Title
Robust and Energy Efficient Hardware-Oriented Security for IoT Systems and Applications

Author
Gu, Hongxiang

Publication Date
2019

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA

Los Angeles

Robust and Energy Efficient Hardware-Oriented Security for IoT Systems and Applications

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Hongxiang Gu

2019

ABSTRACT OF THE DISSERTATION

Robust and Energy Efficient Hardware-Oriented Security
for IoT Systems and Applications

by

Hongxiang Gu
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 2019
Professor Miodrag Potkonjak, Chair

Internet of Things (IoT) is a revolutionary network that is envisioned to connect physical entities to the cyber world. IoT technology has fundamentally changed how we interact with our world. Worldwide spending on IoT is forecast to reach $745 Billion in 2019, and it is expected that investments in the technology will maintain double-digit growth rate for years to come.

Despite wide adoption and strong anticipation in the technology, two major obstacles heavily constrained the further development in IoT, respectively security and energy challenges. From the security perspective, the entire lifecycle of an IoT device could potentially be vulnerable to various types of attacks. Since many devices are deployed in an insecure environment, attackers could gain unauthorized access to the exposed hardware, which invalidates many security assumptions made in traditional security research. From the energy perspective, many IoT devices are incapable of affording traditional cryptographical protection due to low energy and computation budget. Energy efficiency is therefore crucial for designs and establishments of IoT.

To address IoT security problems, we explore and propose novel hardware-oriented security primitive designs and optimization techniques in this thesis. We first investigate the vulnerabilities of physically unclonable function (PUF), a popular low power hardware security primitive used in IoT devices, through the creation of a hardware emulation platform

using programmable delay lines (PDL). To address vulnerabilities in PUFs, we propose a novel security primitive, Interconnected PUF Network (IPN), that interconnects small segments of strong PUFs in a reconfigurable network, limiting the single-bit prediction accuracy to as low as 53.19% against a wide range of modeling attacks. We demonstrated that the interconnections in an IPN can be optimized to maximize output randomness and stability using our proposed evolution-strategies-based algorithm. Looking beyond PUF-based security, we designed content-driven injective functions (CRIF) that rearrange compositions of hardware injective functions based on previous messages, providing secure message encryption/decryption between IoT devices.

Facing the energy challenges, we propose "computing while racing" technique that reduces 40.4% of area overhead and 7.69% of power when implementing arbiter PUF and arbitrary logic on field-programmable-gate-arrays (FPGAs). This is achieved through encoding digital signals in analog forms and achieves a high percentage of hardware sharing, suggesting resource sharing could potentially be a promising direction for power/energy reduction in IoT devices.

Eventually, we propose two practical IoT applications. We first design a device anomaly detection utilizing the inconsistency in environmentally sensitive PUF challenge-response pairs. We show that our detector is more flexible and more power-efficient compared to state-of-the-art system monitors. Secondly, we demonstrate that our proposal of PUF-assisted group key management protocol securely protects IoT group communications while reducing global energy consumption by 47.3% compared to cryptographic key management solutions.

The dissertation of Hongxiang Gu is approved.

Milos D. Ercegovac

Jens Palsberg

Gregory Pottie

Miodrag Potkonjak, Committee Chair

University of California, Los Angeles

2019

To my mother

For your unconditional love

and my father

For your sacrifice for the family

TABLE OF CONTENTS

1 Introduction **1**

1.1 Objectives 2

1.1.1 Security Objectives 3

1.1.2 Energy Objective 3

1.1.3 Applicability Objective 4

1.2 Contributions and Organization 4

2 Stable PUF Emulation Platform Using Programmable Delay Lines . . . **8**

2.1 Motivation 8

2.2 Technical Goal and Contributions 9

2.3 Related Work 9

2.3.1 Physical Unclonable Function (PUF) 9

2.3.2 Programmable Delay Lines (PDL) 11

2.3.3 PUF Attacks 11

2.3.4 PUF Emulation 12

2.4 Preliminaries 12

2.4.1 PUF Model 12

2.4.2 PDL on FPGA 13

2.5 A Motivational Example 14

2.6 PUF Characterization 15

2.6.1 Creating Linear Equations 15

2.6.2 Improving Characterization Accuracy 17

2.7 PUF Emulation - PDL Evaluation 18

2.7.1 Delay Measurement Setup . . . 18
2.7.2 Delay Measurement Results . . . 19
2.7.3 Process Variation . . . 21
2.7.4 Stability . . . 21
2.8 PUF Emulation - Design . . . 27
2.8.1 Perfect Segmental Emulation . . . 27
2.8.2 Delay Difference Scaling . . . 28
2.8.3 Scaling Factor . . . 29
2.9 Emulation Improvement . . . 30
2.9.1 Two-Segment Emulation . . . 30
2.9.2 Output Voting . . . 30
2.10 Experimental Results . . . 31
2.10.1 Characterization Accuracy . . . 31
2.10.2 Baseline Emulation . . . 31
2.10.3 Improved Emulation . . . 33
2.10.4 Emulation Stability . . . 34
2.10.5 Latency Overhead . . . 35
2.11 Chapter Conclusion . . . 35

3 Securing PUFs with Interconnection and Reconfigurability . . . 36
3.1 Motivation . . . 36
3.2 Technical Goals and Contributions . . . 37
3.3 Related Work . . . 38
3.3.1 Modeling Attack . . . 38
3.4 Preliminaries . . . 39

3.4.1 Strong PUF Model . . . 39
3.4.2 IPN . . . 40
3.4.3 IPN Parameters . . . 43
3.5 Attacks models . . . 43
3.5.1 Assumptions . . . 44
3.5.2 Logistic Regressions . . . 44
3.5.3 Evolution Strategies . . . 45
3.5.4 Multilayer perceptron . . . 45
3.5.5 Other Machine Learning Algorithms . . . 46
3.6 Reconfiguration . . . 46
3.6.1 Reconfigure Timing . . . 47
3.6.2 Reconfiguration Logic . . . 48
3.6.3 Protecting Reconfiguration Logic . . . 49
3.7 Evaluation Results . . . 50
3.7.1 Logistic Regressions . . . 50
3.7.2 Evolution Strategies . . . 52
3.7.3 Multilayer perceptron . . . 53
3.7.4 Other Machine Learning Algorithms . . . 54
3.7.5 Implementations on FPGA . . . 55
3.7.6 Implementation Result . . . 57
3.8 Chapter Conclusion . . . 58

4 Optimizing PUFs with Evolution Strategies . . . 60

4.1 Motivation . . . 60
4.2 Technical Goals and Contributions . . . 61

4.3 Related Work . . . 62
4.3.1 PUF Randomness . . . 62
4.3.2 PUF Stability . . . 62
4.4 Preliminaries . . . 63
4.4.1 APUF Model . . . 63
4.4.2 IPN . . . 64
4.5 IPN Randomness . . . 64
4.6 IPN Stability . . . 66
4.7 IPN Optimization Algorithms . . . 66
4.7.1 Random Search . . . 67
4.7.2 Evolution Strategy . . . 68
4.8 Experimental Results on Randomness Improvement . . . 68
4.8.1 Number of Iterations . . . 70
4.8.2 IPN Node Size . . . 73
4.8.3 IPN Structure . . . 74
4.8.4 NIST Test Results . . . 76
4.9 Experimental Results on Stability Improvement . . . 77
4.10 Chapter Conclusion . . . 79

5 Content-driven Reconfigurable Injective Functions . . . 80

5.1 Motivation . . . 80
5.2 Technical Goals and Contributions . . . 80
5.3 Preliminary . . . 81
5.3.1 Injective Functions . . . 81
5.3.2 Properties of Injective Functions . . . 82

5.4 Related Work 82
5.4.1 PUF-based Security Primitives 82
5.4.2 Efficient Implementation of Classical Cryptography 83
5.5 Architecture 83
5.5.1 Overall Structure of CRIF 83
5.5.2 Injective Function 84
5.5.3 Content-driven Reconfiguration 86
5.5.4 Backward CRIF 87
5.6 Security Protocol 87
5.6.1 Assumptions 87
5.6.2 Protocol description 88
5.7 Security Analysis 88
5.7.1 Statistical Analysis 89
5.7.2 Statistical Modeling 92
5.8 Overhead Analysis 95
5.9 Chapter Conclusion 96

6 Hardware Sharing between PUF and Digital Logic 97

6.1 Motivation 97
6.2 Technical Goals and Contributions 98
6.3 Desiderata 99
6.4 Related Work 100
6.4.1 Hardware Random Number Generators 100
6.5 Preliminaries 100
6.5.1 PUF Model 100

6.5.2 Leap-Forward LFSR . . . 101
6.6 Architecture . . . 103
6.6.1 Observations . . . 103
6.6.2 Overall Design . . . 104
6.7 Implementation . . . 105
6.7.1 Implementation of Arbiter PUF . . . 105
6.7.2 Implementation of leap-forward LFSRs . . . 107
6.7.3 Post Process . . . 108
6.8 Experimental Results . . . 108
6.8.1 Area and power . . . 109
6.8.2 Randomness . . . 110
6.9 Chapter Conclusion . . . 111

7 Lightweight Environmental Anomaly Detection . . . 112
7.1 Motivation . . . 112
7.2 Technical Goals and Contributions . . . 113
7.3 Related Work . . . 114
7.3.1 System Monitor . . . 114
7.4 Preliminaries . . . 115
7.4.1 Unstable APUF . . . 115
7.5 CRP Environmental Sensitivity . . . 116
7.5.1 Environmental Variation . . . 116
7.5.2 Environmentally Sensitive CRP . . . 120
7.5.3 ESC Set Generation . . . 123
7.6 The Detection Framework . . . 125

7.6.1 System Design . . . 125
7.6.2 Experimental Results . . . 126
7.6.3 Area and Power . . . 127
7.7 Chapter Conclusion . . . 128

8 Efficient and Secure Group Key Management in IoT using Multistage Interconnected PUF . . . 129

8.1 Motivation . . . 129
8.2 Technical Goals and Contributions . . . 129
8.3 Related Work . . . 131
8.4 Multistage Interconnected PUF . . . 132
8.4.1 Processing Elements (PEs) . . . 132
8.4.2 Switching Elements (SEs) . . . 133
8.4.3 Multistage Interconnection . . . 133
8.4.4 Protecting Network Configuration . . . 134
8.4.5 Security Evaluation of MIPUF . . . 135
8.5 Group Key Management . . . 138
8.5.1 Key Distribution . . . 138
8.5.2 Key Storage . . . 140
8.5.3 Rekeying . . . 140
8.6 Evaluation . . . 142
8.6.1 Security Analysis . . . 142
8.6.2 Overhead Evaluation . . . 143
8.6.3 Implementation Results . . . 145
8.6.4 Impact on IoT Design . . . 145

8.7 Chapter Conclusions 146

9 Concluding Remarks 147

References 149

LIST OF FIGURES

2.1 The model of an n-bit arbiter PUF. 13

2.2 The internal structure of a 2-input LUT. 14

2.3 An example of using PDLs to emulate a PUF. 14

2.4 Delay characterization circuit. 19

2.5 Delay Measurement Results 20

2.6 Signal propagation delay of PDL vs. temperature of two PDL paths 22

2.7 PDL chain used for stability tests. 22

2.8 Delay characteristic variation under different temperature settings. 23

2.9 Delay ratio stability over 4 different temperature settings, VCCINT fixed at 1.2V. Red line: linear regression result. Green line: degree-2 quadratic regression result. 24

2.10 Delay characteristic variation under two VCCINT settings. 26

2.11 Delay ratio stability when increasing FPGA core voltage from 1.2V to 1.26V, operating temperature fixed to 25 °C. Red line shows linear regression result, green line shows degree-2 quadratic regression result. 26

2.12 Internal design of the ith emulation segment E^i. 28

2.13 Characterization results obtained from solving 10 sets of linear equations generated from random and stable CRPs. Boxes indicates 95% confidence interval. . . 32

3.1 An IPN node of size m and length n. If $m = n$, the node is homogenous, otherwise it is heterogenous. 40

3.2 A conceptual illustration of a crossbar switch like shuffler. 41

3.3 Encrypting random configuration logic using existing IPN nodes. K_j is the configuration vector encrypted by a chain of nodes from $node_1$ to $node_j$ 49

3.4 Best results in 100 logistic regression attacks using 30,000 CRP training set on five PUF-based systems. Error vs. iterations. 51

3.5 Evolution strategies attack result using 30,000 CRP training set on five PUF-based systems.. Error vs. iterations. 53

3.6 A 16-bit IPN network with four nodes on the first level, and has 4 levels. 57

4.1 An n-bit homogenous node. 64

4.2 An example of a simple IPN chain with shuffled connection between nodes. . . . 65

4.3 Cumulative percentage of 1,000 IPN instances that pass the NIST run test: random search vs. ES using various numbers of maximum iterations. IPN has a chain structure of two 32-bit homogenous nodes. Scoring function threshold: 99%, near-passing threshold: 90%, backtracking threshold set at 10% of maximum number of iteration. Results collected from 1,000 sets of IPNs. 71

4.4 Percentage of 1,000 IPN instances that pass the NIST run test. Maximum iterations are set to 10,50,100,500 and 1000. Each configuration collects 10,000-bit result. 72

4.5 Percentage of 1,000 IPN instances that pass the NIST run test with different node sizes. Each configuration collects 10,000-bit result. 73

4.6 Percentage of 1,000 IPN instances that pass the NIST run test with different IPN chain lengths. Each configuration collects 10,000-bit result. 74

4.7 Percentage of 1,000 IPN instances that pass the NIST run test with different IPN structure. Each configuration collects 10,000-bit result. 75

4.8 Stability of 1,000 IPN instances of IPN chains and complex IPNs. Each IPN node is a 32-bit homogenous node. Both the IPN chain and complex IPN has 4 levels. Complex IPNs uses a mixture of one-to-one, one-to-many and many-to-one connections. Environmental variance: 0.8∼1.0V, 0∼60 °C. 78

5.1 Overall architecture of our proposed CRIF. 84

5.2 A pair of four bit forward and backward injective function implemented using LUTs. 85

5.3 Statistical analysis on 64-bit CRIF implementations (depth = 4). 91

5.4 Hamming distance distribution for avalanche effect testing measured on a 64-bit CRIF implementations (depth = 4). 92

6.1 The model of arbiter PUFs with an n-bit challenge. 101

6.2 A 4-bit leap forward LFSR example. 102

6.3 High level illustration of proposed design. 105

6.4 The two output signals from LUT6_2 are required to be in phase, but can be inverted simultaneously. 106

6.5 Overall implementation using LUT6_2 on FPGA. 106

6.6 NIST Statistical Test Suite success ratio, one thousand 10,000 bit-streams are passed to each test. The test passes for p-value $\geq \sigma$, where σ is 0.05. The black line indicates a threshold of success ratio of 96%. All test results below this line are considered test failure. Arbiter PUF results without Von Neumann correction have success rate below 5% in most tests, thus are not shown in the figure. . . . 110

7.1 A snippet of core environment change vs. inconsistency of randomly selected APUF CRPs. The y-axis on the left is core temperature corresponding to the blue line, the y-axis on the right is the average CRP inconsistency corresponding to the red line. 118

7.2 VCCINT Power Module, when switch SW is open resistor R_{SW} is inserted into the power rail. 120

7.3 Core environment change leads to variations in inconsistencies of ESCRP. The y-axis on the left is core temperature corresponding to the blue line, the y-axis on the right is the average CRP inconsistency corresponding to the red line. . . 123

7.4 APUF-based anomaly detection framework on FPGA. 126

8.1 Network structure in Omega network style. r_{i-1}^{k} indicates the k-th response (output) bit of the $(i-1)$-th MIPUF node, c_i^k indicates the k-th challenge (input) bit of the i-th MIPUF node. 134

8.2 Inter-configuration and intra-configuration variation of a MIPUF with four nodes. Each node is implemented using 64 32-bit arbiter PUFs. The interconnection between nodes is designed in a blocking fashion as shown in Figure 8.1. 136

8.3 Simulated global energy consumption (J) vs. total number of IoT nodes under the settings introduced in [115]. 144

LIST OF TABLES

1.1 Organization of the dissertation chapters matched three major objectives. 5

2.1 Time required to generate 10,000 CRPs with a 64-bit PUF simulation using a HDL simulator and a 64-bit FPGA-based arbiter PUF implementation. 9

2.2 CRP distribution of a 64-bit arbiter PUF. 10,000 randomly selected challenges were fed into the PUF and the bit inversion rate is calculated over 1,000 measurement of responses. 18

2.3 Delay measurement results on three FPGAs (XC6SLX45). 21

2.4 Linear regression results on PDL temperature stability evaluation. 25

2.5 PUF prediction accuracy for software simulation and PDL-based hardware emulation using random CRP characterization data and stable CRP characterization data. The emulation accuracy is shown in the format of 95% confidence interval. 33

2.6 Overhead of Baseline and two-Segement PDL-based emulation of a 64-bit PUF. 33

2.7 PUF prediction accuracy for baseline emulation, two-segment emulation, and voting. All emulation accuracy results are shown in the format of 95% confidence interval. 33

2.8 Baseline emulation accuracy in different environments. Characterization done using 10 sets of stable CRPs. All emulation accuracy results are shown in the format of 95% confidence interval. 34

2.9 Average latency comparison in generating 10,000 CRPs using software HDL simulator, pure software simulation, and PDL-based emulation. 35

3.1 MLP parameters when modeling IPN. n is the depth of the network and m_i is the total number of PUF segments on the i-th level. 53

3.2 Deep neural network attack results. 54

3.3 Auto-sklearn modeling results on raw CRPs. 55

3.4 Best prediction accuracy on different PUF architectures using machine learning algorithms out of 100 runs. Each cell contains simulated/implementation result. 56

3.5 Prediction accuracy of three attack methods. Results collected from a 100,000 test set. Logistic regression: 14 days; evolution strategies: 250,000 generations, 14 days; MLP: 18 hours. 58

3.6 Area overhead for implementing a IPN shown in Figure 3.6. 58

4.1 Success ratio of passing a NIST test for 100 IPN instances with two 32-bit homogenous nodes. The maximum number of configurations allowed for each IPN is 100, each configuration collects 10,000-bit result. 65

4.2 Success ratio of passing NIST run test for 1,000 IPNs. Each configuration collects 10,000-bit result. 70

4.3 Percentage of 1,000 IPN instances that pass the NIST test suite. The IPN has two 32-bit homogenous nodes. Each configuration collects 10,000-bit result, the maximum number of iterations is set to 1,000. 77

5.1 Corresponding mappings for $f_{forward}$ and $f_{backward}$. 85

5.2 The average success ratio for the NIST statistical test suite. 100 bitstreams of 100,000 bits are passed to each test. The test passes for p-value$\geq \sigma$, where σ is 0.05. Asterisk sign (*) indicates the average case for all templates. 90

5.3 Single bit output prediction accuracy for different CRIF settings. Darker color indicates stronger resilience. 94

5.4 Single bit input prediction accuracy for different CRIF settings. Darker color indicates stronger resilience. 94

5.5 FPGA resource and power characteristics of the hardware support for CRIF-based encryption/decryption scheme. Power measured for scheme running at 5 Mhz. 95

5.6 Power comparison with other power efficient AES design on FPGA. Power measured for scheme running at 5 Mhz. 96

6.1 INIT value rules for implementing XOR gates. 108

6.2 FPGA resource and power characteristics: our design (four 64-bit leap forward LFSRs loaded on eight 64-bit arbiter PUFs) vs. four standalone 64-bit leap forward LFSRs vs. 256 64-bit arbiter PUFs vs. four 64-bit leap forward LFSR and eight 64-bit arbiter PUFs that do not share hardware resources. Power per bit unit: mW/bit. 109

7.1 FPGA resource and power characteristics: Virtex-5 SYSMON vs. our design on Virtex-5 vs. RO-based on-chip power monitor vs. our design on Spartan-6. ADC and analog sensor area and power are not included in calculation. 127

7.2 anomaly detection: true positive rate (%)/false positive rate(%) using 1,000 random challenges vs. single ESC vs. 1,000 ESCs as challenge set per sample. APUFs are placed immediately besides the monitored circuit. A mixture of 1,000 abnormal temperature and VCCINT were applied to the protected circuit. . . . 127

8.1 Best single-bit prediction accuracy on different PUF architectures using logistic regression (LR), evolution strategies (ES) and deep learning (DL) attacks out of 100 runs. Each attack uses 100,000 CRPs. Total number of arbiter PUF segments used in all architectures are fixed to 1,024. 138

8.2 FPGA resource and power characteristics of the hardware support of our proposed key management scheme implemented on Spartan-6 FPGA. 145

8.3 Performance and energy comparison of different wireless standards in IoT. . . . 146

ACKNOWLEDGMENTS

First and foremost, I would like to express my sincere gratitude to my advisor Professor Miodrag Potkonjak. It has been my highest honor to be his Ph.D. student. He has taught me not only the skills and knowledge to research in the field of computer science but also the mentality to be a hungry and forever curious researcher. I appreciate all his contribution of time, energy and ideas to make my Ph.D. study meaningful, productive and enjoyable. He stands as a role model to me as a dedicated researcher and a brave fighter against all the odds. I am forever thankful for his unconditional support in both my study and life.

Besides my advisor, I would like to thank the rest of my thesis committee: Professor Miloš Ercegovac, Professor Jens Palsberg, and Professor Gregory J. Pottie for their insightful comments and motivating encouragement. Their visionary suggestion and questions on my prospectus, oral examination, and final dissertation helped me gain a better understanding of my research from different perspectives.

I would also like to thank Dr. Sheng Wei and Dr. Viswanathan (Vishy) Swaminathan for providing me the opportunity as a research scientist intern at Adobe Research. Their guidance has widened my research beyond pure institutional academic research. The training I received from them made me a more efficient, productive and practical researcher.

Thank you to my colleagues and collaborators Teng Xu, Jia Guo, and Jason Zheng for discussing ideas with me, for helping me practice my talks and dissertations, for encouraging me to stay strong when I was struggling. Dr. Teng Xu spent a countless number of hours working on Recursive Inverse Functions, Hardware sharing in FPGA-based PUF implementation, PUF matching, and PUF-based anomaly detector projects with me. Jia Guo provided precious suggestions for performing machine learning attacks on existing PUF systems. FPGA validation of many circuit designs of mine is based on Jason Zheng's previous implementation of arbiter PUFs.

Last but not least, I could never finish my Ph.D. study without the support of my family and friends. My dearest parents are my greatest inspiration, motivation, and support. I will forever love them, and I can always feel their unconditional love under any circumstances.

And, of course, to Tiancheng Fang, who is always by my side, during my darkest times and my proudest moments of life.

Thank you, thank you all.

VITA

2010–2014 B.S. (Computer Science), University of California, Los Angeles.

2014–2018 M.S. (Computer Science), University of California, Los Angeles.

PUBLICATIONS

T. Xu, H. Gu, M. Potkonjak, "Data Protection Using Recursive Inverse Function," International Conference on Field Programmable Logic and Applications (FPL), 2015.

H. Gu, T. Xu, M. Potkonjak, "An Energy-Efficient PUF Design: Computing While Racing," ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED), pp. 142-147, 2016.

T. Xu, H. Gu, M. Potkonjak, "An Ultra-Low Energy PUF Matching Security Platform Using Programmable Delay Lines," International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2016.

H. Gu, T. Xu, M. Potkonjak, "A low-power APUF-based environmental abnormality detection framework," ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED), 2017.

H. Gu, M. Potkonjak. "Securing Interconnected PUF Network with Reconfigurability," IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), pp. 231-234, 2018.

H. Gu, V. Swaminathan. "From Thumbnails to Summaries - A Single Deep Neural Network to Rule Them All," IEEE International Conference on Multimedia and Expo (ICME), 2018.

J. Guo, H. Gu, M. Potkonjak. "Efficient Image Sensor Subsampling for DNN-Based Image Classification," ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED), 2018.

H. Gu, M. Potkonjak. "Efficient and Secure Group Key Management in IoT using Multistage Interconnected PUF,". ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED), pp.8-14, 2018.

CHAPTER 1

Introduction

After decades of development, the Internet has connected countless computers in a global network system, carrying a vast range of information resources and services. Researchers and engineers in recent years have looked beyond the Internet and came to realize that our physical world is an information system by itself. Much information can be collected and utilized using sensors and actuators. The idea of proliferating physical entities in a new network creates the concept of the Internet of Things (IoT) [1].

The growth in globally connected devices is now mainly driven by IoT devices on both the consumer side and the enterprise side. By the end of the second quarter of 2018, the total number of IoT devices globally has reached 7 billion, and this number is expected to exceed 10 billion by 2020 and 22 billion by 2025 [2]. Globally, over $745 Billion were spent in 2019, and it is expected that the double-digit growth rate would be maintained for years to come [3]. IoT has been envisioned to be a revolutionary network that connects physical devices around us to perform intelligent tasks such as monitoring, communication, operation, and optimization.

Despite the ambitious master plan behind the idea of IoT as well as its rapidly growing speed, security challenges have always been a major concern in the process of further development of the technology. The IoT security problems are particularly challenging due to two seemingly irreconcilable objectives.

On the one hand, IoT requires a maximum level of security. IoT technology has enabled a broad spectrum of applications in a variety of environments. From smart appliances in homes to Industry 4.0 in factories, to intelligent clinical management in hospitals and smart city systems on city streets. In almost all IoT related applications, sensitive data of large

scale are collected, stored and transferred through IoT networks. In case of a security breach, whether it is invasive, semi-invasive or non-invasive [4], in an IoT system, adversaries could potentially gain private information such as identity, health-related records, business secrets, and even military records. The compromise of IoT devices that are associated with cyber-physical systems could be even more dangerous or even lethal in some cases. For example, security researchers Billy Rios and Jonathan Butts have shown that it is possible to hack into pacemakers and control the electrical impulses that are sent to the heart to regulate the patient's heartbeat [5].

On the other hand, security always comes with a price; it takes energy and computational power to detect and defend against adversarial actions. Many IoT devices have highly constrained energy and computational budget due to harsh requirements on device size, weight, and portability, leaving little resource to comply with the high-security requirement. Adopting conventional security approaches in IoT could consequently lead to low performance and high maintenance cost.

In this thesis, we focus on the problem of designing and improving hardware-oriented security subsystems in IoT to perform a variety of crucial security tasks under a highly constrained environment. In defiance of the seemingly contradictory requirements, we show that it is feasible to provide the highest possible level of security under the lowest energy and computational power budget using our proposed methods.

1.1 Objectives

We believe this thesis is an adventurous exploration in three strongly correlated directions. We intend to make our contributions in meeting three objectives, respectively from the perspectives of security, energy, and applicability.

1.1.1 Security Objectives

IoT is a complex system that consists of several layers of abstraction spanning in multiple dimensions. From the physical layer of hardware sensors and controllers to the application layer that utilizes the collected IoT data for specific tasks, security vulnerabilities could be exploited anywhere, resulting in catastrophic compromise of the entire system. Majority of the current attacks on IoT systems are software attacks. Software attacks on IoT are popular because many similar security vulnerabilities are shared with other systems. In most cases, these exploited vulnerabilities are general software security problems instead of IoT specific issues. In recent years, the rise of IoT regime creates new attack surfaces as well as new security challenges at the hardware level.

Hardware security is a natural starting point for research in IoT systems as hardware is the basis of all IoT devices. Protecting IoT systems at hardware level grants system designers great flexibility to minimize energy and area cost. Security at the hardware level also provides a solution to the problem that classical cryptography could not resolve, presenting a new level of protection against emerging attack methods such as side-channel attacks [6] and physical attacks.

The security objective we claim to meet is to examine vulnerabilities in hardware security subsystems and propose robust and secure protection to these subsystems.

1.1.2 Energy Objective

A large portion of IoT devices are compact and powered by batteries due to the high requirement of mobility. Therefore the energy budget for security subsystems is minimum if not none. Devices such as smartphones, tablets, and smartwatches require daily recharging. The high energy consumption in these devices is a major obstacle in terms of adopting advanced security modules to the system. Devices such as wireless sensors have much lower energy consumption but usually equipped with an ultra lightweight power source. These low power devices are often massively deployed over a broad area; recharging or replacing the power source is extremely costly. To reduce the cost and prolong the lifespan of each device, most

device manufacturers leave no room for security subsystems.

All devices are in great need for energy efficient security subsystems to defend against evolving attack techniques. In this thesis, we focus on both creating techniques that could boost energy efficiency in existing security primitive as well as designing novel and ultra lightweight security solutions in IoT devices and IoT applications.

1.1.3 Applicability Objective

IoT system is playing an increasingly irreplaceable role in countless numbers of applications from home automation to city infrastructure management. Newer and more creative applications that could utilize IoT technology is booming at an enormous speed. Our applicability objective is to explore how we could use our efficient and robust hardware-oriented security solutions to protect and secure some of the emerging IoT-based applications.

1.2 Contributions and Organization

The dissertation is organized into three major parts, where each part explains our contribution to meet security, energy and applicability objectives. Note that even though each part focuses on its theme, the underlying logic beneath is coherent and consistent: we aim to design energy efficient hardware-oriented security subsystem for IoT and its applications. We analyze energy efficiency when introducing our proposed security techniques while we also investigate the security impact on our energy efficiency improvement techniques. The detailed organization can be viewed in Table 1.1.

The first part of the dissertation discusses low power hardware security primitives in IoT devices. We investigate security properties in hardware security primitives using two different approaches, respectively improving existing designs and creating novel architectures. We first focus on security primitives based on a promising technology - PUFs. PUF is a category of hardware functions that utilize the inevitable process variation introduced during manufacturing to create unique mappings between inputs and outputs. Recent research on

<table>
<tr><th colspan="2">Security Objective</th></tr>
<tr><td>Optimization of PUFs</td><td>Non-PUF-based Primitives</td></tr>
<tr><td>• Stable PUF Emulation Platform Using Programmable Delay Lines (Chapter 2)
• Securing PUFs with Interconnection and Reconfigurability (Chapter 3)
• Optimizing PUFs with Evolution Strategies (Chapter 4)</td><td>• Content-driven Reconfigurable Injective Functions (Chapter 5)</td></tr>
<tr><th colspan="2">Energy Objective</th></tr>
<tr><td colspan="2">Hardware Sharing between PUF and Digital Logic (Chapter 6)</td></tr>
<tr><th colspan="2">Applicability Objective</th></tr>
<tr><td colspan="2">Lightweight Environmental Anomaly Detection (Chapter 7)
PUF-based Group Key Management (Chapter 8)</td></tr>
</table>

Table 1.1: Organization of the dissertation chapters matched three major objectives.

PUFs indicates that PUFs mainly suffers from stability issues and vulnerabilities against modeling attacks. To exploit vulnerability issues and stability weakness in some PUF designs, we first propose to utilize programmable delay lines (PDL) technology to create a platform that emulates stable arbiter PUFs with matching accuracy of 87.42% in Chapter 2. We propose the design of interconnected PUF Network (IPN) that connects multiple small PUFs in a network in Chapter 3 as an attempt to defend PUFs against modeling attacks. The interconnections in IPN can be reconfigured from time to time to completely remap the input-output pairs with low latency and low energy. We show that reconfigurable IPN is capable of preventing adversaries from collecting sufficient training data to apply modeling attacks and therefore providing robust security to protected hardware. The maximum single-bit prediction accuracy is less than 53.19% against a wide range of modeling attacks. We address the stability issues in IPN by proposing a novel mechanism that uses evolution strategies (ES) to find the best combination of PUF segments in Chapter 4, improving randomness and stability by 220.8% and 22.62% compared to unoptimized configuration. Taking a step forward, we also applying similar reconfiguration ideas to non-PUF-based security primitives.

We eventually propose Content-driven Reconfigurable Injective Function (CRIF) to conduct lightweight encryption/decryption tasks in IoT devices and communication between them in Chapter 5. We demonstrate that CRIF is an excellent alternative to cryptography-based and PUF-based competitors due to stability, efficiency, and reconfigurability, especially on IoT devices where computational power and energy are constrained. We measured CRIF achieves at least 75.04% of power savings compared to popular AES-based schemes.

The second part of the thesis is reducing energy cost in existing security primitives. We propose a novel technique called hardware "free riding" to significantly reduce area and energy overhead of analog-based PUFs. While PUFs are low power and low energy on application specific integrated circuits (ASIC), on reconfigurable hardware platforms such as field-programmable-gate-arrays (FPGA), implementation can be costly. In Chapter 6, we address this problem by introducing a technique that allows arbiter PUFs to be implemented together with any arbitrary logic on the same piece of hardware on FPGA. The evaluation shows our technique reduces 40.4% of area overhead and 7.69% of power consumption when implementing 128-bit arbiter PUFs and eight 32-bit linear-feedback shift registers (LFSR) on FPGAs.

Lastly, in the third part, we propose two applications that could greatly benefit from our proposed designs and techniques introduced in the first two parts. We first present an ultra lightweight onboard anomaly detection mechanism that has excellent potential to accurately detect suspicious voltage and temperature changes in Chapter 7. By actively monitoring stability variations of onboard analog PUFs, our design could perform flexible, fine-grained chip monitoring service while reducing 63% of area and 13% of power compared to sensor-based Xilinx System Monitor. Lastly, we investigated the well-known problem of key management in IoT systems in Chapter 8. We propose to use multistage interconnected PUF (MIPUF) to assist the protection of key management system at both software and hardware level. Our experimental result indicates that our design provides physical protection when compared to Elliptic-Curve Cryptography (ECC) based solutions and reduces global energy consumption by 47.33%.

We believe this thesis provides inspirations, insights, and tools for IoT device designers

and application developers who seek to provide sufficient protection in heavily constrained environments.

CHAPTER 2

Stable PUF Emulation Platform Using Programmable Delay Lines

2.1 Motivation

It has been widely acknowledged that physically unclonable functions (PUFs) as a type of hardware security primitive have great potentials to be used in many applications. A PUF is a device implementing a one-way function which takes advantage of process variation to guarantee the property uniqueness of each piece.

Modern research has shown that some PUF designs are not as perfect as what they seem to be. Many PUFs can be modeled using software or hardware approaches based on characterization results. This process is generally noted as PUF emulation. Though PUFs can be emulated, most state-of-the-art emulation techniques are both inefficient and inaccurate due to complication in PUF architecture and unstable nature of analog systems.

Software simulations with high precision are generally slow and power-hungry, especially for complicated PUF designs. Table 2.1 shows the time required to generate 10,000 CRPs with a 64-bit arbiter PUF and a hardware description language (HDL) simulator. A software simulation of PUF needs significantly more time to produce a response compared to the use of the PUF implemented on a piece of hardware. Physically matching a PUF to a target PUF is an alternative technique. Many studies have taken place in attempting to match two PUFs using device aging technology [7], which require fine-grained control in laboratories. Since aging is a unidirectional process, both overage and underage could lead to low precision in the emulation. Thus, a highly stable operational environment is required.

Type	Latency (s)
HDL simulator	98.38
Hardware PUF	0.001

Table 2.1: Time required to generate 10,000 CRPs with a 64-bit PUF simulation using a HDL simulator and a 64-bit FPGA-based arbiter PUF implementation.

2.2 Technical Goal and Contributions

Our technical goal is to create a new emulation platform to resolve the above issues in standard PUF emulation techniques. The key idea is to emulate an existing PUF using hardware to achieve low latency overhead as well as resilience against operational variations. We achieve our goal by using the look-up table (LUT) based PDLs. We believe the highly stable and low latency PDL serves as an ideal building block as our PUF emulation platform. The PDL is used as a clone that shares similar delay characteristics of each segment in the standard PUFs. By accurately emulating every single segment of the target PUF, the whole emulation is capable of predicting the corresponding responses of the target PUF when provided with an arbitrary challenge.

In this chapter, we first review the related literature on PUFs and PDLs. Then we give the preliminaries of the basic PUF model we use as well as the design of PDL on FPGA. We also provide a motivational example of our PUF emulation scheme. We then propose a high precision PUF characterization mechanism that enables a PUF emulation platform. Later we demonstrate in detail our PUF emulation platform in the subsequent sections. Eventually, we provide our analysis of the reliability and accuracy of our proposed design by presenting experimental results on Spartan-6 XC6SLX45 platforms.

2.3 Related Work

2.3.1 Physical Unclonable Function (PUF)

PUF was first proposed by Pappu et al. using mesoscopic optical systems [8]. Gassend et al. developed the first silicon PUFs through the use of intrinsic process variation in deep

submicron integrated circuits [9]. A variety of other types of PUFs have since been proposed, including arbiter PUFs [9], ring oscillator PUFs [10], SRAM PUFs [11], and butterfly PUFs [12]. Xu et al. has also propose to digitalize PUFs and create digital PUFs [13]. Another popular robust PUF design is proposed by Maiti et al. with a selected PUF challenge-response set [14]. More recently, PUF designs have been focusing on improving randomness and learning-resilience, including [15] [16] [17] [18] [19]. Nozaki et al. even designs more side-channel attack resilient PUFs using statistical tests [20].

Numerous traditional protocols can be interpreted using PUFs, ranging from the traditional security key communication and authentication [21] to more sophisticated public key communication [22] with the vital idea of employing the high unpredictability of PUF responses to secure the information. More recently, efforts have been made to enhance the security and randomness of PUFs. Devadas et al. have proposed to use a syndrome coding scheme to reduce the amount of information leakage caused by the traditional PUF key generation system [23]. At the system level, Zheng proposed to use PUFs as an instruction authentication tool for embedded systems [24].

PUFs have also been well studied in many novel hardware security applications. Devadas et al. proposed to use PUFs in RFID for anti-counterfeiting applications[25]. Gu et al. propose several low power applications based on PUF including computing-while-racing PUF [26] and PUF-based system anomaly detector [27]. Zhang et al. proposed a PUF-FSM binding scheme for IP-protection [28]. Xu et al. proposed an ultra-low energy PUF matching scheme using programmable delay lines [29] and device aging [30]. Gao et al. proposed to use SRAM PUF for key generations [31], Huang et al. proposed a PUF-based identity verification [32]. Gope et al. also applied PUF to RFID authentication protocols [33]. Tajik et al. designed a system monitor using PUFs [34]. Aman et al. embedded PUFs in IoT systems for mutual authentication purposes [35]. PUFs can also be used to construct Recursive Inverse Functions(RIF) that provide fast and ultra-low energy encryption and decryption for data protection [36].

2.3.2 Programmable Delay Lines (PDL)

Programmable delay lines are a series of digital delay lines with electrically programmable and trimmable delay times [37]. Taking advantage of internal structures of LUTs, Majzoobi et al. proposed to implement PDLs through creating and controlling delay biases on FPGAs [38]. PDLs are used in many different applications, including bus timing adjustment [39], programmable pulse generator at high resolution [40] and metastability characterization on FPGA [41]. Since PDLs are usually controlled at *ps*-level, accurate delay measurements is required. Tsai et al. proposed vernier delay line-based built-in delay measurement circuits with a small area overhead and can provide high-resolution delay measurement [42]. Raychowdhury et al. proposed on-chip delay estimation of segment path delays in [43]. Majzoobi et al. designed a delay characterization circuit with clock synthesis that can measure delays at picosecond resolution on FPGAs through probabilistic estimation[44].

2.3.3 PUF Attacks

Even though unclonability and unpredictability are the main characteristics of PUFs, previous work has shown that PUFs are vulnerable to three types of attacks: modeling attacks, physical attacks, and side-channel attacks.

Modeling attacks are a commonly adopted approach for PUF characterization. Ruhrmair et al. proved that arbiter PUFs are weak against machine learning attacks [45]. They further analyzed the PUFs in the context of security protocols in [46]. Ganji et al. proved that ring oscillator PUFs could be completely learned using a Probably Approximately Correct (PAC) learning framework [47].

Physical attacks correspond to gate and transistor level characterization (GLC) where delay, leakage, or some other device metrics are analyzed [48]. Tajik et al. proposed a photonic emission analysis mechanism to characterize an arbiter PUF with extremely high precision [49].

Side-channel attacks have also been studied in breaking existing PUF designs. Mahmoud et al. proposed to combine modeling attacks with power side-channel attacks to better

characterize PUFs [50].

2.3.4 PUF Emulation

Many attacks on PUF designs have enabled a large number of studies in emulating a PUF device. Software simulation is one of the most popular methods to predict responses based on a given challenge. All research mentioned in Section 2.3.3 adopts software simulation to evaluate characterization accuracy. However, software simulation suffers from high latency and high energy consumption, thus not capable of providing a lightweight real-time emulation of a target PUF.

Another way of emulating a target PUF is through PUF matching (creating a physical clone of a PUF). Helfmeier et al. proposed a mechanism to produce a physical clone of an existing SRAM PUF using Focused Ion Beam circuit edit [51]. Meguerdichian et al. proposed a method to match a PUF to another PUF using device aging techniques [7]. Creating a physical clone of a PUF using the above methods requires laboratory equipment and environment, thus very expensive and difficult to achieve.

2.4 Preliminaries

2.4.1 PUF Model

The PUFs we intend to emulate are standard arbiter PUFs. Figure 2.1 shows the schematic diagram of the PUF model. The basic structure of an n-bit PUF consists of n delay segments. The four propagation delays in the ith segment are denoted as d^i_{AC}, d^i_{BD}, d^i_{AD} and d^i_{BC} respectively. d^i_{AC} and d^i_{BD} are considered a delay pair, and d^i_{AD} and d^i_{BC} are considered as another delay pair. The delays within each delay pair are designed to be nominally equal to each other. After manufacturing, however, process variation causes unpredictable delay difference within each pair. When built on an FPGA, each delay pair in each segment is directly implemented using LUTs with the same size. Two identically designed overall paths are generated by connecting delay components in each segment in a chain, and an arbiter is

appended at the end of the two paths. The two paths can be modified using the control bit of each segment. For example, in Figure 2.1, if the control bit of the ith segment is 0, then d^i_{AC} is appended to the upper segmental path and d^i_{BD} appended to the lower segmental path. If the control bit is 1, d^i_{BC} is appended to the upper segmental path and d^i_{AD} is appended to the lower segmental path. The control bit decides which two delays inside a PUF segment is appended to the PUF paths.

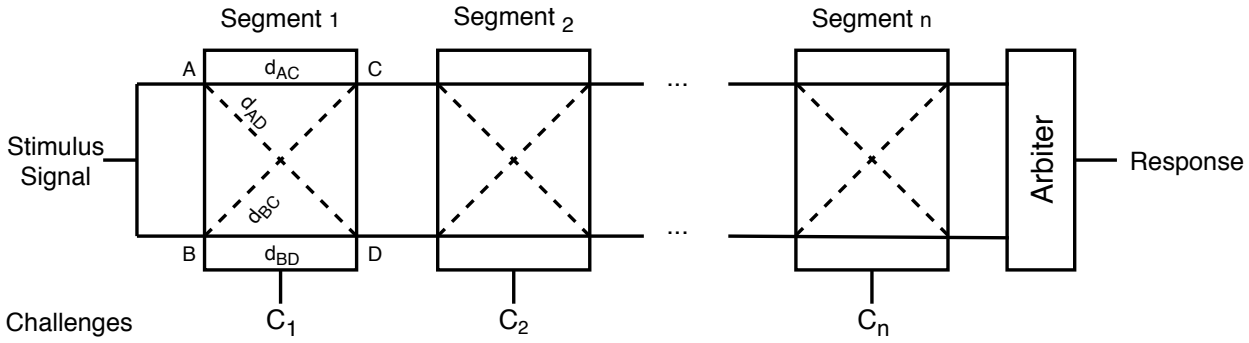


Figure 2.1: The model of an n-bit arbiter PUF.

The vector consisting of all control bits is denoted as the PUF challenge. When an n-bit challenge $(c_1c_2 \ldots c_{n-1}c_n)$ is provided to the PUF, two nominally identical paths are thus configured. To retrieve a response, an impulse signal is fed into the system to excite both paths simultaneously. Because of process variation, the signal traveling along one of the two paths reaches the arbiter earlier, generating a corresponding arbiter output denoted as the PUF response. For an n-bit PUF, there exist 2^n challenge-response pairs.

2.4.2 PDL on FPGA

Majzoobi et al. [38] proposed a general design of PDL on FPGA platforms implemented using a single LUT-2. Figure 2.2 shows an example PDL implemented with LUT-2 with two selection bits S_0 and S_1. The propagation delay from A_0 to O_i when $A_0 = 0$ is displayed in the blue line, and the same delay when $S_0 = 1$ is shown in the red line. The path represented in the red line seems longer than the path marked in blue according to the figure, representing the propagation delay from S_0 to O_i is longer when $S_0 = 1$. This result indicates that different input value, although completely digital, could result in a slight difference in the

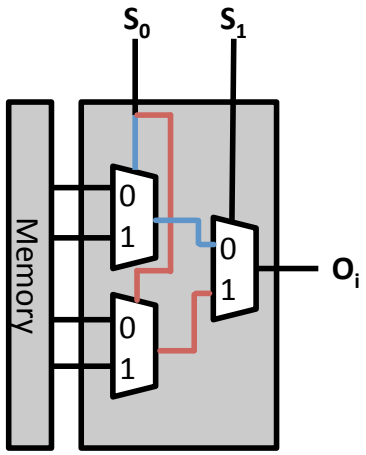


Figure 2.2: The internal structure of a 2-input LUT.

analog propagation delays. We utilize the design of the slightest asymmetry in PDL design to emulate delay-based PUFs.

2.5 A Motivational Example

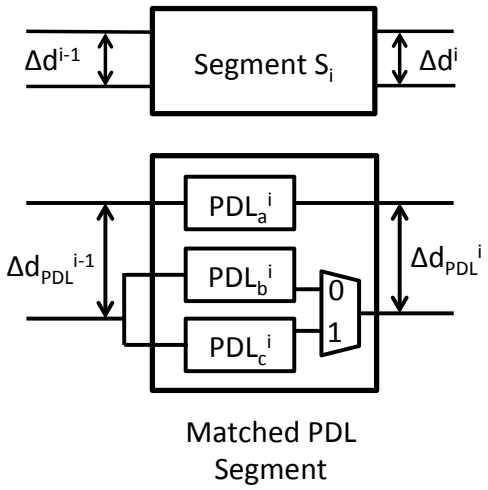


Figure 2.3: An example of using PDLs to emulate a PUF.

Now we illustrate our PDL-based PUF emulation platform using a motivational example in Figure 2.3. We assume the PUF to be emulated the same structure as we explained in the Preliminaries section, where each PUF segment uses a user-provided control bit to configure the competing paths within the segment. For example, if the control bit of segment i is

zero then the delay difference between the upper and lower path $\Delta d^i = d^i_{AC} - d^i_{BD}$; if the control bit of segment i is one then the delay difference between the upper and lower path would instead be $\Delta d^i = d^i_{BC} - d^i_{AD}$. To clone segment i in the target PUF, we create three PDLs to form an emulation segment. We carefully choose the selection bits for each PDL so that the three PDLs (PDL^i_a, PDL^i_b, PDL^i_c) are configured to obtain delays d^i_A, d^i_B and d^i_C respectively. Similar to segment i in the target PUF, the ith emulation segment also takes a control bit to select which two PDL delays are appended to the path. If the control bit is 0, PDL^i_a and PDL^i_b are selected, producing $d^i_{PDL} = d^i_a - d^i_b \approx \Delta d^i = d^i_{AC} - d^i_{BD}$ as output delay difference; if the control bit is 1 PDL^i_a and PDL^i_c are selected, producing $d^i_{PDL} = d^i_a - d^i_c \approx \Delta d^i = d^i_{BC} - d^i_{AD}$ in output delay difference. The same emulation process is thus repeated for all segments of the target PUF. To summarize, the basic idea of emulating an n-bit PUF is to configure $3n$ PDLs where the delays of each PDL are configured in such a way that all delay differences produced by each PUF segment can be reproduced using PDLs.

2.6 PUF Characterization

It is well known that standard arbiter PUFs can be characterized using statistical methods [52]. In this section, we first propose a linear model for an arbiter PUF design. We claim that by solving a set of linear equations constructed by measuring PUF delay differences, we can characterize a target PUF of our choice with low overhead. The resulting model can accurately retrieve the delay differences in each PUF segment.

2.6.1 Creating Linear Equations

Assume that the characterization target is an n-bit arbiter PUF with n segments. We use the same notation as we declared in Figure 2.2. The ith PUF segment has 4 different delays. The four delays are respectively d^i_{AC}, d^i_{AD}, d^i_{BC} and d^i_{BD}. The control bit of the segment decides which pair of delays is appended to the segmental paths shown in Equation 2.1.

$$\text{Delay(Path 1), Delay(Path 2)} = \begin{cases} d^i_{AC}, d^i_{BD} & c_i = 0 \\ d^i_{BC}, d^i_{AD} & c_i = 1 \end{cases} \tag{2.1}$$

The two paths generate one of the two possible delay differences depending on the challenge bit c_i. For clarity, we denote the delay difference of the ith PUF segment to be Δd^i, defined in Equation 2.2.

$$\Delta d^i = \begin{cases} d^i_{AC} - d^i_{BD} & c_i = 0 \\ d^i_{BC} - d^i_{AD} & c_i = 1 \end{cases} \tag{2.2}$$

The value of Δd^i can be either positive or negative. For example, when $c_i = 0$, a positive Δd^i indicates that $d^i_{AC} > d^i_{BD}$ and a negative Δd^i indicates that $d^i_{AC} < d^i_{BD}$.

We observe that the delay difference Δd between the two PUF paths is merely the sum of all segmental delay differences in the PUF. Based on this observation, we can create linear equations if the delay difference between a pair of competing PUF paths is measured. Solving the equations would thus provide accurate delay difference characterization for each PUF segment.

Based on Equation 2.2, we create two variables representing the two possible delay differences in each PUF segment $\Delta d^i_0 = d^i_{AC} - d^i_{BD}$ and $\Delta d^i_1 = d^i_{BC} - d^i_{AD}$. For a 64-bit PUF, there are in total of 128 unknown variables. We also generate 10,000 random challenge vectors and measure the final delay differences between the two nominally identical PUF paths. The delay differences are measured using a pico-second accurate delay characterization circuit.

For N challenges, we are capable of constructing N linear equations with m unknown variables. The jth linear equation is constructed from the jth challenge vector $c^j = \{c^j_1 c^j_2 \dots c^j_m\}$. An example of a set of 10,000 linear equations on a 64-bit arbiter PUF is shown below.

$$\Delta d^1_{k^1_1} + \Delta d^2_{k^1_2} + \ldots + \Delta d^n_{k^1_m} = \Delta d^{measure}_1$$
$$\Delta d^1_{k^2_1} + \Delta d^2_{k^2_2} + \ldots + \Delta d^n_{k^2_m} = \Delta d^{measure}_2$$
$$\ldots$$
$$\Delta d^1_{k^N_1} + \Delta d^2_{k^N_2} + \ldots + \Delta d^n_{k^N_m} = \Delta d^{measure}_N$$
$$N = 10,000, m = 64$$
$$k^j_i = \begin{cases} 0 & c^j_i = 0 \\ 1 & c^j_i = 1 \end{cases}$$

We split the 10,000 equations into 10 sets and apply a linear equation solver to find the least square solutions to solve for a close approximate of delay differences in each segment for each set. We take the average solution for each variable to be the delay difference value.

2.6.2 Improving Characterization Accuracy

PUF as a security primitive has suffered from stability issues. An unstable CRP might alter the corresponding linear equation and eventually lead to a large error rate in the delay characterization result. A stable CRP, on the other hand, is less likely to modify the linear equation, making the solutions more consistent. To improve characterization accuracy, we propose to create linear equations based on only stable CRPs.

Table 2.2 shows the CRP distribution of a 64-bit arbiter PUF. Overall, 87.94% of the responses are stable when provided with the same challenge. We define stable CRPs as CRPs that inverts its response with probability less than 10% when providing the same challenge 1,000 times. 63.5% of 10,000 challenges provides stable responses. By creating linear equations based on only stable CRPs, we believe the characterization accuracy can be improved.

Type	Bit inversion rate range	Frequency
Stable	$\leq 10\%$	63.5%
Mostly Stable	10% - 30%	32.2%
Unpredictable	bit inversion $> 30\%$	5.3%

Table 2.2: CRP distribution of a 64-bit arbiter PUF. 10,000 randomly selected challenges were fed into the PUF and the bit inversion rate is calculated over 1,000 measurement of responses.

2.7 PUF Emulation - PDL Evaluation

In this section, we evaluate the delay characteristics, process variation effect, and stability properties of PDL to examine the feasibility of using PDL to emulate an arbiter PUF.

2.7.1 Delay Measurement Setup

To measure and verify the delay of PDL on the FPGA we use the circuit describe by Majzoobi et al. [38]. The delay characterization circuit is shown in Figure 2.4. We assumes the clock-to-Q delay at the launch FF is t_{clk2Q}, the clock skew between the launch and sample flip-flops (FFs) is t_{skew}, the clock pulse width is denoted as T and the time that a signal propagate through Circuit Under Test (CUT) and reach the sample FF from the moment the launch FF is clocked is denoted as $t_p = t_{CUT} + t_{clk2Q} - t_{skew}$. Noted in our experiment the CUT is essentially PDL segments connected in chains using the configurations to adjust its delay characteristics.

The pulse generator sweeps through different frequencies and calculates the approximate delay from the frequency of the clock signal that causes the timing error probability to be 50%. The measurement is valid because as we sweep the frequency of the function generated from the pulse generator T and makes it approach t_p, the sample FF enters a metastable state because of the setup and hold time violations, and its output becomes nondeterministic. The probability that the metastable state resolves to a 0 or 1 is a function of how close T is to t_p. The metastable state resolves to a 1 with a probability of 0.5 indicate that $T = t_{CUT}$. Through careful adjustment of the pulse frequency at high resolution, the circuit could achieve pico-second resolution in delay measurements.

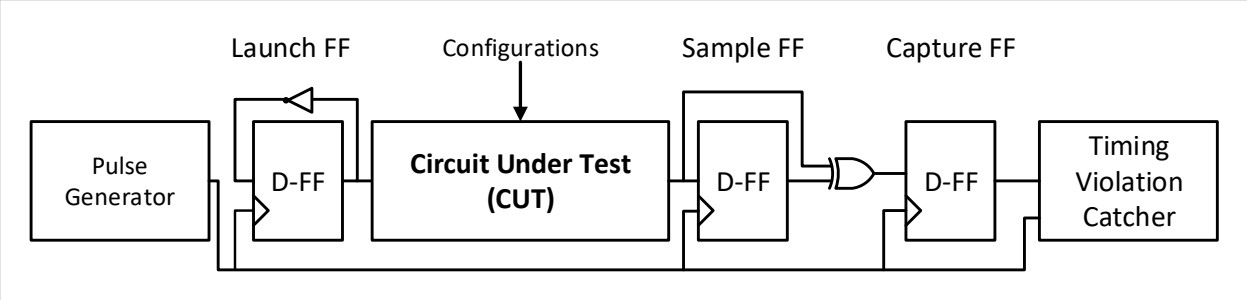


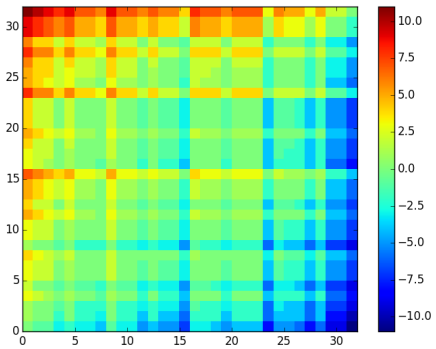
Figure 2.4: Delay characterization circuit.

2.7.2 Delay Measurement Results

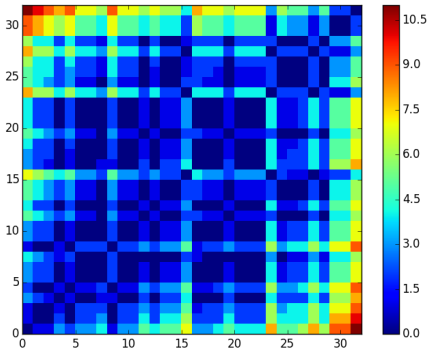
Average delay of each LUT are measured and calculated under 25 °C operating temperature, 1.2V FPGA core voltage. The results are shown in Figure 2.5. Figure 2.5a shows the delay difference between any pair of configuration bits. Figure 2.5b shows the absolute value of the delay difference between any pair of configuration bits and Figure 2.5c shows the Hamming distance heatmap between each pair.

The largest difference is 13 ps, which occurs between 00000 and 11111, located at location (x,y) = (0,31) and location (31,0) in Figure 2.5a and 2.5b. The diagonal line in both figures from the lower left corner to the upper right corner is all 0s because we are comparing each configuration bit to itself.

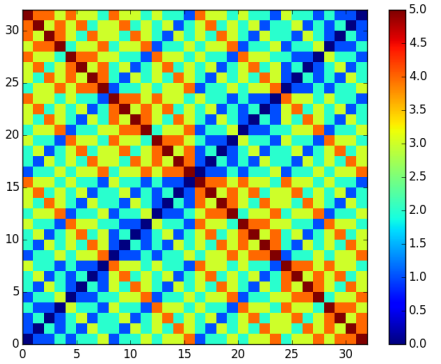
We notice that some patterns shown in Figure 2.5b can be observed in Figure 2.5c. The upper left and the lower right corner of both heatmaps are very similar, indicating that if two configuration vectors have a large delay difference in PDL, these two vectors also have a large Hamming distance. We believe this is an accurate observation because large Hamming distance indicates that the corresponding internal signal paths share very few common routes. Consequently, it is more likely to generate a higher delay difference. However, note that sharing a few common routes does not always indicate a large delay difference. Two very distinct signal paths might produce a small delay difference. Thus, we also see many patterns in the Hamming difference heatmap are not observable in Figure 2.5b



(a) Delay difference between any pair of configuration bits. Delay difference unit measured in *ps*.



(b) Absolute value of delay difference between any pair of configuration bits. Delay difference unit measured in *ps*.



(c) Hamming distance between all pairs of configuration bits.

Figure 2.5: Delay Measurement Results

2.7.3 Process Variation

We have run experiments on three difference FPGA boards to test the effect of process variation. For each board, we implemented PDL on five different locations with the same design. All PDLs are provided 00000 and 11111 as configuration bits. The average delays of PDL on three boards are compared and presented in Table 2.3 indicating process variation leading to approximately 1.6% of fluctuation. However, the delay differences stay relatively stable, with variation less than 3ps. Thus, we believe it is safe to assume process variation has limited impact on PDL when implemented on similar hardware, especially when the size of PDL is relatively small.

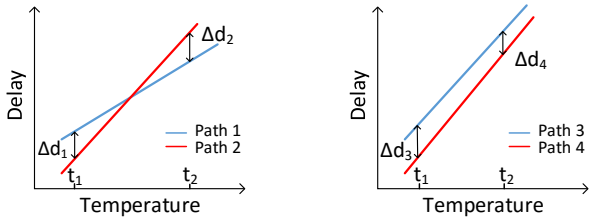
	00000 (ns)	11111 (ns)	Difference (ns)
FPGA 1	0.555	0.568	0.013
FPGA 2	0.553	0.564	0.011
FPGA 3	0.546	0.556	0.010

Table 2.3: Delay measurement results on three FPGAs (XC6SLX45).

2.7.4 Stability

Ideally, any pair of configuration bits c_i, c_j ($c_i \neq c_j$) should produce a pair of delays with a non-zero delay difference when applied to two PDLs, and the delay difference should remain stable. However, this is not always true in reality. PDL is a timing-based scheme. Thus, it is sensitive to environmental changes. Also, the environmental impact on delays may not be equal for each path inside the PDL. Non-uniform impact on delays thus leads to instability in pairwise delay differences.

As an example, Figure 2.6 illustrates two possible outcomes of temperature variations for two PDL paths. Figure 2.6a shows a scenario where path 2 is more sensitive to temperature than path 1. At lower temperature t_1, the delay on PDL path 1 d^{t1}_{Path1} is larger than the delay on PDL path 2 d^{t1}_{Path2} ($\Delta d_1 = d^{t1}_{Path1} - d^{t1}_{Path2} > 0$). However, when the temperature increases to t_2, delay on path 2 increases at a faster rate, and at a point the delay on path 2 is greater than the delay on path 1 ($\Delta d_2 = d^{t2}_{Path1} - d^{t2}_{Path2} < 0$). We intend to avoid this



(a) Scenario 1 (b) Scenario 2

Figure 2.6: Signal propagation delay of PDL vs. temperature of two PDL paths

scenario because this type of instability results in sign inversion of delay differences, creating a significant error with a relatively large probability. Figure 2.6b shows an almost ideal scenario where the sign of delay differences between path 3 and path 4 (Δd_3 and Δd_4) does not change as the temperature varies from t_1 to t_2. Also, $\Delta d_3 \approx \Delta d_4$, which means that the delay difference value stays relatively stable as well. Stable delay difference, in turn, leads to low error rate in emulation results.

Similar to PUFs, PDL is primarily affected by temperature and voltage. To analyze how PDL behaviors can vary, we test PDL in different environmental settings. Our experimental results show that PDL is capable of producing relatively stable delay differences in normal conditions. All delay measurements are done on a chain of four PDLs using the delay characterization circuit. The PDL chain is connected serially as shown in Figure 2.7. All delays are measured from the signal-in port to the signal-out port.

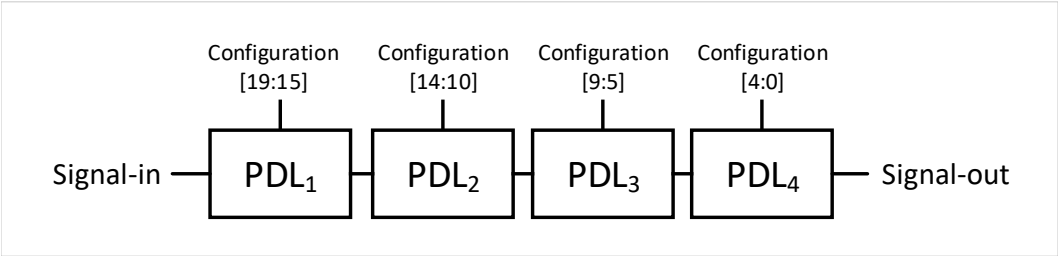


Figure 2.7: PDL chain used for stability tests.

2.7.4.1 Temperature

We evaluate the delays of PDL for 20,000 different configuration bits on a chain of four PDLs at five different temperatures within the allowed operating temperature range ($0 \sim 85\,^{\circ}\mathrm{C}$). The delays are measured using the delay characterization circuit as described in Figure 2.4. We adjust the temperature by placing the FPGA device in a temperature controlled chamber. To evaluate the delay characteristic variations as core temperature changes, we first collected delay measurements for 10,000 PDLs on FPGA. Then, we observe the stability of PDL in different temperatures through constructing 10,000 delay pairs (reference temperature vs. tested temperature) and calculate the delay ratio between each pair. Ideally, if the temperature impact on the PDL chain is uniform over all PDL paths, we should observe those delay ratios stay unchanged as we adjust the temperature. We set the delay ratios measured at $25\,^{\circ}\mathrm{C}$ as our reference and compare all delay ratios at different temperatures with it.

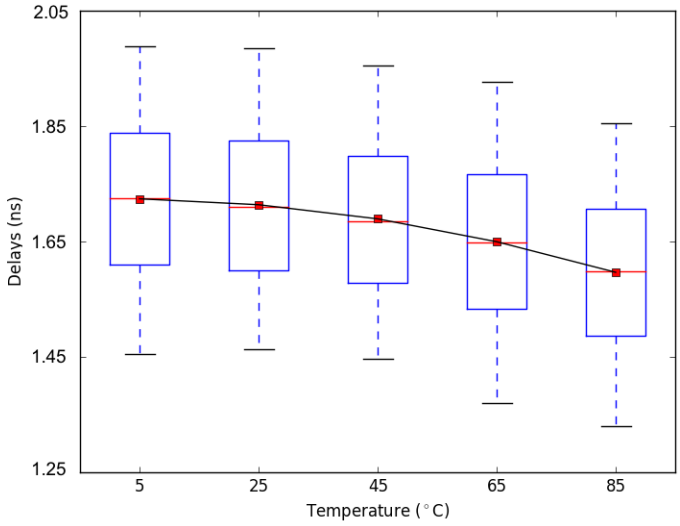
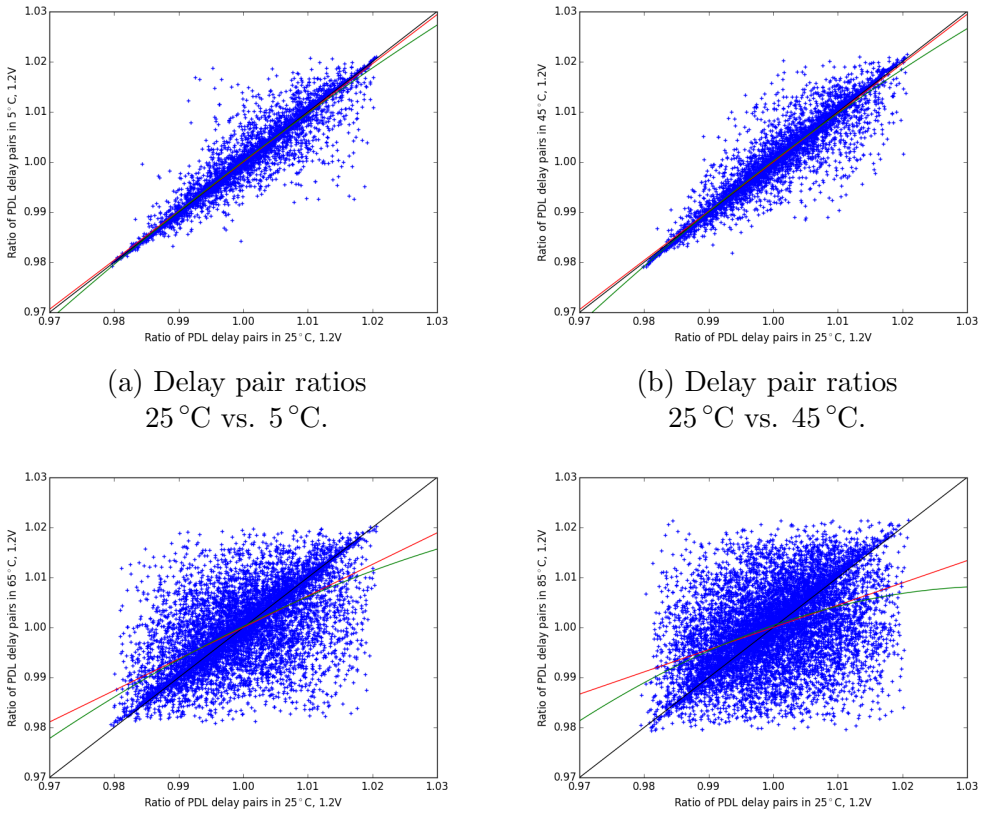


Figure 2.8: Delay characteristic variation under different temperature settings.

Figure 2.8 indicates the delay characteristic changes as core temperature varies. The average delay slightly decreases by 6.89%, while the variance in delays increases by 4.25% as temperature increase from $5\,^{\circ}\mathrm{C}$ to $85\,^{\circ}\mathrm{C}$.

Figure 2.9 shows the comparison results in four different temperature settings: $5\,^{\circ}\mathrm{C}$,



(a) Delay pair ratios 25 °C vs. 5 °C.

(b) Delay pair ratios 25 °C vs. 45 °C.

(c) Delay pair ratios 25 °C vs. 65 °C.

(d) Delay pair ratios 25 °C vs. 85 °C.

Figure 2.9: Delay ratio stability over 4 different temperature settings, VCCINT fixed at 1.2V. Red line: linear regression result. Green line: degree-2 quadratic regression result.

45 °C, 65 °C and 85 °C. Each blue point represents a specific delay ratio between two configuration vectors, where the x coordinate is the ratio calculated in the reference temperature and the y coordinate is the ratio calculated in the test temperature. The black line in each subgraph indicates the result of a perfectly stable PDL. Linear regression (red line) and degree-2 quadratic regression (green line) were performed on the collected data. The degree-2 quadratic regressions in all four settings are very close to straight lines, indicating a linear relationship between the x-axis and the y-axis. To quantify the stability of PDL in all temperature settings, we evaluate the slope, intercept, and standard error of all linear regression results in Table 2.4.

In general, at 5 °C and 45 °C, the regression slope is very close to 1, the intercept is close to 0, and the standard error is almost negligible, meaning the impact of temperature change

Temperature	Slope	Intercept	std_err
5 °C	0.9795	0.0200	0.0027
45 °C	0.9813	0.0186	0.0030
65 °C	0.6304	0.3696	0.0075
85 °C	0.4450	0.5550	0.0093

Table 2.4: Linear regression results on PDL temperature stability evaluation.

is mostly uniform over all PDL paths so that the delay difference stays relatively stable. At higher temperatures 65 °C and 85 °C, the regression slopes are respectively 0.6304 and 0.4450, far from 1, indicating some paths are much more sensitive to the temperatures than others. This result means at very high temperature the PDL-based emulation segment result has a much higher probability of being inconsistent with the results collected at 25 °C. In our emulation platform design, we assume that the temperature varies at most 20 °C from the room temperature. Thus, it is safe to claim that PDL is stable against reasonable thermal fluctuation.

2.7.4.2 Voltage

Similar to the evaluation of thermal variations, we investigate the voltage variation impact on PDL. Our experimental platform Spartan-6 does not contain a freely tunable DC-DC converter in the power module, thus adjusting core voltage cannot be done internally. Moreover, the manufacturer of our experimental platform has a fairly stringent requirement on FPGA core voltage (VCCINT = 1.2V), making it very difficult and risky to directly apply an adjustable external power source to the FPGA core. Fortunately, Spartan-6 provides an extended performance mode that applies to VCCINT = 1.26V. We first evaluate the changes in delay characteristics as we change the VCCINT in Figure 2.10. We observe that the average delay slightly decreases by 11.19%, while the variance in delays decreases by 19.46% as VCCINT increase from 1.2V to 1.26V.

We also evaluate the delay ratio between 10,000 PDL delay pairs in both normal mode and extended performance mode. The result is shown in Figure 2.11.

Figure 2.11 shows that by increasing the FPGA core voltage from 1.2V to 1.26V, the im-

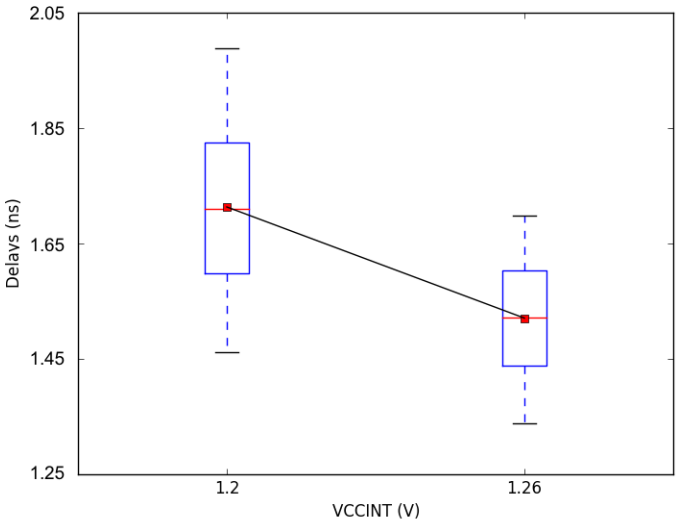


Figure 2.10: Delay characteristic variation under two VCCINT settings.

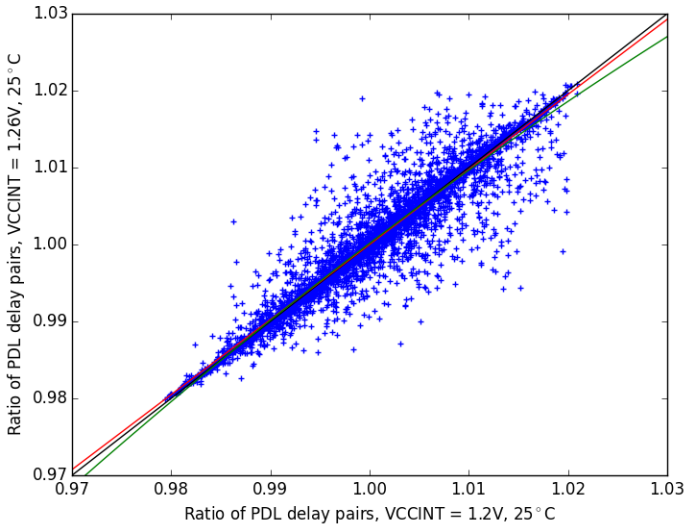


Figure 2.11: Delay ratio stability when increasing FPGA core voltage from 1.2V to 1.26V, operating temperature fixed to 25 °C. Red line shows linear regression result, green line shows degree-2 quadratic regression result.

pact on each path is relatively stable and consistent. Also, both linear and degree-2 quadratic regression were performed and the results are plotted in the figure. The quadratic regression result (green line) is almost flat, indicating that a linear model is a better representation of the data. The linear regression result (red line) indicates that the delay pair ratios stay mostly stable as we increase the voltage. Linear regression has a slope of 0.9754, intercept of

0.02463, and standard error of 0.0027, very close to the ideal result (black line). Based on the results, we claim it is safe to assume that the PDL delay ratio stays relatively stable against minor changes in voltage and the impact of voltage variation is uniform over all paths.

Also, it is interesting to notice that less variance is observed on both ends of the plot. This phenomenon is also observed in temperature variation experiments. When the delay ratios are further away from 1, meaning the delay differences are larger, the PDL is less likely to behave differently when the environment changes.

2.8 PUF Emulation - Design

In this section, we discuss a PDL-based hardware emulation of a characterized PUF. We first propose a segmental emulation approach that emulates each segment in the PUF. Later we propose a method to scale the delay difference by a factor in the new emulation platform while maintaining the challenge-response relationship. Lastly, we introduce a method to find the scaling factor that maximizes emulation accuracy.

2.8.1 Perfect Segmental Emulation

The goal of perfect segmental emulation is to create an exact "clone" for each segment in the PUF so that when connecting all emulation segments together, an accurate emulation of the entire PUF is then constructed.

After the characterization process of a PUF, delay differences for each segment is retrieved. We use three PDL to create an emulation segment E^i with three different delays d_A^i, d_B^i and d_C^i to emulate the ith segment of the PUF. We program the control bits of each PDL so that the three delays are capable of producing two delay differences Δd_1^i and Δd_2^i that are identical to the two delay differences the ith PUF segment generates. The segmental emulation is described in Equation 2.3 and 2.4.

$$\Delta d_1^i = d_a^i - d_b^i \tag{2.3}$$

$$\Delta d_2^i = d_a^i - d_c^i \tag{2.4}$$

If we can guarantee the validity of Equation 2.3 and 2.4, then it is safe to claim that E^i now behaves the same way the ith PUF segment behaves.

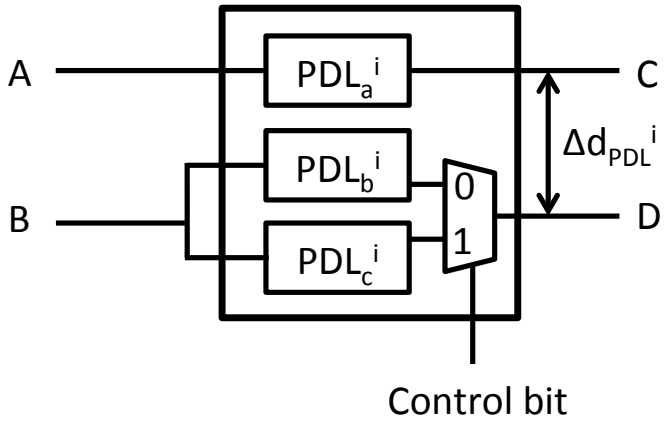


Figure 2.12: Internal design of the ith emulation segment E^i.

Figure 2.12 shows the internal structure of the ith emulation segment E^i. PDL_a^i is configured to have delay of d_a^i, PDL_b^i is configured to have delay of d_b^i and PDL_c^i is configured to have delay of d_c^i. The entire segment takes a control bit to select a pair of PDL delays to generate the desired delay difference: Δd_1^i or Δd_2^i.

2.8.2 Delay Difference Scaling

Perfect segmental emulation guarantees the correctness of the emulation; however, it is challenging to implement. The delay difference characterization results we eventually obtained are relatively small, sometimes even less than the PDL resolution of 1 ps. Thus, it is not possible to create a perfect clone (with the same delays) of a PUF segment using PDL on an FPGA platform.

We observe that an accurate emulation can be created without a perfect segmental emulation. The PUF response does not depend on actual delay value, but instead the sign of the difference between competing for path delays. We can multiply the real delay difference by any positive factor α, and the sign of the difference between path delays should remain the

same. If $d(p) - d(p') > 0$, then $\alpha d(p) - \alpha d(p') > 0$ for $\alpha > 0$. Delay difference scaling allows adjustments of delays in each PDL so that accurate emulation can still be achieved. Note once a suitable scaling factor α has been decided, this value should be applied consistently to all emulation segments to ensure the correctness of the sign of final delay difference.

2.8.3 Scaling Factor

A PDL is able to create a delay d_{PDL} that falls within a specific range $[d_{min}, d_{max}]$. This range usually covers delay difference from 1-100 ps depending on the FPGA platform. Before emulating a target PUF, all emulation segments should agree on a minimum delay difference of Δd^{min}_{PDL}. Δd^{min}_{PDL} should not be less than the PDL resolution because a small delay difference would lead to the domination of process variation and measurement errors over actual PDL path difference, resulting in unpredictable and inaccurate results in the emulation.

The minimum delay difference can be used to identify the smallest scaling factor of α_{min}. We define minimum scaling factor in Equation 2.5.

$$\alpha_{min} = \frac{\Delta d^{min}_{PDL}}{\Delta d^{min}_{PUF}} \tag{2.5}$$

Similarly, the maximum scaling factor α_{max} should be the ratio of largest delay difference obtainable by two PDL and the largest delay difference in the characterization result as shown in Equation 2.6.

$$\alpha_{max} = \frac{\Delta d^{max}_{PDL}}{\Delta d^{min}_{PUF}} \tag{2.6}$$

An ideal scaling factor α should be in the range $[\alpha_{min}, \alpha_{max}]$. Our experimental results in Figure 2.9 and 2.11 show that a large delay difference usually results in better stability against environmental changes. A larger scaling factor creates larger delay differences; thus, in turn, making emulation more stable. Though a large scaling factor is favored for stability, in theory, all scaling factors should provide the same emulation accuracy.

2.9 Emulation Improvement

The newly created emulation should be able to predict a large number of CRPs of the target PUF. However, basic segment-by-segment emulation fails to consider many unpredictable factors. In this section, we provide two techniques that could potentially benefit our emulation scheme in terms of overhead and accuracy. We first explore a mechanism to emulate every two segments in the PUF to reduce the total number of PDLs required for the emulation. We then propose to use multiple PDL emulations to vote for a prediction result to eliminate process variation effects and environmental noise.

2.9.1 Two-Segment Emulation

Originally each PUF segment generates two delay differences Δd_a^i and Δd_b^i, so three PDLs are sufficient to emulate one PUF segment. To emulate an n-bit PUF, we require as many as $3n$ PDLs. However, when treating every two PUF segments as a group, the new group now generates four different delay differences: $\Delta d_a^i + \Delta d_a^{i+1}$, $\Delta d_b^i + \Delta d_a^{i+1}$, $\Delta d_b^i + \Delta d_a^{i+1}$ and $\Delta d_b^i + \Delta d_b^{i+1}$. Now in order to emulate these four delay differences, five PDLs are needed. To emulate an n-bit PUF, only $\frac{5n}{2}$ PDLs are needed, saving $\frac{n}{2}$ PDLs comparing to one-to-one emulation scheme.

2.9.2 Output Voting

As shown in Section 2.7.4, PDLs are sensitive to environmental changes. Even though we scaled the delay difference to a relatively large number to reduce the effects of environmental change, some errors are inevitable. Moreover, hardware malfunctions could also lead to incorrect emulation results. To eliminate the errors, we propose to create multiple emulations of the same PUF. When provided with a challenge, multiple emulations vote for the corresponding response, and we adopt the most-voted response as the correct output. The emulation process and the scaling factor for all emulation copies should be identical. If the majority of emulation copies produces a response r, it is more likely that r is the correct

response.

2.10 Experimental Results

We implement our PDL-based PUF emulation on Spartan-6 XC6SLX45 FPGAs. When applying the same challenge vector to both the target PUF and the PDL-based emulation, the probability that the original PUF (running at 25 °C and 1.2V) and the emulation generate the same response is defined as **emulation accuracy** or **prediction accuracy**. PUF responses are collected 10 times to rule out unstable CRPs.

2.10.1 Characterization Accuracy

Our characterization approach provides highly accurate delay characterization at the segment level. We first test the precision of our results by solving 10 sets of linear equations generated from **random CRPs**, we then characterize the PUF by solving 10 sets of linear equations obtained from only **stable CRPs**. The results for the delay differences in the first 8 PUF segments are shown in Figure 2.13.

Both characterizations provide very similar results. However, the variance for each variable solution is different. For all obtained delay differences from random CRPs, the average variance is 0.0145 ps, 64.77% greater than the average variance of solutions obtained from only stable CRPs (0.0088 ps). This variance difference shows that characterizing using only stable CRPs provide much better accuracy.

2.10.2 Baseline Emulation

The target PUF we intend to emulate is a 64-bit arbiter PUF. We first collect a set of 100,000 randomly selected CRPs to run the characterization. We then collect a set of 100,000 stable CRPs and characterize the PUF only on stable CRPs. We create a software simulation using characterization result on both datasets.

Based on the two PUF characterization results, we also emulate the PUF using PDLs

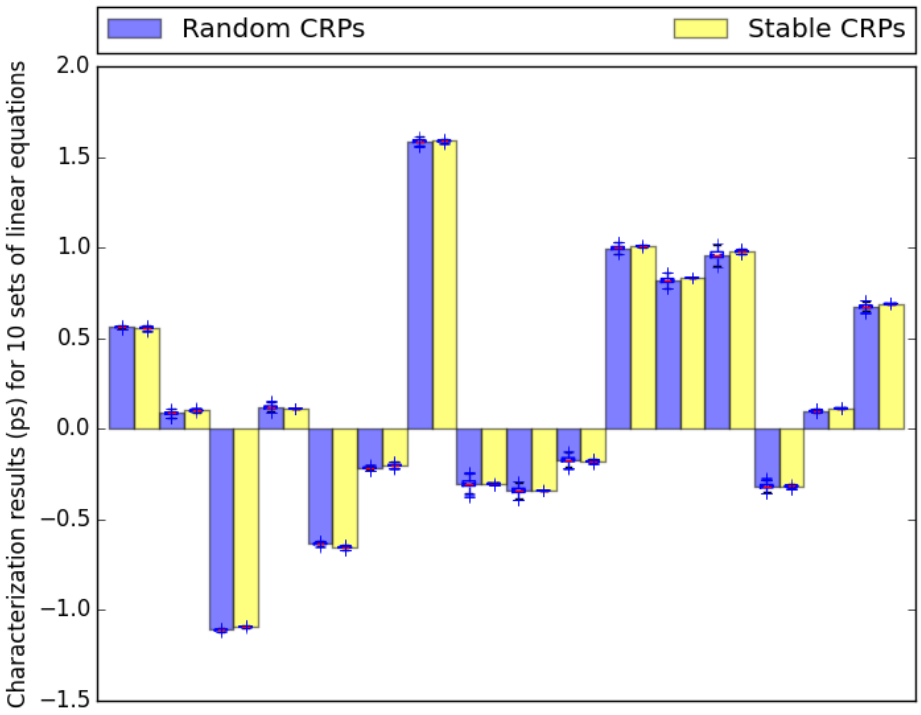


Figure 2.13: Characterization results obtained from solving 10 sets of linear equations generated from random and stable CRPs. Boxes indicates 95% confidence interval.

without emulation improvement techniques (baseline emulation). For the baseline emulation, we calculated the scaling factor range to be between 3 and 8. Based on our observation in Section 2.8.3, we take the maximum value within the range 8 to be our scaling factor.

We measure the PUF prediction accuracy for both simulation and PDL-based emulation. For simulation and emulation based on random CRP characterization, the accuracy is measured by providing newly generated 100,000 random challenges to both the target PUF and its emulation. For simulation and emulation based on stable CRPs, the accuracy is calculated as the successful prediction rate on newly generated stable challenges.

Table 2.5 shows the prediction accuracy using both software simulation and PDL-based emulation. Both random CRP characterization and stable CRP characterization were applied. Using stable CRP as the characterization dataset improves the software simulation accuracy by 5.19%, for a corresponding characterization accuracy of 5.19%. Characterization only on stable CRPs improves the emulation accuracy by 5.07% as well. Experimental

results also show that our PDL-based emulation scheme loses approximately 5% accuracy compared to software simulation.

Dataset	Simulation	Emulation
Random CRP	85.93±1.76%	80.72±2.27%
Stable CRP	91.12±1.49%	85.79±1.78%

Table 2.5: PUF prediction accuracy for software simulation and PDL-based hardware emulation using random CRP characterization data and stable CRP characterization data. The emulation accuracy is shown in the format of 95% confidence interval.

2.10.3 Improved Emulation

We measure the delay, area, and energy consumption for both the baseline and two-segment PDL-based PUF emulation on a 64-bit PUF. The results are shown in Table 2.6. two-segment emulation saves 16.67% LUTs, 47.01% in maximum delay, and 11.84% in terms of energy consumption.

Type	Baseline	Two-Segment
LUTs	192	160
Slices	96	80
Max Delay (ns)	37.84	20.05
Energy (μJ)	3.04×10^{-4}	2.68×10^{-4}

Table 2.6: Overhead of Baseline and two-Segement PDL-based emulation of a 64-bit PUF.

We then apply an emulation voting mechanism. We create three identical PDL-based emulation copies, and we always take the response generated by the majority party as the output. We combined the two techniques and evaluated the overall PUF prediction accuracy.

Dataset	Baseline	Two-segment	Voting	Combined
Random	80.72±2.27%	80.97±1.92%	82.72±0.91%	83.15±1.79%
Stable	85.79±1.78%	86.09±2.06%	87.22±0.62%	87.42±1.59%

Table 2.7: PUF prediction accuracy for baseline emulation, two-segment emulation, and voting. All emulation accuracy results are shown in the format of 95% confidence interval.

Table 2.7 shows our evaluation results. On average two-segment emulation does not affect the emulation accuracy while saving overall area, delay, and energy overhead. The

voting mechanism, on the other hand, produces approximately 2% and 1.43% improvement for random and stable datasets. By combining the two techniques, we achieve approximately 2.43% improvement on a random CRP dataset and 1.63% on a stable CRP dataset.

2.10.4 Emulation Stability

We evaluated the stability of PDL in different temperature and voltage settings. We first apply 50 randomly generated challenges to the PDL-based emulation scheme. Each challenge is applied 1,000 times, and we observe the rate of output consistency. Our evaluation results show that on average 98.45% of the produced responses stay stable where the target PUF has only 87.94% of the responses stays consistent.

We then measure the emulation accuracy comparing to original PUF in different environmental settings to show that our PDL-based emulation scheme is stable against more substantial environmental variations.

Environment	Accuracy
25 °C, 1.2V	85.79±1.78%
5 °C, 1.2V	84.17±2.55%
45 °C, 1.2V	83.78±3.36%
25 °C, 1.26V	82.42±2.05%

Table 2.8: Baseline emulation accuracy in different environments. Characterization done using 10 sets of stable CRPs. All emulation accuracy results are shown in the format of 95% confidence interval.

Table 2.8 shows the resulting accuracy slightly decreases by 1.62% and 2.01% when adjusting the temperature by 20 °C in normal mode (VCCINT =1.2V). When controlling the operating temperature to be consistent, by increasing the voltage to 1.26V, emulation accuracy decreases by 3.37%. However, in general, the accuracy is still maintained at a high level; thus, our proposed emulation scheme is stable against large environmental changes.

2.10.5 Latency Overhead

We compare our design with both PUF simulations using HDL simulators and pure statistical simulations regarding latency when predicting 10,000 CRPs. The results are shown in Table 2.9. The comparison indicates that our design is 9644x faster than HDL simulators and 827x faster than software simulations while maintaining competitive prediction accuracy.

Type	Latency(s)
HDL Simulation	96.44
Statistical Simulation	0.827
PDL-based Emulation	0.001

Table 2.9: Average latency comparison in generating 10,000 CRPs using software HDL simulator, pure software simulation, and PDL-based emulation.

2.11 Chapter Conclusion

In this chapter, we proposed a fast, compact and low energy PUF emulation platform using programmable delay lines on FPGA. Our core idea is to characterize an arbiter PUF and emulate the delay difference of each PUF segment using PDL. The PDLs are configured such that the emulation has almost the same challenge-response mapping function. We also evaluated the stability properties of our emulation platform and demonstrated that our design and implementation is robust against environmental variation. Furthermore, we have proposed two techniques that are capable of reducing overhead and increasing emulation accuracy. Experimental results show that our design is 827x faster than software simulation while providing comparable accuracy.

CHAPTER 3

Securing PUFs with Interconnection and Reconfigurability

3.1 Motivation

As of today, the amount of private information stored on and flows between electronic devices is unimaginable. Adversaries are highly motivated to attack these electronics because of the potential benefits they can gain from the stolen personal information. Secure and robust protection of electronics, as a result, is essential for any individual who seeks security and privacy.

Physical Unclonable Functions (PUFs) came to the stage when traditional cryptography failed to stand its ground against physical attacks, side-channel attacks, and API attacks. A PUF, different from traditional key-based cryptographic systems, does not require a secret binary key; instead, the physical entity itself serves as the key. One huge advantage of a PUF-based system is that the secret key hidden within the physical body is designed to be unclonable since it utilizes uncontrollable, nanoscale process variations. The complex structure of a PUF makes the output much harder to be predicted or derived comparing to those digital systems that stores secret keys in non-volatile memories.

Strong PUFs is a major subtype of PUFs. Like all PUFs, a strong PUF implements a complex function that maps some challenges to some responses. A PUF is considered as a strong PUF if it is capable of meeting all the following requirements:

- Unclonability. Unclonability is the most fundamental feature of a PUF. A specific strong PUF cannot be physically cloned or replicated by anyone. Even the manufac-

turer who produces the PUF should not be able to manufacture a copy of the PUF that implements the same mapping function between challenges and responses.

- Unpredictability. Predicting the response of a random challenge should be extremely difficult, even if the attacker is capable of obtaining a large number of CRPs.

- Determination difficulty. A strong PUF cannot be fully measured or determined within a reasonable amount of time. A PUF with a small challenge-response mapping set does not meet this requirement since all mappings can be recorded given enough time (hours, days or weeks). In most cases, this requirement is equivalent to a large set of possible challenges and limited read-out frequency.

Several strong PUFs have been proposed and studied in the past, yet none have been proven to be secure enough to hold all three requirements. The rise of machine learning technology provides adversaries with a powerful weapon that is capable of creating a model of the function a PUF implements. A mathematical model is a software program that is capable of predicting the corresponding responses of a PUF when provided with random challenges with high probability. Such a mathematical model can be easily established by learning from a small subset of CRPs.

3.2 Technical Goals and Contributions

In this chapter, we intend to address this problem. We propose a reconfigurable interconnected PUF network structure that is capable of providing sufficient robustness and resilient against different types of machine learning attacks. Essentially the idea is to create a network structure that interconnects multiple PUFs so that the system is so complex that current machine learning attack methods are unable to accurately predict the responses given arbitrary challenges in a reasonable amount of time. The proposed design is capable of reconfiguring itself so that challenge-response mappings completely alter. The reconfiguration of an IPN forces an adversary to restart the attack to learn a new mapping function.

To the best of our knowledge, we make the following contribution in this chapter:

- We have proposed a network structure that interconnects multiple PUFs. By doing so, we significantly increase the system complexity as well as breaking the linearity so that the interconnected PUF network shows high resilience against current machine learning attack.
- Our interconnected PUF network is compatible with any strong PUF. In this work, we simulated and implemented interconnected PUF network with only delay-based PUFs and some well-known variations, however, the whole framework can be easily extended to other strong PUFs such as Bitline PUF and current mirror PUF.
- We have tested our interconnected PUF network against different algorithms, with and without the reconfiguration functionality. We show that the sample complexity of an IPN is significantly larger than the state-of-the-art delay-based PUF and its variants. Modeling an IPN requires a much larger training set as well as much longer time. We are the first to propose to reconfigure a PUF-based system before an adversary could collect the theoretical lower bound of the sample complexity.
- Our reconfigurable PUF network design can be reconfigured during runtime with much lower latency and overhead comparing to other reconfigurable PUF design such as [53].

3.3 Related Work

3.3.1 Modeling Attack

PUFs are vulnerable to modeling attacks. Early works on modeling attack targeting PUFs were focused on standard arbiter PUFs [54] [55]. Later on Rhrmair et al. presented modeling attack results on multiple commonly seen PUFs, including APUFs, XOR PUFs, feed-forward PUFs. The proved that all investigated PUFs are vulnerable to machine learning attacks [56]. Vijayakumar et al. later presented more detailed insights on applying different machine learning attacks to popular PUFs and why simple PUF structures are weak against modeling attacks [57]. Our results show that the proposed IPN structure provides sufficient resilience against modeling attacks proposed in the above papers.

The rise of deep neural networks has imposed new challenges to arbiter PUFs and variants. A deep neural network is capable of training a model that simulates the function a PUF implements with high accuracy within a short period of time. Yashiro et al. conducted a security evaluation of authentication systems using arbiter PUFs and concluded that an arbiter PUF and its variants are vulnerable against deep learning attack [58]. In this chapter, we show that an IPN provides high resilience against deep learning attack. The complexity of an IPN is significantly larger than other PUF-based systems so that a deep neural network can easily fall into overfitting problems when attempting to model an IPN. The reconfiguration functionality provides additional protection by changing the mapping function regularly.

3.4 Preliminaries

3.4.1 Strong PUF Model

An IPN can use any strong PUF as fundamental building blocks. We use standard delay-based arbiter PUFs as an illustrative example for simplicity considerations. An n-bit APUF takes an n-bit vector as a challenge and produces a 1-bit response as output. The challenge is provided to configure two nominally identically paths. Each challenge bit controls whether a pair of paths should swap positions within a PUF segment. An impulse signal is fed into the system to excite both paths simultaneously to retrieve a response. Because of the uncontrollable, nanoscale process variation, the signal traveling along one of the two paths reaches the arbiter earlier, generating corresponding output.

Assuming an arbiter PUF implements a function F that maps a set of n-bit challenges C to corresponding response set R. We assume no delays on the connection wires and all delays are contributed by the APUF segments. Given a specific challenge $c \in C$, the ith APUF segment generates a pair of delays with delay difference of Δd_i^c. The corresponding response $r \in R$ can be mathematically represented as Equation chap3:eq:stable:

$$F(c) = r = \begin{cases} 0 & \text{if } \sum_{i=1}^{n} \Delta d_i^c > 0 \\ 1 & \text{if } \sum_{i=1}^{n} \Delta d_i^c < 0 \end{cases} \tag{3.1}$$

3.4.2 IPN

3.4.2.1 IPN Node

An IPN consists of nodes and edges. A node consists of multiple arbiter PUFs of the same length. We define an n-bit IPN node of size m consists of m n-bit APUFs. If $m = n$, a node is denoted as a *homogeneous node*, otherwise it is denoted as a *heterogeneous node*. The size of a node is the total number of arbiter PUFs running in parallel, and the length of a node is the number of segments of each arbiter PUF in the node. A demonstrative diagram of a node is shown in Figure 3.1. An IPN node takes an n-bit vector as the challenge and generates m 1-bit responses. All APUFs within the same IPN node share the same challenges.

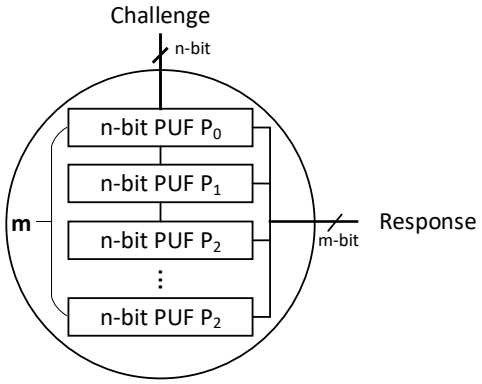


Figure 3.1: An IPN node of size m and length n. If $m = n$, the node is homogenous, otherwise it is heterogenous.

3.4.2.2 IPN Edge

An IPN node connects to other nodes through edges. To achieve reconfigurability, an edge is essentially designed to be a shuffler that takes the output from the previous node, shuffles the order and feed them to the next node. The design of a shuffler is based on a crossbar switch like architecture that redirects inputs to outputs. An conceptual illustration is shown

in Figure 3.2. Each intersection of the crossbar is a tri-state buffer that controls how the inputs can be routed to each output. A configuration vector is used to configure how the connections between two nodes are shuffled, and each bit is used to switch each tri-state buffer. For example, if an edge is an n-bit shuffler that directs the i-th bit of the input to the j-th output bit, the tri-state buffer at the intersection of the i-th and the j-th bar is thus set. All output port numbers are represented in the binary form. Note that our design of the shuffler is non-blocking; thus conflicts are acceptable, though a large number of conflicts could weaken the unpredictability of IPN. Starvation would also not be a problem as the previous input would be buffered and applied in case of output starvation.

Using our design of the shuffler, the connections between nodes can be reconfigured easily by changing the configuration vector. We define a configuration of an IPN as a collection of all configuration vectors for all shufflers in the IPN.

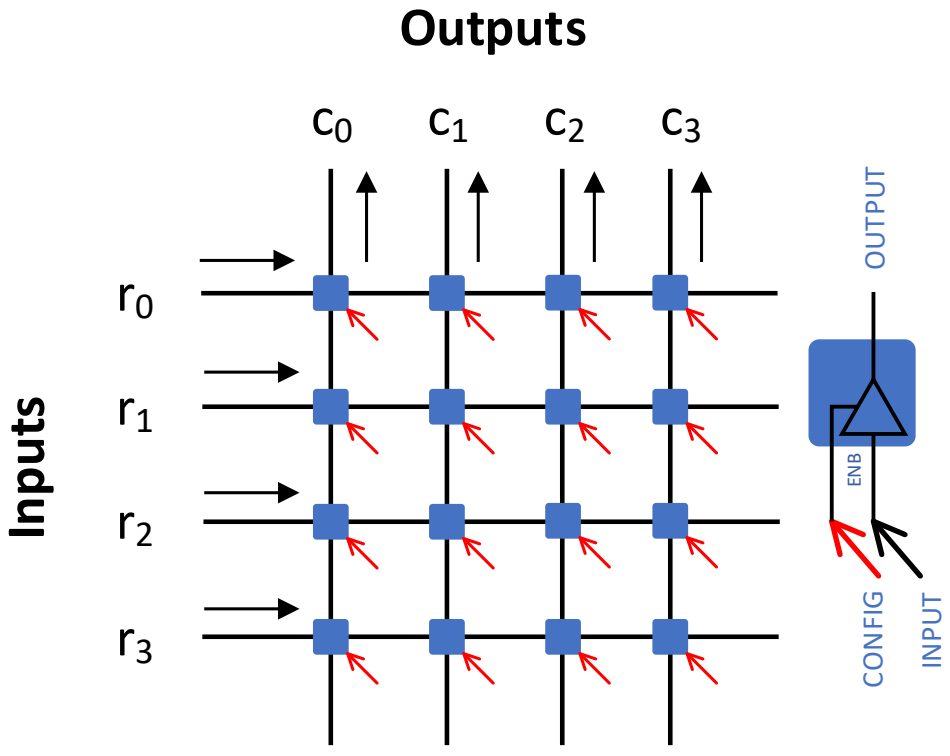


Figure 3.2: A conceptual illustration of a crossbar switch like shuffler.

3.4.2.3 IPN Chain

A simple network can be constructed by connecting IPN nodes using to form a chain. All IPN nodes are connected through IPN edges. An edge from $node_i$ to $node_{i+1}$ indicates that the output of $node_i$ is fed into a shuffler, then connects to all APUFs in $node_{i+1}$. Thus, each APUF in $node_{i+1}$ depends on the outputs of all APUFs in $node_i$.

3.4.2.4 More Complex Connections

IPN nodes can be connected in more complex manners. IPN supports not only one-to-one but also one-to-many, many-to-one and many-to-many connections between nodes.

One-to-many connections can be used to increase the output length as multiple nodes take the output of a specific node as input. The n-bit output of $node_0$ is used as input for two n-bit homogenous nodes $node_1$ and $node_2$, eventually generating a 2n-bit response.

We borrow the idea of XOR PUFs to use logic like AND, OR or XOR to create many-to-one connections in IPNs. Many-to-one connections can be used to break the linearity and to increase system entropy. $node_0$ and $node_1$ takes the same input and generates two sets of corresponding outputs. A logic operation such as XOR is applied to the outputs, and the result is then taken by $node_2$ as input. Many-to-one connections are expensive since the input/output length ratio is significantly larger than that of a one-to-one connection, resulting in requiring more PUF segments to build.

A many-to-many connection is a mixture of both one-to-many and many-to-many connections. The XORed result of both $node_0$ and $node_1$ outputs is fed to both $node_2$ and $node_3$ as input. A many-to-many connection provides additional nonlinearity without sacrificing the output size.

The combination of one-to-one, one-to-many, many-to-one and many-to-many connections enables the possibility of creating larger and more complicated IPN, providing additional resilience and robustness against various attacks.

3.4.3 IPN Parameters

IPN nodes, edges and different connections provide tremendous freedom in constructing a network. In this section, we intend to define some parameters associated with IPN structures.

3.4.3.1 Network Depth

We define the depth of an IPN as the length of the shortest path from an input node to an output node. An IPN with greater depth theoretically creates more dependency within the network, making the entire structure not differentiable so that machine learning techniques based on differentiable models (e.g., Support Vector Machine) inefficient. Also, a deeper IPN has multiple layers of dependency and requires more PUF segments, which increases the system complexity and makes it more difficult to predict. The concept of levels in an IPN is strongly associated with depth. The level of an IPN node is defined by one plus the smallest number of connections between the node and the root node.

3.4.3.2 Network Width

We define the width of an IPN as the maximum number of nodes that shares the same input. According to our design, a wider IPN would have more many-to-one or many-to-many connections comparing to the slimmer topology. A wide IPN has more nonlinearity since many-to-one, and many-to-many connections require nonlinear logics such as AND, OR and XOR.

3.5 Attacks models

In this section, we first discuss the security assumptions we make in this chapter. We then briefly explain some conventional modeling/characterization techniques on PUFs that have been proven to be effective in state-of-the-art PUF-based systems. We later investigate some newly proposed modeling methods including deep-learning-based attacks and autoML modeling.

3.5.1 Assumptions

We consider the same assumption for controlled PUFs [59] that physical attacks on the control logic (which in our case is the reconfiguration logic) are more likely to alter or even destroy the PUF itself. The adversary has physical access to the PUF and its public CRP interface, as it is common in the established PUF attack model. The adversary can thus repeat CRP measurements at will in order to gain output stability.

3.5.2 Logistic Regressions

Logistic regression is proven to be effective against conventional delay-based arbiter PUFs and variations such as XOR PUFs and lightweight secure PUFs. Logistic regression-based PUF attacks use a weight vector $\vec{w}$ to encode the internal parameters within the PUF system. The conditional probability can be represented using sigmoid acting on the PUF function f as shown below. c is a challenge, and r is the corresponding response.

$$p(c, r|\vec{w}) = \text{sigmoid}(rf(\vec{w}))$$

For a training set τ, the goal of the regression is to find a weight vector $\vec{w}$ so that the likelihood of observing this set is maximized, which is equivalent to minimizing the negative log-likelihood shown in Equation 3.2.

$$\hat{\vec{w}} = \arg\min_{\vec{w}} -log\mathcal{L}(\tau, \vec{w}) = \sum_{(c,r)\in\tau} -log\ \text{sigmoid}(rf(\vec{w})) \tag{3.2}$$

Different from general logistic regression problems, the optimal parameter vector $\hat{\vec{w}}$ can not be analyzed analytically, the only option is to optimize iteratively. For the purpose of attacking a single-bit output PUF, the problem can be tackled by transforming it to a binary logistic regression problem and use iteratively reweighted least squares (IRLS) to minimize the Log-likelihood of a Bernoulli distributed process using Newton's method. Other options includes using optimization methods such as gradient decent and RProp etc. The above

optimization methods requires gradient information represented in Equation 3.3:

$$\nabla \mathcal{L}(\tau, \vec{w}) = \sum_{(c,r)\in\tau} r(\text{sigmoid}(rf(c, \vec{w})) - 1)\nabla f(c, \vec{w})) \tag{3.3}$$

3.5.3 Evolution Strategies

Evolution strategies is a commonly seen attack method on PUF-based systems. Evolution strategies is a different type of machine learning algorithms that performs random search intelligently. Inspired by evolutional adaption to environments, the evolution strategies method always choose the best candidates from randomly generated models and further develops on them. In the case of modeling a PUF-based system, one instantiation of internal delay parameters is denoted as an *individual*, and all instantiations together are called the *population*. The population of each selection is called a *generation* and each selected individual is allowed to produce *offsprings* by randomly mutating the instantiation of delay parameters. The selection is performed based on how well an individual instantiation is capable of reproducing the correct CRPs (*fitness*).

Fitness evaluation over the entire training set is expensive and slow. We borrowed the mini-batch idea from stochastic gradient descent to select only a subset of training CRPs for fitness evaluation purposes.

3.5.4 Multilayer perceptron

The development of deep neural network has made tasks that were once believed undoable possible. From speech recognition to image captions, deep neural networks have made miraculous progress and is still improving. It is not surprising that adversaries use artificial neural networks to model a PUF-based system. A multilayer perceptron (MLP) is a type of feedforward artificial neural network. An MLP consists of at least three layers of nodes, respectively input layer, the output layer, and hidden layers. Each node within an MLP is a neuron that uses a nonlinear activation function. MLP learns a model by iteratively changing the connection weights based on the error between the output and ground truth.

This type of supervised learning using MLP is carried out through back-propagation using gradient descent.

For PUF attacking purpose, predicting PUF output is essentially a classification problem. MLP is more advantageous comparing to logistic regression methods in terms of capability to learn non-linear models.

3.5.5 Other Machine Learning Algorithms

As conventional attacks on PUFs depend on attackers to manually choose a model and hyper-parameters, the results of modeling attack might not be optimal concerning both prediction accuracy and speed. AutoML is a new concept that focuses on progressive automation of machine learning. AutoML aims to create an automated process that intelligently performs architecture search over a wide range of machine learning algorithms and choose the one that best fits the data and the task including naive Bayes classifiers, decision trees, etc. AutoML is also capable of performing hyperparameter optimization that aims to find the best-suited hyperparameters for a given model. We choose to use auto-sklearn to search for an algorithm along with corresponding hyperparameters to predict the behavior of an IPN [60].

3.6 Reconfiguration

IPNs benefit from the complex structure so that it requires much larger training set and longer training time to model. We propose to reconfigure the entire network from time to time by changing the connections between IPN nodes so that any obtained knowledge on the IPN would be invalidated. Essentially, we are running a race with adversaries. Before one can finish modeling an IPN or collect enough training set, we reconfigure it so that the input-output mapping alters and the attacker would need to remodel the new IPN.

3.6.1 Reconfigure Timing

We can initiate a reconfiguration either before an attacker could collect sufficient number of CRP or before an attacker can finish modeling. However, since the speed of the attack process is affected by many factors on the adversarial side that we have no control of, we believe limiting the total number of generated CRPs is more secure and feasible. Thus, we intend to find a lower bound for the size of the training set of IPN.

Sufficient size of the training set is also known as the sample complexity. We consider models of all PUF-based system mentioned in this chapter as a binary function that takes a challenge and generates a 1-bit output of either 0 or 1. Vapnik-Chervonenkis theory suggests that a PUF-based system can be learned with a finite sample complexity and the minimum required training size (N) follow the Equation 3.4:

$$N = O(\frac{VC(\mathcal{H}) + ln(\frac{1}{\delta})}{\epsilon}) \tag{3.4}$$

where VC($\mathcal{H}$) is the Vapnik-Chervonenkis dimension of the function $\mathcal{H}$ implemented by the attacked PUF-based system, δ is the failure probability and ϵ is the learning error.

For arbiter PUF, the VC-dimension is the total number of stages, meaning for a k-bit arbiter PUF, $VC(\mathcal{H}) = k$. Rhrmair et al. derived the VC-dimension for XOR PUFs as $VC(\mathcal{H}) = k \cdot l$ where k is the number of stages in each arbiter PUF and l is the total number of XORs. For Feed-forward PUFs, $VC(\mathcal{H}) = k + l$ can be used to describe the model better where k is the total number of stages and l is the total number of feed-forward loops.

The sample space of an IPN on the other hand largely depends on the topology of the network. We have to be conservative in terms of finding a uniform lower bound for all topologies. The depth of the network conceptually is very similar to Feed-forward loops in Feed-forward PUFs, whereas the width of the network can be analogized to the size of XOR PUFs. Equation 3.5 describe a sample size lower bound in terms of the IPN model parameters, where m is the depth of the IPN network, and n is the width of the network. To be noted that we assume every single path within the network to be of width n and depth

m.

$$N \sim \frac{(m \cdot k + m) \cdot n + ln(\frac{1}{\delta})}{\epsilon} \tag{3.5}$$

For each IPN structure, we derive a empirical formula based on equation 3.5 by assuming a linear $y = ax + b$ relationship. The derived formula failed to match with the evolution strategies result due to the random nature of evolution strategies. The data points we collected from evolution strategies show a super-linear relationship between N and ϵ. Thus, we adopt the method proposed by Rhrmair et al. and modify the relationship to equation 3.6 when applying evolution strategies to match the superlinear relationship. c is a constant between 0 and 1.

$$N \sim \frac{(m \cdot k + m) \cdot n + ln(\frac{1}{\delta})}{\epsilon^c} \tag{3.6}$$

An IPN-based system requires much larger training set comparing to standard arbiter PUFs, XOR PUFs and Feed-forward PUFs of the same size. This can be observed when comparing equation 3.5 to the lower bounds proposed in [56]. The conclusion is also confirmed by our experimental results shown in Section 3.5.

3.6.2 Reconfiguration Logic

The reconfiguration is performed by reconfiguring interconnections between IPN nodes. Since each edge is controlled by a shuffler, the interconnect can be reconfigured by changing the configuration vectors in the shufflers.

We use a counter to count how many CRPs have already been generated, and we compare it with a predefined threshold. To be more conservative, we set a reconfiguration threshold Θ to a number that is smaller than the theoretical lower bound of sufficient CRPs using equation 3.7. Instead of assuming the network has the maximum width n on every level (Equation 3.5), we assume every single path within the network to be of minimum width n' and depth m. Once Θ has been reached, a random number generator generates a new set of

configuration vectors, and feed them to the IPN shufflers.

$$\Theta = \frac{m \cdot k \cdot n' + ln(\frac{1}{\delta})}{\epsilon} \tag{3.7}$$

3.6.3 Protecting Reconfiguration Logic

The configuration vectors are essential in such a way that an adversary could potentially use the information to collectively train a model over different sets of samples even if each sample size is intentionally limited below our calculated lower bound. An intuitive idea is to store all configuration vectors in non-volatile memories that lay below PUF delay wires so that damaging any one of those wires would change the PUF, rendering the adversary's attack useless [59]. However, in our reconfiguration logic, a new set of configuration vectors are provided by a random number generator or the user, which is not secure if the adversary has physical access to the device as we described in our assumption. We take a step forward by securing these configuration vectors using existing IPN nodes in the system so that the real interconnection remains hidden. We propose to encrypt the user-provided configuration vector to an IPN node in the previous level. The configuration vectors for all shufflers between level i and $i+1$ depend on the encrypted result of the user provided configuration bits using the IPN nodes from level 1 to level $i-1$, for $i > 1$. To note that we use IPN nodes in the previous levels to encrypt shuffler configurations to reduce the correlation between the output of an IPN node and it's immediate shufflers. An illustrative example is shown in Figure 3.3.

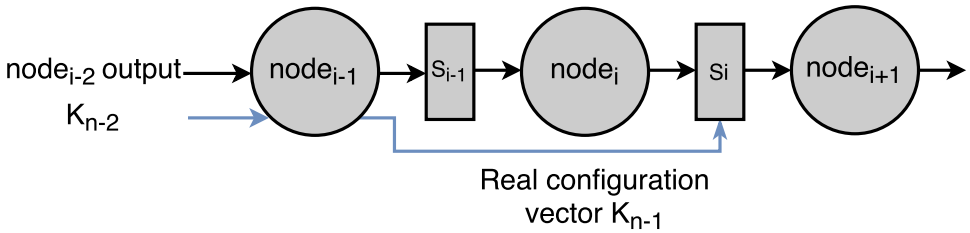


Figure 3.3: Encrypting random configuration logic using existing IPN nodes. K_j is the configuration vector encrypted by a chain of nodes from $node_1$ to $node_j$

All shufflers within an IPN are initialized at the beginning of the reconfiguration process.

The random or user provided configuration vectors are passed to the nodes in the first level propagate along the network to configure the remaining shufflers in the later levels. The shufflers between the first and second layer have a non-reconfigurable static connection. The attacker, even with physical access to the IPN device, cannot obtain information on actual configurations in shufflers since without characterize each IPN nodes. On the other hand, since the attacker cannot obtain enough training data given a specific configuration without knowledge of real configuration vectors in each shuffler.

3.7 Evaluation Results

In this section, we apply all the attack techniques we introduced in Section 3.5.

Our evaluation is conducted on both simulated models as well as implementations on a Xilinx Virtex-5 XC5VLX50T FPGA. Our simulation assumes a Gaussian distribution in all delays and no error in contrast to real distribution and real errors in the implementation. As a comparison, we compare different IPN setups along with standard arbiter PUFs, XOR PUFs, and feed-forward PUFs. For fairness considerations, we maintain the total number of PUF segments used in both simulation and implementation the same over different structures. Since we intend to prove that an IPN structure itself is more resilient against machine learning attacks, meaning it is much harder to predict using a machine learning model, we provided all PUF-based system discussed in this section with the same number of challenge-response pairs as well as same run-time/iterations. Our main focus is on single bit prediction rate even though IPN generates multi-bit outputs. To be noted that the modeling of both simulation and actual FPGA implementation was performed offline, meaning the training set and the test set of CRPs were collected before modeling. Querying the IPN is not allowed during the modeling process.

3.7.1 Logistic Regressions

In our security evaluation of IPN using logistic regression, we use standard gradient descent, IRLS, and RProp as the optimization method. In an attempt to model a simple IPN with

reconfiguration functionality disabled, the difference between all three optimization method is negligible.

We maintain the total number of PUF segments used in all settings to be around 1,024. For all architecture except standard arbiter PUF, the training set contains 30,000 CRPs, and the running time is set to unlimited. For each setting, we run 100 times and the simulated results showed in figure 3.4 is chosen from the best of 100 runs.

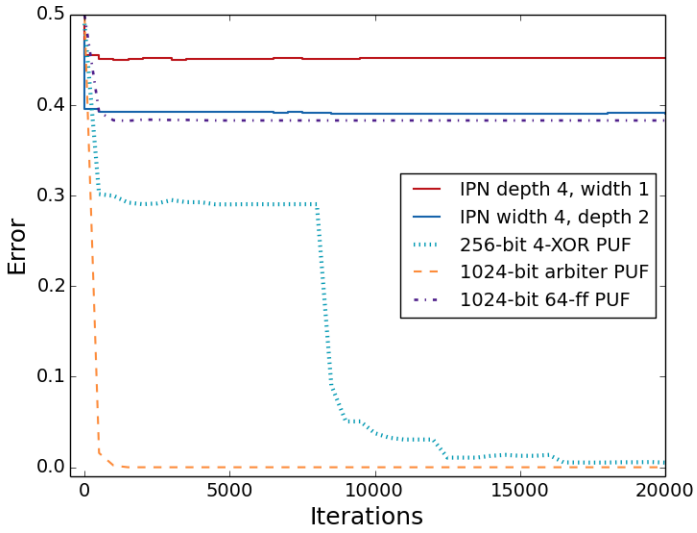


Figure 3.4: Best results in 100 logistic regression attacks using 30,000 CRP training set on five PUF-based systems. Error vs. iterations.

We observe that after around 20,000 iterations, the error for all five structures converges. Logistic regression attack is capable of successfully predicting 1024-bit standard arbiter PUF and 256-bit 4-XOR PUF with 99% accuracy. The simplest standard arbiter PUF architecture compromises immediately after the attack begins, where the 4-XOR PUF eventually converges after around 17,000 iterations.

IPN of depth 4 and width 1 provides better resilience against logistic regression comparing to IPN of width 4 and depth 2, and this result is observed in all 100 runs. IPN of width 4 and depth 2 on the other hand, shows the very similar result with a forward arbiter PUF with 1024 stages and 64 feed-forward loops. All three cases were allowed to run until time-out at 100,000 iterations, which roughly takes 7 days for each run.

Based on the result, we believe it is safe to conclude on two observations. (1) Feed-Forward loops provide excellent resilience against logistic regressions because the internal dependency introduced by feed-forward loops makes the model of the whole architecture no longer differentiable. Any attack methods that take advantage of linear separable or differentiable models would be inefficient or not work at all. (2) A deeper IPN provides better protection against logistic regression attack. The multiple layers of dependencies make the system even more complicated so that gradient information is of no help regarding modeling such a system.

3.7.2 Evolution Strategies

In our security evaluation of IPN using evolution strategies, we use both canonical versions, respectively $(\mu/\rho, \lambda)-ES$ and $(\mu/\rho+\lambda)-ES$ with and without the mini-batch style of fitness evaluation method implemented based on [61]. The difference between the two versions of evolution strategies is that $(\mu/\rho + \lambda) - ES$ takes the parent population into consideration during selection process where $(\mu/\rho, \lambda) - ES$ only selects from the offspring population. Both canonical versions of evolution strategies were applied to all investigated PUF-based systems, each with 100 runs. The best results among the 100 runs are shown in figure 3.5.

IPN of width 4 and depth 2 is the most difficult for evolutional strategies attack to tackle, while the 1,024-bit arbiter PUF with and without feed-forward loops performs the worst. A general trend for all curves in figure 3.5 is that the speed of progress is slowing down. The probability of observing a huge decline regarding errors dramatically decreases as the total number of generations increase. We can observe that for IPN of width 4 depth 2, the curve is almost flat after 30,000 generations.

Based on the result, we believe it is safe to conclude on two observations. (1) Nonlinear logic functions like XORs dramatically increase the difficulty for evolution strategies attack models, whereas feed-forward loop provides limited additional complexity against evolution strategies. (2) A wider IPN provides better protection against evolution strategies attack. This conclusion is not surprising as a wider IPN introduces more XORs which provides much

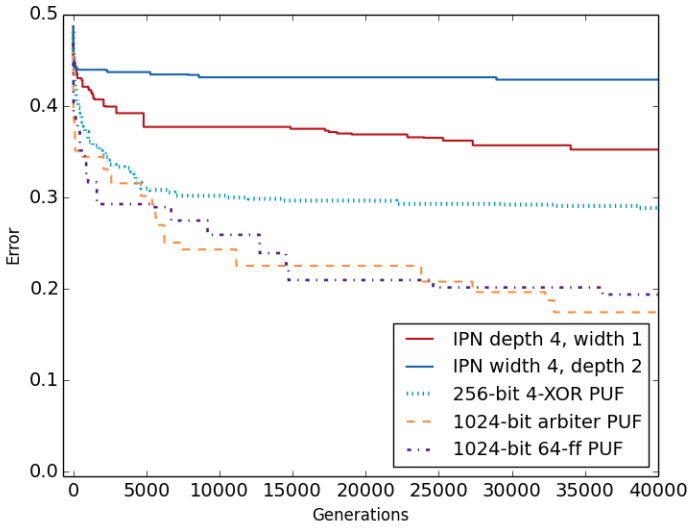


Figure 3.5: Evolution strategies attack result using 30,000 CRP training set on five PUF-based systems.. Error vs. iterations.

more nonlinearity. Despite the nonlinearity introduced by XORs, we still observe that an IPN of width 1 and depth 4 still performs better than 256-bit 4-XOR PUF when provided with the same training set.

3.7.3 Multilayer perceptron

In our security evaluation of IPN using MLP, we reform the task as a binary classification problem. We experiment with different network configuration parameters implemented using Keras [62]. After some experiment, the following setup showed in Table 3.1 provides the best results and speed.

Layers	Layer type	Units	Activation
1	Dense	m_1	ReLu
...	Dense	...	ReLu
n	Dense	m_n	ReLu
n+1	Dense	2	Softmax

Table 3.1: MLP parameters when modeling IPN. n is the depth of the network and m_i is the total number of PUF segments on the i-th level.

The loss function used is binary cross entropy and we use Adam as the optimizer. We

set the number of epochs to a constant of 100 so that we have a total number of CRPs/100 as our batch size.

Comparing to logistic regression and evolution strategies, an MLP does not necessarily require details in PUF architecture; instead, it treats the entire PUF as a black box and learns the function based on only input and output. Table 3.2 shows the result of applying MLP modeling to all discussed PUF systems.

Architecture	Training Acc.	Test Acc.
IPN depth 4 width 1	99.93%	50.18%
IPN depth 2 width 4	99.77%	50.33%
256-bit 4-XOR PUF	99.73%	96.02%
1024-bit arbiter PUF	99.99%	98.28%
1024-bit 64-ff PUF	99.99%	95.68%

Table 3.2: Deep neural network attack results.

MLP with the structure described in section 3.5 is capable of fitting 30,000 CPRs with above 99% training accuracy, and can predict 256-bit 4-XOR PUF, 1024-bit arbiter PUF and 1024-bit 66-ff PUF with above 95% test accuracy. However, it ran into an overfitting problem when modeling IPNs. After attempting various overfitting prevention techniques including regularization layers and dropouts, we conclude that the root of the overfitting problem is insufficient training samples. IPNs is more complicated comparing to other PUF systems. Thus, given a non-sufficient training dataset, the overfitting problem is more severe. When provided with a much larger dataset (5,000,000 CRPs in simulation), the test accuracy can be boosted to 86.49% for IPN with depth 4 width 1 and 78.01% for IPN of depth 2 width 4. When applying the same training set, the test accuracy converges at 54.77% and 62.54% respectively, much lower than MLP attack results.

3.7.4 Other Machine Learning Algorithms

AutoML is still under development, yet it provides promising results compared to MLP attacks in terms of modeling PUF-based systems. We provided only raw CRPs to the auto-sklearn module, and the results for all tested architectures are shown in Table 3.3.

Architecture	Best algo.	Test Acc.
IPN depth 4 width 1	Decision tree	64.87%
IPN depth 2 width 4	Decision tree	67.27%
256-bit 4-XOR PUF	K-NN.	83.88%
1024-bit arbiter PUF	Multinominal NB	89.55%
1024-bit 64-ff PUF	Multinominal NB	72.33%

Table 3.3: Auto-sklearn modeling results on raw CRPs.

In general, the best classifiers for IPNs are decision-tree classifiers, which is capable of predicting over 65% of CRPs in the test set. XOR PUFs, arbiter PUFs, and Feed-forward PUFs are much easier to model since auto-sklearn is capable of finding a classifier (such as K-nearest neighbor or multinominal naive Bayes classifiers) that successfully predicts the test set CRPs with accuracy over 70%.

3.7.5 Implementations on FPGA

Two differences distinguish a simulated PUF-based system and real implementations. (1) The delays located within a PUF implementation do not necessarily follow a certain distribution, whereas in simulations we assume a Gaussian or uniform distribution for all delays in the PUFs. (2) Real-world implementations suffer from stability issues. Since PUFs are extremely sensitive to environmental factors such as temperature and voltage, the response generated by the same PUF might not be consistent when providing the same challenge multiple times.

We repeated all experiments on data collected from FPGA implementations of all discussed architecture. The best results of both simulation and implementation are shown in Table 3.4.

Based on the log provided by Xilinx System Monitor, the largest variations in the core temperature and the core voltage are 2°C and 2.78% respectively during the entire CRP collection process. The environmental variations lead to 10.96% of CRPs being unstable in the training set. By applying ECC on IPN, the amount of instability drops significantly to 3.67%. The stability issue along with random delays in the hardware reduces the prediction

Architecture	Logistic Regression	Evolution Strategies	MLP	Auto ML
IPN depth 4 width 1	53.62% / 53.68%	67.57% / 67.18%	61.98% / 51.59%	64.87% / 50.04%
IPN depth 2 width 4	60.56% / 58.23%	57.20% / 56.33%	62.08% / 54.20%	67.27% / 52.16%
256-bit 4-XOR PUF	98.86% / 95.01%	79.12% / 76.02%	86.30% / 61.82%	83.88% / 67.80%
1024-bit arbiter PUF	99.99% / 96.57%	99.62% / 98.28%	89.55% / 80.85%	89.55% / 73.22%
1024-bit 64-ff PUF	62.27% / 58.29%	97.62% / 95.68%	72.33% / 67.98%	82.33% / 68.58%

Table 3.4: Best prediction accuracy on different PUF architectures using machine learning algorithms out of 100 runs. Each cell contains simulated/implementation result.

accuracy in almost all cases. MLP and autoML models suffer the most from the instability in the training set while the reduction in evolution strategies attacks are minimal.

Regardless of network depth and width, IPN stays robust against all proposed attacks, while XOR PUF and Feed-forward PUF can be modeled using different machine learning algorithms. Standard arbiter PUF, on the other hand, can be accurately modeled by all attack methods.

3.7.6 Implementation Result

We implemented a 64-bit reconfigurable IPN of depth 4 and width 4 (as shown in Figure 3.6) on a Xilinx Virtex-5 board. According to our derived formula on the sample complexity of IPNs, the sufficient number of CRPs required to predict a single bit response with 95% accuracy is 716,703 CRPs. We set the reconfiguration threshold to 358,350 CRPs, and we collected 1,000,000 CPRs (with duplications) as our training set. Table 3.5 shows the prediction accuracy of 10,000 test challenges with and without reconfiguration functionality.

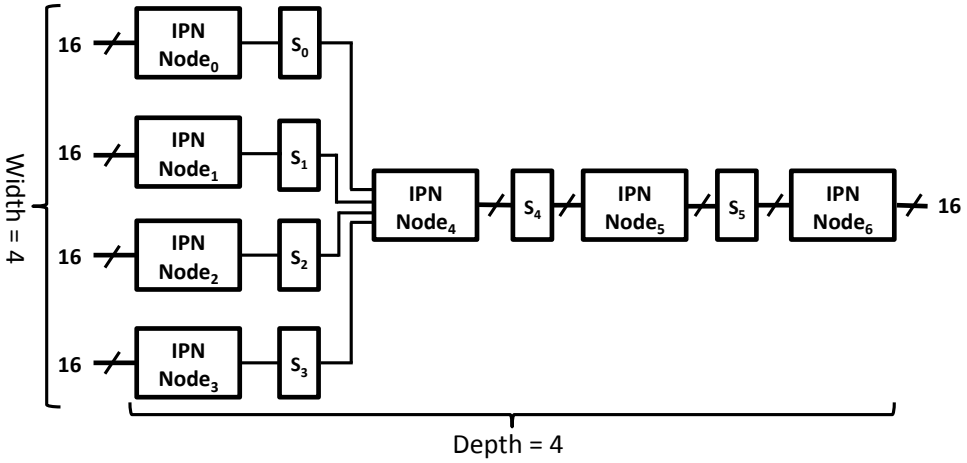


Figure 3.6: A 16-bit IPN network with four nodes on the first level, and has 4 levels.

We excluded Auto-sklearn from our experiment as it is extremely slow when handling very-large data. We observe that without reconfiguration enabled, no attack can reach the theoretical 95% of accuracy. We believe that in addition to the complexity of the IPN structure, the instability in the implemented PUFs increases the difficulty to accurately modeling an IPN. When reconfigurability is enabled, MLP performs the best due to its

Attack Method	w/out Reconfig.	with Reconfig.
Logistic Regression	64.62%	53.19%
Evolution Strategies	72.13%	49.99%
MLP	80.64%	51.89%

Table 3.5: Prediction accuracy of three attack methods. Results collected from a 100,000 test set. Logistic regression: 14 days; evolution strategies: 250,000 generations, 14 days; MLP: 18 hours.

Resource	With Reconfig.	Without Reconfig.
LUTs	2,102	2,001
Occupied Slices	1,227	1,157
Flip-flops	1,525	1,399

Table 3.6: Area overhead for implementing a IPN shown in Figure 3.6.

ability to quickly adapt to the new labels. Evolution strategies perform the worst because at each IPN reconfiguration all selected populations need to re-adapt the new *fitness* function and the algorithm has to learn from scratch again.

The area overhead for implementing the IPN with and without reconfiguration functionality is shown in Table 3.6. The reconfiguration mechanism uses additional 5.04% additional hardware; in return, it provides efficiently reduces the prediction rate by 11.43% at worst and 30.65% at best. If all our assumption holds, modeling a large IPN with reconfigurability is practically impossible.

3.8 Chapter Conclusion

We have carefully studied an interconnected PUF network structure that connects PUFs to build a network in this chapter. Our simulation and implemented results show that the IPN has a complex structure so that it enables itself to stay robust against not only traditional PUF modeling methods like logistic regression and evolution strategies but also to the state-of-the-art methods like deep neural networks and autoML.

To eliminate the possibility of being modeled with a large training set, we propose to make an IPN reconfigurable by shuffling the interconnections between IPN nodes. Before an

adversary can collect sufficient CRP sets for training purposes, the IPN reconfigures itself so that the attacker would not be able to obtain enough information on the IPN. To avoid storing the configuration vectors, we propose to use another set of PUFs to protect the configuration vectors. Our experimental results indicate that no investigated attack could accurately model an IPN. The single bit prediction accuracy for all attacks, when provided with a training set larger than the theoretical lower bound and 14 days of time, is as low as 53.19% in the worst case scenario.

CHAPTER 4

Optimizing PUFs with Evolution Strategies

4.1 Motivation

Security primitives and protocols are essential to embedded systems. Classical cryptography-based primitives rest on the concept of secret keys, under the assumption that the secret key can be securely preserved or completely hidden from adversaries. However, such security assumption cannot be guaranteed in reality. Various physical attacks including but not limited to invasive probing, side-channel attacks easily lead to secret key leakage and system compromise.

The physically unclonable function (PUF) as a unique hardware cryptographic primitive has the natural advantage of being resilient against secret key leakage. On the one hand, PUFs do not require storage of secret keys in the digital form, but instead in the form of nanoscale physical structure so that traditional attacks are inefficient; on the other hand, PUFs depend on uncontrollable process variations during the manufacturing process so that duplicating a PUF is nominally impossible.

APUF is a popular type of PUF which utilizes delay differences in transistors to produce chip-unique outputs. Theoretically, random delay variations in hardware allow APUFs to generate highly random outputs. In practice, however, high randomness cannot be achieved without a randomness booster because the unpredictable and uncontrollable nature of process variation creates unbalanced delay path routing in APUFs. Unbalanced delay paths potentially lead to symptoms such as unequal 0/1 frequencies in the outputs, repeated patterns, etc., which greatly compromises the security properties of APUFs. Two main causes lie at the root of the problems: (1) Some parts of the APUF generate extremely large delay

differences between two delay paths. The delay difference is so large that it dominates over the entire APUF so that it is more likely for the output of the APUF to be in favor of a specific delay path. (2) During the manufacturing and aging process of the hardware [63], certain bias can be introduced to the system that alters the behavior of the APUF, leading to randomness problems.

APUFs outputs are also relatively unstable when environmental factors, such as supply voltage and temperature, vary. As Zhou et al. reported when measuring and evaluating 1 million CRPs in arbiter PUFs, only 79.8% of CRPs are stable for a single bit [64], making the adoption of arbiter PUFs in any applicable security applications unreliable. The instability of APUF output is usually induced by small delay differences between a pair of delay path segments. As slight variations in environmental factors change transistor delays in a non-uniform manner, small delay differences could be easily altered. These small delay differences could easily add up and result in inverted outputs.

4.2 Technical Goals and Contributions

To overcome the limitations and to realize the full potential of APUFs as a basis for the security of lightweight systems, we propose to explore an optimal method to connect APUFs so that we can significantly improve or even overcome the natural weakness of APUFs. In this chapter, we utilize the IPN structure that interconnects APUFs in a network introduced in the previous chapter. We observe that a subset of all connection configurations inside an IPN could generate outputs that meet specific randomness or stability requirements. To our best knowledge, our work is the first effort to optimize the security properties of arbiter PUFs utilizing the IPN structure. In this chapter, we make three contributions:

- We show that different connections in an IPN could result in different output quality. A good configuration could enable an IPN to generate highly random and stable outputs that meet specific user requirement.
- We have proposed an evolution strategy method that is capable of effectively finding

a network configuration generates outputs with high randomness. Our experimental results show that our ES algorithm improves NIST randomness passing rate by 220.8% comparing to an unoptimized configuration and outperforms standard search algorithm an average of 8.5% with faster speed.

- We show that an optimized IPN provides highly stable outputs.

4.3 Related Work

4.3.1 PUF Randomness

Many attempts have been made to improve PUF randomness in the literature. O'Donnell from MIT proposed to use PUF as a hardware random number generator (RNG) [65]. Direct use of PUF generates outputs with mediocre randomness. Thus they proposed to use Von Neumann correction to enhance the randomness. Maiti et al. proposed to combine a delay-based PUF and jitter-based RNGs, where a delay-based PUF can be used to extract chip-unique signatures and volatile secret keys, and the RNGs are used for generating random padding bits and initialization vectors [66]. All past efforts use additional random boosters to help to improve the randomness of PUF whereas our work improves output randomness by configuring the connection of PUFs without any external resources.

4.3.2 PUF Stability

The stability problem of arbiter PUFs has been well recognized for decades. As Zhou et al. pointed out in the study of 1 trillion CRPs, instability is a huge weakness in multiple variations of APUFs [64]. Stable CRPs in a no XOR single-bit PUF (0.8∼1.0V, 0∼60 °C) only count as low as 80% of entire CRP space where a 10-XOR APUF has only 0.0028% of all CPRs being stable over a study of 1 million CRPs. A large number of new APUF designs that address the stability problems have been proposed in the past decades such as [67] [68] [12]. However, most of these PUFs are weak PUFs with a small number of possible CRPs and require an external output stabilizer or error correction code to ensure the output

stability [69]. Dodis et al proposed to use fuzzy extractors and large helper data to achieve extremely low error (10^{-9}) [70]. Pedersen et al. proposed to use the predetermined syndrome of a PUF stored in non-volatile memory to stabilize PUF outputs [71]. Most recently, Yan et al. proposed a novel mechanism to extract stable mappings of PUFs without the help of error correction and is able to tolerate 0.08% of errors in outputs [72]. In this chapter, we take a different approach to eliminate as many unstable segments as possible by actively searching for an interconnection that provides the best stability in APUFs.

4.4 Preliminaries

4.4.1 APUF Model

We propose to construct an IPN using standard delay-based APUFs as building blocks. An n-bit APUF takes an n-bit challenge as input and produces a 1-bit response as output. An n-bit vector (challenge) is provided to configure two nominally identically paths are generated along the APUF. Each challenge bit controls whether the pair of paths should swap positions within a PUF segment. To retrieve a response, an impulse signal is fed into the system to excite both paths simultaneously. Because of the uncontrollable process variation, the signal traveling along one of the two paths will reach the arbiter earlier, generating corresponding output.

Assuming an APUF maps a set of n-bit challenges C to corresponding response set R. We assume no delays on the connection wires and all delays are contributed by the APUF segments. Given a specific challenge $c \in C$, the ith APUF segment generates a pair of delays with delay difference of Δd_i^c. The corresponding response $r \in R$ can be mathematically represented as Equation 4.1:

$$r = \begin{cases} 0 & \text{if } \sum_{i=1}^{n} \Delta d_i^c > 0 \\ 1 & \text{if } \sum_{i=1}^{n} \Delta d_i^c < 0 \end{cases} \tag{4.1}$$

4.4.2 IPN

We have introduced IPN in the previous chapter, and as described, an IPN consists of nodes and edges. We make no modifications to the shufflers previously introduced; however, in this chapter, we assume that all IPNs nodes are homogeneous.An n-bit IPN node of size m consists of m n-bit APUFs. If $m = n$, a node is denoted as a *homogeneous node*, otherwise it is denoted as a *heterogeneous node*. The size of an IPN node is the total number of APUFs running in parallel. A demonstrative diagram of an n-bit homogenous node is shown in figure 4.1. An IPN node takes an n-bit vector as the challenge and generates n 1-bit responses. All APUFs within the same IPN node share the same challenges.

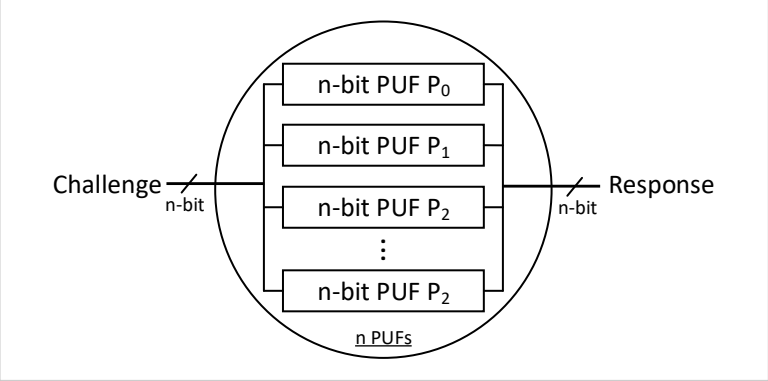


Figure 4.1: An n-bit homogenous node.

4.5 IPN Randomness

An IPN with direct connections, similar to a standard APUF, does not provide results that meet randomness requirements. Take 0/1 frequency requirement for example. If the j_{th} APUF in $node_i$ has much larger probability of producing a "1" as output, all j_{th} segment in $node_{i+1}$ would prefer one delay path over another. If unfortunately, an affected segment generates dominating a delay difference, the 0/1 frequency balance of the APUF in $node_{i+1}$ that contains such segment could be damaged.

To avoid such damage, we first propose to use brute force random search method to try out different connections until an acceptable one is found. We propose to append a shuffler

between every two IPN nodes to shuffle the input-output mapping as shown in figure 4.2. Each shuffler uses a configuration vector to map each output bit from the previous nodes to each input bit of the next node. Here, we define a configuration of an IPN as a collection of all configuration vectors for all shufflers in the IPN. A good configuration is defined as a set of configuration vectors that enables the IPN output to meet certain randomness requirement.

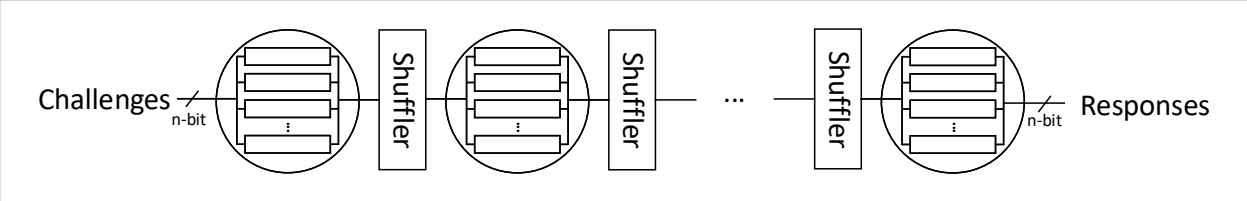


Figure 4.2: An example of a simple IPN chain with shuffled connection between nodes.

We ran simulations on 100 different IPN instances with the simple IPN chain structure. Each IPN has two 32-bit homogenous nodes. We use multiple tests from NIST test suite [73] as our randomness scoring function. Each test is modified to show the quality of IPN output regarding a specific randomness property. For a given configuration, a sample of 10,000 bits is collected before evaluating it using the randomness scoring function. The maximum number of configurations to test on each IPN instance is set to 100. Table 4.1 shows the success ratio of passing a specific random test of all IPN instances.

Test	Direct connection	100 Configs.
Frequency Test	32%	89%
Run Test	14%	76%
Linear Complexity	98%	100%
Approx. Entropy	1%	15%

Table 4.1: Success ratio of passing a NIST test for 100 IPN instances with two 32-bit homogenous nodes. The maximum number of configurations allowed for each IPN is 100, each configuration collects 10,000-bit result.

The result shows that allowing exploration of multiple IPN configurations significantly increase the chance that an IPN is capable of passing a specific randomness test. This result is intuitive and straightforward since delay characteristics in APUFs are unpredictable, certain combinations of such APUFs provide better randomness results than others. For an IPN, having the opportunity to explore more configurations naturally increase the chances of

finding an arrangement of APUFs that provides highly random results.

4.6 IPN Stability

Stability is an essential factor of IPN, especially because all APUF segments are interconnected in an IPN. Therefore, a single bit change in the first few nodes could cause significant avalanche effect to the final output of IPN. We also ran simulations with the simple IPN chain structure. We model the relationship between transistor delays and environmental variations according to equation 4.2 described in [74]. Here, k_{tp} is the delay-fitting parameter, C_L is the sum of the intrinsic capacitance and the load capacitance, V_{dd} is the supply voltage, n is the subthreshold slope, μ is the mobility, C_{ox} is the oxide capacitance, L is the effective channel length, W is the gate width, ϕ is the thermal voltage ($\phi = \frac{kT}{q}$), k_{fit} is a model-fitting parameter and IC represents the inversion coefficient. We fixed all parameters as derived from a curve-fitting Spectre simulation results for a 65-nm CMOS technology as described in [74] except V_{dd} and ϕ. We simulate voltage and temperature variations by sample the value of V_{dd} and ϕ from a normal distribution where the mean and the variance is collected from real chip measurement.

$$t_p = \frac{k_{tp} \cdot C_L \cdot V_{DD}}{2 \cdot n \cdot \mu \cdot C_{ox} \cdot \frac{W}{L} \cdot \phi_t^2} \cdot \frac{k_{fit}}{IC} \tag{4.2}$$

We run our simulation on a single PUF, a 10-XOR APUF and a four-node IPN chain with 1 million CRPs and we obtained 24.232%, 99.985% and 64.312% of unstable CRPs. The first two observations are extremely similar to the results reported by [64].

4.7 IPN Optimization Algorithms

Knowing that exploration for a good configuration could greatly improve the possibility for an IPN instance to pass a random test, we thus propose two algorithms to configure a given IPN so that it can produce outputs that meet certain randomness requirements in this

section.

4.7.1 Random Search

Based on our observation in section 4.5, we first formally propose a random search algorithm as our baseline method in Algorithm 1.

Protocol 1 Random Search in IPN

Require: (1) An *IPN* with n nodes and $n-1$ shufflers. (2) A set of configuration vectors $V = \{V_1, V_2, ..., V_{n-1}\}$ that configures each shuffler. (3) A randomness scoring function $f(x)$ where x is a sampled binary string of size s. (4) A passing threshold Θ. (5) Total number of iterations L.

Ensure: A set of configuration vectors $V = \{V_1, V_2, ..., V_{n-1}\}$ that enables *IPN* to produce outputs that passes $f(x)$. If no such set of vectors can be found in L iterations, returns nothing.

while $L > 0$ **do**
  Sample a binary string x of size s using *IPN*.
  if $f(x) < \Theta$ **then**
    for each V_i in V **do**
      Randomly shuffles V_i
    end for
  else
    return V
  end if
  $L = L - 1$.
end while
If no configuration is found in L iterations, return nothing.

The random search algorithm is a simple brute force algorithm that explores many configurations until a good configuration is found. If no good configuration can be discovered after a large number of iterations, the possibility of finding such configuration is assumed to be low. It is also possible that dominating segments or hardware biases in some APUFs make it impossible to construct a desired IPN. Thus, an upper bound is set as the maximum number of iterations allowed. If no desired configurations are found after the maximum number of iterations, the algorithm returns nothing.

4.7.2 Evolution Strategy

Random search shows significant improvement comparing to direct connection. However, the searching process is entirely random so that the speed of finding a good configuration is based on luck. Here we present an ES algorithm that does not necessarily test a completely different configuration when randomness objective is not achieved, but instead, reconfigures only a portion of the configuration based on how well current configuration performs. The essential idea is not to recreate a new set of configuration vectors but to improve on the current configuration. The ES algorithm is based on the observation that minor modification on a fair configuration vector sometimes leads to better or even excellent results. The detailed algorithm is described in Protocol 2.

The ES algorithm, like random search, uses a random configuration as a start. A swap ratio index is used to determine what portion of the configuration should be shuffled to achieve the desired randomness goal. If the result keeps improving, we gradually decrease the swap ratio to make as few modifications as possible at each iteration so that past progress could be preserved. If the results are worsening, we increase the swap ratio, and in an extreme case, randomly shuffles all configuration vectors.

A key feature of our ES algorithm is the backtracking mechanism. If a shuffler configuration vector V already generates a fair result that is only slightly below expectation, we expect minimum modification could provide sufficient results. If multiple modifications to configuration vector V leads to decrease in the randomness score, we conclude that it is unlikely that slight modification to V provides desired results, so we backtrack to V and increases the swap ratio to 1.

4.8 Experimental Results on Randomness Improvement

We use the following experimental setup to study the impact of the maximum number of iterations, IPN node size, IPN chain length, and IPN connection methods. 1,000 different IPN instances were simulated. For each configuration, we collected $10,000$ bits of results.

Protocol 2 Evolution Strategy in IPN

Require: (1) An *IPN* with n nodes and $n-1$ shufflers. (2) A set of configuration vectors $V = \{V_1, V_2, ..., V_{n-1}\}$ that configures each shuffler. (3) A randomness scoring function $f(x)$ where x is a sampled binary string of size s. (4) A scoring function threshold Θ and a near-passing threshold $\theta < \Theta$ (5) A swap ratio $\alpha \in (0, 1]$, initially, $\alpha = 1$. (6) A backtracking counter $\beta = 0$ and a backtracking threshold λ. (7) Total number of iterations L.

Ensure: A set of configuration vectors $V = \{V_1, V_2, ..., V_{n-1}\}$ that enables *IPN* to produce outputs that passes $f(x)$. If no such set of vectors can be found in L iterations, returns nothing.

while $L > 0$ **do**
  Sample a binary string x of size s using *IPN*.
  if $f(x) < \Theta$ **then**
    if $f(x) > \theta$ **then**
      Decrease the swap ratio α, $\alpha > 0$.
      Save $f(x)$ and V as $f'(x)$ and V'.
    else
      if $f'(x) > \theta$ **then**
        $\beta = \beta + 1$
      end if
      Increase the swap ratio α, $\alpha \leq 1$.
    end if
    if $\beta > \lambda$ **then**
      V backtrack to V'
      $\beta = 0$, $\alpha = 1$
    else
      for each V_i in V **do**
        Select $\frac{1}{2}\alpha|V_i|$ connections in V_i, swap connections with another $\frac{1}{2}\alpha|V_i|$ connections.
      end for
    end if
  else
    return V
  end if
  $L = L - 1$.
end while
If no configuration is found in L iterations, return nothing.

The scoring function is a weighted composite function of results obtained from the NIST test suite [73], Diehard-1997 [75], Diehard-2009 [76] and Soto [77]. The result is normalized to [0,1]. We conduct our experiments with all NIST tests, but we only show the run test result since it accurately represents the majority of the NIST test. For ES algorithm, the scoring function pass threshold Θ is set at 0.99 and the near-pass threshold θ is set at 0.9. The

backtracking threshold λ is set at 10% of the maximum number of iteration. An instance passes the test if it achieves a score of at least $\Theta = 0.99$.

4.8.1 Number of Iterations

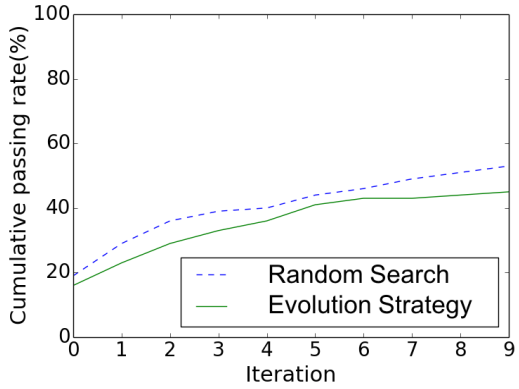
Intuitively, both random search and ES algorithm greatly benefit from a large number of iterations. For random search, more iterations provide a greater chance of finding a network configuration that could pass a given randomness test. For ES, more iterations grant more opportunities to improve from a mediocre configuration.

We conduct our experiment with five different maximum numbers of iterations, respectively 10, 50, 100, 500 and 1,000. For 1,000 IPN instances with two 32-bit homogenous nodes, the trace of the growing possibility of successfully discovering a configuration that is capable of passing the scoring function threshold is shown in Figure 4.3.

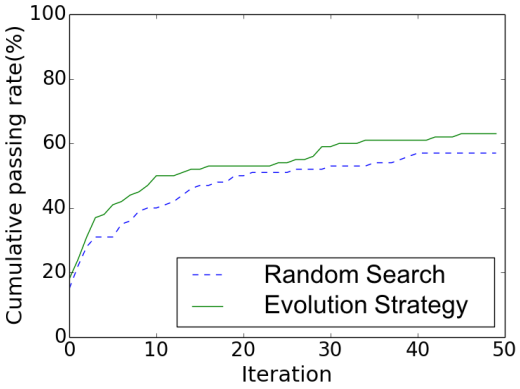
Max # of iterations	Random Search	ES
10	48.2%	44.6%
50	57.5%	68.6%
100	76.0%	82.8%
500	88.8%	89.5%
1,000	89.7%	92.1%

Table 4.2: Success ratio of passing NIST run test for 1,000 IPNs. Each configuration collects 10,000-bit result.

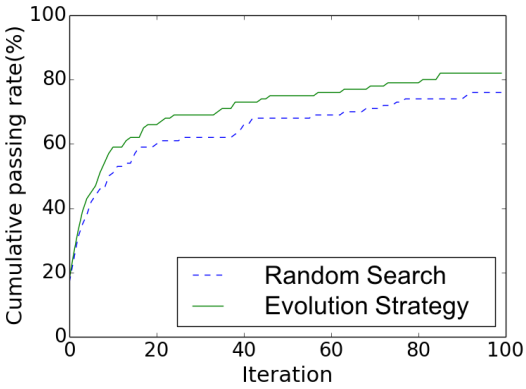
When the maximum number of iterations is small, random search shows better results. We believe that this is caused by the design of backtracking in the ES algorithm. A small number of iterations leads to frequent backtracking activity, which essentially provides no room for improvements in any configuration and many iterations are wasted on the backtracking process. As the number of iterations increases, ES outperforms random search. When the total number of iteration is 50, ES is capable of finding 11.1% more good configurations comparing to random search. As we increase the maximum number of iterations from 500 to 1,000, the results for both random search and ES gradually converge. The additional iterations not only provide very limited improvements in both algorithms but also fail to differentiate the efficiency between them. This is because a sufficient number of APUF



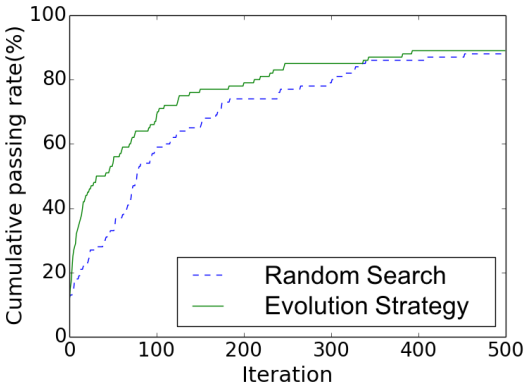
(a) Max number of iterations: 10.



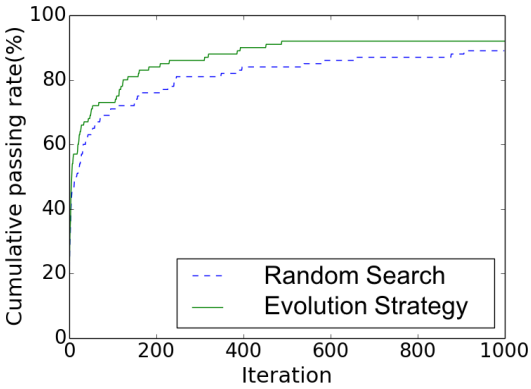
(b) Max number of iterations: 50.



(c) Max number of iterations: 100.



(d) Max number of iterations: 500.



(e) Max number of iterations: 1,000.

Figure 4.3: Cumulative percentage of 1,000 IPN instances that pass the NIST run test: random search vs. ES using various numbers of maximum iterations. IPN has a chain structure of two 32-bit homogenous nodes. Scoring function threshold: 99%, near-passing threshold: 90%, backtracking threshold set at 10% of maximum number of iteration. Results collected from 1,000 sets of IPNs.

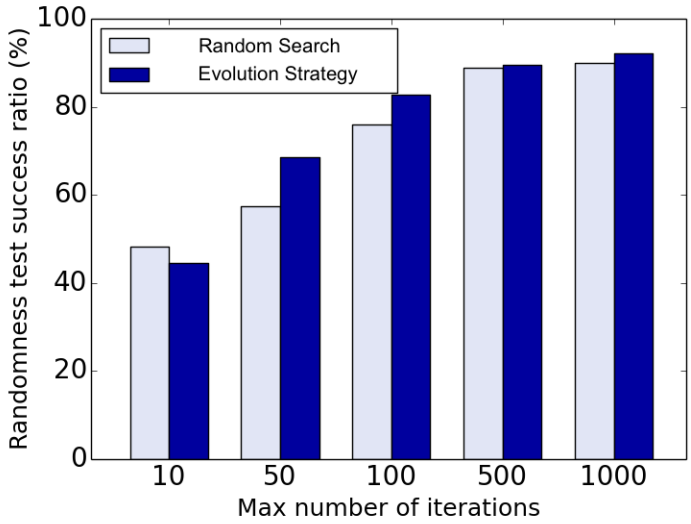


Figure 4.4: Percentage of 1,000 IPN instances that pass the NIST run test. Maximum iterations are set to 10,50,100,500 and 1000. Each configuration collects 10,000-bit result.

combinations have already been tested. According to the experimental result, the sufficient number of APUF combinations is less or close to 500; more iterations would result in very limited improvement.

In addition to finding more desired configurations, we also observe that the ES method use less time in discovering a near-optimal configuration when compared to random searches. Figure 4.4 shows the relationship between cumulative random test passing rate and the iteration number at which a proper configuration is found. At any given passing rate, the iteration number at which a proper configuration is found is lower for ES method comparing to random search method. At any given iteration, the ES method also presents a higher passing rate as well.

Based on the experiment result we can conclude that when the maximum number of iterations is too low, random search provides a higher chance of finding a good configuration. When the maximum number of iterations is too high, both ES and random search method can discover a similar number of proper configurations. When the maximum number of iterations is neither too high or too low, the ES method not only finds more desired configurations but also finds them faster.

4.8.2 IPN Node Size

The total number of possible configurations of a shuffler between every two IPN nodes depends on the size of the node. Two n-bit homogenous nodes can form a total of $n!$ different configurations. We conduct both random search and ES on four different node sizes to observe if both algorithms perform consistently over different node sizes.

We maintain a similar experimental setup as described above except we fixed the maximum number of iterations to 100. We vary the node size for both homogenous nodes: 16-bit, 32-bit, 64-bit, and 128-bit. The success ratio of passing NIST run test for 1,000 IPNs with different node size is shown in Figure 4.5.

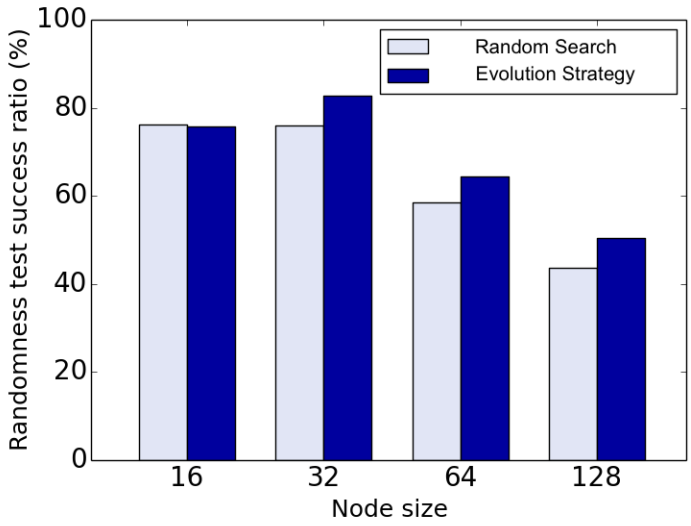


Figure 4.5: Percentage of 1,000 IPN instances that pass the NIST run test with different node sizes. Each configuration collects 10,000-bit result.

Interestingly both random search and ES show lower success ratio of passing NIST run test as the nodes size increases. An explanation for such phenomenon is that the search space is getting larger and larger, respectively 16!, 32!, 64! and 128! configurations. Since we fix the number of the maximum number of iterations allowed to 100, the proportion of explored configurations drops extremely rapidly. The decrease in the proportion of explored configuration would certainly lead to a reduction in the probability of finding a proper configuration.

4.8.3 IPN Structure

We now evaluate the performance of two algorithms on larger IPNs and more complex architecture.

4.8.3.1 Larger IPN

As the size of IPN chain grows, the search space for all configurations grows significantly as well. We first simulated larger IPN chains with three, four, five and six homogeneous nodes and applied both random search algorithm and ES algorithm on the network. The experiment was conducted on 1,000 IPN instances for each case. The size of each node is set to be 32-bit, and the maximum number of iterations is fixed to 100.

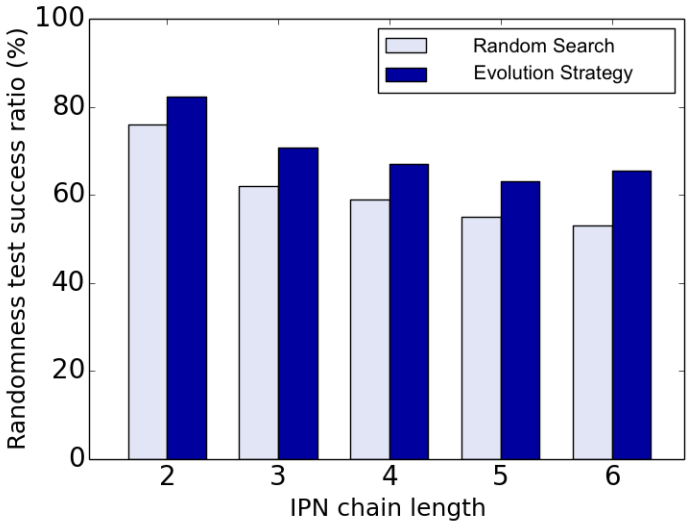


Figure 4.6: Percentage of 1,000 IPN instances that pass the NIST run test with different IPN chain lengths. Each configuration collects 10,000-bit result.

Figure 4.6 shows the experimental result of five different IPN chain length. We observe that as the IPN chain gains more length, both random search and ES methods results in a slight drop in passing ratio. The reason is similar to the situation explained in section 4.8.2. The additional IPN nodes increase the configuration search space, leading to a significant drop in the possibility of finding a proper configuration. Despite the drop in test success ratio, ES method outperforms random search by 16.8%, 17.8%, 18.2% 18.4% and 22.5%

respectively in five different IPN settings.

4.8.3.2 Other IPN Connections

In an IPN chain, each node forms a one-to-one connection with another node. We now investigate more complex network that contains one-to-many, many-to-one and many-to-many connections.

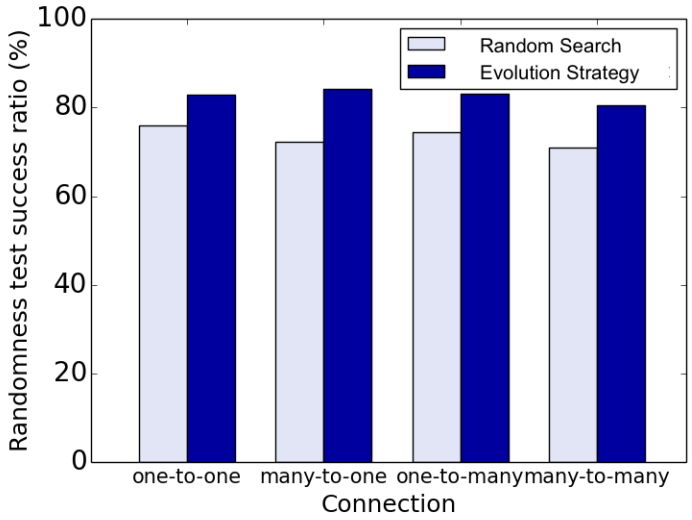


Figure 4.7: Percentage of 1,000 IPN instances that pass the NIST run test with different IPN structure. Each configuration collects 10,000-bit result.

Figure 4.7 shows the run test passing rate for one-to-one, one-to-many, many-to-one and many to many connections. We define the depth of the IPN as the smallest number of edges required from the input of the network to the output and all nodes at the same depth is denoted as a level. We fix the depths for all four different types of connections to 2 and the maximum number of iterations to 100. We observe that varies types of connections have little impact on the passing rate for both algorithms. A one-to-many connection is equivalent to two independent IPN chains where both random search and ES applies uniformly and independently on them. A many-to-one connection, on the other hand, is equivalent to one IPN chain since the XOR operation combines all nodes on the same level conceptually. A many-to-many connection is a combination of both one-to-many and many-to-one connec-

tion, thus can be abstract to IPN chains as well. In terms of applying both random search and ES algorithms, the type of connection does not affect the output randomness. The ES approach provides better output compared to random search over all four types of network connections.

4.8.4 NIST Test Results

Both random search and ES can handle multiple randomness tests at the same time. A large scoring function $G(x)$ with N tests can be created by calculating weighted sum of all tests as shown in Equation 4.3, where each test transforms to a scoring function $f_i(x)$.

NIST statistical test suite is commonly used to evaluate different aspects of the randomness of a binary string [73]. NIST test suite contains 15 different tests, so we created a overall scoring function $G(x)$ using Equation 4.3 where $f_i(x)$ corresponds to each individual test and N is set to 15. All scoring function outputs are normalized to $[0, 1]$. The passing threshold for $G(x)$ should be no less than the passing ratio of the highest threshold for all $f_i(x)$. In our experiment we set the passing ratio for all $f_i(x)$ and $G(x)$ to be 0.99. We evaluate both random search and ES methods on 1,000 IPN chain instances with two 32-bit homogeneous nodes.

$$G(x) = \sum_{i=1}^{N} \frac{1}{N} f_i(x) \tag{4.3}$$

Table 4.3 shows our experimental results for each test in the NIST suite as well as the overall passing rate. The overall passing rate is calculated as the possibility of passing all 15 tests. ES outperforms unoptimized configuration by an average of 220.8% and random search by an average of 21.86% in all test cases.

Both random search and ES algorithms have exploited the potential in generating highly random outputs in APUFs. By configuring the connections between IPN nodes, both algorithms seek to find the best combination of APUFs so that randomness problems caused by hardware and structural factors can be fixed and compensated.

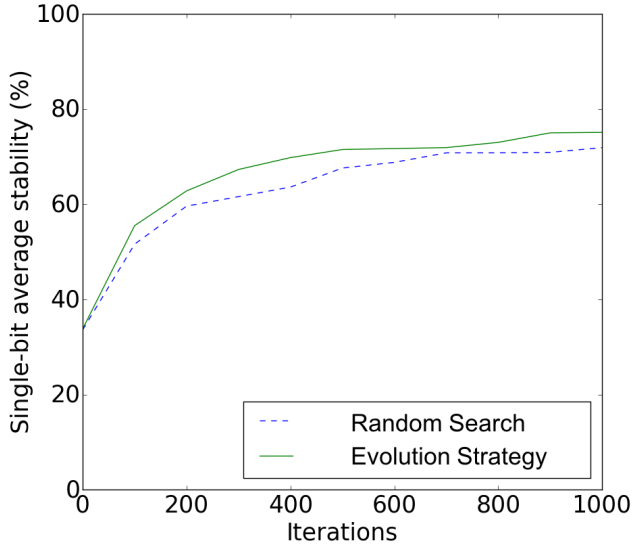
Statistical test	Unoptimized	Rand.	ES
Freq.	36.4%	88.4%	94.8%
Block Freq.	28.5%	82.5%	92.6%
Runs	15.7%	76.0%	82.8%
Longest Runs	13.6%	67.8%	81.1%
Binary Rank	54.5%	99.0%	99.5%
FFT	44.8%	89.7%	92.1%
Aperiodic	6.9%	14.6%	85.0%
Periodic	9.5%	15.4%	95.1%
Maurer	32.8%	79.2%	88.1%
Lin Complex.	85.0%	99.7%	100.0%
Serial	69.4%	80.2%	95.5%
Cusum	44.4%	86.4%	93.7%
Excursions	47.3%	88.4%	94.8%
Variant	37.9%	89.4%	92.7%

Table 4.3: Percentage of 1,000 IPN instances that pass the NIST test suite. The IPN has two 32-bit homogenous nodes. Each configuration collects 10,000-bit result, the maximum number of iterations is set to 1,000.

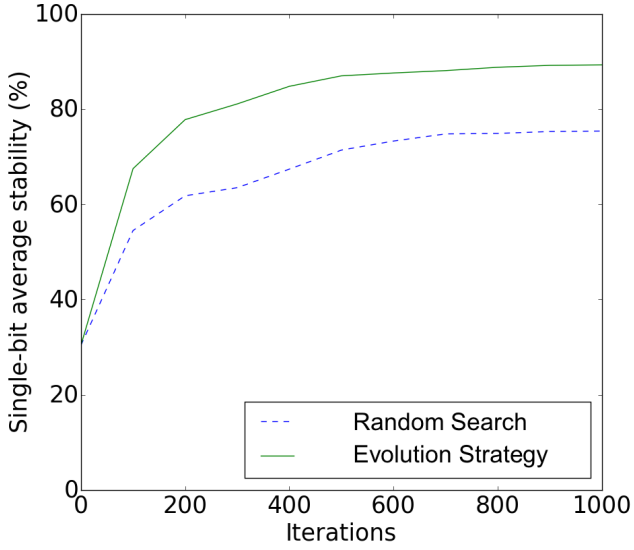
4.9 Experimental Results on Stability Improvement

We reuse the experimental setup in section 4.8 to improve the stability of IPN. Here we define stability as the possibility of observing a stable response over repeated observation of 100 times when provided a single challenge. We conduct our experiments on 1,000 IPN instances of IPN chains and complex IPNs. The result is presented in figure 4.8.

The complex IPN is constructed using a mixture of one-to-one, one-to-many and many-to-one connections. We fix the total number of IPN nodes to be four in IPN chains and a total number of levels in complex IPN to be four. We observe that both ES and random search approach significantly boosts the stability. Specifically, our ES approach outperforms random search in both IPN nodes and complex IPN as shown in Figure 4.8a and Figure 4.8b respectively. We noticed that the advantage of ES algorithm is greater as the number of many-to-one connections increases. An intuitive explanation is that as complex connections provide more stable candidates comparing to direct connections. Despite the significant improvement (130.3% in IPN chains and 167.74% in complex IPNs when comparing to random non-optimized IPNs, 22.62% in IPN chains and 25.38% in complex IPNs when



(a) IPN chain of 4 levels.



(b) Complex IPN of 4 levels.

Figure 4.8: Stability of 1,000 IPN instances of IPN chains and complex IPNs. Each IPN node is a 32-bit homogenous node. Both the IPN chain and complex IPN has 4 levels. Complex IPNs uses a mixture of one-to-one, one-to-many and many-to-one connections. Environmental variance: 0.8∼1.0V, 0∼60 °C.

comparing to baseline random search algorithm), additional error correction code might still be required to generate highly stable outputs for stability sensitive tasks for large IPN.

4.10 Chapter Conclusion

We have proposed a novel optimization algorithm on a type of PUF structure that interconnects APUFs to build a network in this chapter. We show that by configuring the interconnected network using our proposed evolution strategy algorithm, IPN is capable of overcoming randomness problems in APUFs as well as showing significant improvement in stability. Our proposed ES method outperforms the baseline method (random search) in both speed and quality. The advantage of our ES algorithm remains consistent over different IPN settings.

Our experimental results indicate that our ES algorithm outperforms unoptimized configuration by an average of 220.8% and standard search algorithm an average of 21.86% in all NIST randomness tests. We also observe that our method is also capable of improving output stability by 22.62% in IPN chains and 25.38% in more complexed IPNs.

CHAPTER 5

Content-driven Reconfigurable Injective Functions

5.1 Motivation

In the world of computer architectural design, cryptography provides the basis of many conventional security approaches. While the statement remains true, the rise of IoT technology has imposed challenges to the status of conventional security. The compact size and limited resources of many IoT devices cannot afford expensive cryptographic computations. Also, conventional cryptographic protection does not provide defense against attacks at the physical level. As more and more devices are deployed in untrusted environments, attackers could potentially gain valuable secrets through side-channel information. Thus, the need for low power, compact and side-channel attack resilient security primitives are highly desired in the era of IoT.

5.2 Technical Goals and Contributions

Facing these challenges, we propose an ultra-lightweight hardware-based security primitive: Content-driven Reconfigurable Injective Function (CRIF). A CRIF is a hardware function that maps an input to an output distinctively, and this mapping can be reconfigured using contents of historical data while preserving the distinctness of the function. We claim that the mathematical property of CRIF enables many security applications such as encryption and decryption operations. We believe that CRIF as a security primitive is advantageous for security tasks on energy and computation constrained devices compared to conventional cryptographic primitives in the following three aspects.

1. CRIF provides security at multiple layers. Unlike many cryptographic methods, CRIF does not hide a specific secret key in the implementation, similar to [78]. Thus, popular attacks at the physical level such as side-channel attack are incompetent.

2. CRIF is reconfigurable. The mapping between inputs and outputs can be rewired from time to time depending on historical secret message contents. Replay attacks and modeling attacks are extremely inefficient against reconfigurable systems.

3. Hardware implementation of CRIF is simple and energy efficient. CRIF can be efficiently implemented using a network of Lookup Tables (LUTs) on FPGA with a low area overhead. When performing tasks such as encryption or decryption, CRIF achieves at least 75.04% power savings comparing to some modern AES-based schemes.

5.3 Preliminary

5.3.1 Injective Functions

As the name suggests, the fundamental building blocks of CRIF are injective functions. Injective functions or one-to-one functions are a set of mathematical functions that maps every element in its domain to at most one corresponding element in it's co-domain. An injective function is mathematically denoted as shown in Equation 5.1. In the context of this chapter, we assume X and Y are finite domain set and finite codomain set of the injective function f. To be more specific, the distinctness property in injective functions is defined as Equation 5.2.

$$f : X \mapsto Y \tag{5.1}$$

$$\forall a, b \in X, f(a) = f(b) \implies a = b \tag{5.2}$$

5.3.2 Properties of Injective Functions

Injective functions have many useful properties for security tasks, among which we utilize two of them to build CRIF.

First, the composition of injective functions remains injective. If function f and g are injective, then function $h = f \circ g$ is injective. The property provides the basis of our reconfigurable design as we can composite multiple injective functions in different ways to create different input-output mappings without compromising the injective property.

Second, Injective functions can be made invertible. When the codomain of an injective function $f : X \mapsto Y$ is replaced by its actual range $J = f(X)$, the function became bijective (invertible). We show that by using bijective function f, either by itself or with it's clone $f' = f$, efficient and secure encryption/decryption can be implemented as $f^{-1}(f(X)) = X$.

5.4 Related Work

Security primitives are low-level algorithms that are used to build security protocols or security systems [79]. Traditional security primitives are purely cryptographic involving complicated mathematical operations [80]. However, these standard mathematics-based primitives are too costly to be used in lightweight mobile IoT devices.

5.4.1 PUF-based Security Primitives

Facing the challenges introduced by energy constraints, the rise of low power hardware primitives has provided new visions for security researchers. Physically unclonable functions (PUF), as a type of newly proposed low power hardware functions, have provided the basis of many energy efficient security primitives. PUF-based designs such as public PUF [81] and NanoPUF [82] have been proven to be adaptable and energy efficient in numerous security tasks with an emphasis on physical attack resilience and low power. However, PUF-based security primitives often suffer from stability issues and vulnerability against modeling attacks. Our design of CRIF, however, does not depend on noisy analog signals, therefore is

much more stable than PUF-based schemes. The reconfigurable nature of CRIF also renders modeling attack useless.

5.4.2 Efficient Implementation of Classical Cryptography

The advancement in efficient arithmetic circuits made energy-efficient hardware-based implementation of classic cryptography possible. Many hardware implementations of low power AES [83] [84] [85], RSA [86] [87] and elliptic curve cryptography(ECC) [88] [89] are proposed and adopted. Even though the energy efficiency of these cryptographic modules has been significantly improved, we show that our CRIF design is still more advantageous in terms of area and energy overhead.

5.5 Architecture

In this section, we formally introduce the hardware architecture of CRIF. We first show the implementation of our overall architecture. We then propose the LUT-based invertible injective function design. Lastly, we introduce our content-driven reconfiguration mechanism.

5.5.1 Overall Structure of CRIF

Since the composite of injective functions remains injective, we propose to connect multiple injective functions in a layer-by-layer network structure to increase overall complexity. Figure 5.1 displays the overall architecture of CRIF.

A typical CRIF takes an n-bit input and generates an n-bit output. A CRIF has a layered structure where each layer consists of a constant number of k-bit LUT-based injective functions. Multiple layers stacks on top of each other, meaning the output of the previous layer is used as the input of the next layer. We denote the total number of layers stacked together as the depth of CRIF and the number of LUT-based injective functions per layer as the width of CRIF.

To increase system complexity, we propose to insert a shuffler between every two layers.

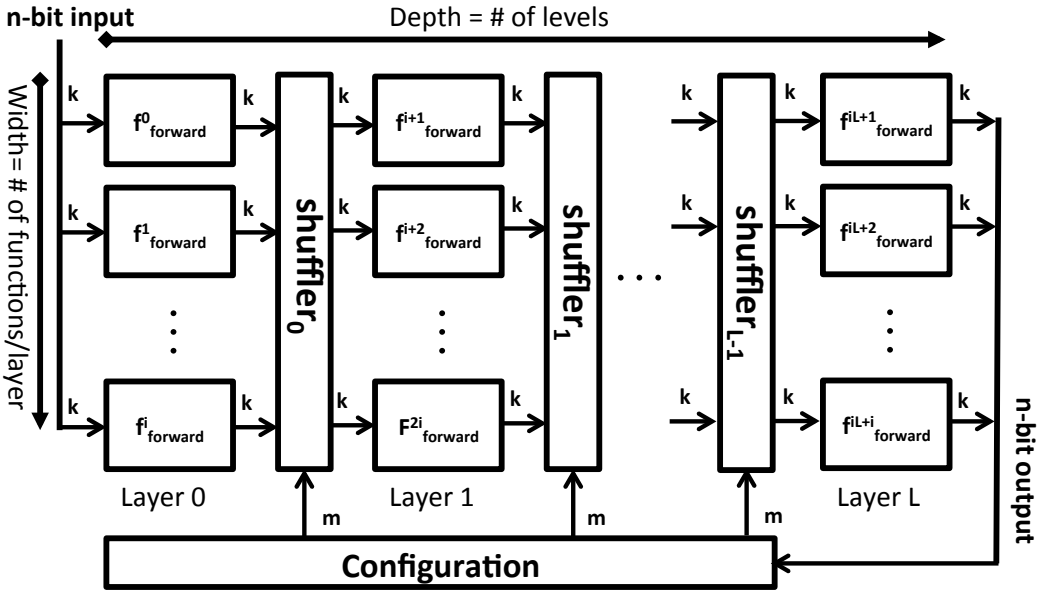


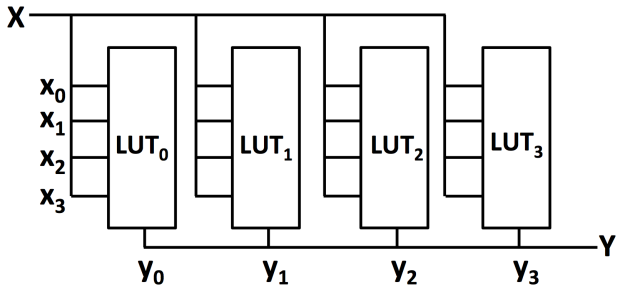
Figure 5.1: Overall architecture of our proposed CRIF.

The i-th shuffler SF_i locates between the i-th layer $Layer_i$ and $i+1$-th layer $Layer_{i+1}$. SF_i takes the output of $Layer_i$ and shuffle the order. The shuffling order solely depends on an m-bit configuration vector. Noted that the shuffle operation only changes the order of the input; the output length and bit frequency remain identical as the input. The shuffled input is then fed to $Layer_{i+1}$.

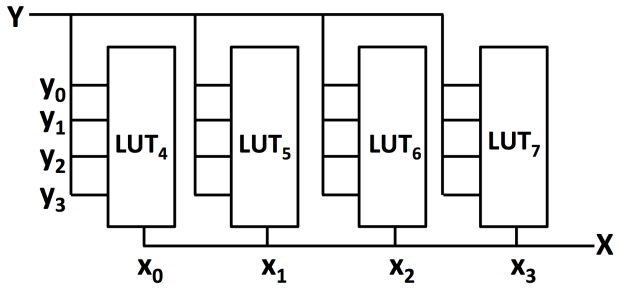
5.5.2 Injective Function

Injective functions, as the building block of CRIF, enforces one-to-one mapping from input x to output y and vice versa. The mapping from y to x should also be one-to-one to ensure the uniqueness of the inverse function. We propose to use LUTs to implement our injective functions.

To illustrate the design of our LUT-based injective functions, we show a motivational example in Figure 5.2. Figure 5.2a demonstrate the implementation of forward injective function $f_{forward} : X \mapsto Y$ using four LUTs where X and Y are 4-bit vectors. The corresponding backward function $f_{backward} : Y \mapsto X$ is shown in Figure 5.2b. The memory cells in each LUTs in $f_{forward}$ is carefully replaced so that requirements specified in Equation 5.2 is



(a) Forward injective function $f_{forward} = X \mapsto Y$.



(b) Backward injective function $f_{backward} = Y \mapsto X$.

Figure 5.2: A pair of four bit forward and backward injective function implemented using LUTs.

$f_{forward}$							
X				Y			
x_0	x_1	x_2	x_3	y_0	y_1	y_2	y_3
0	0	0	0	0	1	1	1
0	0	0	1	1	1	0	1
0	0	1	0	1	0	0	0
0	0	1	1	1	0	0	1
0	1	0	0	0	0	0	0
0	1	0	1	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	1	1	1	1
1	0	0	0	0	0	1	0
1	0	0	1	1	1	0	0
1	0	1	0	0	1	1	0
1	0	1	1	0	1	0	1
1	1	0	0	1	1	1	0
1	1	0	1	0	0	0	1
1	1	1	0	1	0	1	1
1	1	1	1	1	0	1	0

$f_{backward}$							
X				Y			
y_0	y_1	y_2	y_3	x_0	x_1	x_2	x_3
0	0	0	0	0	1	0	0
0	0	0	1	1	1	0	1
0	0	1	0	1	0	0	0
0	0	1	1	0	1	0	1
0	1	0	0	0	1	1	0
0	1	0	1	1	0	1	1
0	1	1	0	1	0	1	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	1	0
1	0	0	1	0	0	1	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	0	1
1	1	0	1	0	0	0	1
1	1	1	0	1	1	0	0
1	1	1	1	0	1	1	1

Table 5.1: Corresponding mappings for $f_{forward}$ and $f_{backward}$.

met. We then reverse the mapping implemented in $f_{forward}$ to construct $f_{backward}$ such that $f_{backward}(f_{forward}(X)) = X$. An example of the described forward and backward mapping is shown in Table 5.1.

When a specific forward mapping is known, the assignment of memory cell values in the backward CRIF is straightforward. For example, in $f_backward$, when $Y = 0101$, the corresponding values for X is 1111, thus we assign the memory location 5 in LUT_4, LUT_5, LUT_6 and LUT_7 to 1. Using this method, we assign all 16 memory cells in LUT_0, LUT_1, LUT_2 and LUT_3 to 0111000101001011, 1100001101111000, 1000010110101011 and 1101010100010110 correspondingly. Similarly, memory cells in LUT_4, LUT_5, LUT_6 and LUT_7 are thus assigned to 0110011000111010, 1101100000110011, 0000111011110001 and 0101010001101101 respectively.

5.5.3 Content-driven Reconfiguration

In the case where a security primitive enforces a static one-to-one mapping, an attacker could exploit the internal structure of the primitive via techniques such as frequency attacks if enough data is collected. In addition, many existing attacks (e.g., modeling attack) are developed to compromise security primitives through statistical simulation. Facing these challenges, we propose to remap the input-output pairs using reconfiguration. Our design of configurable shufflers inserted between two layers allows convenient and low-cost remapping of CRIF. When a shuffler SF_i (residing between $Layer_i$ and $Layer_{i+1}$) changes the way it shuffles, the injective function layer $Layer_{i+1}$ would receive a different input and therefore generates a different output. This effect would propagate until the last layer, resulting in a completely different output.

While many existing reconfigurable security applications propose to perform arbitrary reconfigurations [90] [91], we propose to use the hashed historical contents as configurations for all shufflers in CRIF. Our reconfiguration mechanism is particularly beneficial for encryption/decryption tasks as it forms a dependency chain between not only current output but also all previous inputs. If an attacker intends to steal the current input message, he/she

needs to obtain all historical decrypted input messages, which significantly increases the effort the attacker needs to make to steal a protected message.

5.5.4 Backward CRIF

Note that CRIF always comes in pairs, though we label all injective functions as $f_{forward}$ in Figure 5.1, the backward CRIF maintains the identical architecture as the forward CRIF. The backward CRIF can be easily constructed by replacing all $f_{forward}$s with corresponding $f_{backward}$s as described in section 5.5.2. Note that the configuration for shufflers in the backward CRIF needs to be inverted to maintain the invertibility of CRIF.

5.6 Security Protocol

The mathematical properties and the reconfigurability of CRIF make it a great choice for encryption/decryption tasks. In this section, we introduce a CRIF-based protocol for secure two-party communications.

5.6.1 Assumptions

We assume the there are two parties participating in the protocol, respectively a sensor, and an IoT hub. The communication is bidirectional, meaning messages can be arbitrarily transmitted between the two parties. We assume the communication channel between the sensor and the hub is insecure, meaning an attacker could eavesdrop the channel without being noticed. We also assume that all actions are taken before the IoT deployment is secure; no attacker is allowed to interfere or monitor the CRIF initialization process. Lastly, we assume that the memory used to store the shuffler configurations is only secure before the first message delivery.

5.6.2 Protocol description

Protocol 3 explains how CRIF-based message encryption and decryption works in a two-party communication scenario. Before the first message delivery between two parties, a pair of forward and backward CRIFs are initialized such that both forward CRIF pair are identical and both backward CRIF are identical. Both parties also agree on an initial shuffler configuration seed δ_0 and a hashing function $\mathfrak{h}$. The hash of the configuration seed $_\circ$ is used as the first configuration for both forward CRIF at both parties, while it's inversion is used to configure both parties' backward CRIF. Both the encryption and decryption process can be divided into four steps, respectively decrypting the new configuration from the seed, apply the new configuration, encrypt/decrypt the private message and update configuration seed. Noted that the seed stored in the memory is an encrypted message, it does not compromise the security of CRIF when stored in an insecure memory. Even if the attacker gains the reading privilege of the insecure memory, he/she would not be able to decrypt it to obtain the real configuration. The real configuration in our protocol design is the hash of a historical unencrypted message. Without the correct configuration, it is impossible to model the CRIF function.

We claim our design is safe and robust because of two reasons. First, even though the current configuration depends on previous private messages, we never store any plaintext information in the system. Secondly, current decryption relies on the decryption of a historical configuration, which in turn depends on a more dated historical configuration. This chain-like dependency significantly increases the complexity of the work required to compromise the protocol.

5.7 Security Analysis

We analysis the security properties of our proposed CRIF in this section. We first conduct tests to examine the statistical tests of CRIF including randomness, avalanche effect, and bit-wise correlation. We then apply three modeling attacks to show the modeling attack

Protocol 3 CRIF-based Two Party Communication

Preliminary:

A sensor S possesses a forward CRIF $\mathbb{F}_f^S$ and a backward CRIF $\mathbb{F}_b^S$.

An IoT hub H possesses a forward CRIF $\mathbb{F}_f^H$ and a backward CRIF $\mathbb{F}_b^H$.

Both S and H obtains a hash function $\mathfrak{h}$.

Initialization:

Injective functions in $\mathbb{F}_f^S$ and $\mathbb{F}_f^H$ are identical.

Injective functions in $\mathbb{F}_b^S$ and $\mathbb{F}_b^H$ are identical.

Both S and H stores the same initial configuration seed δ_0 for shufflers.

CRIF-based message Encryption and decryption

Message encryption at time $\tau \geq 1$

1. S uses $\mathbb{F}_b^S$ to decrypt the stored configuration seed $\delta_{\tau-1}$ to $\Delta_{\tau-1}$.
2. S uses $\mathfrak{h}(\Delta_{\tau-1})$ as the configuration vector to configure $\mathbb{F}_f^S$.
3. S uses $\mathbb{F}_f^S$ to encrypt msg_{orig}^{τ} to msg_{enc}^{τ}. S sends msg_{enc}^{τ} to H.
4. S updates the configuration seed to $\delta_\tau = msg_{enc}^{\tau}$.

Message decryption at time $\tau \geq 1$

1. H receives msg_{enc}^{τ}.
2. H uses $\mathbb{F}_b^H$ to decrypt the stored configuration seed $\delta_{\tau-1}$ to $\Delta_{\tau-1}$.
3. H uses $\mathfrak{h}(\Delta_{\tau-1})$ as the configuration vector to configure $\mathbb{F}_b^H$.
4. H uses $\mathbb{F}_b^H$ to decrypt msg_{enc}^{τ} to msg_{orig}^{τ}.
5. H updates the configuration seed to $\delta_\tau = msg_{enc}^{\tau}$.

resilience of CRIF. All analysis is conducted on 100 instances of randomly initialized 64-bit CRIF with 4 layers on a Xilinx Spartan-6 LX45 FPGA.

5.7.1 Statistical Analysis

5.7.1.1 Randomness

The output randomness of CRIF is a strong indication of how predictable CRIF output is. We use the statistical test suite provided by National Institute of Standards and Technology (NIST)[73] to validate the output randomness of CRIF.

We first collect bit streams by recursively feed CRIF output back to the input to eliminate the randomness introduced by an external source. The implementation reconfigures itself based on previous input as explained in the previous section. A total number of 64,000,000 bits were collected and separated into one hundred bit-streams. Table 5.2 shows the NIST test suite result of CRIF. Our CRIF successfully passes all tests with p-value significantly

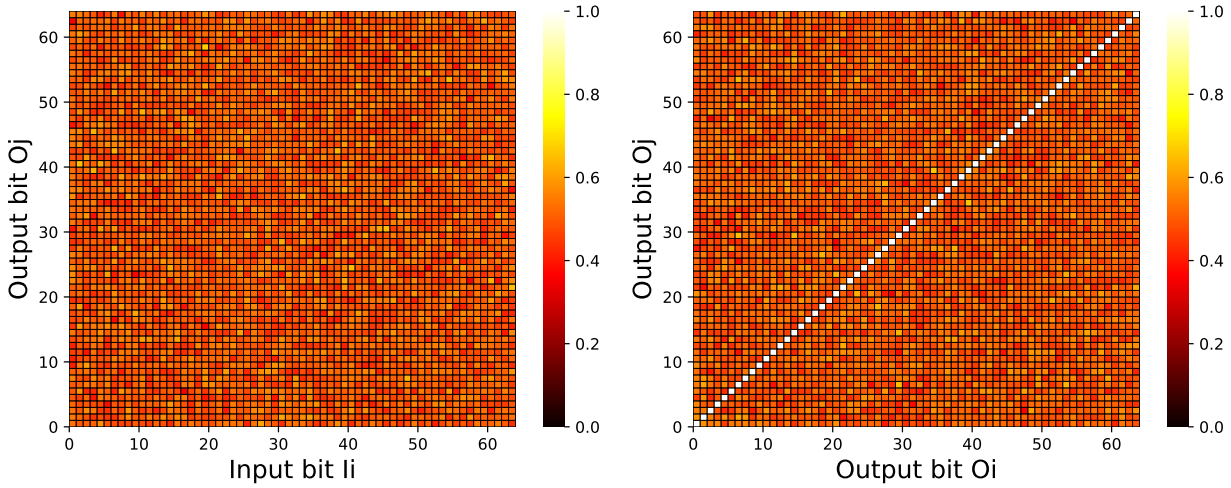
Statistical Test	p-value	Avg. success rate
Frequency	0.738818	99%
Block Frequency (m=128)	0.319084	99%
Cusum-Forward	0.657933	99%
Cusum-Reverse	0.419021	100%
Runs	0.419021	98%
Longest Runs of Ones	0.350485	100%
Rank	0.574903	100%
Spectral DFT	0.779188	98%
Non-overlapping Templates ($m = 9$)	0.699313*	98.8%*
Overlapping Templates ($m = 9$)	0.289667	99%
Universal	0.109188	100%
Approximate Entropy ($m = 10$)	0.058984	99.5%*
Rand. Excursions	-	100%
Rand. Excursions Variant	-	100%
Serial ($m = 16$)	0.275709	99%
Linear Complexity ($M = 500$)	0.637119	99%

Table 5.2: The average success ratio for the NIST statistical test suite. 100 bitstreams of 100,000 bits are passed to each test. The test passes for p-value$\geq \sigma$, where σ is 0.05. Asterisk sign (*) indicates the average case for all templates.

above the 0.05 threshold, proving our CRIF output is highly random.

5.7.1.2 Bit-wise Correlation

Apart from output randomness, we also study the bit-wise correlations in CRIF shown in Figure 5.3. Low correlation in input-output or output-output bits indicates high difficulty in modeling a CRIF. We first investigate the correlation between each bit in CRIF input and each bit in CRIF output. For the convenience of observation, we generate a heatmap for the conditional probability of an output bit O_j given a specific input bit I_i, shown in Figure 5.3a. A conditional probability $P(O_j|I_i) = 1$ indicates output bit O_j stays consistent with input bit I_i and a conditional probability $P(O_j|I_i) = 0$ indicates output bit O_j is negatively correlated with input bit I_i. We expect $P(O_j|I_i) = 0.5$ indicates no correlation between input bit I_i and output bit O_i. Our result shows that the average conditional probability between any CRIF input-output bit pair is 0.5032 with a maximum value of 0.5112 and a minimum value of 0.4821. We then study the correlation between each pair of bits in CRIF output.



(a) Conditional probability $P(O_j|I_i)$. (b) Conditional probability $P(O_j|O_i)$.

Figure 5.3: Statistical analysis on 64-bit CRIF implementations (depth = 4).

Similarly we calculate the conditional probability $P(O_j|O_i)$ of any output bit O_i and output bit O_j. The result is shown ins Figure 5.3b. A white diagonal line shows that $P(O_j|O_i) = 1$ when $i = j$. We expect $P(O_j|O_i) = 0.5, i \neq j$ indicates no correlation between any pair of bits in the CRIF output. The average conditional probability between any output-output bit pair is 0.4996 with a maximum value of 0.5077 and a minimum value of 0.4908.

5.7.1.3 Avalanche Effect

The avalanche effect is a desirable property in cryptographic algorithms. An algorithm with good avalanche effect indicates that a slight change in the input causes a dramatic change in the output. On the one hand, good avalanche effect makes output prediction from input extremely difficult; on the other hand, good avalanche effect makes sure that little about the input can be inferred from the output. We propose to use the Hamming distance distribution between outputs when a single bit is flipped in the input to infer how good CRIF's avalanche effect is. In an ideal case, a single bit flip in a CRIF should flip half of the output bits on average. We conduct such an experiment and show our result in Figure 5.4. The hamming distances indicate a normal distribution with a mean of 31.273 and variance of 20.508, showing an excellent avalanche effect in CRIF.

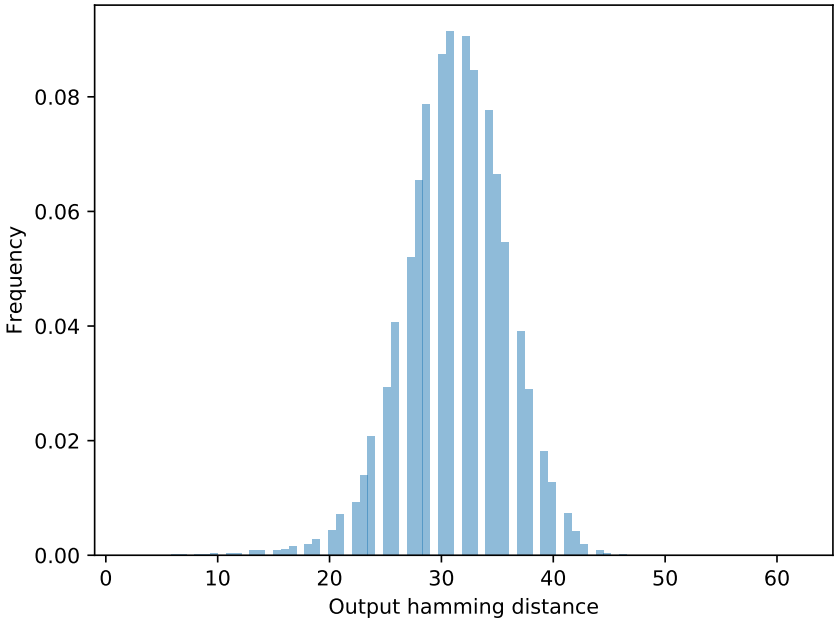


Figure 5.4: Hamming distance distribution for avalanche effect testing measured on a 64-bit CRIF implementations (depth = 4).

5.7.2 Statistical Modeling

Aside from randomness, correlation and avalanche tests, we also test our design under three powerful machine learning algorithms to estimate how robust CRIF stands against modeling attack. The three machine learning algorithms we select are logistic regression (LR), support vector machine (SVM) and multilayer perceptron (MLP). Instead of training a multi-bit model we simplified the problem to single bit prediction. If all methods failed to achieve accurate single bit prediction, then it is impossible to create an accurate multi-bit model.

We transform the single bit prediction to an equivalent binary classification problem to create train suitable machine learning models. We believe the prediction accuracy provides valuable insight in modeling attack resiliency. Our training set size for all three methods is maintained at least 15% of the input space, which is $2^{input\ size}$. We conduct experiments on multiple setting including varying the size of each injective functions in CRIF, adjusting the width and depth of CRIF and also enable/disable our content-driven reconfigurability.

5.7.2.1 Single Output Bit Prediction

We first study if we could create a model that could accurately predict a single output bit from inputs. An accuracy of 100% or 0% indicates that the model successfully models the function the CRIF implements, while a50% accuracy suggests that the algorithm failed to do so. This experiment is an extension to the bit-wise correlation study conducted above. The detailed result is shown in Table 5.3.

5.7.2.2 Single Input Bit Prediction

We then look into the reverse problem of how accurate can a statistical model predicts an input bit from the CRIF output. This experiment is equivalent to train a model that simulates the backward CRIF and is a direct indication of the encryption complexity. A prediction accuracy rate of 50% suggests that CRIF system is complex enough to stand resilient to statistical modeling whereas a high ($\geq 80\%$) or low ($\leq 20\%$) accuracy suggests that CRIF architecture is simple enough to be modeled. The detailed result is shown in Table 5.4.

5.7.2.3 Discussion

We conclude from the results that no machine learning algorithm we adopted is capable of accurately predict single bit input or single bit output, as all prediction accuracy is under 75%, meaning a prediction accuracy for a 32-bit message is below 0.1% even in the worst case scenario.

We also observe that three factors have a significant impact on the output prediction accuracy, respectively CRIF depth, CRIF width and injective function size at each layer. All three factors have a direct negative correlation with prediction accuracy, indicating deeper, wider and larger injective functions makes statistical modeling significantly more difficult. Among these three major factors, we believe that depth is the most important one. When we fix the total number of injective functions, we observe that deeper but narrower CRIF

Size	Layers	Reconfiguable	Logistic Regression Accuracy	Support Vector Machine Accuracy	Multilayer Perceptron Accuracy
4×4	2	×	72.86%	65.48%	69.25%
4×4	4	×	58.16%	61.88%	62.45%
4×4	4	✓	49.24%	50.98%	52.26%
4×4	8	×	50.52%	54.72%	55.99%
4×8	4	×	53.28%	55.97%	61.22%
8×4	4	×	50.15%	55.21%	59.28%
8×8	4	×	49.88%	51.66%	52.17%

Table 5.3: Single bit output prediction accuracy for different CRIF settings. Darker color indicates stronger resilience.

Size	Layers	Reconfiguable	Logistic Regression Accuracy	Support Vector Machine Accuracy	Multilayer Perceptron Accuracy
4×4	2	×	61.38%	74.92%	69.25%
4×4	4	×	54.46%	59.78%	62.45%
4×4	4	✓	50.22%	50.72%	49.86%
4×4	8	×	49.38%	52.72%	53.15%
4×8	4	×	53.29%	55.28%	56.73%
8×4	4	×	54.05%	55.28%	54.11%
8×8	4	×	49.28%	49.62%	50.27%

Table 5.4: Single bit input prediction accuracy for different CRIF settings. Darker color indicates stronger resilience.

outperform wider but shallower CRIF in against all three modeling methods. Note that our content-driven reconfiguration mechanism significantly improves CRIF's modeling attack resiliency, providing the most resilient CRIF marked in dark gray in Figure 5.3 and Figure 5.4. The prediction accuracy on a 2-layered 4×4 CRIF with reconfigurability is as low as 4 layered 8×8 CRIF without reconfigurability, proving our mechanism to be successful.

5.8 Overhead Analysis

We implemented the hardware support for CRIF-based communication protocol on Xilinx Spartan-6 LX45 FPGAs to measure the area and power. In our implementation, we use a 128-bit CRIF with 4 layers. Each layer has sixteen 8-bit injective functions implemented using MUXs and LUTs. The hash function we use to generate the configuration is an efficient SHA-1 FPGA implementation described in [92]. Table 5.5 shows the area and power overhead break down of our implementation. We observe that the majority of the area and power overhead comes from the efficient SHA-1 hash function we use, our CRIF design only occupies 23.17% of the overall area and 37.71% of the overall power consumption. A much more lightweight hash function could further reduce our overhead.

Our design	CRIF	SHA-1 [92]	Overall
Flip-flops	162	1,151	1,588
Slice LUTs	512	1,590	2,209
Block RAMs	0	0	3
Power(mW)	19.80	30.42	52.50

Table 5.5: FPGA resource and power characteristics of the hardware support for CRIF-based encryption/decryption scheme. Power measured for scheme running at 5 Mhz.

We also compare our scheme horizontally with efficient AES designs on FPGAs as shown in Table 5.6. All power data are measured and verified based on the result reported in [93]. Even though all three compared designs are optimized for high throughput and high speed instead of low power, they are still widely adopted. Our design achieves 75.04%, 77.70%, and 84.01% power reduction respectively. On energy constrained IoT devices, where the data transmission is small in size and infrequent, our design is more suitable for the security

tasks.

	CRIF-based design	[94]	[95]	DOR[96]
Power(mW)	52.50	210.38	235.45	172.43

Table 5.6: Power comparison with other power efficient AES design on FPGA. Power measured for scheme running at 5 Mhz.

5.9 Chapter Conclusion

We present a compact and low power security primitive, CRIF, for message encryption/decryption tasks on IoTs. We show that our design of CRIF is secure as it shows excellent security properties in output randomness, bit-wise correlations and avalanche effect. We have also demonstrated that the content-driven reconfigurability significantly increases the system complexity, rendering popular statistical modeling useless. Lastly, our implementation of the hardware support for our proposed CRIF-based security protocol shows that our design is small and low power, achieving at least 75.04% power savings compared to popular AES-based schemes.

CHAPTER 6

Hardware Sharing between PUF and Digital Logic

6.1 Motivation

There are two major concerns for modern logic designs: power and security. The prevailing portable devices such as mobile phones, tablets, and laptops impose high requirements on the low power design and applications due to the highly constrained power supply. Protecting mobile devices is more challenging than securing non-portable devices not only because of the emergence of novel attack methods such as malicious mobile software, mobile phone trojans, and even electromagnetic side-channel attacks, but also due to the fact that traditional security approaches usually demand high power cost, and thus are not applicable to mobile devices due to limited power supply. Therefore, the desire for lightweight security primitives is stronger than ever.

A PUF, as a unique type of hardware security primitive, has excellent low power, high speed, and unclonablity properties. An individual PUF is a piece of hardware implementing a one-way function which takes advantage of the inevitable process variation to guarantee uniqueness of the function. The input-output mapping function of an individual PUF is deterministic but unpredictable, due to the fact that process variation is not predictable. As a hardware security primitive, PUF employs lower overhead comparing to traditional software-based cryptographic approaches. More importantly, a PUF itself is unclonable which provides protection at the physical level.

However, PUFs suffer from significant drawbacks as well. Two of the most critical concerns are related to output throughput and randomness. As an example, an arbiter PUF has an n-bit challenge vector and only a single output bit. When used to encrypt and de-

crypt messages in real applications, n is usually set to be at least 64, in which case the ratio between the PUF output and the PUF input is $1:64$ or lower. The other concern is associated with the randomness of PUF outputs. The unpredictable and uncontrollable nature of process variation creates unbalanced delay path routing in PUFs. The frequency of $0s$ and $1s$ in the outputs are usually not equal which compromises the security of PUFs. Other randomness problems such as repeated patterns in outputs and strong PUF input-output correlations can also be observed in some implementations.

6.2 Technical Goals and Contributions

We propose a novel PUF design which avoids the above PUF problems while keeping it low-delay, low-power and physically unclonable. We focus specifically on arbiter PUFs. Our design is motivated by the following observation. Traditional arbiter PUFs focus on the analog properties of circuits, for example, delays in the case of arbiter PUFs. Two signals are sent as inputs to the two paths of the PUF, and depending on which path has a longer delay, one signal arrives at the arbiter first, thus producing a 1 or 0 accordingly. However, the PUF circuit itself is capable of serving beyond racing delay signals; it can also be used to convey meaningful digital information. In arbiter PUFs, each delay component is made of transistors/gates. Therefore, when connecting these gates in a particular manner, the overall design can compute specific functions.

Our key idea is to use the same piece of hardware to compute digital logic while racing analog signals at the same time. With only small extra area overhead, our circuit design generates two types of outputs, respectively analog outputs from the PUF, and digital outputs from the digital logic. Moreover, both outputs are generated in the same clock cycle. By combining the above two outputs, we expect to keep the unclonability of PUFs while gaining the advantage of digital circuits.

The digital portion of the circuit can be designed to implement any functionality. In our design, we have specifically chosen leap-forward linear feedback shift registers (leap-forward LFSRs) for the following reasons. First, a leap-forward LFSR is a pseudo-random number

generator; by combining PUF outputs with leap forward LFSR outputs (e.g., XOR the two outputs), the combined result will be highly random. Second, a leap-forward LFSR generates large outputs in a compact area; thus, without introducing extra hardware or delay, the output throughput is highly boosted.

To summarize our contributions, we propose a power efficient PUF design by combing traditional arbiter PUFs with leap forward LFSRs. Our design only introduces a small area overhead as the arbiter PUF, and the leap forward LFSR share a majority of hardware and signal resources. With a single execution of the circuit, both outputs are generated simultaneously. By combining the arbiter PUF outputs with the leap forward LFSR outputs, the system gains higher throughput as well as better randomness. Our design and implementation is based on, but not limited to, FPGA platforms. We have described our detailed implementation in Section 6.7.

6.3 Desiderata

Our work is the first effort to take advantage of both analog properties as well as digital properties of a circuit to create a security primitive. The analog properties are used to build an arbiter PUF while the digital properties are realized by creating a leap-forward LFSR. Before discussing our detailed design and implementation, we identify the architectural, operational, and security desiderata of our work.

- In terms of architecture, we have created a "one circuit, two outputs" design. The PUF portion and the LFSR portion of our designed circuit share almost the same hardware resources, consequently, it saves area and lowers the power consumption.
- In terms of operation, our design achieves high throughput by employing only small delay since the digital outputs and the analog outputs are generated in the same clock cycle without timing overhead.
- In terms of security, by combining the arbiter PUF output with the LFSR output, the final system output has kept the unclonability of the original PUF while the randomness

is enhanced from LFSR.

6.4 Related Work

6.4.1 Hardware Random Number Generators

Random number generators are widely used in many security applications. James has reviewed a majority of commonly used random number generators [97]. In the past, random number generation was mostly done by software. However, as hardware systems become cheaper, faster, and more lightweight, it is feasible and more power efficient to implement the random number generators directly in hardware. A number of HRNGs are proposed based on different technologies. Intel has developed HRNGs for use in cryptographic applications [98]. The work by Petrie et al. has applied noise as a random seed to create HRNGs [99]. Additionally, an FPGA-based HRNG is proposed by Kohlbrenner and Gaj by using the intrinsic jitter contained in digital circuits[100]. Using PUFs to build an HRNG was first proposed by O'Donnell from MIT [65]. Our work implements the design of leap-forward LFSRs which is an extension of the standard LFSR by allowing all shifts in the standard design to be applied in a single clock cycle [101].

6.5 Preliminaries

6.5.1 PUF Model

The PUFs we use for signal racing are standard arbiter PUFs. Figure 6.1 shows the schematic diagram of the PUF model. The basic structure of an n-bit PUF consists of n delay segments. The two propagation delays in the ith segment are denoted as d_i^0 and d_i^1 respectively. The two delays are designed to be nominally equal to each other, but after manufacturing, the effect of process variation will cause unpredictable delay difference between them. When built on an FPGA, the upper delay and the lower delay in each segment are directly implemented using LUTs with the same size. Two identically designed paths are generated by connecting

delay components from each segment, and an arbiter is placed at the end of the two paths. The two paths can be modified using the control bit of each segment. When the control bit is 0, the two paths will not shuffle. When the control bit is 1, the two paths swap. For example, in Figure 6.1, if the control bit of the ith segment is 0, then d_i^0 stays with the upper path and d_i^1 stays with the lower path. However, if the control bit is 1, d_i^0 shuffles to the lower path and d_i^1 shuffles to the upper path. Note that when the shuffling happens, all the delays that connect prior to $d_i^0(d_i^1)$ will be shuffled at the same time.

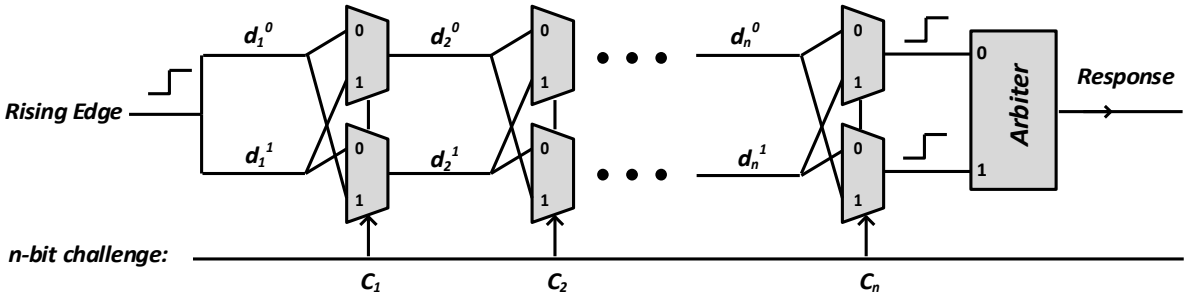


Figure 6.1: The model of arbiter PUFs with an n-bit challenge.

The vector consisting of all control bits is denoted as the PUF challenge. When an n-bit challenge $(c_1c_2...c_{n-1}c_n)$ is provided to the PUF, two identically designed paths are generated. To retrieve a response, an impulse signal is fed into the system to excite both paths simultaneously. Because of process variation, the signal traveling along one of the two paths will reach the arbiter earlier, generating a corresponding arbiter output denoted as the PUF response.

6.5.2 Leap-Forward LFSR

An LFSR is a commonly seen pseudo-random number generator (PRNG). The leap-forward LFSR method utilizes only one LFSR and shifts out several bits. This method is based on the observation that an LFSR is a linear system and the register state is expressed as $Q(i+1) = A * Q(i)$. $Q(i+1)$ and $Q(i)$ are the initial values at $(i+1)th$ and ith steps; A is the transition matrix.

To calculate the content in shift registers after k steps, the equation transforms into:

$Q(i+1) = A^k * Q(i)$. We can compute A^k and determine the XOR structure accordingly. The new circuit leaps k steps in one clock cycle while the circuit uses identical shift registers.

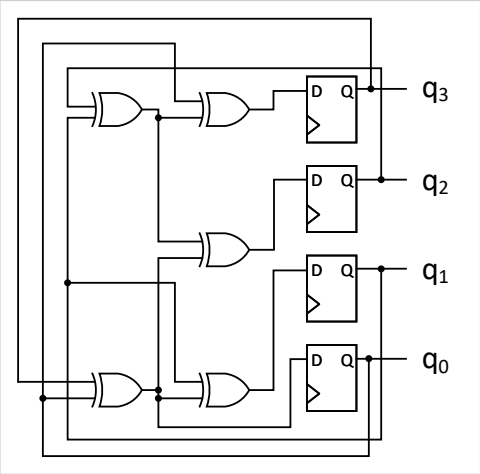


Figure 6.2: A 4-bit leap forward LFSR example.

To illustrate the idea, Chu et al. proposed a motivational example of a 4-bit leap-forward LFSR [101]. The derived new transitional matrix A^4 is calculated from A, which is obtained from a single-bit LFSR random number generator.

$$\mathrm{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix}, A^4 = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

After performing the operations, we can derive the feedback equation for each signal as Equation 6.1 - 6.4:

$$q_0\_next = q_0 \oplus q_3 \tag{6.1}$$

$$q_1\_next = q_0 \oplus q_1 \oplus q_3 \tag{6.2}$$

$$q_2\_next = q_0 \oplus q_1 \oplus q_2 \oplus q_3 \tag{6.3}$$

$$q_3\_next = q_0 \oplus q_1 \oplus q_1 \tag{6.4}$$

The corresponding block diagram is shown in Figure 6.2, requiring 4 flip-flops and 5 XOR gates.

Note that the final transition matrix depends only on the initial transition matrix, thus, different initialization values in shift registers will lead to a different number of XOR logic required in a multi-bit leap-forward LFSR. A 64-bit leap-forward LFSR takes at most 125 XOR gates to build.

6.6 Architecture

6.6.1 Observations

We observed some opportunities that can be taken to further improve a conventional FPGA-based arbiter PUF implementation.

6.6.1.1 Signal Encoding

In a conventional arbiter PUF implementation, the pair of racing clock signals clk_1^i and clk_2^i after the ith segment is always synchronous with the original input signals clk_1^{orig} and clk_2^{orig}. One observation is that the synchronization is not strictly enforced, as long as clk_1^i and clk_2^i are in phase, the arbiter appended at the end of the PUF is able to accurately catch the faster signal. This observation enables us to encode racing signals to digital signals. For example, if clk_1^i and clk_1^{orig} are in phase, we encode clk_1^i as a 0 and if they are in antiphase we encode it as a 1.

6.6.1.2 LUT Utilization

FPGA implementation of arbiter PUFs utilizes LUT6_2 to race signals. A LUT6_2 is a 6-input, 2-output look-up table that is able to act as two LUT5s that shares the same inputs

or just a single LUT6 depending on the control bit. Among the six inputs bits, the control bit (most significant bit) is set to 1 in order to transform the LUT6_2 into two LUT5s. The second most significant bit is usually used as a selection bit that decides the traveling path inside the LUT. This bit serves as a challenge bit of the arbiter PUF. The next two bits are fixed to constant 1s. The two least significant bits are used to take two racing clock signals. We observe that a conventional arbiter PUF implementation does not fully utilize all the possible resources of LUT6_2. If we allow anti-phased signals between PUF segments, we could change the 3rd and 4th bits from static 1s to user-defined values. The conventional design of enforcing two static bits is equivalent to leaving a LUT2 unused for every LUT6_2 in the delay chain.

6.6.2 Overall Design

We propose producing logical output and signal racing results simultaneously on the same hardware. Our design takes a pair of impulse signals as input, and because of process variation, two racing signals do not arrive at the finish line (arbiter) at the same time. Meanwhile, logical computations are performed based on the logical input and the result is carried on the bypassing racing signals using an encoding. Thus, we generate two outputs, each of which is completely uncorrelated. Consequently, an advantage of our design is that we are able to reduce overall power and area by sharing hardware and wires.

We propose to load leap-forward LFSRs on top of a conventional arbiter PUF as shown in Figure 6.3. The arbiter PUF and multiple LFSRs share the same hardware but generate different outputs. The arbiter PUF takes PUF challenges to configure internal paths for the delay signals to race. Meanwhile, leap-forward LFSRs are designed on top of the arbiter PUF to generate digital random numbers.

Eventually, the leap-forward LFSR output and the arbiter PUF response are combined using a post process module (in our case we use XORs and a Von Neumann corrector). The final output leverages advantages from both sources: it inherits high throughput and high randomness from leap-forward LFSRs and physical unclonablity from arbiter PUFs.

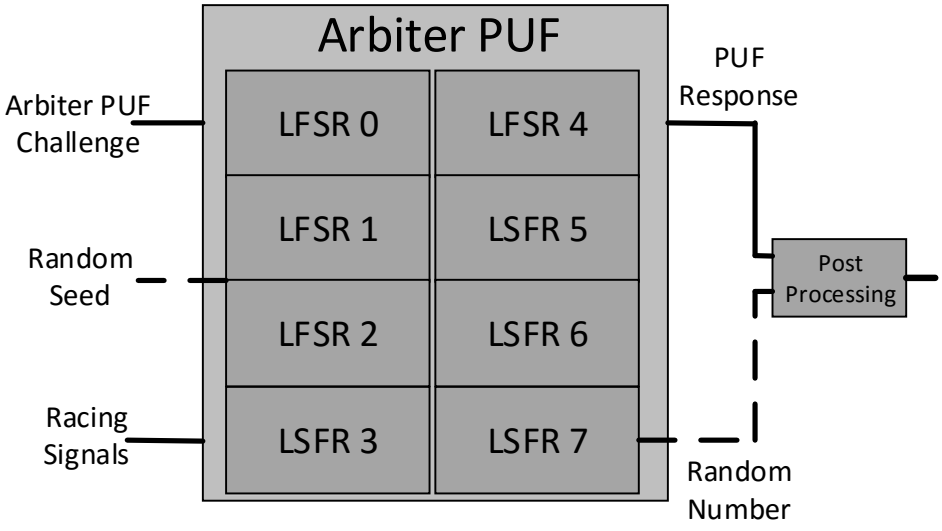


Figure 6.3: High level illustration of proposed design.

6.7 Implementation

We utilize LUT6_2 on FPGAs to demonstrate the feasibility of our design. Figure 6.5 shows our overall implementation. Our design maintains the LUT chain structure and some LUT input pin assignments from the conventional arbiter PUF implementation. However, the third and fourth input bit of each LUT6_2 are no longer restricted to constant 1s. Instead, they are now open to any user-defined values. We use these two bits along with unconstrained initialization vector (INIT) bits to implement an additional XOR gate on top of the LUT6_2. We then use these XOR gates to implement leap-forward LFSRs. The output of the XOR gate is encoded from the racing signal using a flip-flop. Depending on if the signal is in phase or antiphase with the clock signal, the result of the XOR gate is encoded as 0 or 1.

6.7.1 Implementation of Arbiter PUF

The arbiter PUF retains the same structure as a conventional FPGA-based design. Each LUT6_2 implements a single segment of an arbiter PUF. We have made modifications to the implementation to allow racing signals to carry additional digital information. Note that our modification still maintains signal synchronization by keeping two racing signals in phase

throughout the racing process.

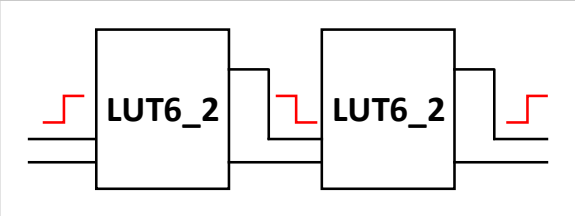


Figure 6.4: The two output signals from LUT6_2 are required to be in phase, but can be inverted simultaneously.

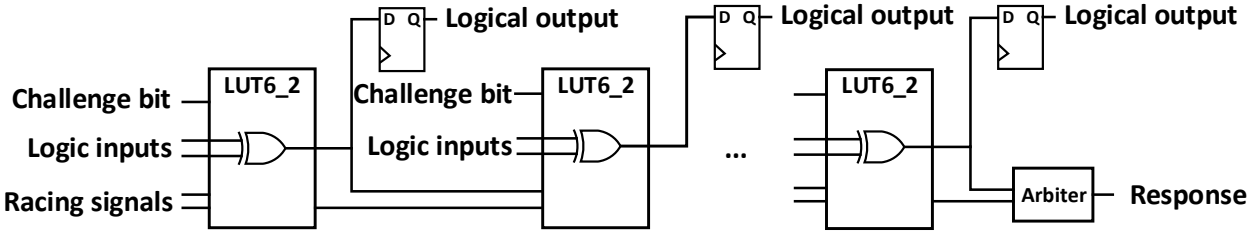


Figure 6.5: Overall implementation using LUT6_2 on FPGA.

6.7.1.1 Allowing Antiphase Signals

In our design, we no longer require signals traveling between LUTs to be synchronous with the clock signal. Figure 6.4 shows a possible scenario where the signals are inverted after traveling through the first LUT and inverted again after the second LUT. Our encoding scheme (in-phase encoded as 0 and antiphase encoded as 1) allows us to carry computational output on racing signals that are traveling through the arbiter PUF chain.

Additional constraints on the INIT value of LUTs need to be applied to allow such functionality. All INIT values must fulfill both requirements below:

- The most significant 32 bits must be mirrored to the least significant 32 bits in units of 4 bits. For example, the mirror image of 1100_1010 would be 1010_1100.
- $INIT[4k] \oplus INIT[4k+3] = 1$ for k ($k \in \{0, 1, ..., 15\}$). The value of $INIT[4k]$ and $INIT[4k+3]$ are decided by the computational logic.

The first constraint is applied to guarantee the upper and lower LUT5s within the LUT6_2 to be the same, thus creating theoretically identical racing paths. The second constraint

is enforced to create racing paths that signals can propagate through. These constraints together guarantee that the two outputs of each LUT6_2 are two in-phase clock signals instead of static outputs or out of phase signal pairs. All bits that are not restricted by the rules can be customized to program arbitrary logic.

6.7.1.2 Arbiter

The arbiter used to capture the faster-racing signal is as simple as a $\overline{SR}$ latch. However, since we allow the signals to be inverted, the phase of the final output of each chain now relies on the implemented logic instead of staying unchanged. Whether the racing signals at the "doorstep" of the arbiter are in phase with the clock signal or not, our original $\overline{SR}$ latch-based arbiter is still able to produce the correct result. Thus, the original PUF functionality is not compromised.

6.7.2 Implementation of leap-forward LFSRs

Leap-forward LFSRs are implemented using XOR gates and flip-flops. A 64-bit leap-forward LFSR requires only 64 flip-flops and 125 XOR gates. However, the wiring complexity of a 64-bit leap-forward LFSR is nontrivial. As the size of leap-forward LFSRs grows, the wiring complexity grows dramatically. To avoid the wiring overhead, we implement multiple 64-bit leap-forward LFSRs on top of our arbiter PUF design.

6.7.2.1 XOR Gates

According to our modification on the arbiter PUF design, it is possible to implement an additional XOR gate on the LUT6_2. Table 6.1 shows the rules needed to implement the XOR logic in addition to the constraints described in section 6.7.1.1. For example, a valid assignment of INIT[31:0] is $8518C3EA$, the mirrored INIT[63:31] would be $AE3C9158$.

The XOR results are then encoded according to rules described in section 6.6.1.1. The two outputs of LUT6_2 are guaranteed to be identical based on the constraints we set, so the XOR result can be retrieved by encoding either of them.

Position in INIT[0:31]	Init value
0,7,11,12,16,23,27,28	0
3,4,8,15,19,20,24,31	1
Other	-

Table 6.1: INIT value rules for implementing XOR gates.

6.7.2.2 Flip-Flops

The flip-flops serve two purposes. First they are used to extract digital information from the racing signal. Second, they are used to store the results of leap-forward LFSRs. In our implementation, we reuse some flip-flops for both purposes to save area and power. Each leap-forward LFSR requires at most 125 flip-flops.

6.7.3 Post Process

64 LUT6_2s and an $\overline{SR}$ latch are needed to implement a 64-bit arbiter PUF while a 64-bit leap-forward LSFR implementation uses 125 XOR gates and 125 flip-flops. This means that we can load four 64-bit leap forward LSFRs on top of eight 64-bit arbiter PUFs. The throughput ratio of arbiter PUF and LFSR is then 1:32 ($8 : 4 \times 64$). We claim that simple XOR operations to combine both outputs provide sufficient randomness. The XOR operation is done between the concatenation of all leap forward LFSR outputs ($4 \times 64 bits$), and a string consists of self-concatenation of arbiter PUF outputs (32 times, which makes it $8 \times 32 bits$). By combining the two outputs, we are able to boost the system throughput by $32\times$ compared to conventional eight 64-bit arbiter PUFs. Note that the randomness can be further improved by applying Von Neumann correction on the arbiter PUF results before the XOR operations. Randomness is evaluated in section 6.8.2.

6.8 Experimental Results

In our implementation, we combine leap-forward LFSRs with an arbiter PUF by sharing hardware and wires. Our motivation is to create a security primitive that inherits the advantages of both while staying free of their drawbacks. We carefully evaluate area, power

and output randomness of our proposed implementation in this section.

6.8.1 Area and power

We claim that by sharing hardware and signals, we are able to utilize area and power more efficiently. To evaluate our design, we implement four 64-bit leap forward LFSRs and eight 64-bit arbiter PUFs using the same hardware resources on a Spartan-6 XC6SLX45 FPGA. In each clock cycle, our design generates 256-bit post-processed output. We compare our design with standalone leap-forward LFSRs and arbiter PUFs that generate the same output throughput (256 bits). We have also compared our hardware sharing design with non-sharing designs on four 64-bit leap forward LFSRs and eight 64-bit arbiter PUFs. The comparison result is shown in Table 6.2.

	Our design	LFSR	PUF	Non-share
Throughput	256+8	256	256	256+8
Flip-flops	532	256	512	320
LUTs	544	250	16,384	764
Slices	288	135	8192	402
Unclonable	yes	no	yes	-
Power(mW)	6.92	3.39	117	7.38
Power/bit	0.026	0.013	0.457	0.028

Table 6.2: FPGA resource and power characteristics: our design (four 64-bit leap forward LFSRs loaded on eight 64-bit arbiter PUFs) vs. four standalone 64-bit leap forward LFSRs vs. 256 64-bit arbiter PUFs vs. four 64-bit leap forward LFSR and eight 64-bit arbiter PUFs that do not share hardware resources. Power per bit unit: mW/bit.

Even though we spend more power and area than leap-forward LFSRs, our design gains the advantage of physical unclonability. When compared to arbiter PUFs, we have reduced LUT cost by 30.1× and reduced power by 16.9×.

When compared to four independent 64-bit leap-forward LFSRs and eight 64-bit arbiter PUFs that do not share hardware as shown in the last column of Table 6.2, our design also does better in terms of both area and power.

The result shows that our design uses more flip-flops compared to the non-sharing scheme. This is due to the fact in order to capture the digital information on the racing signals we

use additional flip-flops for signal encoding. However, we are able to save 40.4% of LUTs and 39.58% of occupied slices. Considering flip-flops are much smaller than LUTs in size on FPGA, we claim our design improves in terms of area.

As of power consumption, our sharing scheme reduces overall power consumption and power per bit by 7.69% comparing to the non-sharing scheme. This number is relatively small considering the large area improvement. This is caused by the larger number of flip-flops used in the implementation. Flip-flops tend to dissipate more switching power than LUTs while in our sharing scheme the majority of shared hardware resources are LUTs.

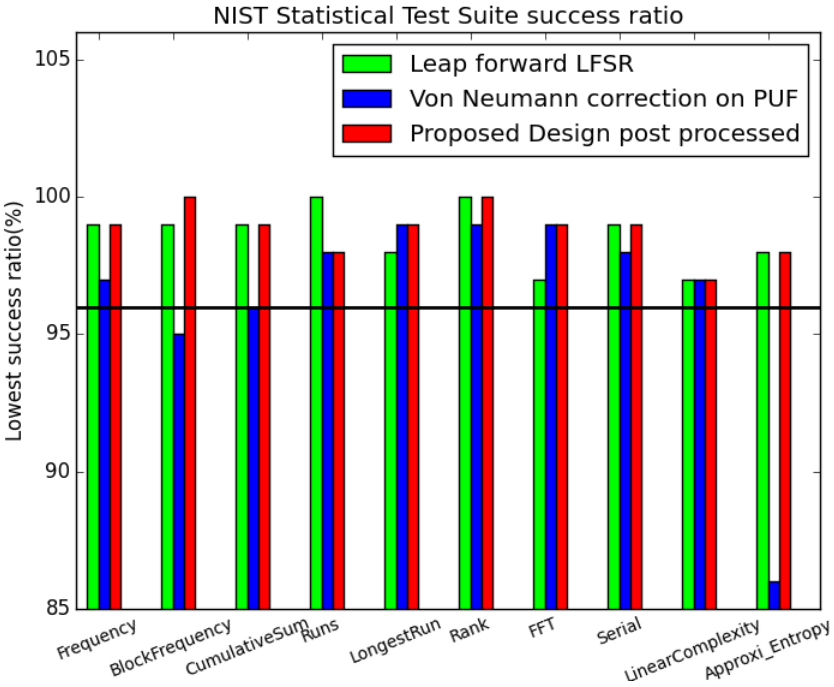


Figure 6.6: NIST Statistical Test Suite success ratio, one thousand 10,000 bit-streams are passed to each test. The test passes for p-value $\geq \sigma$, where σ is 0.05. The black line indicates a threshold of success ratio of 96%. All test results below this line are considered test failure. Arbiter PUF results without Von Neumann correction have success rate below 5% in most tests, thus are not shown in the figure.

6.8.2 Randomness

We quantify the statistical randomness of our design by applying the industry-standard National Institute of Standards and Technology (NIST) Statistical Test Suite to post-processed

outputs [73]. An output stream is generated in such a way that the output of the system in the current clock cycle is fed back to the design as a challenge in the next clock cycle. We repeat this stream production process until all output bits $(1,000 \times 10,000)$ are collected. Figure 6.6 displays the lowest success ratio of outputs generated by leap forward LFSRs, arbiter PUFs with Von Neumann correction, and our design. Three conclusions can be drawn from the figure:

- The output of our design shows excellent randomness, passing all tests in the test suite.
- Our results is at least as random as leap-forward LFSRs.
- Our results outperform arbiter PUF and Von Neumann correction in block frequency and approximate entropy. In all, our proposed method provides better randomness while maintaining physical unclonability.

6.9 Chapter Conclusion

We propose a mechanism to combine signal racing and logic computation through signal and hardware sharing. Employing such a mechanism greatly reduces the area overhead and power consumption. We illustrate our idea by combining leap-forward LFSRs and arbiter PUFs on a Spartan-6 FPGA. The evaluation shows that our sharing design saves 40.4% LUTs while achieving 7.69% of power improvement compared to the non-sharing scheme. Our design maintains the physical unclonability inherited from arbiter PUFs while, as suggested by NIST statistical test suite, achieving much better randomness. We conclude that by racing signals and computing logical operations simultaneously, we are able to create a power- and area-efficient PUF design with unclonability and high randomness.

CHAPTER 7

Lightweight Environmental Anomaly Detection

7.1 Motivation

The massive deployment of mobile reconfigurable devices has imposed a high-reliability requirement. It has become necessary that these devices should operate reliably irrespective of the change in the external operating environment. In order to monitor the reliability of these mobile reconfigurable devices, measurement of physical operating parameters like on-chip power supply voltages and die temperatures are desired in many situations. An alarm should be raised whenever the reliability is compromised due to the abnormal or extreme operating environment. Environmental abnormalities are dangerous because, on the one hand, sudden change in the environment could indicate faulty hardware (circuit short) or logic failure (excessive switching); on the other hand, abrupt variation could also be caused by a variety of physical attack methods, putting the integrity of the operation and the intellectual property embedded in the hardware at the fringe of danger. In both cases, notification of such changes is desired to avoid further damage. Conventionally, detection of abnormal variations requires on-chip temperature and voltage sensors. However, sensor-based monitor system has its drawbacks such as relatively high power consumption and low sample rate. As a matter of fact, many low-end reconfigurable devices like the Internet of Things (IoT) nodes do not have such sensors embedded. Thus, a low cost, high accuracy abnormal environment detection method is desired to monitor mobile devices without the help of temperature and voltage sensors.

APUF, a unique type of hardware security primitive that has been used in many security applications, has excellent properties include low power and high speed. However,

APUFs suffer from the notorious reputation of instability and environmental sensitivity. Challenge-response mapping for a specific APUF is usually not stable: the mapping changes as environmental factors varies. A slight change in temperature and voltage could lead to the unpredictable remapping of many CRP. Several APUF-based security applications intend to minimize the impact introduced by instability using stable CRP selection or error correction code (ECC). While considerable efforts are seeking to eliminate the problem, we see the environmentally sensitive nature of an APUF as an opportunity. The instability due to environmental changes can be exploited to detect abnormalities during the operation cycle of protected hardware.

7.2 Technical Goals and Contributions

We propose to use an APUF as an environmental anomaly detector. The core design idea is based on the observation that changes in an APUF's challenge-response mapping imply a highly probable variation in operating voltage or temperature. We propose to monitor the remapping activity of only ESCRPs instead to save additional energy and latency. Our design is advantageous comparing to current sensor-based system monitors for the following three reasons:

- Implementation is flexible. Detection of environmental variation is not a must for all applications. Sensors, once embedded, occupy space and drain power regardless of application needs. Our detection framework, however, can be implemented or removed easily on reconfigurable hardware depending on user's demand.

- High sample frequency. Many modern reconfigurable devices such as Xilinx Virtex family have system monitors that provide information on both internal voltage VCCINT and core temperature. The problem with these sensor-based system monitors is that the sampling frequency is relatively low. Many abnormalities only affect the device for a few clock cycles, the current system monitor sample rates on Xilinx Virtex devices (200kHz maximum) fails to detect these variations. Our APUF-based

framework is capable of sampling at a much higher frequency (around 10 Mhz in our implementation).

- Low power and compact size. Our detection framework requires only a single APUF to detect abnormalities, which grants us a huge advantage in terms of area and power. After careful off-line ESC seed generation and alarm threshold calibrations, the system provides comparable detection rates with state-of-the-art detection mechanism.

To our best knowledge, our work is the first effort to take advantage of environmental sensitivity in APUF CRP-mappings to accurately detect abnormal environmental variations.

7.3 Related Work

7.3.1 System Monitor

Many efforts have been made to monitor the physical parameters of a reconfigurable system. Xilinx has embedded its System Monitor in its recent FPGA products [102]. The System Monitor function is built around a 10-bit Analog-to-Digital Converter (ADC) and a number of on-chip sensors. The sensors are used to measure FPGA physical operating parameters like on-chip power supply voltages and die temperatures. However, Xilinx System monitor has a relatively low sample rate. Many variations occur within a few clock cycles and the 200k sample rate is not enough to detect short variations. Le Masle and Luk introduced a ring oscillator-based attack detection system that monitors the core power of a circuit through observing the behavior of embedded ring-oscillators. The system reports any suspicious drop in internal voltage and handles the threat accordingly [103]. Even though the ring oscillator-based power monitor framework is capable of detecting abnormalities at a much higher sample rate, it is also much larger in size. Despite the existing studies, our proposed APUF-based anomaly detection framework is capable of achieving a high sample rate while remaining simple, compact and low power.

7.4 Preliminaries

7.4.1 Unstable APUF

Again the PUF we use in our anomaly detection framework is standard delay-based APUFs. An n-bit APUF takes an n-bit challenge as input and produces a 1-bit response as output. When an n-bit challenge is provided to the APUF, two identically designed paths are generated along a chain of multiplexer pairs. Each challenge bit controls whether the pair of paths should swap positions. A pair of multiplexers is denoted as an APUF segment. To retrieve a response, an impulse signal is fed into the system to excite both paths simultaneously. Because of the uncontrollable process variation, the signal traveling along one of the two paths will reach the arbiter earlier, generating a corresponding arbiter output.

Assuming an APUF maps challenges C to corresponding responses R. The APUF has n segments meaning all challenges C are n-bit long. We assume no delays on the connection wires and all delays are contributed by the APUF segments. Given a specific challenge $c \in C$, the ith APUF segments generates a pair of delays with delay difference of Δd_i^c. In a stable environment, the corresponding response $r \in R$ can be mathematically represented as:

$$r = \begin{cases} 0 & \text{if } \sum_{i=1}^{n} \Delta d_i^c > 0 \\ 1 & \text{if } \sum_{i=1}^{n} \Delta d_i^c < 0 \end{cases} \tag{7.1}$$

In real life, however, environmental variations change the delay difference in each APUF segment. For simplicity purposes, we assume that a minor change in environmental would change the delay difference in the ith segment by Δd_i^e where $\Delta d_i^e \in [-e, e]$, e is the maximum change each delay difference could be altered in a normal operational environment. Δd_i^e is a function of temperature t and voltage v. Due to on-chip temperature and voltage gradient, t, v are different from segment to segment. When provided with a specific challenge $c \in C$, the response $r \in R$ can be mathematically represented by

$$r = \begin{cases} 0 & \text{if } \sum_{i=1}^{n} \Delta d_i^c + \Delta d_i^e > 0 \\ 1 & \text{if } \sum_{i=1}^{n} \Delta d_i^c + \Delta d_i^e < 0 \end{cases} \tag{7.2}$$

Noted that our anomaly detection framework is not limited to only standard APUF, many other delay-based PUFs like ring oscillator PUFs serve our need as well. However, APUF is more environmentally sensitive, more compact and low power comparing to ring oscillator-based PUFs, so in this work, our design and implementation are entirely based on the above APUF structure.

7.5 CRP Environmental Sensitivity

Physical properties of transistors vary as environment changes, thus a large number of APUF CRPs are sensitive to fluctuations in physical parameters. In this section, we first discuss correlations between CRP consistencies and physical parameters in the operating environment. We then present a method to collect a set of ESCRP with a strong correlation with environmental parameters. We intend to show that the CRP inconsistency in APUFs can serve as a good indicator of environment variations. We define **CRP inconsistency** as the probability for a corresponding response being inverted when the challenge remains unchanged.

7.5.1 Environmental Variation

Operating environment is a general concept that includes all on and off-chip physical conditions during the process of operation. Humidity, room temperature, supply power all play an important role during the process of logic operations on a reconfigurable device. In this chapter, we mainly focus on studying on-chip voltage and temperature. We carefully examine the core temperature and voltage levels in a reconfigurable platform and their relationship with the on-chip APUF CRP inconsistency.

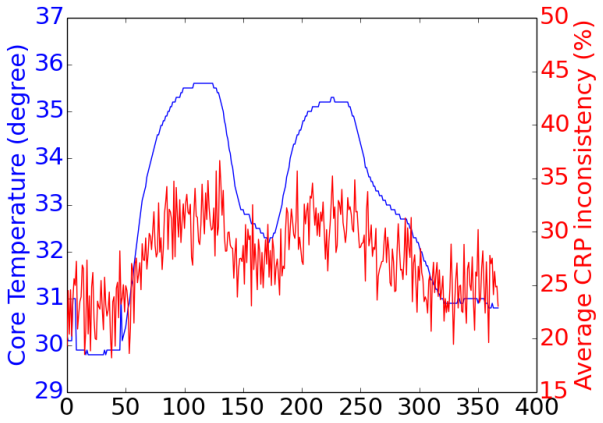
7.5.1.1 Core Temperature vs. CRP inconsistency

Core temperature can be fluctuated by many factors including but not limited to circuit switching activity, room temperature, heat sink efficiency, etc. A sudden, unintended and significant increase or decrease in the core temperature is defined as **temperature anomaly**. Temperature abnormalities could result in circuit behavior alternation, hardware malfunction or even physical damage. APUF serves as an excellent temperature anomaly detector because transistor delays are sensitive to temperature. Change in transistor delays potentially invert the signal racing results between two delay paths in an APUF.

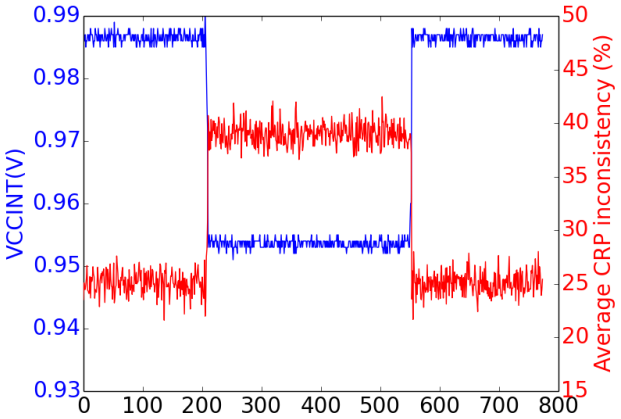
We first randomly selected 1,000,000 64-bit binary strings as our test challenge set. Each challenge within the test challenge set is repeatedly fed to the 64-bit APUF implemented on a Virtex-5 FPGA. We adopt the majority voting scheme to decide on a standard response. For a challenge C, if a majority of the corresponding responses is R, the tuple (C, R) is recorded as the reference CRP. All reference CRPs are stored in a dictionary $Dict_{CRP}$. Core temperature is being sampled and recorded simultaneously using on-chip sensors. A set of 100 challenges were being evaluated between every two sensor samples. Each set is being evaluated for 1,000 samples until being replaced by a new set of 100 challenges. The result is shown in Figure 7.1.

Figure 7.1a shows a snippet of our experiment. The blue line is the core temperature gathered from the built-in SYSMON hard macro. The red line is the average inconsistency of a set of 100 test CRPs. After the 50th sample, we intentionally increase the core temperature by 5 °C using excessive switching circuits [104]. An intuitive conclusion can be drawn from the figure that the pattern of both core temperature (blue line) and average CRP inconsistency (red line) are correlated. Analysis of the entire CRP test set shows that the CRP inconsistency and collected core temperature are correlated with a correlation coefficient of 0.7461. When the core temperature increases, a rise in CRP inconsistency occurs. The inconsistency falls back as the core temperature recovers. The 5 °C increment in the core temperature leads to approximately 13.5% increment in CRP inconsistency.

However, the CRP test set is chosen at random in our experiment; a large number of



(a) Core temperature(°C) vs. average APUF CRP inconsistency(%) for a set of 100 randomly selected challenges.



(b) VCCINT(V) vs. average APUF CRP inconsistency(%) for for a set of 100 randomly selected challenges.

Figure 7.1: A snippet of core environment change vs. inconsistency of randomly selected APUF CRPs. The y-axis on the left is core temperature corresponding to the blue line, the y-axis on the right is the average CRP inconsistency corresponding to the red line.

stable and ultra unstable CRPs were included. Stable CRPs are always consistent regardless of the increment in the core temperature, thus greatly reduces the environmental sensitivity of the APUF. Ultra unstable CRPs on the other hand always provide near 50% inconsistency, which adds additional noise to our evaluation.

7.5.1.2 Core Voltage vs. CRP inconsistency

Core supply voltage inside a reconfigurable device is not static. In most cases, core supply voltage VCCINT varies in a tolerable range. Many events could lead to abnormal variations in VCCINT: simple power analysis(SPA) could lead to an abnormal decrease in the VCCINT since power measurement tool introduces additional resistance to the power rail; Electrostatic discharge(ESD) could lead to abnormal increment on VCCINT, etc. APUF serves as a good VCCINT detector because a slight variation in supply power results in the remapping of some CRP in APUF, thus by actively monitoring the mapping of APUF CRP inconsistency, variations in internal supply power can be detected through calculation on CRP inconsistency.

We produce a SPA scenario that results in a sudden decrease in VCCINT. The experiment Virtex-5 board uses a power module(TI TPS54620) to provide power to FPGA core with the internal supply voltage(VCCINT). The power module is essentially a buck converter that generates output voltage $0.8\text{V} < V_{out} < 15\text{V}$. Figure 7.2 shows a simplified diagram of the module modified by us. We produce a SPA attack by adding an additional switch resistor to modify VCCINT directly through the power module. VCCINT can be instantly decreased by opening the switch SW to add a resistor R_{SW} into the circuit. The output voltage of the power module before opening the switch SW can be calculated using equation 7.3. When mimicking the probe insertion, we open the switch SW, and the output voltage VCCINT can thus be calculated using equation 7.4.

$$\text{VCCINT} = \frac{R_1 \cdot V_{ref}}{R_2} + V_{ref} \tag{7.3}$$

$$\text{VCCINT} = \frac{(R_{sw} + R_1) \cdot V_{ref}}{R_2} + V_{ref} \tag{7.4}$$

Xilinx Virtex-5 allows VCCINT to vary as much as 5% [105]. We use three configurations of R_{SW}. The replacement of resistors is capable of changing the VCCINT by 2.08%, 4.26%, and 5.37%. We apply the same evaluation as the previous experiment on the same 1,000,000

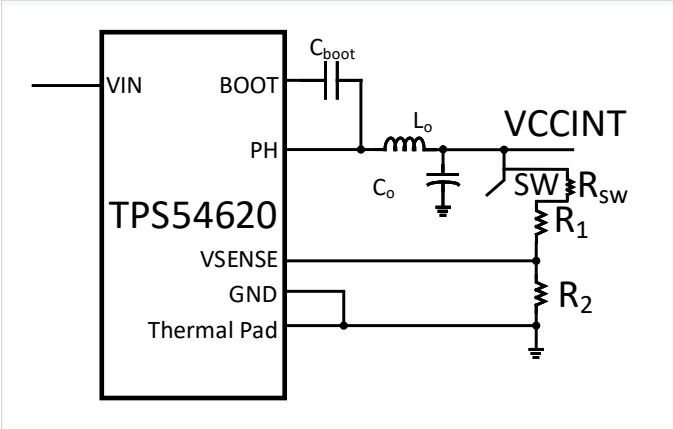


Figure 7.2: VCCINT Power Module, when switch SW is open resistor R_{SW} is inserted into the power rail.

randomly selected test challenges and observe the relationship between CRP inconsistency and VCCINT. A visualized result for CRP inconsistency for a sudden change of 5.37% in VCCINT is shown in figure 7.1b. After the 150th voltage sensor sample, the switch is opened so that VCCINT decreased from 0.986V to 0.935V. We observe that the average CRP inconsistency increased from 23% to roughly 41% inconsistency, which later drops back to 22% as we closed the switch and restored VCCINT to 0.986V.

7.5.2 Environmentally Sensitive CRP

The above evaluation above shows that CRP inconsistency in an APUF is correlated with core temperature and supply voltage. However, the correlation is not strong enough to meet our application needs. After review the whole CRP set, we conclude that not all CRPs are equally sensitive to the environment. Based on the notation given in equation 7.1 and 7.2, we define stable, ultra unstable and ESCRPs as below.

A stable challenge produces consistent response regardless of operating environment. A stable CRP appears if one or more segments dominate over all remaining segments so that the final delay difference is sufficiently large enough to overcome delay variations. Stable challenges can be defined in Equation 7.5.

$$|\sum_{i}^{n}(\Delta d_i^c)| > \sum_{i}^{n}|\Delta d_i^e| \text{ for all possible } \Delta d_e \tag{7.5}$$

so that the APUF generate the same response regardless of environmental variation. Our APUF implementation shows that roughly 30% of CRPs are stable.

An ultra unstable challenge on the other hand creates two paths with near zero delay difference. This can be represented by Equation 7.6:

$$\sum_{i=1}^{n} \Delta d_i^c + \Delta d_i^e \approx 0 \text{ for } \Delta d_e \in [-e, e] \tag{7.6}$$

With the given challenge, there is a near 50% probability that the corresponding response would be a 1 or a 0. Our APUF implementation shows that roughly 5-10% of CRPs are ultra unstable.

To be an indicator of environmental changes, we are primarily interested in searching for environmentally sensitive challenges(ESCs) that neither produces a stable nor completely unpredictable response. An ideal ESC should at least fulfill the following two conditions described in Equation 7.7 and 7.8:

1.
$$|\sum_{i}^{n} (\Delta d_i^c)| < \sum_{i}^{n} |\Delta d_i^e| \text{ for } \Delta d_e \in [-e, e] \tag{7.7}$$

   when so that no dominating segment group exists in the APUF, and environmental variations could possibly invert the response.

2.
$$\sum_{i=1}^{n} \Delta d_i^c + \Delta d_i^e \approx 0 \text{ for } \Delta d_e \in [-e, e] \tag{7.8}$$

   At a given environment the response is stable in a normal operating environment.

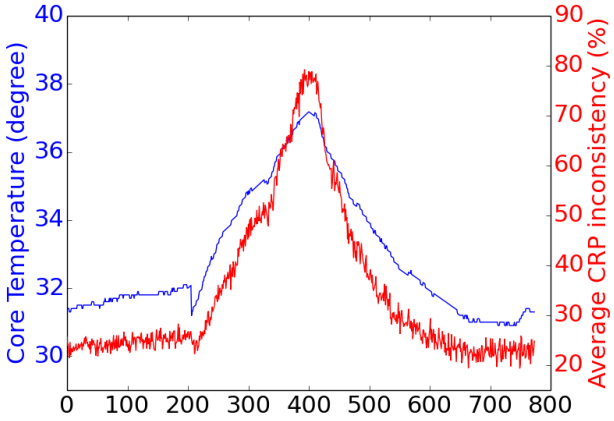
In addition to the previous two requirements, an ideal ESC should be able to obtain an inverted response when the environment varies to an abnormal range:

3.

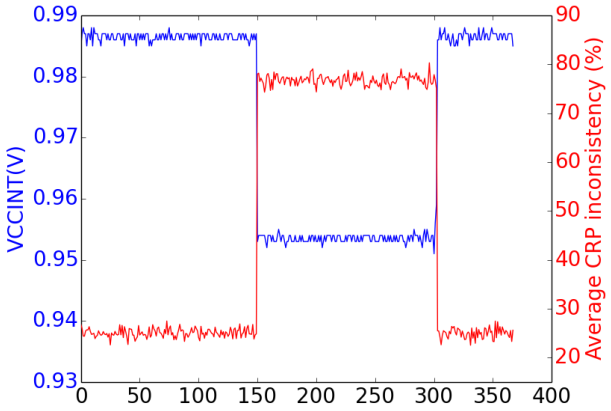
$$sign(\sum_{i=1}^{n} \Delta d_i^c + \Delta d_i^e) = -sign(\sum_{i=1}^{n} \Delta d_i^c + \Delta d_i^{e\prime}) \quad \text{for } \Delta d_e \in [-e, e] \text{ and } \Delta d_i^{e\prime} > e \text{ or } \Delta d_i^{e\prime} < -e. \tag{7.9}$$

In our experiment, we defined normal operating environment as: 1. core temperature $T = 30.5 \pm 5\,^{\circ}\text{C}$, 2. VCCINT $= 1 \pm 0.5V$. From the 1,000,000 test challenge set, we observe that no challenge fulfills all three requirements for ESCs. Non-uniform physical properties over the chip makes qualifying both equation 7.8 and 7.9 extremely difficult. We relaxed the requirement on these two conditions by 30% meaning allowing at most 30% violations in both equations. Thus we are able to collect 10,000 qualified CRPs out of a 1,000,000 random challenge set. We repeated the evaluation with only these challenges again, and a snippet of the result can be seen in figure 7.3.

Figure 7.3a shows that the CRP inconsistency of the ESCRPs (red line) dramatically increases when the core temperature (blue line) is intentionally increased through circuit switching. The correlation coefficient between CRP inconsistency and core temperature increased from 0.7461 to 0.8722, much higher comparing to using a random challenge set. The average CRP inconsistency increased from 23% to roughly 79.8% inconsistency when we increase the core temperature to $37.5\,^{\circ}\text{C}$, which later drops back to 22% as switching circuits cool down. A similar observation can be made in Figure 7.3b as we close the switch and change VCCINT instantly. The average inconsistency of the selected CRPs instantly changed to over 80%. Comparing to randomly selected CRPs, these sensitive CRPs response much faster and dramatic because stable CRPs have been eliminated. We conclude that ESCRPs are much efficient and environmentally sensitive when using APUF as environmental variation indicator.



(a) Core temperature(°C) vs. APUF CRP inconsistency(%) for a set of 100 ESCs.



(b) VCCINT(Volt) vs. APUF CRP inconsistency(%) for a set of 100 ESCs.

Figure 7.3: Core environment change leads to variations in inconsistencies of ESCRP. The y-axis on the left is core temperature corresponding to the blue line, the y-axis on the right is the average CRP inconsistency corresponding to the red line.

7.5.3 ESC Set Generation

APUFs serve as a good environmental variation detector when ESCs are applied. Since ESCs are different from APUF to APUF, and there is no general pattern in them, acquiring a large set of ESCs at run time is no trivial task.

We present an efficient, two-step method to collect such a large set of ESCs. We first do a random search to collect a single ESC seed using evolution strategies (described in Protocol 4). Evolution strategies method is inspired by the evolutionary adaptation of a population

of individuals to certain environmental conditions. When applying evolution strategies on ESC generation, one individual in the population is a specific challenge vector and mutation on an individual is defined as randomly flipping multiple bits. The environmental fitness of the individual is determined by the consistency of the corresponding PUF response under a small environmental variation benchmark where environmental factors vary beyond tolerable range using switching circuit/power rail resistor. The tolerable range is defined by the user based on the nature of the application. The pseudocode of ESC exploration is described in Protocol 4)

Protocol 4 Explore ESC seed

Require: (1) Number of candidates in the parent generation μ. (2) Number of candidate solutions generated from the parent generation λ. (3) Expected ESC set size κ (4) A fitness evaluation function EvaluatePopulation(). (5) A mutation rate τ. (6) Maximum number of generations M.
Ensure: A set of ESCs S_{best}.

```
Population ← InitializePopulation(μ).
EvaluatePopulation(Population)
S_best ← GetBest(Population, κ)
while i < M do
  Children ← ∅
  for j = 0 to λ do
    Parent_j = GetParent(Population, j)
    S_j ← Mutate(Parent_j, τ)
    Children ← S_j
  end for
  EvaluatePopulation(Children)
  S_best ← GetBest(Population+S_best, κ)
  Population ← SelectBest(Population, Children, μ)
  i = i + 1.
end while
Return(S_best)
```

We observe that when inverting a small number of bits in an ESC seed, the new challenge is most likely to stay environmentally sensitive due to the fact that nearby transistor process variations are somewhat correlated. We thus generate more ESCs by randomly inverting a small number of bits in the ESC seed (described in Protocol 5).

Protocol 5 Generate ESC set

Create empty challenge set ESC, define the number of the size of the ESC set m, and a growth index i. An ESC seed esc is taken as an input
Count = 0
while Count $< m$ **do**
  for k in range(i) **do**
    Randomly flips k bits in esc to obtain c^k_{Count}
    Add c^k_{Count} to ESC
  end for
  Count += 1
end while

7.6 The Detection Framework

7.6.1 System Design

Figure 7.4 shows a high-level implementation of APUF-based anomaly detector on FPGA platforms. The detector consists of three major modules.

The challenge querier is used to generate challenges, and feed them into the APUF at run-time. Before the actual run-time detection, a calibration process generates an ESC seed off-line using Protocol 4. The challenge querier generates a set of ESCs using Protocol 5 at run-time. A random number generator embedded inside of the challenge querier decides at each clock cycle which bits should be inverted. The challenge querier is placed at the furthest end of the logic blocks to minimize errors.

The APUF process the same challenge multiple times and send all generated responses to the response verifier module. The APUF itself resides immediately next to the logic being monitored to achieve the best accuracy.

The response verifier calculates the CRP inconsistency by counting the frequency of response inversion, and raise an alarm if the inconsistency exceeds a user-defined threshold. Since all ESCs are correlated, the 0 and 1 distribution are also very similar, thus a single counter is enough for counting and recording bit inversions. The alarm threshold is recommended to be set to the selected ESC's fulfillment rate of equation 7.8 and 7.9, however it is adjustable as the definition of the normal operational environment may be subject to change

based on user requirement. The response verifier is placed at the furthest end of the logic blocks to minimize errors.

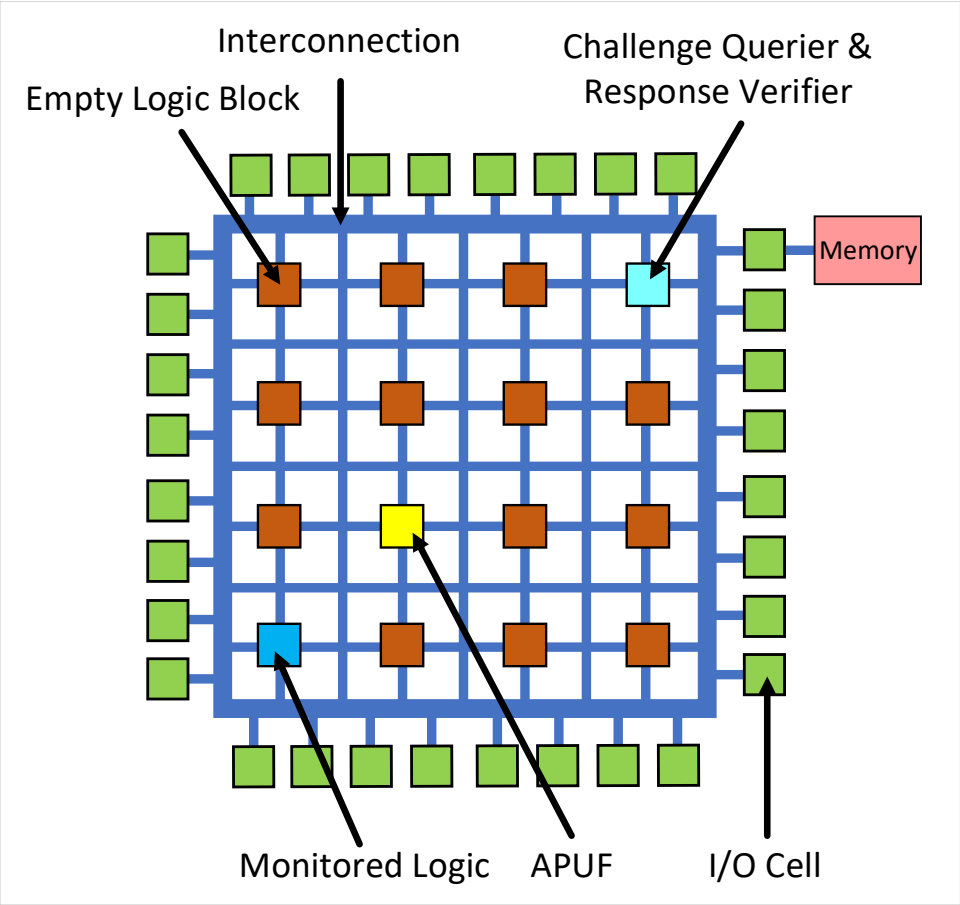


Figure 7.4: APUF-based anomaly detection framework on FPGA.

7.6.2 Experimental Results

In this section, we carefully evaluate our anomaly detection framework.

Our experiments are conducted on two Genesys boards. The Genesys board has a Virtex-5 LX50 FPGA. The system is tested with the implementation of an RSA-1024 and an AES-128 crypto-system [106]. A mixture of 1,000 abnormal core temperature and VCCINT was applied during a 10-hour long operation. In our setting, an anomaly in core temperature is defined as a variation of over 5 °C and an anomaly in VCCINT is defined as a variation of over 5%. We generated three distinctive challenge sets, respectively 1,000 random chal-

	SYSMON[102]	Our Design(Virtex-5)	[103](Spartan-6)	Our Design(Spartan-6)
Sample rate	200kHz	200kHz	8MHz	8MHz
LUTs	251	92	3300	49
Flip-flops	139	69	unknown	57
Slices	63	36	825	22
Power(mW)	653	569	1953(estimated)	29

Table 7.1: FPGA resource and power characteristics: Virtex-5 SYSMON vs. our design on Virtex-5 vs. RO-based on-chip power monitor vs. our design on Spartan-6. ADC and analog sensor area and power are not included in calculation.

lenges(RC), a single ESC seed, and 1,000 ESCs derived from the ESC seed. The ESC seed is generated based on our requirement on core temperature and VCCINT, which in our case is our definition of the normal range of temperature and VCCINT defined above. The alarm threshold is set to raise an alarm when CRP inconsistency reaches 40% for 1,000 random challenges, 70% for both single ESC seed and 1,000 ESCs case. A UART core is used to monitor the real-time CRP inconsistency and communicates with RSA/AES cores. The results are displayed in Table 7.2.

Circuit	1,000 RC	Single ESC	1,000 ESC
AES-128	69.4/0.7	98.1/5.9	100/2.1
RSA-512	78.4/3.0	100/3.9	100/1.1

Table 7.2: anomaly detection: true positive rate (%)/false positive rate(%) using 1,000 random challenges vs. single ESC vs. 1,000 ESCs as challenge set per sample. APUFs are placed immediately besides the monitored circuit. A mixture of 1,000 abnormal temperature and VCCINT were applied to the protected circuit.

We observe that 1,000 ESC challenge set provides the best detection rate while random CRPs provides the worst. ESCs benefit from the elimination of stable and ultra unstable challenges. With a good ESC seed, we are capable of achieving a true positive rate of 100% with 1,000 generated ESCs at run-time, while the false positive rate is as low as 1.1%.

7.6.3 Area and Power

We compare our design's area and power overhead with both Xilinx System Monitor [102] and ring oscillator-based power monitor[103]. Since Xilinx System Monitor only supports

Virtex family product and Le Masle's ring oscillator-based power monitor was originally implemented on Spartan-6 LX45, we implement our design on both platforms for comparison purposes. The sample rate for Virtex-5 implementations is fixed to 200kHz while the implantation on Spartan-6 devices has a sample rate of 8MHz. The FPGA resource and power characteristics for Xilinx System Monitor, ring oscillator-based power monitor and our design are shown in table 7.1.

Our design uses 63% less FPGA area comparing to Xilinx System Monitor logics while achieving 13% of power savings. To be noted that we do not include the area and power of both analog sensor and ADC in the SYSMON hard macro in our calculation due to lack of information. The actual saving is expected to be more. Comparing to Le Masle's ring oscillator-based power monitor, our design is 98.4% smaller. Since APUF does not impose high switching activities like ring oscillators, our design consumes only 1.5% of the power.

7.7 Chapter Conclusion

In conclusion, we discovered that APUF CRP inconsistency is highly correlated with the core temperature and voltage on reconfigurable platforms. Based on the observation we designed a framework to utilize the environmental sensitivity of APUFs to detect suspicious operational environment variations. We propose to apply only ESCs on a given APUF to efficiently and accurately detect environmental abnormalities. When integrated with an AES-128 and an RSA-1024 implementation, our framework is capable of detecting 100% of applied abnormalities with a false positive rate as low as 1.1%. Our design provides competitive detection rate with both sensor-based Xilinx System Monitor and Le Masle's ring oscillator-based power monitor while reducing 63% and 98.4% of area as well as 13% and 98.5% of power.

CHAPTER 8

Efficient and Secure Group Key Management in IoT using Multistage Interconnected PUF

8.1 Motivation

Internet of Things (IoT) has been envisioned to be a revolutionary network that connects physical devices around us to perform intelligent tasks such as monitoring, communication, operation, and optimization. The advancement in IoT technology has enabled a wide spectrum of applications in a variety of environments to measure the various environmental parameters [107]. While IoT technology has greatly improved the efficiency and quality of our lives and works, various security challenges have become a major concern and doubt for further adoption of the technology. Security improvement in IoT system has become an increasingly popular topic in both academia and industry due to its urgency and profitability. In this chapter, we are particularly interested in efficient and secure key management schemes in group communications in an IoT setting.

8.2 Technical Goals and Contributions

Group communication through multicast/broadcast enables direct communication with the whole group, which is more efficient when compared to an equivalent unicast-based solution. Securing group communications consists of providing confidentiality, authenticity, and integrity of messages exchanged within the group [108]. Among all security problems in IoT, group key management is one of the fundamentals in securing group communications. A group key essentially is a secret key shared by all members of a group so that all group

communication packages are encrypted before they are being transmitted using this group key. An unauthorized user may receive group communication packages due to a network error or intentional interception; however, without the right group key, the illegal user cannot decrypt the received packages [109].

Group key management schemes in IP networks, though have been studied for decades, cannot be directly applied to IoT as IoT devices are heavily constrained by the limited resource and energy capacity. Limited resources impose new challenges regarding storage and computation requirements, meaning each node is incapable of storing a large key database or conduct heavy cryptographic computation. The energy constraint additionally requires key verification and computation procedures to be energy efficient.

For the above two reasons, physically unclonable functions (PUFs), a type of low-power security primitive with unclonable and unpredictable properties, naturally appears as an ideal solution to the problem. In this chapter, we propose to apply a novel low power PUF structure called Multistage Interconnected PUF (MIPUF) to the domain of group key management in IoT. We believe the low power and unclonable, unpredictable nature of MIPUF not only improves the security of group key management protocols but also meet the tighter energy requirements on IoT nodes. Our design of interconnection reconfiguration in MIPUF is robust and secure against modeling attacks by changing the challenge-response mapping. The group key is stored and managed by a new set of PUF functions every time we reconfigure the MIPUF in every IoT device, creating an additional layer of security and protection. We also show that our key management scheme including key distribution, key storage and rekeying is resilient against a wide range of attacks. Lastly, we show that our group key management protocol is power and energy efficient. Our simulation results show that we are 47.33% more energy efficient comparing to state-of-the-art ECC-based key management schemes.

8.3 Related Work

Numerous group key management frameworks have been proposed historically including but not limited to VersaKey, GKMP[110] [111]. Several efforts have been made in creating efficient group key management protocols for group communications in IoT and wireless sensor networks (WSN) to meet the energy and computation constraints. Kung et al. proposed GroupIt to address the scalability problem in GKMP [112]. Zhu et al. proposed an efficient security mechanism for large-scale distributed sensor networks [113]. Roman et al. analyzed the applicability of public-key cryptography based protocols and link-layer oriented key management systems (KMS) in IoT settings [114]. Abdallah et al. proposed a novel efficient and scalable key management mechanism for wireless sensor networks and proposed to reduce power and energy consumption by using ECC [115]. Parrilla et al. proposed an ECC-AES co-processor to handle security tasks in IoT group management [116]. Kandi et al. specifically focus on rekeying schemes in IoT key management [117]. Gebremichael proposed to use One-way Accumulator to handle key establishment [118] while Ferrari suggests more efficient key establishment methods in [119]. Lastly, Lei et al. suggest that blockchain technology is also applicable to IoT group key management in [120]. All work listed above utilize expensive cryptographic primitives to secure their group key management protocols without investigating the possibility of utilizing some novel low-power hardware security primitives to meet the energy requirements.

Recently, PUFs, as a popular type of low-power security primitive, have been proposed to be used in a number of key management subtasks in IoT settings [121]. Gu et al. proposed multiple PUF architectures that provide compatibility to secure authentication. [122] [29] [26]. Mukhopadhyay proposed a novel device authentication method that takes advantage of the unclonable property of PUFs [123]. Rahman proposed an RO-PUF-based key generation scheme that is aging resilient [124]. Most recently Huang et al. investigated a key distribution protocol assisted using ring oscillator PUFs (ROPUFs)[125] which significantly reduces the storage overhead and latency for securely distributing secret keys. Unfortunately, these works only focus on a specific subtask of key management and fail to provide detailed security or

overhead analysis. We differentiate ourselves by design a novel PUF architecture that can be applied to the entire key management lifecycle including key distribution, key storage and rekeying in IoT. We have also performed a security and overhead analysis to prove that our work is both secure and efficient.

Besides ECC and

8.4 Multistage Interconnected PUF

In this section, we propose a novel PUF structure called Multistage Interconnected PUF (MIPUF). We borrow the idea of multistage interconnection networks (MINs) from computer networks field. MINs allow the processing elements (PEs) to be interconnected using Switching Elements (SEs) such that the interconnection provides high configurability and speed with low cost. We propose to use such a structure to interconnect PUFs so that the interconnected PUFs can be configured easily. The interconnected PUFs significantly increase the system complexity as well as break the linearity, resulting in increased difficulty in modeling the system. The configurability also allows the challenge-response pairs (CRPs) of the network to be remapped from time to time, protecting the system from modeling attacks.

8.4.1 Processing Elements (PEs)

We name the PE in a MIPUF a MIPUF node. A MUPUF node is the most fundamental building block of the network. A MIPUF node is a single or a group of strong PUFs that take an n-bit challenge and generate an m-bit response. A strong PUF is defined as a PUF that supports a large number of CRPs and there exists a number of implementation. For the sake of implementation easiness, our implementation of a MIPUF node consists of m n-bit arbiter PUFs running in parallel and sharing the same pulse signal and challenge vector. Even though arbiter PUFs are known to be weak against various modeling attack, our experimental results show that multistage interconnection significantly improves the resilience against them. In this chapter, we merely use MIPUF node implemented with

arbiter PUFs as an illustrative example and a proof of concept. Security properties of MIPUF implemented using more advanced strong PUFs such as LRR-DPUF [126] are expected to exceed our collected results.

8.4.2 Switching Elements (SEs)

Similar to the concept of SEs in computer networks, the SE in MIPUF serves as a way to route and switch signals. In our implementation, an SE is a set of multiplexers that switch or not switch n signals based on a configuration bit. In our case, we use SEs to connect the response of a previous MIPUF node to the next node as the new challenge. The SEs between two nodes are controlled by a configuration vector.

8.4.3 Multistage Interconnection

Multistage interconnection networks find a balance between the cost and configurability. We believe a blocking multistage connection is the most cost-efficient for MIPUF implementation and provides sufficient configurability. A blocking multistage connection cannot realize all possible connections between inputs and outputs since a connection between one free input to another free output is blocked by an existing connection in a network; however, it is much cheaper to implement. While there are different interconnection styles such as Clos [127], we propose to implement a blocking interconnection in a MIPUF in an Omega network style [128] which consists of 2×2 SEs. Each input has a dedicated connection to an output, providing 2^N different switchings and having a complexity of O(N log(N)) for an N×N connection between two MIPUF nodes. An example of such interconnection is shown in Figure 8.1. To be noted that we do not allow port rearrangement in MIPUF, each input should be routed to a unique output and each output should be directed from a unique input given a specific configuration.

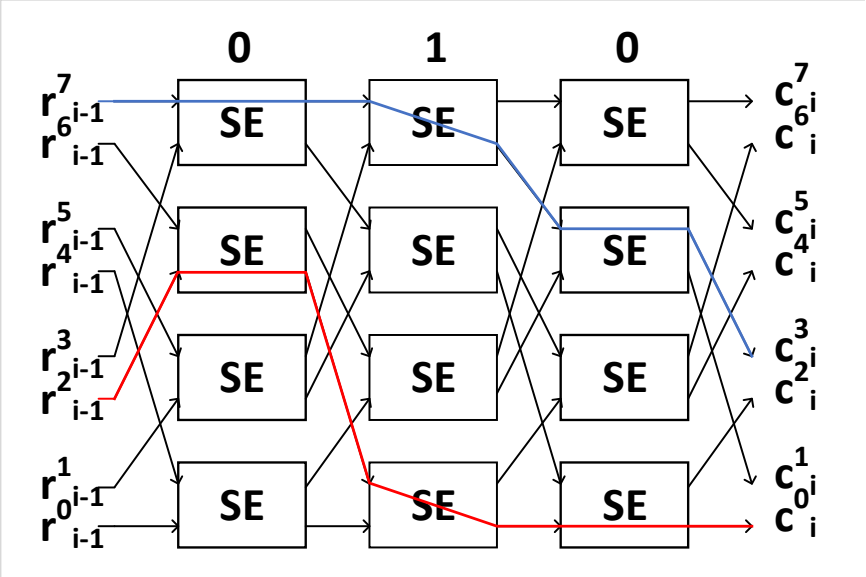


Figure 8.1: Network structure in Omega network style. r_{i-1}^{k} indicates the k-th response (output) bit of the $(i-1)$-th MIPUF node, c_i^k indicates the k-th challenge (input) bit of the i-th MIPUF node.

8.4.4 Protecting Network Configuration

The signal routing between any two MIPUF nodes is controlled by a configuration vector. We propose to secure the configuration vectors using existing MIPUF nodes in the system so that the real interconnection configuration remains hidden. The configuration vector for the interconnection between node i and $i+1$ depends on the encrypted result of the user provided configuration bits using the nodes from node 1 to node $i-1$, for $i > 1$. To note that we use MIPUF nodes in the previous levels to encrypt SE configurations to reduce the correlation between the output of a node and it's immediate SEs. The user provided configuration is passed to the SEs between the first two nodes and propagate along the network to configure the remaining SEs connected to the later nodes. An attacker or even the user who provided the configuration, cannot obtain information on the real interconnection between MIPUF nodes without characterizing each node. Besides, we also significantly reduce the number of bits a user needs to provide. In a key management protocol, our proposed method could also significantly reduce the communication cost.

8.4.5 Security Evaluation of MIPUF

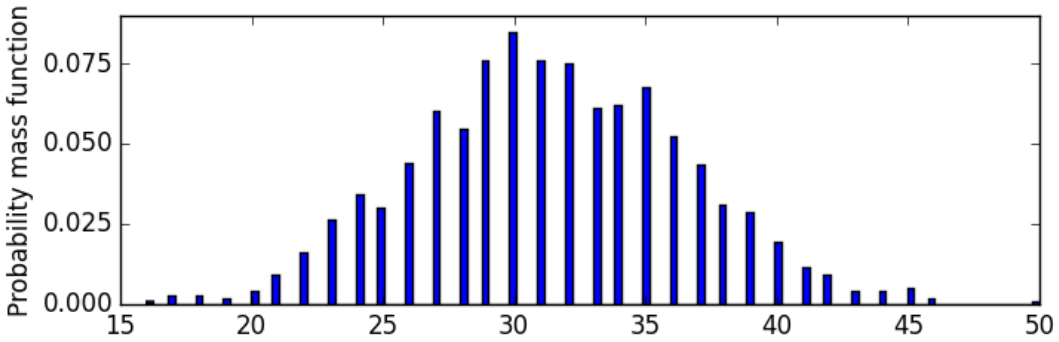
8.4.5.1 Uniqueness and Reliability

Two most important properties of PUFs are uniqueness and reliability. Uniqueness means that the responses for a specific PUF design implemented on different devices should be significantly different when provided with the same challenge. Reliability indicates the response should be stable enough when repeating the same challenge on the same device. Since our design of MIPUF depends on existing PUF implementations, our focus is that our multistage interconnection does not compromise the security properties of the PUF implementation we depend on. We modify the definition of uniqueness and reliability as follows.

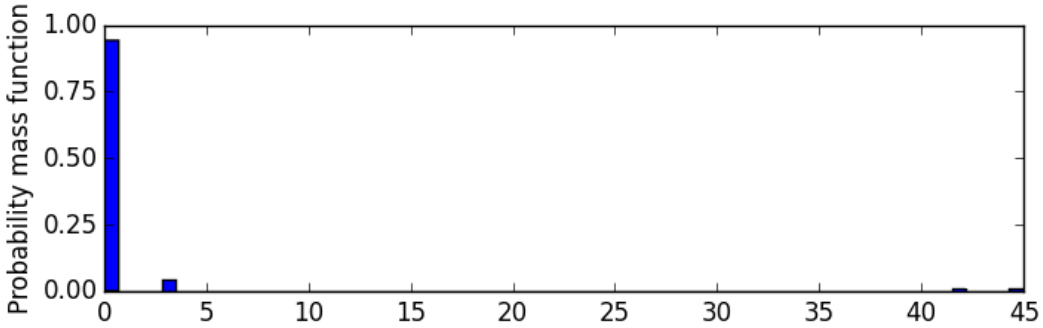
- *Inter-configuration variation (uniqueness).* How many MIPUF output bits are different between two different configurations of the same MIPUF? Ideally, this variation should be 50%.
- *Intra-configuration variation (reliability).* How many MIPUF output bits differs when re-generated again from a MIPUF with a specific configuration? Ideally, this variation should be 0%.

We directly compare these two metrics with intra-chip variation and inter-chip variation metrics in regular PUF evaluations. As a proof of concept, we compare our arbiter PUF based MIPUF implemented using arbiter PUFs with regular FPGA-based arbiter PUFs implemented on five different FPGAs. The results are collected from the Xilinx Spartan-6 XC6SLX45 platform using the implementation described in [24].

Figure 8.2a illustrate the probability distribution of the inter-configuration variation of a MIPUF. The x-axis is the number of output bits that are different between two different interconnection configurations; the y-axis is the probability. The bars show experimental results collected on 1,225 pairs of outputs collected from 50 different configurations. Our experiment results (47.9%) is very close to the ideal case of 50%. Our results even show a slight improvement comparing to the inter-chip variation of arbiter PUFs implemented on FPGAs (47.0%).



(a) Inter-configuration variation for MIPUF is 47.9% (Avg = 30.7 bits / 64 bits).



(b) Intra-configuration variation for MIPUF with fuzzy extractor is 2.67% (Avg = 1.71 bits / 64 bits). Environment range from 20 °C, 0.95V to 65 °C, 1.2V.

Figure 8.2: Inter-configuration and intra-configuration variation of a MIPUF with four nodes. Each node is implemented using 64 32-bit arbiter PUFs. The interconnection between nodes is designed in a blocking fashion as shown in Figure 8.1.

We calculated a 35.37% intra-configuration variation when no error correction is applied. Consider the intra-chip variation of 64 128-bit arbiter PUFs implemented on five different FPGAs is only as little as 2.90%, MIPUF is very unstable without error correction. The reason is simple and intuitive, as all MIPUF nodes are connected in such a way that each node takes the output of the previous node as the input, an error in the first node could result in avalanche effect in intra-configuration variation. Thus, we propose to use a lightweight fuzzy extractor between every MIPUF node as an error correction mechanism [129], and the resulting intra-variation is significantly reduced to 2.67%. Since a MIPUF with n nodes requires n clock cycles to generate the result, the fuzzy extractors can be shared for all node outputs.

Figure 8.2b illustrates the probability distribution of the intra-configuration variation of the same MIPUF implemented using arbiter PUFs and 32-bit of help data in the fuzzy

extractor. The environments parameters ranging from 20 °C, 0.95V to 65 °C, 1.2V. The bars show experimental results collected on 50,000 different random interconnections. Each configuration is performed on 10,000 challenges and repeated 20 times. Noted that the major contributors to the intra-configuration variation are two extremely rare ($< 0.5\%$) case of Hamming distance greater than 40, in this case, a blacklist mechanism could be applied to improve stability. With the help of larger helper data, existing literature proves that the unreliability could be significantly reduced to 10^{-9} [129].

Our measurements are conducted on analog PUF which are unreliable by nature. In real-world applications digital PUFs, which is as stable as SRAM, are more suitable for applications with extremely high-reliability requirements at the cost of higher power consumption.

8.4.5.2 Resilience Against Modeling Attack

Several PUF-based systems are vulnerable to a variety of modeling attacks [56] [130] [131] [132] [133] [134] [135]. We observed that MIPUF significantly boost modeling attack resilience by increasing the system complexity and breaking the system linearity. Table 8.1 shows the best prediction accuracy on MIPUF vs. a variety of PUFs implemented on FPGA using attack approaches described in [56] and [58]. We observe that all prediction accuracies for a single-bit in MIPUF outperforms other designs and are all close to the ideal case of 50%. Noted that the evaluated MIPUF in this section has only four nodes. We have shown in Chapter 3 that deeper and wider interconnections between PUF systems would result in lower modeling attack accuracy, which eventually converges to 50%.

In addition to high resilience against modeling attacks, MIPUF also allows cheap and fast reconfiguration. Frequent reconfiguration of MIPUF renders modeling attacks almost impossible. We investigate this topic in more detail in Section 8.4.5.2.

Architecture	LR	ES	DL
MIPUF - 4 nodes	**51.33%**	**59.18%**	**50.59%**
256-bit 4-XOR PUF	97.21%	76.02%	78.42%
1024-bit arbiter PUF	96.57%	98.28%	88.98%
1024-bit 64-ff PUF	58.29%	95.68%	87.70%

Table 8.1: Best single-bit prediction accuracy on different PUF architectures using logistic regression (LR), evolution strategies (ES) and deep learning (DL) attacks out of 100 runs. Each attack uses 100,000 CRPs. Total number of arbiter PUF segments used in all architectures are fixed to 1,024.

8.5 Group Key Management

In this section we show that we can utilize the MIPUF structure to securely establish a group key management protocol with three major components, respectively key distribution, key storage and rekeying. Key distribution is the process to securely deliver the shared secret key to every authorized group member. After the group key has been successfully distributed, the most important task would be to securely store the secret key so that the user could easily access the key when needed, but an adversarial is forbid to peek or tamper with the secret key. Lastly, rekeying allows a group to renew or replace the group key from time to time.

To illustrate our protocol, we first define an IoT model consists of a control unit with higher computational power and multiple IoT device/nodes that are constraint by both computational power and battery life. Each IoT node embeds a MIPUF, a hardware hashing function and a very compact AES implementation.

8.5.1 Key Distribution

According to the model described above, a well-designed group key distribution protocol is proposed. The protocol is shown in Protocol 6. For each node, a group key can be delivered securely with an exchange of two messages.

Protocol 6 Group Key Distribution Protocol

- Input: A list of group member in group $\mathcal{G} = \{N_0 \cdots N_n\} \subseteq \mathcal{N}$ n being the total number of IoT nodes in the group. A random group key key_g.
- Goal: Securely deliver key_g to all $N_i \in \mathcal{G}$.

1. **Preliminary Phase**
   (a) Before the deployment of an IoT node, the control unit assigns a unique ID (N_i) to it. Initially the node derives the interconnection configuration $\gamma_i = H(N_i)$ from N_i and generates a CRP ($c_i^{\gamma_i}$, $r_i^{\gamma_i}$) using the MIPUF $\mathbb{F}_i$ where $r_i^{\gamma_i} = \mathbb{F}_i^{\gamma_i}(c_i^{\gamma_i})$.
   (b) The control unit securely store the tuple (γ_i, $c_i^{\gamma_i}$, $r_i^{\gamma_i}$) in the database, and node N_i securely stores $c_i^{\gamma_i}$ and γ_i.
2. **Key Delivery Phase**
   (a) When a group $\mathcal{G}$ is formed, the control unit first check if all group members exists based on the unique ID. If not, the protocol is aborted.
   (b) For IoT node $N_i \in \mathcal{G}$, the control unit generates a random new configuration γ_i'.
   (c) For IoT node $N_i \in \mathcal{G}$, a group key hint $p_i = r_i^{\gamma_i} \otimes key_g$ and a new configuration hint $f_i = r_i^{\gamma_i} \otimes \gamma_i'$ are generated.
   (d) An encrypted message msg_i^k containing p_i and f_i is transmitted using unicast to each group member $N_i \in \mathcal{G}$. $msg_i^k = \{E_{r_i^{\gamma_i}}(N_i \| p_i \| f_i) \| H(N_i \| c_i^{\gamma_i})\}$, "$\|$" indicates the concatenation operation, E is the encryption operation using AES and H is a hashing operation.
3. **CRP update Phase**
   (a) Upon receiving msg_i^k, IoT node N_i first decrypts the message using $r_i^{\gamma_i}$: $D_{r_i^{\gamma_i}}(E_{r_i^{\gamma_i}}(N_i \| p_i \| f_i))$, D being the AES decryption operation. N_i verifies the validity of the message by comparing the hash $H(N_i \| c_i^{\gamma_i})$. If there exists a mismatch in the hash, report error to the control unit.
   (b) The group key key_g and the new interconnection γ_i' are derived from p_i and f_i decrypted from the decrypted msg_i^k.
   (c) N_i generates a new CRP ($c_i^{\gamma_i'}$, $r_i^{\gamma_i'}$) where $c_i^{\gamma_i'} = H(c_i^{\gamma_i})$, $r_i^{\gamma_i'} = \mathbb{F}_i^{\gamma_i'}(c_i^{\gamma_i'})$. N_i sends an encrypted message $msg_i^u = E_{r_i^{\gamma_i}}(N_i \| c_i^{\gamma_i'} \| r_i^{\gamma_i'})$ back to the control unit.
   (d) The control unit decrypt msg_i^u using r_i^{γ} and updates the database by replacing the tuple (γ_i, $c_i^{\gamma_i}$, $r_i^{\gamma_i}$)$with$(γ_i', $c_i^{\gamma_i'}$, $r_i^{\gamma_i'}$). If the control unit has not received an update message msg_i^u after some predefined timeout or $c_i^{\gamma_i'} = H(c_i^{\gamma_i})$, an abort is called.

8.5.2 Key Storage

After the key distribution, the group key can be extracted from the MIPUF when the correct challenge and interconnection configuration are provided. Unlike other crypto-based key management systems, we do not directly store the group key in the memory. Instead, the group key is extracted on the fly from the group key hint p_i. We believe this approach is secure because an attacker can only obtain the real group key if he has access to both the storage (containing p_i, f_i and $c_i^{\gamma_i}$) and the MIPUF ($\mathbb{F}^{\gamma_i}$, compromising either the storage or the MIPUF does not compromise the security of the whole design. Also, the group key is only used upon receiving or transmitting group messages, thus storing the real key using low-power MIPUF is also highly energy efficient.

8.5.3 Rekeying

Group keys need to be regenerated, redistributed or updated whenever there is a dynamic change to the group to preserve security. One important motivation to rekey is that groups are not always static. When a member leaves the group, it should not be able to decrypt future group communications (**forward security**); when a new member joins, it should not be able to decrypt past group communications (**backward security**). Group key should also be completely rekeyed when potential leakage is detected for security considerations. Here we discuss all three possible cases.

8.5.3.1 New Member Joins the Group

Without loss of generality, we assume a new IoT node N_α intend to join a group $\mathcal{G}$, $N_\alpha \notin \mathcal{G}$. For efficiency considerations, redistributing a new key to all group members is expensive and inefficient. Instead, we propose to use the current secret to encrypt the new group key and this process is leakage free. Specifically, the control unit sends out a message $msg^{join} = \{E_{key_g}(key'_g)\}$ to $\forall N_i \in \mathcal{G}$. The existing group members calculate and store the new group key hint $p'_i = r_i^{\gamma_i} \otimes key'_g$ and deletes key'_g upon receiving and decrypting msg_i^{join}. The new member will have to complete the whole key distribution process described in

Section 8.5.1. Backward security is preserved using this method since the new member has no information about the old group key.

8.5.3.2 Existing Member Leaves the Group

Removing an existing member from the group is more complicated than adding a new member. Here we propose to divide group $\mathcal{G}$ into m subgroups $\mathrm{g}_j \subset \mathcal{G} = \{\mathrm{g}_1 \cdots \mathrm{g}_m\}, 1 \leq j \leq m$. All nodes in the same subgroup share the same interconnection configuration $\gamma_{\mathrm{g_j}}$. Again, without losing the generality, we assume an IoT node $N_\beta \in \mathrm{g}_j \subset \mathcal{G}$ intend to leave the subgroup where all members in the subgroup use the same MIPUF interconnection configuration γ_j. The control node first multicast/broadcast $m-1$ messages $msg_i^{leave} = \{E_{\gamma_i}(key'_g \| H(\gamma_i)\}$ containing the new key to all the subgroups encrypted using the configuration $\gamma_i, i \neq j, 1 \leq i \leq m$. Upon receiving the message, each node first decrypts the message using its own configuration γ_i and check if H(γ_i) matches the one in the decrypted msg_i^{leave}. If so then the decrypted new group key key'_g is valid, otherwise, discard the message. No member of g_j including the leaving node have any knowledge of the configurations of other subgroups, thus incapable of decrypting the message correctly. The control unit should then perform unicast communications to all members of g_j by distributing the new group key key'_g and a new configuration γ'_j to replace γ_j.

8.5.3.3 Complete Rekeying

MIPUF can still be modeled if a significantly large enough set of CRPs is collected. However, MIPUF can be reconfigured to neutralize modeling attacks by completely remap the input-output mapping. We propose to perform a full rekeying once the total number of CRPs generated exceeds a calculated sample complexity lower bound that equals to the sufficient training set size to break the MIPUF. Equation 8.1 describe a sample size lower bound in terms of the IPN model parameters, where m is the number of nodes in MIPUF and n is the maximum number of PUFs in a MIPUF node. $k = VC(\mathbb{F})$ where VC is the Vapnik-Chervonenkis-dimension and $\mathbb{F}$ is the largest single PUF in MIPUF. δ is the failure

probability and ϵ is the learning error.

$$\text{Sample complexity} \sim \frac{(m \cdot k + m) \cdot n + ln(\frac{1}{\delta})}{\epsilon} \tag{8.1}$$

8.6 Evaluation

8.6.1 Security Analysis

We make the following assumptions for our security analysis. The physical security of the MIPUF is secured; however, an attacker is allowed to query the CRPs as much as needed. The wireless channels used for communication are not secured after the initial preliminary phase. The hash function and compact AES on each node are secure. The control unit key database is secure. We summarize our security analysis against several popular attacks as below:

Eavesdropping Attack: During the key distributing and rekeying process, all messages containing γ_i, $c_i^{\gamma_i}$, $r_i^{\gamma_i}$ or key_g are encrypted by AES. Thus eavesdropping attack is invalid.

Man-in-the-middle Attack: Before updating the new CRP in **3c** in Protocol 6, the new challenge is a one-way hash of the previous challenge which is checked by the control unit, thus rendering the attack useless.

Replay Attack: Neither the IoT node nor the control unit would be able to correctly decrypt a message encrypted using a previous response since old responses are discarded after the update.Thus the hash check in **3a**, **3d** in Protocol 6 would fail during key distribution. Forward security in the rekeying process is designed to protect the system from such attacks.

Impersonation Attack: Based on our assumption, the preliminary phase is secure thus the initial key and MIPUF configuration are secured. Also, the modeling attack resilience and the reconfigurability of MIPUF prevents an attacker to impersonate an IoT node even if we allow him to query the CRPs.

Comparing to other designs (e.g. ECC-based scheme) our group key management scheme enjoys at least the same level of security while adding an additional level of security at the

physical level utilizing the unclonability properties of MIPUF.

8.6.2 Overhead Evaluation

In this section we assume that there are N nodes in the group and the group is split into M subgroups. The MIPUF we use in each IoT nodes takes an **a**-bit challenge, a **b**-bit configuration vector and generates a **c**-bit output (assuming $\mathbf{b} \geq \mathbf{c}$). Node ID is a **l**-bit vector. The hash function hashes any input to an **a**-bit string. The group key has length of **c**-bits. The random number generator cost E_R units of energy per operation. The energy consumption of MIPUF, hash function, random number generator, XOR operation and the very compact AES are $\boldsymbol{E_P}$, $\boldsymbol{E_H}$, $\boldsymbol{E_R}$, $\boldsymbol{E_X}$ and $\boldsymbol{E_A}$.

Communication Cost: The length of messages: $\boldsymbol{msg_i^k}$, $\boldsymbol{msg_i^u}$, $\boldsymbol{msg^{join}}$ and $\boldsymbol{msg_i^{leave}}$ are: $\mathbf{a+b+c+l}$, $\mathbf{a+c+l}$, **a+c** and **a+c**. Thus the total number of messages need to be sent for key distribution and node join/leave rekeying are: **N**, **3** and $(\frac{\mathbf{2N}}{\mathbf{M}}\mathbf{-2})+\mathbf{(M-1)}$. For node leave rekeying, minimum cost is achieved when $\boldsymbol{M=\sqrt{N}}$.

Storage Overhead: The control unit stores the Node ID, CRPs and the current configuration of all nodes; thus the storage overhead at the control unit is $\boldsymbol{N}\cdot\mathbf{(a+b+2c+l)}$ bits. Each IoT node stores the Node ID, current challenge, current configuration and the group key hint which has a storage overhead of **a+b+c+l** bits.

Energy Cost: The control unit spends $\mathbf{2}\boldsymbol{E_A}+\mathbf{2}\boldsymbol{E_H}+\mathbf{2}\boldsymbol{E_R}+\mathbf{2}\boldsymbol{E_X}$ units of energy to distribute the group key to one node. Each node spends $\mathbf{2}\boldsymbol{E_A}+\mathbf{2}\boldsymbol{E_H}+\boldsymbol{E_P}+\mathbf{2}\boldsymbol{E_R}$ to receive the key and update the CRP. During member join rekeying, the control units spends $\boldsymbol{E_A}+\boldsymbol{E_R}$ units of energy to update the group key to existing members and $\mathbf{2}\boldsymbol{E_A}+\mathbf{2}\boldsymbol{E_H}+\mathbf{2}\boldsymbol{E_R}+\mathbf{2}\boldsymbol{E_X}$ to the new member. The new node spends $\mathbf{2}\boldsymbol{E_A}+\mathbf{2}\boldsymbol{E_H}+\boldsymbol{E_P}+\mathbf{2}\boldsymbol{E_X}$ and old members spend $\boldsymbol{E_A}+\boldsymbol{E_X}$ units of energy respectively. During member leave rekeying, the control unit spends $\boldsymbol{(M-1)}\cdot\boldsymbol{(E_A+E_H+E_R)}$ units of energy to update the group key to existing members that are not in the same subgroup as the leaving node and $(\frac{\boldsymbol{N}}{\boldsymbol{M}}\boldsymbol{-1})\cdot\boldsymbol{(2E_A+2E_H+2E_R+2E_X)}$ to the update all members in the same subgroup. All members that are in and not in the same subgroup as the leaving node spends $\boldsymbol{(M-1)\cdot(E_A+E_H+E_R)}$ and $\boldsymbol{E_A+E_H}$

units of energy.

We compare our global energy consumption to two other key management protocols: Localized Encryption and authentication protocol (LEAP) [113] and Elliptic Curve Public Key Cryptography (ECPKC) [115] in simulation using the parameters described in [115] for a fair comparison. The energy consumption parameters for our design are estimated from our implementation described in Section 8.6.3. The comparison for simulated results for global energy consumption for three key management schemes can be seen in Figure 8.3. LEAP uses significantly much more energy than both ECPKC and our proposed scheme as it grows quadratically. The global energy consumption of both ECPKC and our proposed scheme grows linearly. Since our proposed design uses low-power MIPUF instead of energy-hungry ECC to achieve power efficiency. We observe that our proposed scheme uses about 47.33% less energy for key distribution.

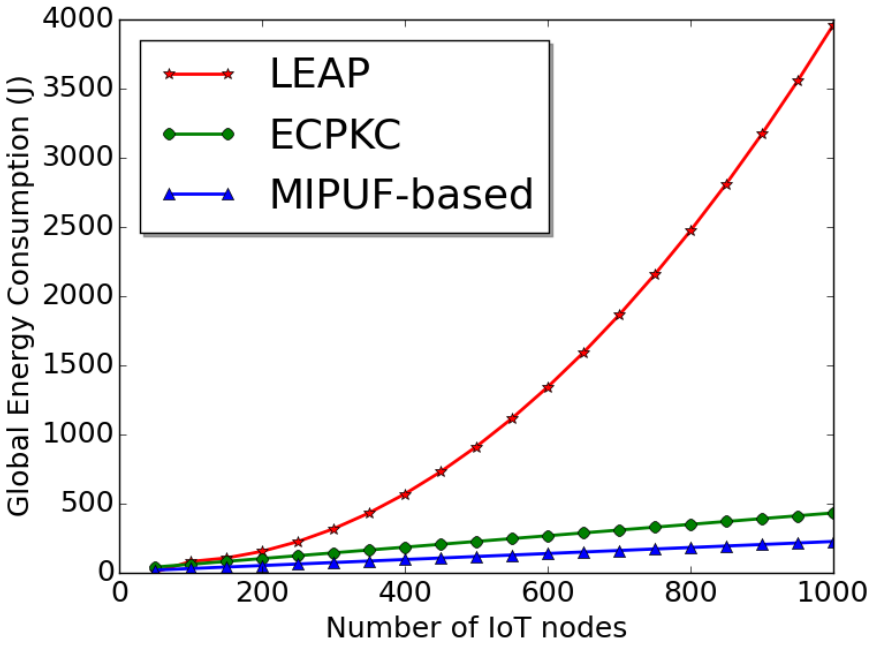


Figure 8.3: Simulated global energy consumption (J) vs. total number of IoT nodes under the settings introduced in [115].

8.6.3 Implementation Results

We implemented our key management hardware support for IoT nodes on Xilinx Spartan-6 LX45 FPGAs to measure the area and power. Our implementation consists of a MIPUF with three nodes each consists of 128 32-bit Arbiter PUFs. The fuzzy extractor, the hash function and the AES module are implemented based on [129] [92] and [136] accordingly. Table 8.2 shows the area and power overhead break down of our implementation. Our hardware support design is 4.3× smaller and 4.2× power efficient comparing to state-of-the-art ECC design on FPGA. Our MIPUF seems to be more expensive due to FPGA-based arbiter PUF implementations are known to be inefficient. The overhead of MIPUF and the hardware support are expected to be significantly reduced if implemented on ASIC. We are also expected to see further improvement if MIPUF is built using more efficient and advanced strong PUFs.

Our design	MIPUF	SHA-1	AES	Overall	Efficient ECC alone [137]
LUTs	3,717	1,151	598	4,553	16,090
Flip-flops	7,028	1,590	501	9,219	3,747
Slices	1,626	544	222	3,401	11,777
Block RAMs	0	0	3	3	0
Power($\boldsymbol{mW}$)	123.7	30.4	16.2	175.7	515.9

Table 8.2: FPGA resource and power characteristics of the hardware support of our proposed key management scheme implemented on Spartan-6 FPGA.

8.6.4 Impact on IoT Design

According to Singelée et al. [138], the communication energy cost should be the major component in the total energy budget. We believe that power and energy per bit are good indications of how low energy security modules should be. Table 8.3 shows that the numbers vary by different standards

In our key management protocol, the MIPUF needs to be executed whenever there is a message needs to be transferred or received. Regardless of the size of the message, the execution of MIPUF is only performed once. The total amount of MIPUF energy consumed

Wireless	Throughput	Freq.	Power		Energy/bit	
	(kbps)	(Ghz)	TX (mW)	RX (mW)	TX (nJ/bit)	RX (nJ/bit)
802.11G[139]	54,000	2.4	2,300	1,900	42.59	35.19
Zigbee[140]	250	2.4	46.44	33.30	185.76	133.20
Bluetooth Classic[141]	2,100	2.4	99.90	67.50	47.57	32.14
BLE[142]	1,000	2.4	48.90	39.20	48.00	39.20

Table 8.3: Performance and energy comparison of different wireless standards in IoT.

is measured at as low as 17nJ per MIPUF bit, which is 64.58% lower than the 48 nJ/bit of the most energy efficient wireless standard - Bluetooth Low Energy (BLE). In addition, we expect the power and energy consumption to further drop by at least 10× when implemented on ASIC [143]. We estimated that if MIPUF is implemented on ASIC and the key length is 255-bit (commonly used in Curve25519 [144]), the total amount of energy consumed per transaction is at most 433.48 nJ, and this amount of energy is **108**× lower than the minimum Bluetooth 4.0 transaction of 45,000 nJ on a 1.5V battery [145]. We claim that it is safe to conclude that our key management design is indeed low power, low energy and applicable to real-world IoT device designs.

8.7 Chapter Conclusions

In this chapter, we first proposed a novel PUF structure: MIPUF that is both secure and reconfigurable. We showcased the uniqueness, reliability, modeling attack resilience and reconfigurability of MIPUF. We then proposed a group key management scheme in IoT consists of key distribution, key storage and rekeying based on MIPUF. Security and overhead analysis on the scheme show that our design is not only secure against multiple attack methods but also low power. Our simulation result indicates that our proposed scheme spends 47.33% less energy compared to the state-of-the-art crypto-based scheme ECPKC [115] since we use low-power and energy efficient MIPUF instead of power-hungry ECC.

CHAPTER 9

Concluding Remarks

Decades of development in IoT technology has already turned "connect everything" from a motivational slogan and an inspiring vision to reality. What comes alongside with IoT prosperity are also challenges and opportunities in infrastructure, device design, and applications at multiple levels. In this thesis, we sought to make exploration in a critical but often overlooked aspect of IoT research: hardware security. While many traditional security approaches and mechanisms seem applicable on IoT, the compact and energy constrained nature of IoT devices greatly challenges the compatibility of these existing methods. Modern IoT devices require low-power, small area, reliable and low maintenance security subsystems to protect and the sensitive information that are collected, stored or flowed through them.

At the beginning of this thesis, we established three objectives, respectively the security objectives, the energy objectives, and the applicability objectives. To meet these objectives, we focus on proposing new security primitives and protocols, applying energy reduction techniques and designing secure and low-power security applications throughout the entire thesis.

When addressing the security objectives, we first studied a promising low-power hardware security primitive - PUF. We first conduct a thorough evaluation of analog PUF and its variations by creating a stable emulation platform. We exposed the randomness and reliability problems in analog PUFs and proved that we could create a stable clone of an analog PUF with at least 87.42% of accuracy. Realizing the vulnerability against modeling attack and the environmentally sensitive problem of many PUF designs, we propose two solutions to resolve the issue. The first one is enhancing modeling attack resilience through PUF interconnection and reconfiguration. Interconnecting smaller PUFs dramatically boosts

the system complexity and linearity, increasing the difficulty of accurate statistical modeling by several magnitudes. Besides, we propose to reconfigure the interconnection whenever the total number of CRPs generated by PUF exceeds the lower bound of the sampling complexity. This approach guarantees that no attacker could collect sufficient amount of training data, resulting in less than 53.19% of single-bit prediction accuracy against a wide range of modeling attacks. The second solution we propose to address PUF's weakness is to optimize PUF connections using evolution strategies. We observed that different PUF connections often lead to different output randomness and stability. Utilizing our proposed evolution strategies algorithm, we could quickly find a near-optimal connection that achieves 220.8% improvement in output randomness and 22.62% improvement in reliability. Aside from PUFs, we also proposed content-driven reconfigurable injective functions that achieve secure encryption/decryption between IoT devices with 75.04% power savings comparing to cryptographic approaches.

In an attempt to meet the energy objectives, we propose the idea of logic free-riding. We modified the current PUF implementation on FPGA platforms and allowed an arbitrary logic to share the hardware with an existing PUF implementation through signal phase encoding, saving 40.4% of area and 7.69% of power.

Lastly, to meet the applicability objectives, we introduced two novel security applications. The first application is a PUF-based anomaly detection device that reports suspicious environmental variations inferred from changes in environmentally sensitive CRPs. Compared to commercialized Xilinx SYSMON, our design uses 63% less FPGA area and 13% of power, detecting 100% of applied abnormalities with a false positive rate as low as 1.1%. The second application utilized MIPUF to provide a robust, secure and low-power solution to all process in the group key management process. Our simulation result indicates that our proposed scheme spends 47.33% less global energy compared to the state-of-the-art crypto-based scheme while providing additional protection at the hardware level.

Though challenges remain in the domain of hardware-oriented IoT security, we have made our contribution towards a more robust and energy efficient hardware security system in IoT.

REFERENCES

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[2] K. L. Lueth, "State of the IoT 2018: Number of IoT devices now at 7B Market accelerating," Aug 2018.

[3] M. Torchia and M. Shirer, "IDC Forecasts Worldwide Spending on the Internet of Things to Reach $745 Billion in 2019, Led by the Manufacturing, Consumer, Transportation, and Utilities Sectors," Jan 2019.

[4] S. P. Skorobogatov, *Semi-invasive Attacks: a New Approach to Hardware Security Analysis*. PhD thesis, Citeseer, 2005.

[5] S. Shin and J. Lipton, "Security Researchers Say They Can Hack Medtronic Pacemakers," Aug 2018.

[6] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *Annual International Cryptology Conference*, pp. 388–397, Springer, 1999.

[7] S. Meguerdichian and M. Potkonjak, "Matched Public PUF: Ultra Low Energy Security Platform," in *Proceedings of the 17th IEEE/ACM international symposium on Low-power electronics and design*, pp. 45–50, IEEE Press, 2011.

[8] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, "Physical One-way Functions," *Science*, vol. 297, no. 5589, pp. 2026–2030, 2002.

[9] B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas, "Silicon Physical Random Functions," in *Proceedings of the 9th ACM conference on Computer and communications security*, pp. 148–160, ACM, 2002.

[10] G. E. Suh and S. Devadas, "Physical Unclonable Functions for Device Authentication and Secret Key Generation," in *Proceedings of the 44th annual Design Automation Conference*, pp. 9–14, ACM, 2007.

[11] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, *FPGA Intrinsic PUFs and Their Use for IP Protection*. Springer, 2007.

[12] S. S. Kumar, J. Guajardo, R. Maes, G.-J. Schrijen, and P. Tuyls, "The Butterfly PUF Protecting IP on Every FPGA," in *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on*, pp. 67–70, IEEE, 2008.

[13] T. Xu and M. Potkonjak, "Robust and Flexible FPGA-based Digital PUF," in *24th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–6, IEEE, 2014.

[14] A. Maiti, I. Kim, and P. Schaumont, "A Robust Physical Unclonable Function with Enhanced Challenge-response Set," *Information Forensics and Security, IEEE Transactions on*, vol. 7, no. 1, pp. 333–345, 2012.

[15] B. Srinivasu, P. Vikramkumar, A. Chattopadhyay, and K.-Y. Lam, "CoLPUF: A Novel Configurable LFSR-based PUF," in *2018 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, pp. 358–361, IEEE, 2018.

[16] W. Liu, L. Zhang, Z. Zhang, C. Gu, C. Wang, M. O'neill, and F. Lombardi, "XOR-based Low-Cost Reconfigurable PUFs for IoT Security," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 3, p. 25, 2019.

[17] A. Wali, A. Dodda, Y. Wu, A. Pannone, L. K. R. Usthili, S. K. Ozdemir, I. T. Ozbolat, and S. Das, "Biological Physically Unclonable Function," *Communications Physics*, vol. 2, no. 1, p. 39, 2019.

[18] F. Dan, Y. Xu, Z. Li, J. Wen, B. Liu, S. Chen, and B. Li, "A Modeling Attack Resistant R-XOR APUF Based on FPGA," in *2018 IEEE 3rd International Conference on Signal and Image Processing (ICSIP)*, pp. 577–581, IEEE, 2018.

[19] D. B. Roy, S. Bhasin, I. Nikolić, and D. Mukhopadhyay, "Combining PUF with RLUTs: A Two-party Pay-per-device IP Licensing Scheme on FPGAs," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 2, p. 12, 2019.

[20] Y. Nozaki and M. Yoshikawa, "Side-Channel Resistance Evaluation Method using Statistical Tests for Physical Unclonable Function," in *2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, pp. 189–194, IEEE, 2019.

[21] L. Bolotnyy and G. Robins, "Physically Unclonable Function-based Security and Privacy in RFID Systems," in *Pervasive Computing and Communications, 2007. PerCom'07. Fifth Annual IEEE International Conference on*, pp. 211–220, IEEE, 2007.

[22] U. Rührmair, "SIMPL Systems: On a Public Key Variant of Physical Unclonable Functions," *IACR Cryptology ePrint Archive*, vol. 2009, p. 255, 2009.

[23] M.-D. Yu and S. Devadas, "Secure and Robust Error Correction for Physical Unclonable Functions," *IEEE Design & Test of Computers*, vol. 27, no. 1, pp. 48–65, 2010.

[24] J. X. Zheng, D. Li, and M. Potkonjak, "A Secure and Unclonable Embedded System using Instruction-Level PUF Authentication," in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–4, IEEE, 2014.

[25] S. Devadas, E. Suh, S. Paral, R. Sowell, T. Ziola, and V. Khandelwal, "Design and Implementation of PUF-based 'Unclonable' RFID ICs for Anti-counterfeiting and Security Applications," in *RFID, 2008 IEEE International Conference on*, pp. 58–64, IEEE, 2008.

[26] H. Gu, T. Xu, and M. Potkonjak, "An Energy-Efficient PUF Design: Computing While Racing," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, pp. 142–147, ACM, 2016.

[27] H. Gu, T. Xu, and M. Potkonjak, "A Low-Power APUF-based Environmental Abnormality Detection Framework," in *Low Power Electronics and Design (ISLPED, 2017 IEEE/ACM International Symposium on*, pp. 1–6, IEEE, 2017.

[28] J. Zhang, Y. Lin, Y. Lyu, and G. Qu, "A PUF-FSM Binding Scheme for FPGA IP Protection and Pay-per-device Licensing," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 6, pp. 1137–1150, 2015.

[29] T. Xu, H. Gu, and M. Potkonjak, "An Ultra-Low Energy PUF Matching Security Platform using Programmable Delay Lines," in *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2016 11th International Symposium on*, pp. 1–8, IEEE, 2016.

[30] T. Xu and M. Potkonjak, "Stable and Secure Delay-based Physical Unclonable Functions using Device Aging," in *Circuits and Systems (ISCAS), 2015 IEEE International Symposium on*, pp. 33–36, IEEE, 2015.

[31] Y. Gao, Y. Su, W. Yang, S. Chen, S. Nepal, and D. C. Ranasinghe, "Building Secure SRAM PUF Key Generators on Resource Constrained Devices," *arXiv preprint arXiv:1902.03031*, 2019.

[32] Z. Huang and Q. Wang, "A PUF-based Unified Identity Verification Framework for Secure IoT Hardware via Device Authentication," *World Wide Web*, pp. 1–32, 2019.

[33] P. Gope, J. Lee, and T. Q. Quek, "Lightweight and Practical Anonymous Authentication Protocol for RFID Systems Using Physically Unclonable Functions," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 11, pp. 2831–2843, 2018.

[34] S. Tajik, J. Fietkau, H. Lohrke, J.-P. Seifert, and C. Boit, "Pufmon: Security Monitoring of FPGAs Using Physically Unclonable Functions," in *2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pp. 186–191, IEEE, 2017.

[35] M. N. Aman, K. C. Chua, and B. Sikdar, "Mutual Authentication in IoT Systems Using Physical Unclonable Functions," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1327–1340, 2017.

[36] T. Xu, H. Gu, and M. Potkonjak, "Data Protection Using Recursive Inverse Function," in *Field Programmable Logic and Applications (FPL), 2015 25th International Conference on*, pp. 1–4, IEEE, 2015.

[37] T. Hui and R. W. Mounger, "Programmable Delay Line," Aug. 3 1999. US Patent 5,933,039.

[38] M. Majzoobi, F. Koushanfar, and S. Devadas, "FPGA PUF Using Programmable Delay Lines," in *Information Forensics and Security (WIFS), 2010 IEEE International Workshop on*, pp. 1–6, IEEE, 2010.

[39] G. M. Godfrey, "Technique for Controlling System Bus Timing With On-chip Programmable Delay Lines," Sept. 11 2001. US Patent 6,289,468.

[40] X. Wu and K. Sengupta, "Programmable Picosecond Pulse Generator in CMOS," in *2015 IEEE MTT-S International Microwave Symposium*, pp. 1–4, IEEE, 2015.

[41] T. Polzer, F. Huemer, and A. Steininger, "A Programmable Delay Line for Metastability Characterization in FPGAs," in *2016 Austrochip Workshop on Microelectronics (Austrochip)*, pp. 51–56, IEEE, 2016.

[42] M.-C. Tsai, C.-H. Cheng, and C.-M. Yang, "An All-Digital High-Precision Built-in Delay Time Measurement Circuit," in *VLSI Test Symposium, 2008. VTS 2008. 26th IEEE*, pp. 249–254, IEEE, 2008.

[43] A. Raychowdhury, S. Ghosh, and K. Roy, "A Novel On-Chip Delay Measurement Hardware for Efficient Speed-binning," in *On-Line Testing Symposium, 2005. IOLTS 2005. 11th IEEE International*, pp. 287–292, IEEE, 2005.

[44] M. Majzoobi, E. Dyer, A. Elnably, and F. Koushanfar, "Rapid FPGA Characterization Using Clock Synthesis and Signal Sparsity," in *International Test Conference (ITC)*, pp. 1–10, 2010.

[45] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling Attacks on Physical Unclonable Functions," in *Proceedings of the 17th ACM conference on Computer and communications security*, pp. 237–249, ACM, 2010.

[46] U. Ruhrmair and M. van Dijk, "PUFs in Security Protocols: Attack Models and Security Evaluations," in *Security and Privacy (SP), 2013 IEEE Symposium on*, pp. 286–300, IEEE, 2013.

[47] F. Ganji, S. Tajik, and J.-P. Seifert, "Let Me Prove It to You: RO PUFs Are Provably Learnable," in *Information Security and Cryptology-ICISC 2015*, pp. 345–358, Springer, 2015.

[48] M. Potkonjak and V. Goudar, "Public Physical Unclonable Functions," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1142–1156, 2014.

[49] S. Tajik, E. Dietz, S. Frohmann, J.-P. Seifert, D. Nedospasov, C. Helfmeier, C. Boit, and H. Dittrich, "Physical Characterization of Arbiter PUFs," in *Cryptographic Hardware and Embedded Systems–CHES 2014*, pp. 493–509, Springer, 2014.

[50] A. Mahmoud, U. Rührmair, M. Majzoobi, and F. Koushanfar, "Combined Modeling and Side Channel Attacks on Strong PUFs," *IACR Cryptology ePrint Archive*, vol. 2013, p. 632, 2013.

[51] C. Helfmeier, C. Boit, D. Nedospasov, and J.-P. Seifert, "Cloning Physically Unclonable Functions," in *Hardware-Oriented Security and Trust (HOST), 2013 IEEE International Symposium on*, pp. 1–6, IEEE, 2013.

[52] T. Xu, D. Li, and M. Potkonjak, "Adaptive Characterization and Emulation of Delay-based Physical Unclonable Functions Using Statistical Models," in *Proceedings of the 52nd Annual Design Automation Conference*, pp. 76–81, ACM, 2015.

[53] S. Gehrer and G. Sigl, "Reconfigurable PUFs for FPGA-based SoCs," in *Integrated Circuits (ISIC), 2014 14th International Symposium on*, pp. 140–143, IEEE, 2014.

[54] E. Öztürk, G. Hammouri, and B. Sunar, "Towards Robust Low Cost Authentication for Pervasive Devices," in *Pervasive Computing and Communications, 2008. PerCom 2008. Sixth Annual IEEE International Conference on*, pp. 170–178, IEEE, 2008.

[55] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Testing Techniques for Hardware Security," in *Test Conference, 2008. ITC 2008. IEEE International*, pp. 1–10, IEEE, 2008.

[56] U. Rührmair, J. Sölter, F. Sehnke, X. Xu, A. Mahmoud, V. Stoyanova, G. Dror, J. Schmidhuber, W. Burleson, and S. Devadas, "PUF Modeling Attacks on Simulated and Silicon Data," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 11, pp. 1876–1891, 2013.

[57] A. Vijayakumar, V. C. Patil, C. B. Prado, and S. Kundu, "Machine Learning Resistant Strong PUF: Possible or a Pipe Dream?," in *Hardware Oriented Security and Trust (HOST), 2016 IEEE International Symposium on*, pp. 19–24, IEEE, 2016.

[58] R. Yashiro, T. Machida, M. Iwamoto, and K. Sakiyama, "Deep-Learning-based Security Evaluation on Authentication Systems Using Arbiter PUF and Its Variants," in *International Workshop on Security*, pp. 267–285, Springer, 2016.

[59] G. Clarke, D. Van Dijk, and S. Devadas, "Controlled Physical Random Functions," *Proceedings. 18th Annual*, pp. 149–160, 2002.

[60] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and Robust Automated Machine Learning," in *Advances in Neural Information Processing Systems 28* (C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, eds.), pp. 2962–2970, Curran Associates, Inc., 2015.

[61] "Evolutionary Computation: An Introduction ecspy 1.1 Documentation." `https://pythonhosted.org/ecspy/ec_intro.html`, Retrieved Jun 15, 2017.

[62] F. Chollet *et al.*, "Keras." `https://github.com/fchollet/keras`, 2015.

[63] S. Meguerdichian and M. Potkonjak, "Device Aging-based Physically Unclonable Functions," in *Proceedings of the 48th Design Automation Conference*, pp. 288–289, ACM, 2011.

[64] C. Zhou, K. K. Parhi, and C. H. Kim, "Secure and reliable xor arbiter puf design: An experimental study based on 1 trillion challenge response pair measurements," in *Proceedings of the 54th Annual Design Automation Conference 2017*, p. 10, ACM, 2017.

[65] C. W. O'Donnell, G. E. Suh, and S. Devadas, "PUF-based Random Number Generation," *In MIT CSAIL CSG Technical Memo*, vol. 481, 2004.

[66] A. Maiti, R. Nagesh, A. Reddy, and P. Schaumont, "Physical Unclonable Function and True Random Number Generator: a Compact and Scalable Implementation," in *Proceedings of the 19th ACM Great Lakes symposium on VLSI*, pp. 425–428, ACM, 2009.

[67] M. Bhargava and K. Mai, "A High Reliability PUF Using Hot Carrier Injection Based Response Reinforcement," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 90–106, Springer, 2013.

[68] A. Alvarez, W. Zhao, and M. Alioto, "14.3 15fJ/b Static Physically Unclonable Functions for Secure Chip Identification with $< 2\%$ Native Bit Instability and $140\times$ Inter/Intra PUF Hamming Distance Separation in 65nm," in *Solid-State Circuits Conference-(ISSCC), 2015 IEEE International*, pp. 1–3, IEEE, 2015.

[69] V. Van Der Leest, B. K. B. Preneel, and E. Van Der Sluis, "Physically Unclonable Function (PUF) with Improved Error Correction," July 19 2016. US Patent 9,396,357.

[70] Y. Dodis, L. Reyzin, and A. Smith, "Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data," in *International conference on the theory and applications of cryptographic techniques*, pp. 523–540, Springer, 2004.

[71] B. Pedersen, "Secure physically unclonable function (puf) error correction," Oct. 12 2017. US Patent App. 15/633,572.

[72] W. Yan, F. Tehranipoor, and J. A. Chandy, "Puf-based fuzzy authentication without error correcting codes," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 9, pp. 1445–1457, 2017.

[73] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, and E. Barker, "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications," tech. rep., DTIC Document, 2001.

[74] D. Markovic, C. C. Wang, L. P. Alarcon, T.-T. Liu, and J. M. Rabaey, "Ultra Low-Power Design in Near-threshold Region," *Proceedings of the IEEE*, vol. 98, no. 2, pp. 237–252, 2010.

[75] G. Marsaglia, "The Diehard Random Number Test Suite," *http://stat.fsu.edu/pub/diehard*, 1997.

[76] R. G. Brown, D. Eddelbuettel, and D. Bauer, "Dieharder: A Random Number Test Suite," *Duke University Physics Department*, 2009.

[77] J. Soto, "Statistical Testing of Random Number Generators," in *Proceedings of the 22nd National Information Systems Security Conference*, vol. 10, p. 12, NIST Gaithersburg, MD, 1999.

[78] J. W. Lee, D. Lim, B. Gassend, G. E. Suh, M. Van Dijk, and S. Devadas, "A Technique to Build a Secret Key in Integrated Circuits for Identification and Authentication Applications," in *VLSI Circuits, 2004. Digest of Technical Papers. 2004 Symposium on*, pp. 176–179, IEEE, 2004.

[79] R. Canetti, "Universally Composable Security: A New Paradigm for Cryptographic Protocols," in *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pp. 136–145, IEEE, 2001.

[80] D. Coppersmith, "The Data Encryption Standard (DES) and its Strength against Attacks," *Journal of research and development*, vol. 38, no. 3, pp. 243–250, 1994.

[81] N. Beckmann and M. Potkonjak, "Hardware-based Public-Key Cryptography with Public Physically Unclonable Functions," in *Information Hiding*, pp. 206–220, Springer, 2009.

[82] J. Rajendran, G. S. Rose, R. Karri, and M. Potkonjak, "Nano-PPUF: A Memristor-based Security Primitive," in *Computer Society Annual Symposium on VLSI*, pp. 84–87, IEEE, 2012.

[83] S. Morioka and A. Satoh, "An Optimized S-Box Circuit Architecture for Low Power AES Design," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 172–186, Springer, 2002.

[84] P. Hamalainen, T. Alho, M. Hannikainen, and T. D. Hamalainen, "Design and Implementation of Low-Area and Low-Power AES Encryption Hardware Core," in *9th EUROMICRO conference on digital system design (DSD'06)*, pp. 577–583, IEEE, 2006.

[85] E. Trichina, T. Korkishko, and K. H. Lee, "Small Size, Low Power, Side Channel-Immune AES Coprocessor: Design and Synthesis Results," in *International Conference on Advanced Encryption Standard*, pp. 113–127, Springer, 2004.

[86] X. Zheng, Z. Liu, and B. Peng, "Design and Implementation of an Ultra Low Power RSA Coprocessor," in *4th International Conference on Wireless Communications, Networking and Mobile Computing*, pp. 1–5, IEEE, 2008.

[87] P. Yalla and J.-P. Kaps, "Lightweight Cryptography for FPGAs," in *Reconfigurable Computing and FPGAs, 2009. ReConFig'09. International Conference on*, pp. 225–230, IEEE, 2009.

[88] M. Srinivasan and G. Tamilselvan, "VLSI Implementation of Low Power High Speed ECC Processor using Versatile Bit Serial Multiplier," *Journal of Circuits, Systems and Computers*, vol. 26, no. 07, 2017.

[89] U. Banerjee, C. Juvekar, A. Wright, A. P. Chandrakasan, *et al.*, "An Energy-Efficient Reconfigurable DTLS Cryptographic Engine for End-to-End Security in IoT Applications," in *International Solid-State Circuits Conference-(ISSCC)*, pp. 42–44, IEEE, 2018.

[90] K. Kepa, F. Morgan, K. Kosciuszkiewicz, and T. Surmacz, "Serecon: A Secure Dynamic Partial Reconfiguration Controller," in *Computer Society Annual Symposium on VLSI*, pp. 292–297, IEEE, 2008.

[91] R. Horstmeyer, S. Assawaworrarit, U. Ruhrmair, and C. Yang, "Physically Secure and Fully Reconfigurable Data Storage using Optical Scattering," in *International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 157–162, IEEE, 2015.

[92] I. Kawazome, "Secure Hash." `https://github.com/ikwzm/SECURE_HASH`, 2013.

[93] U. Farooq and M. F. Aslam, "Comparative Analysis of Different AES Implementation Techniques for Efficient Resource Usage and Better Performance of an FPGA," *Journal of King Saud University-Computer and Information Sciences*, vol. 29, no. 3, pp. 295–302, 2017.

[94] K. Gaj and P. Chodowiec, "Comparison of the Hardware Performance of the AES Candidates Using Reconfigurable Hardware," in *AES Candidate Conference*, pp. 40–54, 2000.

[95] H. Satyanarayana, "Opencore 128-bit AES," Dec 2004.

[96] J. Van Dyken and J. G. Delgado-Frias, "FPGA Schemes for Minimizing the Power-Throughput Trade-off in Executing the Advanced Encryption Standard Algorithm," *Journal of Systems Architecture*, vol. 56, no. 2-3, pp. 116–123, 2010.

[97] F. James, "A Review of Pseudorandom Number Generators," *Computer Physics Communications*, vol. 60, no. 3, pp. 329–344, 1990.

[98] B. Jun and P. Kocher, "The Intel Random Number Generator," *Cryptography Research Inc. white paper*, 1999.

[99] C. S. Petrie and A. Connelly, "A Noise-based IC Random Number Generator for Applications in Cryptography," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, vol. 47, no. 5, pp. 615–621, 2000.

[100] P. Kohlbrenner and K. Gaj, "An Embedded True Random Number Generator for FPGAs," in *Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, pp. 71–78, ACM, 2004.

[101] P. P. Chu and R. E. Jones, "Design Techniques of FPGA-based Random Number Generator," in *Military and Aerospace Applications of Programmable Devices and Technologies Conference*, vol. 1, pp. 28–30, Citeseer, 1999.

[102] Xilinx, *Virtex-5 FPGA System Monitor User Guide*. Xilinx.

[103] A. Le Masle and W. Luk, "Detecting Power Attacks on Reconfigurable Hardware," in *22nd International Conference on Field Programmable Logic and Applications (FPL)*, pp. 14–19, IEEE, 2012.

[104] M. Happe, H. Hangmann, A. Agne, and C. Plessl, "Eight Ways to Put your FPGA on Fire A Systematic Study of Heat Generators," in *Reconfigurable Computing and FPGAs (ReConFig), 2012 International Conference on*, pp. 1–6, IEEE, 2012.

[105] Xilinx, *Virtex-5 FPGA Data Sheet:DC and Switching Characteristics*. Xilinx.

[106] "Cryptographic Hardware Project - Cores." `http://www.aoki.ecei.tohoku.ac.jp/crypto/web/cores.html`. Accessed: 2017-01-12.

[107] J. Guo, T. Xu, T. Stavrinos, and M. Potkonjak, "Enabling Environmentally-Powered Indoor Sensor Networks with Dynamic Routing and Operation," in *Power and Timing Modeling, Optimization and Simulation (PATMOS), 2016 26th International Workshop on*, pp. 213–220, IEEE, 2016.

[108] L. Veltri, S. Cirani, S. Busanelli, and G. Ferrari, "A Novel Batch-based Group Key Management Protocol Applied to the Internet of Things," *Ad Hoc Networks*, vol. 11, no. 8, pp. 2724–2737, 2013.

[109] Y. Challal and H. Seba, "Group Key Management Protocols: A Novel Taxonomy," *International journal of information technology*, vol. 2, no. 1, pp. 105–118, 2005.

[110] M. Waldvogel, G. Caronni, D. Sun, N. Weiler, and B. Plattner, "The VersaKey Framework: Versatile Group Key Management," *IEEE Journal on selected areas in communications*, vol. 17, no. 9, pp. 1614–1631, 1999.

[111] H. Harney, "Group Key Management Protocol (GKMP) Architecture," *Network Working Group*, 1997.

[112] Y.-H. Kung and H.-C. Hsiao, "GroupIt: Lightweight Group Key Management for Dynamic IoT Environments," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 5155–5165, 2018.

[113] S. Zhu, S. Setia, and S. Jajodia, "LEAP+: Efficient security mechanisms for Large-scale distributed sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 2, no. 4, pp. 500–528, 2006.

[114] R. Roman, C. Alcaraz, J. Lopez, and N. Sklavos, "Key Management Systems for Sensor Networks in the Context of the Internet of Things," *Computers & Electrical Engineering*, vol. 37, no. 2, pp. 147–159, 2011.

[115] W. Abdallah, N. Boudriga, D. Kim, and S. An, "An Efficient and Scalable Key Management Mechanism for Wireless Sensor Networks," in *ICACT*, pp. 480–493, IEEE, 2015.

[116] L. Parrilla, E. Castillo, J. López-Ramos, J. Álvarez-Bermejo, A. García, and D. Morales, "Unified Compact ECC-AES Co-Processor with Group-Key Support for IoT Devices in Wireless Sensor Networks," *Sensors*, vol. 18, no. 1, p. 251, 2018.

[117] M. A. Kandi, H. Lakhlef, A. Bouabdallah, and Y. Challal, "An Efficient Multi-Group Key Management Protocol for Internet of Things," in *2018 26th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pp. 1–6, IEEE, 2018.

[118] T. Gebremichael, U. Jennehag, and M. Gidlund, "Lightweight IoT Group Key Establishment Scheme Using One-way Accumulator," in *2018 International Symposium on Networks, Computers and Communications (ISNCC)*, pp. 1–7, IEEE, 2018.

[119] N. Ferrari, T. Gebremichael, U. Jennehag, and M. Gidlund, "Lightweight Group-Key Establishment Protocol for IoT Devices: Implementation and Performance Analyses," in *2018 Fifth International Conference on Internet of Things: Systems, Management and Security*, pp. 31–37, IEEE, 2018.

[120] A. Lei, Y. Cao, S. Bao, P. Asuquom, H. Cruickshank, and Z. Sun, "Blockchain-based Dynamic Key Management for IoT-Transportation Security Protection," *Blockchain for Distributed Systems Security*, p. 117, 2019.

[121] C. Herder, M.-D. Yu, F. Koushanfar, and S. Devadas, "Physical Unclonable Functions and Applications: A Tutorial," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1126–1141, 2014.

[122] H. Gu and M. Potkonjak, "Securing Interconnected PUF Network with Reconfigurability," in *International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 231–234, IEEE, 2018.

[123] D. Mukhopadhyay, "PUFs as Promising Tools for Security in Internet of Things," *IEEE Design & Test*, vol. 33, no. 3, pp. 103–115, 2016.

[124] M. T. Rahman, F. Rahman, D. Forte, and M. Tehranipoor, "An Aging-Resistant RO-PUF for Reliable Key Generation," *IEEE Transactions on Emerging Topics in Computing*, vol. 4, no. 3, pp. 335–348, 2016.

[125] M. Huang, B. Yu, and S. Li, "PUF-assisted Group Key Distribution Scheme for Software-Defined Wireless Sensor Networks," *IEEE Communications Letters*, 2017.

[126] J. Miao, M. Li, S. Roy, and B. Yu, "LRR-DPUF: Learning Resilient and Reliable Digital Physical Unclonable Function," in *ICCAD*, pp. 1–8, IEEE, 2016.

[127] C. Clos, "A Study of Non-Blocking Switching Networks," *Bell Labs Technical Journal*, vol. 32, no. 2, pp. 406–424, 1953.

[128] D. H. Lawrie, "Access and Alignment of Data in an Array Processor," *IEEE Transactions on Computers*, vol. 100, no. 12, pp. 1145–1155, 1975.

[129] C. Herder, L. Ren, M. van Dijk, M.-D. Yu, and S. Devadas, "Trapdoor Computational Fuzzy Extractors and Stateless Cryptographically-secure Physical Unclonable Functions," *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 1, pp. 65–82, 2017.

[130] J. Delvaux, "Machine-Learning Attacks on PolyPUFs, OB-PUFs, RPUFs, LHS-PUFs, and PUF-FSMs," *IEEE Transactions on Information Forensics and Security*, 2019.

[131] Y. Fang, C. Wang, Q. Ma, C. Gu, M. ONeill, and W. Liu, "Attacking Arbiter PUFs Using Various Modeling Attack Algorithms: A Comparative Study," in *2018 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, pp. 394–397, IEEE, 2018.

[132] J. Miskelly, C. Gu, Q. Ma, Y. Cui, W. Liu, and M. ONeill, "Modelling Attack Analysis of Configurable Ring Oscillator (CRO) PUF Designs," in *2018 IEEE 23rd International Conference on Digital Signal Processing (DSP)*, pp. 1–5, IEEE, 2018.

[133] W. Ge, J. Q. Huang, B. Liu, M. Zhu, and Y. Cao, "A Deep Learning Modeling Attack Method for MISR-APUF Protection Structures," in *2018 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, pp. 398–402, IEEE, 2018.

[134] A. O. Aseeri, Y. Zhuang, and M. S. Alkatheiri, "A Subspace Pre-learning Approach to Fast High-Accuracy Machine Learning of Large XOR PUFs with Component-Differential Challenges," in *2018 IEEE International Conference on Big Data (Big Data)*, pp. 1563–1568, IEEE, 2018.

[135] H. L. França, C. B. Prado, V. C. Patil, and S. Kundu, "Defeating Strong PUF Modeling Attack via Adverse Selection of Challenge-Response Pairs," in *2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pp. 25–30, IEEE, 2018.

[136] P. Chodowiec and K. Gaj, "Very Compact FPGA Implementation of the AES Algorithm," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 319–333, Springer, 2003.

[137] Z. U. Khan and M. Benaissa, "High-Speed and Low-Latency ECC Processor Implementation Over GF 2m on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 1, pp. 165–176, 2017.

[138] D. Singelée, S. Seys, L. Batina, and I. Verbauwhede, "The Energy Budget for Wireless Security: Extended Version," *IACR Cryptology ePrint Archive*, vol. 2015, p. 1029, 2015.

[139] I. Corporation, "Intel Wireless Wifi Link 5300 Module Quick Specs Datasheet," *http://h18000.www1. hp.com/products/quickspecs/13085˙na/13085˙na.PDF*, 2008.

[140] "Texas Instruments CC2520 2.4 GHz IEEE 802.15.4/Zigbee RF Transceiver Datasheet." `http://www.ti.com/lit/ds/symlink/cc2520.pdf`, 2007.

[141] "Texas Instruments Bluetooth BRF6100 and BRF6150 Product Bulletin." http://focus.ti.com/pdfs/wtbu/TI_brf6100_6150.pdf, 2004.

[142] "Texas Instruments CC2540 2.4-GHz Bluetooth Low Energy System-on-Chip Datasheet." http://www.ti.com/lit/ds/symlink/cc2540.pdf, 2011.

[143] I. Kuon and J. Rose, "Measuring the Gap between FPGAs and ASICs," *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 26, no. 2, pp. 203–215, 2007.

[144] D. J. Bernstein, "Curve25519: New Diffie-Hellman Speed Records," in *International Workshop on Public Key Cryptography*, pp. 207–228, Springer, 2006.

[145] J. Decuir, "Bluetooth 4.0: Low Energy," *Cambridge, UK: Cambridge Silicon Radio SR plc*, vol. 16, 2010.