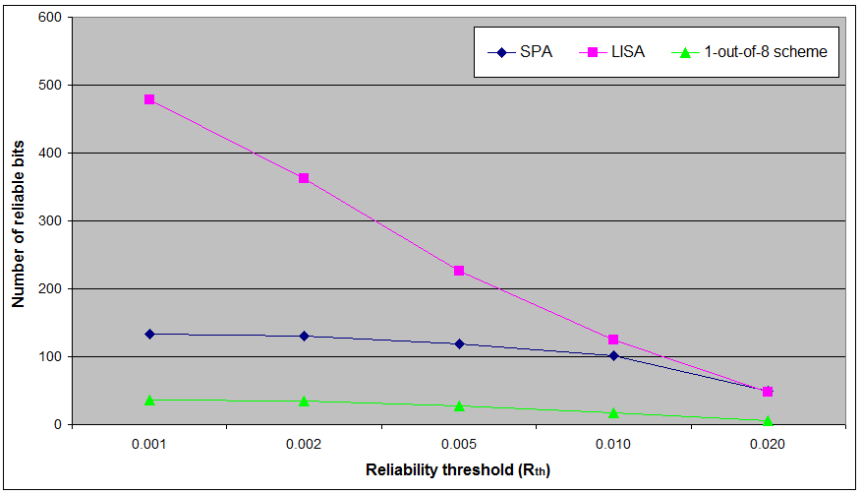

Figure 3.3: The total number of reliable unclonable bits k discovered out of 288 target ROs on 9 Xilinx XC4010XL FPGAs

cost and delay depend on the hybrid coefficient h. Here the value h is picked such that the hardware cost is minimized given a predefined delay constraint. Figures 3.4 and 3.5 show the dynamic of the cost and the delay as a function of the hybrid coefficient h. For a fair comparison, the difference in the cost structure between the hybrid architecture and the 1-out-of-8 hardware architecture is also taken into account, see Table 3.3. In terms of the number of reliable unclonable bits generated per NAND gate, Figure 3.6 reports the hardware efficiency of SPA and LISA when compared with the benchmark. When R_{th} is as small as 0.001, it is to the advantage of LISA for it is easier to find LIS's in great length. When R_{th} moves up from 0.001 to 0.01, the hardware efficiency gains of SPA and LISA decrease. In this interval tightening up the threshold does not bring down as much the reliable bits generated from the benchmark as in the case of SPA and LISA. Interestingly, when raising the bar further with R_{th} between 0.01 to 0.02, it hinders the secret extraction power

Parameter	Value	Parameter	Value
n: num of target RO	288	h: hybrid coefficient	4
b: stages of ref. counter	11	q: stages of target cnt.	7
f: ref. RO freq. (MHz)	150	c: delay constraint (us)	200

Table 3.2: Values applied to the cost and delay metrics

Architecture	Hardware Cost (NAND/bit)
1-out-of-8	$36 \cdot (8 \cdot 5 + 2 \cdot 7 \cdot 3 + 2 \cdot 7 \cdot 7)/k$
Hybrid ($h = 4$)	$(5 \cdot 289 + 3(\frac{288}{4})(4-1) + 7 \cdot 7 \cdot (\frac{288}{4} + 1))/k$

Table 3.3: Cost models of two architectures

of the benchmark more than it does to the proposed schemes. However, different reacting rates cause a crossover between SPA and LISA in their uptick tails. Our estimation suggests that when R_{th} is 0.01, to generate a 128-bit RO PUF secret using LISA only requires 5818 NAND gates, about 8x saving on hardware.

3.5 Summary

In this chapter, we proposed two algorithms SPA and LISA to solve the optimization problem we formulated for secrecy extraction. Theoretically, LISA can achieve the upper bound $\log_2 n!$ by forming o giant group containing all ROs. Unlike TAC RO PUF, solutions in this chapter do not require an on-board temperature sensor. We also analyzed the hardware cost and delay of a generic RO PUF architecture such that the optimal configuration can be quickly determined afterwards.

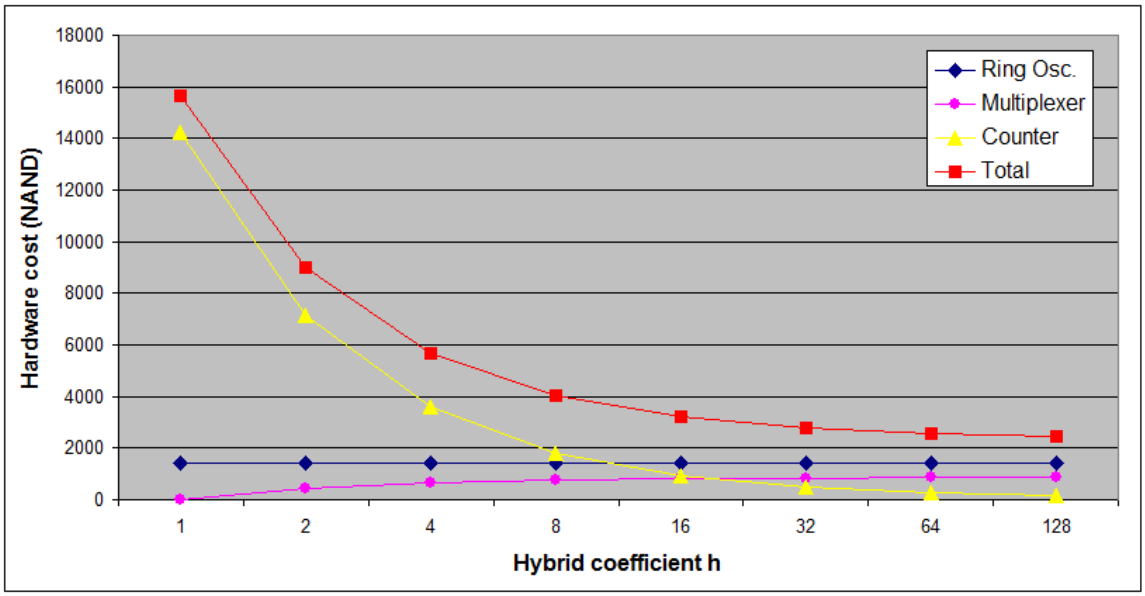

Figure 3.4: $h - c$ characteristic

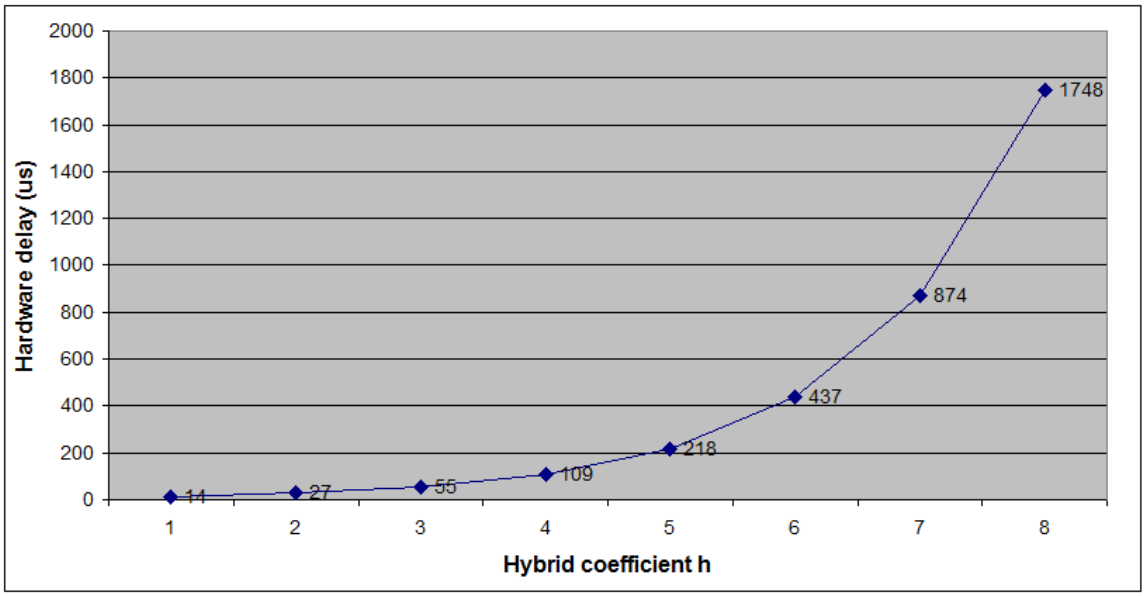

Figure 3.5: $h - d$ characteristic

Based on the cost analysis upon our experimental results, LISA is about 8x more hardware efficient than the 1-out-of-8 benchmark. However, recall Chapter 1 where we mentioned that the systematic variability may undermine the security of the

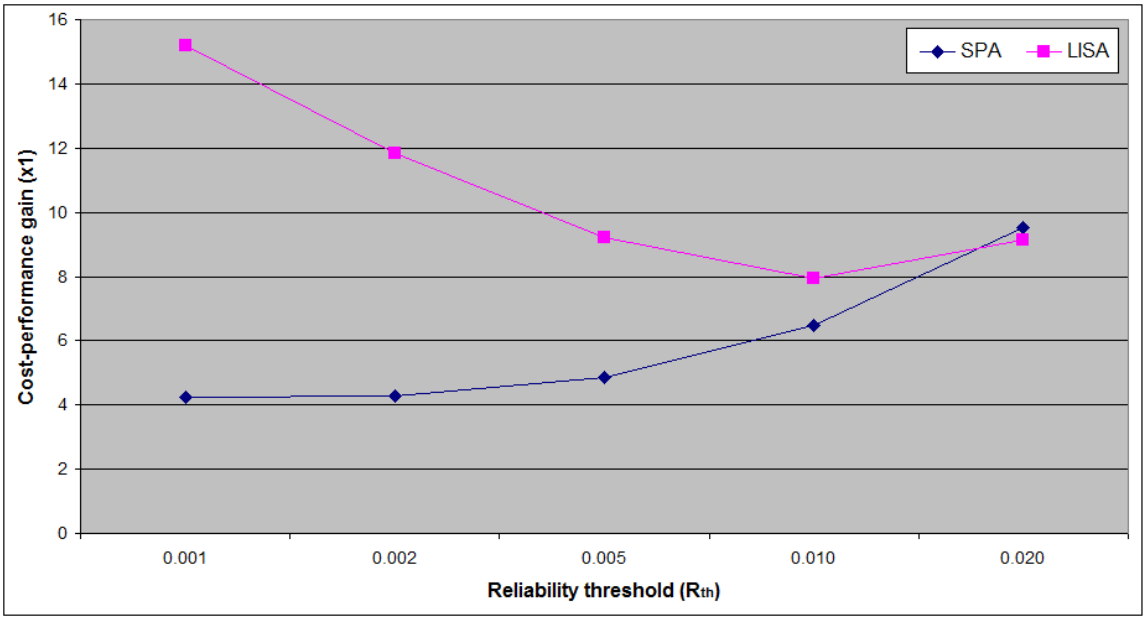

Figure 3.6: The cost-performance gains of SPA and LISA vs. the benchmark

secret we generate. In the next chapter, we will shift our gears from hardware efficiency to security and use a regression-based entropy distiller to tackle the potential threat.

Chapter 4

Regression-Based Entropy Distiller

In previous chapters, we mentioned how to extract a reliable secret efficiently from fabrication variations on each chip. One may postulate that the frequencies of ROs can be modeled by independent and identically distributed (i.i.d.) normal random variables; however, it is not the case because of the existence of spatial correlation. In fact, spatial correlation can weaken the secrecy of RO PUFs. We found that when applying the NIST statistical tests [5] to the random sequences generated from a population of 125 RO PUFs [31] with both the classic 1-out-of-8 coding [47, 56] and the chain-like neighbor coding [32], none of them can pass all the test. In this chapter, we propose a regression-based distiller to decouple the unwanted systematic variation from the desired random variation. Indeed, through modelling and subtracting the systematic effect on each chip, the distillation process can eliminate the unwanted spatial correlation, leaving the residue an ideal i.i.d. uniform. Indeed, by applying the 2^{nd}- or 3^{rd}-order distiller to the same benchmark data [31], the decoupled neighbor coding can pass all the NIST randomness test, so can the 1-out-of-8 coding with the help from the 4^{th}-order distiller. There are two security premises for our scheme: first, it is difficult for adversaries to derive a model more accurate than the one calculated on the chip; second, the disclosure of the model does not compromise the security of the extracted secret.

1-out-of-8		**Chain-like**		**Decoupled**		
P-VALUE	PROP.	P-VALUE	PROP.	P-VALUE	PROP.	STATISTICAL TEST
0.013689	122/125	0.000072 *	125/125	0.000003 *	115/125 *	Frequency
0.166594	125/125	0.000000 *	125/125	0.050764	120/125	BlockFrequency
0.231636	121/125	0.000000 *	125/125	0.000000 *	119/125 *	CumulativeSums (m-2)
0.059743	122/125	0.000000 *	125/125	0.000000 *	118/125 *	CumulativeSums (m-3)
0.002320	117/125 *	0.000000 *	0/125 *	0.302788	120/125	Runs
0.000603	123/125	0.000000 *	62/125 *	0.000062 *	124/125	LongestRun
0.000001 *	117/125 *	0.000000 *	0/125 *	0.000001 *	119/125 *	ApproximateEntropy
0.004904	124/125	0.000000 *	1/125 *	0.070160	116/125 *	Serial (forward)
0.552185	125/125	0.000000 *	117/125 *	0.192277	123/125	Serial (backward)

Table 4.1: NIST test results with respect to the random sequences generated by **1-out-of-8 Coding**, **Chain-like Neighbor Coding** and **Chain-like Neighbor Coding**. For **1-out-of-8 Coding**: $M = 32$ for Block Frequency Test, $m = 1$ for Approximate Entropy Test and $m = 4$ for Serial Test. For **Chain-like Neighbor Coding** and **Decoupled Neighbor Coding**: $M = 32$ for Block Frequency Test, $m = 2$ for Approximate Entropy Test and $m = 5$ for Serial Test. '*' marks a failure.

4.1 Security Analysis

The security aspects of a random number generator (RNG) can be argued by the statistical characteristics of the random sequences it produces and the NIST test suite [5] is regarded as the industry standard in testing cryptographic RNGs. As one may expect that random sequences generated by well-known secret selection strategies such as 1-out-of-8 Coding and Neighbor Coding shall pass the NIST randomness tests, to our surprise, based on the frequency characterization collected from 125 FPGAs [31], no random sequence actually pass all tests that are applicable to their length, see Table 4.1. While the underlying causes of the failures can be composite and infinite, there are at least two main causes that are detrimental to PUF security.

4.1.1 Failure Cause 1: Chain Dependency

The high failure rate of Chain-like Neighbor Coding can be attributed to the non-independent comparison chain. Take 3 ROs RO_A, RO_B and RO_C for example, two random bits are generated by comparing RO_A with RO_B and RO_B with RO_C. As we know, to pass NIST test for randomness, the random sequence is expected to demonstrate no significant deviation from the probability mass function (p.m.f) of tossing a fair coin twice, i.e., the 4 possible outcomes '00', '01', '10' and '11' are expected to equally likely with probability 1/4. In fact, it is not case for the two bits we generate from the 3 ROs, assuming that the 6 outcomes $RO_A < RO_B < RO_C$, $RO_A < RO_C < RO_B$, $RO_B < RO_A < RO_C$,..., $RO_C < RO_B < RO_A$ are equally

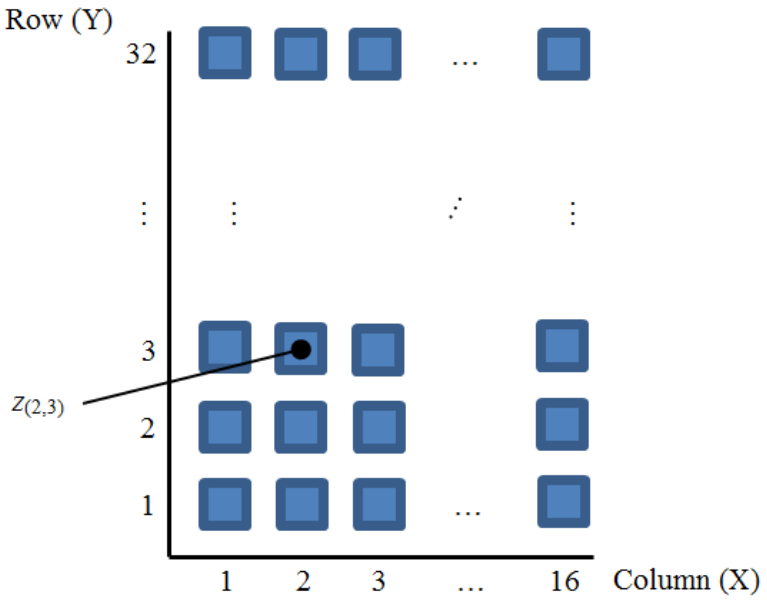

Figure 4.1: The placement of 512 ROs as a 16 (columns) by 32 (rows) array; for site $RO_{x,y}$, its running frequency is labeled $z_{x,y}$

likely with probability exactly 1/6. In turn, the probability of the outcome '00', '01', '10' and '11' of the 2-bit random sequence is 1/6, 1/3, 1/3 and 1/6 respectively, a clear deviation from the p.m.f of the 'ideal' random sequence; consequently, the min-entropy is actually 1.58 rather than 2, our first thought. One solution is to break up the chain such that all ROs only pair up with its neighbor once, i.e., (RO_A, RO_B), (RO_C, RO_D)... . As Table 4.1 shows, the decoupling improves the pass rate but still is not good enough to pass all.

4.1.2 Failure Cause 2: Spatial Correlation

Generally RO PUFs places its ROs as a 2-D array. The dataset we use lays out 512 ROs as a 16 (columns) by 32 (rows) grid on each of the 125 FPGA devices [31] as illustrated in Figure 4.1, where $z_{x,y}$ denotes the running frequency of RO at

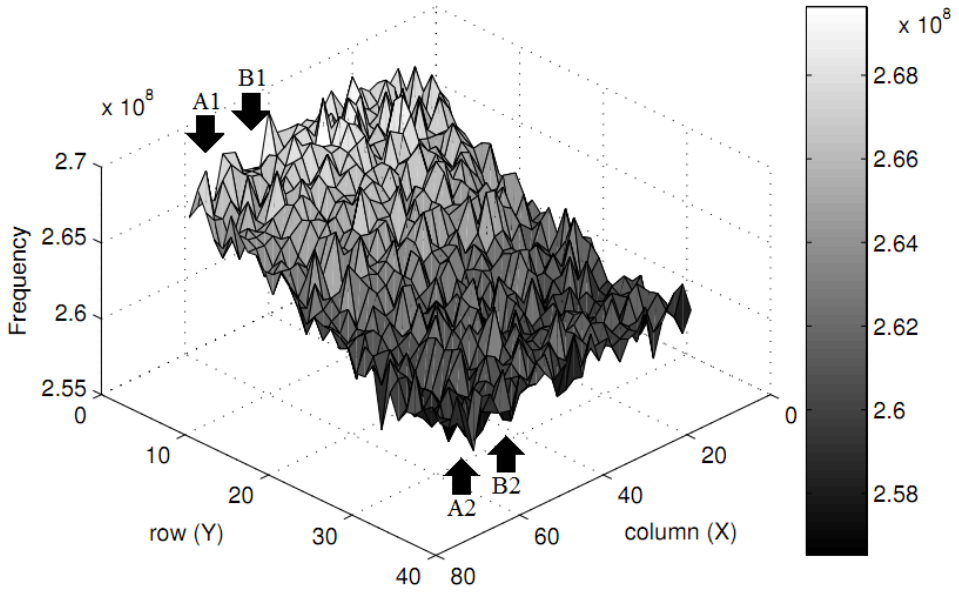

Figure 4.2: The across-die frequency topology of a RO array. The roughness of the surface represents the random variation while the slope represents the systematic [44]

site $RO_{x,y}$. One may conjecture that we can generate 1-bit secrecy out of any RO pair; however, it is not secure if we consider about the underlying spatial correlation. Figure 4.2 shows how the fabrication variation of the semiconductor process portrays: The roughness of the surface (random variation) is superimposed upon a spatial trend (systematic variation). The systematic component can significantly reduce the min-entropy of the extracted secrecy; for instance, if one generates bit A as '0' if $A1 < A2$, else '1' and similarly, bit B '0' if $B1 < B2$, else '1', spatial correlation would render $p(A = B) >> p(A \neq B)$.

The security threat of spatial correlation was first acknowledged by Chain-like Neighbor Coding [32]. In fact, it is exactly the reason behind their two design

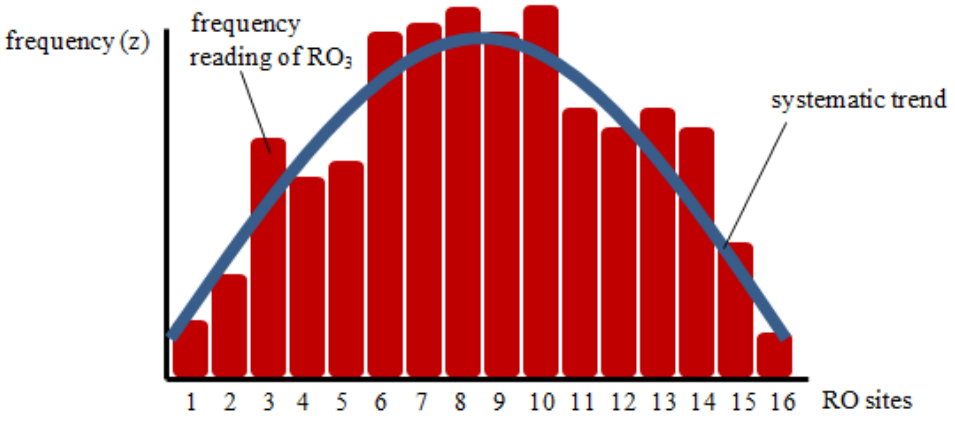

Figure 4.3: Illustration of the impact from systematic variation even after pairs are decoupled.

principles: 1) place ROs as close as possible and 2) pair ROs located adjacent to each other. The main idea is to let the systematic effect cancel out with each other, extracting secrecy out of the random effect. We see the principles have been accepted by [**?**, 50]. Nevertheless, to explain those failure cases we postulate that the small remnant of the systematic effect can still be captured by the test. To illustrate, Figure 4.3 shows a hypothetical frequency characterization of 16 consecutive ROs, assuming a convex systematic trend superimposed upon i.i.d. Gaussian random components. If, say, frequency relation $z_i < z_{i+1}$ gives us a '0', else '1', it is likely for the up slope to yield more '0's than '1's and vice versa for the down slope.

The same argument can also explain the failures we found in 1-out-of-8 coding strategy, or more generally, 1-out-of-k coding strategy; when $k = 2$, it reduces to the proposed Decoupled Neighbor Coding mentioned above. Recently, Yu et al. [56] used a similar strategy in their Index-Based Syndrome (IBS) Coding but encoded a secret bit '0' or '1' as a helper index '000', '001',...,'111' pointing to the pair

yielding the largest positive (say case '0') or negative (say case '1') magnitude. Like its predecessor 1-out-of-8 Coding, the problem of spatial correlation was not addressed explicitly. Note that the systematic trend can stay undiscovered when one tallies the total number of '0's and '1's or even when one calculates the inter-die uniqueness [47] via Hamming distance, e.g., 46.15-48.51% for RO PUFs [47, 20] and 49.97% for SRAM PUFs [24].

4.1.3 A Cautious Note on Spatial Correlation

As Table 4.1, Figure 4.2 as well as Figure 7 in [18] indicate, it is questionable to assume the raw RO PUF frequency characterization i.i.d.; nevertheless, the i.i.d. assumption is not uncommon in PUF literature and even has support from statistical results [56]. A possible explanation is that we conducted the test directly on raw output as opposed to their 'controlled output' obfuscated by a linear feedback shift register (LFSR) and/or an output hash function that de-correlate the challenge and the response [7, 18, 56]. Since the seed for obfuscation is typically provisioned externally, they have to trust the seed has never been compromised. In contrast, our scheme requires no external seed and thus more secure in this sense. Secondly, the issue of spatial correlation may be more serious than we thought because across-die spatial variation mostly results from deterministic across-wafer variation [29], that is, once process parameters surrounding chips under attack get exposed, the opponent can gain unexpected advantage to model the systematic trend and attack the min-entropy we over-estimate [53, 18, 17, 39]. Next, we describe how to eliminate

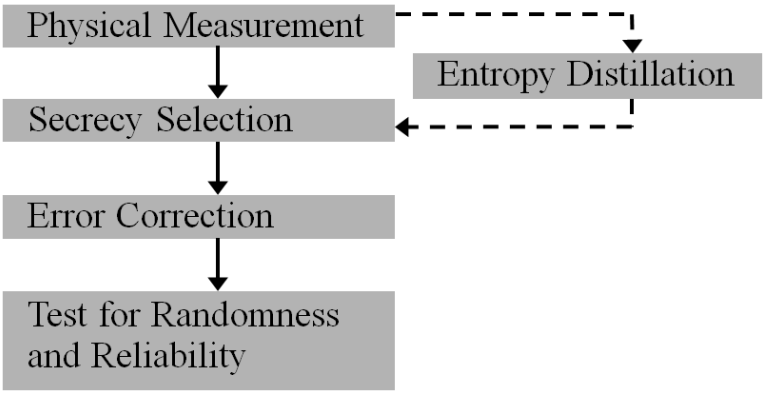

Figure 4.4: The typical workflow of a Weak RO PUF augmented with a new entropy distillation step marked in dashes

the systematic variation upfront.

4.2 Systematic Variation Elimination

Our main objective is to model the unwanted systematic variation so that we can decouple it from the desired random variation. To achieve this, Figure 4.4 adds a new distillation process to the typical workflow previously depicted in Chapter 1.

4.2.1 Modeling Fabrication Variations

To identify, model and decompose fabrication variations, Stine et al. [6] first proposed a comprehensive framework. Their methods can model variations at wafer level as well as at die level by analyzing the measurements collected from each chip on the same wafer. We find the framework not that suitable for our use because it requires revealing on-die characterization as well as trusting external information for model building, either of which would seriously compromise the security of PUFs.

Also surveyed was Liu's framework for spatial correlation modeling [30], where the author used Generalized Least Square (GLS) fitting and structured correlation function to improve the accuracy of prediction for unobservable sites. However, since we can always take measurement at any RO site of interest as wish, prediction for unobservable sites is not required. Also partly related are works devoted to extracting the correlation matrix of the entire fabrication process to facilitate, for example, statistical static timing analysis (SSTA) [52, 29]. It is worth noting that the authors in [29] concluded that once across-wafer systematic variation is captured and removed by a quadratic polynomial, within-die variation no longer contains useful spatial correlation.

As mentioned, relying on external information for model building would result in a security loophole; hence, we require that each PUF chip build its own across-die systematic model. In line with this, previous works include [43, 44, 48]. Ohkawa et al. first employed a 4^{th}-order polynomial model and demonstrated its effectiveness by contrasting the correlation coefficients of the systematic component and its random counterpart [43]. Later on, Sedcole et al. employed a quadratic model to gauge the expected magnitude of the random variation versus the maximum magnitude of the systematic variation [44]. More recently Sato et al. argued that high order polynomials could result in an over-fit and that corrected Akaike information criterion (AICc) was accurate in selecting the optimal model; quite remarkably, they found out that 1^{st} order polynomial turned out to be the best fit in most cases with respect to the variability of NMOS threshold voltages [48].

Our unique application to RO PUFs, however, distinguishes us from the above.

In our case, the optimal model is determined by the results of the NIST statistical tests for randomness rather than AICc or D'Agostino-Pearson (D-P) and Kolmogorov-Smirnov (K-S) statistical tests for normality against the variation profile as did in [48]. Indeed, our model selection can find us the best-fit polynomial-based regression distillers that fix all the failures in Table 4.1.

4.2.2 The Causes of Process Variation

The semiconductor process variation can be modeled as the sum of a systematic component and a random component [21, 6, 14, 43, 44, 12, 30, 52, 48, 29]. The systematic component attempts to capture a deterministic trend and other identifiable patterns through one or a collection of estimators. The main causes of the systematic variation can be attributed to equipment and process nonuniformity such as the focus shift of photolithography, the gradient of thermal annealing, dissimilar interactions between circuit layout and the chemical mechanical polishing (CMP) process [6, 12]. On the other hand, the random component accounts for the difference between those estimates and the observed data; the constituents include unidentified patterns, measurement errors and most importantly, atomic-level stochastic phenomena such as random dopant profiles [21, 12].

4.2.3 Polynomial Regression

Polynomial regression is a form of linear regression in which the relationship between independent variables and a dependent variable is modeled by a polynomial

of order k, where k is a non-negative integer. For a RO PUF with its m ROs arranged as r rows by c columns, the Cartesian coordinate (x, y) of ROs is regarded as two independent variables and the oscillating frequency z is the single dependent variable. In such a 2D setting, our polynomial regression model of order k, i.e., $x^p y^q, p + q \leq k, 0 \leq p, q \leq k$, takes the general form of

$$z_{x,y} = \sum_{i=0}^{k} \sum_{j=0}^{i} \beta_{k,i,j} x^{i-j} y^j + \epsilon_{k,x,y} \tag{4.1}$$

where $1 \leq x \leq c, 1 \leq y \leq r; x, y, c, r, i, j, k \in \mathbb{Z}^+ \bigcup 0; z, \beta, \epsilon \in \mathbb{R}$. On the right hand side of the equation, the summation term models the systematic variation and the residual term $\epsilon_{k,x,y}$ models the random variation. Equivalently, they can be written in matrix form

$$\boldsymbol{Z} = \boldsymbol{\Omega_k} \boldsymbol{\beta_k} + \boldsymbol{\epsilon_k} \tag{4.2}$$

where

$$\boldsymbol{\Omega_k} = \begin{pmatrix} \omega_{k,1,1} & \omega_{k,1,2} & \cdots & \omega_{k,1,n} \\ \omega_{k,2,1} & \omega_{k,2,2} & \cdots & \omega_{k,2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \omega_{k,m,1} & \omega_{k,m,2} & \cdots & \omega_{k,m,n} \end{pmatrix},$$

$$\boldsymbol{Z} = \begin{pmatrix} z_{1,1} \\ \vdots \\ z_{c,1} \\ z_{1,2} \\ \vdots \\ z_{c,2} \\ \vdots \\ z_{1,r} \\ \vdots \\ z_{c,r} \end{pmatrix}, \boldsymbol{\beta_k} = \begin{pmatrix} \beta_{k,0,0} \\ \beta_{k,1,0} \\ \beta_{k,1,1} \\ \beta_{k,2,0} \\ \beta_{k,2,1} \\ \beta_{k,2,2} \\ \vdots \\ \beta_{k,k,0} \\ \vdots \\ \beta_{k,k,k} \end{pmatrix}, \boldsymbol{\epsilon_k} = \begin{pmatrix} \epsilon_{k,1,1} \\ \vdots \\ \epsilon_{k,c,1} \\ \epsilon_{k,1,2} \\ \vdots \\ \epsilon_{k,c,2} \\ \vdots \\ \epsilon_{k,1,r} \\ \vdots \\ \epsilon_{k,c,r} \end{pmatrix},$$

$m = r \times c$, $n = \frac{(k+1)(k+2)}{2}$ and $\omega_{k,p,q} = x^{i-j}y^j$ in which $x = ((p-1) \mod r) + 1$, $y = \lfloor \frac{p-1}{r} \rfloor + 1$, $i = \lfloor \frac{-1+\sqrt{1+8(q-1)}}{2} \rfloor$, $j = (q-1) - \frac{i^2+i}{2}$, $1 \leq p \leq m$ and $1 \leq q \leq n$. For each model of order k, it is an overdetermined system, i.e., $m > n$, and can be solved by the ordinary least squares (OLS) method, which produces the 'best' estimates $\hat{\boldsymbol{\beta}}$ in the sense of minimum sum of squared errors as (4.3) indicates. By taking partial directives of (4.4) with respect to each $\beta_{k,i,j}$ and letting each gradient to zero, the solution of OLS can be expressed as (4.5) in matrix form. Interested readers are referred to [51, 35] for a detailed derivation and background information.

$$\hat{\boldsymbol{\beta}}_{\boldsymbol{k}} = \arg\min_{\boldsymbol{\beta_k}} \left\{ \sum_{x=1,y=1}^{c,r} \epsilon_{k,x,y}^2 \right\} \tag{4.3}$$

$$= \arg\min_{\boldsymbol{\beta_k}} \left\{ \sum_{x=1,y=1}^{c,r} (z_{x,y} - \sum_{i=0}^{k} \sum_{j=0}^{i} \beta_{k,i,j} x^{i-j} y^j)^2 \right\} \tag{4.4}$$

$$= (\boldsymbol{\Omega_k^T \Omega_k})^{-1} \boldsymbol{\Omega_k^T Z} \tag{4.5}$$

4.2.4 Model Selection

The higher the order we use, the less the error in the least squares sense results. Therefore, a model in high order is expected to yield less residual terms, that is, increase the difficulties in error control and eventually reduce the efficiency of RO PUFs. A model in low order, on the other hand, may not be able to capture all deterministic variation in the systematic term and in turn weaken the security. Due to these concerns, the goal of our model selection is to find out the minimal (optimal) order of the polynomial-based entropy distiller through which the output bitstrings demonstrate strong randomness. Two tasks remain: i) construct random sequences highlighting underlying spatial correlation, and ii) apply effective tests for randomness on the sequences. For simplicity, we pick an order and find a solution against that model for each PUF device before considering another order. Due to the characteristic of Weak PUFs, we only consider solutions applying to all devices but in the case of Strong PUFs, however, it is possible to come up with different solutions for different devices.

4.2.4.1 Random Sequence Generation

There are numerous ways to generate a random sequence from the frequency profile of a RO array. As shown earlier, secret selection schemes are design specific. Instead of exhausting all possibilities, we try to devise general-purpose random sequences and use them to gauge the existence of spatial correlation in the distilled random component. For this reason, we purposely pair up ROs at locations far apart,

simply opposite to the design principle of Neighbor Coding whose goal is to avoid spatial correlation. Also wanted is the length of sequence to be greater than certain minimum values so that certain tests can bear statistical significance. As a result, two indicator random sequences $\boldsymbol{S}$ and $\boldsymbol{T}$ are formulated according to rules (4.6) and (4.7) respectively. Simply put, we cut the array into two subsets in the middle along the axis and form pairs with two elements, one subset each, in distance half the corresponding side length of the given rectangle RO array. Similarly, more random sequences can be constructed for better coverage, e.g., by partitioning along two diagonals. Note that $\boldsymbol{S}$ and $\boldsymbol{T}$ are meant to be general-purpose, when considering a specific PUF, its own secret selection scheme should be used instead if applicable.[1]

$$\begin{aligned}\boldsymbol{S} &= X_1, \ldots, X_{l_X}, \ldots, X_{L_X} \text{ where } X_{l_X} \\ &= \begin{cases} 0 & \text{if } z_{u_X, v_X} \leq z_{u_X + \lfloor \frac{c}{2} \rfloor, v_X} \\ 1 & \text{otherwise} \end{cases} \end{aligned} \tag{4.6}$$

$$\begin{aligned}\boldsymbol{T} &= Y_1, \ldots, Y_{l_Y}, \ldots, Y_{L_Y} \text{ where } Y_{l_Y} \\ &= \begin{cases} 0 & \text{if } z_{u_Y, v_Y} \leq z_{u_Y, v_Y + \lfloor \frac{r}{2} \rfloor} \\ 1 & \text{otherwise} \end{cases} \end{aligned} \tag{4.7}$$

where in (4.6), $u_X = ((l_X - 1) \bmod \lfloor \frac{c}{2} \rfloor) + 1, v_X = \lfloor (l_X - 1) / \lfloor \frac{c}{2} \rfloor \rfloor + 1, 1 \leq l_X \leq L_X = r \times \lfloor \frac{c}{2} \rfloor$; similarly in (4.7), $u_Y = \lfloor (l_Y - 1) / \lfloor \frac{r}{2} \rfloor \rfloor + 1, v_Y = ((l_Y - 1) \bmod \lfloor \frac{r}{2} \rfloor) + 1, 1 \leq l_Y \leq L_Y = c \times \lfloor \frac{r}{2} \rfloor$.

[1] Group-Based Coding [55] is not applicable since its secret selection will inject algorithmic randomness.

4.2.4.2 Tests for Randomness

An 'ideal' random sequence is regarded as the result of consecutive flips of a fair coin: when a head turns out, denote the outcome as, say, '1' and if it is a tail, then '0', while both events have probability exactly 1/2 and each toss is independent of one another [5]. According to the NIST hypothesis testing, a RNG under test is deemed 'bad' if the statistical properties of its random sequences indicate a clear deviation from the 'ideal', otherwise it is deemed 'good'. In our case, given a polynomial order and a secrecy selection scheme, the random sequence subject to the NIST test comprises $N \times L$ bits, where N is the total number of PUFs and L is the output length for each PUF. Since there are uncountable ways to identify non-random patterns, no finite set of statistical tests can certify that a RNG is truly random. Regardless, as the sample size and the number of independent statistical tests increase, we gain more evidence to support our decision regarding whether the RNG of concern is 'truly good'.[2] Consequently, instead of viewing one device as one RNG at a time, the conceptual RNG we test against generates multiple random sequences through multiple fabricated devices and can be characterized by the involved fabrication process, measurement circuitry, distiller and secret selection scheme. Given a set of PUF devices in identical design, a range of polynomial orders 0 through k, and random sequence generation rules $\boldsymbol{S}$ and $\boldsymbol{T}$, we form $2 \times (k+1)$ RNGs and test them individually. At the end of the test, a distiller is deemed 'good' if it passes all

[2]Independence of tests are discussed in Section 'Application of Multiple Tests' of [5].

tests according to NIST's guidelines.[3]

4.3 Experimental Results

Our test bench is the frequency characterization of a population of RO PUFs implemented on 125 Xilinx Spartan-3 FPGA devices. For each device, 512 ROs were placed as a 16 × 32 array and for each RO, 100 frequency readings were measured at room temperature [31].

4.3.1 Random Variation Distillation

Take the dataset of Chip No. 1 for instance, we first average the 100 measurement readings for each RO site and use the resulting 16 × 32 data points as its frequency variation profile. Next, we apply the polynomial-based distiller of order 0 through 6 to model the systematic variation respectively. The solutions of (4.5) with respect to order 0 through 6 are illustrated in Figure 4.6, while the corresponding residual terms, the distilled random variations, are depicted in Figure 4.7, where 0^{th}-order corresponds to the raw frequency profile subtracting the chip average. Notably, we see the 'bull's eye', i.e., the radial pattern close to the center, vanishing in the cases of 2^{nd} order model and beyond. From the histograms of the distilled random variations shows Figure 4.8, all of them appear drawing from a

[3]Tests like Overlapping Template Matching Test, Linear Complexity Test, Random Excursions Test and Random Excursion Variant Test require a random sequence at least a million bit long and thus may not be applicable to a True RNG like ours. To clarify, testing on a Pseudo RNG seeded with ours is not the main interest to show of this work.

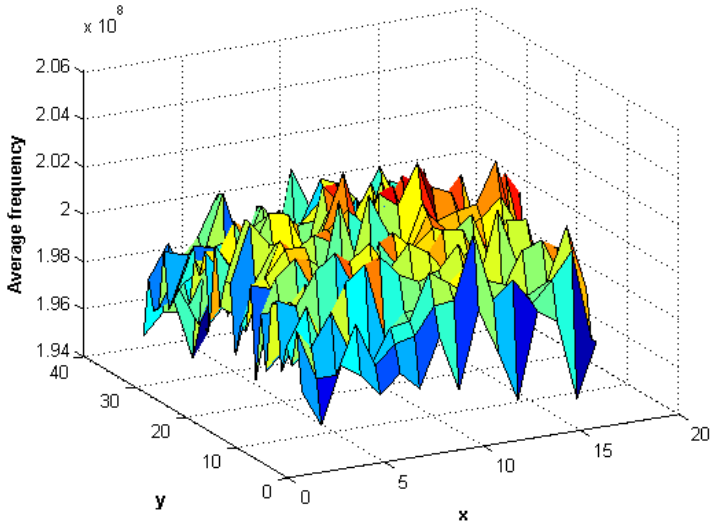

Figure 4.5: The fabrication variation of Chip No. 1 [31] with the z-axis indicating the average value of the 100 frequency readings of each RO site.

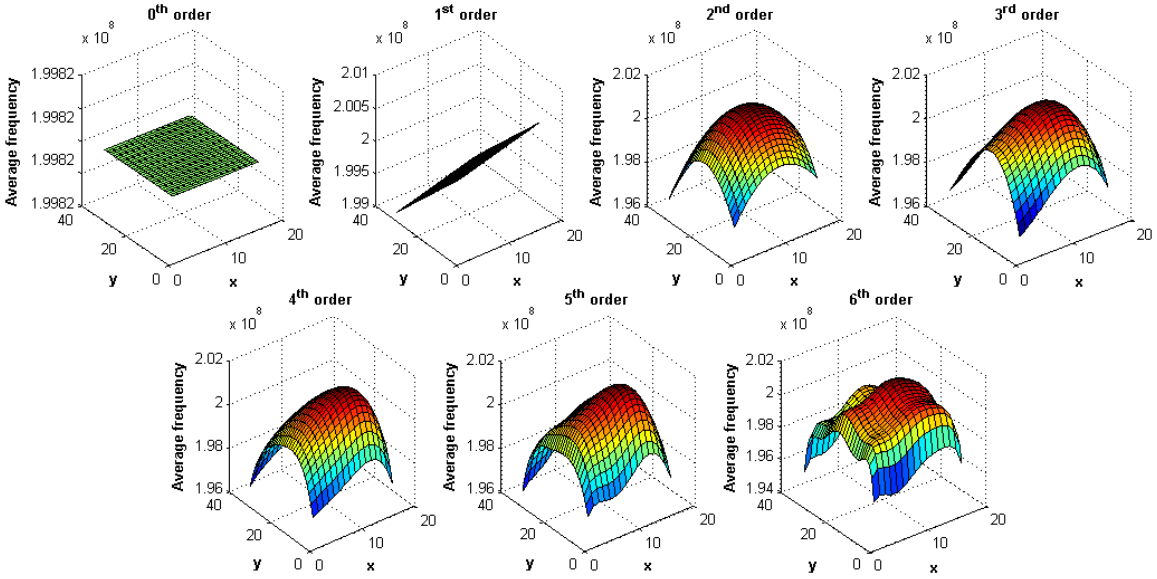

Figure 4.6: The modeled systematic variation after applying 0^{th} through 6^{th}-order polynomial regression to the dataset of Chip No. 1.

normal distribution with a diminishing variance as the order of the applied model increases.

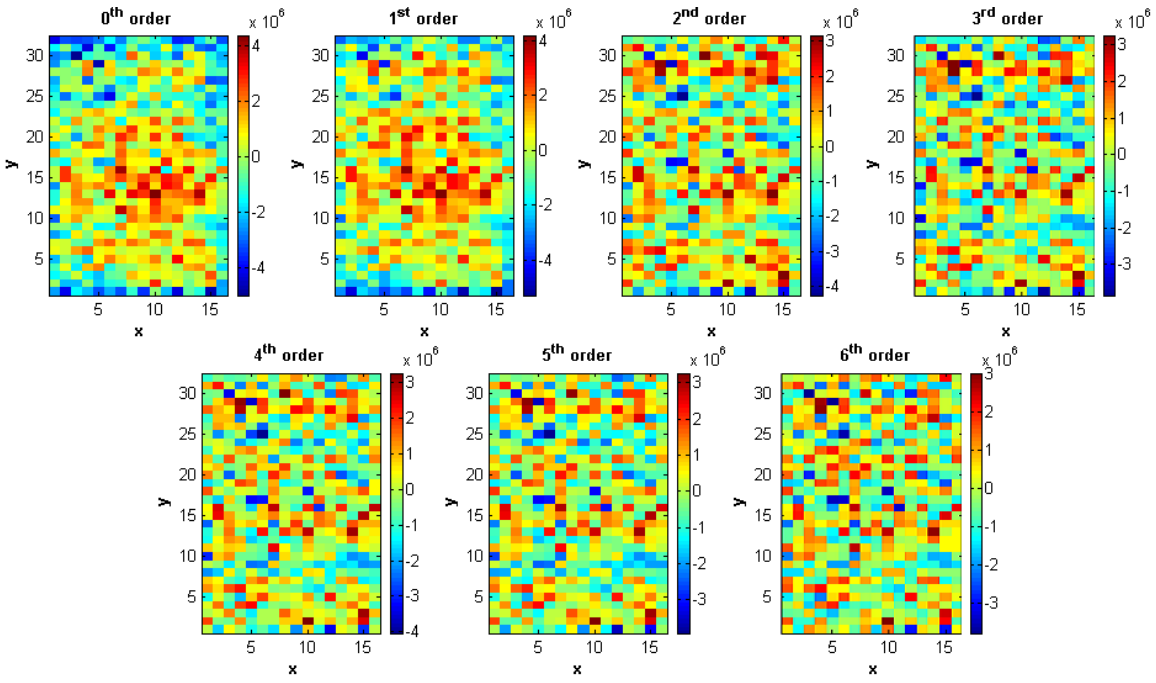

Figure 4.7: The distilled random variation after applying 0^{th} through 6^{th}-order polynomial regression to the dataset of Chip No. 1. Notably, we see the 'bull's eye', i.e., the radial pattern close to the center, vanishing in the cases of 2^{nd} order model and beyond.

4.3.2 Model Selection

4.3.2.1 Random Sequences $\boldsymbol{S}$ and $\boldsymbol{T}$

For each RNG, the input file fed to the test suite contains 32000 bits, a concatenation of the 256-bit bitstrings generated from Chip No. 1–125. Tests applicable to this length include Frequency Test, Block Frequency Test, Cumulative Sums Test, Runs Test, Longest Run Test, Serial Test and Approximate Entropy Test. Parameters are chosen according to NIST recommendations, in particular, block length $M = 32$ for Block Frequency Test, block length $m = 2$ for Approximate Entropy

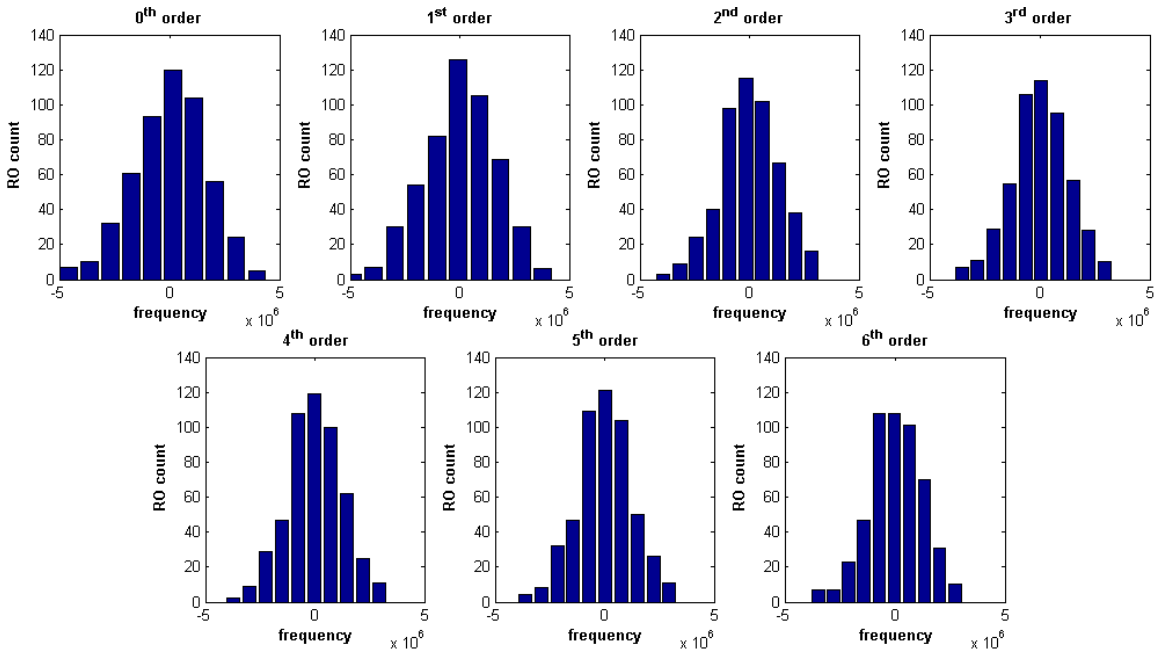

Figure 4.8: The histogram of the distilled random variation after applying 0^{th} through 6^{th}-order polynomial regression to the dataset of Chip No. 1. It is difficult to judge simply from the chart which model is the best fit without running NIST tests for all appears normal but with a diminishing variance as the order increases

Test and block length $m = 5$ for Serial Test. According to [5], empirical results have to be interpreted in two forms of analysis: First, the proportion of sequences passing a test shall be above a the minimum rate, 0.96 in our case, i.e., to pass 120 sequences out of a sample size of 125 sequences at significance level $\alpha = 0.01$. Secondly, the *P-values* of all the random sequences shall be uniformly distributed. Based on χ^2 Goodness-of-Fit Test, the underlying distribution is deemed uniform if the *P-value* of the *P-values* is equal or greater than 0.0001 given a population of 125 sequences. Whenever either of these two approaches fails, further analysis

drawn from a different sample space is necessary to conclude the failure either as a statistical anomaly or a clear non-randomness. The left four columns of Table 4.2 [4] summarizes the test results of random sequences $\boldsymbol{S}$ and $\boldsymbol{T}$; see Table 4.5 and 4.8 for complete C1–C10 distribution of 'P-VALUE OF P-VALUES'. As the 0^{th}-order section shows, random sequences generated without entropy distillation fail miserably for both forms of analysis 'PROP. (PROPORTION)' and 'P-VAL. (P-VALUE OF P-VALUES)', where '*' marks a failure. This strongly suggests the existence of systematic variation in the raw data. The failure rate decrease sharply when applied with 1^{st}-, 2^{nd}- or 3^{rd}-order distiller in the case of $\boldsymbol{S}$ and with 2^{nd}- or 3^{rd}-order distiller in the case of $\boldsymbol{T}$. Unfortunately, there is at least one failure with respect to $\boldsymbol{S}$, though the failure is only slightly below the cutting value. In such a boarder case where a weak existence of systematic variation is inferred, further investigation with different dataset is necessary to conclude the RNGs and the corresponding distillers 'good' or 'bad'. If simply taking the sum of failure rates with respect to $\boldsymbol{S}$ and $\boldsymbol{T}$, either 2^{nd}- or 3^{rd}-order distillers can be considered optimal. To break the tie, one may favor the 2^{nd}-order distiller due to computational simplicity and hardware efficiency as the latter is contingent on the variance of residual terms as discussed earlier with Figure 4.8. Still, one may prefer the 3^{rd}-order because in general it

[4] The seven sections correspond to distillers of order 0^{th} (top) through 6^{th} (bottom). In each section, the nine rows corresponds to 1) Frequency Test, 2) Block Frequency Test, 3) Cumulative Sums Test $(m-2)$, 4) Cumulative Sums Test $(m-3)$, 5) Runs Test, 6) Longest Run Test, 7) Approximate Entropy Test, 8) Serial Test (forward), 9) Serial Test (backward). The last two columns correspond to two forms of analysis while '*' marks a failure.

passes more sequences in the 'PROPORTION' analysis while yielding comparable performance in the 'P-VALUE OF P-VALUES' analysis. Moreover, the pass rate of the 'P-VALUE OF P-VALUES' analysis drops when applied with a model in 4^{th} order or beyond, whereas the pass rate of the 'PROPORTION' analysis remains unchanged. The rate drop is due to the clustering phenomenon of *P-values* towards the C10 side, as opposed to the clustering phenomenon towards the C1 side in the low order cases. These can be used as indicators for model over-fitting and model under-fitting respectively.

4.3.2.2 1-out-of-8 Coding

To evaluate the effectiveness of our distiller upon existing PUF designs, the general secret selection schemes are substituted with 1-out-of-8 Coding and Neighbor Coding. In the case of 1-out-of-8 Coding, a 3-bit index '000', '001',...,'110', or '111' is generated by pointing to the fastest RO out of 8 consecutive ROs in the same row, resulting a random sequence of length 192 bits per device.[5] Accordingly, block sizes are set as 32 for Block Frequency Test, 1 for Approximate Entropy Test and 4 for Serial Test. For 1-out-of-8 Coding, a 3-bit index '000', '001',...,'110', or '111' is generated by pointing to the fastest RO out of 8 consecutive ROs in the same row, resulting a random sequence of length 192 bits per device. [6] Accordingly, block sizes are set as 32 for Block Frequency Test, 1 for Approximate Entropy Test and 4 for Serial Test. Table 4.2 shows that distillers of 4^{th} order and beyond

[5]The original design takes 8 independent *pairs* of ROs to generate one index.

[6]The original design takes 8 independent *pairs* of ROs to generate one index.

	S		***T***		**1-out-of-8**		**Chain-like**		**Decoupled**		
	P-VAL.	PRO.	P-VAL.	PRO.	P-VAL.	PRO.	P-VAL.	PRO.	P-VAL.	PRO.	STAT TEST
0^{th}-order	0.000000	*45 *	0.000000	*38 *	0.013689	122	0.000072	*125	0.000003	*115 *	Frequency
	0.000000	*59 *	0.000000	*49 *	0.166594	125	0.000000	*125	0.050764	120	BlockFrequency
	0.000000	*46 *	0.000000	*39 *	0.231636	121	0.000000	*125	0.000000	*119 *	CumulativeSums (m-2)
	0.000000	*46 *	0.000000	*38 *	0.059743	122	0.000000	*125	0.000000	*118 *	CumulativeSums (m-3)
	0.000000	*65 *	0.000000	*31 *	0.002320	117 *	0.000000	*0 *	0.302788	120	Runs
	0.000000	*66 *	0.000000	*44 *	0.000603	123	0.000000	*62 *	0.000062	*124	LongestRun
	0.000000	*53 *	0.000000	*23 *	0.000001	*117 *	0.000000	*0 *	0.000001	*119 *	ApproximateEntropy
	0.000000	*65 *	0.000000	*25 *	0.004904	124	0.000000	*1 *	0.070160	116 *	Serial (forward)
	0.000000	*103 *	0.000000	*74 *	0.552185	125	0.000000	*117 *	0.192277	123	Serial (backward)
1^{st}-order	0.166594	124	0.003598	122	0.000949	120	0.000072	*125	0.130323	124	Frequency
	0.000002	*120	0.889414	121	0.529142	125	0.000000	*125	0.056599	122	BlockFrequency
	0.000100	120	0.136969	122	0.063046	120	0.000000	*125	0.082208	124	CumulativeSums (m-2)
	0.405918	122	0.020616	119 *	0.043046	120	0.000000	*125	0.034444	123	CumulativeSums (m-3)
	0.082208	124	0.000000	*68 *	0.192277	124	0.000000	*0 *	0.096097	122	Runs
	0.048059	120	0.000000	*90 *	0.000067	*123	0.000000	*62 *	0.000274	124	LongestRun
	0.025948	120	0.000000	*75 *	0.002471	120	0.000000	*0 *	0.130323	122	ApproximateEntropy
	0.112055	122	0.000000	*80 *	0.262219	124	0.000000	*1 *	0.956806	122	Serial (forward)
	0.474938	121	0.000000	*117 *	0.551044	125	0.000000	*117 *	0.620686	123	Serial (backward)

Table 4.2: The results of NIST 'P-VAL. (P-VALUE of P-VALUES)' and 'PROP. (PROPORTION)' analyses with respect to random sequences generated by $\boldsymbol{S}$, $\boldsymbol{T}$, 1-out-of-8 Coding, Chain-like Neighbor Coding and Decoupled Neighbor Coding accompanied by 0^{th}- to 1^{st}-order distillers, where '*' marks a failure.

	S		***T***		**1-out-of-8**		**Chain-like**		**Decoupled**		
	P-VAL.	PRO.	P-VAL.	PRO.	P-VAL.	PRO.	P-VAL.	PRO.	P-VAL.	PRO.	STAT TEST
2^{nd}-order	0.369588	122	0.012159	121	0.000000	*115 *	0.001228	125	0.086622	121	Frequency
	0.000782	122	0.422488	122	0.059743	124	0.000000	*125	0.262219	123	BlockFrequency
	0.020616	120	0.086622	120	0.000000	*118 *	0.000000	*125	0.073984	123	CumulativeSums (m-2)
	0.575157	122	0.066516	122	0.000000	*116 *	0.000000	*125	0.389809	120	CumulativeSums (m-3)
	0.316158	125	0.915772	122	0.552185	123	0.000000	*0 *	0.493319	124	Runs
	0.000062	*122	0.000782	120	0.000000	*123	0.000000	*58 *	0.000115	124	LongestRun
	0.750075	124	0.474938	120	0.000000	*120	0.000000	*0 *	0.316158	122	ApproximateEntropy
	0.874833	124	0.077998	122	0.000123	123	0.000000	*0 *	0.643139	121	Serial (forward)
	0.231636	123	0.302788	125	0.457002	124	0.000000	*110 *	0.262219	123	Serial (backward)
3^{rd}-order	0.011457	125	0.136969	124	0.000000	*111 *	0.000000	*125	0.389809	123	Frequency
	0.262219	124	0.551044	125	0.003829	123	0.000000	*125	0.457002	122	BlockFrequency
	0.002320	125	0.000131	125	0.000000	*112 *	0.000000	*125	0.551044	124	CumulativeSums (m-2)
	0.002984	125	0.017315	125	0.000000	*111 *	0.000000	*125	0.192277	123	CumulativeSums (m-3)
	0.643139	124	0.529142	121	0.889414	124	0.000000	*0 *	0.529142	124	Runs
	0.000058	*123	0.012903	124	0.000000	*120	0.000000	*48 *	0.000000	*123	LongestRun
	0.020616	125	0.422488	123	0.000000	*114 *	0.000000	*0 *	0.889414	125	ApproximateEntropy
	0.369588	124	0.915772	123	0.000017	*121	0.000000	*0 *	0.493319	123	Serial (forward)
	0.439517	124	0.506075	122	0.575157	124	0.000000	*114 *	0.439517	125	Serial (backward)
4^{th}-order	0.000001	*125	0.000000	*125	0.000407	121	0.000000	*125	0.192277	124	Frequency
	0.166594	125	0.000051	*125	0.344248	123	0.000000	*125	0.807956	124	BlockFrequency
	0.000000	*125	0.000000	*125	0.143910	121	0.000000	*125	0.070160	124	CumulativeSums (m-2)
	0.000011	*125	0.000000	*124	0.130323	120	0.000000	*125	0.117876	124	CumulativeSums (m-3)
	0.316158	125	0.903069	122	0.437182	125	0.000000	*0 *	0.414457	125	Runs
	0.004904	123	0.045489	125	0.007522	120	0.000000	*54 *	0.000000	*123	LongestRun
	0.289860	125	0.265309	122	0.708591	122	0.000000	*0 *	0.143910	122	ApproximateEntropy
	0.571108	125	0.665311	123	0.571108	125	0.000000	*1 *	0.457002	123	Serial (forward)
	0.405918	123	0.283039	124	0.551044	125	0.000000	*110 *	0.825875	124	Serial (backward)

Table 4.3: Cont'd with the results using $2^{nd} - 4^{th}$ order models

	S		***T***		**1-out-of-8**		**Chain-like**		**Decoupled**		
	P-VAL.	PRO.	P-VAL.	PRO.	P-VAL.	PRO.	P-VAL.	PRO.	P-VAL.	PRO.	STAT TEST
5^{th}-order	0.000029	*125	0.211194	125	0.316158124		0.000000	*125	0.096097	125	Frequency
	0.004074	125	0.000000	*125	0.493319124		0.000000	*125	0.552185	124	BlockFrequency
	0.000000	*125	0.000000	*125	0.665311124		0.000000	*125	0.043046	124	CumulativeSums (m-2)
	0.000000	*125	0.000000	*125	0.166594123		0.000000	*125	0.036430	125	CumulativeSums (m-3)
	0.493319	124	0.687147	124	0.036430125		0.000000	*0 *	0.192277	123	Runs
	0.006661	124	0.001801	125	0.036430124		0.000000	*42 *	0.000000	*124	LongestRun
	0.059743	125	0.302788	124	0.729586125		0.000000	*0 *	0.665311	122	ApproximateEntropy
	0.687147	125	0.304210	121	0.096097125		0.000000	*0 *	0.512137	124	Serial (forward)
	0.262219	123	0.789315	123	0.457002125		0.000000	*114 *	0.474938	125	Serial (backward)
6^{th}-order	0.000009	*125	0.001586	125	0.001080125		0.000000	*125	0.231636	122	Frequency
	0.000000	*125	0.000000	*125	0.086622125		0.000000	*125	0.437182	123	BlockFrequency
	0.000000	*125	0.000000	*125	0.231636125		0.000000	*125	0.091249	123	CumulativeSums (m-2)
	0.000000	*125	0.000000	*125	0.050764124		0.000000	*125	0.211194	123	CumulativeSums (m-3)
	0.101175	125	0.130323	123	0.422488124		0.000000	*0 *	0.529142	122	Runs
	0.000643	124	0.007992	124	0.211194125		0.000000	*44 *	0.000017	*122	LongestRun
	0.000006	*125	0.643139	124	0.130323124		0.000000	*0 *	0.598008	124	ApproximateEntropy
	0.552185	124	0.289860	123	0.329976124		0.000000	*0 *	0.277369	123	Serial (forward)
	0.874833	124	0.529142	124	0.405918124		0.000000	*113 *	0.843024	121	Serial (backward)

Table 4.4: Cont'd with the results using $5^{th} - 6^{th}$ order models

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROP.	STATISTICAL TEST
0^{th}-order	93	7	3	4	2	6	1	2	4	3	0.000000 *	45/125 *	Frequency
	95	8	8	3	3	1	1	4	0	2	0.000000 *	59/125 *	BlockFrequency
	95	6	3	5	4	2	3	0	3	4	0.000000 *	46/125 *	CumulativeSums (m-2)
	96	7	3	3	7	1	4	1	0	3	0.000000 *	46/125 *	CumulativeSums (m-3)
	67	5	5	13	9	6	6	4	7	3	0.000000 *	65/125 *	Runs
	82	6	7	11	6	3	1	3	3	3	0.000000 *	66/125 *	LongestRun
	88	10	4	7	3	6	1	2	2	2	0.000000 *	53/125 *	ApproximateEntropy
	81	8	7	5	10	3	1	6	1	3	0.000000 *	65/125 *	Serial (forward)
	41	12	16	6	16	10	8	5	5	6	0.000000 *	103/125 *	Serial (backward)
1^{st}-order	13	19	12	12	13	7	11	7	11	20	0.166594	124/125	Frequency
	29	20	16	16	8	9	8	6	5	8	0.000002 *	120/125	BlockFrequency
	18	26	15	16	4	8	9	9	15	5	0.000100	120/125	CumulativeSums (m-2)
	18	14	17	14	16	10	10	8	9	9	0.405918	122/125	CumulativeSums (m-3)
	12	15	19	14	11	20	9	5	9	11	0.082208	124/125	Runs
	16	17	15	21	13	7	8	13	7	8	0.048059	120/125	LongestRun
	24	14	10	15	9	10	10	15	13	5	0.025948	120/125	ApproximateEntropy
	18	16	13	15	6	11	16	4	14	12	0.112055	122/125	Serial (forward)
	15	15	17	11	10	11	7	12	9	18	0.474938	121/125	Serial (backward)

Table 4.5: NIST test results with respect to random sequence $\boldsymbol{S}$ using 0^{th} and 1^{st} distillers, where $M = 32$ for Block Frequency Test, $m = 2$ for Approximate Entropy Test and $m = 5$ for Serial Test and '*' marks a failure.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROP.	STATISTICAL TEST
2^{nd}-order	17	12	16	13	8	11	12	9	10	17	0.369588	122/125	Frequency
	27	14	12	19	7	11	7	10	9	9	0.000782	122/125	BlockFrequency
	13	22	12	12	5	13	14	4	13	17	0.020616	120/125	CumulativeSums (m-2)
	15	16	14	10	9	11	17	9	11	13	0.575157	122/125	CumulativeSums (m-3)
	15	6	10	18	11	16	16	14	8	11	0.316158	125/125	Runs
	9	13	13	30	14	14	9	7	5	11	0.000062 *	122/125	LongestRun
	18	9	12	9	13	13	13	14	13	11	0.750075	124/125	ApproximateEntropy
	13	14	11	15	10	14	16	9	11	12	0.874833	124/125	Serial (forward)
	12	9	22	13	14	11	12	10	15	7	0.231636	123/125	Serial (backward)
3^{rd}-order	4	9	11	15	8	16	22	8	15	17	0.011457	125/125	Frequency
	8	14	14	14	6	16	11	12	17	13	0.262219	124/125	BlockFrequency
	6	5	11	10	11	14	18	8	23	19	0.002320	125/125	CumulativeSums (m-2)
	4	10	10	8	6	18	20	12	18	19	0.002984	125/125	CumulativeSums (m-3)
	9	10	14	14	11	18	11	15	13	10	0.643139	124/125	Runs
	9	5	14	24	24	14	7	5	11	12	0.000058 *	123/125	LongestRun
	5	12	12	7	15	14	20	10	9	21	0.020616	125/125	ApproximateEntropy
	7	17	15	12	9	12	16	12	15	10	0.369588	124/125	Serial (forward)
	12	19	17	12	13	12	14	10	6	10	0.439517	124/125	Serial (backward)
4^{th}-order	2	2	8	26	13	16	21	10	17	10	0.000001 *	125/125	Frequency
	11	7	8	7	11	15	16	16	19	15	0.166594	125/125	BlockFrequency
	2	3	5	11	10	24	19	8	20	23	0.000000 *	125/125	CumulativeSums (m-2)
	2	4	5	15	13	16	16	17	11	26	0.000011 *	125/125	CumulativeSums (m-3)
	11	11	13	5	16	13	10	19	16	11	0.316158	125/125	Runs
	9	6	14	22	20	10	17	5	11	11	0.004904	123/125	LongestRun
	4	11	9	15	13	12	12	17	15	17	0.289860	125/125	ApproximateEntropy
	14	11	6	13	10	15	11	12	14	19	0.571108	125/125	Serial (forward)
	8	16	11	21	11	11	9	13	13	12	0.405918	123/125	Serial (backward)

Table 4.6: Cont'd with the results using $2^{nd} - 4^{th}$ order models

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROP.	STATISTICAL TEST
5^{th}-order	0	5	12	20	13	10	11	20	11	23	0.000029 *	125/125	Frequency
	6	10	8	8	11	16	15	14	11	26	0.004074	125/125	BlockFrequency
	0	4	8	9	14	14	16	10	16	34	0.000000 *	125/125	CumulativeSums (m-2)
	0	8	4	13	10	16	10	12	24	28	0.000000 *	125/125	CumulativeSums (m-3)
	9	13	16	10	10	8	13	11	19	16	0.493319	124/125	Runs
	9	11	18	23	17	7	13	4	12	11	0.006661	124/125	LongestRun
	5	9	11	11	14	15	22	15	15	8	0.059743	125/125	ApproximateEntropy
	15	12	15	13	10	11	11	10	18	10	0.687147	125/125	Serial (forward)
	19	11	17	13	8	11	14	12	11	9	0.262219	123/125	Serial (backward)
6^{th}-order	4	8	2	19	10	8	14	15	22	23	0.000009 *	125/125	Frequency
	2	8	12	9	6	9	10	18	20	31	0.000000 *	125/125	BlockFrequency
	2	5	4	11	7	11	20	7	19	39	0.000000 *	125/125	CumulativeSums (m-2)
	3	9	5	5	5	14	13	9	18	44	0.000000 *	125/125	CumulativeSums (m-3)
	8	18	8	6	19	14	12	17	13	10	0.101175	125/125	Runs
	7	6	16	22	23	9	7	12	8	15	0.000643	124/125	LongestRun
	1	5	16	14	13	14	9	11	13	29	0.000006 *	125/125	ApproximateEntropy
	9	6	12	14	14	13	13	13	16	15	0.552185	124/125	Serial (forward)
	8	14	11	13	10	16	13	13	14	13	0.874833	124/125	Serial (backward)

Table 4.7: Cont'd with the results using $5^{th} - 6^{th}$ order models

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROP.	STATISTICAL TEST
0^{th}-order	100	6	3	4	2	1	2	2	0	5	0.000000 *	38/125 *	Frequency
	88	9	3	5	4	4	5	1	6	0	0.000000 *	49/125 *	BlockFrequency
	100	5	4	5	1	3	2	2	0	3	0.000000 *	39/125 *	CumulativeSums (m-2)
	100	10	0	6	1	3	0	2	1	2	0.000000 *	38/125 *	CumulativeSums (m-3)
	108	7	4	1	2	0	1	1	1	0	0.000000 *	31/125 *	Runs
	100	4	7	7	3	2	0	1	1	0	0.000000 *	44/125 *	LongestRun
	114	3	2	1	1	1	1	2	0	0	0.000000 *	23/125 *	ApproximateEntropy
	112	5	4	1	1	0	1	1	0	0	0.000000 *	25/125 *	Serial (forward)
	73	11	8	7	10	3	4	2	1	6	0.000000 *	74/125 *	Serial (backward)
1^{st}-order	25	8	10	13	6	11	7	13	13	19	0.003598	122/125	Frequency
	14	8	13	16	12	14	10	13	12	13	0.889414	121/125	BlockFrequency
	22	11	13	7	13	13	15	12	6	13	0.136969	122/125	CumulativeSums (m-2)
	25	14	11	13	12	9	8	6	15	12	0.020616	119/125 *	CumulativeSums (m-3)
	86	14	4	6	4	4	1	2	2	2	0.000000 *	68/125 *	Runs
	66	15	13	16	5	3	3	3	0	1	0.000000 *	90/125 *	LongestRun
	78	15	7	9	2	5	2	2	2	3	0.000000 *	75/125 *	ApproximateEntropy
	80	6	9	6	4	6	6	0	5	3	0.000000 *	80/125 *	Serial (forward)
	40	15	13	14	6	8	6	6	7	10	0.000000 *	117/125 *	Serial (backward)

Table 4.8: NIST test results with respect to random sequence $\boldsymbol{T}$ using 0^{th} and 1^{st} distillers , where $M = 32$ for Block Frequency Test, $m = 2$ for Approximate Entropy Test and $m = 5$ for Serial Test and '*' marks a failure.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROP.	STATISTICAL TEST
2^{nd}-order	20	15	14	22	6	9	8	12	7	12	0.012159	121/125	Frequency
	13	11	15	14	10	18	9	9	8	18	0.422488	122/125	BlockFrequency
	20	12	6	18	13	10	11	7	11	17	0.086622	120/125	CumulativeSums (m-2)
	18	15	9	15	9	17	13	2	13	14	0.066516	122/125	CumulativeSums (m-3)
	16	14	14	11	9	10	12	14	13	12	0.915772	122/125	Runs
	18	8	15	18	24	11	6	13	6	6	0.000782	120/125	LongestRun
	19	14	17	12	13	7	9	10	11	13	0.474938	120/125	ApproximateEntropy
	19	6	11	13	11	18	16	8	16	7	0.077998	122/125	Serial (forward)
	16	16	9	10	11	9	11	21	12	10	0.302788	125/125	Serial (backward)
3^{rd}-order	10	7	10	18	10	14	15	10	10	21	0.136969	124/125	Frequency
	10	10	12	16	9	8	13	12	17	18	0.551044	125/125	BlockFrequency
	8	9	8	15	9	10	27	5	14	20	0.000131	125/125	CumulativeSums (m-2)
	8	8	9	10	14	8	17	11	16	24	0.017315	125/125	CumulativeSums (m-3)
	14	13	13	19	11	7	12	12	11	13	0.529142	121/125	Runs
	9	11	11	24	20	10	11	11	12	6	0.012903	124/125	LongestRun
	15	10	11	6	14	11	9	18	15	16	0.422488	123/125	ApproximateEntropy
	11	9	11	15	12	12	15	11	14	15	0.915772	123/125	Serial (forward)
	11	18	15	9	13	12	9	10	12	16	0.506075	122/125	Serial (backward)
4^{th}-order	3	6	18	26	12	10	4	9	12	25	0.000000 *	125/125	Frequency
	6	10	5	5	12	15	8	19	22	23	0.000051 *	125/125	BlockFrequency
	2	5	8	18	15	9	18	8	11	31	0.000000 *	125/125	CumulativeSums (m-2)
	2	6	11	15	12	20	14	4	12	29	0.000000 *	124/125	CumulativeSums (m-3)
	14	13	12	13	8	11	11	13	14	16	0.903069	122/125	Runs
	12	6	23	15	16	8	13	8	13	11	0.045489	125/125	LongestRun
	18	7	11	7	13	19	12	11	12	15	0.265309	122/125	ApproximateEntropy
	13	16	8	13	13	15	13	15	8	11	0.665311	123/125	Serial (forward)
	14	13	13	12	12	13	18	15	10	5	0.283039	124/125	Serial (backward)

Table 4.9: Cont'd with the results using $2^{nd} - 4^{th}$ order models

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROP.	STATISTICAL TEST
5^{th}-order	6	10	11	17	9	14	10	20	15	13	0.211194	125/125	Frequency
	6	5	4	6	7	13	18	13	22	31	0.000000 *	125/125	BlockFrequency
	2	8	8	19	5	9	16	10	19	29	0.000000 *	125/125	CumulativeSums (m-2)
	3	9	5	12	8	14	18	6	14	36	0.000000 *	125/125	CumulativeSums (m-3)
	13	17	9	17	11	10	13	11	11	13	0.687147	124/125	Runs
	11	9	17	25	18	9	13	5	7	11	0.001801	125/125	LongestRun
	9	15	15	7	14	14	15	18	12	6	0.302788	124/125	ApproximateEntropy
	13	11	11	10	10	11	17	11	20	11	0.304210	121/125	Serial (forward)
	12	13	13	16	10	16	11	12	14	8	0.789315	123/125	Serial (backward)
6^{th}-order	2	13	10	14	11	11	20	9	11	24	0.001586	125/125	Frequency
	5	2	4	8	8	8	16	22	19	33	0.000000 *	125/125	BlockFrequency
	1	4	11	10	10	12	12	15	19	31	0.000000 *	125/125	CumulativeSums (m-2)
	2	6	7	10	6	13	17	7	20	37	0.000000 *	125/125	CumulativeSums (m-3)
	23	14	12	14	7	10	13	8	13	11	0.130323	123/125	Runs
	9	12	13	24	19	6	15	8	9	10	0.007992	124/125	LongestRun
	13	17	11	18	12	10	11	12	11	10	0.643139	124/125	ApproximateEntropy
	11	14	11	18	11	13	11	4	17	15	0.289860	123/125	Serial (forward)
	15	9	14	13	11	13	15	14	6	15	0.529142	124/125	Serial (backward)

Table 4.10: Cont'd with the results using $5^{th} - 6^{th}$ order models

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROP.	STATISTICAL TEST
0^{th}-order	21	15	7	20	11	13	9	14	11	4	0.013689	122/125	Frequency
	7	19	19	10	10	12	13	7	13	15	0.166594	125/125	BlockFrequency
	21	9	9	14	13	10	18	10	11	10	0.231636	121/125	CumulativeSums (m-2)
	22	10	9	17	13	7	11	8	11	17	0.059743	122/125	CumulativeSums (m-3)
	26	20	13	11	10	11	8	8	9	9	0.002320	117/125 *	Runs
	22	13	16	13	22	11	3	11	9	5	0.000603	123/125	LongestRun
	30	21	15	8	13	7	10	3	9	9	0.000001 *	117/125 *	ApproximateEntropy
	27	16	14	11	11	8	10	11	8	9	0.004904	124/125	Serial (forward)
	10	16	18	15	9	11	10	12	11	13	0.552185	125/125	Serial (backward)
1^{st}-order	24	21	13	14	9	10	11	13	4	6	0.000949	120/125	Frequency
	11	18	16	10	10	11	14	15	10	10	0.529142	125/125	BlockFrequency
	18	22	13	11	11	8	13	6	14	9	0.063046	120/125	CumulativeSums (m-2)
	21	17	17	6	14	6	13	11	9	11	0.043046	120/125	CumulativeSums (m-3)
	17	19	13	15	6	16	11	10	10	8	0.192277	124/125	Runs
	13	13	13	13	27	20	7	9	5	5	0.000067 *	123/125	LongestRun
	22	20	16	14	12	15	7	6	7	6	0.002471	120/125	ApproximateEntropy
	12	14	16	16	13	16	14	7	8	9	0.262219	124/125	Serial (forward)
	8	14	13	15	14	16	14	8	7	16	0.551044	125/125	Serial (backward)

Table 4.11: NIST test results with respect to the random sequence generated by **1-out-of-8 Coding** using 0^{th} and 1^{st} distillers, where $M = 32$ for Block Frequency Test, $m = 1$ for Approximate Entropy Test and $m = 4$ for Serial Test and '*' marks a failure.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROP.	STATISTICAL TEST
2^{nd}-order	46	21	6	16	6	12	7	7	3	1	0.000000 *	115/125 *	Frequency
	20	18	10	15	9	7	18	10	10	8	0.059743	124/125	BlockFrequency
	41	17	15	10	5	9	9	6	6	7	0.000000 *	118/125 *	CumulativeSums (m-2)
	39	16	14	9	11	6	10	2	12	6	0.000000 *	116/125 *	CumulativeSums (m-3)
	19	10	13	12	9	13	14	9	14	12	0.552185	123/125	Runs
	21	11	27	12	23	14	2	6	4	5	0.000000 *	123/125	LongestRun
	41	18	10	12	9	10	5	9	8	3	0.000000 *	120/125	ApproximateEntropy
	15	28	16	10	8	12	16	8	7	5	0.000123	123/125	Serial (forward)
	14	10	9	14	14	15	19	13	11	6	0.457002	124/125	Serial (backward)
3^{rd}-order	55	14	4	15	7	8	6	3	8	5	0.000000 *	111/125 *	Frequency
	22	19	20	11	13	8	9	6	9	8	0.003829	123/125	BlockFrequency
	47	21	9	9	5	5	10	3	8	8	0.000000 *	112/125 *	CumulativeSums (m-2)
	47	19	7	4	13	4	12	4	12	3	0.000000 *	111/125 *	CumulativeSums (m-3)
	15	10	13	13	14	13	15	8	13	11	0.889414	124/125	Runs
	21	16	33	12	17	10	3	7	4	2	0.000000 *	120/125	LongestRun
	42	17	13	7	10	11	8	5	8	4	0.000000 *	114/125 *	ApproximateEntropy
	26	25	12	9	7	7	12	12	7	8	0.000017 *	121/125	Serial (forward)
	16	12	13	11	9	16	11	17	11	9	0.575157	124/125	Serial (backward)
4^{th}-order	19	23	7	16	8	7	15	17	10	3	0.000407	121/125	Frequency
	19	14	9	15	11	16	15	7	9	10	0.344248	123/125	BlockFrequency
	20	20	13	11	13	7	10	10	9	12	0.143910	121/125	CumulativeSums (m-2)
	19	17	14	12	10	4	15	9	15	10	0.130323	120/125	CumulativeSums (m-3)
	10	13	8	14	14	12	16	9	18	11	0.437182	125/125	Runs
	15	12	23	18	17	8	8	6	10	8	0.007522	120/125	LongestRun
	19	11	15	10	13	11	12	11	12	11	0.708591	122/125	ApproximateEntropy
	12	15	6	10	11	13	19	14	14	11	0.571108	125/125	Serial (forward)
	9	7	9	15	14	17	10	13	16	15	0.551044	125/125	Serial (backward)

Table 4.12: Cont'd with the results using $2^{nd} - 4^{th}$ order models

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROP.	STATISTICAL TEST
5^{th}-order	12	18	8	18	11	12	9	17	12	8	0.316158	124/125	Frequency
	12	16	13	9	15	19	10	9	14	8	0.493319	124/125	BlockFrequency
	14	14	14	11	12	10	17	7	12	14	0.665311	124/125	CumulativeSums (m-2)
	10	15	18	8	10	11	18	6	17	12	0.166594	123/125	CumulativeSums (m-3)
	9	10	12	23	6	11	10	12	19	13	0.036430	125/125	Runs
	11	18	18	15	20	11	9	8	10	5	0.036430	124/125	LongestRun
	12	12	13	9	9	18	14	11	14	13	0.729586	125/125	ApproximateEntropy
	7	14	8	6	14	20	10	16	14	16	0.096097	125/125	Serial (forward)
	9	9	11	10	12	10	17	13	14	20	0.457002	125/125	Serial (backward)
6^{th}-order	7	20	5	18	15	15	12	12	19	2	0.001080	125/125	Frequency
	7	18	5	15	14	10	17	11	10	18	0.086622	125/125	BlockFrequency
	9	18	11	13	12	9	12	8	21	12	0.231636	125/125	CumulativeSums (m-2)
	6	15	18	10	10	6	20	10	16	14	0.050764	124/125	CumulativeSums (m-3)
	5	9	9	13	16	16	13	14	14	16	0.422488	124/125	Runs
	10	14	18	12	21	11	7	10	11	11	0.211194	125/125	LongestRun
	4	12	7	16	15	9	14	15	17	16	0.130323	124/125	ApproximateEntropy
	11	7	9	15	15	11	9	19	17	12	0.329976	124/125	Serial (forward)
	15	15	15	14	7	6	13	9	16	15	0.405918	124/125	Serial (backward)

Table 4.13: Cont'd with the results using $5^{th} - 6^{th}$ order models

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROP.	STATISTICAL TEST
0^{th}-order	0	9	6	12	12	20	15	23	10	18	0.000072 *	125/125	Frequency
	0	0	0	1	0	1	4	4	15	100	0.000000 *	125/125	BlockFrequency
	0	0	8	5	5	9	13	16	9	60	0.000000 *	125/125	CumulativeSums (m-2)
	0	3	8	2	5	10	13	19	12	53	0.000000 *	125/125	CumulativeSums (m-3)
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	Runs
	113	6	4	0	2	0	0	0	0	0	0.000000 *	62/125 *	LongestRun
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	ApproximateEntropy
	125	0	0	0	0	0	0	0	0	0	0.000000 *	1/125 *	Serial (forward)
	36	21	12	11	8	8	9	8	7	5	0.000000 *	117/125 *	Serial (backward)
1^{st}-order	0	9	6	12	12	20	15	23	10	18	0.000072 *	125/125	Frequency
	0	0	0	1	0	1	4	4	15	100	0.000000 *	125/125	BlockFrequency
	0	0	8	5	5	9	13	16	9	60	0.000000 *	125/125	CumulativeSums (m-2)
	0	3	8	2	5	10	13	19	12	53	0.000000 *	125/125	CumulativeSums (m-3)
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	Runs
	113	6	4	0	2	0	0	0	0	0	0.000000 *	62/125 *	LongestRun
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	ApproximateEntropy
	125	0	0	0	0	0	0	0	0	0	0.000000 *	1/125 *	Serial (forward)
	36	21	12	11	8	8	9	8	7	5	0.000000 *	117/125 *	Serial (backward)

Table 4.14: NIST test results with respect to the random sequence generated by the **original Neighbor Coding** using using $0^{th} - 1^{st}$ order models, where $M = 32$ for Block Frequency Test, $m = 2$ for Approximate Entropy Test and $m = 5$ for Serial Test and '*' marks a failure.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROP.	STATISTICAL TEST
2^{nd}-order	1	6	10	9	14	17	15	22	17	14	0.001228	125/125	Frequency
	0	0	0	0	2	2	6	2	20	93	0.000000 *	125/125	BlockFrequency
	0	2	6	6	6	11	12	14	7	61	0.000000 *	125/125	CumulativeSums (m-2)
	1	1	4	11	7	7	13	10	14	57	0.000000 *	125/125	CumulativeSums (m-3)
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	Runs
	111	8	3	2	0	0	1	0	0	0	0.000000 *	58/125 *	LongestRun
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	ApproximateEntropy
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	Serial (forward)
	38	24	14	11	6	6	12	6	3	5	0.000000 *	110/125 *	Serial (backward)
3^{rd}-order	1	2	6	6	15	24	8	30	17	16	0.000000 *	125/125	Frequency
	0	0	0	0	2	1	3	7	11	101	0.000000 *	125/125	BlockFrequency
	1	0	3	4	3	7	7	23	16	61	0.000000 *	125/125	CumulativeSums (m-2)
	0	1	4	3	9	5	9	19	16	59	0.000000 *	125/125	CumulativeSums (m-3)
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	Runs
	115	6	2	0	1	1	0	0	0	0	0.000000 *	48/125 *	LongestRun
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	ApproximateEntropy
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	Serial (forward)
	38	27	10	12	11	9	6	6	4	2	0.000000 *	114/125 *	Serial (backward)
4^{th}-order	0	2	6	11	8	23	11	36	10	18	0.000000 *	125/125	Frequency
	0	0	0	0	1	2	1	6	14	101	0.000000 *	125/125	BlockFrequency
	0	2	1	3	5	7	9	17	12	69	0.000000 *	125/125	CumulativeSums (m-2)
	0	1	2	3	5	9	6	22	10	67	0.000000 *	125/125	CumulativeSums (m-3)
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	Runs
	113	8	2	0	0	0	2	0	0	0	0.000000 *	54/125 *	LongestRun
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	ApproximateEntropy
	125	0	0	0	0	0	0	0	0	0	0.000000 *	1/125 *	Serial (forward)
	40	22	15	14	3	10	7	6	4	4	0.000000 *	110/125 *	Serial (backward)

Table 4.15: Cont'd with the results using $2^{nd} - 4^{th}$ order models

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROP.	STATISTICAL TEST
5^{th}-order	2	3	6	6	10	16	15	22	17	28	0.000000 *	125/125	Frequency
	0	0	0	0	0	1	3	5	14	102	0.000000 *	125/125	BlockFrequency
	1	1	2	7	3	5	6	11	17	72	0.000000 *	125/125	CumulativeSums (m-2)
	1	1	5	2	4	5	6	21	11	69	0.000000 *	125/125	CumulativeSums (m-3)
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	Runs
	110	9	2	1	2	1	0	0	0	0	0.000000 *	42/125 *	LongestRun
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	ApproximateEntropy
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	Serial (forward)
	40	28	13	12	8	10	5	3	3	3	0.000000 *	114/125 *	Serial (backward)
6^{th}-order	2	3	4	7	13	19	13	26	18	20	0.000000 *	125/125	Frequency
	0	0	0	0	0	0	3	8	12	102	0.000000 *	125/125	BlockFrequency
	1	1	4	2	8	5	8	12	10	74	0.000000 *	125/125	CumulativeSums (m-2)
	1	3	1	1	2	10	6	20	14	67	0.000000 *	125/125	CumulativeSums (m-3)
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	Runs
	115	6	2	0	1	1	0	0	0	0	0.000000 *	44/125 *	LongestRun
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	ApproximateEntropy
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	Serial (forward)
	41	23	20	13	7	6	8	0	2	5	0.000000 *	113/125 *	Serial (backward)

Table 4.16: Cont'd with the results using $5^{th} - 6^{th}$ order models

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROP.	STATISTICAL TEST
0^{th}-order	30	21	12	10	8	9	5	7	10	13	0.000003 *	115/125 *	Frequency
	20	9	11	16	11	12	9	21	8	8	0.050764	120/125	BlockFrequency
	31	14	9	9	9	14	7	3	9	20	0.000000 *	119/125 *	CumulativeSums (m-2)
	32	15	11	6	10	14	5	3	11	18	0.000000 *	118/125 *	CumulativeSums (m-3)
	22	12	11	13	16	11	12	9	9	10	0.302788	120/125	Runs
	11	16	15	26	16	17	9	7	5	3	0.000062 *	124/125	LongestRun
	29	17	7	20	10	6	9	3	11	13	0.000001 *	119/125 *	ApproximateEntropy
	21	16	12	11	11	15	4	16	9	10	0.070160	116/125 *	Serial (forward)
	17	7	10	14	18	17	13	13	10	6	0.192277	123/125	Serial (backward)
1^{st}-order	21	17	13	15	11	7	13	7	9	12	0.130323	124/125	Frequency
	16	7	9	21	12	7	15	18	10	10	0.056599	122/125	BlockFrequency
	23	10	12	13	12	9	9	7	13	17	0.082208	124/125	CumulativeSums (m-2)
	24	9	12	12	9	15	16	6	12	10	0.034444	123/125	CumulativeSums (m-3)
	22	11	10	11	8	18	13	7	11	14	0.096097	122/125	Runs
	7	13	16	27	17	9	13	11	4	8	0.000274	124/125	LongestRun
	21	16	12	14	9	10	5	10	13	15	0.130323	122/125	ApproximateEntropy
	14	12	16	11	12	15	12	10	12	11	0.956806	122/125	Serial (forward)
	10	12	13	15	12	17	11	13	15	7	0.620686	123/125	Serial (backward)

Table 4.17: NIST test results with respect to the random sequence generated by the **decoupled Neighbor Coding** using $0^{th} - 1^{st}$ order models, where $M = 32$ for Block Frequency Test, $m = 2$ for Approximate Entropy Test and $m = 5$ for Serial Test and '*' marks a failure.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROP.	STATISTICAL TEST
2^{nd}-order	20	16	15	17	5	14	10	10	9	9	0.086622	121/125	Frequency
	17	6	15	12	8	14	16	12	13	12	0.262219	123/125	BlockFrequency
	21	16	11	13	12	7	12	5	11	17	0.073984	123/125	CumulativeSums (m-2)
	20	10	13	10	15	13	15	6	11	12	0.389809	120/125	CumulativeSums (m-3)
	20	14	11	13	13	8	7	13	14	12	0.493319	124/125	Runs
	7	11	20	23	22	10	9	12	6	5	0.000115	124/125	LongestRun
	20	15	15	12	11	5	10	13	11	13	0.316158	122/125	ApproximateEntropy
	15	15	11	8	16	11	15	14	10	10	0.643139	121/125	Serial (forward)
	16	7	7	11	15	12	13	17	14	13	0.262219	123/125	Serial (backward)
3^{rd}-order	18	17	12	10	17	8	14	11	9	9	0.389809	123/125	Frequency
	13	9	13	10	9	12	18	15	18	8	0.457002	122/125	BlockFrequency
	17	15	14	9	13	8	14	9	9	17	0.551044	124/125	CumulativeSums (m-2)
	16	11	12	13	8	16	19	5	15	10	0.192277	123/125	CumulativeSums (m-3)
	18	15	13	15	8	9	11	12	13	11	0.529142	124/125	Runs
	5	13	20	29	16	16	8	8	2	8	0.000000 *	123/125	LongestRun
	17	13	13	11	12	11	14	9	14	11	0.889414	125/125	ApproximateEntropy
	15	11	8	13	16	5	16	14	13	14	0.493319	123/125	Serial (forward)
	16	10	8	10	10	10	19	16	15	11	0.439517	125/125	Serial (backward)
4^{th}-order	18	17	15	17	10	10	10	13	5	10	0.192277	124/125	Frequency
	15	8	14	13	12	12	11	11	17	12	0.807956	124/125	BlockFrequency
	20	17	12	8	11	14	5	13	8	17	0.070160	124/125	CumulativeSums (m-2)
	16	18	7	16	15	9	13	5	10	16	0.117876	124/125	CumulativeSums (m-3)
	18	14	12	9	16	13	9	13	8	13	0.414457	125/125	Runs
	7	8	19	30	21	7	13	5	5	10	0.000000 *	123/125	LongestRun
	15	17	17	7	11	5	17	11	10	15	0.143910	122/125	ApproximateEntropy
	12	12	15	8	11	16	11	14	19	7	0.457002	123/125	Serial (forward)
	13	6	12	13	13	15	13	14	13	13	0.825875	124/125	Serial (backward)

Table 4.18: Cont'd with the results using $2^{nd} - 4^{th}$ order models

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROP.	STATISTICAL TEST
5^{th}-order	20	17	14	8	15	11	9	6	9	16	0.096097	125/125	Frequency
	10	9	12	12	13	13	16	15	17	8	0.552185	124/125	BlockFrequency
	21	13	10	14	10	9	13	3	15	17	0.043046	124/125	CumulativeSums (m-2)
	16	18	8	12	12	13	7	5	21	13	0.036430	125/125	CumulativeSums (m-3)
	16	10	13	10	7	13	12	19	7	18	0.192277	123/125	Runs
	4	13	17	30	20	11	10	5	3	12	0.000000 *	124/125	LongestRun
	15	14	14	12	9	10	18	12	10	11	0.665311	122/125	ApproximateEntropy
	13	15	13	12	14	13	19	7	12	7	0.512137	124/125	Serial (forward)
	11	10	20	10	13	16	12	10	15	8	0.474938	125/125	Serial (backward)
6^{th}-order	21	12	10	13	10	12	17	7	9	14	0.231636	122/125	Frequency
	16	10	11	9	9	11	17	17	13	12	0.437182	123/125	BlockFrequency
	21	12	11	11	7	15	11	6	13	18	0.091249	123/125	CumulativeSums (m-2)
	18	12	7	15	12	13	9	10	9	20	0.211194	123/125	CumulativeSums (m-3)
	14	10	15	15	15	15	11	6	13	11	0.529142	122/125	Runs
	7	7	22	25	20	10	7	13	6	8	0.000017 *	122/125	LongestRun
	17	12	11	13	11	14	17	8	12	10	0.598008	124/125	ApproximateEntropy
	19	14	6	11	9	14	16	16	9	11	0.277369	123/125	Serial (forward)
	15	10	10	13	10	13	11	17	14	12	0.843024	121/125	Serial (backward)

Table 4.19: Cont'd with the results using $5^{th} - 6^{th}$ order models

are deemed 'good' but it is not clear to us why the failure rate increases until 3^{rd} order and drops suddenly. The performance of 1-out-of-8 Coding is generally worse when applied column-wise for both forms of analysis; failures persist throughout the models we consider in this work.

4.3.2.3 Neighbor Coding

In the case of Chain-like Neighbor Coding, 15 bits are generated per row by pairing up row neighbors and a total 480-bit random sequence results per device. As shown in the fourth column of Table 4.2, none of our distillers can make meaningful improvement. The phenomenon aligns with our expectation for the failures are caused by the intrinsic chain dependencies of the pairing strategy rather than spatial correlation. The argument also draws support from the pass rate of Decoupled Neighbor Coding enhanced by a distiller in 1^{st} order or beyond. Overfitting is mild compared with other coding strategies. Like 1-out-of-8 Coding, pairing column-wise yields worse pass rate regardless chained or not; results are omitted for brevity. The high failure rate is attributed to the mutual-dependent comparison chain formed in every row. Take an 'ideal' 2-bit random sequence for example: the 4 outcomes '00', '01', '10' and '11' would occur all with probability exactly 1/4. However, given 3 ROs RO_A, RO_B and RO_C and assuming that the 6 outcomes $RO_A < RO_B < RO_C$, $RO_A < RO_C < RO_B$, $RO_B < RO_A < RO_C$,..., $RO_C < RO_B < RO_A$ are equally likely with probability exactly 1/6, then the probability of the 2-bit random sequence '00', '01', '10' and '11' generated by Neighbor Coding would be 1/6, 1/3, 1/3 and

1/6, a clear deviation from the 'ideal'. To validate our thought, we modify the scheme and pair up any RO no more than once; that is, we compare $RO_{i,j}$ and $RO_{i+1,j}$ where $1 \leq i \leq c$ and i is odd. After such a decoupling, the random sequence becomes 256-bit long. It turned out that a significant increase in pass rate was observed in the fifth column of Table 4.2. Block sizes selected for Block Frequency Test, Approximate Entropy Test and for Serial Test are 32, 2, and 5.

4.4 Summary

The systematic component of fabrication variation has posed a security threat to RO PUFs such that none of the current coding schemes can pass all NIST randomness tests without further secrecy amplification. To tackle the problem, we proposed a family of regression-based entropy distillers to separate random variation from systematic variation. The effectiveness of our distiller is affirmed by all passes in the new test results. In the next chapter, we will incorporate the entropy distiller into the complete design of the group-based RO PUF; meanwhile, we will consider ECC and devise a more error-resilient syndrome coding scheme to reduce its complexity.

Chapter 5

Kendall Syndrome Coding (KSC)

In Chapter 3, we designed the group-based algorithm LISA that searches for longest increasing subsequences to maximize the entropy extraction power of RO PUFs. Then in Chapter 4, we discussed the security threats caused by the underlying spatial correlation and tackled them with the regression-based entropy distiller that removed the systematic component so as to extract purely the i.i.d. Gaussian variations. In this chapter, we will incorporate the error correcting code (ECC) into the design and introduce Kendall Syndrome Coding (KSC) to further enhance the hardware efficiency. Recall that in Chapter 3 we encode a permutation of n elements into a binary integer $0 \dots n! - 1$ in a compact fashion, which here we refer to as Compact Syndrome Coding (CSC). The problem of CSC is that its error weight distribution does not correlate well with the underlying error probability distribution such that many errors occurring frequently are encoded in greater (instead of smaller) Hamming distance than those less likely to happen. In light of this, KSC preserves the underlying Kendall tau distance [26] between two rank permutations in the encoded/mapped Hamming space. Not only is the new design more error tolerant, it is also more robust in security and much easier to fabricate. Overall, the new group-based design is 9% more efficient than the previous and 50% or more efficient than an index-based (IBS) approach [56] in most error correcting scenar-

00000_2	$\{ABCD\}$	01000_2	$\{BCAD\}$	10000_2	$\{CDAB\}$
00001_2	$\{ABDC\}$	01001_2	$\{BCDA\}$	10001_2	$\{CDBA\}$
00010_2	$\{ACBD\}$	01010_2	$\{BDAC\}$	10010_2	$\{DABC\}$
00011_2	$\{ACDB\}$	01011_2	$\{BDCA\}$	10011_2	$\{DACB\}$
00100_2	$\{ADBC\}$	01100_2	$\{CABD\}$	10100_2	$\{DBAC\}$
00101_2	$\{ADCB\}$	01101_2	$\{CADB\}$	10101_2	$\{DBCA\}$
00110_2	$\{BACD\}$	01110_2	$\{CBAD\}$	10110_2	$\{DCAB\}$
00111_2	$\{BADC\}$	01111_2	$\{CBDA\}$	10111_2	$\{DCBA\}$

Table 5.1: The codebook of the rank permutations of 4 ROs using Compact Syndrome Coding (CSC), where $\{ABCD\}$ is a shorthand of the frequency relation $RO_A < RO_B < RO_C < RO_D$

ios we constructed with BCH. The i.i.d. uniform output assumption is validated by the NIST random test [5] results, which indicate that our output statistics is indistinguishable from the uniform distribution.

5.1 Problems of the Previous Design

LISA heuristically extract the maximal comparison-based entropy $\log_2 M!$ out of M ROs by searching for longest increasing subsequences (LIS) iteratively to form independent groups that contain the largest set of ROs while maintaining the stability threshold between any two ROs within the group. Still, there are four issues that may reduce its usability.

- **Manufacturing Complexity** In the enrollment phase, LISA requires sam-

pling of RO frequencies at two temperature points, e.g., 0°C and 100°C, to ensure the reliability. This complicates and slows down the fabrication process.

- **Error Tolerance** While LISA is effective to address temperature variability that has a linear impact on RO frequencies, it is not as efficient to deal with non-linear variability such as supply voltage fluctuation. Because it relies on the reliability threshold to accommodate all potential errors, it has to maintain a relatively high margin in choosing the value of the threshold. Being conservative in the choice reduces the hardware efficiency and thus may not yield the best solution when one can also leverage on ECC for error control.

- **Spatial Correlation** As mentioned in the last chapter, when two ROs are distant from each other, the generated secret can be better predicted if one has the knowledge of the underlying systematic trend. We have shown statistically how the trend weakens the security of PUFs as the output bits are not independent and identically distributed (i.i.d.).

- **Coding Inefficiency** The previous work encodes the frequency rank permutation of M ROs in binary of integer $0 \ldots M! - 1$, here we refer to as Compact Syndrome Coding (CSC). CSC does not work well with error correcting codes (ECC). To illustrate, let us consider a list of four ROs RO_A, RO_B, RO_C and RO_D; the CSC codebook in lexicographic order is listed in Table 5.1. Suppose a rank permutation $\{RO_A < RO_D < RO_C < RO_B\}$ is first enrolled with its codeword 00101_2; later on, we would like to regenerate the codeword given certain errors. Let us say a flipover occurs between RO_A and RO_D, yielding

us a new frequency measurement $\{RO_D < RO_A < RO_C < RO_B\}$ or 10011_2 in its encoded form. Since the Hamming distance d_h between the two codewords 00101_2 and 10011_2 is three, the error can be corrected by BCH($n = 15$, $k = 5$, $t = 3$) code using the Code-Offset technique in [53], assuming there is no more error in the subsequent 10 bit of the 15-bit code block. To estimate the effective min-entropy, which is defined later, let us assume the 10 bits are derived from other two independent lists of 4 ROs. If each rank permutation is equally likely with probability $\frac{1}{4!}$, then the raw entropy of the block is $3 \log_2 4!$ or 13.754 bits. The min-entropy, however, is merely 3.7543 bits because we have to deduct 10 bits of entropy loss due to public syndrome disclosure[1]. In fact, the loss can be reduced by better correlating the error weight distribution in Hamming space with the probability mass function (p.m.f.) of erroneous flipovers. In other words, if an error event happens in higher probability, it should be encoded closer in Hamming space. To see how this may not be true in CSC, consider three flipovers happening at the same time: the first between RO_A and RO_B, the second between RO_A and RO_C and the third between RO_A and RO_D, with presumably fairly low probability. But instead, the Hamming distance between the new codeword 10111_2 of $\{DCBA\}$ and the enrolled codeword 00101_2 of $\{ADCB\}$ is only two, even closer than the previous error pattern presumably in higher probability.

To address the four issues, this chapter provides a new group-based design with improved entropy efficiency, security and fabrication simplicity: 1) we propose a new

[1]More generally, $(n - k)$-bit loss for linear codes [53].

syndrome coding based on Kendall tau distance for better stability in the encoded Hamming space; 2) we incorporate two new processes for entropy distillation and entropy packing to strengthen security; 3) we redesign the grouping algorithm to simplify the manufacturing process. The improvements make the new design 9% more efficient than its predecessor and 50% or more efficient than the index-based (IBS) approach [56] in most of our experimental scenarios. The NIST hypothesis testing affirms that our PUF output is indistinguishable from those drawn from an ideal uniform distribution. We emphasize that our results do *not* resort to the help from universal hash function (UHF) nor linear feedback shift register (LFSR) typically employed by PUFs for secrecy amplification. Without an external seed, our PUF can generate the secret in a fully autonomous fashion.

5.2 Kendall Syndrome Coding (KSC)

Kendall's correlation statistic [26] is used to address the issue of coding inefficiency. To begin with, rank permutation of n elements is defined as permutation of integers (ranks) $1 \dots n$; Kendall tau distance $d_\tau(\sigma, \pi)$ is defined as the minimum number of transpositions of adjacent ranks required to change from one rank permutation σ into another π of the same size [11]. For instance, consider two rank permutations of three elements $\sigma = \{\sigma(1) = 1, \sigma(2) = 3, \sigma(3) = 2\}$ and $\pi = \{\pi(1) = 2, \pi(2) = 3, \pi(3) = 1\}$. The Kendall distance $d_\tau(\sigma, \pi)$ is one because σ would be equal to π after transposing of rank 1 and rank 2 of the first element

and the third element[2]. d_τ can also defined as the number of pairwise disagreements between two rank permutations [15], namely,

$$d_\tau(\sigma, \pi) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} s(i, j) \tag{5.1}$$

where

$$s(i,j) = \begin{cases} 1 & \text{if } (\sigma(i) < \sigma(j) \cap \pi(i) > \pi(j)) \\ & \cup\ (\sigma(i) > \sigma(j) \cap \pi(i) < \pi(j)) \\ 0 & \text{otherwise.} \end{cases}$$

For a group-based RO PUF, d_τ is equal to the number of flipovers in a group during secret regeneration and presumably reversely proportional to the error probability. Now we want to encode the frequency rank permutation of a list of ROs from the Kendall space into the Hamming space such that the Kendall distance d_τ between any two rank permutations is equivalent to the Hamming distance d_h of the corresponding codewords. Indeed, such a class of codes exists and the conversion is efficient [16]. Given a group of ROs g, $\{RO_1 \dots RO_n\}$ in certain physical order, its rank permutation can be encoded pairwisely into a bitstring s_g of length $\frac{n(n-1)}{2}$,

$$s_g = (s_g(1,2) \dots s_g(1,n), s_g(2,3) \dots s_g(n-1,n)) \tag{5.2}$$

where $\forall s_g(i,j), i < j, 1 \leq i, j \leq n$,

$$s_g(i,j) = \begin{cases} 0 & \text{if } RO_i < RO_j \\ 1 & \text{otherwise.} \end{cases}$$

[2]In the previous section, the elements are listed in the order of ranks, different from here the ranks are listed in the order of elements.

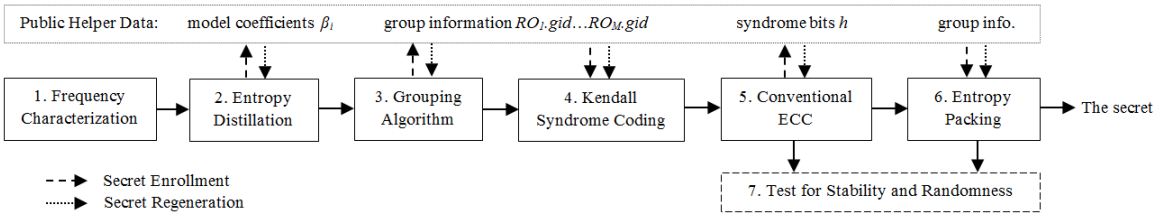

Figure 5.1: The architecture of the proposed group-based RO PUF

For instance, consider a group g composed of 3 ROs with frequency readings $\{RO_A = 38, RO_B = 97, RO_C = 54\}$ in the same rank permutation as the $\sigma = \{1, 3, 2\}$ defined earlier; from Eqn. (5.2), we can KSC encode g into $s_g = (0, 0, 1)$. Let us say later certain environmental variations cause a re-measurement of the frequencies to $g' = \{RO_A = 78, RO_B = 103, RO_C = 60\}$, which has the same rank permutation as the $\pi = \{2, 3, 1\}$ defined above. In turn, Eqn. (5.2) yields a KSC encoded bitstring $s_{g'} = (0, 1, 1)$. As we see, $d_h(s_g, s_{g'}) = d_\tau(\sigma, \pi) = 1$, where the pairwise disorder takes place between elements (1, 3) for $g(1) < g(3)$ but $g'(1) > g'(3)$. Reversely, [16] also shows how to decode s_g into $\sigma = (\sigma(1) \ldots \sigma(n))$, where

$$\sigma(i) = 1 + \sum_{j=1}^{i-1}(1 - s(i, j)) + \sum_{j=i+1}^{n} s(i, j). \tag{5.3}$$

5.3 The Proposed Group-Based RO PUF

The architecture of the proposed PUF is depicted in Figure 5.1. Our public helper data comprises three parts (i) distiller coefficients, (ii) group information and (iii) syndrome bits; all of them are determined and stored publicly in non-volatile memory such as EEPROM or NAND/NOR flash when the secret is first enrolled.

The seven-step secrecy extraction procedure is explained in the following.

5.3.1 Frequency Characterization

A RO PUF typically consists of a RO array as well as counters and multiplexers to help acquire frequency readings of the RO array. Given M ROs, the frequency readings are denoted as $RO_1 \dots RO_M$ in certain physical order, say, scanning sequentially by rows when ROs are placed as a 2-D array. One may take an average over multiple measurements for each RO as the output of the step.

5.3.2 Randomness Distillation

The output from Step 1 contains both random and systematic variations. Since the systematic component may render the output lack of randomness, we apply polynomial regression to remove it [54]. Each PUF calculates its own model coefficients and stores them as public helper data. The optimal order of the polynomial regression model can be determined empirically or by AICc [48]. As an example, let us model the systematic trend of a m-by-n 2-D RO array by means of 1^{st}-order polynomial $sys_{x,y} = \beta_1 x + \beta_2 y + \beta_3$, where β_i's are the model coefficients to be solved and $(1,1) \leq (x,y) \leq (n,m)$. The frequency measurement can then be written as $RO_{x,y} = sys_{x,y} + \epsilon_{x,y}$, where $sys_{x,y}$ denotes the systematic variability and $\epsilon_{x,y}$ denotes the random variation at location (x,y). Putting $m \times n$ model equations in matrix form,

$$\boldsymbol{z} = \boldsymbol{\Omega\beta} + \boldsymbol{\epsilon}, \tag{5.4}$$

where

$$\boldsymbol{z} = \begin{pmatrix} RO_{1,1} \\ \vdots \\ RO_{n,1} \\ \vdots \\ RO_{1,m} \\ \vdots \\ RO_{n,m} \end{pmatrix}, \boldsymbol{\Omega\beta} = \begin{pmatrix} 1 & 1 & 1 \\ \vdots & \vdots & \vdots \\ n & 1 & 1 \\ \vdots & \vdots & \vdots \\ 1 & m & 1 \\ \vdots & \vdots & \vdots \\ n & m & 1 \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{pmatrix}, \boldsymbol{\epsilon} = \begin{pmatrix} \epsilon_{1,1} \\ \vdots \\ \epsilon_{n,1} \\ \vdots \\ \epsilon_{1,m} \\ \vdots \\ \epsilon_{n,m} \end{pmatrix}.$$

To solve the model coefficients $\boldsymbol{\beta}$, one can use the least squares method that takes the first derivative on the sum of the squares of the residual terms $\boldsymbol{\epsilon}$ and set the derivative to zero, or formally,

$$\frac{\partial \boldsymbol{\epsilon}^2}{\partial \boldsymbol{\beta}} = \frac{\partial (\boldsymbol{z} - \boldsymbol{\Omega\beta})^{\boldsymbol{T}} (\boldsymbol{z} - \boldsymbol{\Omega\beta})}{\partial \boldsymbol{\beta}} \tag{5.5}$$

$$= \frac{\partial (\boldsymbol{z^T z} - \boldsymbol{z^T \Omega\beta} - \boldsymbol{\beta^T \Omega^T z} + \boldsymbol{\beta^T \Omega^T \Omega\beta})}{\partial \boldsymbol{\beta}} \tag{5.6}$$

$$= -2\boldsymbol{\Omega^T z} + 2\boldsymbol{\Omega^T \Omega\beta} = \boldsymbol{0} \tag{5.7}$$

$$\Longrightarrow \boldsymbol{\beta} = (\boldsymbol{\Omega^T \Omega})^{-1} \boldsymbol{\Omega^T z}, \tag{5.8}$$

where $(\boldsymbol{\Omega^T \Omega})^{-1} \boldsymbol{\Omega^T}$ is a constant solver and $\boldsymbol{z}$ is the frequency characterization of the $m \times n$ ROs at enrollment. Once $\boldsymbol{\beta}$ is determined, we can easily calculate the output of the step, that is, the distilled random variation $\boldsymbol{\epsilon'}$, which is equal to $\boldsymbol{z'} - \boldsymbol{\Omega\beta}$ for any frequency re-characterization $\boldsymbol{z'}$.

5.3.3 Grouping Algorithm

The goal of the step is to form independent groups while meeting certain stability criterion. The key difference from its predecessor [55] is that only *one* environmental condition is needed to take the measurements in the enrollment phase. This resolves the manufacturing problem mentioned in the very beginning. Due to the change, the maximization problem in [55] is rephrased as: given M ROs whose frequency output are i.i.d., we want to find a partition $G = \{g_1 \dots g_{|G|}\}$ that maximizes the total group entropy $\sum_{i=1}^{|G|} \log_2 |g_i|!$ while ensuring that no RO pair in the same group g_i have their frequency difference less than a stability threshold f_{th} at enrollment time, that is,

maximize $\sum_{i=1}^{|G|} \log_2 |g_i|!$ ***subject to***

1. $g_i \cap g_j = \emptyset$, where $1 \leq i, j \leq |G|, i \neq j$
2. $g_1 \cup g_2 \cup \ldots \cup g_{|G|} = RO_1, \ldots RO_M$
3. $\forall RO_i, RO_j \in g_k$, $|RO_i - RO_j| \geq f_{th}$, where $1 \leq i, j \leq |g_k|, i \neq j, 1 \leq k \leq |G|$.

Constraint a) ensures that no RO is used twice to maintain the i.i.d. PUF output assumption; b) leaves no RO unexplored[3]; c) parametrizes PUF stability with f_{th}, whose value can be determined empirically. To solve the new maximization problem, LISA can be simplified as LISA-lite as below. LISA-lite completes in $O(M^2)$ and the worst case occurs, for instance, when all ROs have the same frequency readings such that only one RO can be removed from the linked list for each run of the for

[3]The constraint is redundant in the setting of a maximization problem but kept for clarity

loop. Lastly, each RO is assigned with a group ID in solution G and all $RO_i.gid$'s are stored in their physical order for future reference.

5.3.4 Kendall Syndrome Coding

Each group g_i in solution G is KSC encoded by Eqn. (5.2). The resulting bitstrings $s_{g_1} \ldots s_{g_{|G|}}$ are concatenated as the output of the step.

5.3.5 Conventional ECC

A linear block code (n, k, t) can be used to correct runtime errors, where n denotes block size, k the number of information bits, $n-k$ the number of parity bits and t maximum errors within the block that are correctable by the code. The Code-Offset technique [53] is assumed to bound the min-entropy loss due to public disclosure of syndrome bits. At enrollment, the output from the previous step is divided into blocks of n-bit secret w. The first k bits of w is encoded with certain ECC to produce $n-k$ parity bits p. The parity bits p then exclusive-or with the last $n-k$ bits of w to produce the $n-k$ syndrome bits h, which are then saved as public helper data to assist secrecy recovery. To recover the enrolled secret block w given new input w' from Step 4, we retrieve the saved syndrome bits h and exclusive-or with the last $n-k$ bits of w' to produce $n-k$ parity bits p'. The first k-bit of w' is then appended with p' to form a n-bit block to decode. As long as $d_h(w, w') \leq t$, ECC decoder can correct all errors in w' and output the w enrolled at first. All restored secret blocks w's form the output of the step and input of Step 6–7.

Procedure 4 LISA-lite

Input: (i) M ROs, $RO_1 \dots RO_M$, in their physical order with $RO_i.phy$ denoting i^{th} RO's physical position, $RO_i.frq$ its frequency reading at enrollment, and $RO_i.gid$ its group ID (with initial value -1); (ii) reliability threshold f_{th}

Output: (i) $RO_1 \dots RO_M$ with all $RO_i.gid \neq -1$

1: sort $RO_1 \dots RO_M$ in increasing order of $RO_i.frq$'s and keep the sorted objects $RO'_1 \dots RO'_M$ on a linked list L

2: $gid \leftarrow 1$

3: $frq_{pre} \leftarrow -\infty$ // previous frequency

4: **while** $frq_{pre} \neq -\infty$ **do**

5: $\quad frq_{pre} \leftarrow -\infty$

6: $\quad$ **for** $i \leftarrow 1$ to $|L|$ **do**

7: $\quad\quad$ **if** $(RO'_i.frq - frq_{pre}) \geq f_{th}$ **then**

8: $\quad\quad\quad RO'_i.gid \leftarrow gid$

9: $\quad\quad\quad frq_{pre} \leftarrow RO'_i.frq$

10: $\quad\quad\quad$ remove RO'_i from L

11: $\quad\quad\quad i = i - 1$

12: $\quad\quad$ **end if**

13: $\quad$ **end for**

14: $\quad gid = gid + 1$

15: **end while**

16: **return** $G = \{RO_1 \dots RO_M\}$

5.3.6 Entropy Packing

Although KSC is designed to help reduce the complexity of ECC, it does not encode entropy efficiently. Many bitstrings are left unused; taking three ROs for example, (0, 1, 0) would never occur due to the contradiction ($RO_A < RO_B, RO_A \geq RO_C, RO_B < RO_C$). Therefore, we have to re-encode each group g_i compactly with CSC with help from the stored group information $RO_i.gid$'s. Since error correction has been done, the coding inefficiency issue mentioned in the very beginning is no longer a concern. Before encoding in CSC, we have to KSC decode each group s_{g_i} into its rank permutation via Eqn. 5.3. To encode a group g in the most compact form c_g, we can calculate the inversion vector (line 2–8,10) and interpret it in factorial number system (line 9) as below [27]. Note that the algorithm we are showing does not produce a codebook in lexicographic order like Table 5.1. CSC decoding can also be done efficiently but is beyond the scope of the work. Lastly, in order to construct a secret in uniform distribution, we have to close the unused gap between $|g|!$ and $2^{\lceil \log_2 |g|! \rceil} - 1$ for each output c_g as we put together all c_{g_i}'s to form the final PUF secret. This can be done by simple addition, subtraction and shift operations; the pseudo code is omitted for brevity.

5.3.7 Test for Randomness and Stability

This verification process not only helps us ensure security and reliability of the final secret but also helps us choose the parameter f_{th} and the exact regression model. This leads our discussion to the next section.

Procedure 5 Compact Syndrome Coding (CSC) Encoding

Input: a group g containing ordered ROs $RO_1 \dots RO_{|g|}$

Output: a CSC encoded integer c_g in $\lceil \log_2 |g|! \rceil$ bits

1: $c_g = 0$
2: **for** $i \leftarrow |g|$ to $|2|$ **do**
3: $inv = 0$ // number of inversions
4: **for** $j \leftarrow 1$ to $i - 1$ **do**
5: **if** $RO_i < RO_j$ **then**
6: $inv = inv + 1$
7: **end if**
8: **end for**
9: $c_g = (c_g + inv) \times (i - 1)$
10: **end for**
11: **return** c_g

5.4 Experimental Results

This section provides the details of the last step in the previous section. In particular, we will describe how to determine the polynomial model empirically and how statistical results affirm the i.i.d. uniform assumption we have made on the secret. In addition, we will search for the optimal value of the stability threshold f_{th} and calculate the amount of entropy our PUF can generate in several error correcting scenarios. Likewise, the estimate for the IBS-based scheme will be derived for comparison. Two datasets are used in the experiment and results will be presented separately. The prototype of the proposed PUF has been realized as an embedded system on Xilinx Virtex-5 FPGAs using Xilinx ISE and Xilinx Platform Studio (XPS) EDK/SDK 13.2 and 9.2 tool chains[4]. The PUF is implemented as an IP core connecting to the MicroBlaze™ soft processor core through Processor Local Bus (PLB). The bus exchanges control signals and RO frequency characterization between the processor and the IP via user defined soft registers. The frequency characterization is in turn passed through to a RS232 serial port logger on PC for the following analyses. Timing critical logic such as RO is instantiated as a hard macro in ISE. Polynomial regression employed in Step 2 can be solved quickly as the Floating Point Unit (FPU) of MicroBlaze™ is enabled. In addition to our own dataset, the public dataset [31] is also used to confirm the results.

[4] v13.2 for ML506 and ML510 development boards and v9.2 for ML501 boards.

5.4.1 Test for Randomness

The first goal is to test whether our PUF is secure. To this end, the output of Step 6 is subject to the NIST statistical test for randomness [5]. An 'ideal' random sequence is regarded as the outcome of consecutive flips of a fair coin. In other words, the random variable assigned to each toss is identical and independent distributed (i.i.d.) and uniformly distributed between 0 and 1 with equal probability 1/2. The null hypothesis of the test is that the random sequence under test is 'ideal' unless the test statistic indicates a clear deviation. NIST test results are interpreted in two ways: a) the proportion of total bitstrings that passes a test shall be above a minimum value; b) the *P-values* of all bitstrings shall be uniformly distributed such that the *P-value* of the *P-values* is equal or greater than a minimum value; default settings were used in the test suite. Eleven out of fifteen tests in the suite are applicable to our output length; for each of the two datasets, we use the first half to select the order of the polynomial model and the second half to validate the randomness of the output.

5.4.1.1 Virginia Tech Dataset

This dataset comprises frequency characterization of 125 Xilinx Spartan-3 (90-nm) FPGAs [31]. For each FPGA, 512 ROs are implemented in 32 rows by 16 columns. Although there are 100 frequency measurements available for each RO in one operating condition, only the first measurement at 1.2V 25°C is used as the output of Step 1. The outputs of Step 6 from chip No.1 to No.125 are concatenated

altogether to construct a long random sequence subject to test. Test parameters are set as follows: bitstring length 400 (except 3× for FFT Test in order to meet with the minimum length requirement), block length for Frequency Test 32, 2 for Approximate Entropy Test and 5 for Serial Test, all following NIST recommendations. The test results from the first half of the dataset suggest that we can select 1^{st}-order polynomial to remove the systematic component with least computation. Indeed, the results from the second half affirm the selection and more importantly the i.i.d. uniform assumption we have made on our PUF output; see Table 5.3 and Table 5.9 at the end of the chapter for detailed reports.

5.4.1.2 In-House Dataset

The second dataset is collected from 9 Xilinx Virtex-5 (65-nm) FPGAs in our own lab. The 9 FPGAs come with three different types of development boards: 3 ML501, 3 ML506 and 3 ML510. Because of different sizes, they are placed with different number of RO arrays: 3 for ML501, 6 for ML506 and 12 for ML510, 63 in total. Each array contains 32 × 16 ROs just as in the previous dataset and is regarded as an independent PUF for the test. Although multiple measurements are available for each array, only the first measurement at 20°C is used as the output of Step 1 to produce the final secret. All test parameters are the same as mentioned previously. Similar results are derived from this dataset: The first half suggests the choice of the first order model since it passes all tests with the lowest computing and storage cost. The i.i.d. uniform assumption along with the selected model is

affirmed by the second half. Respective results are listed in Table 5.6 and Table 5.10 at the end of the chapter.

5.4.2 Test for Stability

The second goal is to determine the optimal value of f_{th} given a class of ECC in Step 5; moreover, in the criterion that all errors have to be corrected, we want to estimate the effective min-entropy we can extract given a fixed number of ROs. Three classes of BCH codes are considered, namely, BCH($n = 31$, k, t), BCH($n = 63$, k, t) and BCH($n = 127$, k, t), where n denotes code block size, k the number of information bits per block and t the maximum correctable errors within a block. As we know from [53], the larger the k, the smaller the min-entropy loss $n - k$ per block; both depend on t. For each class of code, we first pick a f_{th} and find out the largest t among *all* ECC blocks under test. As long as t is small enough such that all errors can be corrected by the given class of code, we have at least one solution k for the picked f_{th}, among which the largest k is used to calculate the effective min-entropy of PUF schemes. For the proposed PUF, the effective min-entropy H_{∞}^{KSC} is estimated as

$$\sum_{i=1}^{|G|} \log_2 |g_i|! - \lceil \frac{\sum_{i=1}^{|G|} |s_{g_i}|}{n} \rceil (n-k) \text{ if } \geq 0, \text{ else } 0, \tag{5.9}$$

where the first term represents the total entropy extracted from the grouping algorithm in Step 3 and the second term discounts the total min-entropy loss due to public disclosure of syndrome bits for ECC in Step 5. To compare, when CSC

rather than KSC is used in Step 4 (as proposed in [55]), replacing the s_{g_i}'s in Eqn. (5.9) with the c_{g_i}'s defined in Step 6 forms the estimate of H_∞^{CSC}. In case there is no solution k for a code class to correct all errors with a given f_{th}, the effective min-entropy is then set to 0 as the case when the result of Eqn. (5.9) turns negative. For both datasets, f_{th} is swept from 0.1 to 3 standard deviation of the 512 random variations distilled in Step 2.

5.4.2.1 Virginia Tech Dataset

This dataset consists of frequency characterization of 5 Spartan-3 FPGAs provisioned with ±10% and ±20% core supply voltage fluctuation and separately temperature variation from 25°C to 65°C [31]. We rule out the cases with ±20% voltage swings for they would drive the results too conservative. Rather, our estimate is based on the criterion that *all* errors must be corrected in rest of the 7 cases: Case 1) the nominal condition at 1.2V 25°C; Case 2–3) provisioned in 25°C with core supply voltage at 1.08V and 1.32V; Case 4–7) provisioned at 1.2V 35°C, 45°C, 55°C and 65°C. The secret is enrolled in Case 1 and regenerated in Case 2–7). Also, we scan through 0^{th} to 6^{th} order model in Step 2 and use the average to report.

Figure 5.2 5.3 and 5.4 draws two key results when BCH(n=31, k, t), BCH(n=63, k, t) and BCH(n=127, k, t) is considered respectively: (i) the maximum number of errors among all ECC blocks, see vertical bars that correspond to the secondary y-axis; (ii) the averaged H_∞^{CSC} and H_∞^{KSC} given (i), see lines and the primary y-axis. As we see, both H_∞^{CSC} and H_∞^{KSC} drop to zero when f_{th} approaches zero, meaning

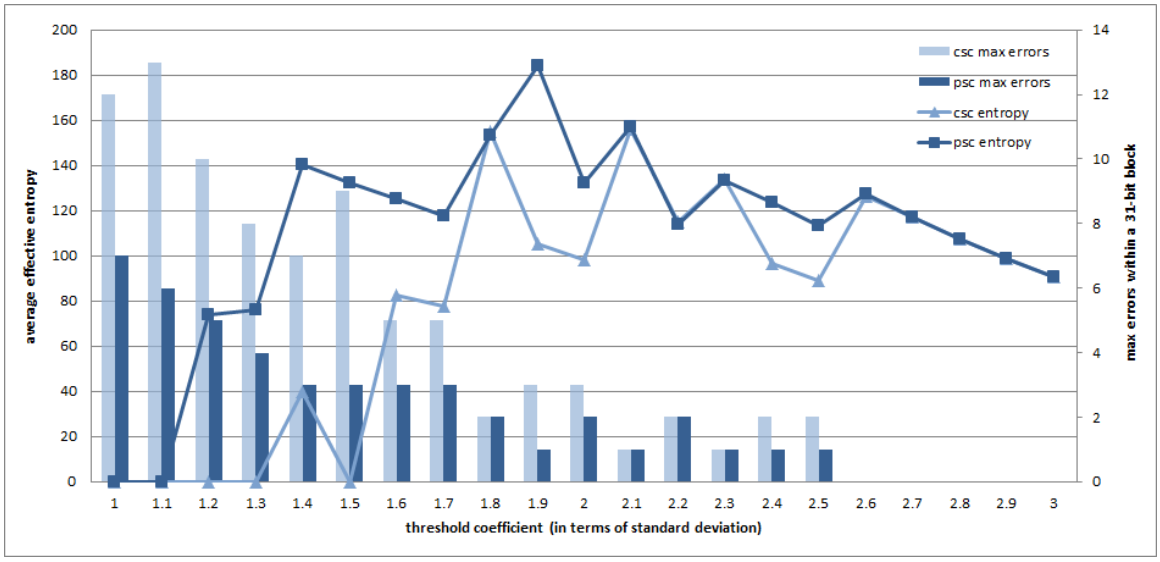

Figure 5.2: The estimated H_∞^{CSC} and H_∞^{KSC} derived with BCH(31, k, t) based on the Virginia Tech dataset

that errors are so overwhelming that either there is no solution to make all blocks error-free or the entropy loss surpasses the group entropy in Eqn. 5.9. On the other hand, as f_{th} approaches 3 there is no error to be corrected such that H_∞^{CSC} and H_∞^{KSC} converge. When f_{th} is in between, H_∞^{CSC} and H_∞^{KSC} increase if the reduction in min-entropy loss is greater than the reduction in group entropy extraction; otherwise, they decrease. The tipping point, 1.9 for H_∞^{KSC}, suggests the optimal value we can choose for f_{th}. On average H_∞^{KSC} outperforms H_∞^{CSC} by 8% in those optimal cases.

5.4.2.2 In-House Dataset

The second dataset is derived from the 9 FPGAs mentioned earlier. Frequency measurements are provisioned at chip temperature 20°C, 50°C and 100°C respec-

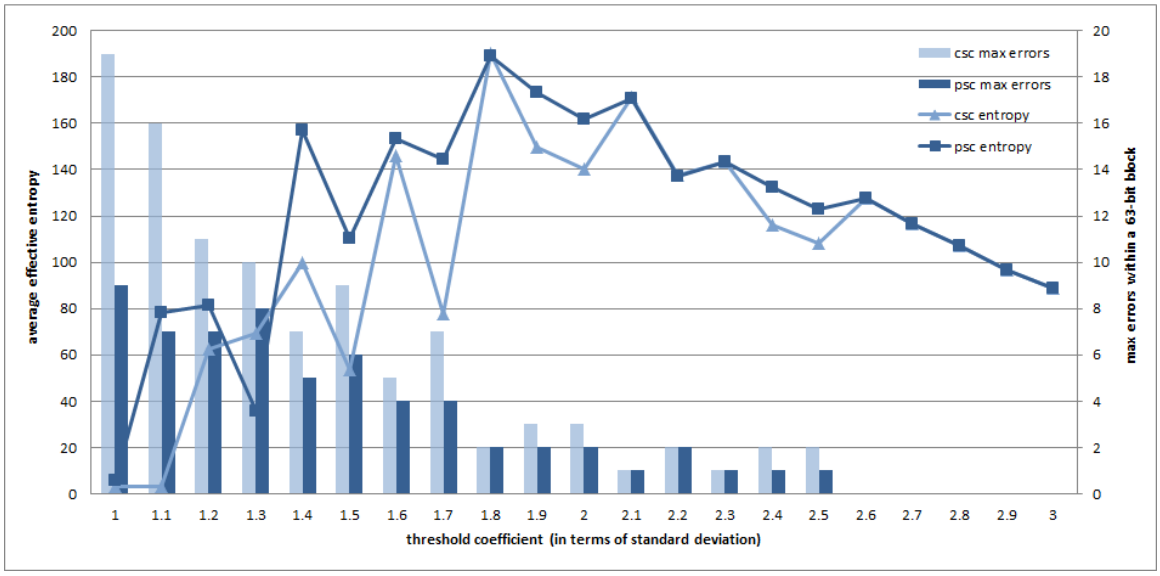

Figure 5.3: The estimated H_{∞}^{CSC} and H_{∞}^{KSC} derived with BCH(63, k, t) based on the Virginia Tech dataset

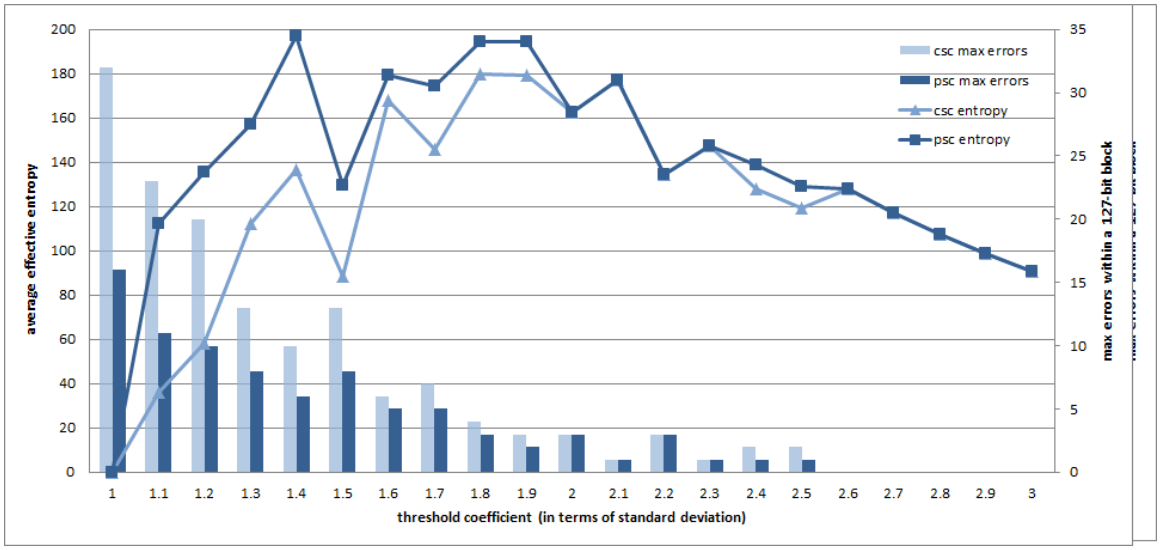

Figure 5.4: The estimated H_{∞}^{CSC} and H_{∞}^{KSC} derived with BCH(127, k, t) based on the Virginia Tech dataset

tively with no intended core supply voltage variation. Temperatures are manually controlled by monitoring the on-chip system sensor. Ten measurements are taken

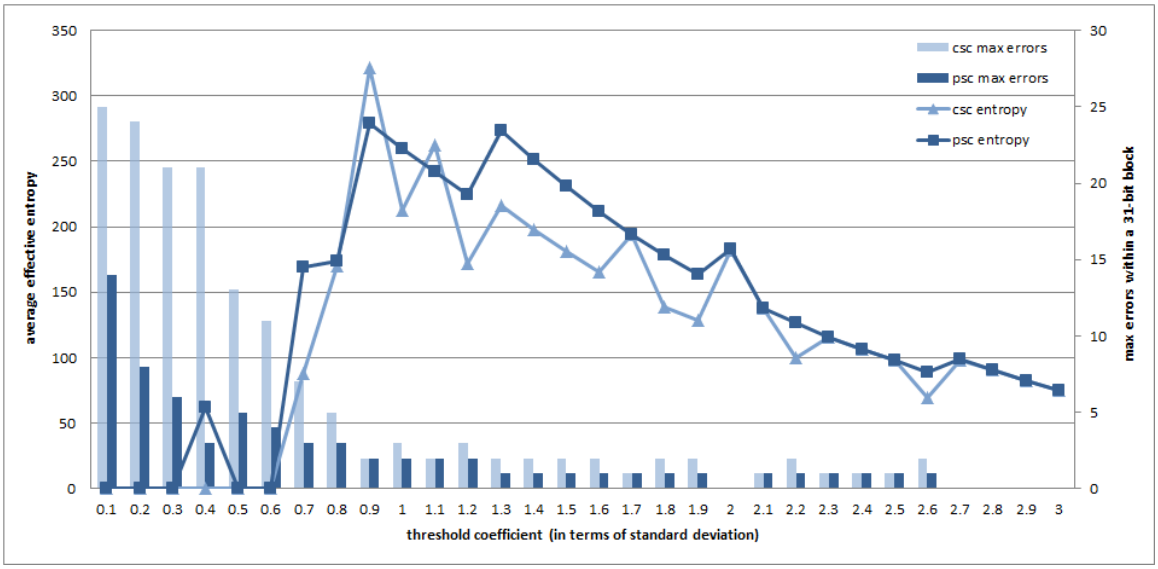

Figure 5.5: The estimated H_{∞}^{CSC} and H_{∞}^{KSC} derived with BCH(31, k, t) based on the in-house dataset

for each of the three scenarios; the first measurement at 20°C is used to generate the secret and then we try to recover it from the rest of 29 measurements with help from ECC. As in the previous dataset, polynomial orders 0^{th} to 6^{th} are all tested out. The test results are similar to those derived from the previous dataset only the tipping point shift lower to f_{th} around 0.4 to 0.9 standard deviation due to less errors are introduced with a stable voltage supply. Consequently, the estimates of H_{∞}^{CSC} and H_{∞}^{KSC} in those optimal cases almost double than in the previous dataset to 353 and 385 bits respectively, which translates to a 9% gain. Complete charts are reported in Figure 5.5, 5.6, 5.7.

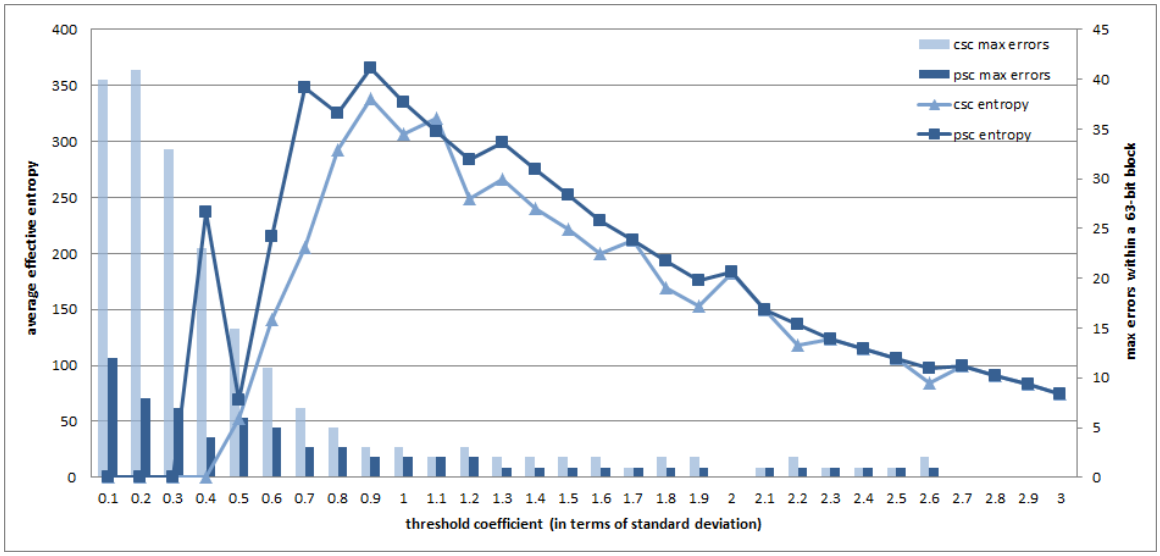

Figure 5.6: The estimated H_{∞}^{CSC} and H_{∞}^{KSC} derived with BCH(63, k, t) based on the in-house dataset

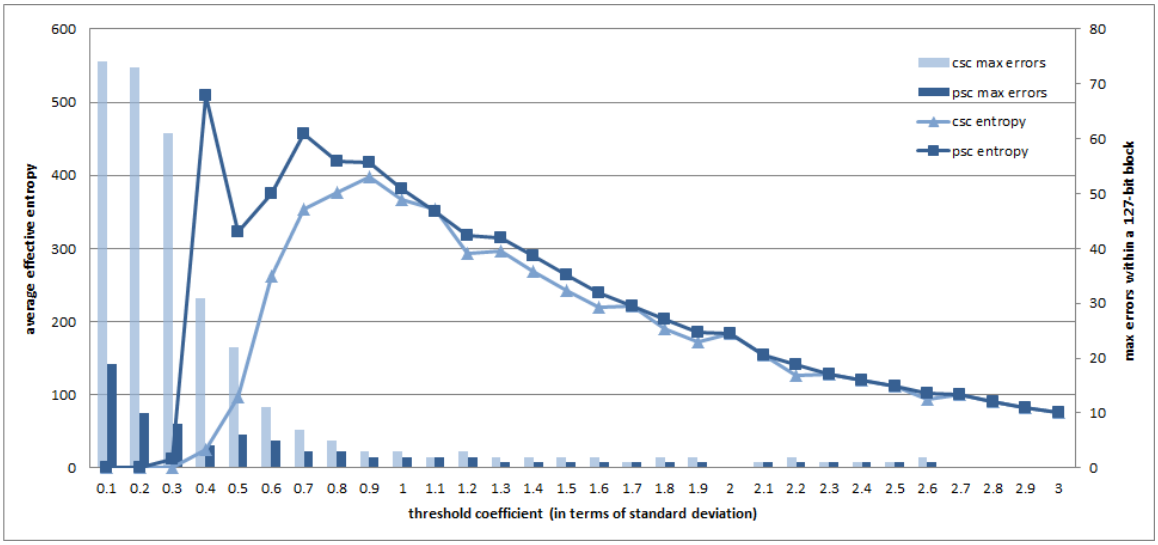

Figure 5.7: The estimated H_{∞}^{CSC} and H_{∞}^{KSC} derived with BCH(127, k, t) based on the in-house dataset

5.4.3 In Comparison with IBS-Based RO PUF

To compare with the IBS-based scheme [56] in the context of the i.i.d. uniform PUF output assumption, we form IBS blocks each out of k consecutive ROs[5] and

[5]Since there is no distillation process assumed, a block has to be formed out of ROs physically as close as possible to reduce systematic correlation [32].

	Virginia Tech			In-house		
	H_∞^{IBS}	H_∞^{KSC}	Gain	H_∞^{IBS}	H_∞^{KSC}	Gain
No ECC	85	128	50%	128	102	-20%
BCH(31)	96	184	92%	128	280	118%
BCH(63)	104	189	82%	134	366	173%
BCH(127)	121	197	63%	172	509	196%
Average	107	190	72%	140	314	123%

Table 5.2: The estimate of H_∞^{IBS} and H_∞^{KSC} in various ECC scenarios with Virginia Tech's dataset on the left and our in-house dataset on the right

ensure no blocks share a common RO. Each IBS block generates one bit secret through comparing the ROs that yield the largest frequency difference at enrollment time; the selected pair is recalled by keeping a $\log_2 \binom{k}{2}$-bit index as public helper data. The k we consider ranges from 2 to 6. Applying a similar methodology we estimate the effective min-entropy H_∞^{IBS} using the the same datasets and BCH codes. For the Virginia Tech dataset no error takes place when $k = 6$, whereas no error is observed when $k \geq 4$ for the in-house dataset. The H_∞^{IBS} vs. k relation is drawn in Figure 5.8 and 5.9. Table 5.2 summarizes the best case H_∞^{IBS} and H_∞^{KSC} when different error correcting capability is assumed. As we see, in most cases the proposed group-based RO PUF is 50% or more efficient than the conjectured IBS-based RO PUF in terms of effective min-entropy given the same number of ROs. The only case KSC underperforms is due to one single error that cannot be cured until f_{th} reaches 2.7; the study of the root cause and preventing mechanisms is among our future work. Note that when $k = 2$ it is a disjointed neighbor pairing scheme with no index needed [54].

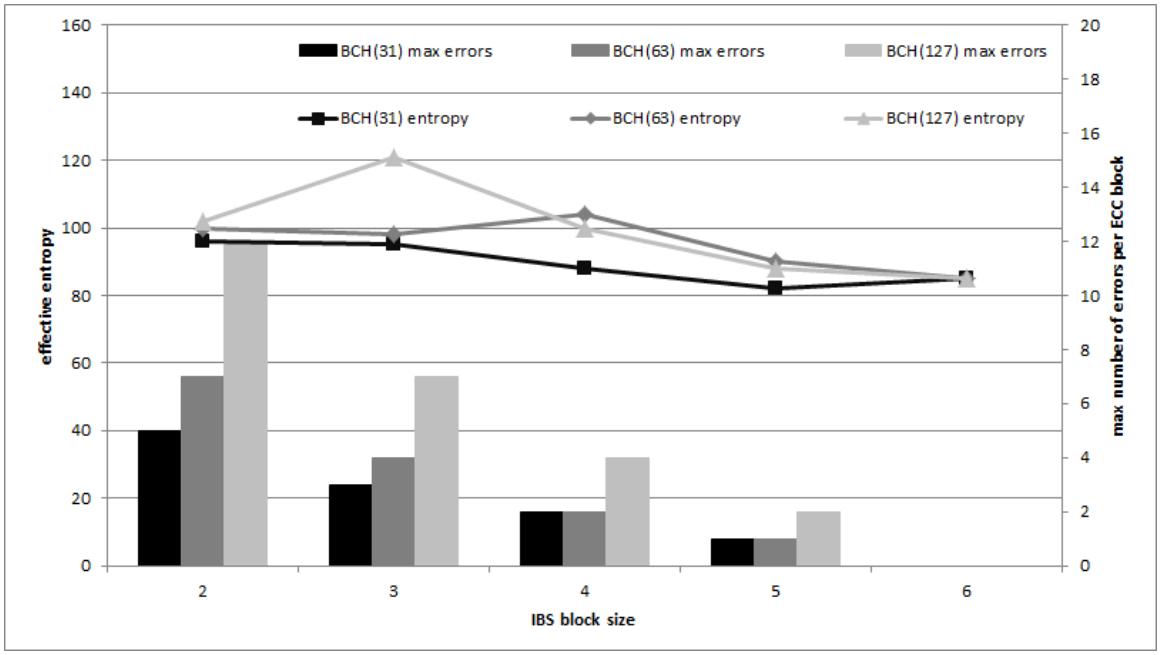

Figure 5.8: The estimated H_{∞}^{IBS} derived from the Virginia Tech dataset

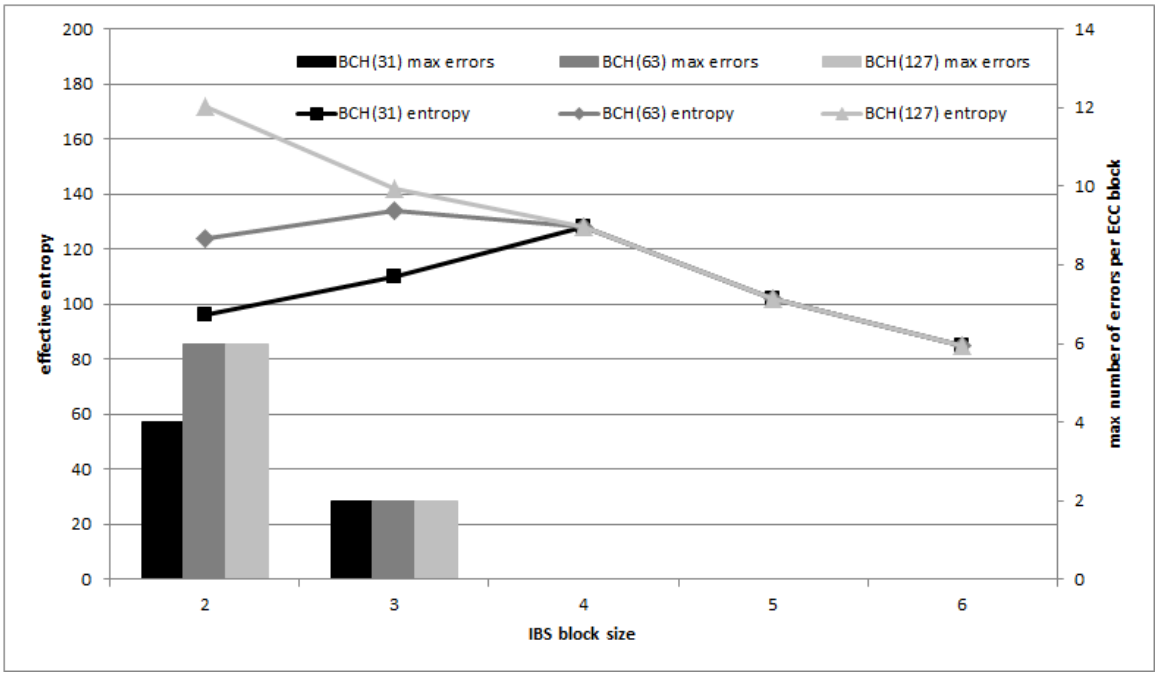

Figure 5.9: The estimated H_{∞}^{IBS} derived from the in-house dataset

5.5 Summary

This chapter provided the complete design of our group-based RO PUF. First of all, we designed a more hardware-efficient syndrome coding scheme based on Kendall tau distance. Kendall Syndrome Coding (KSC) can achieve a 9% gain in terms of effective min-entropy when compared with Compact Syndrome Coding (CSC) used previously. Besides, we incorporated the regression-based entropy distiller and the entropy packing process to achieve stronger secrecy. In this regard, two independent datasets were tested against the NIST test suite and both affirmed the i.i.d. uniform assumption we have made on our PUF output. Furthermore, we revised the previous grouping algorithm LISA as LISA-lite which simplifies the manufacturing process. In most error correcting scenarios we considered, our design is 50% or more efficient when compared with an IBS-based PUF under the i.i.d. assumption. We also showed that our group-based RO PUF can be quickly realized as an embedded system on FPGAs with a modern tool chain.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROP.	STATISTICAL TEST
0^{th}-order	10	9	4	6	4	4	9	3	5	11	0.187777	64/65	Frequency
	11	10	12	6	5	3	6	4	5	3	0.068316	64/65	BlockFrequency
	12	8	5	11	6	3	9	3	5	3	0.061374	63/65	CumulativeSums (m-2)
	9	10	5	7	4	7	5	6	7	5	0.676274	63/65	CumulativeSums (m-3)
	19	9	5	8	4	6	5	4	2	3	0.000049 *	59/65 *	Runs
	7	6	7	14	14	4	5	1	3	4	0.000919	65/65	LongestRun
	11	4	8	6	7	6	4	5	10	4	0.484073	64/65	Rank
	4	2	2	0	3	3	0	3	0	4	0.323011	21/21	FFT
	18	9	5	6	6	5	8	5	2	1	0.000145	62/65	ApproximateEntropy
	15	9	5	9	6	5	1	7	4	4	0.009867	63/65	Serial (forward)
	6	3	4	7	11	11	7	6	9	1	0.075967	65/65	Serial (backward)
1^{st}-order	7	7	6	7	9	2	8	5	5	9	0.494547	63/65	Frequency
	9	7	11	6	4	4	6	5	4	9	0.521707	61/65	BlockFrequency
	8	8	6	10	6	2	5	6	9	5	0.314919	63/65	CumulativeSums (m-2)
	6	7	9	7	5	6	4	8	7	6	0.921761	63/65	CumulativeSums (m-3)
	6	9	5	6	8	7	7	8	3	6	0.798722	64/65	Runs
	8	3	6	8	14	4	4	5	8	5	0.093645	65/65	LongestRun
	7	8	7	4	7	6	9	7	6	4	0.867205	62/65	Rank
	1	3	4	0	2	3	0	4	0	4	0.187777	21/21	FFT
	9	6	5	7	6	5	6	7	9	5	0.896359	63/65	ApproximateEntropy
	13	4	6	11	3	4	10	6	7	1	0.011121	65/65	Serial (forward)
	7	5	6	7	13	7	2	5	10	3	0.093645	63/65	Serial (backward)

Table 5.3: NIST test results derived from **the first half of the dataset** [31] using $0^{th} - 1^{st}$ order distillers, where the length of one bitstring is 400 (except 3x for FFT test), the block length for Frequency Test 32, the block length for Approximate Entropy Test 2 and the block length for Serial Test 5. '*' marks a failure.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROP.	STATISTICAL TEST
2^{nd}-order	8	7	9	6	10	5	4	5	5	6	0.631944	64/65	Frequency
	11	8	11	7	4	5	5	6	3	5	0.296269	64/65	BlockFrequency
	10	6	7	6	7	3	5	5	8	8	0.631944	64/65	CumulativeSums (m-2)
	8	10	6	7	5	6	4	5	6	8	0.760113	64/65	CumulativeSums (m-3)
	13	8	9	4	8	2	6	3	5	7	0.084389	62/65	Runs
	8	1	8	12	11	1	7	7	6	4	0.017828	64/65	LongestRun
	9	8	6	2	5	5	7	7	8	8	0.540669	64/65	Rank
	1	0	4	0	8	5	0	3	0	0	0.000065 *	21/21	FFT
	12	8	8	3	6	8	6	5	5	4	0.414146	63/65	ApproximateEntropy
	14	9	5	6	10	3	5	5	4	4	0.044252	64/65	Serial (forward)
	12	5	4	5	8	5	9	6	4	7	0.448203	64/65	Serial (backward)
3^{rd}-order	6	9	6	7	10	2	7	3	8	7	0.521707	65/65	Frequency
	7	8	9	9	8	2	6	9	1	6	0.226378	64/65	BlockFrequency
	3	13	5	10	10	10	3	0	7	4	0.002550	65/65	CumulativeSums (m-2)
	3	11	7	5	10	5	5	7	7	5	0.521707	65/65	CumulativeSums (m-3)
	10	10	6	8	8	4	10	4	3	2	0.127107	64/65	Runs
	4	5	7	8	14	2	3	9	4	9	0.022481	65/65	LongestRun
	6	10	6	3	5	3	2	8	14	8	0.020027	65/65	Rank
	2	6	1	0	2	8	0	2	0	0	0.000097 *	21/21	FFT
	11	6	6	11	7	3	7	3	5	6	0.296269	64/65	ApproximateEntropy
	9	11	8	3	6	5	7	4	5	7	0.561026	64/65	Serial (forward)
	8	8	5	8	2	7	7	5	6	9	0.540669	63/65	Serial (backward)
4^{th}-order	9	5	4	7	6	3	11	6	3	11	0.170659	63/65	Frequency
	8	13	8	4	3	10	4	5	4	6	0.093645	64/65	BlockFrequency
	7	5	9	5	2	4	10	9	8	6	0.448203	63/65	CumulativeSums (m-2)
	8	7	4	6	10	3	10	4	6	7	0.561026	62/65	CumulativeSums (m-3)
	7	9	6	6	6	1	6	9	8	7	0.358516	64/65	Runs
	4	10	5	8	11	9	4	5	3	6	0.271286	65/65	LongestRun
	9	5	5	3	6	11	10	5	6	5	0.414146	63/65	Rank
	3	4	2	0	3	3	0	2	0	4	0.323011	20/21	FFT
	8	10	4	7	8	5	2	7	4	10	0.351554	63/65	ApproximateEntropy
	9	5	9	5	5	7	6	4	7	8	0.760113	65/65	Serial (forward)
	3	6	6	7	10	5	8	9	3	8	0.272584	65/65	Serial (backward)

Table 5.4: Cont'd with the results using $2^{nd} - 4^{th}$ order models

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROP.	STATISTICAL TEST
5^{th}-order	6	8	5	3	11	3	11	6	6	6	0.271286	64/65	Frequency
	13	5	8	5	10	7	5	7	0	5	0.039609	63/65	BlockFrequency
	7	8	6	3	9	3	4	8	9	8	0.272584	64/65	CumulativeSums (m-2)
	8	9	7	6	5	3	5	13	6	3	0.170659	64/65	CumulativeSums (m-3)
	13	9	5	9	6	3	6	7	3	4	0.114933	61/65	Runs
	3	9	12	10	13	4	3	1	4	6	0.001981	64/65	LongestRun
	5	3	13	7	3	7	3	9	7	8	0.103803	63/65	Rank
	0	2	4	0	4	7	0	1	0	3	0.003277	21/21	FFT
	18	8	6	1	9	5	5	1	5	7	0.000074 *	64/65	ApproximateEntropy
	12	7	3	11	7	8	7	4	4	2	0.068316	64/65	Serial (forward)
	6	8	8	6	5	8	6	10	7	1	0.561026	65/65	Serial (backward)
6^{th}-order	7	4	9	2	4	5	9	8	5	12	0.154893	64/65	Frequency
	14	13	13	8	2	2	4	4	2	3	0.000019 *	65/65	BlockFrequency
	7	10	8	8	5	6	3	3	11	4	0.271286	65/65	CumulativeSums (m-2)
	5	6	8	6	12	7	4	6	6	5	0.403161	65/65	CumulativeSums (m-3)
	15	14	3	6	5	2	6	7	2	5	0.000323	64/65	Runs
	7	3	13	9	10	7	5	2	4	5	0.049394	65/65	LongestRun
	8	8	5	5	7	5	6	6	7	8	0.960834	63/65	Rank
	1	4	3	0	1	5	0	4	0	3	0.075967	21/21	FFT
	15	7	5	9	7	5	3	4	7	3	0.028264	64/65	ApproximateEntropy
	13	7	8	10	5	6	4	0	8	4	0.025217	64/65	Serial (forward)
	10	8	5	6	9	4	4	6	4	9	0.314919	64/65	Serial (backward)

Table 5.5: Cont'd with the results using $5^{nd} - 6^{th}$ order models

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROP.	STATISTICAL TEST
0^{th}-order	13	3	3	1	1	1	4	1	2	5	0.000008 *	30/34 *	Frequency
	16	3	4	3	0	2	2	1	2	1	0.000000 *	29/34 *	BlockFrequency
	14	4	3	1	2	2	2	1	3	2	0.000001 *	27/34 *	CumulativeSums (m-2)
	10	5	5	2	3	0	2	2	3	2	0.007297	29/34 *	CumulativeSums (m-3)
	15	6	4	2	0	3	1	1	1	1	0.000000 *	26/34 *	Runs
	12	1	4	3	3	4	4	1	2	0	0.000119	31/34	LongestRun
	5	3	6	7	4	1	2	2	3	1	0.196868	33/34	Rank
	1	2	1	0	2	1	0	1	0	3	0.581286	11/11	FFT
	17	4	3	4	1	1	1	2	1	0	0.000000 *	24/34 *	ApproximateEntropy
	16	4	2	6	1	1	2	1	0	1	0.000000 *	26/34 *	Serial (forward)
	6	5	3	2	4	2	3	5	2	2	0.471531	34/34	Serial (backward)
1^{st}-order	6	5	0	5	5	0	5	2	3	3	0.133610	33/34	Frequency
	6	6	4	2	4	1	2	2	4	3	0.541162	33/34	BlockFrequency
	6	3	5	3	4	1	1	3	4	4	0.380722	33/34	CumulativeSums (m-2)
	6	5	1	6	3	1	1	2	5	4	0.196868	33/34	CumulativeSums (m-3)
	7	5	4	2	4	0	4	2	0	6	0.058152	33/34	Runs
	3	1	4	7	6	3	2	2	5	1	0.196868	34/34	LongestRun
	4	3	4	5	0	4	4	3	6	1	0.465914	32/34	Rank
	2	4	0	0	0	1	0	4	0	0	0.003950	11/11	FFT
	6	1	6	6	3	3	3	1	3	2	0.283561	32/34	ApproximateEntropy
	6	5	6	4	3	1	3	2	0	4	0.236992	32/34	Serial (forward)
	3	5	7	7	4	2	0	1	1	4	0.037462	34/34	Serial (backward)

Table 5.6: NIST test results derived from **the first half of the in-house dataset** using $0^{th} - 1^{st}$ order distillers, where the length of one bitstring is 400 (except 3x for FFT test), the block length for Frequency Test 32, the block length for Approximate Entropy Test 2 and the block length for Serial Test 5. '*' marks a failure.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROP.	STATISTICAL TEST
2^{nd}-order	9	3	3	1	4	2	4	1	2	5	0.058152	32/34	Frequency
	8	4	4	4	1	2	4	6	1	0	0.037462	32/34	BlockFrequency
	6	6	2	4	3	5	3	1	2	2	0.465914	33/34	CumulativeSums (m-2)
	11	1	4	4	5	0	2	3	3	1	0.000757	32/34	CumulativeSums (m-3)
	11	6	2	0	2	1	6	1	1	4	0.000069 *	31/34	Runs
	6	3	7	3	3	4	0	3	4	1	0.196868	33/34	LongestRun
	9	8	2	4	1	3	2	2	3	0	0.002716	32/34	Rank
	2	2	4	0	0	2	0	1	0	0	0.064760	10/11	FFT
	12	3	6	4	4	1	2	0	1	1	0.000023 *	32/34	ApproximateEntropy
	11	1	7	1	5	2	2	0	4	1	0.000053 *	32/34	Serial (forward)
	5	5	6	3	2	1	2	2	7	1	0.133610	34/34	Serial (backward)
3^{rd}-order	3	5	4	5	1	2	5	3	4	2	0.563683	34/34	Frequency
	5	6	5	2	5	2	2	3	3	1	0.541162	33/34	BlockFrequency
	5	2	5	6	2	2	5	3	2	2	0.293582	34/34	CumulativeSums (m-2)
	4	2	2	5	7	3	4	3	2	2	0.293582	34/34	CumulativeSums (m-3)
	7	5	7	1	3	3	4	1	1	2	0.072049	31/34	Runs
	1	1	5	4	6	5	4	5	1	2	0.283561	34/34	LongestRun
	6	4	4	3	4	4	2	3	1	3	0.654263	33/34	Rank
	1	4	2	0	1	1	0	1	0	1	0.216159	11/11	FFT
	4	5	9	3	3	2	1	2	4	1	0.058152	34/34	ApproximateEntropy
	7	3	5	1	4	2	4	2	3	3	0.541162	33/34	Serial (forward)
	4	4	8	2	1	3	3	1	4	4	0.236992	34/34	Serial (backward)
4^{th}-order	5	7	1	1	5	4	4	2	1	4	0.196868	33/34	Frequency
	7	2	6	1	7	0	4	3	2	2	0.029914	34/34	BlockFrequency
	5	5	3	2	5	2	2	3	4	3	0.739897	33/34	CumulativeSums (m-2)
	4	7	1	6	2	2	3	2	4	3	0.337055	34/34	CumulativeSums (m-3)
	5	8	4	2	2	0	4	7	1	1	0.011803	33/34	Runs
	4	2	4	4	7	2	2	4	4	1	0.541162	34/34	LongestRun
	6	1	2	3	5	3	4	4	2	4	0.471531	34/34	Rank
	1	4	2	0	0	3	0	1	0	0	0.033490	11/11	FFT
	7	6	1	4	5	1	5	1	3	1	0.072049	33/34	ApproximateEntropy
	4	4	5	3	4	5	3	4	2	0	0.471531	34/34	Serial (forward)
	2	5	4	5	6	4	0	2	1	5	0.236992	34/34	Serial (backward)

Table 5.7: Cont'd with the results using $2^{nd} - 4^{th}$ order models

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROP.	STATISTICAL TEST
5^{th}-order	4	3	2	2	4	2	6	3	2	6	0.380722	34/34	Frequency
	5	1	2	2	3	1	2	6	5	7	0.133610	34/34	BlockFrequency
	2	5	1	4	3	1	4	7	5	2	0.283561	34/34	CumulativeSums (m-2)
	4	0	1	2	4	5	3	4	5	6	0.337055	34/34	CumulativeSums (m-3)
	5	7	3	5	1	2	3	2	1	5	0.236992	34/34	Runs
	2	1	5	6	9	4	3	3	1	0	0.009292	34/34	LongestRun
	3	5	2	3	6	1	4	1	4	5	0.541162	34/34	Rank
	1	2	1	0	2	1	0	3	0	1	0.581286	11/11	FFT
	7	5	1	5	2	1	3	2	4	4	0.283561	34/34	ApproximateEntropy
	5	6	5	3	1	4	3	4	1	2	0.541162	34/34	Serial (forward)
	5	5	2	1	4	8	2	2	4	1	0.109242	34/34	Serial (backward)
6^{th}-order	7	4	3	4	5	1	3	1	4	2	0.397806	34/34	Frequency
	6	2	4	5	4	3	5	0	0	5	0.162612	33/34	BlockFrequency
	6	4	5	3	2	3	2	4	3	2	0.654263	34/34	CumulativeSums (m-2)
	8	1	4	8	3	1	2	1	2	4	0.011803	34/34	CumulativeSums (m-3)
	11	3	3	0	4	3	3	3	2	2	0.003488	32/34	Runs
	3	4	9	7	4	1	0	3	2	1	0.005718	34/34	LongestRun
	8	3	6	3	3	0	3	3	3	2	0.133610	31/34	Rank
	2	2	2	0	0	2	0	2	0	1	0.581286	11/11	FFT
	9	5	5	1	2	3	4	1	2	2	0.037462	33/34	ApproximateEntropy
	8	6	2	3	6	2	4	1	2	0	0.023812	31/34	Serial (forward)
	4	4	6	3	3	2	2	4	4	2	0.739897	32/34	Serial (backward)

Table 5.8: Cont'd with the results using $5^{nd} - 6^{th}$ order models

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROP.	STATISTICAL TEST
1^{st}-order	7	6	8	4	7	6	10	4	8	4	0.609372	63/64	Frequency
	12	9	7	4	6	6	5	4	7	4	0.465914	62/64	BlockFrequency
	8	6	11	8	5	5	5	7	5	4	0.517608	63/64	CumulativeSums (m-2)
	7	8	10	8	11	4	8	3	2	3	0.109242	63/64	CumulativeSums (m-3)
	7	2	14	10	6	2	5	5	8	5	0.023812	63/64	Runs
	6	3	12	7	4	9	7	7	5	4	0.366511	63/64	LongestRun
	7	9	6	6	5	4	8	7	8	4	0.817009	64/64	Rank
	1	2	3	0	4	4	0	6	0	1	0.028264	21/21	FFT
	10	8	3	9	5	5	7	6	5	6	0.517608	63/64	ApproximateEntropy
	10	5	6	7	6	5	6	6	6	7	0.933004	64/64	Serial (forward)
	4	9	9	6	6	8	3	6	4	9	0.380722	64/64	Serial (backward)

Table 5.9: NIST test results derived from **the second half of the Virginia Teach dataset** [31] with 1^{st}-order polynomial regression model applied in Step 2. All tests are passed.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROP.	STATISTICAL TEST
1^{st}-order	5	4	3	2	7	0	2	5	3	2	0.259438	31/33	Frequency
	9	6	1	3	2	1	4	2	3	2	0.033490	31/33	BlockFrequency
	7	5	2	4	2	1	4	1	6	1	0.120885	31/33	CumulativeSums (m-2)
	6	4	3	3	4	3	5	1	4	0	0.502674	30/33	CumulativeSums (m-3)
	8	2	3	2	1	4	2	2	5	4	0.216159	32/33	Runs
	6	7	2	5	2	4	0	2	1	4	0.098607	33/33	LongestRun
	6	3	3	8	1	2	3	2	4	1	0.120885	31/33	Rank
	0	3	1	0	1	1	0	5	0	0	0.003950	11/11	FFT
	9	4	2	4	3	0	2	4	3	2	0.064760	33/33	ApproximateEntropy
	7	3	2	5	4	5	4	3	0	0	0.120885	32/33	Serial (forward)
	3	3	5	3	2	5	3	5	1	3	0.697921	32/33	Serial (backward)

Table 5.10: NIST test results derived from **the second half of our own dataset** with 1^{st}-order polynomial regression model applied in Step 2. All tests are passed

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROP.	STATISTICAL TEST
0^{th}-order	17	16	12	12	13	12	17	6	7	17	0.231636	128/129	Frequency
	27	18	23	11	11	5	10	6	12	6	0.000004 *	125/129	BlockFrequency
	20	15	19	18	11	10	11	8	10	7	0.056599	126/129	CumulativeSums (m-2)
	16	20	14	12	17	15	7	9	12	7	0.130323	127/129	CumulativeSums (m-3)
	39	16	12	11	11	12	10	4	6	8	0.000000 *	118/129 *	Runs
	13	9	24	23	18	9	10	9	4	10	0.000150	129/129	LongestRun
	18	13	14	12	12	17	11	9	14	9	0.460053	125/129	Rank
	6	6	6	0	7	6	0	7	0	5	0.018355	42/43	FFT
	35	17	15	12	9	14	10	7	5	5	0.000000 *	124/129	ApproximateEntropy
	22	21	12	16	17	9	6	11	6	9	0.002632	124/129	Serial (forward)
	11	9	12	18	14	17	11	8	17	12	0.493319	129/129	Serial (backward)
1^{st}-order	16	17	8	13	16	5	19	9	11	15	0.091249	126/129	Frequency
	18	19	17	12	10	10	10	8	11	14	0.289860	124/129	BlockFrequency
	20	15	13	13	12	6	9	12	17	12	0.277369	126/129	CumulativeSums (m-2)
	18	11	16	11	16	13	9	12	11	12	0.552185	126/129	CumulativeSums (m-3)
	19	18	11	9	12	11	12	12	7	18	0.211194	128/129	Runs
	10	9	13	17	27	11	7	10	14	11	0.004074	129/129	LongestRun
	15	15	17	7	8	12	19	16	12	8	0.174809	124/129	Rank
	3	4	9	0	6	4	0	10	0	7	0.000434	42/43	FFT
	17	15	14	14	18	9	10	9	16	7	0.302788	127/129	ApproximateEntropy
	21	7	16	20	8	14	16	10	10	7	0.015401	127/129	Serial (forward)
	13	10	17	14	18	15	6	10	16	10	0.316158	126/129	Serial (backward)

Table 5.11: NIST test results derived from **the entire Virginia Tech's dataset** [31] using $0^{th} - 1^{st}$ order distillers, where the length of one bitstring is 400 (except 3x for FFT test), the block length for Frequency Test 32, the block length for Approximate Entropy Test 2 and the block length for Serial Test 5. '*' marks a failure.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROP.	STATISTICAL TEST
2^{nd}-order	20	15	11	10	19	10	14	7	9	14	0.143910	128/129	Frequency
	17	16	17	16	9	8	15	11	8	12	0.358975	127/129	BlockFrequency
	21	13	13	11	11	9	14	6	16	15	0.201550	127/129	CumulativeSums (m-2)
	19	14	11	15	11	12	8	11	13	15	0.437182	128/129	CumulativeSums (m-3)
	22	14	17	12	11	9	13	7	12	12	0.174809	124/129	Runs
	14	4	19	17	22	6	10	14	7	16	0.001690	128/129	LongestRun
	19	19	12	6	10	13	13	12	12	13	0.302788	128/129	Rank
	6	4	6	0	9	8	0	6	0	4	0.003598	43/43	FFT
	21	15	15	13	10	19	11	9	9	7	0.077998	124/129	ApproximateEntropy
	22	19	12	12	16	6	14	11	11	6	0.021842	126/129	Serial (forward)
	20	13	9	8	13	16	21	12	7	10	0.045489	128/129	Serial (backward)
3^{rd}-order	13	18	11	12	18	6	17	10	12	12	0.316158	129/129	Frequency
	15	17	15	17	12	5	12	16	7	13	0.201550	128/129	BlockFrequency
	10	25	11	17	16	14	8	8	10	10	0.013689	129/129	CumulativeSums (m-2)
	10	18	17	9	16	8	13	12	15	11	0.493319	129/129	CumulativeSums (m-3)
	23	17	12	12	13	9	17	9	6	11	0.034444	126/129	Runs
	9	12	19	18	25	7	6	12	7	14	0.000733	129/129	LongestRun
	18	20	9	6	7	12	11	15	18	13	0.045489	127/129	Rank
	5	9	2	0	5	13	0	7	0	2	0.000003 *	43/43	FFT
	24	14	13	19	10	8	12	11	6	12	0.015401	126/129	ApproximateEntropy
	19	19	15	15	9	13	11	9	11	8	0.231636	128/129	Serial (forward)
	16	13	13	12	7	16	16	11	11	14	0.552185	127/129	Serial (backward)
4^{th}-order	11	13	9	21	16	6	22	8	4	19	0.000381	126/129	Frequency
	17	21	14	8	11	17	9	14	7	11	0.091249	127/129	BlockFrequency
	11	12	12	13	16	11	16	14	16	8	0.665311	126/129	CumulativeSums (m-2)
	13	17	6	14	19	9	15	12	12	12	0.358975	125/129	CumulativeSums (m-3)
	14	17	16	10	17	7	7	17	15	9	0.166594	126/129	Runs
	9	14	19	15	23	15	8	7	11	8	0.013689	128/129	LongestRun
	16	12	16	8	12	18	17	13	9	8	0.344248	126/129	Rank
	6	10	6	0	5	6	0	6	0	4	0.003598	42/43	FFT
	19	15	8	9	17	17	7	12	10	15	0.151154	125/129	ApproximateEntropy
	19	10	16	11	12	13	12	6	12	18	0.289860	129/129	Serial (forward)
	10	11	14	12	21	7	13	16	11	14	0.329976	128/129	Serial (backward)

Table 5.12: Cont'd with the results using $2^{nd} - 4^{th}$ order models

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROP.	STATISTICAL TEST
5^{th}-order	10	13	10	9	18	9	20	14	8	18	0.112055	128/129	Frequency
	25	14	10	14	13	12	11	13	3	14	0.010167	126/129	BlockFrequency
	7	18	14	8	19	7	11	12	19	14	0.056599	128/129	CumulativeSums (m-2)
	10	15	13	9	13	12	11	19	14	13	0.575157	128/129	CumulativeSums (m-3)
	21	20	8	16	11	9	11	15	9	9	0.048059	124/129	Runs
	6	12	22	21	24	9	10	4	8	13	0.000024 *	128/129	LongestRun
	12	8	17	18	11	11	10	13	17	12	0.571108	125/129	Rank
	2	6	7	0	8	10	0	4	0	6	0.000529	43/43	FFT
	25	13	12	7	18	12	5	7	16	14	0.001801	128/129	ApproximateEntropy
	21	12	9	19	11	12	13	7	13	12	0.166594	126/129	Serial (forward)
	13	15	11	12	13	16	10	15	13	11	0.915772	128/129	Serial (backward)
6^{th}-order	14	6	13	9	14	11	21	15	7	19	0.043046	128/129	Frequency
	21	25	20	17	10	5	10	7	4	10	0.000007 *	127/129	BlockFrequency
	14	13	11	15	13	14	11	8	21	9	0.405918	129/129	CumulativeSums (m-2)
	7	15	15	11	21	11	7	12	18	12	0.101175	129/129	CumulativeSums (m-3)
	25	22	9	10	11	6	13	13	7	13	0.000890	128/129	Runs
	12	6	26	14	21	13	8	7	10	12	0.000529	129/129	LongestRun
	19	12	11	8	10	13	16	12	12	16	0.304210	125/129	Rank
	3	7	7	0	2	12	0	9	0	3	0.000009 *	42/43	FFT
	21	12	11	20	15	13	5	9	15	8	0.024505	128/129	ApproximateEntropy
	20	16	17	16	10	9	10	5	14	12	0.091249	128/129	Serial (forward)
	20	15	12	15	10	7	9	15	8	18	0.117876	128/129	Serial (backward)

Table 5.13: Cont'd with the results using $5^{th} - 6^{th}$ order models

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROP.	STATISTICAL TEST
0^{th}-order	21	9	4	1	5	2	8	3	5	9	0.000000 *	62/67 *	Frequency
	28	6	5	5	1	6	5	2	7	2	0.000000 *	57/67 *	BlockFrequency
	25	6	7	5	2	5	6	2	5	4	0.000000 *	59/67 *	CumulativeSums (m-2)
	20	10	6	7	4	1	4	4	7	4	0.000004 *	60/67 *	CumulativeSums (m-3)
	21	11	7	10	3	3	1	6	2	3	0.000000 *	58/67 *	Runs
	16	5	10	7	7	5	6	2	4	5	0.006867	64/67	LongestRun
	10	5	11	7	7	5	4	5	10	3	0.248010	66/67	Rank
	2	2	6	0	2	2	0	5	0	3	0.033490	22/22	FFT
	24	8	10	9	3	2	3	5	2	1	0.000000 *	57/67 *	ApproximateEntropy
	23	8	3	11	4	3	6	4	3	2	0.000000 *	57/67 *	Serial (forward)
	9	10	7	4	7	4	6	10	6	4	0.561026	67/67	Serial (backward)
1^{st}-order	9	11	3	8	8	0	10	5	6	7	0.055085	66/67	Frequency
	13	12	10	3	5	3	7	3	4	7	0.009867	66/67	BlockFrequency
	10	5	10	9	8	4	2	6	5	8	0.296269	65/67	CumulativeSums (m-2)
	11	7	5	11	6	4	7	3	7	6	0.351554	66/67	CumulativeSums (m-3)
	14	7	6	8	4	3	6	6	3	10	0.049394	64/67	Runs
	5	3	9	13	12	7	6	3	7	2	0.012526	67/67	LongestRun
	11	5	9	8	2	6	10	4	9	3	0.103803	63/67	Rank
	2	6	1	0	0	4	0	6	0	3	0.001859	22/22	FFT
	12	5	7	7	9	8	6	2	7	4	0.271286	64/67	ApproximateEntropy
	8	9	12	9	5	4	8	4	0	8	0.039609	63/67	Serial (forward)
	6	7	13	11	7	2	1	9	7	4	0.012526	67/67	Serial (backward)

Table 5.14: NIST test results derived from **the entire in-house dataset** using $0^{th} - 1^{st}$ order distillers, where the length of one bitstring is 400 (except 3x for FFT test), the block length for Frequency Test 32, the block length for Approximate Entropy Test 2 and the block length for Serial Test 5. '*' marks a failure.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROP.	STATISTICAL TEST
2^{nd}-order	19	9	5	5	6	2	7	1	4	9	0.000014 *	60/67 *	Frequency
	17	6	6	7	6	4	7	7	6	1	0.003277	60/67 *	BlockFrequency
	15	12	6	9	4	6	5	3	3	4	0.003277	62/67 *	CumulativeSums (m-2)
	21	6	5	7	8	2	5	5	6	2	0.000002 *	61/67 *	CumulativeSums (m-3)
	18	8	8	0	8	4	8	3	4	6	0.000065 *	63/67	Runs
	13	3	10	9	7	8	3	7	5	2	0.031655	64/67	LongestRun
	13	13	3	6	4	5	5	6	8	4	0.022481	63/67	Rank
	4	2	8	0	1	3	0	3	0	1	0.001268	21/22	FFT
	23	6	10	10	5	2	4	3	2	2	0.000000 *	61/67 *	ApproximateEntropy
	20	3	11	3	7	9	5	3	5	1	0.000000 *	63/67	Serial (forward)
	10	8	8	8	7	3	4	5	11	3	0.226378	67/67	Serial (backward)
3^{rd}-order	8	12	8	8	3	5	9	6	4	4	0.248010	67/67	Frequency
	13	14	7	2	8	4	6	7	3	3	0.002550	66/67	BlockFrequency
	11	8	8	10	6	5	7	5	5	2	0.323011	67/67	CumulativeSums (m-2)
	10	7	6	9	13	5	5	6	4	2	0.084389	67/67	CumulativeSums (m-3)
	10	9	10	5	8	5	6	2	6	6	0.414146	63/67	Runs
	4	7	12	13	8	6	4	6	3	4	0.039609	67/67	LongestRun
	10	8	12	3	7	7	6	4	6	4	0.248010	66/67	Rank
	3	5	3	0	3	3	0	4	0	1	0.120885	21/22	FFT
	7	10	13	7	4	3	8	7	5	3	0.093645	66/67	ApproximateEntropy
	15	8	10	4	8	2	7	5	4	4	0.009867	65/67	Serial (forward)
	6	8	17	6	4	4	7	5	5	5	0.008749	67/67	Serial (backward)
4^{th}-order	11	9	1	7	9	8	7	3	3	9	0.068316	65/67	Frequency
	12	5	12	6	10	5	5	3	6	3	0.044252	67/67	BlockFrequency
	10	10	7	4	9	5	4	5	7	6	0.272584	65/67	CumulativeSums (m-2)
	8	10	7	10	4	6	4	8	5	5	0.314919	66/67	CumulativeSums (m-3)
	7	11	4	4	9	4	8	12	2	6	0.061374	66/67	Runs
	8	4	9	12	11	6	4	6	5	2	0.075967	67/67	LongestRun
	8	6	8	3	5	9	4	7	9	8	0.448602	66/67	Rank
	2	7	4	0	2	5	0	2	0	0	0.001859	22/22	FFT
	12	8	4	9	5	2	10	6	7	4	0.114933	66/67	ApproximateEntropy
	6	7	10	5	6	9	8	7	4	5	0.631944	67/67	Serial (forward)
	3	9	6	9	8	9	1	7	6	9	0.248010	67/67	Serial (backward)

Table 5.15: Cont'd with the results using $2^{nd} - 4^{th}$ order models

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROP.	STATISTICAL TEST
5^{th}-order	9	9	9	2	6	2	8	6	4	12	0.061374	66/67	Frequency
	13	5	6	6	6	2	4	9	8	8	0.140402	63/67	BlockFrequency
	7	10	4	7	7	2	7	12	7	4	0.187777	66/67	CumulativeSums (m-2)
	11	5	4	6	9	6	4	7	8	7	0.358516	66/67	CumulativeSums (m-3)
	6	10	4	11	3	7	4	8	3	11	0.084389	66/67	Runs
	6	4	11	10	12	8	5	4	4	3	0.061374	67/67	LongestRun
	10	9	8	5	8	5	7	3	5	7	0.403161	67/67	Rank
	4	2	1	0	4	2	0	7	0	2	0.008237	22/22	FFT
	12	6	6	9	3	3	6	8	6	8	0.296269	67/67	ApproximateEntropy
	9	9	7	10	2	10	5	5	3	7	0.206325	66/67	Serial (forward)
	8	11	5	6	6	11	5	4	5	6	0.448203	67/67	Serial (backward)
6^{th}-order	14	9	6	8	7	1	5	2	8	7	0.017828	66/67	Frequency
	10	8	7	11	6	6	9	1	2	7	0.084389	65/67	BlockFrequency
	13	8	8	5	7	5	4	6	7	4	0.323011	67/67	CumulativeSums (m-2)
	14	6	7	13	6	3	7	2	2	7	0.002550	67/67	CumulativeSums (m-3)
	14	9	4	3	7	7	8	6	5	4	0.084389	65/67	Runs
	8	6	12	14	9	4	2	6	3	3	0.003712	66/67	LongestRun
	14	5	10	7	6	4	8	3	4	6	0.061374	63/67	Rank
	3	4	2	0	2	4	0	4	0	3	0.216159	22/22	FFT
	17	6	10	5	6	4	7	4	4	4	0.002892	66/67	ApproximateEntropy
	13	8	9	7	8	3	7	3	7	2	0.061374	63/67	Serial (forward)
	6	7	14	6	7	6	5	4	6	6	0.296269	64/67	Serial (backward)

Table 5.16: Cont'd with the results using $5^{nd} - 6^{th}$ order models

Chapter 6

Conclusion

A Physical Unclonable Function (PUF) is a physical structure whose functional characteristic is hard to predict before fabrication but once fabricated its characteristic is rather stable and unique. PUF is a security primitive that can generate cryptographic keys and evade physical attacks. In this dissertation, we discussed several original methodologies to improve RO PUFs in terms of security, reliability and hardware efficiency. We started with temperature-aware cooperation (TAC) and then introduced LISA to extract the maximal entropy of a comparison-based RO PUF through finding longest increasing subsequences iteratively. After that, we broke up the chains of Neighbor Coding and applied the regression-based entropy distiller right after frequency measurement to resolve the security problems posed by the underlying spatial correlation. We also incorporated ECC into the design and devised the more error tolerant Kendall Syndrome Coding (KSC) to further reduce the complexity of ECC. According to the NIST test results of two large-scale datasets, the output of our group-based RO PUF is indistinguishable from those drawn from the ideal uniform distribution. Under the same i.i.d. output assumption and reliability criteria, our design is 50% or more efficient than the state-of-the-art IBS-based design in most cases. In addition, our design is self-sufficient, requiring no seeding from outside the chip that others do. While existing proposals are pre-

dominantly in pure hardware, we take advantage of the 'free' computing resource of the existing microprocessor for less cost on the PUF IP. Our software-hardware co-design can be quickly realized on FPGAs as an embedded system using today's tool chain.

Bibliography

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

[2] Ross Anderson and Markus Kuhn. Low cost attacks on tamper resistant devices. *Proceedings of 5th International Workshop on Security Protocols, LNCS Vol. 1361, pp. 125–136, Springer*, Apr. 1997.

[3] Ross Anderson and Markus Kuhn. Tamper resistance — a cautionary note. *Proceedings of 2nd USENIX Workshop on Electronic Commerce, pp. 1–11*, Nov. 1996.

[4] Ross J. Anderson. Security engineering: A guide to building dependable distributed systems. *Wiley Computer Publishing*, Jan. 2001.

[5] James Nechvatal Miles Smid Elaine Barker Stefan Leigh Mark Levenson Mark Vangel David Banks Alan Heckert James Dray San Vo Andrew Rukhin, Juan Soto and Lawrence E Bassham III. A statistical test suite for random and pseudorandom number generators for cryptographic applications. *NIST Special Publication 800-22 Revision 1a*, Apr. 2010.

[6] D. S. Boning B. E. Stine and J. E. Chung. Analysis and decomposition of spatial variation in integrated circuit processes and devices. *IEEE Transactions on Semiconductor Manufacturing, Vol. 10, Issue 1, pp. 24–91*, Feb. 1997.

[7] M. van Dijk B. Gassend, D. Clarke and S. Devadas. Controlled physical random functions. *Proceedings of the 18th Annual Computer Security Applications Conference*, Dec. 2002.

[8] M. van Dijk B. Gassend, D. Clarke and S. Devadas. Silicon physical random functions. *Proceedings of 9th ACM Computer and Communications Security Conference (CCS)*, Nov. 2002.

[9] T. Kevenaar B. Škorić, S. Maubach and P. Tuyls. Information-theoretic analysis of capacitive physical unclonable functions. *AIP Journal of Applied Physics Vol. 100*, Jul. 2006.

[10] W. Ophey R. Wolters N. Verhaegh B. Škorić, G.-J. Schrijen and J. van Geloven. Experimental hardware for coating pufs and optical pufs. *Security with Noisy Data, pp. 255–268, Springer*, Apr. 2007.

[11] Alexander Barg and Arya Mazumdar. Codes in permutations and error correction for rank modulation. *Proceedings of 2010 IEEE International Symposium on Information Theory (ISIT)*, June 2010.

[12] K. Bernstein, D. J. Frank, A. E. Gattiker, W. Haensch, B. L. Ji, S. R. Nassif, E. J. Nowak, D. J. Pearson, and N. J. Rohrer. High-performance cmos variability in the 65-nm regime and beyond. *IBM Journal of Research and Development*, 50(4.5):433–449, 2006.

[13] Eduardo I. Boemo and Sergio López-Buedo. Thermal monitoring on fpgas using ring-oscillators. *Proceedings of 7th International Workshop on Field-Programmable Logic and Applications(FPLA), LNCS Vol. 1304, pp. 69–78, Springer*, Sep. 1997.

[14] Duane Boning and Sani Nassif. Chapter 6: Models of process variations in device and interconnect. *Design of High-Performance Microprocessor Circuits, Wiley, John & Sons Incorporated*, Oct. 2000.

[15] H. Chadwick and L. Kurz. Rank permutation group codes based on kendall's correlation statistic. *IEEE Transactions on Information Theory, Vol. 15, No. 2, pp. 306–315*, 1969.

[16] H. Chadwick and I. Reed. The equivalence of rank permutation codes to a new class of binary codes. *IEEE Transactions on Information Theory, Vol. 16, No. 5, pp. 640–641*, 1970.

[17] Ahmad-Reza Sadeghi Jamshid Shokrollahi Christoph Bösch, Jorge Guajardo and Pim Tuyls. Efficient helper data key extractor on fpgas. *Proceedings of 10th IACR International Workshop on Cryptographic Hardware and Embedded Systems (CHES), LNCS Vol. 5154, pp. 181–197, Springer*, Aug. 2008.

[18] W. Burleson D. Holcomb and K. Fu. Initial sram state as a fingerprint and source of true random numbers for rfid tags. *Proceedings of the Conference on RFID Security 07*, Jul. 2007.

[19] B. Gassend M. van Dijk E. Suh D. Lim, J-W. Lee and S. Devadas. Extracting secret keys from integrated circuits. *IEEE Transactions on VLSI Systems, Vol. 13, Issue 10*, Oct. 2005.

[20] Frederic Stumpf Dominik Merli and Claudia Eckert. Improving the quality of ring oscillator pufs on fpgas. *Proceedings of the 5th Workshop on Embedded Systems Security*, Oct. 2010.

[21] T. Maung R. Divecha D. Boning J. Chung K. Chang G. Ray D. Bradbury O. Nakagawa S. Oh E. Chang, B. Stine and D. Bartelink. Using a statistical metrology framework to identify systematic and random sources of die- and wafer-level ild thickness variation in cmp processes. *International Electron Devices Meeting*, pages 499–502, Dec. 1995.

[22] Blaise Gassend Marten ven Dijk Srinivas Devadas G. E. Suh, Dwaine Clarke. Aegis: Architecture for tamper-evident and tamper-resistant processing. *Proceedings of 17th ACM International conference on Supercomputing (ICS), pp. 160–171*, Jun. 2003.

[23] C. W. O'Donnell G. E. Suh and S. Devadas. Aegis: A single-chip secure processor. *IEEE Journal of Design and Test of Computers, Vol. 24, Issue 6, pp. 570–580*, Nov. 2007.

[24] Geert-Jan Schrijen Jorge Guajardo, Sandeep S. Kumar and Pim Tuyls. Physical unclonable functions and public-key crypto for fpga ip protection. *Proceedings of 17th IEEE International Conference on Field Programmable Logic and Applications (FPL*, Aug. 2007.

[25] Geert-Jan Schrijen Jorge Guajardo, Sandeep S. Kumar and Pim Tuyls. Fpga intrinsic pufs and their use for ip protection. *Proceedings of 9th IACR International Workshop on Cryptographic Hardware and Embedded Systems (CHES), LNCS 4727, Springer*, Sep. 2007.

[26] Maurice Kendall. Rank correlation methods. *London: Griffin*, 1958.

[27] Donald E. Knuth. Volume 2: Seminumerical algorithms (3^{st} edition). *the Art of Computer Programming, Addison-Wesley*, 1998.

[28] Y. Mathieu L. Sauvage, S. Guilley. Electromagnetic radiations of fpgas: High spatial resolution cartography and attack on a cryptographic module. *ACM Transactions on Reconfigurable Technology and Systems, Vol. 2, No. 1*, Jul. 2009.

[29] Costas Spanos-Kun Qian Lerong Cheng, Puneet Gupta and Lei He. Physically justifiable die-level modeling of spatial variation in view of systematic across wafer variability. *Proceedings of 46th ACM/IEEE International Annual Design Automation Conference (DAC)*, Jul. 2009.

[30] Frank Liu. A general framework for spatial correlation modeling in vlsi design. *Proceedings of 44th ACM/IEEE Design Automation Conference (DAC) pp. 817–822*, Jun. 2007.

[31] A. Maiti and P. Schaumont. A large scale characterization of ro-puf. *Proceedings of 3rd IEEE International Workshop on Hardware Oriented Security and Trust (HOST)*, Jun. 2010.

[32] A. Maiti and P. Schaumont. Improving the quality of a physical unclonable function using configurable ring oscillators. *Proceedings of 19th IEEE International Conference on Field Programmable Logic and Applications (FPLA)*, Sep. 2009.

[33] S. Masuda and K. Nakajima. Optimal algorithms for interval containment graphs. *Proceedings of 29th Midwest Symposium on Circuits and Systems, pp. 431–434*, Aug. 1986.

[34] B. Jun P. Kocher, J. Jaffe. Differential power analysis. *Proceedings of Crypto 99, LNCS Vol. 1666, Springer*, Aug. 1999.

[35] Michael J. Panik. Advanced statstics from an elementary point of viev. *Elsevier Academic Press*, 2005.

[36] R. Pappu. Physical one-way functions. *PhD thesis*, Mar. 2001.

[37] Boris Škorić Jan van Geloven Nynke Verhaegh Pim Tuyls, Geert-Jan Schrijen and Rob Wolters. Read-proof hardware from protective coatings. *Proceedings of 8th IACR International Workshop on Cryptographic Hardware and Embedded Systems (CHES), LNCS Vol. 4249, Springer*, Oct. 2006.

[38] J. Taylor R. Pappu, B. Recht and N. Gershenfeld. Physical one-way functions. *Science Magazine Vol. 297. no. 5589, pp. 2026–2030*, Sep. 2002.

[39] Pim Tuyls Roel Maes and Ingrid Verbauwhede. Low-overhead implementation of a soft decision helper data algorithm for sram pufs. *Proceedings of 11th IACR International Workshop on Cryptographic Hardware and Embedded Systems (CHES), LNCS Vol. 5747, pp. 332–347, Springer*, Sep. 2009.

[40] S. Paral R. Sowell T. Ziola S. Devadas, E. Suh and V. Khandelwal. Design and implementation of puf-based "unclonable" rfid ics for anti-counterfeiting and security applications. *Proceedings of 2nd IEEE International Conference on RFID*, Apr. 2008.

[41] R. Maes G.J. Schrijen S. Kumar, J. Guajardo and P. Tuyls. the butterfly puf: Protecting ip on every fpga. *Proceedings of 1st IEEE International Workshop on Hardware Oriented Security and Trust (HOST)*, Jun. 2008.

[42] J. Garrido S. Lopez-Buedo and E. I. Boemo. Dynamically inserting, operting and eliminating thermal sensors of fpga-based systems. *IEEE Transacations on Components and Packaging Technologies, Vol. 25, No. 4*, Dec. 2002.

[43] M. Aoki S. Ohkawa and H. Masuda. Analysis and characterization of device variations in an lsi chip using an integrated device matrix array. *IEEE Transactions on Semiconductor Manufacturing, Vol. 17, Issue 2, pp. 155–165*, May 2004.

[44] Pete Sedcole and Peter Y. K. Cheung. Within-die delay variability in 90nm fpgas and beyond. *Proceedings of 16th IEEE International Conference on Field Programmable Technology (FPT) pp. 97–104*, Dec. 2006.

[45] Paul Cunningham Robert Mullins Simon Moore, Ross Anderson and George Taylor. Improving smart card security using self-timed circuits. *Proceedings of 8th International Symposium on IEEE Asynchronous Circuits and Systems, pp. 211–218*, Apr. 2002.

[46] Sergei P. Skorobogatov and Ross J. Anderson. Optical fault induction attacks. *Proceedings of 4th International Workshop on Cryptographic Hardware and Embedded Systems (CHES), LNCS Vol. 2523, pp. 2–12 Springer*, Aug. 2002.

[47] G. E. Suh and S. Devadas. Physical unclonable functions for device authentication and secret key generation. *Proceedings of 44th ACM/IEEE Design Automation Conference (DAC) pp. 9–14*, Jun. 2007.

[48] Noriaki Nakayama Takashi Sato, Hiroyuki Ueyama and Kazuya Masu. Determination of optimal polynomial regression function to decompose on-die systematic and random variations. *Proceedings of 13th IEEE/ACM Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 518–523*, Jan. 2008.

[49] J. Soelter G. Dror S. Devadas U. Ruhrmair, F. Sehnke and J. Schmidhuber. Modeling attacks on physical unclonable functions. *Proceedings of 17th ACM Computer and Communication Security Conference (CCS)*, Oct. 2010.

[50] Xiaoxiao Wang and Mohammad Tehranipoor. Novel physical unclonable function with process and environmental variations. *Design, Automation & Test in Europe (DATE)*, Mar. 2010.

[51] Wikipedia. Linear regression, polynomial regression, least squares. *Wikipedia*.

[52] Jinjun Xiong, V. Zolotov, and Lei He. Robust extraction of spatial correlation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(4):619–631, 2007.

[53] Leonid Reyzin Yevgeniy Dodis and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *Proceedings of IACR Eurocrypt 2004 International Conference, LNCS Vol. 3027, pp. 523–540, Springer*, May 2004.

[54] Chi-En Yin and Gang Qu. A regression-based entropy distiller for ro pufs. *Institute for Systems Research (ISR) Technical Reports (TR 2011-12), UMCP (DAC WIP 2012)*, 2012.

[55] Chi-En Yin and Gang Qu. Lisa: Maximizing ro puf's secret extraction. *Proceedings of 3rd IEEE International Workshop on Hardware Oriented Security and Trust (HOST)*, Jun. 2010.

[56] Meng-Day Yu and Srinivas Devadas. Secure and robust error correction for physical unclonable functions. *IEEE Journal of Design & Test Computers, Vol. 27, Issue 1*, Jan. 2010.