



AFRL-RY-WP-TR-2020-0372

LOGIC ENCRYPTION FOR RESOURCE CONSTRAINED DESIGNS

**David Luria
University of Cincinnati**

**DECEMBER 2020
Final Report**

Approved for public release; distribution is unlimited.

See additional restrictions described on inside pages

© David Luria

STINFO COPY

**AIR FORCE RESEARCH LABORATORY
SENSORS DIRECTORATE
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7320
AIR FORCE MATERIEL COMMAND
UNITED STATES AIR FORCE**

REPORT DOCUMENTATION PAGE				<i>Form Approved</i> <i>OMB No. 0704-0188</i>	
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YY) December 2020		2. REPORT TYPE Thesis		3. DATES COVERED (From - To) 9 November 2020 – 9 November 2020	
4. TITLE AND SUBTITLE LOGIC ENCRYPTION FOR RESOURCE CONSTRAINED DESIGNS				5a. CONTRACT NUMBER W911QY-17-C-0114	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER N/A	
6. AUTHOR(S) David Luria				5d. PROJECT NUMBER N/A	
				5e. TASK NUMBER N/A	
				5f. WORK UNIT NUMBER N/A	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Cincinnati 2600 Clifton Ave. Cincinnati, OH 45221				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory Sensors Directorate Wright-Patterson Air Force Base, OH 45433-7320 Air Force Materiel Command United States Air Force				10. SPONSORING/MONITORING AGENCY ACRONYM(S) AFRL/RDYT	
				11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S) AFRL-RY-WP-TR-2020-0372	
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.					
13. SUPPLEMENTARY NOTES PAO case number 88ABW-2020-3465, Clearance Date 9 November 2020. © David Luria. Submitted to the Graduate School of the University of Cincinnati in partial fulfillment of the requirements for the degree of Master of Science in the Dept. of Electrical Engineering and Computing Systems of the College of Engineering and Applied Sciences. This work was funded in whole or in part by Department of the Air Force. The U.S. Government has for itself and others acting on its behalf an unlimited, paid-up, nonexclusive, irrevocable worldwide license to use, modify, reproduce, release, perform, display, or disclose the work by or on behalf of the U.S. Government. Report contains color.					
14. ABSTRACT In the face of the increasing cost of manufacturing semiconductor devices, many designers of Integrated Circuits (IC) have been electing to outsource the fabrication of ICs to pure play foundries in order to reduce manufacturing costs. However, this may lead to IC designs being stolen, counterfeited, maliciously modified, or otherwise mishandled by bad actors in the supply chain. In response, researchers have devised methods of thwarting such attacks against intellectual property in the IC supply chain. One such method is known as logic encryption. Logic encryption is a hardware security strategy that adds key inputs, which require the correct secret key input, known only to the designer, to be applied for the circuit to have correct output behavior. The field of logic encryption has evolved over the past decade during which increasingly strong attacks and increasingly strong encryption strategies have been developed. However, the cost in terms of power, performance, and area (PPA) of implementing logic encryption has often been ignored in favor of increasing the level of security for proposed methods, and how well they protect against known attacks. Considering these costs is an important hurdle in transitioning the technology to commercial-grade designs, and a strategy for constraining the cost of logic encryption is needed.					
15. SUBJECT TERMS constrained design, logic obfuscation, power, performance, area					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT: SAR	18. NUMBER OF PAGES 118	19a. NAME OF RESPONSIBLE PERSON (Monitor) Justin Pelan 19b. TELEPHONE NUMBER (Include Area Code) N/A
a. REPORT Unclassified	b. ABSTRACT Unclassified	c. THIS PAGE Unclassified			

Logic Encryption for Resource Constrained Designs

A thesis submitted to the
Graduate School
of the University of Cincinnati
in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in the Dept. of Electrical Engineering and Computing Systems
of the College of Engineering and Applied Sciences
3 September 2020

By

David Luria

Bachelor of Science in Astrophysics

Bachelor of Science in Physics

Minor in Mathematics

University of Cincinnati, Cincinnati, Ohio, United States of America

April 2018

Thesis Advisor and Committee Chair: Ranga Vemuri, Ph.D.

ABSTRACT

In the face of the increasing cost of manufacturing semiconductor devices, many designers of Integrated Circuits (IC) have been electing to outsource the fabrication of ICs to pure play foundries in order to reduce manufacturing costs. However, this may lead to IC designs being stolen, counterfeited, maliciously modified, or otherwise mishandled by bad actors in the supply chain. In response, researchers have devised methods of thwarting such attacks against intellectual property in the IC supply chain. One such method is known as logic encryption.

Logic encryption is a hardware security strategy that adds key inputs, which require the correct secret key input, known only to the designer, to be applied for the circuit to have correct output behavior. The field of logic encryption has evolved over the past decade during which increasingly strong attacks and increasingly strong encryption strategies have been developed. However, the cost in terms of power, performance, and area (PPA) of implementing logic encryption has often been ignored in favor of increasing the level of security for proposed methods, and how well they protect against known attacks. Considering these costs is an important hurdle in transitioning the technology to commercial-grade designs, and a strategy for constraining the cost of logic encryption is needed.

A methodology for applying known logic encryption methods to a design is proposed in this work. In this methodology, named Constraint-Directed Logic Encryption (CDLE), potential encrypted versions of an IC design are considered. A designer can set the security requirement for encryption as well as power, performance, and area constraints, based on the design specification. There is a subset of designs that meet the security requirement and another that are within the PPA constraints. CDLE targets the overlap of these subsets, and maximizes security within it to produce an optimally encrypted design that meets the security requirement while staying within PPA constraints.

In this work, two specific CDLE methods are proposed. The first is a concurrent search method, which explores encryption schemes and key sizes to arrive at an optimally-encrypted design. More than one computation thread can be leveraged to sample the design space concurrently using this method. Two different methods of estimating the PPA of sampled designs in this process are proposed, one using estimations based on an IC layout, and one based on the netlist. The former proved to be heavier in computation time, but more accurate in estimating PPA and provided higher security in the final design. The second CDLE method proposed is a machine learning approach. Several multivariable regression-based learning models were created, and predictions for optimally-encrypted designs were made with them and compared to determine the model with the best quality of results. The best model of this method was promising, so there is potential for machine learning in constraint-directed logic encryption.

Acknowledgements

The completion of this thesis could not have been possible without the contributions and support of faculty members, funding partners, colleagues, family, and friends. I would like to name those people that especially helped me in the completion of my degree. For those not named who helped me with my studies, know that your support is very much appreciated and recognized as well.

First of all, I would like to thank my research advisor and faculty mentor, Dr. Ranga Vemuri. He was willing to take me under his wing when I was new to the field and provide me with endless opportunities to grow greatly as a researcher, student, and professional throughout my Master's studies and beyond. He taught me to be a self-motivated researcher, guiding me to ask critical questions, learn more, and figure out solutions and new approaches to difficult research problems. I also learned to work effectively in a research team, to learn from those that are more knowledgeable, and to be a leader in areas where I excel. His help with networking during my job searched helped tremendously, and landed me a job with my dream company. Without his patience, guidance, and support I would not be where I am today. I am endlessly thankful that I had him as an advisor.

I extend a special thank you to Edaptive Computing, Inc. and the Air Force Research Laboratory for funding my research assistantship. This funding was key to the completion of my education, and allowed me the peace of mind to more effectively complete my studies and research with relieved financial pressure.

I would like to thank Rob Montjoy and Adam Kunkel for their technical support throughout the completion of my degree. I relied on their knowledge and help with system administration and software installation countless times throughout my studies. I also thank Julie Muenchen for her help in handling the administrative processes related to my degree and studies.

I want to acknowledge the CODES team, made up of Ysaswy Kasarabada, Juneeth Kumar Meka, Haimanti Chakraborty, Nikhil Saxena, Vaishali Muralidharan, and Shrinidhi Venkatesh, for their work in implementing various software modules that apply encryption methods, and made CDLE possible. Their assistance in developing software, project planning, and collecting data contributed directly to the methods and results shown in this work. Without their support and hard work, this thesis would not have been possible. More specifically, the CDLE experiments performed in this work were a joint effort between Shrinidhi and myself. Shrinidhi performed CDLE experiments using Design Compiler estimation, including those for the SS PCM, stepper motor, tv80, functional USB, AC97 control, SASC, b19, SystemC DES, and DES3 performance designs, as well as producing a layout for DES3 performance. I want to thank her for her help in performing these experiments and sharing her results so they could be included in this work.

My work in this thesis and beyond would not have been possible without support from my friends at

the UC Digital Design Environments Laboratory, so I would like to take a moment to thank them. First, thank you to Yasaswy Kasarabada. From the moment I first stepped foot in DDEL, he was there to mentor me in countless ways, including learning about hardware security techniques, commercial tools, research in general, software development, and classwork. Collaborating with him on several projects throughout my studies was such a great experience, and directly contributed to the results in this work. I'm lucky to have worked with such a great mind. I also want to thank the now DDEL alumni that helped me get started in the lab and taught me to use Synopsys software like Design Compiler and IC Compiler, Xiaobang Liu and Sudheer Ram Thulasi Raman. Also, I extend my gratitude to Sanjana Sekar and Richa Agrawal, who helped me greatly to reach out to my network and opened doors for me professionally that helped greatly during my job search.

My admittance to the EECS graduate school was not possible without support as well. During my time as a physics undergrad, I could not have made it to the steps of the graduate school without the help of Dr. Richard Gass and Dr. Michael Sitko. Their guidance helped me greatly throughout my physics degree, and their recommendations for my admittance to graduate school made my application stand out among many. Dr. Gass' recommendation also helped me greatly throughout my job search. I am forever thankful of their help and the time I spent in the UC Physics Department.

Finally, I thank my family and friends for being there with me throughout my studies and being the all important emotional support. Graduate school is not an easy journey, and I could not have made it without being able to cool off with friends, and spend time with my loved ones. I especially want to thank my parents, Joseph and Mary Luria. I can't thank them enough for being wonderful parents throughout my life, providing me with endless love, support, and life lessons, and for helping me with my studies when they could. I also want to thank my partner, Andrew Bohlen, for being my rock and being there for me through the hardest parts of my studies. Thank you to all my loved ones for supporting me during a very difficult and rewarding journey.

Contents

Abstract	i
Acknowledgements	ii
List of Tables	viii
List of Figures	xii
Chapter 1 Introduction	1
1.1 IC Supply Chain	1
1.2 Threats to Intellectual Property	2
1.2.1 Threat Model	2
1.2.2 IP Piracy and IC Overproduction	2
1.2.3 Reverse Engineering	2
1.2.4 Hardware Trojan Insertion	3
1.2.5 Counterfeiting	3
1.3 Logic Encryption Overview	3
1.3.1 Protection Against IP Threats	5
1.3.2 Metrics	5
1.4 Thesis Statement	6
1.5 Thesis Overview	7
Chapter 2 Logic Encryption Methods, Attacks, and Countermeasures	9
2.1 Brief History of Logic Encryption	9
2.2 SAT-Vulnerable Encryption Methods	11
2.3 Decryption Attacks	13
2.4 SAT-Resilient Encryption Methods	18
2.5 Sequential Locking Methods	23

2.6	Advanced Attacks	24
2.7	Summary	27
Chapter 3 Constraint-Directed Logic Encryption		28
3.1	Motivation	28
3.2	Methodology Overview	29
3.3	CDLE using Binary Search	30
3.4	CDLE using Concurrent Tree Search	32
3.4.1	PPA Estimation Strategy	33
3.4.2	Security Evaluation of Encryption Schemes	36
3.4.3	Design-Space Exploration Algorithm	39
3.5	Circuit Designs for Experimentation	42
3.6	Concurrent Tree Search Experimentation	42
3.6.1	Experimental Setup	42
3.6.2	Results	44
3.6.3	Case Study: Area Optimized 3-DES	49
3.7	Concurrent Tree Search Using DC	50
3.7.1	Motivation	50
3.7.2	Results	52
3.7.3	Case Study: Area Optimized 3-DES	60
3.7.4	Comparison with CDLE using ICC	62
3.8	Summary	63
Chapter 4 Machine Learning Approach		65
4.1	Motivation	65
4.2	Method Overview	66
4.3	Training the Model	68
4.3.1	Training Set	68
4.3.2	Results	70
4.4	Experimentation Results	73
4.4.1	Model Adjustments	77
4.5	Summary	82
Chapter 5 Conclusions and Future Work		83

5.1	Discussion of CDLE Methods	83
5.1.1	Concurrent Tree Search	84
5.1.2	Machine Learning Approach	85
5.2	Future Work	86
5.2.1	Improved cost metrics estimation	86
5.2.2	Improved security metrics estimation	87
5.2.3	Machine learning direction	87
5.2.4	Other design-space search methods	87
References		88
Appendix: Additional Constraint-Directed Logic Encryption Case Studies		91
A.1	Concurrent Tree Search with ICC	92
A.1.1	Simple SPI	92
A.1.2	SystemC DES	94
A.1.3	Area Optimized DES	94
A.1.4	SASC	96
A.2	Concurrent Tree Search with DC	98
A.2.1	CF RCA	98
A.2.2	Performance Optimized 3-DES	99
A.2.3	SystemC DES	101

List of Tables

2.1	Logic encryption methods surveyed	10
2.2	Attacks against logic encryption surveyed	11
3.1	Resiliency table for combinational encryption methods. $\mathbf{X}$ denotes resilience, $\approx$ denotes resilience based on sub-method	36
3.2	Resiliency Table for sequential encryption methods. $\mathbf{X}$ denotes resilience	37
3.3	Opencores[14] and ITC'99 [2] circuits chosen for CDLE experimentation	43
3.4	Pre-encryption PPA estimations for each design tested	44
3.5	Obfuscation strategies chosen by CDLE concurrent tree search for various PPA constraints. A '-' denotes failure of the algorithm to find an optimally encrypted design	45
3.6	Pre-encryption PPA estimations using the modified estimation strategy, for each design tested	52
3.7	Obfuscation strategies chosen by CDLE concurrent tree search without ICC for various PPA constraints. A '-' denotes failure of the algorithm to find an optimally encrypted design . . .	53
3.8	Comparison of CDLE results using ICC and DC PPA estimation with 25% PPA constraints .	63
3.9	Comparison of CDLE results using ICC and DC PPA estimation with 10% PPA constraints .	64
3.10	Comparison of CDLE results using ICC and DC PPA estimation with 5% PPA constraints .	64
3.11	Comparison of CDLE results using ICC and DC PPA estimation with 1% PPA constraints .	64
4.1	Designs from the Altera Synthesis Cookbook [3] selected to train the CDLE machine learning model	69
4.2	Encryption scheme and key size predicted by the regression with the highest quality of results using ICC training estimations, interaction regression with a Cauchy cost function.	76
4.3	Encryption scheme and key size predicted for each test design and PPA constraint set by the regression with the highest quality of results using DC training estimations, interaction regression with a Cauchy cost function.	77

4.4	Encryption scheme and key size predicted for each test design and PPA constraint set by the regression with the highest quality of results using ICC training estimations, interaction regression with a Cauchy cost function, after adjustments were made.	79
4.5	Encryption scheme and key size predicted for each test design and PPA constraint set by the regression with the highest quality of results using DC training estimations, interaction regression with a Cauchy cost function, after adjustments were made.	80

List of Figures

1.1	Basic IC supply chain flow and the risk of IP theft at the foundry	2
1.2	Basic block diagram of a logic encryption implementation	4
1.3	Example of logic encryption of a netlist by inserting XOR, XNOR, and OR gates with correct key K=01000.	4
2.1	Example of a circuit encrypted using SLL [38] strategy.	12
2.2	Key dependency [10] cone analysis. G_{strong} represents the input cones in which key gates are placed.	12
2.3	Key Sensitization Attack [17] Strategies	14
2.4	The SAT [28] miter circuit	15
2.5	Example circuit to be unrolled	17
2.6	Sequential SAT [11] unrolled example circuit with unroll count = 2	18
2.7	SARLock [36] architecture	18
2.8	Anti-SAT [34] locking block architecture	19
2.9	TTLock [40] architecture	20
2.10	Cyclic modification [24] by added key-controlled multiplexer	21
2.11	D-type flip flops encrypted with Encrypt-Flip Flop [9]	23
2.12	Flip flop chains, with a sneak path through FF5	24
3.1	Parallel PPA collection strategy	35
3.2	PPA cost of each design for successful CDLE encryption (1/2)	46
3.3	PPA cost of each design for successful CDLE encryption (2/2)	47
3.4	Execution time in seconds of each successful CDLE execution vs pre-encryption gate count	48
3.5	Execution time in seconds vs thread count of CDLE encryption of Hilbert Transformer	48
3.6	Execution time in seconds vs thread count of CDLE encryption of Simple SPI	48
3.7	CDLE Trajectory and result of des3_area with PPA constrained to 5% increases	49

3.8	Layout of encrypted des3_area. On right, the key inputs and their respective key gates are highlighted, yellow corresponding to ChainLock, orange to SRClock, and red to randomly placed gates.	50
3.9	Modified PPA collection strategy without IC Compiler II	51
3.10	PPA cost of each design for successful CDLE encryption using only the DC flow (1/3)	55
3.11	PPA cost of each design for successful CDLE encryption using only the DC flow (2/3)	56
3.12	PPA cost of each design for successful CDLE encryption using only the DC flow (3/3)	57
3.13	Overall execution time in seconds as a function of pre-encryption gate count for each CDLE design experiment	58
3.14	Execution time in seconds vs thread count of DC-Only CDLE encryption of ss_pcm	58
3.15	Execution time in seconds vs thread count of DC-Only CDLE encryption of Simple SPI . . .	59
3.16	Execution time in seconds vs thread count of DC-Only CDLE encryption of Stepper Motor .	59
3.17	Execution time in seconds vs thread count of DC-Only CDLE encryption of Hilbert Transformer	60
3.18	Execution time in seconds vs thread count of DC-Only CDLE encryption of USB Physical . .	60
3.19	CDLE Trajectory and result of des3_area with PPA constrained to 5% increases	61
3.20	Layout of encrypted des3_area after CDLE with DC only. On right, the key inputs and their respective key gates are highlighted, yellow corresponding to ChainLock, orange to SRClock, and red to randomly placed gates.	62
4.1	Selected examples of regression for key size vs PPA cost	70
4.2	Selected examples of regression for gate count vs PPA cost	71
4.3	Average mean relative error among encryption schemes for each type of candidate regression model, using ICC estimations on left, and DC on left. In each graph, the bars correspond to, starting from left, linear soft_l1, linear cauchy, linear arctan, interaction soft_l1, interaction cauchy, interaction arctan, linear + interaction soft_l1, linear + interaction cauchy, and linear + interaction arctan.	73
4.4	Percent of CDLE experiments that matched chosen encryption scheme when compared with the corresponding DC or ICC concurrent tree search results for each machine learning model tested. Cases in which no encryption scheme could be found in either case were counted as a match. Each cost function produced the same results, so these are not shown.	74

4.5	Percent of CDLE experiments that matched chosen encryption scheme when compared with the corresponding DC or ICC concurrent tree search results for each machine learning model tested. Cases in which no encryption scheme could be found in either case were not considered. Each cost function produced the same results, so these are not shown.	75
4.6	Percent of CDLE experiments in which no encryption strategy could be chosen for DC or ICC concurrent tree search results, but one was found for the machine learning model in question. Each cost function produced the same results, so these are not shown.	75
4.7	Percent of CDLE experiments that matched chosen encryption scheme when compared with the corresponding DC or ICC concurrent tree search results for the adjusted and original versions of the highest quality machine learning model. Cases in which no encryption scheme could be found in either case were counted as a match.	80
4.8	Percent of CDLE experiments that matched chosen encryption scheme when compared with the corresponding DC or ICC concurrent tree search results for the adjusted and original versions of the highest quality machine learning model. Cases in which no encryption scheme could be found in either case were not considered.	81
4.9	Percent of CDLE experiments in which no encryption strategy could be chosen for DC or ICC concurrent tree search results, but one was found for the machine learning model in question.	81
A.1	CDLE Trajectory and result of simple_spi with PPA constrained to 50% increases	93
A.2	Layout of encrypted Simple SPI after CDLE. On right, the key inputs and their respective key gates are highlighted, yellow corresponding to ChainLock, orange to SARLock, and red to randomly placed gates.	93
A.3	CDLE Trajectory and result of systemcdes with PPA constrained to 25% increases	94
A.4	Layout of encrypted systemcdes after CDLE. On right, the key inputs and their respective key gates are highlighted, yellow corresponding to ChainLock, orange to SFLL, and red to randomly placed gates.	95
A.5	CDLE Trajectory and result of des_area with PPA constrained to 5% increases	95
A.6	Layout of encrypted des_area after CDLE. On right, the key inputs and their respective key gates are highlighted, yellow corresponding to ChainLock, orange to Cyclic, and red to randomly placed gates.	96
A.7	CDLE Trajectory and result of SASC with PPA constrained to 1% increases	97

A.8	Layout of encrypted SASC after CDLE. On right, the key inputs and their respective key gates are highlighted, yellow corresponding to ChainLock, orange to SRClock, and red to randomly placed gates.	97
A.9	CDLE Trajectory and result of cfrca with PPA constrained to 1% increases	98
A.10	Layout of encrypted cfrca after DC-only CDLE. On right, the key inputs and their respective key gates are highlighted, yellow corresponding to ChainLock, orange to Cyclic, and red to randomly placed gates.	99
A.11	CDLE Trajectory and result of des3_perf with PPA constrained to 0.1% increases	100
A.12	Layout of encrypted des3_perf after DC-only CDLE. On right, the key inputs and their respective key gates are highlighted, yellow corresponding to ChainLock, orange to AntiSAT, and red to randomly placed gates.	101
A.13	CDLE Trajectory and result of systemcdes with PPA constrained to 25% increases	102
A.14	Layout of encrypted systemcdes after DC-only CDLE. On right, the key inputs and their respective key gates are highlighted, yellow corresponding to ChainLock, orange to SFULL, and red to randomly placed gates.	102

Chapter 1

Introduction

1.1 IC Supply Chain

Integrated circuits (ICs) make up the backbone of modern semiconductor computing devices and have a wide variety of applications, including consumer devices, military electronics, microprocessors, CPUs, and internet-of-things (IoT) devices. Fabricating ICs has become increasingly complex and expensive due to many factors, including shrinking technology node size and increasingly demanding power usage restrictions. Owning and maintaining the equipment necessary to produce powerful and competitive semiconductor devices has become prohibitively expensive. Therefore, many electronics and computer device companies have elected to outsource the manufacturing of their IC designs by contracting specialized 3rd party semiconductor foundries, known as pure play foundries. This has allowed these electronics companies to avoid costs associated with owning and maintaining production equipment. [5, 21]

Lowering IC production cost by contracting foundries comes at the cost of trust. The globalization of the IC supply chain has resulted in reliance on 3rd party semiconductor foundries that has raised a concern for untrusted foundries. This is a continually growing concern, especially considering some popular foundries are located in countries with weak IP protection policies and enforcement [21]. Bad actors in the IC supply chain, specifically located in outsourced foundries, can steal and misuse hardware IP from electronics companies. The combined financial impact of IP theft on the global semiconductor industry is estimated to be over \$100 billion annually [15, 18].

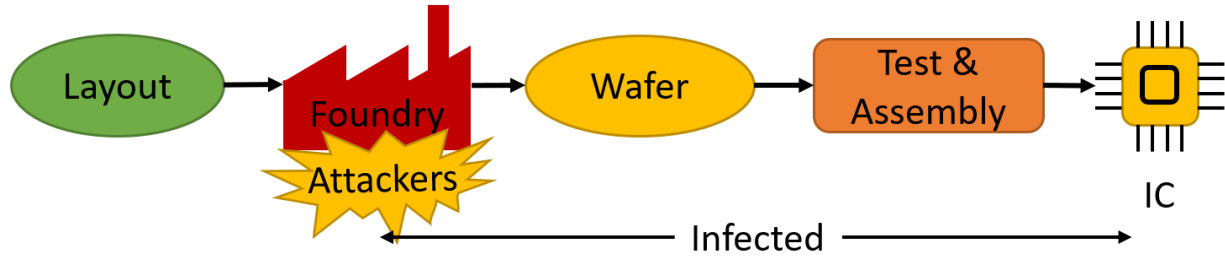


Figure 1.1: Basic IC supply chain flow and the risk of IP theft at the foundry

1.2 Threats to Intellectual Property

Bad actors at untrusted foundries may steal and misuse IP in several different ways. These include, but are not limited to, IP piracy/IC overproduction [20], reverse engineering [32], hardware Trojan insertion [20], and counterfeiting [8]. These forms of IP theft cost the semiconductor industry billions of dollars by flooding the market with fake, counterfeit, and invalid ICs.

1.2.1 Threat Model

An adversarial threat model [41] should be considered for threats to IP in the IC supply chain. The circuit designer is the IP rights holder and can be trusted. The attacker is an untrusted agent at an outsourced foundry. Other parties that cannot be trusted include the test facility and the end-user. The goal of an attacker is to overproduce an IC design or uncover the function of the design and reproduce it. An attacker will attempt to bypass any security measures in place to prevent them from doing so. The adversary can use a stolen design to sell ICs and undercut the designer's ICs on the market.

1.2.2 IP Piracy and IC Overproduction

IP piracy is the theft of an IC design to produce and sell illegal copies of it. Illegal copies come in the form of overproduction, when more ICs are produced than requested in the fabrication order. The illegal copies can then be sold by the bad actor for a fraction of the price of the legal copies sold by the original vendor. [20]

1.2.3 Reverse Engineering

Reverse engineering is the process of uncovering the function of an IC by observing its structural properties. This can be done by recovering the netlist from the layout. If standard cells and nets can be identified from the GDSII layout view of the design, then the netlist can be reconstructed by structural analysis of

the layout. Reverse engineering can also be performed physically, through a process called delayering. This involves removing the ceramic cap of an IC to reveal metal and semiconductor layers contained therein. Every layer of the IC is then revealed by removing them one-by-one. Microscopic images of each layer can be stitched together to construct a layout view of each layer, and the netlist can be recovered by structural analysis of metal connections and semiconductor standard cells. [32]

1.2.4 Hardware Trojan Insertion

Hardware Trojans are extra components maliciously added to IC designs to disrupt/disable functionality or to secretly leak valuable information to an attacker. A Trojan is small and deeply embedded into the design to evade detection during post-silicon validation. Trojans consist of a trigger and a payload. The trigger determines the manner and time in which the payload is activated, for example a specific primary input pattern, or a rarely-accessed machine state. The payload is the manner in which the functionality of the design is affected, either by denial of service or information leaking by hijacking communication hardware. Trojans are particularly dangerous, due to the difficulty of detection and the potential damage of delayed denial of service. [20]

1.2.5 Counterfeiting

Counterfeiting is the act of producing fake designs of an IC based on a stolen IP. Unlike overproduction, a counterfeit is an imitation of the original design. A bad actor may forge a component or design illegally and attempt to reproduce it. This degrades the trust of the original design and allows an attacker to cheaply produce similar designs or components. Counterfeited designs can also be dangerous due to small functional differences from the original design. This can disrupt devices in which they are implemented due to unexpected behavior from the counterfeited component. [8]

1.3 Logic Encryption Overview

To combat the growing problem of IP theft in the IC supply chain, several strategies have been proposed. One such strategy, proposed by Roy, Koushanfar, and Markov in 2010 [21] known as logic encryption (or logic obfuscation, or logic locking) has been a growing area of research in hardware security in the past decade. In their method, Roy et al. modify the IC design process to add a locking step, in which a master key is devised by the IP rights holder. Unless the master key is known and applied, the output behavior of the IC is corrupted, effectively disabling it. When the chip is fabricated and packaged, the correct function of

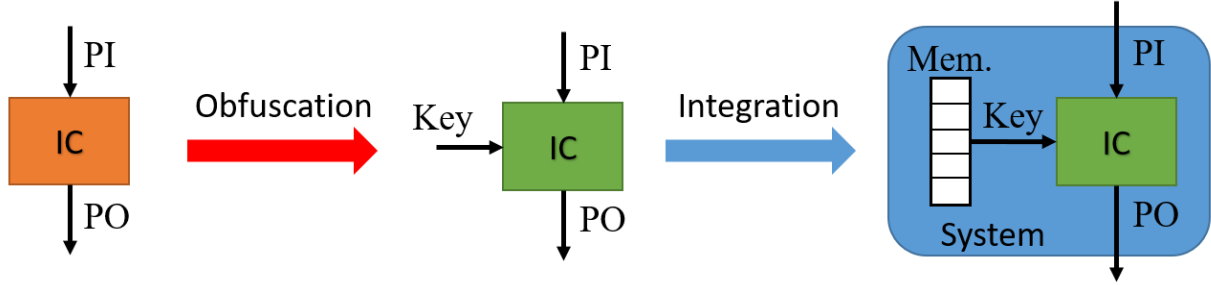


Figure 1.2: Basic block diagram of a logic encryption implementation

the IC cannot be recovered without knowledge of the correct key. Ideally, the key is not feasibly deciphered, so only the IP rights holder can activate the IC, after receiving the packaged chips.

Logic locking is one method of locking a design with a master key. This involves modifying the netlist of a design through the addition of extra logic gates, known as key gates, which each contain one or more added primary input pins, known as key inputs, in their transitive fan-in. When the (ideally single) correct pattern is applied to the key inputs, the output behavior of the IC is equivalent to the original design specification. Under the application of an incorrect key, the added key gates corrupt the output behavior of the circuit for one or more primary (non-key) input patterns. Thus, uncovering the correct functionality of the IC by brute force has a complexity of at least $O(2^k)$ for k added key inputs. The correct key is applied to the key input pins by the IP rights holder to unlock it after manufacturing. The key can be loaded into non-volatile memory, which the IC reads from for activation. This way, the designer can sell the working design directly to the end user.

After Roy, et al. proposed logic encryption, many attack methods were proposed to decrease the attack complexity against it and efficiently recover the correct key. These attacks can be divided into two regimes: Oracle-guided and Oracle-less. An oracle is a black box, working copy of the IC with the correct key pattern applied. This can be used to learn the correct input-output behavior without a netlist. An oracle can be

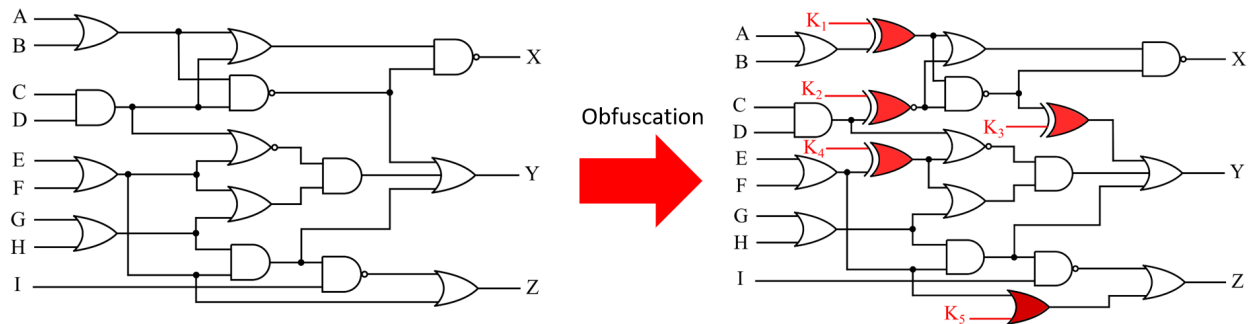


Figure 1.3: Example of logic encryption of a netlist by inserting XOR, XNOR, and OR gates with correct key $K=01000$.

obtained by purchasing a working IC from the market, or by black box simulation of the circuit design. In the original logic encryption proposal, no key gate placement strategy was given, other than to avoid critical paths in the design. In response to proposed attacks, stronger logic encryption strategies have been developed to counteract increasingly complex attacks. [41]

1.3.1 Protection Against IP Threats

Logic encryption is designed to protect design IP from being stolen and misused in the ways introduced in Section 1.2 of this chapter. The strategy's effectiveness relies on the secret key not being known to the attacker. Assuming this, logic encryption prevents the following threats:

- **IP Piracy and IC Overproduction** – Overproduced ICs are useless unless the secret key is known. Therefore, attackers cannot illegally sell overproduced ICs on the market without activating them using the key.
- **Reverse Engineering** – Even if a layout is recovered by reverse engineering, key gates obfuscate the true function of the design. This can be further ensured by layout-level obfuscation techniques.
- **Hardware Trojan Insertion** – Hardware Trojan insertion relies on finding locations in the design which will make it difficult to detect. This requires specific knowledge of the behavior of the circuit, which is obfuscated by logic encryption.
- **Counterfeiting** – Any copies of the design are useless without knowledge of the correct key. In addition, the design cannot be copied unless its true function is known, which is impossible for an encrypted design.

1.3.2 Metrics

Two types of metrics can be considered when applying logic encryption to protect a design [5]. One set of metrics involve the strength of security provided by the applied encryption. Examples of these metrics are:

- **Output Corruption** – The degree to which the IC output is affected by incorrect keys. This is typically quantified by the Hamming Distance (HD) between the correct output behavior and the output behavior under each correct key. A basic ideal value is 50%, large enough such that application of an incorrect key is noticeable in post-silicon validation testing, but small enough to ensure difficulty in cracking the key.

- **Key Size** – The number of key inputs pins that are added to the design. In general, the larger the key size, the more difficult the encryption is to crack.
- **Attack Resilience** – The known attack strategies against which the encryption strategy is resilient. This depends on the particular key gate placement strategy used.

The second set of metrics involve the cost of applying logic encryption. These are PPA (power, performance, and area) metrics. The addition of key gates have an impact on these metrics for any design:

- **Power Usage** – Additional logic gates consume additional power when operating the IC. This comes in two forms, static power from leakage current, and dynamic power from gate output switching activity.
- **Performance Degradation** – The speed at which the design can be operated. Despite attempts to lessen the critical timing of a design, adding logic encryption can affect the critical path or introduce a new critical path. If this results in negative timing slack, then the IC must be operated at a lower clock speed with the added encryption.
- **Area Footprint** – The additional area on die required for the IC with added encryption. Adding key gates increases the logic gate count, requiring additional die area for the added standard cells corresponding to the key gates.

Novel logic encryption methods of the past decade often respond directly to security metrics, especially attack resilience. Known attacks present a challenge to the effectiveness of logic encryption and responding to these attacks is an important design consideration. Some encryption methods consider cost metrics, especially area. However, cost metrics are often ignored in favor of improving security metrics.

1.4 Thesis Statement

Logic encryption is an evolving field of research in hardware security that is built to tackle the problem of IP piracy and misuse that plagues the semiconductor industry. The field is sufficiently advanced to have a plethora of implementation schemes, and an adversarial-style evolution chain of attacks and encryption strategies that respond to them. However, the primary obstacle for logic encryption is technology transition. One dimension of this that is rarely considered is that IC designs have strict power, performance, and area (PPA) specifications. The PPA costs of applying logic encryption are not well known, which is prohibitive to utilizing it in designs that could benefit from it. A couple logic encryption methods attempt to reduce area impact [39] or performance [21], but not power consumption. In most cases, encryption methods focus on

improving security metrics and protection against known attacks [5]. In addition to this, benchmark testing of logic encryption is mostly limited to very small test designs. An important part of transitioning the technology lies in proving it can be applied to real, large-scale designs. Logic encryption schemes are rarely tested on benchmark circuits totaling greater than 10,000 logic gates. Evaluations of state of the art logic encryption methods in terms of PPA impact are sorely lacking, and a rigorous methodology for implementing logic encryption in a physically efficient way only exists in a limited capacity [13], and considerations for large-scale designs are rarely made.

The purpose of this thesis is to respond to the need for power, performance, and area evaluations of state of the art logic encryption strategies and present a methodology for logic encryption driven by constraints on both cost and security metrics. The methods created in this work as a part of the constraint-directed logic encryption methodology are a binary design space search algorithm, a concurrent tree design space search algorithm, and a machine learning based approach. These algorithms are tested using both small and large design benchmarks. The primary benchmark sources are from the ITC '99 suite [2] and the Opencores database [14], the largest design of which contains over 200,000 logic gates. In the methodology, estimations of the PPA cost of logic encryption utilize commercial off-the-shelf (COTS) electronic design automation (EDA) tools. Primarily, power, timing, and area estimates from logic synthesis using Synopsys Design Compiler [29], and more accurate estimations from the design layout after place and route using Synopsys IC Compiler II [31] are used. The quality and efficiency of each of these estimations in optimizing PPA cost is evaluated. Other COTS EDA tools are utilized to inform and improve these estimations as well [30, 33]. Security metrics are estimated based on key size and known attack resilience for each logic encryption scheme. Methods that are a part of the proposed methodology use these metrics estimation strategies to rigorously craft an encryption strategy that meets PPA requirements.

The primary contribution of this work is a constraint-directed logic encryption methodology that acts as a framework for strategies that respond to constraints in the PPA cost and security provided by encryption. In addition to the overall methodology, a tree search-based method and a machine learning-based approach are presented to explore possible methods within the methodology. The effectiveness of each method is evaluated, and research directions are suggested based on the findings of the evaluation.

1.5 Thesis Overview

In Chapter 2, a survey of state of the art logic encryption methods and decryption attacks is given. Encryption methods are split into two regimes based on the SAT attack [28]: SAT-susceptible and SAT-resilient, along with additional sequential encryption methods. Attacks are focused on SAT-based attacks, though

others are also included.

In Chapter 3, a constraint-directed logic encryption methodology is presented. This gives a framework for which methods to optimize logic encryption for both security and cost metrics can be developed. As a part of the methodology, a concurrent tree search design-space exploration algorithm is proposed, and the results of encrypting chosen circuit designs with this algorithm are shown.

In Chapter 4, A machine learning approach of the constraint-directed logic encryption methodology is presented. The training details and results are shown, as well as results of encrypting chosen circuit designs with this approach.

In Chapter 5, the results of testing constraint-directed logic encryption methods are evaluated and discussed. Future research directions and improvements to the methodology are proposed based on the results.

Chapter 2

Logic Encryption Methods, Attacks, and Countermeasures

2.1 Brief History of Logic Encryption

Logic encryption is a hardware security technique first introduced by Roy, Koushanfar, and Markov in their 2010 paper, Ending Piracy of Integrated Circuits (EPIC) [21]. Their security strategy was devised to combat IP theft, which is a continuously growing concern in the field of hardware security. IP theft and misuse can come in the form of IP piracy/IC overproduction [20], reverse engineering [32], hardware Trojan insertion [20], and IC counterfeiting [8]. The locking mechanism proposed by Roy, et al. involved the addition of new primary inputs, called key inputs, to the circuit design. If the correct key is applied to these inputs, the output behavior of the circuit will be equivalent to the design specification. Under application of an incorrect key, the output behavior of one or more primary (non-key) input patterns is corrupted. This prevents theft of circuit IP by rendering the design unusable unless the correct key is known, which ideally is only known to the circuit designer. The designer can then activate each chip using the key, stored in non-volatile memory, before selling them on the market.

Early logic encryption methods aimed for enough output corruption to make an incorrect key obvious during testing, but not enough to easily eliminate incorrect keys. Thus, 50% hamming distance from normal output behavior under application of an incorrect key was the primary goal [5]. However, attacks against logic encryption were devised, necessitating novel and specific logic encryption strategies to respond to the attacks [17, 41]. Attacks can be classified into two types: oracle-guided and oracle-less. An oracle is a working copy of the circuit with which the correct input-output behavior of the circuit can be observed

without access to the netlist. This can come in the form of an IC obtained from the market, or a black box simulation of the layout. The most potent attack proposed among these was an oracle-guided attack proposed by Subramanyan, Ray, and Malik called the satisfiability attack, or SAT [28]. This attack leveraged a satisfiability engine to eliminate entire classes of keys at a time by only observing input-output behavior of the locked circuit and oracle. This was a vast improvement over other attacks of the time, which relied on brute force and/or structural analysis. Unlike other attacks, SAT was guaranteed to find the exact correct key without any brute force required, making it particularly potent.

After its proposal, the focus of the field shifted to SAT. Many logic encryption strategies were devised to specifically thwart SAT [24, 34, 36], new attacks and improvements to SAT were created to combat these methods specifically [23, 26, 37, 42], and even stronger encryption methods [19, 40, 39] attempted to prevent ever-strengthening attacks. This arms race of strengthening logic encryption versus increasingly potent attacks has lead to improved security metrics of logic encryption methods being the focus of the field, with cost metrics more or less thrown to the wayside. However, considering cost metrics is an important step to transitioning this technology to the commercial realm [5, 13].

For most of the past decade, logic encryption methods and attacks have exclusively been limited to combinational logic. For encryption methods, sequential behavior was not considered at all and attacks generally relied on flip-flops being a part of controllable and observable scan chains [5, 28]. However, recent research has involved developing encryption methods [9, 16, 12] that lock sequential logic and state machines, and attacks that target sequential behavior specifically, including an improvement to SAT that converts sequential logic to combinational through an unrolling process [11, 25].

In the rest of this chapter, a survey of logic encryption methods in Table 2.1 is performed. They are divided into SAT-vulnerable methods and SAT-resilient methods. Attacks in Table 2.2 are also discussed, with a focus on SAT-based attacks. The purpose of this survey is to identify logic encryption methods as candidates for inclusion in the constraint-directed logic encryption methodology. Encryption methods will

Table 2.1: Logic encryption methods surveyed

Encryption Schemes		
SAT-Vulnerable	SAT-Resilient	Sequential
EPIC [21]	SARLock [36]	Encrypt Flip-Flop [9]
SLL [38]	Anti-SAT [35]	ChainLock [11]
Key Interdependency [10]	TTLock [40]	StateLock [12]
Hardware Enlightening [22]	SFLL [39]	
	Cyclic Modification [24]	
	SRCLock [19]	

Table 2.2: Attacks against logic encryption surveyed

Attacks	
Oracle-Guided	Oracle-Less
Key Sensitization [17]	SPS [37]
SAT [28]	FALL [27]
AppSAT [23]	
CycSAT [42]	
AGR [37]	
Sequential SAT [11]	
BeSAT [26]	
SMT [1]	
KC2 [25]	

be chosen based on their resilience to known attacks, as well as their expected power, performance, and area costs.

2.2 SAT-Vulnerable Encryption Methods

- **EPIC** [21] – Ending Piracy of Integrated Circuits (EPIC) introduces logic encryption as a hardware security strategy. In this paper, Roy et al. suggest a method of protecting IC designs by locking them with a secret key before being sent to an outsourced foundry. When the correct key is applied to the circuit, then the IC functions normally. If an incorrect key is applied, then the output behavior of the circuit is corrupted for one or more input patterns. Only the circuit design holds the correct key value, so this locking meant to prevent the IP from being misused, stolen, or maliciously modified at the foundry assuming the correct key value cannot be discovered. When the ICs are returned from the foundries, the circuit designer can activate them with the correct key, and store the key in non-volatile memory. Then, the activated ICs are sold to the end user. The mechanism for this locking is logic locking. This involves adding additional primary inputs to the circuit that serve as key inputs. Additional logic gates, called key gates, are added to the design and are driven by the key inputs. The key gates are added in such a way that, under the correct key, the original output behavior of the circuit is recovered.
- **SLL** [38] – To combat the oracle-guided key sensitization attack against logic encryption (Section 2.3), Yasin et al. proposed Strong Logic Locking (SLL), with improvements to the placement of key gates in logic encryption to decrease the possibility of discovering correct key values through output

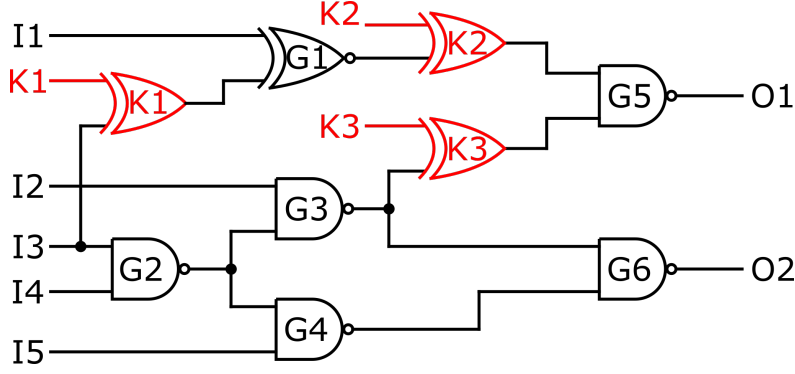


Figure 2.1: Example of a circuit encrypted using SLL [38] strategy.

observations. Their proposed attack strategy, key sensitization, relied on propagating key values to primary outputs with specific input patterns. They saw that some key gates interfered with each other, which required very specific input patterns to propagate a key value. Some key values simply could not even be propagated at all. These are known as mutable and non-mutable key gates respectively. They found that key gates that were placed in *pairwise secure* configurations, such as K2 and K3 in Figure 2.1 were non-mutable. Additionally, dominating key gates, such as K2, which dominates K1 in Figure 2.1, must be sensitized in a specific order, increasing attack complexity. Therefore, the authors suggest creating gate placements that maximize the instances of pairwise secure and dominating key gates to thwart their proposed attack.

- **Key Interdependency** [10] – The Key Interdependency technique is a three-step process for finding ideal locations to insert key gates. First, a set of gates in the design with the largest number of primary

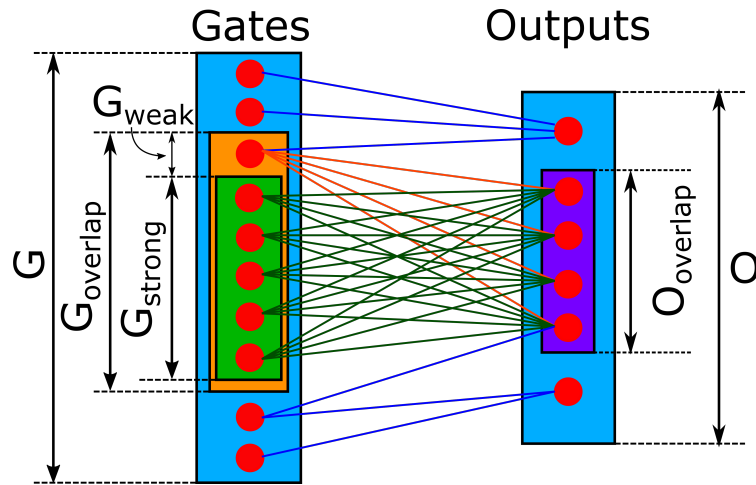


Figure 2.2: Key dependency [10] cone analysis. G_{strong} represents the input cones in which key gates are placed.

outputs in their combined cone of dependency is identified. For each of these gates, the fault impact metric $FI(g) = NP_0 * NO_0 + NP_1 * NO_1$ where NP_i and NO_i are the number of primary input patterns that sensitize a stuck-at- i fault and number of primary outputs affected by stuck-at- i faults, respectively. The gates with the highest FI are chosen for key gate insertion. This ensures maximum interference between key gates with overlapping output cones of influence.

- **Hardware Enlightening** [22] – In the Hardware Enlightening encryption method, ideal hardware Trojan insertion locations are chosen for locking. This ensures that Trojans cannot be easily inserted, and thus this method specifically counters Trojan insertion. The metric used to calculate ideal key gate locations is controllability and observability. Nets with low in these metrics are difficult to detect changes in (low observability) and only activated for a small amount of input patterns (low controllability). This combination makes them hard to detect during post-silicon validation, but still able to be activated. Key gates are inserted at nets with low controllability and observability, and these values are re-calculated at each insertion. To avoid reverse engineering, an inverter is inserted after some key gates, determined by the designer.

2.3 Decryption Attacks

- **Key Sensitization** [17] – The Key Sensitization attack is an early oracle-guided attack to decipher a logic encryption key. The attack attempts to sensitize correct key values by propagating them from key gates to a primary output using an oracle IC. This involves finding a suitable primary input pattern that propagates the correct key value by structural analysis of the netlist and key gate placement. Then, the output of the oracle IC can be observed to determine the correct key bit corresponding to the key gate being sensitized. This is performed for all key gates that can be sensitized, and the rest of the key bits are solved by brute force. The authors group key gates into several types based on their ease of sensitization:

- **Runs of Key Gates** – Key gates such that one or more are in the fan-in of another. These have several correct key values, and reduce the incorrect key sapce.
- **Isolated Key Gates** – These are key gates that do not interfere with each other, and can be sensitized individually
- **Dominating Key Gates** – A key gate is dominating if it lies on every path between another key gate and the primary outputs. The dominating key gate must be sensitized first.

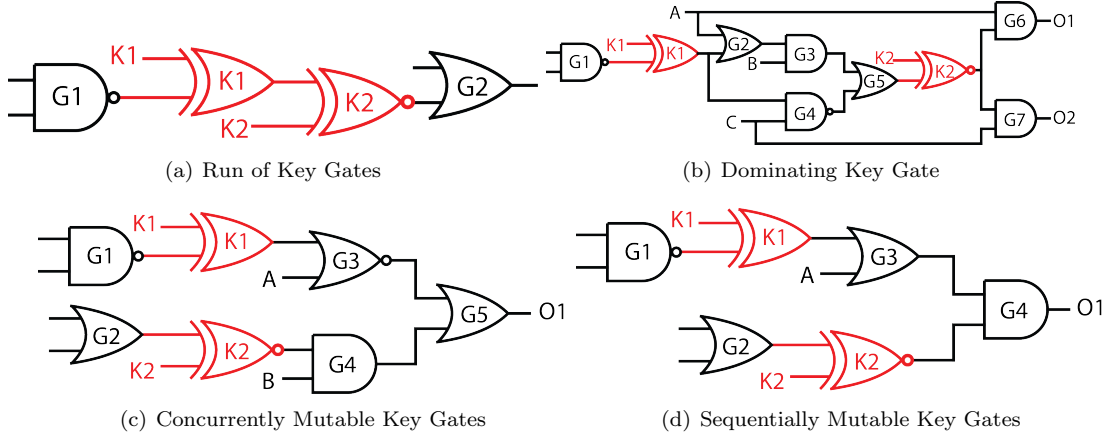


Figure 2.3: Key Sensitization Attack [17] Strategies

- **Concurrently Mutable Convergent Key Gates** – These key gates converge on a primary output, and may interfere with each other for some primary input patterns. Therefore, a specific primary input pattern that "mutes" the effects of one key gate must be found to sensitize the other, in any order.
 - **Sequentially Mutable Convergent Key Gates** – Similar to concurrently mutable key gates, a primary input pattern that mutes one key gate must be found to sensitize the other. However, a specific order of muting must be followed to successfully sensitize these.
 - **Non-Mutable Key Gates** – These key gates will always interfere for every primary input pattern. These must be solved by brute force after sensitization of other key bits.
- **SAT** [28] – The Satisfiability Attack (SAT) of Subramanyan, et al. is a particularly potent oracle-guided attack against logic encryption that uses a satisfiability engine with a miter circuit to quickly find and eliminate equivalence classes of incorrect keys and converge on the class of the correct key(s). The miter circuit consists of two copies of the encrypted circuit, such that they have the same primary input pattern, but different key inputs, and their outputs are compared, and the miter outputs high when the outputs are not equivalent. Primary input patterns that cause the miter to output high for different key patterns are called Distinguishing Input Patterns (DIPs). The attack is an algorithmic approach, which builds a Quantified Boolean Formula (QBF) by iteratively finding DIPs that satisfy the QBF, checking the oracle for the correct output, and adding clauses to the QBF that rule out key inputs that produce the incorrect output.

The initial QBF starts with two copies of the encrypted circuit, $C_E(\vec{X}, \vec{K}, \vec{Y})$, which has primary input $\vec{X}$, key input $\vec{K}$, and primary output $\vec{Y}$. A satisfiability engine finds $\vec{K}_1$ and $\vec{K}_2$ such that the QBF

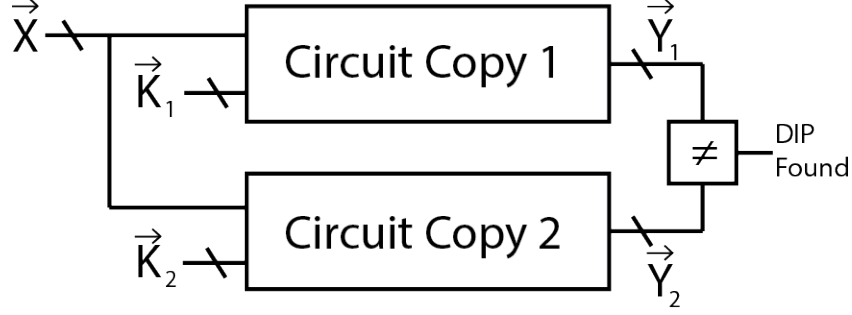


Figure 2.4: The SAT [28] miter circuit

$F = C_E(\vec{X}, \vec{K}_1, \vec{Y}_1) \wedge C_E(\vec{X}, \vec{K}_2, \vec{Y}_2) \wedge (\vec{Y}_1 \neq \vec{Y}_2)$ (i.e. the miter signal) is satisfied. The primary input pattern, $\vec{X}_d$, that satisfies this is a DIP. The DIP is applied to the IC oracle to obtain the correct output, $\vec{Y}_d$. Any key input that does not produce the output $\vec{Y}_d$ with the DIP applied can be eliminated. So, this clause is added to the QBF, which is reassigned as $F = F \wedge C_E(\vec{X}_d, \vec{K}_1, \vec{Y}_d) \wedge C_E(\vec{X}_d, \vec{K}_2, \vec{Y}_d)$, so that any new $\vec{K}_1$ and $\vec{K}_2$ that are found must produce $\vec{Y}_d$ for that DIP. Then, a new DIP that satisfies the QBF is found, and the process is repeated until no more satisfiable assignments for the QBF can be found. Any remaining key inputs represent the class of correct keys (which has a size equivalent to the number of correct keys, often one).

One reason SAT is a particularly potent attack is due to its ability to eliminate many keys in one attack iteration. Any key that does not produce the correct output for one DIP can be eliminated before moving on to the next DIP. This significantly reduces the number of attack iterations required versus a brute force method. Also, SAT is guaranteed to find the exact correct key for any logic encryption scheme upon termination. If SAT can find a satisfiable assignment for the QBF, it will be able to eliminate keys with the DIP found, otherwise the correct key(s) is found. Therefore, SAT will always converge towards the correct key. Due to this, sat resilient encryption methods will often attempt to thwart SAT by increasing the number of DIPs required to converge on the correct key, until it approaches the execution time of a brute force attack, or target the satisfiability engine itself rather than the attack behavior.

- **AppSAT** [23] – An SAT-based method that uses approximate methods, AppSAT, was proposed by Shamsi, et al. as a response to SAT-resilient encryption methods that rely on an added locking block, such as SARLock (Section 2.4). One important behavior of SAT recognized by the authors of AppSAT is the exactness constraint of the attack. That is, the attack does not terminate until it cannot find satisfying DIPs, which signifies that the exact correct key has been found. Instead, AppSAT finds a key with very low output corruption, rather than requiring the found key to be exactly correct.

This is done by modeling SAT as a query-by-disagreement problem, shrinking the key space until the remaining key space has an output corruption within an error threshold by estimating the error rate with queries. A key is selected from this reduced key space as "approximately correct".

The error rate of a key is found by intermediate output querying during SAT. Every d DIP iterations, the attack is paused to estimate the error rate, ϵ . The encrypted circuit, with a key that satisfies the SAT QBF applied, is queried with r random input patterns to estimate ϵ . This estimation is repeated every d iterations until ϵ is below a set error threshold for q consecutive queries (to ensure the error rate has "settled"). After this, the key used in the last query is accepted.

- **CycSAT** [42] – The SAT-resilient encryption scheme known as cyclic obfuscation (Section 2.4) was proposed to thwart the satisfiability engine of SAT by destabilizing outputs on application of an incorrect key. In response, Zhou, et al. proposed a SAT-based method to crack cyclic encryption, CycSAT. CycSAT adds a pre-processing step before executing SAT in which key conditions that disable combinational cycles are identified. These conditions are added to the initial SAT QBF as a "no cycle" clause in conjunctive normal form (CNF). This effectively converts the cyclic graph into a directed acyclic graph, which allows SAT to find DIPs again and crack the key. The authors propose two forms of the attack algorithm. The first is a structural CycSAT algorithm, in which the condition on combinational cycles is assumed to be on the key inputs only. The "no cycle" clause created in this algorithm, therefore, only contains key inputs. They also propose a logical CycSAT algorithm, in the case that the design is originally cyclic. This version of the algorithm extends the "no cycle" clause from only key inputs to all signals.
- **Removal Attacks** [37] – A set of attacks known as Removal Attacks were devised to target SAT-resilient encryption schemes that use a combinational locking block separate from the original circuit, such as SARLock (Section 2.4). In removal attacks, the logic corresponding to the locking block is first identified, and then it is disconnected from the rest of the design, without affecting the original design. Therefore, the key inputs driving the locking block will be effectively disabled, and can be ignored. The authors use two different methods to identify the locking block, which separates removal attacks into two attacks: Signal Probability Skew (SPS), and AppSAT-Guided Removal (AGR). In SPS, the probability skew of all nets in the design are calculated, where the probability skew is a metric defined by the skew of the signal's value towards '0' or '1' ($skew = |P(1) - 0.5|$). Due to the low corruption of SAT-resilient locking blocks, the output signal of the locking block is often heavily skewed. Therefore, the signals with the highest probability skew are selected as candidates for removal. The high skewed signal is disconnected from its fan-out, and the fan-out gate type determines the new signal that should

drive it to recover the correct function of the circuit ('0' or '1').

The second type of removal attack, AGR, is an oracle-guided removal attack that leverages AppSAT (see Section 2.3) to identify the locking block. The authors noticed that, during the execution of AppSAT, key bits corresponding to key gates outside of the SAT-resilient locking block settled into a constant value after only a few query iterations, while those corresponding to the locking block continued to change after many iterations. Therefore, AppSAT can be executed for some small amount of AppSAT queries (the authors suggest 4 queries) to see which key bits settle, and which are unstable. Structural analysis of the unstable key inputs is attempted to determine the locking block, which is disconnected from its fan-out similar to SPS. This is attempted repeatedly until the locking block is successfully removed.

- **Sequential SAT** [11] – SAT relies on attacking sequential designs using observable and controllable scan flip flops. If there are non-scan chain flip flops, then SAT is unusable against the encrypted sequential design. Therefore, Sequential SAT was developed by Kasarabada, et al. as a SAT-based method of attacking sequential designs without access to scan chains. The primary mechanism by which this is achieved is a circuit unrolling process. In unrolling, a sequential circuit is converted into a combinational equivalent for a certain number of clock cycles, determined by the unroll count. A number of copies of the circuit equal to the unroll count are created. Then, flip flops are replaced by buffers (assuming D-type flip flops), and connected to the corresponding fan-out of the next circuit copy iteration (or, to primary output for the final iteration). The buffers of the first copy are driven by logic '0', to simulate the 0 state. All key gates are all driven by a single set of key inputs. In sequential SAT, a circuit is unrolled at count 2, SAT is executed on the unrolled circuit, and a key is found. However, this key is not guaranteed to be correct, so the original circuit and oracle IC are simulated using a large amount of input vectors to test the key. In the case of an incorrect key, the

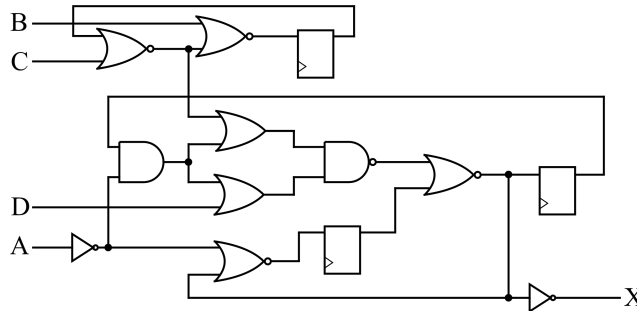


Figure 2.5: Example circuit to be unrolled

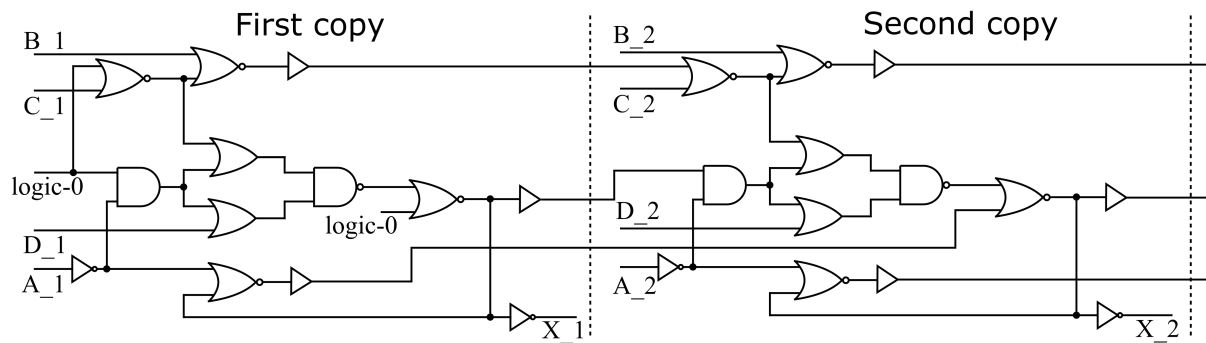


Figure 2.6: Sequential SAT [11] unrolled example circuit with unroll count = 2

attack is repeated for a larger unroll count. The unroll count is iteratively increased this way until the correct key is found.

2.4 SAT-Resilient Encryption Methods

- **SARLock** [36] – In response to the potent SAT attack (Section 2.3), Yasin, et al. proposed SARLock, a logic encryption method more advanced than its predecessors that is built to directly counteract SAT. This locking method began a series of methods which introduce a combinational locking block into the design. This group of methods, including SARLock, will be referred to as **point function** methods. In its proposed implementation SARLock, and other point function encryption methods, corrupt one or more primary output values with XOR or XNOR gates driven by the primary output and the output of the combinational locking block (thus replacing the primary output driver) and a "flip" signal produced by the locking block. The input of the combinational locking block consists of a number of primary input signals and new key inputs. If the number of primary inputs is not sufficient to drive the combinational logic block, additional signals in the corrupted logic cone(s) may be selected as inputs of the locking block.

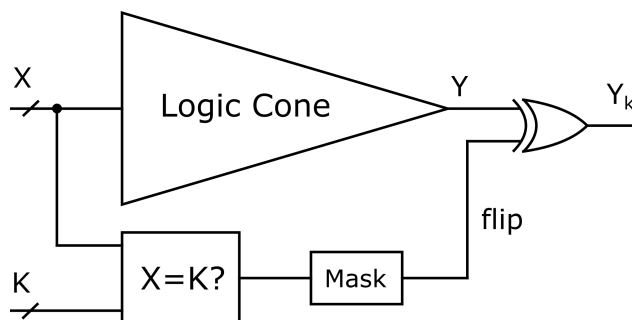


Figure 2.7: SARLock [36] architecture

For SARLock specifically, the combinational locking block consists of a comparator and a mask. The comparator tests for equality between the key inputs and primary inputs/chosen internal signals. If they are equal, the flip signal is activated. Otherwise, the flip signal is disabled. This behavior is not desired under the correct key, so a mask sets the flip signal to always be disabled under the correct key. This causes only one primary input pattern to be corrupted for each incorrect key pattern, thwarting SAT by only allowing it to eliminate one incorrect key per DIP. This effectively turns SAT into a brute force attack. The output corruption of SARLock is very low by design. Therefore, the authors suggest a compound encryption strategy, such that the design, locked with SARLock, is additionally locked with a SAT-susceptible method with a hamming distance of about 50% on incorrect keys, increasing the overall corruptability of the encryption. The key bits belonging to the SAT-susceptible scheme will be solved quickly by SAT, but solving the SARLock key bits still take an effectively indefinite amount of time.

- **Anti-SAT** [34] – Similar to SARLock, Anti-SAT, proposed by Xie and Srivastava, is a point function style encryption method with the same goal as SARLock to defeat SAT: introduce key inputs with low corruptability to eliminate the effectiveness of finding DIPs during SAT. Like other point function encryption schemes, Anti-SAT adds a combinational locking block that outputs a "flip" signal to corrupt one or more primary outputs, and has primary inputs, possibly internal signals, and added key inputs as inputs of the locking block. The architecture of the locking block includes two sub-blocks, one that implements a function $g(\vec{X}, \vec{K}_1)$, and another that implements its complement, $\bar{g}(\vec{X}, \vec{K}_2)$, where $\vec{X}$ are the non-key inputs of the block, and $\vec{K}_i$ are two separate sets of key inputs, each equal in size to $\vec{X}$. These sub-blocks drive an AND or OR gate (depending on if the flip signal should be active-high or low respectively), which drives the flip signal. The expected behavior of this architecture is that, under the correct key, g and $\bar{g}$ should output complimentary signals, and the flip signal will

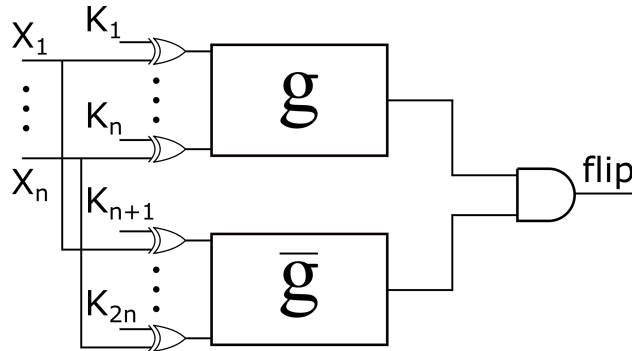


Figure 2.8: Anti-SAT [34] locking block architecture

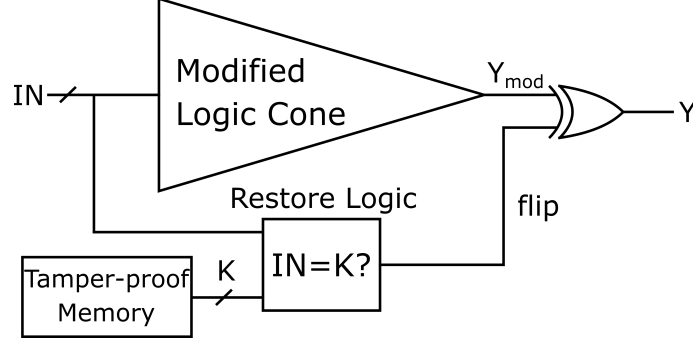


Figure 2.9: TTLock [40] architecture

always be inactive. Under the incorrect key, $g = \bar{g}$ for some small amount of input patterns, activating the flip signal with low corruptability. This architecture was shown to be effective against SAT. Like other point-functions, it is recommended to use a compound encryption scheme with a SAT-susceptible encryption scheme to increase output corruption for incorrect keys.

- **TTLock** [40] – The point function logic locking block of schemes like SARLock and Anti-SAT were shown to be vulnerable to attacks that target this architecture, especially removal attacks (Section 2.3). Yasin, et al. proposed TTLock as a way to render a circuit useless without the point function block. In TTLock, the logic cone(s) targeted for locking are modified by corrupting a "protected input pattern". This is a specifically chosen pattern, which is equivalent to the desired key. Then, a restoration block is added to the design, which compares the key input with the primary input (and any internal signals, depending on the desired key size). When these are equal, a flip signal is activated, which drives an XOR/XNOR gate on the corrupted output(s). Therefore, under the correct key, the output behavior of the protected input pattern is restored. Under an incorrect key, the output is corrupted for both the protected input pattern, and one other input pattern. This combats the removal attack, because if the locking block is removed, then the circuit still does not behave correctly. Unless the protected input pattern is known (which is only known to the circuit designer), then the circuit is still rendered unusable after the removal attack is executed. This also combats SAT because, outside of the protected input pattern, the locking block behaves equivalently to SARLock [36]. Unless SAT happens to find the protected input pattern as a DIP (which is highly unlikely), then SAT effectively becomes a brute force attack.
- **SFLL** [39] – Stripped-Functionality Logic Locking (SFLL) was developed soon after TTLock as an improvement to the logic cone modification technique. The authors of this method developed two sub-methods as a part of SFLL, SFLL-HD^h and SFLL-flex. This method was developed to be flexible, such

that the strength of the encryption scheme against known attacks, especially SAT and removal, could be configured. Both methods are similar to TTLock, but with the possibility of multiple protected input patterns (PIPs). In SPLL-HD^h, PIPs are any input patterns that are hamming distance of h (HD= h) from the secret key. When the secret key is chosen, the corrupted logic cones are modified such that the output(s) are corrupted when the PIPs are applied. Then, a restoration block is added, with an output flip signal that drives XOR/XNOR gates at the corrupted output(s), and the key and primary inputs (and any internal signals chosen) as inputs. The restoration block measures HD between the key input and non-key input, and activates the flip signal for any HD= h . Under the correct key, this restores the correct functionality for all PIPs. Under the incorrect key, the PIPs are still corrupted, along with other potential input patterns due to the HD restore block. It is noted by the authors that SPLL-HD⁰ is equivalent to TTLock. The authors found that values of h close to 0 or the key size provided strong SAT-resilience, while h around half the key size provided strong removal resilience.

SPLL-flex is a fully configurable sub-method of SPLL. In SPLL-flex specific PIPs are chosen by the circuit designer. PIPs can be any input patterns in any amount. For each PIP, a "flip vector" is chosen that determines which chosen primary outputs should be corrupted for that PIP, and the circuit is modified accordingly. Then, when the circuit is packaged for the market, a lookup table is added so each PIP will output its corresponding flip vector. Therefore, as presented, SPLL-flex has no key input. Rather, the PIPs themselves act as the key. The flip vector signals drive XOR/XNOR gates at the corrupted outputs. SPLL-flex was devised as a way to reduce the area cost of adding a restoration block by allowing the circuit designer to only strip input patterns that are critical to the design. Therefore, the cost of adding SPLL-flex is proportional to the size of the added LUT. The authors propose ways to mitigate this, such as a cube-compression process to decrease the size of the LUT.

- **Cyclic Modification** [24] – Unlike the other SAT-resilient encryption schemes discussed so far, which increase the number of DIPs required to crack the key, Cyclic Modification of Shamsi, et al. attempts to thwart SAT by breaking the satisfiability engine itself. In this method, combinational feedback

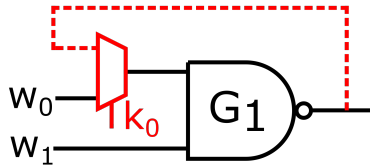


Figure 2.10: Cyclic modification [24] by added key-controlled multiplexer

loops are introduced into the design, which are broken upon application of the correct key. Under an incorrect key, the feedback cycles cause unstable output behavior, which breaks the SAT engine’s ability to find DIPs that satisfy the miter. Feedback cycles are controlled by key inputs, which each drive logic that can disable cycles, such as a multiplexer on an added feedback net. These cycles are ‘hard’, irreducible cycles with more than one entry point, which increases the difficulty of an attacker to break the combinational cycles by depth-first traversal. Specifically, a key-controlled multiplexer can be added on each edge of the cycle to make it irreducible.

- **SRCLock** [19] – After the introduction of CycSAT (Section 2.3), which is touted as a SAT-based attack which can decrypt cyclic encryption strategies, SRCLock was introduced by Roshanisefat, et al. as a set of countermeasures against CycSAT. The goal of the SRCLock encryption strategy is to create a densely connected feedback network that exponentially increases the number of feedback loops as key inputs are added. Increasing the number of feedback loops increases the time required to perform the key constraint pre-processing step of CycSAT. Therefore, with enough feedback loops, the attack execution time will become prohibitively long. This is primarily achieved using one of methods proposed by the authors: super-cycle generation and logarithmic feedback networks.

In super-cycle generation, first “micro-cycles” are created, which are key-controlled, non-reducible combinational feedback cycles, like those created in the cyclic modification encryption method. Once micro-cycles are created, they are connected together into super-cycles by creating key-controlled two-way connections between edges of micro-cycles. Under a correct key value, the micro-cycles are separate, but under an incorrect key value, the paths of the micro-cycles are crossed, connecting a super-cycle. This was shown to exponentially increase the number of feedback cycles in an encrypted design under an incorrect key with a linear increase in key size.

The other primary method proposed by the authors is a logarithmic feedback network. In this sub-method, several path segments in the cone of influence of a primary output are chosen. Each path segment is broken at a midpoint to create two segments, each with a start point and an end point. Start points and end points are interconnected with key-controlled multiplexers to create a densely connected feedback network. When connecting M startpoints and endpoints, $M(1 + \log_2(M))$ key bits are added. Therefore, the number of feedback cycles created is exponential with a linear increase in key size (number of multiplexers).

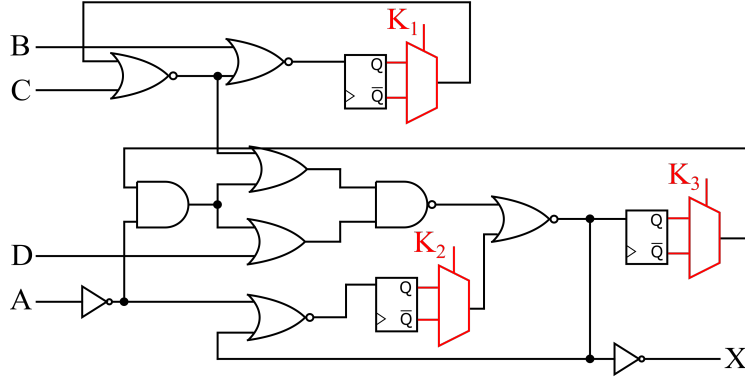


Figure 2.11: D-type flip flops encrypted with Encrypt-Flip Flop [9]

2.5 Sequential Locking Methods

- Encrypt Flip-Flop** [9] – Encrypt Flip-Flop (EFF) is a gate-level sequential encryption method in which scan-chain accessible flip-flops are encrypted by added multiplexers on their output. The non-select inputs of the multiplexer are connected to Q and $\bar{Q}$ outputs of the flip-flop, in an order depending on the desired correct key bit. Therefore, under an incorrect key, a corrupted output is passed from the flip-flop on the next clock edge. Key-controlled gates are placed in the fan-out of flip-flops for this method, and flip-flops are chosen for encryption using a method similar to the key interdependency method (Section 2.2) to ensure overlapping of the output cone of influence of encrypted flip-flops.
- ChainLock** [11] – Proposed by Kasarabada, et al. as a part of their proposed SAT-based attack on encrypted sequential circuits, ChainLock (renamed by the author after publishing) is another gate-level encryption scheme that locks the output of flip-flops. The authors propose a placement strategy of key gates based on the relative position of flip-flops, such that continuous chains of flip-flops between primary inputs and outputs are identified, and the longest chains are targeted for encryption. This is in response to the mechanism with which they attack sequential circuits with SAT, through a combinational unrolling process (see Section 2.3). The more unroll iterations required to find the correct key, the more complex the SAT problem is. Therefore, by encrypting long chains of flip flops, more unroll iterations are required, making the SAT problem prohibitively complex by execution time. In addition to encrypting long chains, chains with large fan-out leading outside the chain into a shorter chain (called sneak paths) are considered. In general, flip-flops with maximum distance to primary outputs in flip-flop chains are ideal targets for encryption in this scheme.
- StateLock** [12] – StateLock is a state-based locking method targeting sequential circuits developed

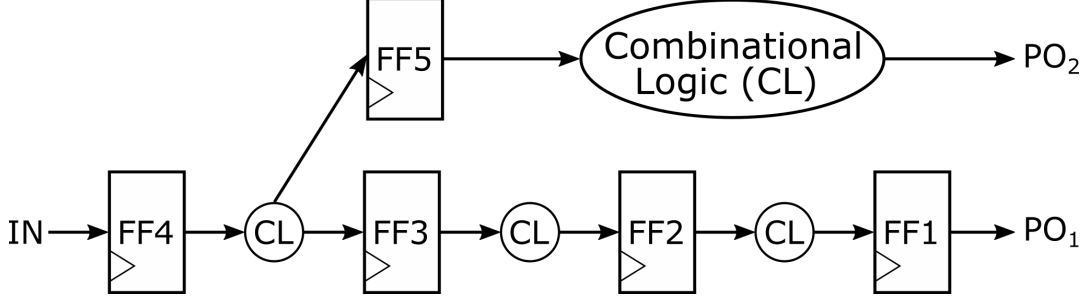


Figure 2.12: Flip flop chains, with a sneak path through FF5

by Kasarabada and Vemuri. This sequential logic encryption method locks the circuit when the finite state machine encounters a chosen rare state transition. When the rare state is encountered a set number of times (a counter is introduced to keep track of this), a corruption signal is activated to corrupt the output of the circuit. This thwarts unrolling-based SAT attacks (like Sequential SAT, see Section 2.3) by requiring a large amount of unroll iterations before the rare state is encountered by the solver. This obfuscation method has a low error rate (as is common among SAT-resilient locking methods), since the output is only corrupted when the rare state transition is encountered a certain number of times. The strategy proposed to increase the error rate of the locked circuit is to select several rare state transitions, which corrupt the output of the circuit when the incorrect key is applied.

2.6 Advanced Attacks

- **FALL** [27] – SFLL-HD^h (Section 2.4) was touted as a highly configurable and strong SAT-resilient encryption method. As a direct response to this encryption scheme, Sirone and Subramanyan developed Functional Analysis attacks on Logic Locking (FALL). FALL is a set of three oracle-less attack algorithms that each target the architecture of SFLL-HD^h specifically. All FALL attacks start by identifying the cube-stripping unit of SFLL-HD^h. In SFLL-HD^h, the original circuit is modified by corrupting protected input patterns (PIPs), which is done with a cube-stripping unit. Then, a restoration block restores the correct logic under application of the correct key. The cube-stripping unit calculates the hamming distance (HD) between the non-key input and the correct key, and contains the same non-key input as the restoration block. Therefore, the restoration block can be found by looking for comparisons between the non-key and key inputs. Then, the logic cone which has the same non-key inputs as the restoration block represents the cube-stripping unit.

The correct key is "baked" into the HD calculation of the cube stripping unit, and can be discovered by structural analysis. The authors created three structural analysis algorithms to uncover the correct

key, based on the value of h :

- **Analyze Unateness** ($h = 0$) – This algorithm specifically cracks TTLock/SFLL-HD⁰, using the unateness property. A Boolean function is positive unate in variable x_i if $\neg f_{\neg x_i} \vee f_{x_i}$ always holds and negative unate in x_i if $\neg f_{x_i} \vee f_{\neg x_i}$ always holds. The authors found that, for the cube-stripping function, all x_i inputs are unate, and x_i is positive unate when $k_i = 1$, and x_i is negative unate when $k_i = 0$. Therefore, the full correct key can be found by analyzing unateness of the cube-stripping block.
- **Distance2H** ($4h \leq k$) – Distance2H exploits the HD calculation of the cube stripping unit. Two satisfiable assignments on the cube-stripping unit are found that have exactly $HD = 2h$ from each other (they are also $HD=h$ from the correct key). Considering this, any bits that are equivalent between these two inputs must be equivalent to the corresponding correct key values. This is the ”non-overlapping” property of the cube-stripping unit, which will uncover $k - 2h$ correct key bits. The remaining key bits can be found by repeating the process, but with a clause excluding the inputs from the first iteration.
- **Sliding Window** ($2h < k$) – This algorithm is an extension of Distance2H. Following from the ”non-overlapping” property of the cube-stripping unit, for two inputs of the cube-stripping unit $\vec{X}_1$ and $\vec{X}_2$, $strip(\vec{X}_1) \wedge strip(\vec{X}_2) \wedge HD(\vec{X}_1, \vec{X}_2) = 2h \wedge x_j^1 = x_j^2 \wedge x_j^1 = b$ is satisfiable if and only if $b = k_j$ (the corresponding correct key value). After doing one iteration of Distance2H, the remaining correct key bits can be found by iteratively finding this satisfiability assignment on the remaining key inputs.

The authors found that these algorithms, as shown, increase in complexity and attack time, but also increase in flexibility in number of encrypted designs against which the attack successfully finds the correct key (by succeeding against a wider range of values for h). FALL is not, however, built to work against SFLL-flex at all.

- **BeSAT** [26] – BeSAT is a behavioral SAT-based attack that was developed due to CycSAT’s (Section 2.3) inability to capture all the cycles in a design. In BeSAT, two steps are added to SAT to find two regimes of keys identified by the authors: stateful and oscillatory keys. In cyclic encryption schemes, output values are unstable while combinational cycles are active. This causes statefulness in the output, that is, the measured output can vary. This can cause the satisfiability engine to find the same assignment for all variables in each SAT iteration, and SAT will be stuck in an infinite loop trying a single DIP repeatedly. To respond to this behavior, BeSAT checks the DIP at each SAT iteration for

equivalence with the previous DIP. On a recurring DIP, the attack tests $\vec{K}_1$ and $\vec{K}_2$ for statefulness. A clause is added to the QBF to ensure either or both keys are assigned these stateful values during the attack. This will release SAT from the infinite loop.

BeSAT also checks for oscillatory keys, key values that result in oscillation of output values. This is another behavior introduced with combinational cycles. An oscillating output can cause the SAT QBF to be unsatisfiable, which causes the oscillating key to be returned by SAT as the correct key. In BeSAT, this is avoided by recognizing "unknown" as a third logic value. The problem is seen as a ternary logic problem, with binary encoding achieved by dual-rail encoding. With this ternary system and an inclusion of "unknown" signal detection, a satisfiable assignment for the QBF can be found regardless of output oscillation.

- **SMT** [1] – The Satisfiability Modulo Theory (SMT) attack was proposed by Azar, et al. as a superset of SAT-based attacks. That is, SMT is an attack that includes SAT as a solving method, but SAT is only one of several methods used in the attack methodology. SAT is a strong solver for Boolean logic-based locking. However, recent locking methods include behavioral locking methods, such as changes to setup and hold time. These are behavioral locking methods, rather than Boolean locking methods, which SAT cannot possibly unlock. Therefore SMT, rather than being a Boolean Satisfiability method, uses graph theory to introduce other forms of Satisfiability and is capable of solving locking problems outside of Boolean locking, including behavioral locking. This makes it much more flexible than SAT, allowing it to solve approximate problems and cyclic encryption methods as well.
- **KC2** [25] – Key-Condition Crunching (KC2) is a sequential attack strategy proposed by Shamsi et al. which, similar to Sequential SAT, uses iterative circuit unrolling and SAT solving to uncover the correct key. However, a model checker is also used to introduce speedup into this process, making this attack an improvement to the Sequential SAT attack. Various techniques are used in this method to simplify key conditions on the unrolled circuit in each iteration in order to speed up the SAT solver. These techniques attempt to decrease the number of clauses that the SAT engine must solve in various ways. This includes iterative SAT-solving that builds on past calls of the SAT solver, simplifying the Binary Decision Diagram (BDD) on which the SAT problem is performed, and setting early termination conditions.

2.7 Summary

In this chapter, a summary of the history of logic encryption was given, and several state of the art encryption schemes and known decryption attacks developed in the past decade were outlined. In the next chapter, a new approach to logic encryption, driven by design constraints, will be proposed.

Chapter 3

Constraint-Directed Logic Encryption

3.1 Motivation

The evolution of logic encryption has been almost exclusively guided by improvements in security metrics to better protect against attacks and threats that ever increase in complexity. Security metrics are important to consider for encrypted design, as an encryption scheme that is vulnerable to attack is wasteful. However, the design must also meet strict cost constraints in a typical VLSI flow, especially in terms of power usage, performance, and die area (PPA). A careful study of the trade-off between security and cost metrics is lacking in research related to logic encryption [5]. A couple of novel encryption schemes, such as SFLL [39], consider the cost of encryption. A holistic, flexible, and extensible methodology for balancing security and cost constraints must be developed in order for logic encryption to be considered for real VLSI solutions. This is an important step the technology must take to transition to commercial use. Rather than consider PPA cost within each novel encryption scheme, a methodology for crafting an encryption strategy optimized for both security and cost using available encryption methods should be developed. Such a methodology is extensible to novel encryption methods as they are developed, and can evolve with the field of logic encryption rather than being limited to one encryption method or a group of methods.

A method for exploring encrypted designs within PPA constraints has been introduced by Luria, et al. [13]. In that method, a search algorithm based on binary search was presented, and a generalized methodology was mentioned. However, it relied on a heavy-handed process based on place and route for PPA estimation, and was only executed on small, combinational benchmarks. In addition, the broader methodology was not discussed in detail, and other methods that could be a part of the methodology were not discussed. Therefore, in this work, the methodology will be generalized and other methods for finding an

optimal design will be explored. In this chapter, in addition to giving an overview of the method proposed in that work, a concurrent tree search algorithm will also be presented. This algorithm is a direct improvement to the binary search-based method by executing the PPA estimation of several designs in parallel. The parallelization of PPA collection of several designs is straightforward, so this introduces a direct speedup related to the number of threads available for computation. Other methods of speeding up the search are proposed as well. In addition to this, a machine learning-based approach will be introduced in the next chapter.

3.2 Methodology Overview

A proposed constraint-directed logic encryption (CDLE) methodology balances cost and security to find an optimal encryption strategy. Consider a design C to be encrypted with a desired security goal in mind, S_{opt} , which can include, for example, attacks to protect against and an allowed key size range. Assume that the design specification of C includes a set, PPA_{inc} , which represents the allowed increase in power, performance, and area after encryption. There is a version space of possible designs, D_C , containing all possible encrypted versions of C using known encryption methods. An encrypted version of the design $C_E^i \in D_C$ has its own encryption parameters which include, but is not limited to, an obfuscation scheme and key size, and has its own security and cost metrics. There is a subset of designs that meet S_{opt} , $D_C^S \subseteq D_C$, and another subset of designs that have PPA costs less than PPA_{inc} , $D_C^{ppa} \subseteq D_C$. Consider the set $D_C^{opt} = D_C^S \cap D_C^{ppa}$, which represents the set of designs in D_C that meet the security goal while also staying within the PPA constraints of the design specification. If $D_C^{opt} \neq \emptyset$, then $\exists C_E^{opt} \in D_C^{opt}$, the encrypted version of C within D_C^{opt} which maximizes all security metrics.

The goal of CDLE is, first, to identify the subset of encrypted designs D_C^{opt} , and then find the optimized design C_E^{opt} within that set which maximizes each security metric. Therefore, the methodology can be seen as a set of functions $F_i(C, S_{opt}, PPA_{inc})$ that have the design C , security goal S_{opt} , and the maximum allowed PPA increase PPA_{inc} as input, and output an optimally encrypted design C_E^{opt} . Theoretically, there is an exact correct C_E^{opt} , such that increasing the security metrics, even slightly, exits the optimized design space D_C^{opt} .

Ideally, all algorithms in the methodology would return the same C_E^{opt} . However, different methods will estimate PPA and security metrics for each design in D_C differently, and may produce a variety of estimations of C_E^{opt} as well. Additionally, methods will have a variety of expected execution time, which can depend on the method itself, as well as design-specific parameters such as post-synthesis gate count. Methods in the methodology should aim to return an accurate C_E^{opt} , but should also have a reasonable execution time, based

on the needs of the circuit designer. In general, using more accurate PPA and security estimation methods that produce a more accurate C_E^{opt} could result in longer execution time, creating a trade-off between these two properties of the CDLE methodology. However, this trade-off does not necessarily hold. As methods of metrics estimation are explored, some methods could prove to be a direct improvement to others. This work begins to explore the methodology with this in mind.

3.3 CDLE using Binary Search

The constraint-directed logic encryption method of Luria, Kasarabada, and Vemuri [13], though basic, does fit into the scheme of the methodology proposed in this work. Therefore, it can serve as an introductory example for the methodology. In this design-space search algorithm, the set of encrypted designs that meet the security goal, D_C^S , was determined by grading the available encryption methods in terms of attack resilience. In this method, only SAT-resilient encryption schemes were included in D_C^S , using randomly generated correct key patterns for all encrypted designs. Only combinational benchmarks were tested, so sequential locking methods were ignored. The quality of security provided by a design was based on the encryption scheme grade, and the key size. Encryption schemes used were (in order of decreasing security grade):

- | | |
|------------|-------------|
| 1. SFLL-HD | 4. TTLock |
| 2. SRCLock | 5. Anti-SAT |
| 3. Cyclic | 6. SARLock |

The search algorithm proposed by Luria, et al. [13] is shown in Algorithm 1. First, the PPA of the original design is measured. Then, for each encryption scheme in the ranking, the design is encrypted at the minimum allowed key size, k_{min} . If $C_E^{k_{min}}$ does not meet the PPA constraint, then the algorithm moves to the next encryption scheme in the rating. If it does meet the constraint, then the design is encrypted at the maximum allowed key size, k_{max} . If $C_E^{k_{max}}$ meets the PPA constraint, then the design is immediately accepted as the optimal encrypted design, C_E^{opt} . Otherwise, the optimally secure design has a key size within k_{min} and k_{max} . In that case, the space of potential key sizes, between the maximum and minimum key size, is searched by binary search, encrypting the design and measuring the PPA of C_E^i at each searched key size. The algorithm terminates when the optimal key size is found, such that increasing the key size by 1 breaks PPA_{max} . If all encryption schemes are tried without finding C_E^{opt} , then the algorithm fails ($D_C^{opt} = \emptyset$).

The authors used the following strategy for constraining PPA for the original design and each encrypted

Algorithm 1 Constraint-Directed Logic Encryption using Binary Search

Require: Original Circuit C ; Key width bounds k_{min}, k_{max} ; Cost constraints PPA_{max} ; Ranked encryption methods E

Ensure: Encrypted circuit C_E^{opt}

```
1:  $PPA_o \leftarrow get\_cost(C)$ 
2: for  $e \in E$  do
3:    $C_e^{k_{min}} \leftarrow encrypt(C, e, k_{min})$ 
4:    $PPA_e^{k_{min}} \leftarrow get\_cost(C_e^{k_{min}})$ 
5:   if  $PPA_e^{k_{min}} \leq PPA_{max}$  then
6:      $C_e^{k_{max}} \leftarrow encrypt(C, e, k_{max})$ 
7:      $PPA_e^{k_{max}} \leftarrow get\_cost(C_e^{k_{max}})$ 
8:     if  $PPA_e^{k_{max}} \leq PPA_{max}$  then
9:       return  $C_e^{k_{max}}$ 
10:    end if
11:    $k_{left} = k_{min}, k_{right} = k_{max}$ 
12:   while  $k_{left} \leq k_{right}$  do
13:      $k_{mid} = floor((k_{left} + k_{right})/2)$ 
14:      $C_e^{k_{mid}} \leftarrow encrypt(C, e, k_{mid})$ 
15:      $PPA_e^{k_{mid}} \leftarrow get\_cost(C_e^{k_{mid}})$ 
16:     if  $PPA_e^{k_{mid}} \leq PPA_{max}$  then
17:        $k_{left} = k_{mid}$ 
18:     else
19:        $k_{right} = k_{mid} - 1$ 
20:     end if
21:   end while
22:   return  $C_e^{k_{left}}$ 
23: end if
24: end for
25: return failure
```

version:

- **Power** – This was constrained as a percentage of the power usage of the original design. Both switching and leakage power were considered in estimations.
- **Performance** – The maximum delay overhead on the critical path was determined by the allowed slack. Any encrypted design should have a slack greater than or equal to 0 on the critical path.
- **Area** – The sum total combinational area and interconnect area was used as the area calculation. A set percentage increase from the original design area was the constraint for encrypted designs (similar to the power constraint).

For this method, the PPA of each design was estimated using both Synopsys Design Compiler (DC) [29] and Synopsys IC Compiler II (ICC) [31]. For each of these, a design was simulated using Icarus Verilog [33] to collect switching information to inform and increase the accuracy of DC and ICC PPA estimations. This collection represents the *get_cost* function of Algorithm 1. For simulation, a testbench of 10,000 total test patterns was generated once per design, and used for all encrypted versions of the design. Synopsys

Tetramax [30] automated test pattern generator was utilized to stimulate critical paths in the design, and then the rest of the test patterns were randomly generated.

The algorithm had three main steps, the first of which was encryption of the original design which involves DC execution time, t_{DC} , and ICC execution time, t_{ICC} . It was found that $t_{ICC} \gg t_{DC}$, so this had an execution time of t_{ICC} . In the next step, PPA is measured at k_{min} for potentially every encryption scheme in the set of encryption schemes. Therefore, step two had a worst-case execution time of $N_E t_{ICC}$ for N_E encryption schemes tried. Then, in step three, the best key size is found by binary search. This had a worst-case execution time of $\log_2(k_{max} - k_{min}) t_{ICC}$ to run PPA collection at every step of the binary search. As a whole, the algorithm had a worst-case execution time of $t_{exec} = t_{ICC}(1 + N_E + \log_2(k_{max} - k_{min}))$, assuming t_{ICC} is approximately equal for all designs.

This constraint-directed logic encryption method was only tested for small, combinational benchmarks with less than 4,000 logic gates. Execution time was not reported in their work. However, place and route is a complicated problem, and increases in complexity with design gate count. The complexity of PPA estimation and execution time limited the data set in this CDLE method. Furthermore, the algorithm's time complexity is directly related to t_{ICC} , resulting in a very heavy-handed, though accurate, solution to the constraint-directed logic encryption methodology. In the remainder of this chapter, improvements to this algorithm will be explored to, one, extend this method to sequential designs, two, drastically decrease execution time, and three, improve its applicability to a set of larger, more realistic designs.

3.4 CDLE using Concurrent Tree Search

In the binary search constraint-directed logic encryption implementation shown in Section 3.3, PPA measurements are carried out for one design at a time. Therefore, for every design that needs to be produced and have PPA estimated to converge on C_E^{opt} , the overall execution time of that CDLE algorithm is increased by the execution time of IC Compiler place and route, t_{ICC} . In the worst case, the number of times ICC was executed to find the best encryption scheme among N_E schemes is $O(N_E)$, and $O(\log(k_{range}))$ times to find the best key size among the range of acceptable key sizes, k_{range} . Using ICC to estimate PPA is very accurate, but also very time consuming. Therefore, effectively reducing the number of times this estimation needs to be performed to arrive at C_E^{opt} is one strategy for decreasing the overall execution time of the algorithm.

One property of the PPA estimation of different encrypted designs in D_C is the *parallelization* property. At each step of the binary search algorithm of CDLE, an encrypted version of the original design, $C_E^i \in D_C$ is produced and PPA is estimated using Design Compiler and IC Compiler. However, this process is completely

independent of doing the same process for a separate encrypted design, C_E^j . Therefore, the PPA of C_E^i and C_E^j can be estimated simultaneously. Due to this property of this PPA collection strategy, the CDLE search method can be done in parallel for different designs. The parallelization property of PPA collection will be the basis upon which the Concurrent Tree Search solution to CDLE is built. Therefore, it can be seen as a direct improvement to the CDLE binary search algorithm. This modification to the algorithm will theoretically return the same C_E^{opt} , but in faster execution time than the binary search algorithm.

3.4.1 PPA Estimation Strategy

The PPA estimation strategy of the concurrent tree search is almost identical to that of the binary search algorithm (Section 3.3), except executed in parallel. In this strategy, there are two versions of the PPA estimation flow: one for the original design, C , and one for encrypted versions that are explored, C_E^i . Each of these PPA estimation flows involves a series of steps to inform and carry out the estimation. This involves interfacing several external commercial off-the-shelf (COTS) electronic design automation (EDA) tools. These tools are Synopsys Design Compiler (DC) [29] for logic synthesis, Synopsys IC Compiler II (ICC) [31] for place and route synthesis, Synopsys Tetramax [30] for automated test pattern generation (ATPG), and Icarus Verilog [33] for netlist-level simulation. PPA estimations are based on the following factors:

- **Power** – Power is reported with both cell switching and leakage power. Switching power is determined by annotating cells with user-provided switching information, and leakage power is determined by the cell library.
- **Performance** – The critical timing of the design determines its performance. The maximum delay with primary inputs and flip-flops serving as start points and primary outputs and flip-flops serving as end points is measured as the critical path delay. The allowed performance increase is used to determine the slack on this path.
- **Area** – The area is measured as the total footprint of the core area, including logic cells, net connections, I/O, and filler area in the design.

Each PPA estimation flow is divided into two sub-flows. The first is estimation using Design Compiler, in which the design is simulated for switching information, and then PPA estimated for the netlist after logic synthesis. The second is estimation using IC Compiler, in which the design is simulated again, using the synthesized netlist, and then the IC layout is placed and routed. Then, PPA is estimated more accurately using the produced layout.

Pre-encryption Design

The PPA of C is estimated, first using DC, with the following steps:

1. **Preliminary Logic Synthesis** – The design is synthesized to produce a test strategy for ATPG, and a technology-mapped netlist for simulation.
2. **Test Pattern Generation** – Tetramax fast sequential ATPG is used to produce test patterns that stimulate critical paths of the design. Additional random test patterns are generated to produce a total of 5000 patterns, including those collected from ATPG.
3. **Simulation** – The design is simulated using the test patterns generated in the previous step. This produces switching information for nets in the design, which will improve the accuracy of the power estimation.
4. **DC PPA Estimation** – PPA is estimated using logic synthesis. After synthesis is complete, PPA is reported by Design Compiler using the identified metrics.

After a fully synthesized and technology-mapped netlist is produced, the PPA is estimated more accurately using ICC:

1. **Simulation** – The design is simulated again using the same test patterns as the DC estimation.
2. **ICC PPA Estimation** – PPA is estimated using an IC layout. After place and route are complete, PPA is reported by IC Compiler using the same identified metrics.

Encrypted Designs

The PPA estimation flow of all C_E^i is similar to that of C , but with certain modifications. One of the main modifications is the removal of test pattern generation. Since under application of the correct key $C_E^i = C$, the same test pattern strategy from the PPA estimation of C can be used for C_E^i , with the addition of the correct key pattern being constantly applied to set of key inputs. Therefore, the test pattern generation step can be skipped for all C_E^i , and the testbench of the simulation step can reuse the test patterns used for C , with the correct key being constantly applied throughout simulation. Therefore, The steps to estimate PPA with DC are:

1. **Encryption** – C is encrypted using the encryption parameters corresponding to the target C_E^i , including encryption scheme and key size. The design is encrypted with a randomly generated correct key pattern, unique to the specific C_E^i being produced.

2. **Preliminary Logic Synthesis** – C_E^i is synthesized to produce a technology-mapped netlist for simulation.
3. **Simulation** – The encrypted design is simulated using the same test patterns as C . This produces switching information for nets in the design, which will improve the accuracy of the power estimation.
4. **DC PPA Estimation** – PPA is estimated using logic synthesis. After synthesis is complete, PPA is reported by Design Compiler using the identified metrics.

After PPA is estimated using DC, the results are compared with the ICC PPA results of the original design. If the PPA of the encrypted design meet the allowed PPA cost set by the designer, then the design is accepted as a part of D_C^{opt} , and PPA estimation is concluded for this C_E^i . If the PPA constraints are not met, then PPA estimation continues to ICC estimation for more accurate results:

1. **Simulation** – The design is simulated again using the same test patterns as the DC estimation.
2. **ICC PPA Estimation** – PPA is estimated using an IC layout. After place and route are complete, PPA is reported by IC Compiler using the same identified metrics.

Figure 3.1 shows the combined PPA estimation strategy for the concurrent tree search CDLE method. The number of PPA estimations for encrypted designs that can be run concurrently is dependent on the number of available threads to execute the estimation flows. One additional piece of the strategy to improve the parallel execution of the method is concurrent execution of PPA estimations for C and the first set of encrypted designs that are explored. Rather than waiting for PPA estimations of C to finish before beginning parallel execution of encrypted designs, the PPA estimation of the C can be added to the first parallel exploration of encrypted designs. This is feasible with the addition of a few locks and considerations:

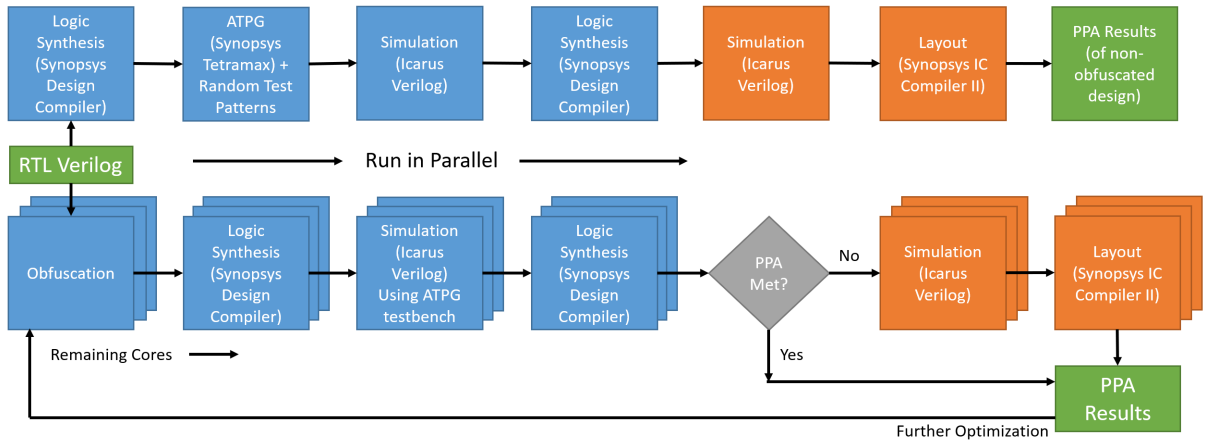


Figure 3.1: Parallel PPA collection strategy

1. Simulation of any design cannot begin without test patterns. Therefore, the test pattern generation step of the PPA flow of C must complete before any simulation is done for any designs. This can cause execution of PPA estimations of C_E^i to pause until test patterns are available.
2. For any DC PPA estimations of C_E^i , if the post-ICC PPA estimation of C is not completed, then ICC PPA estimation of C_E^i will be executed without checking if the DC PPA meets the constraints.
3. For any ICC PPA estimations of C_E^i , if the post-ICC PPA estimation of C is not completed, then the process must wait until the post-ICC PPA estimation of C is completed before deciding if the PPA constraints have been violated.

This PPA estimation strategy will be used in the concurrent tree method of CDLE as an attempt to improve execution time over the binary search method, while still maintaining the same level of accuracy. This can be seen as an improved *get_cost* function that can handle several designs at once, rather than one. Since ppa estimation is embarrassingly parallel, especially for any C_E^i that are being explored, this should introduce a speedup directly related to the number of parallel computation threads that are available.

3.4.2 Security Evaluation of Encryption Schemes

Each logic encryption schemes has attack resistance, often by design, that has been experimentally shown by the authors of the scheme. Refer to Tables 3.1 and 3.2 for a summary of this theoretical attack resilience per encryption scheme. SMT was excluded from the attacks in this table because, behavioral locking mechanisms are outside the scope of this work, and otherwise SMT behaves as SAT does. This table will be the basis for which to measure attack resilience of a particular encryption strategy. The attack resilience will be assumed to be the sum of the attacks against which each encryption scheme applied to each design protects. Therefore, several compound encryption schemes can be formulated to create a security strategy that protects against specific attacks.

Table 3.1: Resiliency table for combinational encryption methods. $\times$ denotes resilience, $\approx$ denotes resilience based on sub-method

	Sensitization	SAT	AppSAT	CycSAT	Removal	FALL	BeSAT
EPIC					$\times$	$\times$	
SLL	$\times$				$\times$	$\times$	
Key Int.	$\times$				$\times$	$\times$	
SARLock	$\times$	$\times$		$\times$		$\times$	$\times$
AntiSAT	$\times$	$\times$		$\times$		$\times$	$\times$
TTLock	$\times$	$\times$		$\times$	$\times$		$\times$
SFLL	$\times$	$\times$	$\times$	$\times$	$\times$	$\approx$	$\times$
Cyclic	$\times$	$\times$	$\times$		$\times$	$\times$	
SRClock	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	

Table 3.2: Resiliency Table for sequential encryption methods. ✗ denotes resilience

	Sequential SAT	KC2
EFF		
ChainLock	✗	
StateLock	✗	✗

Below is an attack resilience evaluation of each logic encryption scheme shown in Tables 3.1 and 3.2:

- **EPIC** [21] – The random key gate placement strategy of EPIC makes it vulnerable to all attacks against logic encryption unless used in a compound encryption scheme.
- **SLL** [38] – Key sensitization is the attack that SLL is built to combat, but it does not protect against SAT-based attacks and others developed after it.
- **Key Interdependency** [10] – Due to its placement strategy of key gates interfering, Key Interdependency protects against key sensitization. It does not protect against any SAT-based or post-SAT attacks.
- **Hardware Enlightening** [22] – Hardware enlightening is built to protect against Trojan insertion, however it does not specifically protect against attacks that target logic encryption.
- **SARLock** [36] – SARLock was one of the first SAT-resilient encryption schemes developed. Also, as a SAT-resilient method, it also protects against key sensitization due to the complex nature of the locking block. However, it is susceptible to removal and AppSAT.
- **Anti-SAT** [34] – Similar to SARLock, Anti-SAT was also one of the first SAT-resilient encryption schemes developed, but is susceptible to removal and AppSAT.
- **TTLock** [40] – As a logic cone modification technique, TTTLock protects against removal attacks as well as SAT. It is still susceptible to AppSAT, though, but not as susceptible as SARLock and Anti-SAT.
- **SFLL** [39] – SFLL-HD^h is an improvement to TTTLock, and therefore also protects against removal attacks as well as SAT, and fully protects against AppSAT by increasing the corruptability of each incorrect key. The level of protection from these attacks can be configured with h . SFLL-flex does not lock a design with key inputs, and therefore logic encryption attacks are not applicable.
- **Cyclic Modification** [24] – Cyclic modification thwarts SAT and AppSAT by breaking the satisfiability engine, and the logic is embedded into the design, thwarting removal attacks. CycSAT and BeSAT are built to crack cyclic encryption, so it is susceptible to those.

- **SRCLock** [19] – As an improvement to cyclic encryption, This attack protects against SAT-based methods. SRCLock specifically protects against CycSAT as well, though its behavior against BeSAT is less known.
- **Encrypt Flip-Flop** [9] – Encrypt flip-flop is susceptible to sequential attack methods because no particular strategy for the placement of key gates is given to thwart them.
- **ChainLock** [11] – Developed specifically to increase the unrolling complexity of Sequential SAT, ChainLock protects against that attack. However, more advanced sequential-based attacks, such as KC2, can work around this.
- **StateLock** [12] – StateLock protects against Sequential SAT by protecting states deep in the finite state machine. This also increases the complexity of executing KC2.

For each design, a ranking of possible compound encryption strategies based on their proven resilience to known attack methods is used to quantify the attack resilience security metric, with increasing attack coverage among rank. First, encryption methods were chosen based on their resilience to SAT-based and removal attacks:

1. SFL
2. SRCLock
3. Cyclic Modification
4. SARLock
5. Anti-SAT

Then, methods were ranked in order of resilience to sequential attack methods. StateLock was excluded to reduce encryption implementation overhead:

1. ChainLock
2. Encrypt Flip-Flop

These SAT-resilient and sequential encryption methods were combined to produce the final ranking:

- | | |
|------------------------|-------------------------|
| 1. ChainLock + SFLL | 6. EFF + Cyclic |
| 2. EFF + SFLL | 7. ChainLock + SARLock |
| 3. ChainLock + SRCLock | 8. EFF + SARLock |
| 4. EFF + SRCLock | 9. ChainLock + Anti-SAT |
| 5. ChainLock + Cyclic | 10. EFF + Anti-SAT |

This set of compound encryption schemes forms the set of acceptable encryption schemes implemented in D_C^S , the set of encrypted versions of the design C which fulfill the security goal, S_{opt} . The rating will serve to find the maximum level of security within D_C^{opt} , the set of designs that meet both the PPA constraint and the security goal. Designs within D_C^{opt} which maximize the encryption scheme rating will maximize the number of attacks against which the design is resilient, maximizing the overall level of security. Then, the key size can be maximized to further maximize security. These are the two security metrics upon which the concurrent tree search-based CDLE method bases D_C^S , and C_E^{opt} .

Authors of several SAT-resilient encryption schemes recommend pairing a SAT-resilient encryption scheme with a SAT-susceptible scheme to ensure an adequate amount of output corruption from the obfuscation strategy, as several SAT-resilient schemes have low corruption on incorrect keys. Randomly placed key gates (as in EPIC [21]) was chosen as the SAT-susceptible scheme in this method, due to its straightforward and quick implementation, and since no additional attack resilience is required from the added scheme. For each compound encryption scheme listed, 45% of key inputs are dedicated to each listed obfuscation method, and 10% are dedicated to randomly placed key gate encryption.

3.4.3 Design-Space Exploration Algorithm

The algorithm for the concurrent tree search CDLE method is a modified version of the binary search-based method in Section 3.3. Refer to Algorithm 2. The original circuit C , minimum and maximum integer key sizes allowed k_{min} and k_{max} , cost constraints PPA_{max} , ranked encryption methods E , and a new input, the integer number of concurrent computation threads T ($T \geq 1$). In this algorithm, the function *encrypt* produces a set of encrypted designs of C , rather than one design as in the binary search-based algorithm, using a set of encryption schemes at one key size, or one encryption scheme at a set of key sizes. Also, *get_cost* becomes *get_costs*, which returns sets of PPA values for a set of designs, rather than one design, as described in Section 3.4.1. The encrypted design with maximum security within D_C^{opt} is ensured upon

Algorithm 2 Constraint-Directed Logic Encryption using Concurrent Tree Search

Require: Original Circuit C ; Key width bounds k_{min}, k_{max} ; Cost constraints PPA_{max} ; Ranked encryption methods E ; Computation threads T

Ensure: Encrypted circuit C_E^{opt}

```
1:  $E_+ \leftarrow null \cup E$  // Encrypting with  $null$  does nothing
2: for  $i = 0, \dots, \lfloor \frac{|E_+|}{T} \rfloor$  do
3:    $D_C^{k_{min}} \leftarrow \text{encrypt}(C, \{E_i, E_{i+1}, \dots, E_{i+T-1}\}, k_{min})$  // Done in parallel
4:    $PPA^{k_{min}} \leftarrow \text{get\_costs}(D_C^{k_{min}})$  // Done in parallel
5:   for  $C_{E_{max}}^{k_{min}} \in D_C^{k_{min}} \mid C_E^{k_{min}} \neq C$  do
6:     if  $PPA_{E_{max}}^{k_{min}} \leq PPA_{max}$  then
7:        $k_{left} = k_{min}, k_{right} = k_{max}$ 
8:       while  $k_{left} < k_{right}$  do
9:         if  $k_{right} \neq k_{max}$  then
10:           $k_{step} = \text{floor}(\frac{k_{right} - k_{left}}{T+1})$ 
11:           $kvals = \{k_{left} + k_{step}, \dots, k_{left} + Tk_{step}\}$ 
12:        else
13:           $k_{step} = \text{floor}(\frac{k_{max} - k_{min}}{T})$ 
14:           $kvals = \{k_i \in \{k_{min} + k_{step}, \dots, k_{min} + (T-1)k_{step}, k_{max}\} \mid k_i > k_{min}\}$ 
15:        end if
16:         $D_C^{kvals} \leftarrow \text{encrypt}(C, E_{max}, kvals)$  // Done in parallel
17:         $PPA^{kvals} \leftarrow \text{get\_costs}(D_C^{kvals})$  // Done in parallel
18:         $kvals_{left} = \{k_i \in kvals \mid PPA_i^{kvals} \leq PPA_{max}\}$ 
19:         $kvals_{right} = \{k_i \in kvals \mid PPA_i^{kvals} > PPA_{max}\}$ 
20:        if  $kvals_{left} \neq \emptyset$  then
21:           $k_{left} = \max(kvals_{left})$ 
22:        end if
23:        if  $kvals_{right} \neq \emptyset$  then
24:           $k_{right} = \min(kvals_{right}) - 1$ 
25:        end if
26:      end while
27:      return  $C_{E_{max}}^{k_{left}}$ 
28:    end if
29:  end for
30: end for
31: return failure
```

successful execution of the algorithm. The algorithm may not succeed if the PPA constraint is too tight for the set of encryption methods and key size range used, in which case $D_C^{opt} = \emptyset$.

This algorithm has the same three basic steps as the binary search-based algorithm. First, the PPA of C is estimated. Concurrently, as described in Section 3.4.1, $T - 1$ encrypted designs at key size k_{min} are produced, with encryption schemes starting at the top of the ranking. The PPA of these designs is estimated using Design Compiler and IC Compiler, and checked for breaking PPA_{max} . The design with the highest ranked encryption scheme that does not break PPA_{max} is used for the next step. If all of the encrypted designs break PPA_{max} , then the next T encryption schemes are attempted at key size k_{min} . If all possible encryption schemes break PPA_{max} at k_{min} , then the algorithm returns a failure, because no design in D_C^S meets the PPA constraints.

If one or more encryption schemes are found to be within PPA_{max} , then the strongest in the ranking among them, E_{max} is chosen for the final step, the concurrent tree search of the key space of potential key sizes. In the first loop of the search, the range of key sizes between and including k_{min} and k_{max} is divided into T sections. The set of key sizes, $kvals$ at the end of each division, including k_{max} but not k_{min} , are selected for encryption. C is encrypted using E_{max} and the selected key sizes, producing T encrypted designs, for which the PPA of each design is estimated. For all $k_i \in kvals$ that meet the PPA constraints, the maximum is used as the new minimum of the search space (k_{left}), unless no key sizes met the PPA constraints in which case the minimum of the search space remains the same. For all $k_i \in kvals$ that do not meet the PPA constraints, one less than minimum is used as the new maximum of the search space (k_{right}). If all key sizes met the PPA constraints, then the maximum remains the same. The loop breaks once the minimum of the search space meets the maximum. So, in the first loop, where k_{max} as a part of the tested $kvals$, if k_{max} meets the constraints, then $k_{left} = k_{right} = k_{max}$, so the loop terminates and returns the design with a key size of k_{max} . This way, if the entire key space is within PPA_{max} , then the algorithm will return k_{max} in one iteration of the key size search.

After the first loop, if k_{max} does not meet the PPA constraints, the key size space to be explored shrinks to between the new k_{left} and k_{right} . In iterations after the first, $kvals$ no longer includes k_{right} , so the search space is divided into $T + 1$ sections, and C is encrypted with E_{max} and the key sizes at division points, not including k_{left} and k_{right} (so that the number of encrypted designs created is always T). The PPA of each encrypted design is estimated. The space of potential key sizes shrinks to the set between the maximum key size that met the PPA constraint ($\max(kvals_{left})$) and before, but not including, the minimum key size that did not meet the PPA constraint ($\min(kvals_{right})$). This explores the design space iteratively by ruling out key sizes between sampled ones that fail, accepting those between ones that succeed as a part of D_C^{opt} , and exploring the rest of the design space in the next iteration. This continues until the unexplored design space shrinks to nothing, and the design with the maximum rated encryption scheme with the maximum key size that does not violate PPA_{max} is returned as C_E^{opt} . Because the design space is guaranteed to shrink each iteration, the algorithm is guaranteed to converge on a single C_E^{opt} if an encryption method, E_{max} can be found. The specific C_E^{opt} found, however, may be inaccurate, based on the margin of error of ICC PPA estimation for each design which it is performed.

Since the first and second step of the algorithm are executed concurrently, their worst-case execution time can be combined. Similar to Section 3.3, it will be assumed that the execution time of PPA collection steps is equivalent among versions of C . Additionally, it will be assumed that the execution time of estimating PPA with ICC eclipses that of DC ($t_{ICC} \gg t_{DC}$), so t_{ICC} will be used as the worst case timing for PPA estimation. For the estimation of PPA for C and C_E^i at k_{min} , the worst case execution time is the number of

PPA calculation "batches" necessary for C and to explore the entire ranking, $t_{ICC} \frac{1+|E|}{T}$. In the concurrent tree key size search, the binary search problem is sped up with concurrent sampling points. This converts the time complexity from $O(\log_2(k_{range}))$ to $O(\log_{T+1}(k_{range}))$ for T concurrent threads and k_{range} potential key sizes. This results in a worst-case execution time of $\log_{T+1}(k_{max} - k_{min})$ for the key size search step of the algorithm. The overall worst case execution time is therefore $t_{exec} = t_{ICC}(\frac{1+|E|}{T} + \log_{T+1}(k_{max} - k_{min}))$.

The execution time of this algorithm is a direct improvement over that of the binary search-based algorithm. At $T = 1$ thread, the algorithm becomes equivalent to the binary search algorithm, both functionally and by execution time, with a worst case of $t_{exec} = t_{ICC}(1 + |E| T + \log_2(k_{max} - k_{min}))$ (the same that was found in Section 3.3). Additionally, the execution time of the first two steps of Algorithm 2 is $O(\frac{1}{T})$ with T parallel threads, and of the final step is $O(\frac{1}{\log T})$, both of which strictly decrease as the number of threads increases (again, $T \geq 1$). Therefore, not only is this method a direct improvement of the binary search-based method, but it is a superset of that method, in which the binary search method is the worst version.

3.5 Circuit Designs for Experimentation

To test the CDLE concurrent tree search algorithm, a group of 21 test circuit designs were selected from the Opencores IP core database [14] and ITC '99. benchmark circuit suite [2]. These designs were chosen as examples of modern IP cores that showcase the possible applications of logic encryption, in line with the goal of technology transition. They were also chosen for their range of post-logic synthesis gate counts. All designs were synthesized using a 90nm technology node. Some small designs were selected for proof-of-concept, and then larger designs will show the limits of execution time and PPA estimations for designs with gate counts over 100,000. See Table 3.3 for more information for each benchmark circuit.

3.6 Concurrent Tree Search Experimentation

3.6.1 Experimental Setup

All experimentation was performed using an AMD Ryzen Threadripper 1950X 3.4GHz 16 core (32 thread) processor and 64GB RAM. The number of threads utilized in each experiment was controlled to 8 ($T = 8$), unless otherwise specified. Circuits were encrypted using the compound encryption schemes mentioned in Section 3.4.2, with a minimum key size $k_{min} = 32$, and a maximum key size $k_{max} = 512$. For all circuit simulation, 5000 input patterns were used between those from automated test pattern generation and additional randomly generated test patterns. All synthesis during experimentation used a 90nm technology node.

Table 3.3: Opencores[14] and ITC’99 [2] circuits chosen for CDLE experimentation

CIRCUIT	GATE COUNT	FUNCTION
Stepper Motor	193	Module for controllong 4 or 6 wire stepper motor
SS_PCM	427	PCM Interface
USB Phy	507	USB 1.1
sasc	647	Simple asynchronous serial controller which includes 4 byte receive and a 4 byte transmit FIFO and external baud rate generator
Simple SPI	847	Enhanced version of SPI with a wider operating frequency range, 4deep read and write fifo and 8 bit wishbone interface
caprng	913	Generates complex pseudo random numbers
Hilbert Tranformer	1630	Approximates the hilbert transform with a digital filter
systemcdes	2096	Implementation of DES algorithm in System C for low power applications
des_area	2698	Area optimised single DES IP core working in CBC mode (Cipher block chaining), sequential implemenation requiring 16 cycles to complete one encryption or decryption cycle
des3_area	3072	Area optimised triple DES IP core working in CBC mode (Cipher block chaining), sequential implemenation requiring 48 cycles to complete one encryption or decryption cycle
TV80	6221	8 bit Z80 compatible microprocessor core
ac97_ctrl	12121	AC97 Controller core. It provides an interface to external AC97 audio Codec
USB Func	12984	USB 1.1 slave/device IP core
aes_cipher	17153	128 bit AES encryption algorithm
sha256core	17958	Implementation of SHA 256 hashing algorithm
des_perf	23430	Performance optimised single DES IP core, working in ECB mode. This is a pipelined architecture having 16 cycle pipeline. So it can perform encryption/decryption every cycle
aes_inv_cipher	27427	AES decryption
des3_perf	71991	Performance optimised triple DES IP core, working in ECB mode. This is a pipelined architecture having 48 cycle pipeline. So it can perform encryption/decryption every cycle
vga_lcd	124350	VGA/LCD Controller core is a wishbone revB.3 compliant embedded VGA core capable of driving CRT and LCD displays.
b19	174519	Cross-connected Viper and Intel 80386 microprocessors
cf_rca	196946	A platform for dynamic reconfigurable computing.

The concurrent tree search algorithm was developed for this work in the Go programming language [7] as a part of the Constrained and Obfuscated Design-Space Exploration for Security (CODES) platform. CODES provides many tools and methods for encrypting designs, including this CDLE method, and carries out encryption scheme implementations as described in Chapter 2. Go includes a straightforward multithreading construct, *goroutines*, which were utilized for parallel design space exploration. For each time CDLE was

Table 3.4: Pre-encryption PPA estimations for each design tested

Design Name	Power (mW)	Performance (ns)	Area (mm ²)
steppermotor	0.073	0.13	0.006046618
ss_pcm	0.151	1.02	0.010418688
usb_phy	0.249	6.38	0.01327104
sasc	0.38	2.65	0.02261975
simple_spi	0.266	2.34	0.017593344
caprng	0.154	0.3	0.010225152
hilbert	0.607	1.86	0.03211223
systemcdes	1.02	25.71	0.050894438
des_area_opt	0.468	21.37	0.026114458
des3_area_opt	0.623	20.03	0.034268774

performed, the final encryption scheme and key sizes chosen for C_E^{opt} (if C_E^{opt} exists), the total execution time, PPA of the final and original designs as well as the PPA cost as a percentage increase from the original PPA, and the algorithm trajectory were collected. CDLE was executed for several sets of PPA constraints for each design. The chosen encryption strategies of all PPA constraint sets tested will be reported for all designs successfully encrypted, as well as which constraint sets resulted in failure of the tree search algorithm (i.e. the PPA constraints were too strict). One design, the Hilbert Transformer, was selected for more detailed execution time data collection. A set of CDLE tree search algorithm tests were selected as case studies for closer examination as a representative group in terms of encryption scheme chosen and algorithm trajectory. The case studies will include full algorithm trajectory and PPA results, as well as more detailed information about the designs being studied.

3.6.2 Results

The CDLE Concurrent Tree Search algorithm was attempted for all test designs in Table 3.3. However, for designs with gate counts over 5,000, execution time of the algorithm became prohibitively long. Therefore, a full results set was obtained for the 10 smallest designs of the set. This already shows one limitation of the algorithm already, which will be discussed in Section 3.7.

First the PPA of each design tested was estimated before encryption, which are reported in Table 3.4. The PPA of the pre-encrypted design is always calculated in the first step of the algorithm. Therefore, these values are not used to calculate the PPA cost during each algorithm execution. Rather, these values are presented as an estimated baseline for each design for the reader.

The concurrent tree search algorithm was executed for 5 sets of PPA constraints for each test design.

Table 3.5: Obfuscation strategies chosen by CDLE concurrent tree search for various PPA constraints. A ”-” denotes failure of the algorithm to find an optimally encrypted design

Design Name	50%	25%	10%	5%	1%
steppermotor	CHL+SFL 149	CHL+SFL 67	-	-	-
ss_pcm	CHL+SRC 150	CHL+SRC 54	CHL+SRC 32	-	-
usb_phy	CHL+SFL 133	CHL+CYC 103	CHL+SRC 53	-	-
sasc	EFF+SFL 136	CHL+SFL 38	EFF+SRC 257	CHL+SRC 73	CHL+SRC 82
simple_spi	CHL+SAR 345	CHL+SAR 88	-	-	-
caprng	CHL+SRC 153	CHL+SRC 68	CHL+SRC 55	-	-
hilbert	CHL+SFL 502	CHL+SFL 266	CHL+SFL 88	CHL+SFL 37	-
systemcdes	CHL+SFL 512	CHL+SFL 512	CHL+SFL 444	CHL+SFL 255	EFF+SFL 309
des_area_opt	CHL+SFL 512	CHL+SFL 316	CHL+SFL 151	CHL+CYC 32	-
des3_area_opt	CHL+SFL 512	CHL+SFL 482	CHL+SFL 87	CHL+SRC 180	-

Constraints are represented as a percentage increase from the PPA of the original, pre-encrypted design. For each constraint set, the power, performance, and area were constrained to the same percentage increase. The set of constraints used in these experiments were 50%, 25%, 10%, 5%, and 1% for each power, performance, and area. The results of this experimentation are reported in Table 3.5. A ”-” in the table denotes failure of the algorithm to find a suitable encryption strategy (i.e. the PPA constraints were too tight). The compound encryption scheme and key size selected are reported for each experiment. The schemes correspond to the compound encryption schemes utilized, with the following codes for each encryption scheme:

- SFL: SFL
- SRC: SRC
- CYC: Cyclic
- AST: Anti-SAT
- SAR: SAR
- EFF: Encrypt Flip-Flop
- CHL: Chain

The most chosen encryption scheme was Chain+SFL, which is expected because the algorithm prioritizes encryption scheme over key size for security. For sequential encryption schemes, Encrypt Flip-Flop was rarely selected over Chain. In most cases, less secure encryption schemes were used for tighter PPA constraints. However, there are some notable exceptions. For usb_phy, SRC was chosen at 10% PPA, while Cyclic was chosen at 25%. Also, for SASC, EFF was chosen for Chain at 50%, but not for 25%, and the key size increases from 5% and 1% PPA. These inconsistencies could be due to some randomness in the power estimation based on the test patterns chosen during the test pattern generation step of the PPA estimation strategy. Otherwise, the ranking behaves as expected. SFL was the most

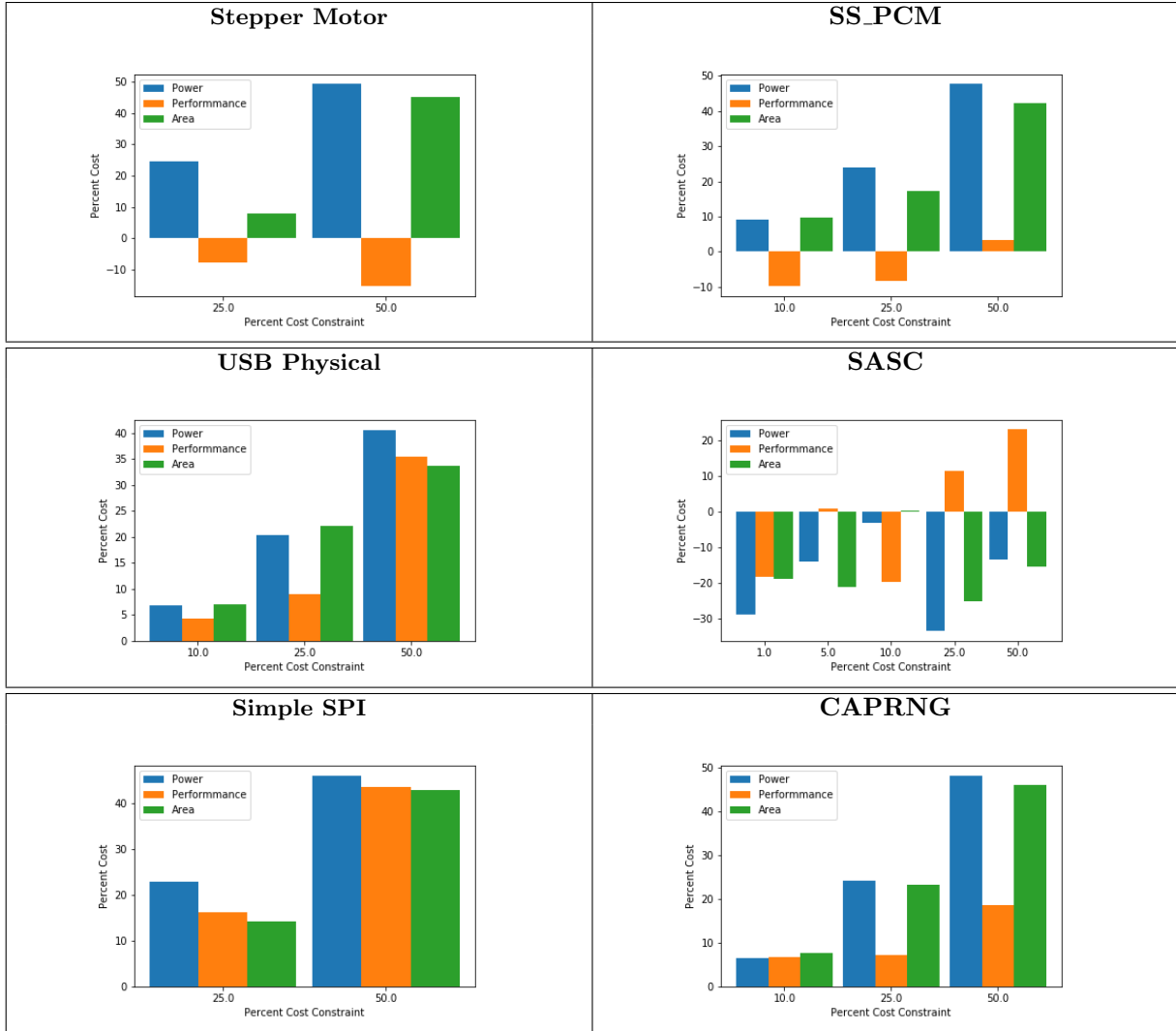


Figure 3.2: PPA cost of each design for successful CDLE encryption (1/2)

popular commbination scheme, and AntiSAT was never used (likely, SARLock was always the better option at similar cost). Also, As the gate count of a design increases, the more likely it is to meet stricter PPA requirements. There are exceptions though, especially SASC.

In Figures 3.2 and 3.3, the PPA costs of each final encrypted design for all CDLE experiments for which one was produced are reported. There are some cases in which the PPA was estimated to be an improvement compared to the original design. This can occur if added encryption gates allow logic synthesis and layout placement and routing to introduce new optimizations to lower costs. These cases are reported as negative costs, and are allowed by the algorithm. In most cases, area and power were the limiting factor for the final design. However, in the case of SASC and systemcdes, performance was the limiting factor in several cases, and a lot of improvements in cost were able to be made for each PPA constraint set. Therefore, the ability

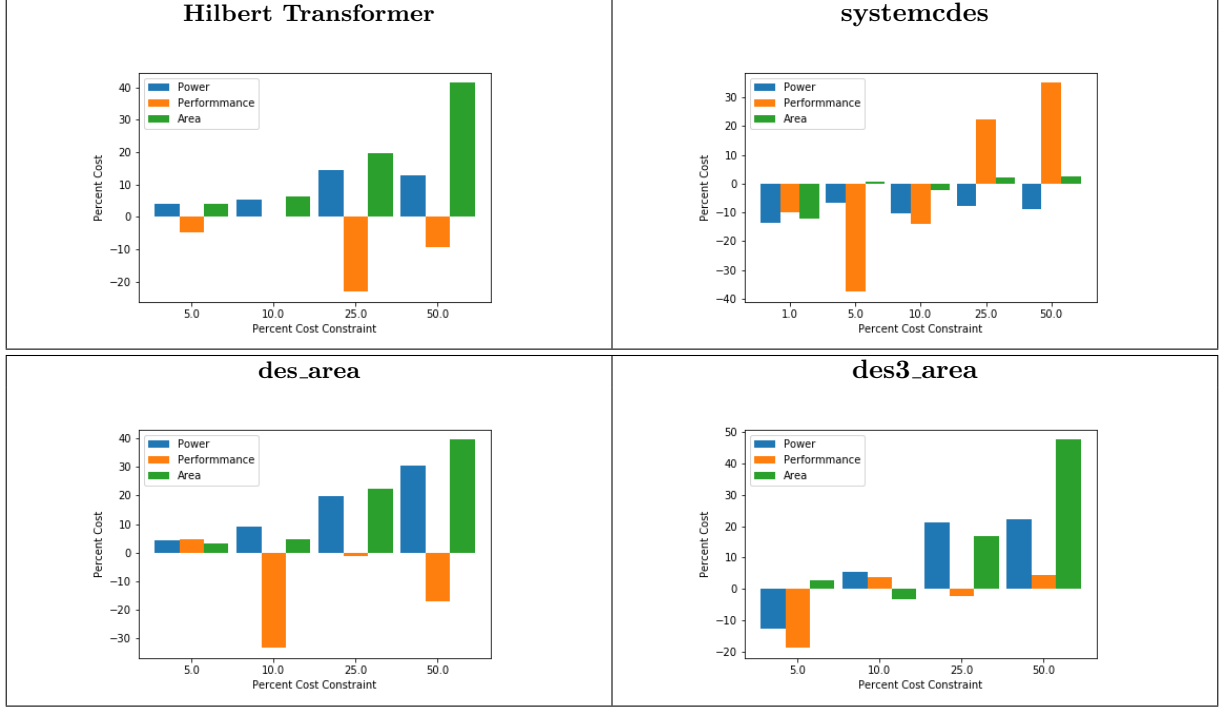


Figure 3.3: PPA cost of each design for successful CDLE encryption (2/2)

to make improvements may be dependent primarily on the design structure.

In Figure 3.4, the execution times of each CDLE experiment for which an encrypted design is found are reported vs the pre-encryption gate count of each design. This appears to be a roughly linear relationship, with about 1.55s of execution time added per logic gate. This is likely related to the direct proportionality found between t_{exec} and t_{ICC} . It also appears that the spread of execution time tends to increase with gate count. Additionally, the best-case time levels off for larger designs. The best-case scenario for execution time is the case in which the best possible encryption scheme and key size are chosen. Since this involves less steps than the worst-case, and less layouts are produced, the result is a smaller increase in execution time vs gate count.

In addition to the CDLE experiments performed with 8 concurrent threads, additional experiments were performed for various thread counts for two selected designs, the Hilbert Transformer and Simple SPI. Data points at 150%, 100%, 50%, 25%, and 10% PPA constraints were collected, for thread counts of 1, 2, 4, and 8 each. The execution time of each of these experiments are plotted in Figures 3.5 and 3.6. For each, the second plot contains the curve representing the predicted worse-case execution time at each thread count. For the worst case curve, the longest ICC execution time among all of the plotted experiments was used as t_{ICC} . As expected, all of these experiments executed in less time than the predicted worst case. In fact, these experiments did not tend to approach the worst case execution time at all, with very little difference

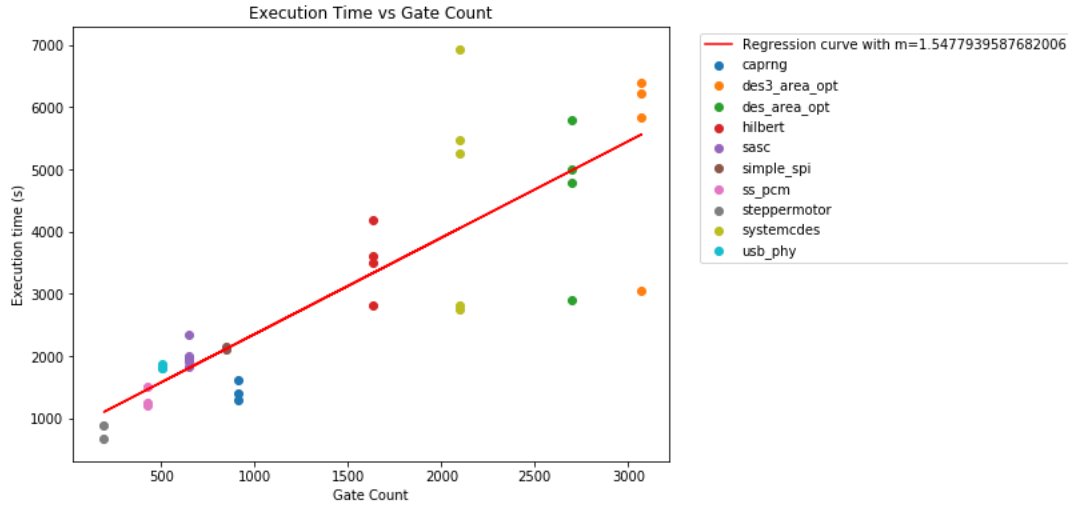


Figure 3.4: Execution time in seconds of each successful CDLE execution vs pre-encryption gate count

between them compared to the worst case, especially for the Simple SPI design. In general, the execution times do follow the same curve shape as the worst case, with some exceptions such as simple SPI at 25% constraints. The spread of execution time also appears to decrease with increased thread count.

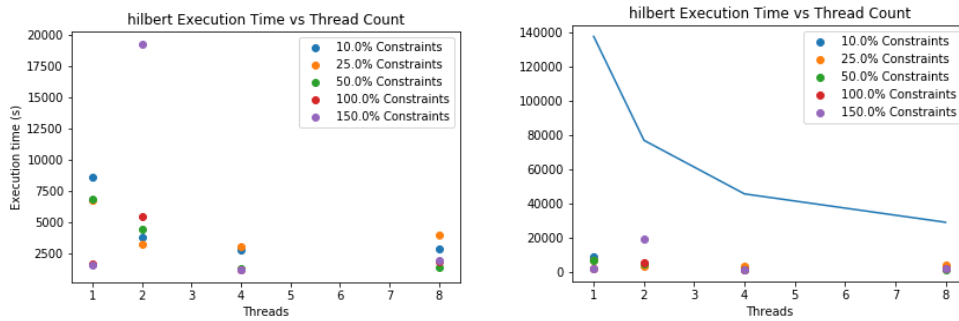


Figure 3.5: Execution time in seconds vs thread count of CDLE encryption of Hilbert Transformer

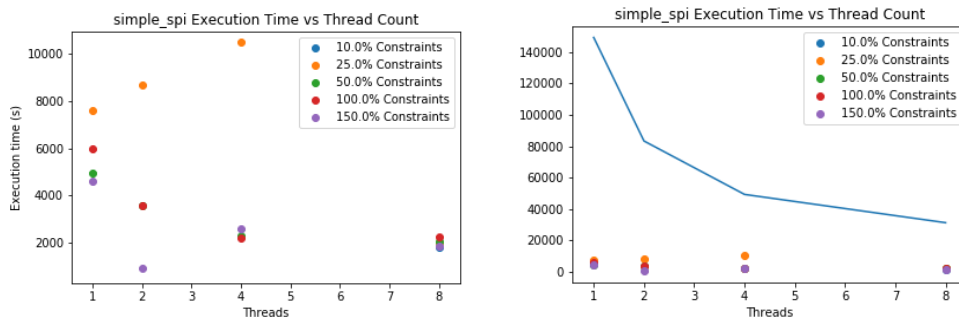


Figure 3.6: Execution time in seconds vs thread count of CDLE encryption of Simple SPI

3.6.3 Case Study: Area Optimized 3-DES

Several individual CDLE experiments have been selected as brief case studies that highlight different behaviors and trajectories of the algorithm and edge cases. One case is presented in this section, and the remaining case studies are in Section A.1 of the Appendix. The execution trajectory of the algorithm is represented as a series of bars, from top to bottom. In the first step, encryption schemes are tried in parallel at k_{min} until the best within the PPA constraints is found. In the remaining steps, the key size space is explored until the maximum allowed key size is found. The ticked key sizes between the maximum and minimum of the remaining exploration space are the sampled key sizes at each step, each tried in parallel. The maximum and minimum key sizes in each step are not sampled, except k_{max} , which is always sampled in the second step. The green "constraints met" area is determined by the maximum sampled key size which met the PPA constraints. The smallest key size that violated the PPA constraints determines the "constraints violated" area. The area between is unexplored, and determines the key size space of the next step. In the final step, the best key size is determined.

The area optimized 3-DES is the largest design among the set of designs used for this set of experiments. The algorithm trajectory is shown in Figure 3.7. This example shows that even when tight PPA requirements

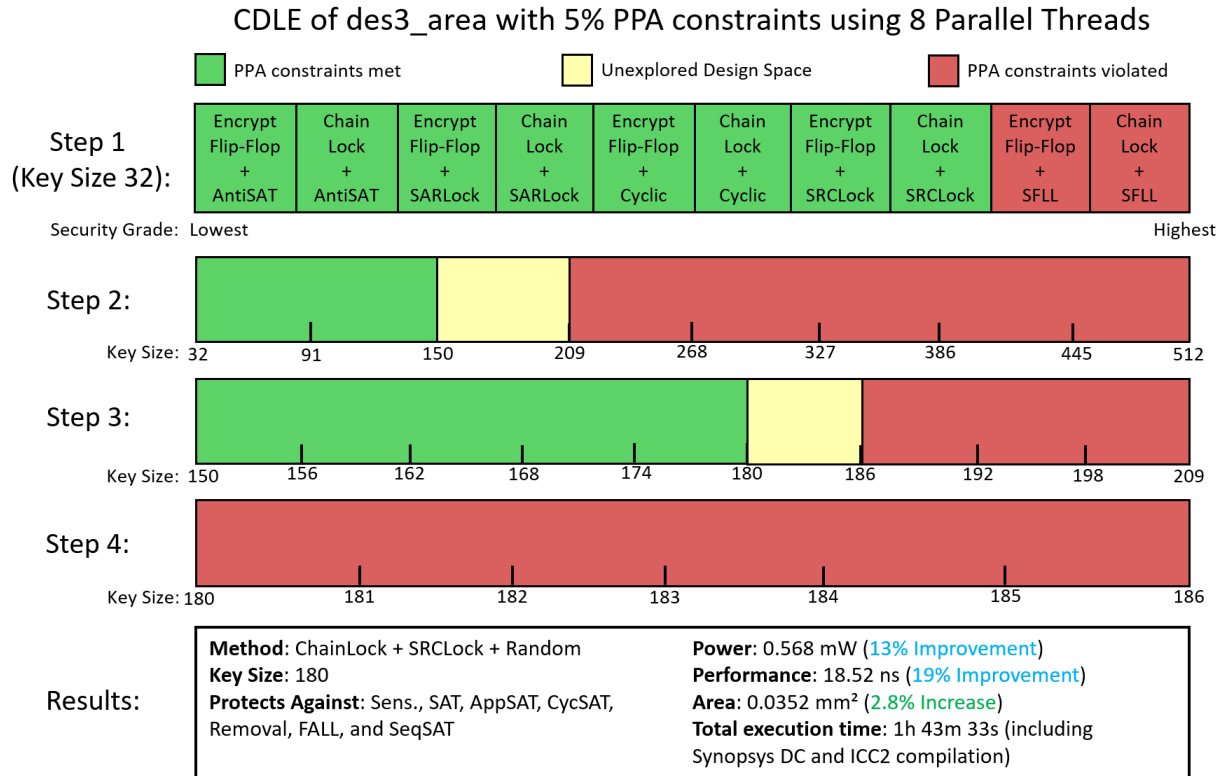


Figure 3.7: CDLE Trajectory and result of des3_area with PPA constrained to 5% increases

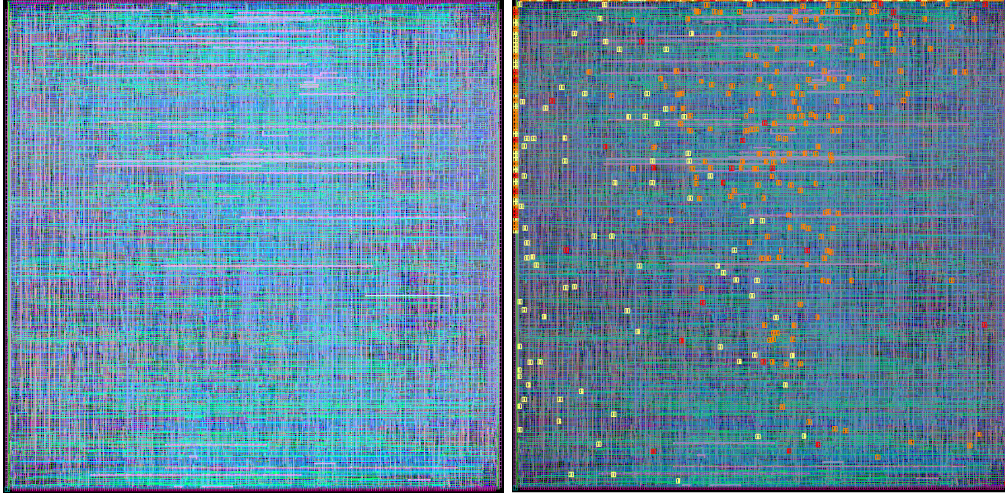


Figure 3.8: Layout of encrypted des3_area. On right, the key inputs and their respective key gates are highlighted, yellow corresponding to ChainLock, orange to SRCLock, and red to randomly placed gates.

are used on larger designs, a secure encryption scheme can be chosen due to the low impact of logic encryption relative to the size of the design. Otherwise, the algorithm trajectory is normal. The chosen encryption scheme was ChainLock+SRCLock at key size 180. In the final step, no key sizes were within the PPA constraints, so the maximum key size of the previous step within the constraints was chosen. The layout produced by ICC2 for the final encrypted design is in Figure 3.8. The algorithm executed in 1h 43m 33s. The execution time of CDLE quickly began to increase for designs with logic gate counts of more than a few thousand, including this design. The PPA values estimated during this case were:

- Original Power: 0.649 mW
- Final Power: 0.568 mW (13% Improvement)
- Original Performance: 22.78 ns
- Final Performance: 18.52 ns (19% Improvement)
- Original Area: 0.0342 mm²
- Final Area: 0.0352 mm² (2.8% Increase)

3.7 Concurrent Tree Search Using DC

3.7.1 Motivation

As hypothesized in Section 3.4 and shown in Section 3.6, the execution time of the CDLE concurrent tree search algorithm grows linearly with ICC execution time, as seen in Figure 3.4. For designs with gate counts larger than even 1,000, execution time grew to several hours, and designs with gate counts over 10,000 were

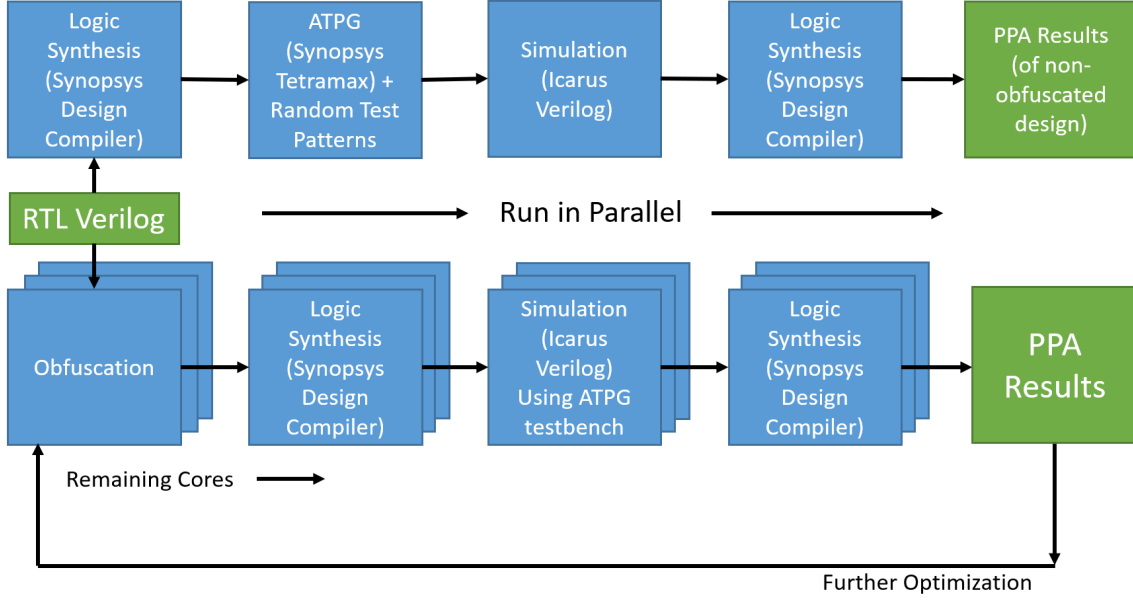


Figure 3.9: Modified PPA collection strategy without IC Compiler II

skipped in experimentation due to execution time. Furthermore, Figures 3.5 and 3.6 show the predicted behavior of increasing thread count on execution time. That is, increasing the thread count has diminishing returns in that regard for the concurrent tree search algorithms. Therefore, simply increasing the number of threads for execution is not a viable strategy for handling larger designs than 10,000 gates.

The best strategy for decreasing the worst-case execution time, $t_{exec} = t_{ICC}(\frac{1+|E|}{T} + \log_{T+1}(k_{max} - k_{min}))$, should be to decrease t_{ICC} . Estimating PPA with IC Compiler II is, in general, very accurate to that of the final packaged IC. However, producing a layout for every design explored in D_C before reaching C_E^{opt} is not practical for large designs. As discussed earlier in Section 3.3, $t_{ICC} \gg t_{DC}$. Logic synthesis executes much more quickly for any given design, and produces PPA estimations based on the resulting netlist. Therefore, to save on execution time, a modified PPA estimation flow is proposed in Figure 3.9. This flow removes PPA estimation using IC Compiler, as well as the simulation step that informed that estimation.

Effectively, for the new worst case execution time, t_{exec}^{DC} , t_{ICC} is replaced with t_{DC} . It follows that, since $t_{exec}^{ICC} \propto t_{ICC}$ and $t_{ICC} \gg t_{DC}$, that $t_{exec}^{ICC} \gg t_{exec}^{DC}$. The caveat to basing the algorithm on DC PPA estimation, rather than ICC, is increased inaccuracy of the estimation. During place and route, IC Compiler makes optimizations to the design and routing that change (especially reduce) power usage and area, and improve performance. Therefore, it is expected that improved execution time will be exchanged with quality of results from this CDLE algorithm. However, this opens the possibility of using CDLE with larger designs. In this section, the modified CDLE using only DC will be explored. Other than the PPA collection strategy, the algorithm and design exploration process is unchanged from the one laid out in Algorithm 2.

Table 3.6: Pre-encryption PPA estimations using the modified estimation strategy, for each design tested

Design Name	Power (mW)	Performance (ns)	Area (mm ²)
steppermotor	0.0545944	48.07	0.003718226
ss_pcm	0.1156321	34.64	0.0071144
usb_phy	0.1858693	75.58	0.011316732
sasc	0.158051	19.68	0.01038126
simple_spi	0.567131	45.04	0.015706944
caprng	0.2711458	79.34	0.008681535
hilbert	1.6166	26.49	0.029028885
systemcdes	0.3904485	103.43	0.026080892
des_area_opt	0.5329576	98.92	0.021435786
des3_area_opt	0.7274816	97.98	0.028041375
tv80	0.8435477	427.56	0.057884155
ac97_ctrl_top	2.7783	492.54	0.20399322
usb_funct	2.9954	714	0.185062582
aes_cipher	2.3238	1937.47	0.164356869
sha256core	6.5133	2175.8	0.377932456
des_perf_opt	5.8425	211.21	0.316213103
aes_inv_cipher	5.2658	2270.96	0.350300632
des3_perf_opt	18.75	668.98	1.204347393
vga_lcd	30.666	798.91	2.095086463
b19	18.236	19629.21	1.322780298
cfrc	43.561	8920.85	3.000304324

3.7.2 Results

The experimental setup of the concurrent tree search algorithm is equivalent to that of other experimentation for this algorithm (refer to Section 3.6.1). The power, performance, and area of each test design was measured again, using the new modified PPA estimation flow with only DC. The results are shown in Table 3.6. The CDLE concurrent tree search algorithm first estimates the PPA of the unencrypted design, so these estimations were not used for PPA cost during experimentation, but rather presented as a baseline for these designs, and also to compare to those of the estimation flow with ICC. In that regard, there are some very noticeable disparities. The power estimation averages 19% more from DC, and area 26% less, compared to the same unencrypted designs that were estimated with ICC. Performance, however, is much more egregious. The performance for unencrypted designs estimated with DC were, on average, 8.25 times slower than those with ICC. This large disparity shows the importance of estimating these post-layout when accuracy is desired. For CDLE, this may or may not be important, depending on if the disparity between

Table 3.7: Obfuscation strategies chosen by CDLE concurrent tree search without ICC for various PPA constraints. A '-' denotes failure of the algorithm to find an optimally encrypted design

Design Name	25%	10%	5%	1%	0.10%
steppermotor	CHL+SFL 55	-	-	-	-
ss_pcm	CHL+CYC 38	-	-	-	-
usb_phy	EFF+SFL 78	CHL+AST 45	-	-	-
sasc	CHL+SFL 33	-	-	-	-
simple_spi	EFF+SFL 37	-	-	-	-
caprng	CHL+SRC 85	CHL+SRC 32	-	-	-
hilbert	CHL+SFL 368	CHL+SFL 133	CHL+SAR 74	-	-
systemcdes	CHL+SFL 225	EFF+SFL 74	EFF+SAR 35	-	-
des_area_opt	CHL+SFL 275	-	-	-	-
des3_area_opt	CHL+SFL 368	CHL+SFL 85	CHL+SRC 34	-	-
tv80	CHL+SFL 512	CHL+SFL 90	-	-	-
ac97_ctrl_top	CHL+SFL 512	-	-	-	-
usb_funct	CHL+SFL 512	EFF+SFL 76	-	-	-
aes_cipher	CHL+SFL 512	CHL+SFL 512	CHL+SFL 313	-	-
sha256core	EFF+SFL 444	EFF+SRC 76	-	-	-
des_perf_opt	CHL+SFL 512	CHL+SFL 512	CHL+SFL 489	-	-
aes_inv_cipher	CHL+SFL 512	CHL+SFL 503	CHL+SFL 503	EFF+SAR 86	-
des3_perf_opt	CHL+SFL 512	CHL+SFL 512	CHL+SFL 512	CHL+SFL 35	CHL+AST 50
vga_lcd	CHL+SFL 512	CHL+SRC 254	-	-	-
b19	CHL+SFL 512	CHL+SFL 512	CHL+SFL 512	-	-
cfrca	CHL+SFL 512	CHL+SFL 512	CHL+SFL 512	CHL+CYC 60	-

DC and ICC is consistent within the scope of a design to be encrypted. If it is, then CDLE should produce the same results with or without ICC.

The modified design-space search algorithm using only DC estimation was performed for all test benchmarks in Table 3.3 for 5 sets of PPA constraints. For all constraint sets, the allowed power, performance, and area increase are represented by the allowed percent increase in each of these values compared to those of the original design. For each set of constraints, the same allowed increase was used for each power, performance, and area. For example, 5% denotes a 5% allowed increase in power usage, the same increase for critical path timing, and the same percentage increase for total area. All PPA estimations were collected from Design Compiler, including that of the original design against which the PPA of encrypted designs were compared. For these experiments, 25%, 10%, 5%, 1%, and 0.1% PPA constraints were used, which use smaller values than the ICC experimentation due to the inclusion of larger designs. The encryption strategies, including encryption schemes and key sizes chosen, for each design at each PPA constraint set are shown in Table 3.7.

A '-' denotes failure of the algorithm to find an optimally encrypted design (i.e. the PPA constraints are too strict to find a viable encryption strategy). The three-letter codes for encryption schemes used are the same as those used in Section 3.6.2. Additionally, like in previous encryption strategies as laid out in 3.4.2, 45% of key inputs are used for each listed encryption strategy, and 10% are reserved to drive randomly placed key gates.

The predominant encryption schemes chosen using the algorithm was ChainLock+SFLL. By design, the process will select the strongest encryption algorithm when possible, to protect against as many attacks as possible. There are several instances, especially for larger designs, where the strongest possible encryption strategy, ChainLock+SFLL at key size 512, was selected. This indicates situations in which the PPA constraints are loose enough to allow any encryption scheme in the security goal S_{opt} . As expected, tightening PPA constraints lowered the optimized security metrics, until an encrypted design within the PPA constraints could not be found.

In Figures 3.10, 3.11, and 3.12, the resulting PPA costs of the final, encrypted design for each test design are reported. They are measured as a percentage of the PPA of the original design, estimated at the beginning of each CDLE experiment using Design Compiler. In some cases, either power, performance, or area were improved in the final design. This can happen if optimizations can be made during logic synthesis with the addition of logic encryption to improve the PPA of the resulting design. This can also be due to some randomness and inconsistency in estimating PPA.

For smaller designs in 3.10, such as the Hilbert transformer and systemcdes, PPA tended to not be improved by logic synthesis, except in a few cases. In general, the algorithm was not driven by any cost metric in particular, except systemcdes which appears to have been limited primarily by power cost. Significant improvements in performance start to happen for the USB Functional circuit, in which performance was improved for both successful CDLE experiments. The SHA256 and performance-optimized DES cores, on the other hand, were limited by performance costs exclusively for all cases. For larger designs, performance appeared to either play a large role, either being vastly improved (as in cf_rca), or completely limiting (as in b19).

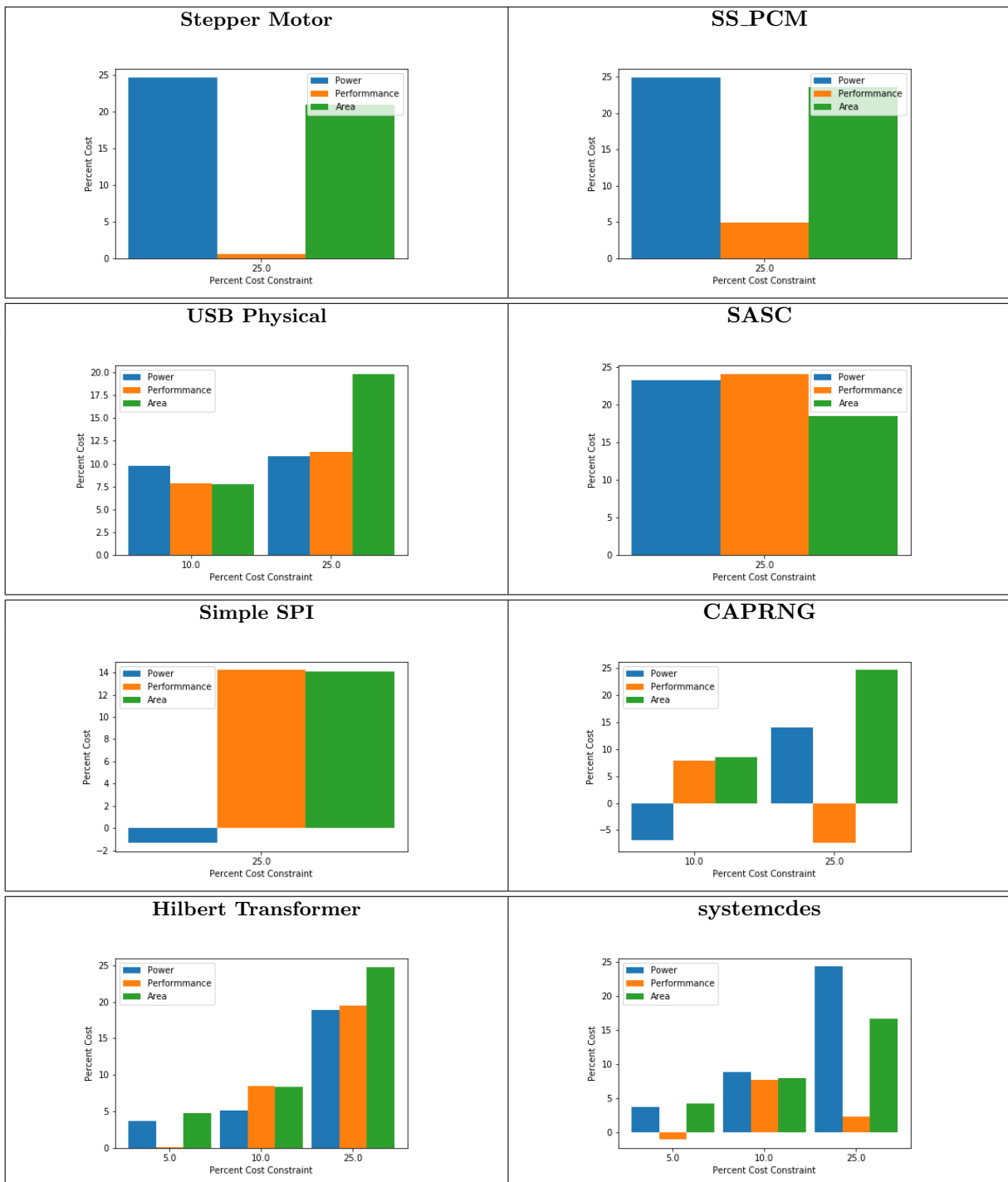


Figure 3.10: PPA cost of each design for successful CDLE encryption using only the DC flow (1/3)

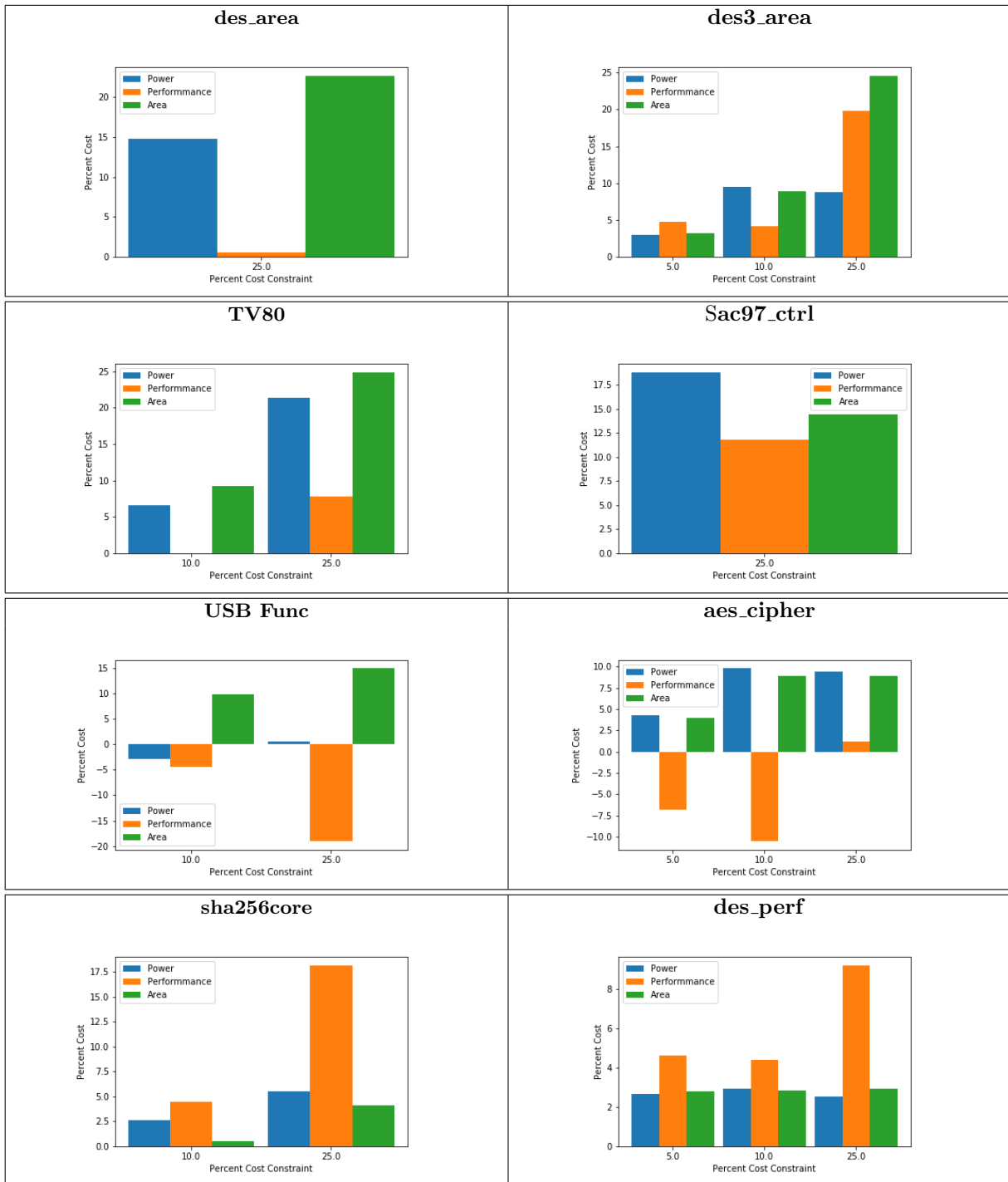


Figure 3.11: PPA cost of each design for successful CDLE encryption using only the DC flow (2/3)

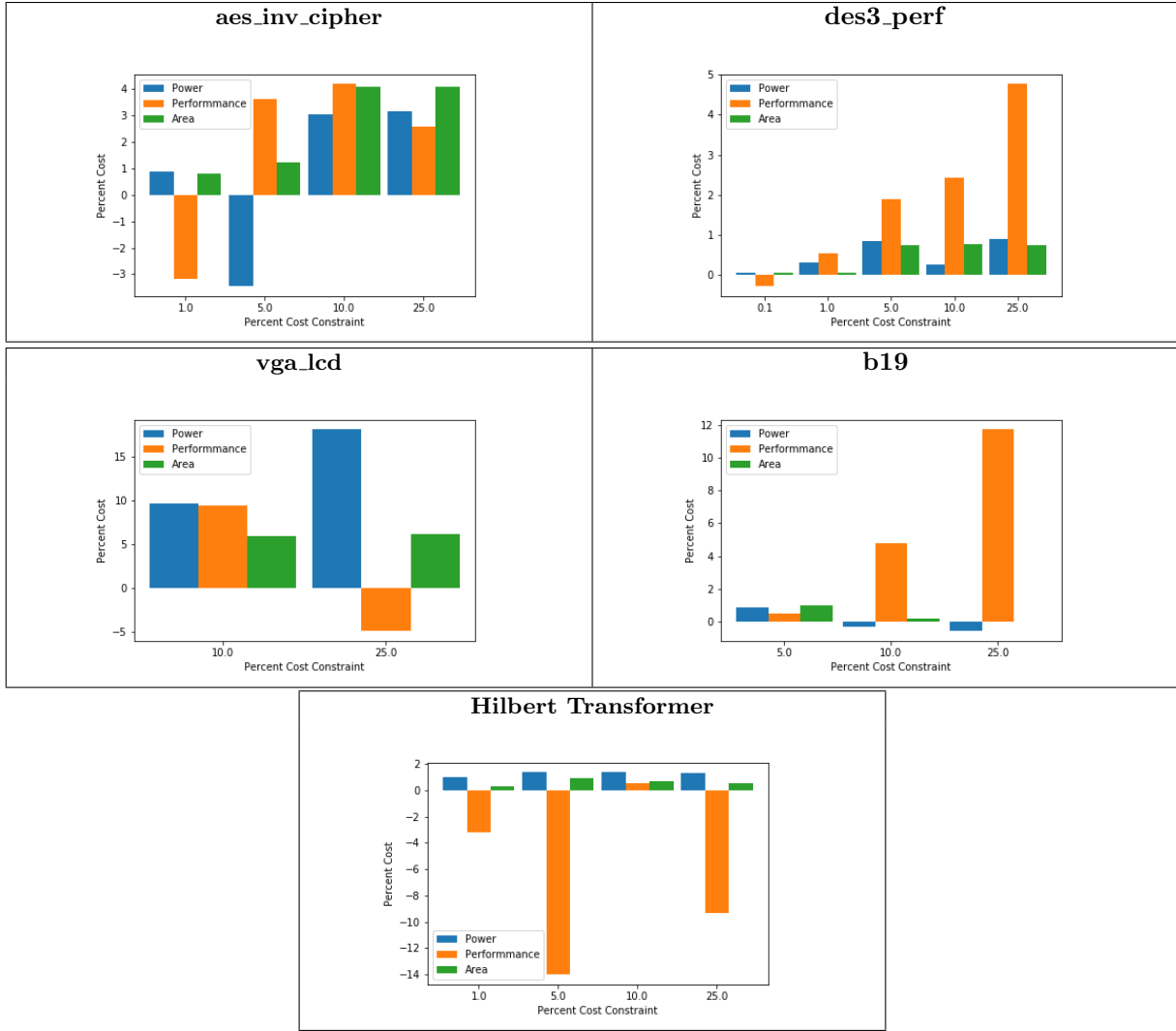


Figure 3.12: PPA cost of each design for successful CDLE encryption using only the DC flow (3/3)

The execution time of each successful CDLE experiment is plotted as a function of the gate count of the original design in Figure 3.13. These are the overall execution times of the algorithm, including each DC execution. The execution time of each PPA constraint for each design in Table 3.7 is reported and therefore each gate count has several data points associated with it. As expected, the overall execution time is much improved compared to that of the concurrent tree search using ICC, having an expected average execution time of about 0.124s per pre-encryption logic gate, compared to 1.55s with ICC. The largest designs, which contain close to 200,000 logic gates, have an execution time on the order of a few hours, which is comparable to that of 10,000 gate designs when ICC is used for PPA estimation. There appears to be a linear increase in average execution time for any given design for gate count. This is in line with the linear relationship between t_{DC} , the worst-case execution time of Design Compiler, and the worst-case overall execution time.

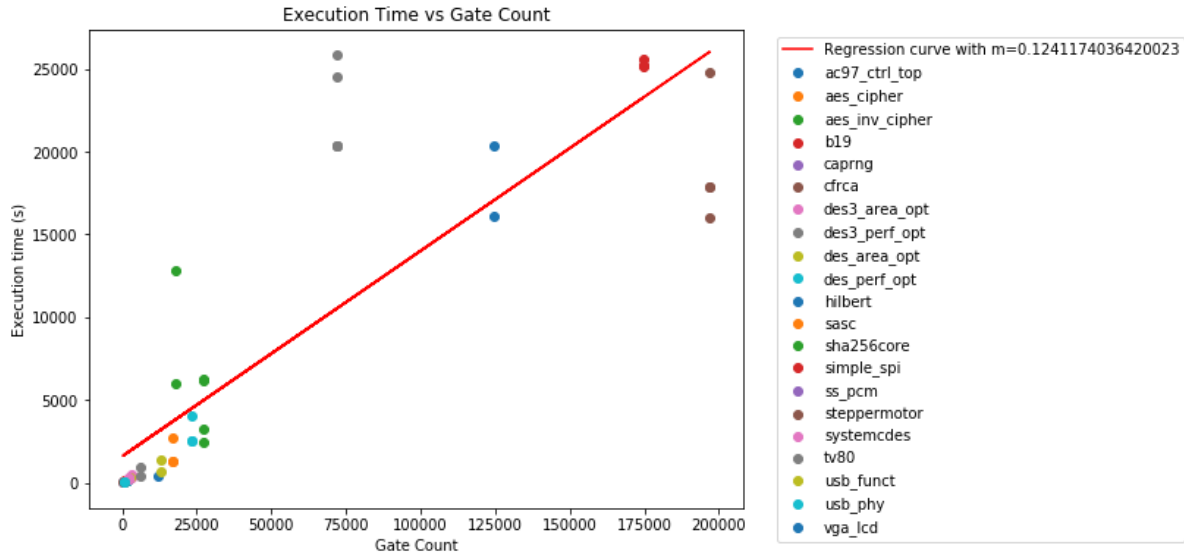


Figure 3.13: Overall execution time in seconds as a function of pre-encryption gate count for each CDLE design experiment

However, the maximum execution time for each design appears to level off just above 25,000 seconds. This could be due to the maximum algorithm execution time being a rare case for each experiment.

Similar to the experiments using ICC, execution time was also explored as a function of thread utilization in a separate set of experiments. The results of this experimentation are shown in Figures 3.14 - 3.18, which have both a plot containing only the data, and one that also contains the expected worst-case execution time for each design. The designs used for these experiments were the Hilbert Transformer, Simple SPI, SS_PCM, Stepper Motor, and USB Physical designs. However, PPA constraints of 150%, 100%, 50%, 25%, and 10% increases in each metric were used for all designs tested. Higher PPA constraints were used compared to other DC experimentation due to the selected designs being among the smallest in terms of pre-encryption gate count. The worst-case execution time in each case is plotted as well, using the worst case DC execution

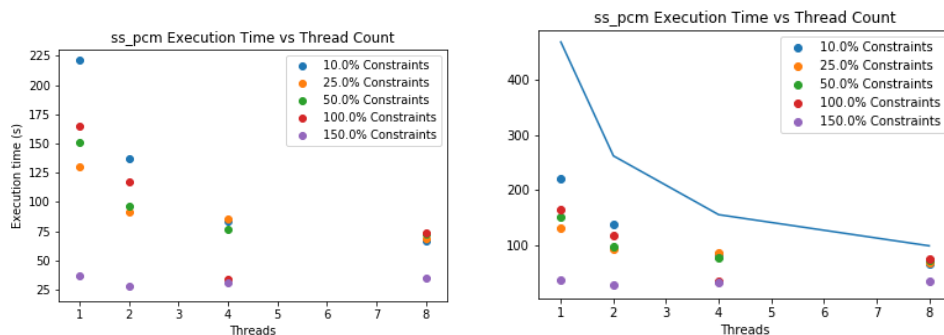


Figure 3.14: Execution time in seconds vs thread count of DC-Only CDLE encryption of ss_pcm

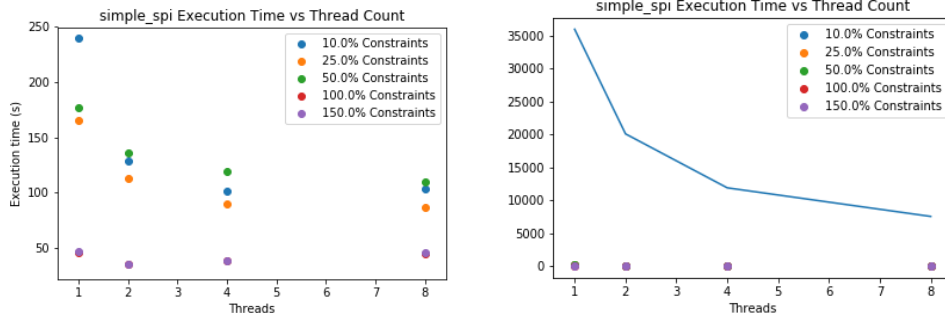


Figure 3.15: Execution time in seconds vs thread count of DC-Only CDLE encryption of Simple SPI

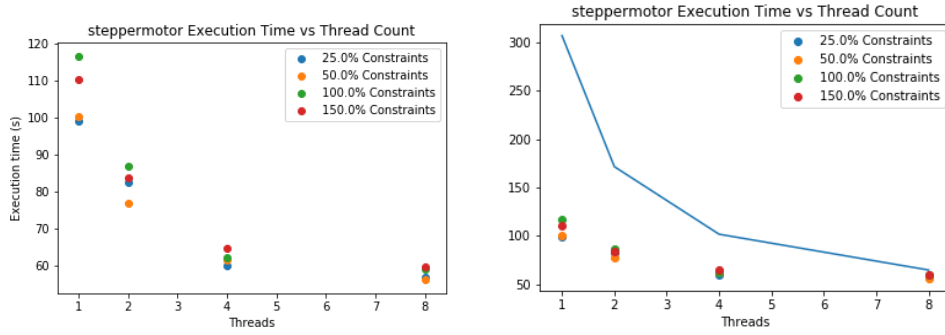


Figure 3.16: Execution time in seconds vs thread count of DC-Only CDLE encryption of Stepper Motor

time among these sets of experiments as t_{DC} for each design. As expected, in all cases, the overall execution time of each experiment was below the worst-case execution time limit, and tended to follow a similar shapes to their respective worst-case curves. Simple SPI and the Hilbert Transformer, however, had execution times well below the worst-case for all tested constraints.

The results of the USB Physical core show an interesting case. All of the data points lie below the worst-case curve, except for one experiment in which the maximum execution time was met for one 8-thread experiment. This data point corresponds to the 10% PPA constraint experiment, in which the resulting encryption strategy was ChainLock+Anti-SAT at key size 45. ChainLock+Anti-SAT is the second lowest ranked encryption scheme, as seen in the ranking in Section 3.4.2. With a total of 10 encryption schemes and 8 threads, this places the encryption scheme in the worst-case group in execution order in the second step of the algorithm. Also, at a key size of 45, the third step of the algorithm represented a full concurrent tree search. Therefore, this experiment represents the worst-case execution of the algorithm. This shows the accuracy of the predicted worst case execution time. Another interesting data point is the 100% PPA constraint experiment of the same design. In this case, an extra step needed to be added to the algorithm to fully explore the key size space. This made the trajectory approach the worst case execution time, but

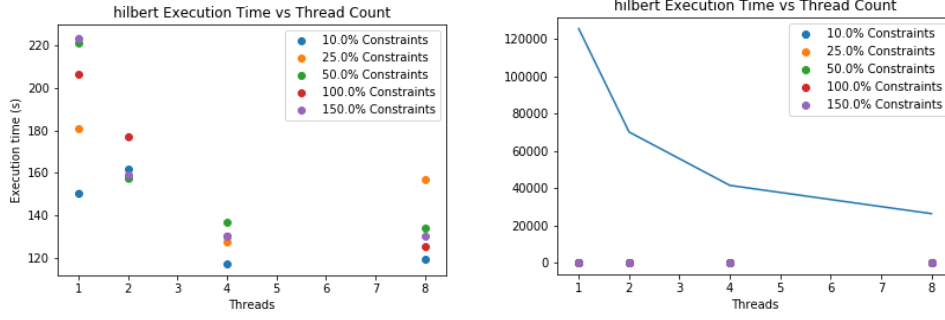


Figure 3.17: Execution time in seconds vs thread count of DC-Only CDLE encryption of Hilbert Transformer

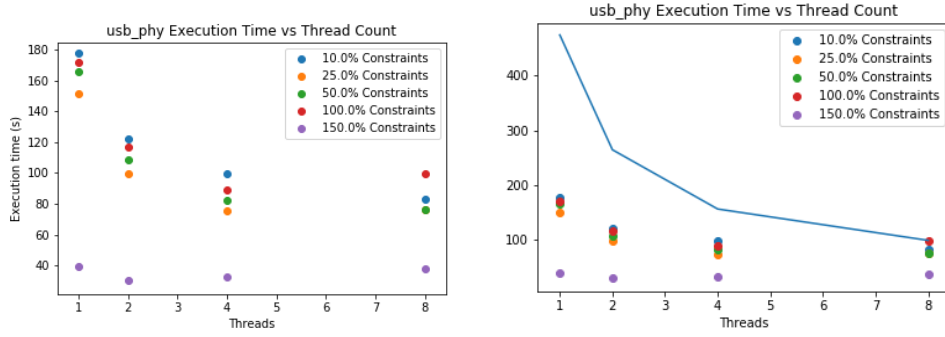


Figure 3.18: Execution time in seconds vs thread count of DC-Only CDLE encryption of USB Physical

not violate it.

3.7.3 Case Study: Area Optimized 3-DES

Similar to Section 3.6.3, brief case studies have been selected to highlight specific CDLE experiments using only DC for PPA estimation. Some studies have been selected as interesting cases, and some for comparison of the corresponding case study in Section 3.6.3. In these studies, a layout was produced with ICC2 after the completion of CDLE, and PPA estimations were made from the layout for comparison with the DC estimations. One case study is shown in this section, and other case studies are presented in Section A.2 of the Appendix. The figures are organized in the same way as in the previous case studies.

The area optimized 3-DES case was selected as a comparison between the concurrent tree search algorithm using ICC PPA estimation and DC. The algorithm trajectory is shown in Figure 3.19. In the corresponding case study in Section 3.6.3, ChainLock+SRClock with key size 180 was the optimal encryption strategy. In this case, the same encryption scheme was chosen, but at a smaller key size of 34. This result is slightly less sure, but will definitely be within the PPA constraints based on the ICC result. This case finished executing after 5m 57s, versus the 1h 43m 33s execution time of the same case with ICC.

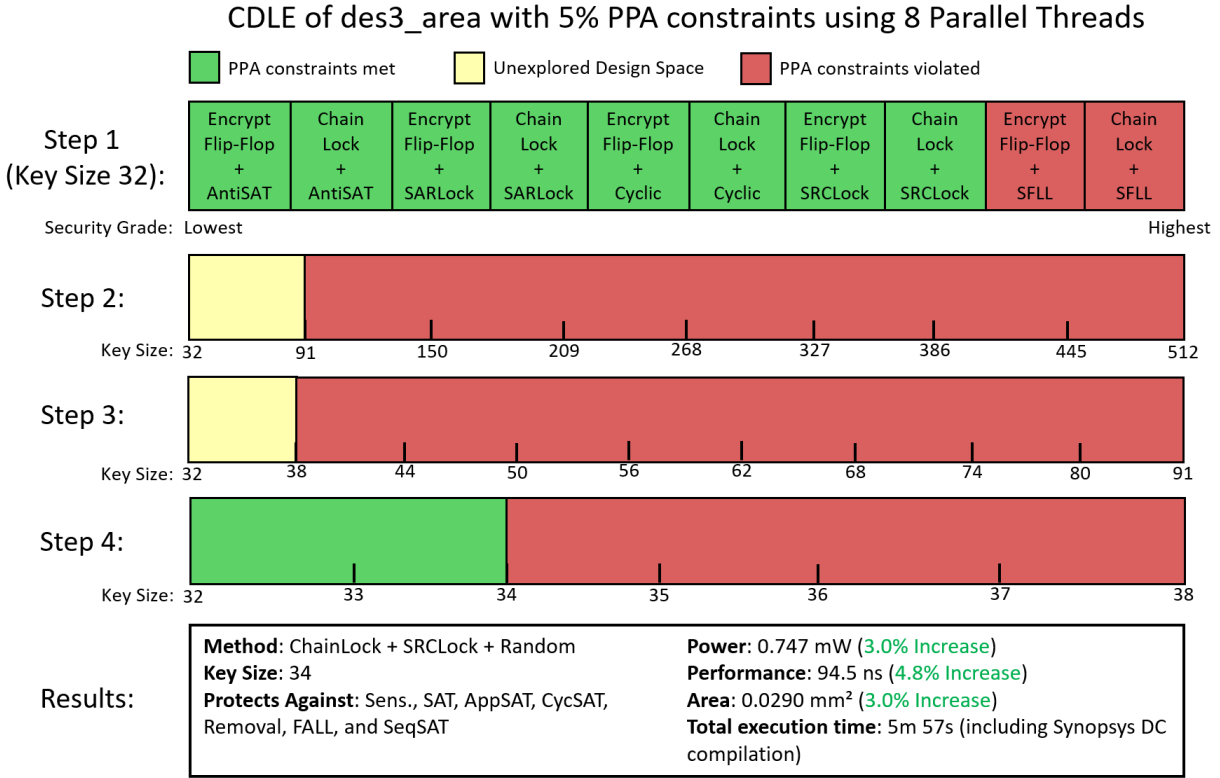


Figure 3.19: CDLE Trajectory and result of des3_area with PPA constrained to 5% increases

There was some disparity between the PPA values from ICC versus DC. For the unencrypted des3_area, the power estimation is 27% larger from DC compared to ICC, 296% slower in performance, and 18% smaller in area. This disparity, especially in performance, could degrade the quality of results from CDLE when using only DC to estimate PPA. The PPA estimated in this experiment were:

- Original Power: 0.725 mW
- Final Power: 0.747 mW (3.0% Increase)
- Original Performance: 90.2 ns
- Final Performance: 94.5 ns (4.8% Increase)
- Original Area: 0.0282 mm²
- Final Area: 0.0290 mm² (3.0% Increase)

The layout produced with ICC2 after CDLE is shown in Figure 3.20, with key gates and their respective driving key inputs highlighted on right. The design PPA was also estimated from this layout. The power usage was significantly lower, the performance drastically improved, and area usage was slightly greater compared to the DC estimation:

- Layout Power: 0.605 mw
- Layout Performance: 11.4 ns

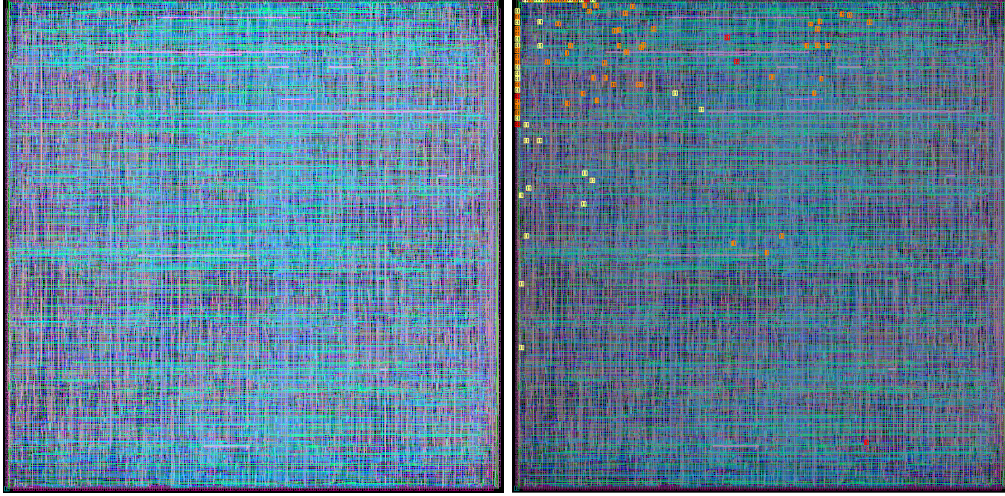


Figure 3.20: Layout of encrypted des3_area after CDLE with DC only. On right, the key inputs and their respective key gates are highlighted, yellow corresponding to ChainLock, orange to SRCLock, and red to randomly placed gates.

- Layout Area: 0.0309 mm²

3.7.4 Comparison with CDLE using ICC

Comparing the execution times of the concurrent tree search of CDLE using the full PPA estimation strategy with IC Compiler II, and the modified strategy using Design Compiler only, there is a clear advantage of using DC only for estimations over ICC. The execution time of the algorithm is greatly reduced in the former case, as anticipated. However, the PPA estimations from DC can differ quite a lot from those of ICC, as seen when comparing the PPA of unencrypted designs using DC in Table 3.6 to those when using ICC in Table 3.4. This was also seen in comparing the case studies of des3_area and systemcdes in Sections 3.7.3, 3.6.3, A.1, and A.2 between the two estimation strategies.

In Tables 3.8 - 3.11, the security strategy results of experimentation of CDLE using each estimation strategy are directly compared. All designs which were included in both sets of experimentation are included. The results for one set of PPA constraints is shown in each table, for all cases where at least one of the strategies returned a security strategy within the PPA constraints.

For all of the cases that can be compared, 67.5% of 40 experiments returned matching encryption schemes but different key sizes, which includes cases where no encryption strategy was found for either strategy and are not shown in the tables. Of the 25 cases shown, 48% returned matching encryption schemes. Five cases were able to return a result using ICC estimation, but no result for DC estimation. There are no cases in which DC returned a result, and ICC did not. There are also no cases in which both strategies returned the

exact same encryption scheme and key size. However, a couple cases are within 10 key size within each other, such as des3_area_opt with 10% constraints, SASC at 25%. When at least one strategy found an encryption strategy, estimation with ICC returned a higher security strategy than estimation with DC in 80% of cases either by scheme or key size, including those in which no strategy was found with DC.

3.8 Summary

The Constraint-Directed Methodology is proposed as a framework for logic encryption to be applied to a design in a way that is driven by both security and cost constraints. A new concurrent search method was introduced, and two versions of the method were tested using cost estimation with IC Compiler II and Design Compiler. In the next chapter, a different approach will be shown using machine learning.

Table 3.8: Comparison of CDLE results using ICC and DC PPA estimation with 25% PPA constraints

Design Name	ICC - 25%	DC - 25%
steppermotor	CHL+SFL 67	CHL+SFL 55
ss_pcm	CHL+SRC 54	CHL+CYC 38
usb_phy	CHL+CYC 103	EFF+SFL 78
sasc	CHL+SFL 38	CHL+SFL 33
simple_spi	CHL+SAR 88	EFF+SFL 37
caprng	CHL+SRC 68	CHL+SRC 85
hilbert	CHL+SFL 266	CHL+SFL 368
systemcdes	CHL+SFL 512	CHL+SFL 225
des_area_opt	CHL+SFL 316	CHL+SFL 275
des3_area_opt	CHL+SFL 482	CHL+SFL 368

Table 3.9: Comparison of CDLE results using ICC and DC PPA estimation with 10% PPA constraints

Design Name	ICC - 10%	DC - 10%
ss_pcm	CHL+SRC 32	-
usb_phy	CHL+SRC 53	CHL+AST 45
sasc	EFF+SRC 257	-
caprng	CHL+SRC 55	CHL+SRC 32
hilbert	CHL+SFL 88	CHL+SFL 133
systemcdes	CHL+SFL 444	EFF+SFL 74
des_area_opt	CHL+SFL 151	-
des3_area_opt	CHL+SFL 87	CHL+SFL 85

Table 3.10: Comparison of CDLE results using ICC and DC PPA estimation with 5% PPA constraints

Design Name	ICC - 5%	DC - 5%
sasc	CHL+SRC 73	-
hilbert	CHL+SFL 37	CHL+SAR 74
systemcdes	CHL+SFL 255	EFF+SAR 35
des_area_opt	CHL+CYC 32	-
des3_area_opt	CHL+SRC 180	CHL+SRC 34

Table 3.11: Comparison of CDLE results using ICC and DC PPA estimation with 1% PPA constraints

Design Name	ICC - 1%	DC - 1%
sasc	CHL+SRC 82	-
systemcdes	EFF+SFL 309	-

Chapter 4

Machine Learning Approach

4.1 Motivation

In Chapter 3, the Constraint-Directed Logic Encryption (CDLE) methodology was introduced as a strategy to maximize the security provided by known logic encryption methods, while also respecting cost constraints outlined in the design specification. The goal of this methodology is to outline a framework for many potential methods to explore the space of possible encrypted design in order to find an encrypted version of a circuit design to achieve maximum security while also meeting PPA goals. Potential methods that are a part of the methodology can execute this exploration in a variety of ways, each presenting a trade-off of metrics related to the methods themselves, such as execution time and accuracy of the results in terms of both security and cost.

The Concurrent Tree Search CDLE method, an improvement to the binary search design-space exploration algorithm proposed by Luria, et al. [13], was presented in Chapter 3 as well. This method was proposed as a way to speed up the search algorithm by sampling the search space concurrently with parallel computation threads. This was shown to introduce significant speedup compared to the binary search-based algorithm, while maintaining the same level of accuracy. However, the Concurrent Tree Search was limited in several ways.

First, the method in which the cost metrics, specifically power, performance, and area, of a design is estimated in this method utilizes Synopsys IC Compiler II (ICC) [31] as the primary estimation tool. PPA estimated from the layout produced by ICC are very accurate, however at the cost of very long execution time for large designs. Since the execution time of the algorithm is linearly proportional to the execution time of ICC for any design sampled, this can make the overall algorithm execution time prohibitive for large designs.

To mitigate this, a PPA estimation strategy relying on logic synthesis (with Synopsys Design Compiler [29]) instead of ICC, which executes much more quickly for any given design, was proposed. However, this came at the cost of more inaccurate PPA estimations and lower quality of results. The second limitation of the algorithm is diminishing returns on thread count. The worst-case execution time of the algorithm was shown to be $O(\frac{1}{\log N})$ with the number of parallel thread utilized. Therefore, simply increasing the number of threads was not a viable strategy for adequately decreasing the execution time for large designs.

The limitations of the Concurrent Tree Search CDLE algorithm show a need for CDLE methods that can accurately estimate the cost metrics of a design, without having an algorithm execution time that increases so heavily with the size of the design. Therefore, a machine learning approach to CDLE is proposed in this chapter. Utilizing machine learning to guide the encrypted design-space search can offload the execution time from the algorithm, and instead load it into a training period. If an adequate amount of high-accuracy training data can be acquired, and an accurate machine learning model devised from it, then a powerful and accurate search algorithm can be devised based on machine learning prediction. Such an approach can also drastically decrease execution time by removing heavy-handed cost metric estimation methods, such as simulation, netlist, and layout estimation, from the algorithm execution itself. These methods can still be used during the training loop, in order to collect high-quality cost estimations. This results in a potentially highly accurate and quick CDLE method.

In this chapter, a simple regression-based machine learning implementation of CDLE will be presented to introduce this approach to the methodology. A training set and training data will be outlined, and results of CDLE using the machine learning model will be shown.

4.2 Method Overview

The machine learning implementation used for CDLE in this work will involve two steps. The first is the training loop. In the training loop, several designs will be selected as a training set. The gate pre-encryption post-logic synthesis gate count of each design will be collected. Each design will be encrypted with the individual encryption schemes that make up the compound encryption schemes identified for implementation in Section 3.4.2. Therefore, each training design is encrypted with the encryption methods below individually, at key sizes ranging from 32 to 256 at steps of 32. Therefore, a prediction for each compound encryption scheme from Section 3.4.2 can be built at any key size between 32 and 512. The individual encryption schemes used in training are:

- SFL [39]
- SRClock [19]
- Cyclic [24]
- Anti-SAT [34]
- SARLock [36]
- Encrypt Flip-Flop [9]
- ChainLock [11]
- Random (EPIC [21])

The power, performance, and area of each encrypted design will be estimated using the full PPA estimation strategy from Section 3.4.1, using IC Compiler II estimations to build the machine learning model. Each PPA estimation will be compared to that of the unencrypted design, so each PPA data point will be a percent cost from the original design. The data from PPA estimation will be analyzed so an accurate regression of the data can be built. The regression model will use encryption scheme, key size, and design gate count as independent variables to predict each power, performance, and area cost of any encrypted design. It is expected that three submodels will be built for each PPA metric. Furthermore, due to the potential difficulty of quantifying the encryption scheme variable, a regression will be made for each encryption scheme, with key size and design gate count as independent variables. This will be done for each PPA submodel, so the total number of expected regressions created is $3 * |E|$, where E is the set of training encryption schemes. Additionally, a strategy for building predictions for compound encryption schemes with the model will be devised. This should provide a very accurate model with which to predict the PPA of designs within a design space of potential encrypted designs in the context of a CDLE flow.

The second step of the machine learning implementation is the experimentation step. In this step, the regression model built in the training loop will be utilized to make predictions for a set of experimental designs separate from the training set. The same designs that were used to experiment with the CDLE Concurrent Tree Search, identified in Section 3.5, will be used for this step. To search the design space, Algorithm 3, which is a modification of the binary search-based Algorithm 1 from Section 3.3, will be used. In this algorithm, the *encrypt* and *get_cost* functions are replaced with *predict_cost*, which uses the regression model to predict the PPA of an encrypted version of C based on the given encryption scheme and key size without needing to encrypt. The Concurrent Tree Search algorithm could also be used, but the main expected complexity of the experimentation is calculating the regression from training data, not in the execution of the algorithm itself. Therefore, the expected overall speedup from parallel threads is negligible.

One goal of the machine learning approach is for it to be extensible to designs outside of the training set, so the training set will not overlap with this experimentation step at all to test the generalization limits of the machine learning model. This experimentation step should give a look into the capability of the machine learning approach to CDLE, and ideas for improvements to the approach so it can become a robust solution

Algorithm 3 Constraint-Directed Logic Encryption Driven by Machine Learning

Require: Original Circuit C ; Key width bounds k_{min}, k_{max} ; Cost constraints PPA_{max} ; Ranked encryption methods E

Ensure: Encrypted circuit C_E^{opt}

```
1: for  $e \in E$  do
2:    $PPA_e^{k_{min}} \leftarrow predict\_cost(e, k_{min})$ 
3:   if  $PPA_e^{k_{min}} \leq PPA_{max}$  then
4:      $PPA_e^{k_{max}} \leftarrow predict\_cost(e, k_{max})$ 
5:     if  $PPA_e^{k_{max}} \leq PPA_{max}$  then
6:       return  $C_e^{k_{max}}$ 
7:     end if
8:      $k_{left} = k_{min}, k_{right} = k_{max}$ 
9:     while  $k_{left} \leq k_{right}$  do
10:       $k_{mid} = floor((k_{left} + k_{right})/2)$ 
11:       $PPA_e^{k_{mid}} \leftarrow predict\_cost(e, k_{mid})$ 
12:      if  $PPA_e^{k_{mid}} \leq PPA_{max}$  then
13:         $k_{left} = k_{mid}$ 
14:      else
15:         $k_{right} = k_{mid} - 1$ 
16:      end if
17:    end while
18:    return  $C_e^{k_{left}}$ 
19:  end if
20: end for
21: return failure
```

to the CDLE problem.

4.3 Training the Model

4.3.1 Training Set

A set of 22 designs from the Altera Synthesis Cookbook [3] were selected to train the CDLE machine learning model. There is no overlap between this training set and the designs identified in 3.5, used to perform experiments for the concurrent tree search-based algorithm. Instead, these designs will be used to perform experiments using the machine learning model. The training set designs, their post-logic synthesis gate count, and functions are listed in Table 4.1.

Several designs were generated by varying the input/output bus, and therefore the size of the overall circuit in terms of gate count. This was done to ensure a predictable spread of gate counts among the designs, to increase coverage of the potential use cases for the prediction. The gate counts of the overall set was restricted to below 30,000 to reduce training time.

There are two main limitations to this training set. One, the size of the training set is lacking. Ideally, training data for hundreds or even thousands of designs would be collected to ensure adequate coverage of

Table 4.1: Designs from the Altera Synthesis Cookbook [3] selected to train the CDLE machine learning model

CIRCUIT	GATE COUNT	FUNCTION
Approx FP Sqrt	92	Approximate square root calculator, accepts a floating point input
CRC32 DAT48	378	Cycle redundancy check for FPGA. Handles 48 -bit data
Stream MUX	382	Multiplexing unit for moving binary data streams from FPGA to a binary file
UART	383	UART communication interface
VGA	474	VGA monitor controller
Descrambler 64	683	Communication data de-scrambler, with 64-bit data in and out buses
Scrambler 64	699	Communication data scrambler, with 64-bit data in and out buses
Descrambler 128	1088	Communication data de-scrambler, with 128-bit data in and out buses
Scrambler 128	1211	Communication data scrambler, with 128-bit data in and out buses
Gearbox 32 33	1319	Adapts a 32-bit serial flow to a 33-bit word
Divider 64	1853	Arithmetic 64-bit unsigned integer divider
Descrambler 256	1888	Communication data de-scrambler, with 256-bit data in and out buses
Scrambler 256	2452	Communication data scrambler, with 256-bit data in and out buses
Descrambler 512	3493	Communication data de-scrambler, with 512-bit data in and out buses
Divider 128	3733	Arithmetic 128-bit unsigned integer divider
Divider 256	7788	Arithmetic 256-bit unsigned integer divider
Scrambler 512	7971	Communication data scrambler, with 512-bit data in and out buses
AES	10139	AES Encryption using 128 key and data widths
Descrambler 1024	13972	Communication data de-scrambler, with 1024-bit data in and out buses
Divider 512	14919	Arithmetic 512-bit unsigned integer divider
Scrambler 1024	19745	Communication data scrambler, with 1024-bit data in and out buses
Divider 1024	29543	Arithmetic 1024-bit unsigned integer divider

potential logic patterns and netlist constructions in the estimation. The second limitation is the size of the designs. Due to the large amount of data collection being attempted, the gate size was limited compared to the test set. The purpose of this exploration into a machine learning approach is to offer a possible alternative to the methods presented so far in this work. This machine learning model will serve as a limited example and impetus for machine learning in logic encryption.

4.3.2 Results

For each design in the training set, encryption was attempted using the base encryption schemes listed in Section 4.2, at key sizes between 32 and 256 with a step size of 32 (up to 64 total encrypted versions of each design). At times, encryption can fail, especially for small designs using encryption schemes that have a requirement on input space and/or non-flop gate count, or if the circuit structure otherwise does not support the encryption scheme. For each encrypted design, the PPA was estimated using both the full ICC PPA

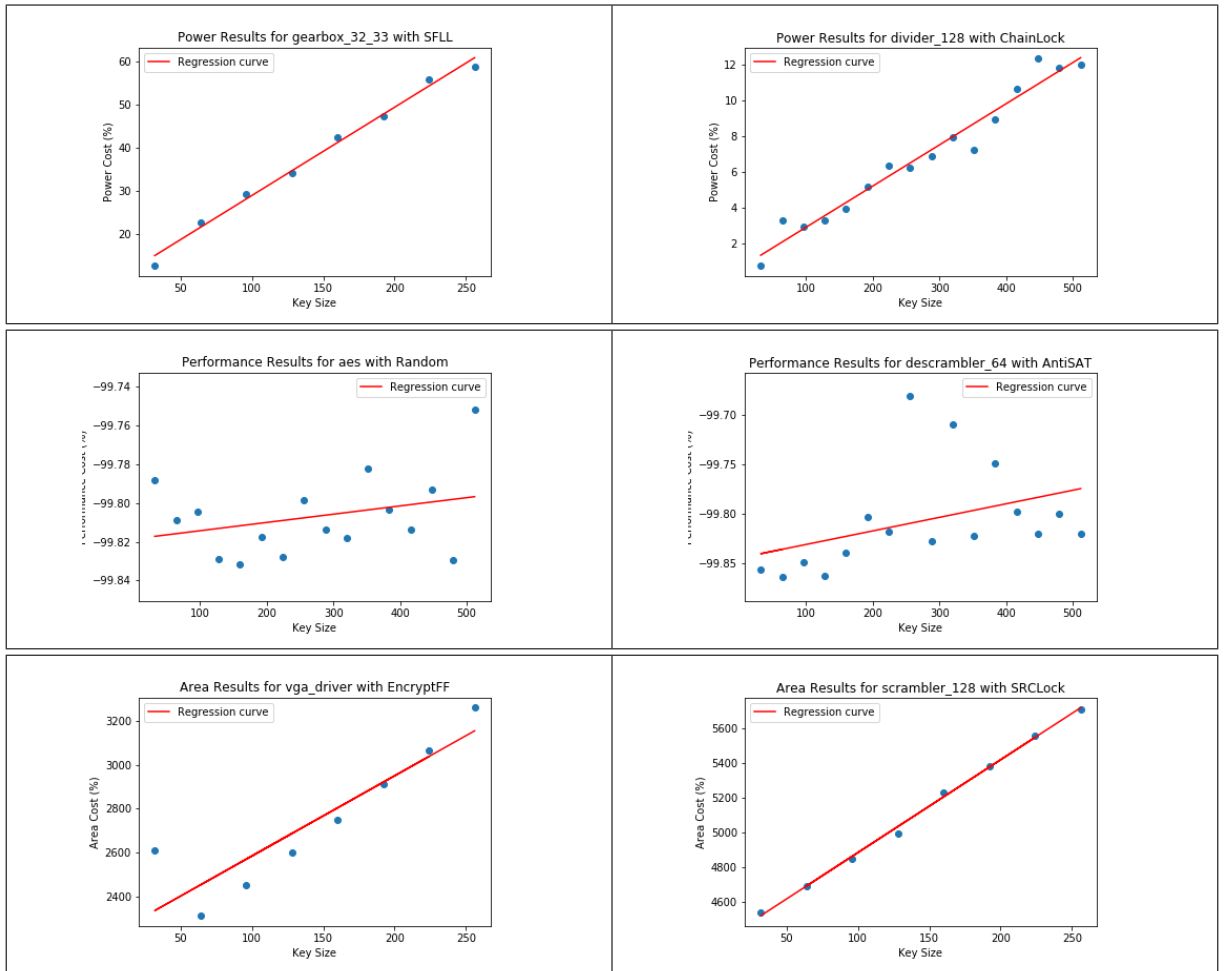


Figure 4.1: Selected examples of regression for key size vs PPA cost

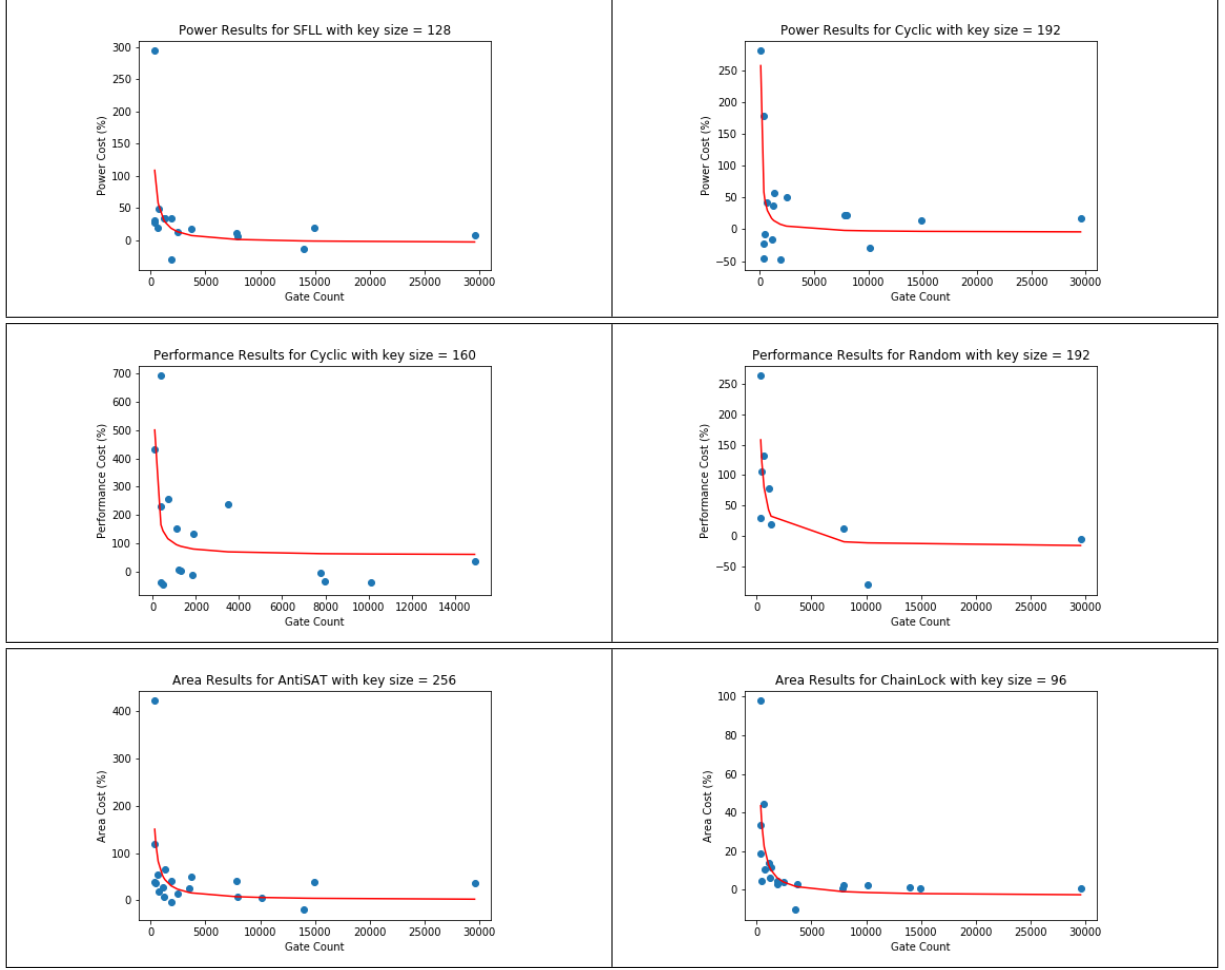


Figure 4.2: Selected examples of regression for gate count vs PPA cost

estimation strategy outlined in Section 3.4.1, as well as the modified one using only DC in Section 3.7, and compared to that of the pre-encrypted design. The percentage increases in PPA were recorded as one data point for each (3 total data points per encrypted design). A total of 8,808 data points were collected for all PPA for all designs, representing a total of 1,468 ICC/DC PPA estimations.

Several machine learning models were considered which each utilize separate regressions for power, performance, and area for each encryption scheme, a total of 24 regressions per model. Each regression has two independent variables: key size and pre-encryption gate count. These were found to be the two main variables on which the PPA costs were dependent. The relationship between key size and PPA cost was strongly linear in most cases. See the examples in Figure 4.1. Area and power costs both showed strong positive linear correlation. Performance, on the other hand, tended to be constant in key size in most cases. Simple linear regression was chosen for power, performance, and area to model the key size behavior, if gate count is held constant. To determine the correlation between gate count and PPA cost, the costs were plotted as a

function of pre-encryption gate count for all training designs, controlling for key size and encryption scheme. See the examples in Figure 4.2. For most cases, a positive inverse correlation fit best for pre-encryption gate count. With these correlations in mind, three candidate multivariable regression equations to relate PPA cost to both key size and design gate count tested were considered:

- **Linear:** $PPA = a + bk + \frac{c}{g}$
- **Interaction:** $PPA = a + b\frac{k}{g}$
- **Linear + Interaction:** $PPA = a + bk + \frac{c}{g} + d\frac{k}{g}$

where PPA is power, performance, or area, k is the key size, g is the gate count, and all other variables are regression coefficients. Each of these models could potentially result in the single-variable behavior seen during training. Therefore, the results of models using each regression equation will be compared.

Each regression model was created using Python 3.6.3 [6] using the SciPy 1.5.2 [4] least squares method. Three candidate robust least squares regression methods implemented in SciPy were chosen for comparison of quality of results. Each least squares method varies the loss function of residuals when minimizing the cost function, as described in the SciPy optimize least squares function documentation [4]:

- **Soft_l1:** $\rho(z) = 2(\sqrt{1+z} - 1)$: A smooth approximation of absolute value loss
- **Cauchy:** $\rho(z) = \ln(1+z)$: Greatly weakens the influence on the regression from outliers
- **Arctangent:** $\rho(z) = \arctan(z)$: Similar to Cauchy, limits maximum loss from a residual

Each of these least squares methods was tested using each regression equation for all necessary regressions to build the models. This resulted in a total of 9 candidate regression models for CDLE, with two submodels each using ICC and DC training estimation. Each submodel used 24 regressions. Each regression model was tested and the results of each will be compared to determine the model that provides the best quality of results. However, the average relative error among encryption schemes is reported in Figure 4.3 as a preliminary comparison of model quality, for power, performance, and area models using DC and ICC estimations. Based on the error values alone, the interaction model using the Cauchy loss function is the highest quality fit.

The last attribute of the regression model to consider is how to build predictions for compound encryption schemes using those for individual schemes. When applying a compound encryption scheme, the schemes are applied in order, and a fraction of the key space is dedicated to each encryption scheme. Considering the linear relationship between cost and key size found during training for power and area, it will be assumed



Figure 4.3: Average mean relative error among encryption schemes for each type of candidate regression model, using ICC estimations on left, and DC on left. In each graph, the bars correspond to, starting from left, linear soft_l1, linear cauchy, linear arctan, interaction soft_l1, interaction cauchy, interaction arctan, linear + interaction soft_l1, linear + interaction cauchy, and linear + interaction arctan.

that the costs of encryption schemes in a compound strategy are simply additive. So, power and area costs will be predicted for each encryption scheme for the key size equivalent to the size of the key space dedicated to them, and then summed. For performance, which is constant for key size, the cost of each scheme will be predicted in a similar manner, and the maximum among them will be accepted as the final performance cost prediction. The performance cost being based on longest timing path lends itself to this sort of prediction model as well.

4.4 Experimentation Results

To test the effectiveness of the 9 candidate CDLE machine learning models, predictions of C_E^{opt} were made for all test designs identified in Section 3.5 for each regression model. The experimental setup used is the same as that explained in Section 3.6.1, except using the machine learning models to make predictions for C_E^{opt} of each test design, for several sets of PPA constraints. The models were implemented into the CODES

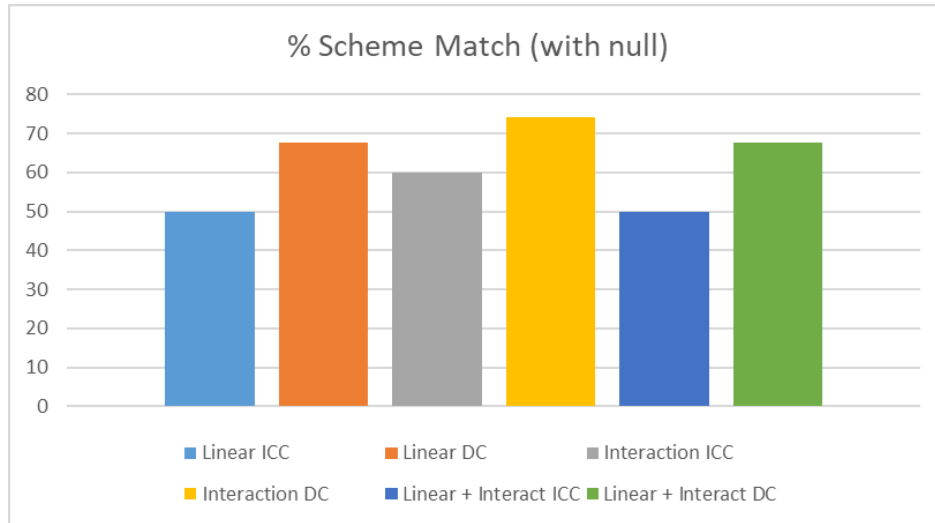


Figure 4.4: Percent of CDLE experiments that matched chosen encryption scheme when compared with the corresponding DC or ICC concurrent tree search results for each machine learning model tested. Cases in which no encryption scheme could be found in either case were counted as a match. Each cost function produced the same results, so these are not shown.

platform described in Section 3.6.1, and predictions were made using the regression equation pertaining to the model under test, and the regression coefficients determined by least squares. Predictions were made using both the models built from PPA estimations using IC Compiler II, as well as the model using Design Compiler PPA estimations with PPA constraints of 25%, 10%, 5%, and 1%, and 0.1%. These values were chosen to match the experimentation done in 3.7, since the full set of experimental designs was utilized for each model. After CDLE results were predicted for each model, they were each compared to the results of the CDLE concurrent tree search experiments for quality of results. The regression models using ICC PPA estimations were compared to the ICC experimentation in Section 3.6, and likewise for regression models using DC estimations and the DC results in Section 3.7 for all test cases that overlap.

In Figures 4.4 – 4.6, the quality of results of the regression models tested are compared to the control data, which are the respective concurrent tree search ICC and DC results. The three metrics used to determine quality of results were the percent of overlapping test cases that produced the same encryption scheme as the control including cases in which no valid encryption strategy could be found (called "null" cases), the same metric but excluding all null cases, and the percentage of cases in which the machine learning model returned a result, but the concurrent tree search algorithm returned no result. The final metric is referred to as "false positives". The models for each individual cost functions are grouped together under their overarching regression equation, because the results are equivalent among different cost functions in terms of these metrics. The only variation in results among the cost functions was some change in key sizes.

When considering null results as matching between the control and machine learning cases, the interaction

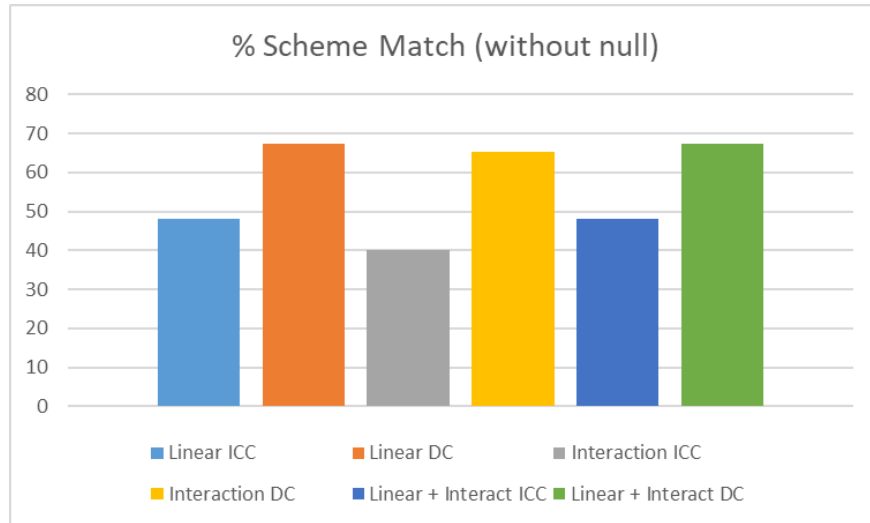


Figure 4.5: Percent of CDLE experiments that matched chosen encryption scheme when compared with the corresponding DC or ICC concurrent tree search results for each machine learning model tested. Cases in which no encryption scheme could be found in either case were not considered. Each cost function produced the same results, so these are not shown.

models performed significantly better than the purely linear and combination models. However, when the null matches are not considered, the interaction model performed slightly worse than the other models. However, when considering false positive cases, the interaction model performed much better than the others. In fact, the interaction model using ICC training estimations showed almost no false positive cases. For the linear and combination models, cases in which an encryption strategy could be found for both machine learning

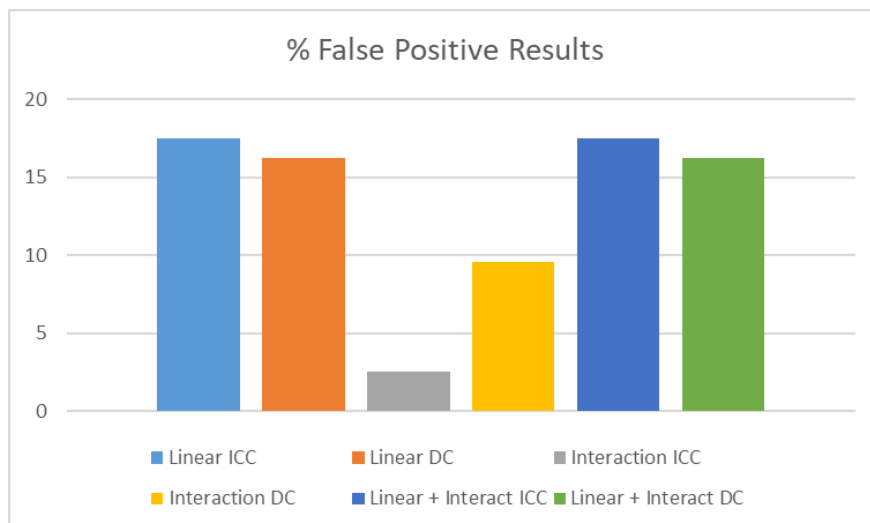


Figure 4.6: Percent of CDLE experiments in which no encryption strategy could be chosen for DC or ICC concurrent tree search results, but one was found for the machine learning model in question. Each cost function produced the same results, so these are not shown.

Table 4.2: Encryption scheme and key size predicted by the regression with the highest quality of results using ICC training estimations, interaction regression with a Cauchy cost function.

Design Name	25%	10%	5%	1%	0.10%
steppermotor	CHL+SFL 35	-	-	-	-
ss_pcm	CHL+SFL 80	-	-	-	-
usb_phy	CHL+SFL 94	-	-	-	-
sasc	CHL+SFL 122	CHL+SFL 32	-	-	-
simple_spi	CHL+SFL 157	CHL+SFL 42	-	-	-
caprng	CHL+SFL 166	CHL+SFL 46	-	-	-
hilbert	CHL+SFL 297	CHL+SFL 81	-	-	-
systemcdes	CHL+SFL 414	CHL+SFL 118	-	-	-
des_area_opt	CHL+SFL 503	CHL+SFL 145	CHL+SAR 35	-	-
des3_area_opt	CHL+SFL 512	CHL+SFL 168	CHL+SAR 47	-	-
tv80	CHL+SFL 512	CHL+SFL 301	CHL+SFL 47	-	-
ac97_ctrl_top	CHL+SFL 512	CHL+SFL 512	CHL+SFL 106	-	-
usb_funct	CHL+SFL 512	CHL+SFL 512	CHL+SFL 97	-	-
aes_cipher	CHL+SFL 512	CHL+SFL 512	CHL+SFL 136	-	-
sha256core	CHL+SFL 512	CHL+SFL 512	CHL+SFL 136	-	-
des_perf_opt	CHL+SFL 512	CHL+SFL 512	CHL+SFL 171	-	-
aes_inv_cipher	CHL+SFL 512	CHL+SFL 512	CHL+SFL 203	-	-
des3_perf_opt	CHL+SFL 512	CHL+SFL 512	CHL+SFL 512	-	-
vga_lcd	CHL+SFL 512	CHL+SFL 512	CHL+SFL 512	-	-
b19	CHL+SFL 512	CHL+SFL 512	CHL+SFL 512	-	-
cfrc4	CHL+SFL 512	CHL+SFL 512	CHL+SFL 512	-	-

and concurrent search tended to be more accurate than those for the interaction model. However, this comes at the cost a large number of false positive cases.

The linear and combination models did not vary in result for increasing gate count. However, the interaction model followed the expected behavior the closest, that is, as the design gate count increased encryption strategies could be found using tighter PPA constraints. Therefore, the interaction model showed the highest quality of results. Since the results among cost functions were similar, the cost function with the lowest average relative error as shown in Figure 4.3, the Cauchy function, was chosen as the highest quality result. The experimentation results for the interaction model using Cauchy cost are shown in Table 4.2 using ICC training estimation, and in Table 4.3 using DC training estimation. The results in each case tended to be dominated by ChainLock+SFL encryption. While this was the most chosen scheme in the concurrent tree search algorithm as well, there were a significant amount of cases in which other schemes were chosen. However, in these results, only a couple cases show other obfuscation schemes used. Additionally,

Table 4.3: Encryption scheme and key size predicted for each test design and PPA constraint set by the regression with the highest quality of results using DC training estimations, interaction regression with a Cauchy cost function.

Design Name	25%	10%	5%	1%	0.10%
steppermotor	CHL+SFL 32	-	-	-	-
ss_pcm	CHL+SFL 80	-	-	-	-
usb_phy	CHL+SFL 83	-	-	-	-
sasc	CHL+SFL 124	CHL+SFL 35	-	-	-
simple_spi	CHL+SFL 143	CHL+SFL 47	-	-	-
caprng	CHL+SFL 151	CHL+SFL 47	-	-	-
hilbert	CHL+SFL 270	CHL+SFL 76	-	-	-
systemcdes	CHL+SFL 370	CHL+SFL 117	CHL+SAR 35	-	-
des_area_opt	CHL+SFL 467	CHL+SFL 145	CHL+SFL 35	-	-
des3_area_opt	CHL+SFL 512	CHL+SFL 166	CHL+SFL 46	-	-
tv80	CHL+SFL 512	CHL+SFL 319	CHL+SFL 82	-	-
ac97_ctrl_top	CHL+SFL 512	CHL+SFL 512	CHL+SFL 184	-	-
usb_funct	CHL+SFL 512	CHL+SFL 512	CHL+SFL 182	-	-
aes_cipher	CHL+SFL 512	CHL+SFL 512	CHL+SFL 229	-	-
sha256core	CHL+SFL 512	CHL+SFL 512	CHL+SFL 221	-	-
des_perf_opt	CHL+SFL 512	CHL+SFL 512	CHL+SFL 313	-	-
aes_inv_cipher	CHL+SFL 512	CHL+SFL 512	CHL+SFL 364	-	-
des3_perf_opt	CHL+SFL 512	CHL+SFL 512	CHL+SFL 512	-	-
vga_lcd	CHL+SFL 512	CHL+SFL 512	CHL+SFL 512	-	-
b19	CHL+SFL 512	CHL+SFL 512	CHL+SFL 512	-	-
cfrc4	CHL+SFL 512	CHL+SFL 512	CHL+SFL 512	-	-

the variation in chosen schemes with gate count is not as strong as in the concurrent tree search results. In the following section, adjustments are made to the interaction Cauchy regression model to better fit the results from concurrent tree search.

4.4.1 Model Adjustments

As shown in Section 4.4, the regression model fitting the interaction equation and using the Cauchy cost equation, each described in Section 4.3.2, showed the highest quality of results among the 9 regression equation and cost function combinations tested. When compared to the results of the concurrent tree search algorithm, this model matched the results the most closely among the models tested, especially considering cases in which other models found a "false positive" and returned a result when the concurrent tree search did not. However, even considering matches of encryption scheme chosen between machine learning and

concurrent tree search, the best machine learning model still showed very little variation in encryption scheme chosen, with an overabundance of cases in which Chainlock and SFLL were chosen. Additionally, the relation between gate count and encryption strategy was not as strong in the interaction and Cauchy regression model as it was in the concurrent tree search. With these differences in mind, several adjustments were made to the highest quality machine learning model in order to better fit the concurrent tree search results. These adjustments were made under the assumption that the limitations of the training set caused these inconsistencies and that the adjustments merely account for the training set. The adjustments made were:

1. All regression coefficients for SFLL were scaled up by a factor of 10. It is possible that the training designs chosen happened to favor the SFLL block architecture, which includes hamming distance calculation. Many training designs were communication modules, which include comparators that could also be found in SFLL hamming distance hardware. This could have caused significant underestimation of the cost of SFLL.
2. The intersection term of all regressions were multiplied by a factor of 0.01. A key size of 0 should result in no added cost, regardless of overall gate count. Therefore this term was effectively eliminated from each model.
3. For all regressions, the coefficient of the interaction term was scaled by a factor of 5. This adjustment was made to compensate for the reduction of the intersection term.

After the identified adjustments were made to each regression in the interaction model using Cauchy cost, predictions were made for each test design again, as shown in Table 4.4 using ICC training data and Table 4.5 using DC training data. As expected, the behavior of the machine learning model now more closely resembles that of the concurrent tree search. ChainLock and SFLL is still the most chosen scheme, but does not dominate the results as it did before the model was adjusted. Now other schemes have been chosen as the PPA constraints are tightened. Additionally, the increasing amount of successful results with gate count much more closely resembles the concurrent tree search results for both ICC and DC estimations.

Adjusting the model shows overall behavior closer to the concurrent tree search. However, the metrics used in Section 4.4 (percent of cases in which the chosen encryption scheme matched with and without null results, and false positives in the machine learning model) do not confirm this. The comparisons of the adjusted, or compensated, model and the original model using these metrics are shown in Figures 4.7 – 4.9. The adjusted machine learning model showed some loss of accuracy in choosing encryption schemes overall. More egregiously, the number of false positives almost doubled for the model using DC estimations. The

Table 4.4: Encryption scheme and key size predicted for each test design and PPA constraint set by the regression with the highest quality of results using ICC training estimations, interaction regression with a Cauchy cost function, after adjustments were made.

Design Name	25%	10%	5%	1%	0.10%
steppermotor	-	-	-	-	-
ss_pcm	EFF+SFL 32	-	-	-	-
usb_phy	CHL+SRC 62	-	-	-	-
sasc	EFF+SFL 32	CHL+SAR 34	-	-	-
simple_spi	EFF+SFL 35	EFF+SRC 32	-	-	-
caprng	CHL+SRC 79	CHL+SRC 35	-	-	-
hilbert	CHL+SFL 92	CHL+SFL 32	CHL+SRC 39	-	-
systemcdes	CHL+SFL 48	CHL+SRC 77	CHL+SRC 39	-	-
des_area_opt	CHL+SFL 167	CHL+SRC 91	CHL+SRC 48	-	-
des3_area_opt	CHL+SFL 78	CHL+SRC 111	CHL+SRC 58	-	-
tv80	CHL+SFL 134	CHL+SFL 56	CHL+SFL 65	EFF+CYC 32	-
ac97_ctrl_top	CHL+SFL 512	CHL+SFL 152	CHL+SFL 182	EFF+SFL 32	-
usb_funct	CHL+SFL 333	CHL+SFL 115	CHL+SFL 122	CHL+SRC 46	-
aes_cipher	CHL+SFL 376	CHL+SFL 212	CHL+SFL 212	CHL+SRC 61	-
sha256core	CHL+SFL 409	CHL+SFL 512	CHL+SFL 82	CHL+SFL 32	-
des_perf_opt	CHL+SFL 509	CHL+SFL 204	CHL+SFL 332	CHL+SRC 82	-
aes_inv_cipher	CHL+SFL 512	CHL+SFL 240	CHL+SFL 272	EFF+SFL 49	-
des3_perf_opt	CHL+SFL 512	CHL+SFL 512	CHL+SFL 395	CHL+SFL 152	CHL+SAR 36
vga_lcd	CHL+SFL 512	CHL+SFL 512	CHL+SFL 512	CHL+SFL 117	CHL+SRC 43
b19	CHL+SFL 512	CHL+SFL 512	CHL+SFL 502	CHL+SFL 311	CHL+SRC 39
cfrc4	CHL+SFL 512	CHL+SFL 512	CHL+SFL 512	CHL+SFL 181	CHL+SRC 77

ICC model, however, performed just as well after adjustment. By these metrics, the adjusted model did not show any improvement. However, further adjustment could improve the edge behavior of successful cases in choosing an encryption scheme. There are more cases in which ChainLock and SRClock are chosen in the machine learning model, than in the concurrent tree search. These cases tend to happen on the edge of tight PPA constraints, but this edge of SRClock cases is tighter in the concurrent tree search. Additionally, there are designs in the concurrent tree search which did not follow the trend of increasing gate count shown in this machine learning model. A more advanced machine learning model could take circuit structure into account in more detail than it was for this model.

Table 4.5: Encryption scheme and key size predicted for each test design and PPA constraint set by the regression with the highest quality of results using DC training estimations, interaction regression with a Cauchy cost function, after adjustments were made.

Design Name	25%	10%	5%	1%	0.10%
steppermotor	-	-	-	-	-
ss_pcm	CHL+SRC 50	-	-	-	-
usb_phy	CHL+SRC 42	-	-	-	-
sasc	CHL+SRC 57	CHL+SAR 32	-	-	-
simple_spi	CHL+SRC 70	CHL+SAR 40	-	-	-
caprng	CHL+SRC 84	CHL+SRC 33	-	-	-
hilbert	CHL+SFL 62	CHL+SRC 55	CHL+SRC 32	-	-
systemcdes	CHL+SFL 107	CHL+SRC 95	CHL+SRC 39	-	-
des_area_opt	CHL+SFL 58	CHL+SRC 107	CHL+SRC 65	-	-
des3_area_opt	CHL+SFL 107	CHL+SRC 168	CHL+SRC 56	-	-
tv80	CHL+SFL 152	CHL+SFL 51	CHL+SRC 152	-	-
ac97_ctrl_top	CHL+SFL 512	CHL+SFL 112	CHL+SFL 56	EFF+SFL 32	-
usb_funct	CHL+SFL 512	CHL+SFL 152	CHL+SFL 62	CHL+SRC 44	-
aes_cipher	CHL+SFL 512	CHL+SFL 395	CHL+SFL 72	EFF+SFL 32	-
sha256core	CHL+SFL 371	CHL+SFL 228	CHL+SFL 100	CHL+SRC 62	-
des_perf_opt	CHL+SFL 471	CHL+SFL 189	CHL+SFL 107	CHL+SRC 114	-
aes_inv_cipher	CHL+SFL 512	CHL+SFL 222	CHL+SFL 152	CHL+SRC 92	-
des3_perf_opt	CHL+SFL 512	CHL+SFL 512	CHL+SFL 431	CHL+SFL 107	CHL+SRC 32
vga_lcd	CHL+SFL 512	CHL+SFL 512	CHL+SFL 498	CHL+SFL 170	CHL+SRC 49
b19	CHL+SFL 512	CHL+SFL 512	CHL+SFL 460	CHL+SFL 107	CHL+SRC 47
cfrca	CHL+SFL 512	CHL+SFL 512	CHL+SFL 512	CHL+SFL 157	EFF+SFL 35

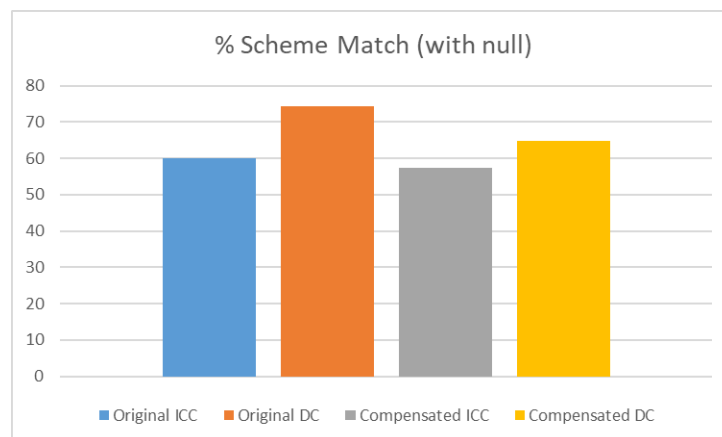


Figure 4.7: Percent of CDLE experiments that matched chosen encryption scheme when compared with the corresponding DC or ICC concurrent tree search results for the adjusted and original versions of the highest quality machine learning model. Cases in which no encryption scheme could be found in either case were counted as a match.

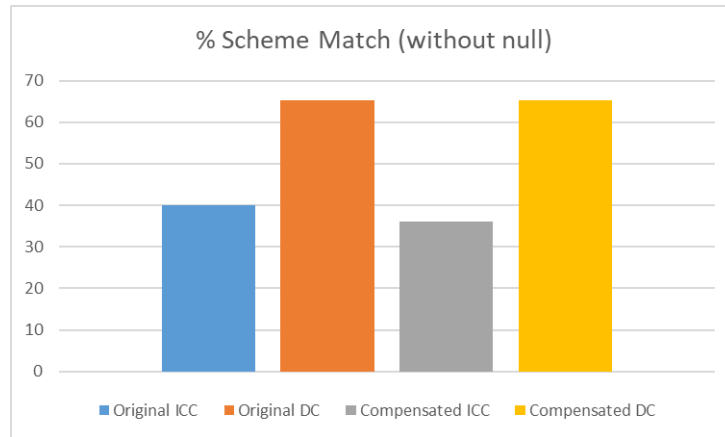


Figure 4.8: Percent of CDLE experiments that matched chosen encryption scheme when compared with the corresponding DC or ICC concurrent tree search results for the adjusted and original versions of the highest quality machine learning model. Cases in which no encryption scheme could be found in either case were not considered.

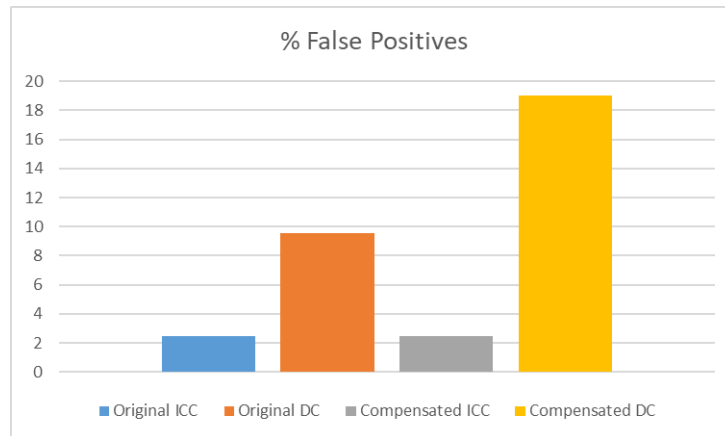


Figure 4.9: Percent of CDLE experiments in which no encryption strategy could be chosen for DC or ICC concurrent tree search results, but one was found for the machine learning model in question.

4.5 Summary

A machine learning approach to CDLE was proposed in this chapter. Several regression models were tested using different least squares methods. Although the training set was limited, some models showed similar behavior to the concurrent tree search after some adjustment.

Chapter 5

Conclusions and Future Work

5.1 Discussion of CDLE Methods

In this work, a generalized Constraint-Directed Logic Encryption (CDLE) methodology was presented. Since its inception of logic encryption, research related to it has been mostly driven by finding new potential attacks and developing new encryption methods with security metrics in mind. However, cost metrics have not been considered in general. A major issue in the field as it stands is implementation in commercial-grade designs with cost requirements, and one of the hurdles for achieving that is the lack of analysis of the PPA costs of adding logic encryption to a design. CDLE was developed to respond to the need for a holistic, extensible approach to applying logic encryption to any given design using known encryption methods in a way that responds to both security and cost requirements.

Constraint-Directed Logic Encryption is a framework with which to develop methods for applying logic encryption, rather than a single method. With cost and security constraints identified, CDLE methods explore the version space of possible encrypted versions of a given design. The subset of designs that meet the cost constraints and the subset that meet the security constraints are identified. If they overlap, then the designs within the subspace in which they overlap can be explored to find encrypted design that maximizes the level of security given by logic encryption, determined by designated security metrics.

The methodology presented can be used to create different methods that achieve this goal. The methodology is extensible, since new logic encryption methods and attacks that are developed can be still be considered within the scope of CDLE and design-space exploration. CDLE methods all share the same goal, but the method in which they each achieve the goal can vary, especially when considering the estimation of metrics. Therefore, in this work, two possible CDLE methods were introduced: concurrent tree search

and the machine learning approach, which provides two possible directions in which CDLE methods can be explored. In this chapter, these methods will be discussed with regards to their ability to achieve the CDLE goal of finding an encrypted design using constraints and their effectiveness in encrypting large-scale designs. Then, potential future work related to CDLE will be identified.

5.1.1 Concurrent Tree Search

The CDLE concurrent tree search algorithm is a method of concurrently exploring several designs in the encrypted version space of designs, bounded by security and cost requirements. In each iteration of the algorithm, the optimally encrypted design is converged on by shrinking the unexplored design space through targeted sampling of encrypted designs. First, a ranking of allowed logic encryption methods is built based on their attack resilience, and a minimum and maximum key size are given. These serve as the security requirements for encryption. Then, constraints on the increase in power, performance, and area (PPA) are given as a percentage increase allowed from the PPA of the original design. First, the PPA cost of each encryption scheme at the minimum key size is estimated in parallel, and the best scheme is chosen based on the ranking. Then, the maximum key size that meets the PPA constraints is found by sampling key sizes in parallel, shrinking the space of unexplored key sizes after each sampling, until the maximum key size allowed is found.

The PPA of each sampled encrypted design was estimated using a series commercial off-the-shelf tools, including Synopsys Design Compiler (DC) and IC Compiler II (ICC). These lead up to a final PPA estimation using a layout produced by ICC. The PPA estimations provided by ICC are, in general, very accurate. However, the trade-off is in execution time, as place and route is a lengthy process that increases in time with the size of the design. The CDLE concurrent tree search method was tested against commercial-grade designs, including designs with more than 100,000 logic gates after logic synthesis. The execution time of the algorithm became prohibitively long for designs with greater than 10,000 logic gates and was, on average, 1.54s per logic gate. Additionally, increasing the computation threads to sample more designs concurrently showed diminishing returns, so this proved to not be an effective strategy for reducing execution time. Therefore, experimentation using ICC for PPA estimation was limited to designs smaller than 10,000 logic gates. The algorithm was effectively able to converge on an optimally encrypted design in all cases which the PPA constraints were not too restrictive. The accuracy of PPA estimations with ICC proved the effectiveness of this algorithm. However, the PPA estimation method proved to be too heavy-weight for use with commercial-grade designs other than the smallest ones that were tested.

To try and improve the execution time of the algorithm and lighten the load of PPA estimation, a modified

approach was suggested. PPA estimation can be performed using DC logic synthesis only, without going through the lengthy place and route process using ICC. These estimations may be less accurate, but they execute much more quickly than using ICC. Therefore, the CDLE experimentation with the concurrent tree search was performed again, but using only DC to estimate PPA, rather than ICC. In this case, all of the designs identified for experimentation were able to be tested, including those with more than 100,000 logic gates after synthesis. As expected, the algorithm was able to converge on an optimally encrypted design much more quickly than the same algorithm using ICC for PPA estimation, with an average execution time of about 0.12s per logic gate. However, this came at the cost of quality of results.

PPA estimations from DC were significantly inaccurate compared to those from ICC, especially for performance. However, if they are inaccurate in a predictable manner, CDLE can still return the same result. Therefore, the results were compared with those using ICC for PPA estimation. For designs that were tested using both ICC and DC for PPA estimation, only 48% of cases returned the same encryption scheme when an encrypted design was able to be found (i.e. when the PPA constraints were not too restrictive). In general, the level of security provided by designs using DC tended to be lower than those found by ICC. Therefore, choosing between these PPA estimation strategies is a question of trade-offs. A result can be converged on much more quickly when using DC for PPA estimation, which extends the algorithm to more and larger commercial-grade designs. But, this comes at the cost of reduced security provided by logic encryption, as well as lower trust in the final design meeting cost and security requirements. Therefore, it is suggested that ICC be used to drive CDLE for small designs and DC for very large designs depending on allowed execution time of the algorithm, or to target specific, critical modules for encryption using ICC.

5.1.2 Machine Learning Approach

The machine learning approach to CDLE was proposed as a way to further reduce execution time of CDLE compared to the concurrent tree search algorithm, while maintaining a high level of accuracy in cost metric estimation, as seen in the PPA estimation strategy using ICC in the concurrent tree search method. This method included two main steps: training and experimentation. During the training step, a group of designs were identified as a training set, which was a separate set from the test set used during concurrent tree search experimentation. For each encryption scheme, PPA cost estimations from both ICC and DC were collected at a range of key sizes to build several regression models. For each encryption scheme, a graph for each power, performance, and area estimations was made as a function of training design gate count and key size. These two observed values were chosen because a roughly inverse relation for the former and a linear relation for the latter, were observed in each power, performance, and area. Three multivariable regression

fits were tested which could result in these single-variable behaviors: a linear model $(a + bk + \frac{c}{g})$, a model using interaction between key size and gate count $(a + b\frac{k}{g})$, and a combination of the two $(a + bk + \frac{c}{g} + d\frac{k}{g})$. Each regression fit was implemented using three different types of least squares cost functions: approximate absolute value, Cauchy, and arctangent. A regression model was tested for each combination of cost function and fit equation, totaling 9 unique models, each with a submodel using ICC and DC training estimation.

Each machine learning model was tested using the same test set as the concurrent tree search, and the results of the concurrent tree search were used as a control for determining the quality of results for each model. Among the tested models, the model that used interaction regression with Cauchy cost produced the highest quality of results. Although this model matched the results of the concurrent tree search the most closely among the tested machine learning models, there was still some inconsistency in the behavior of the model. The diversity of encryption schemes chosen was low, and the relation in encryption scheme to gate count was not as strong as it was in the concurrent tree search. Some adjustments were able to be made to the regression model so its behavior more closely resembled the concurrent tree search. Such adjustment can be explained as compensating for a limited training set. The adjustments made increased the diversity of encryption schemes chosen, and the behavior under increasing gate count more closely resembled the concurrent tree search. However, this resulted in a greater number of cases in which the machine learning model returned a result, and the concurrent tree search did not. Therefore, CDLE could benefit from a much larger and more diverse training set and a more mature machine learning model to capture more detailed structural behaviors of circuits to be encrypted. But, even this limited model shows promising results for machine learning in Constraint-Directed Logic Encryption.

5.2 Future Work

Since considering cost constraints for logic encryption is a new direction for the field, there is a lot of room for future work. CDLE is a methodology, not a single method for driving the application of logic encryption. Therefore many new methods can be developed under the CDLE umbrella. Improvements to the CDLE methodology itself could be made as well.

5.2.1 Improved cost metrics estimation

Using commercial tools like Synopsys IC Compiler II and Design Compiler proved to have a trade-off between execution time and accuracy of cost metrics estimation. More lightweight solutions to estimating the cost of applying different logic encryption schemes to a design need to be considered. Such methods could be tied into logic synthesis, or perform RT-level analysis so no synthesis is required.

5.2.2 Improved security metrics estimation

The security metrics in the CDLE methods proposed in this work were assumed based off of theoretical attack resilience. A more robust CDLE approach could tie decryption attempts into the design-space exploration processes, using known state of the art attack methods. This would improve the trust of the result from CDLE.

5.2.3 Machine learning direction

In this work, a limited example of machine learning was shown. There is much room to grow in this direction in logic encryption, and the method would benefit greatly from a much more mature machine learning model and a larger training data set. Though the models in this work were limited, they still showed promising results and machine learning should be considered as a direction for future work.

5.2.4 Other design-space search methods

More methods can be developed for CDLE that improve on the ones presented in this work, or take the methodology in new directions. The methodology is intended to be flexible and evolve with the field of logic encryption. More logic encryption methods driven by design constraints need to be explored for this hardware security method to transition to production, and CDLE provides a framework for these methods.

References

- [1] Kimia Zamiri Azar, Hadi Mardani Kamali, Houman Homayoun, and Avesta Sasan. SMT attack: Next generation attack on obfuscated circuits with capabilities and performance beyond the sat attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 97–122, 2019.
- [2] Fulvio Corno, Matteo Sonza Reorda, and Giovanni Squillero. RT-level ITC’99 benchmarks and first ATPG results. *IEEE Design & Test of computers*, 17(3):44–53, 2000.
- [3] Altera Corporation. Altera advanced synthesis cookbook. https://github.com/thomasrussellmurphy/stx_cookbook. Accessed: August 27, 2020.
- [4] SciPy Developers. Scipy. <https://docs.scipy.org/doc/>.
- [5] S. Dupuis and M. Flottes. Logic locking: A survey of proposed methods and evaluation metrics. *Journal of Electronic Testing*, 35:273–291, 2019.
- [6] Python Software Foundation. Python 3.6.3. <https://www.python.org/>, 2020. Accessed: September 15, 2020.
- [7] Google. Go language, version 1.14.2. <https://golang.org/>, 2020.
- [8] Ujjwal Guin, Ke Huang, Daniel DiMase, John M. Carulli, Mohammad Tehranipoor, and Yiorgos Makris. Counterfeit integrated circuits: A rising threat in the global semiconductor supply chain. *Proceedings of the IEEE*, 102(8):1207–1228, 2014.
- [9] Rajit Karmakar, Santanu Chatopadhyay, and Rohit Kapur. Encrypt Flip-Flop: A novel logic encryption technique for sequential circuits. *arXiv preprint arXiv:1801.04961*, 2018.
- [10] Rajit Karmakar, Harshit Kumar, and Santanu Chattopadhyay. On finding suitable key-gate locations in logic encryption. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2018.

- [11] Yasaswy Kasarabada, Suyuan Chen, and Ranga Vemuri. On SAT-based attacks on encrypted sequential logic circuits. In *20th International Symposium on Quality Electronic Design (ISQED)*, pages 204–211. IEEE, 2019.
- [12] Yasaswy Kasarabada and Ranga Vemuri. StateLock: State transition based logic locking for sequential circuits. In *2020 33rd International Conference on VLSI Design and 2020 19th International Conference on Embedded Systems (VLSID)*. IEEE, 2020.
- [13] David Luria, Yasaswy Kasarabada, and Ranga Vemuri. Constraint-directed logic encryption methodology. In *Proceedings of GOMACTech 2020 Conference*. GOMACTech, 2020.
- [14] Oliscience. Opencores IP cores. <https://opencores.org>. Accessed: March 15, 2020.
- [15] M. Pecht and S. Tiku. Bogus: electronic manufacturing and consumers confront a rising tide of counterfeit electronics. *IEEE Spectrum*, 43(5):37–46, 2006.
- [16] Seetal Potluri, Aydin Aysu, and Akasj Kumar. SeqL: Secure scan-locking for sequential circuits. <https://eprint.iacr.org/2019/656>, 2019. Accessed: 23-February-2020.
- [17] Jeyavijayan rajendran, Youngok Pino, Ozgur Sinanoglu, and Ramesh Karri. Security analysis of logic obfuscation. *DAC '12: Proceedings of the 49th Annual Design Automation Conference*, pages 83–89, 2012.
- [18] IHS Technology Press Release. Top 5 most counterfeited parts represent a \$169 billion potential challenge for global semiconductor industry. <https://technology.informa.com/405654/top-5-most-counterfeited-parts-represent-a-169-billion-potential-challenge-for-global-semiconductor-market>, 2012. Accessed: August 8, 2020.
- [19] Shervin Roshanisefat, Hadi Mardani Kamali, and Avesta Sasan. SRCLock: Sat-resistant cyclic logic locking for protecting the hardware. In *Proceedings of the 2018 on Great Lakes Symposium*, pages 153–158. ACM, 2018.
- [20] Masoud Rostami, Farinaz Koushanfar, and Ramesh Karri. A primer on hardware security: Models, methods, and metrics. *Proceedings of the IEEE*, 102(8):1283–1295, 2014.
- [21] Jarrod A Roy, Farinaz Koushanfar, and Igor L Markovs. EPIC: Ending piracy of integrated circuits. In *Proceedings of the conference on Design, automation and test in Europe*, pages 1069—1074, 2008.

- [22] Mohammad Saleh Samimi, Ehsan Aerabi, Zahra Kazemi, Mahdi Fazeli, and Ahmad Patooghy. Hardware enlightening: No where to hide your hardware trojans! In *2016 IEEE 22nd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 251–256. IEEE, 2016.
- [23] Kaveh Shamsi, Meng Li, Travis Meade, Zheng Zhao, David Z Pan, and Yier Jin. AppSAT: Approximately deobfuscating integrated circuits. In *Hardware Oriented Security and Trust (HOST), 2017 IEEE International Symposium on*, pages 95–100. IEEE, 2017.
- [24] Kaveh Shamsi, Meng Li, Travis Meade, Zheng Zhao, David Z Pan, and Yier Jin. Cyclic obfuscation for creating SAT-unresolvable circuits. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*, pages 173–178. ACM, 2017.
- [25] Kaveh Shamsi, Meng Li, David Z Pan, and Yier Jin. KC2: Key-condition crunching for fast sequential circuit deobfuscation. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 534–539. IEEE, 2019.
- [26] Yuanqi Shen, You Li, Amin Rezaei, Shuyu Kong, David Dlott, and Hai Zhou. BeSAT: behavioral SAT-based attack on cyclic logic encryption. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, pages 657–662, 2019.
- [27] Deepak Sirone and Pramod Subramanyan. Functional analysis attacks on logic locking. *IEEE Transactions on Information Forensics and Security*, 15:2514–2527, 2020.
- [28] Pramod Subramanyan, Sayak Ray, and Sharad Malik. Evaluating the security of logic encryption algorithms. In *Hardware Oriented Security and Trust (HOST), 2015 IEEE International Symposium on*, pages 137–143. IEEE, 2015.
- [29] Synopsys. Design compiler. <https://www.synopsys.com/>, 2018.
- [30] Synopsys. Tetramax. <https://www.synopsys.com/>, 2018.
- [31] Synopsys. IC compiler II. <https://www.synopsys.com/>, 2020.
- [32] Randy Torrance and Dick James. The state-of-the-art in ic reverse engineering. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009*, pages 363–381, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
- [33] Icarus Verilog. Icarus verilog. <http://iverilog.icarus.com/>, 2015.
- [34] Yang Xie and Ankur Srivastava. Mitigating sat attack on logic locking. In *International conference on cryptographic hardware and embedded systems*, pages 127–146. Springer, 2016.

- [35] Yang Xie and Ankur Srivastava. Anti-SAT: Mitigating sat attack on logic locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(2):199–207, 2018.
- [36] Muhammad Yasin, Bodhisatwa Mazumdar, Jeyavijayan Rajendran, and Ozgur Sinanoglu. SARLock: Sat attack resistant logic locking. *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 1–1, 2016.
- [37] Muhammad Yasin, Bodhisatwa Mazumdar, Ozgur Sinanoglu, and Jeyavijayan Rajendran. Removal attacks on logic locking and camouflaging techniques. *IEEE Transactions on Emerging Topics in Computing*, pages 1–1, 2017.
- [38] Muhammad Yasin, Jeyavijayan JV Rajendran, Ozgur Sinanoglu, and Ramesh Karri. On improving the security of logic locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(9):1411–1424, 2016.
- [39] Muhammad Yasin, Abhrajit Sengupta, Mohammed Thari Nabeel, Mohammed Ashraf, Jeyavijayan JV Rajendran, and Ozgur Sinanoglu. Provably-secure logic locking: From theory to practice. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1601–1618. ACM, 2017.
- [40] Muhammad Yasin, Abhrajit Sengupta, Benjamin Carrion Schafer, Yiorgos Makris, Ozgur Sinanoglu, and Jeyavijayan Rajendran. What to lock? functional and parametric locking. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*, pages 351–356, 2017.
- [41] Muhammad Yasin and Ozgur Sinanoglu. Evolution of logic locking. In *2017 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE, 2017.
- [42] Hai Zhou, Ruifeng Jiang, and Shuyu Kong. CycSAT: SAT-based attack on cyclic logic encryptions. In *Proceedings of the 36th International Conference on Computer-Aided Design*, pages 49–56. IEEE Press, 2017.

Appendix: Constraint-Directed Logic Encryption Case Studies

In this appendix, several additional brief case studies are presented to supplement the results and case studies shown in Sections 3.6 and 3.7. The case studies highlight edge behaviors of the concurrent tree search algorithm and interesting cases from the experimentation results. The figures follow the same format as previous case study sections. For all DC results, a layout was produced after the completion of CDLE using ICC2 and PPA was estimated from the layout for comparison with DC PPA estimations.

A.1 Concurrent Tree Search with ICC

A.1.1 Simple SPI

In this case, the Simple SPI design was encrypted using CDLE with PPA constraints of 50% each. The algorithm trajectory is shown in Figure A.1. The chosen encryption strategy was ChainLock+SARLock with added Random encryption at key size 345. This shows a usual execution trajectory of the algorithm. In the final step, none of the tried key sizes met the PPA constraints. Therefore, the maximum accepted key size from the previous step was used as the best key size. The layout produced for the final encrypted design is in Figure A.2. The algorithm executed in 35m 49s total, including all logic synthesis and layout place and route. The original and final PPA values estimated in this experiment were:

- Original Power: 0.265 mW
- Final Power: 0.387 mW (46% Increase)
- Original Performance: 2.2 ns
- Final Performance: 3.16 ns (44% Increase)
- Original Area: 0.0176 mm²
- Final Area: 0.0251 mm² (43% Increase)

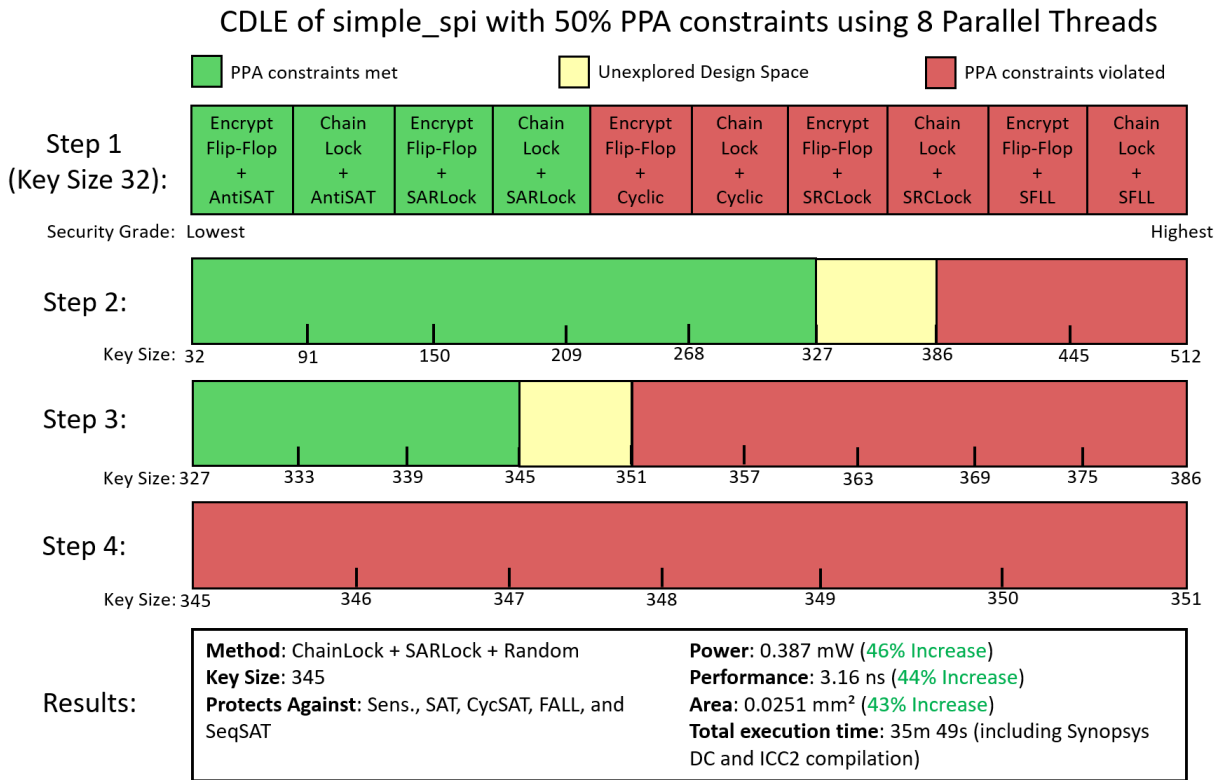


Figure A.1: CDLE Trajectory and result of simple_spi with PPA constrained to 50% increases

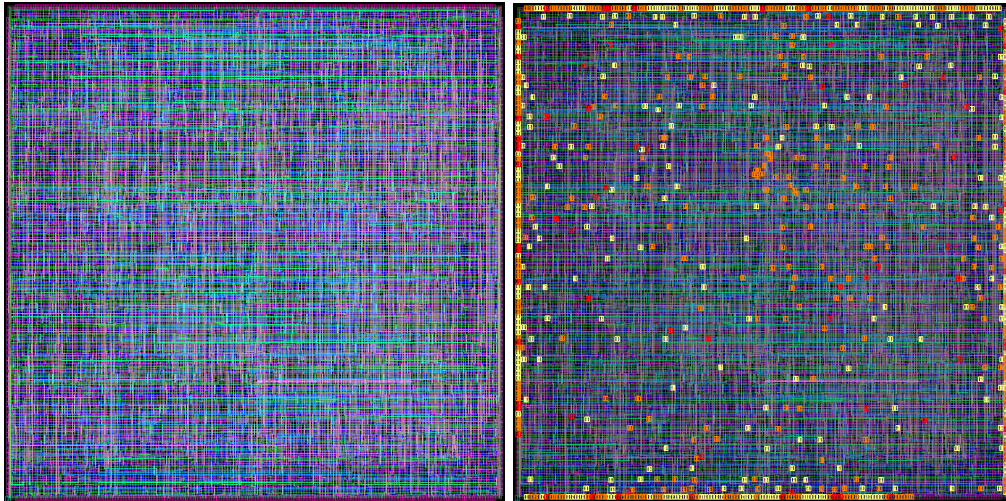


Figure A.2: Layout of encrypted Simple SPI after CDLE. On right, the key inputs and their respective key gates are highlighted, yellow corresponding to ChainLock, orange to SARLock, and red to randomly placed gates.

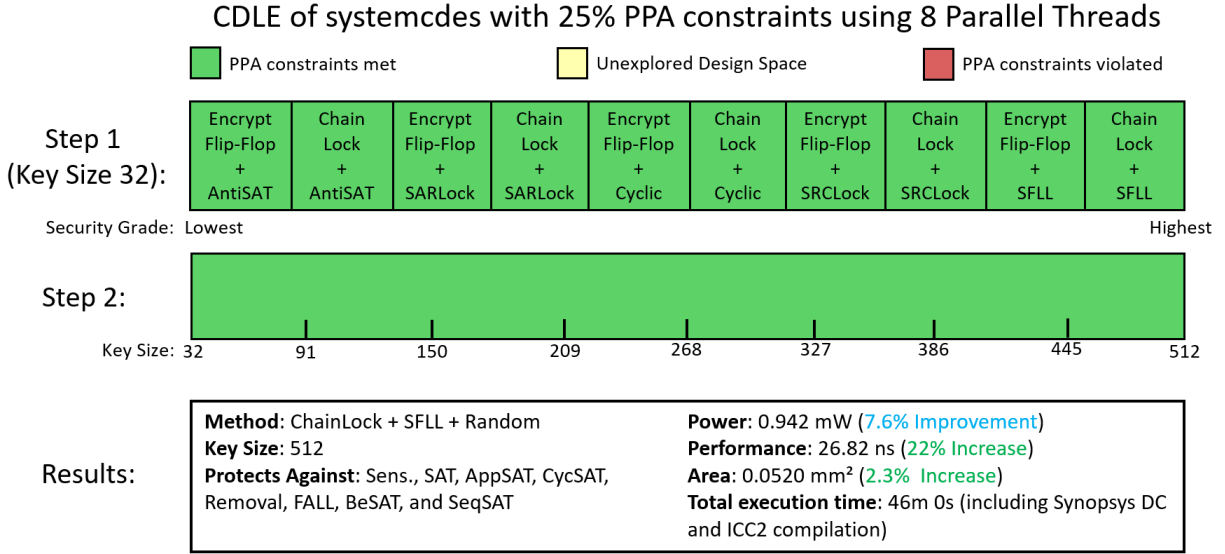


Figure A.3: CDLE Trajectory and result of systemcdes with PPA constrained to 25% increases

A.1.2 SystemC DES

This trajectory is an example of the edge case in which the PPA constraints are loose enough to allow the maximum possible security provided by the possible encryption strategies. The algorithm trajectory is shown in Figure A.3. In the first step, the encryption scheme that protected the most attacks was within the PPA constraints, and then in the second step, the maximum key size met the PPA constraints. Therefore, it was immediately accepted in this step. This way, the algorithm executes more quickly if the PPA constraints are loose compared to the cost of encryption. Therefore, the chosen encryption strategy was ChainLock+SFL at key size 512. Additionally, power usage was improved by synthesis with encryption in this case. The layout produced for the final encrypted design is in Figure A.4. The total execution time of the algorithm was 46m. The original and final PPA values were estimated at:

- Original Power: 1.02 mW
- Final Power: 0.942 mW (7.6% Improvement)
- Original Performance: 21.92 ns
- Final Performance: 26.82 ns (22% Increase)
- Original Area: 0.0508 mm²
- Final Area: 0.0520 mm² (2.3% Increase)

A.1.3 Area Optimized DES

This case shows an edge case in which the algorithm executes successfully, but the final key size is k_{min} . The algorithm trajectory is shown in Figure A.5. The encryption strategy chosen in the first step was

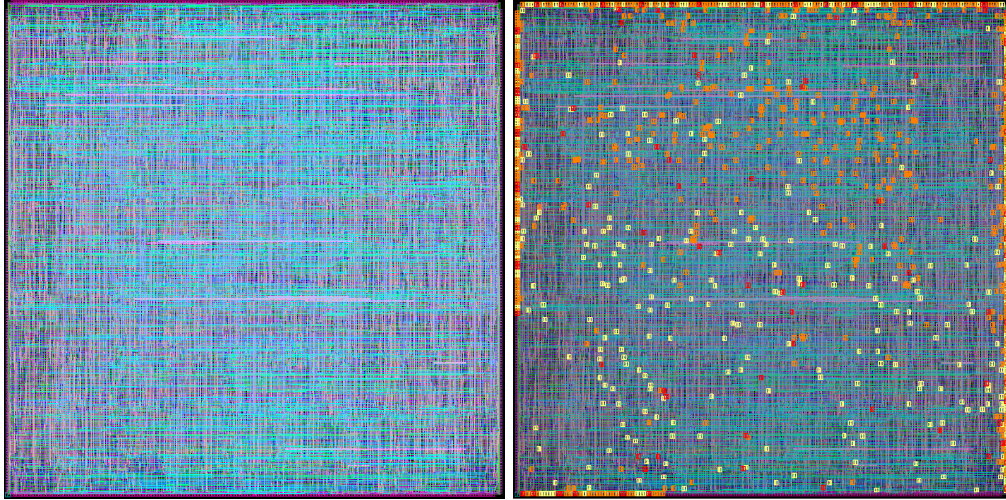


Figure A.4: Layout of encrypted systemcdes after CDLE. On right, the key inputs and their respective key gates are highlighted, yellow corresponding to ChainLock, orange to SFLL, and red to randomly placed gates.

ChainLock+Cyclic, and in the remaining steps, none of the sampled designs met the PPA constraints. Therefore, at each step, the unexplored design space shrunk at the small end of the key size space, until it collapsed to key size 32. Since 32 passed the PPA constraints in step 1, it was accepted as the best key size.

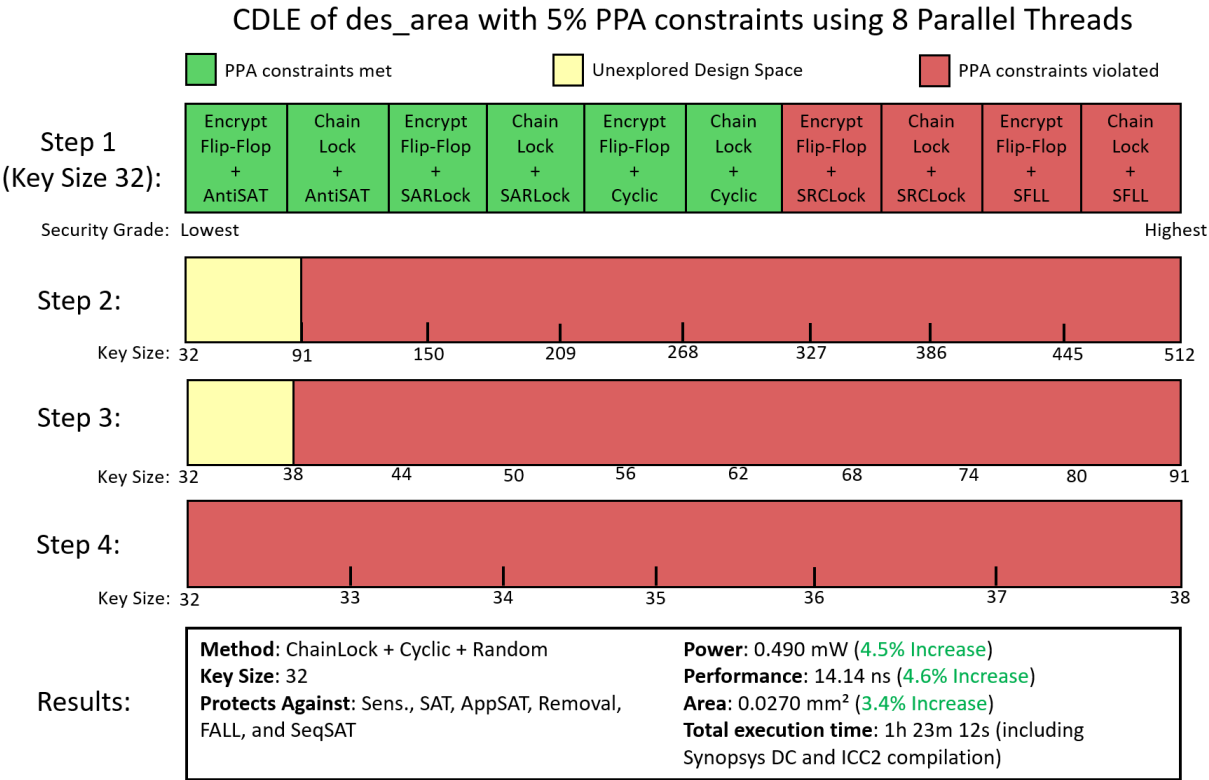


Figure A.5: CDLE Trajectory and result of des_area with PPA constrained to 5% increases

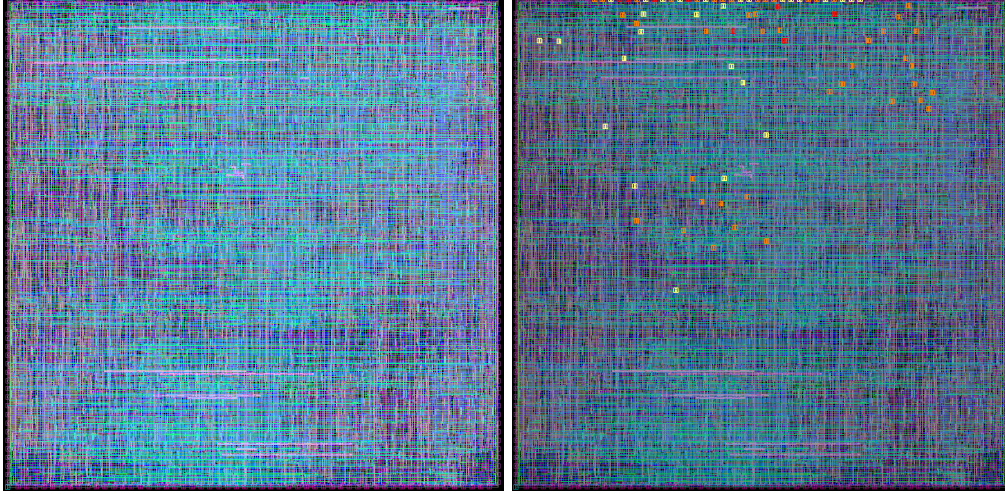


Figure A.6: Layout of encrypted des_area after CDLE. On right, the key inputs and their respective key gates are highlighted, yellow corresponding to ChainLock, orange to Cyclic, and red to randomly placed gates.

This still happens in a normal amount of time compared to other key sizes. The layout produced for the final encrypted design is in Figure A.6. The total execution time was 1h 23m 12s, and the estimated PPA values were:

- Original Power: 0.469 mW
- Original Performance: 13.52 ns
- Original Area: 0.0261 mm²
- Final Power: 0.490 mW (4.5% Increase)
- Final Performance: 14.14 ns (4.6% Increase)
- Final Area: 0.0270 mm² (3.4% Increase)

A.1.4 SASC

The SASC CDLE experiments show peculiar behavior. The algorithm trajectory is shown in Figure A.7. For one, the security metrics of the final designs found by the algorithm from 10% to 1% PPA constraints increase, which is the opposite of the expected behavior, which is to decrease the security metrics and meet the PPA requirements. This can be explained by randomness in the power estimation with random test patterns. However, in this specific SASC case, all PPA were improved by synthesis. SASC is a simple design, containing only a 4 byte data receiver and FIFO. Therefore, even a slight optimization in a cost metric can make a large impact in the overall cost of the design. The final encryption scheme chosen in this case was ChainLock+SRCLock at key size 82, found in 33m 30s. The layout produced for the final encrypted design is in Figure A.8. The PPA values estimated were:

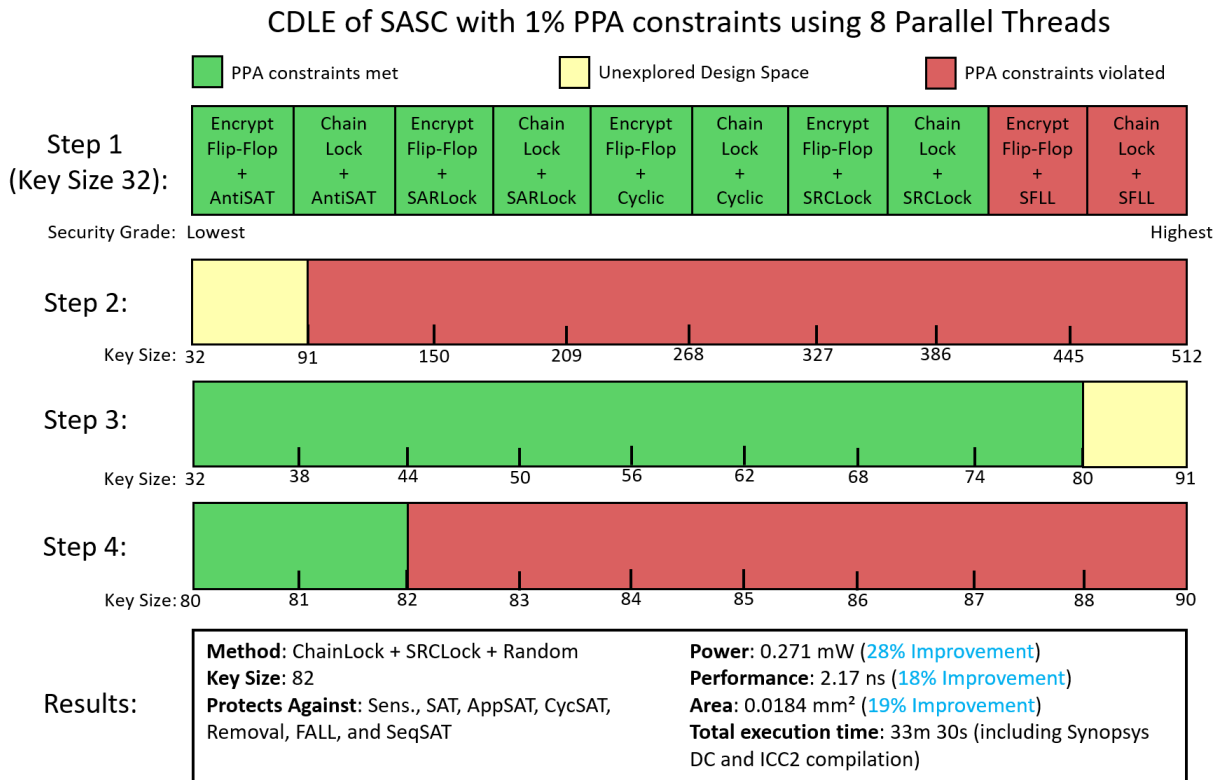


Figure A.7: CDLE Trajectory and result of SASC with PPA constrained to 1% increases

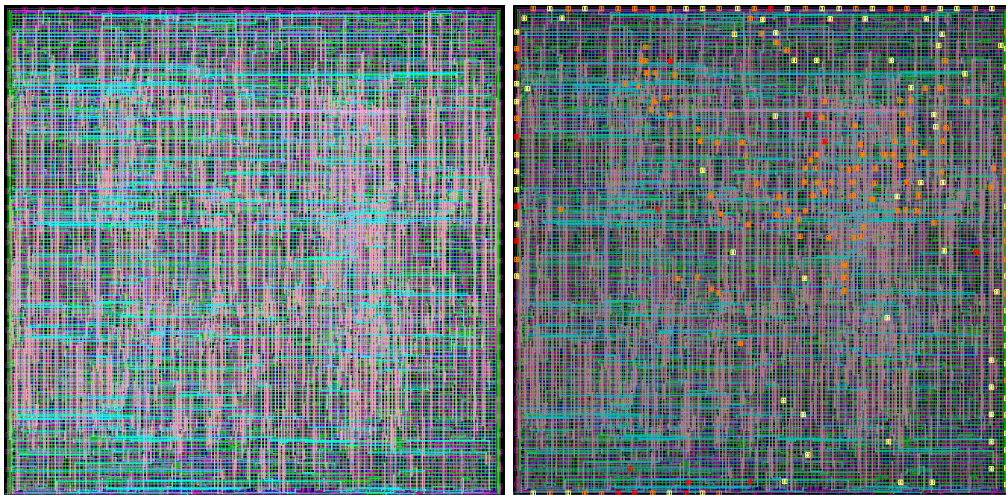


Figure A.8: Layout of encrypted SASC after CDLE. On right, the key inputs and their respective key gates are highlighted, yellow corresponding to ChainLock, orange to SRCLock, and red to randomly placed gates.

- Original Power: 0.380 mW
- Original Performance: 2.65 ns
- Original Area: 0.0226 mm²
- Final Power: 0.271 mW (28% Improvement)
- Final Performance: 2.17 ns (18% Improvement)
- Final Area: 0.0184 mm² (19% Improvement)

A.2 Concurrent Tree Search with DC

A.2.1 CF RCA

This is a case of the largest design in the test set by gate count, encrypted using CDLE with 1% constraints for each power, performance, and area. The algorithm trajectory is shown in Figure A.9. In the first step, ChainLock+Cyclic was chosen as the best encryption scheme within the PPA constraints. In the second step, none of the sampled key sizes were within the constraints, so the smallest key sizes were tested. By the final step, key size 60 was found to be the largest within the constraints. The algorithm was executed in 6h 54m 3s. So, even with only DC, the execution time can grow quickly for gates with large gate counts. The estimated PPA values in this experiment for the original and optimally encrypted designs were:

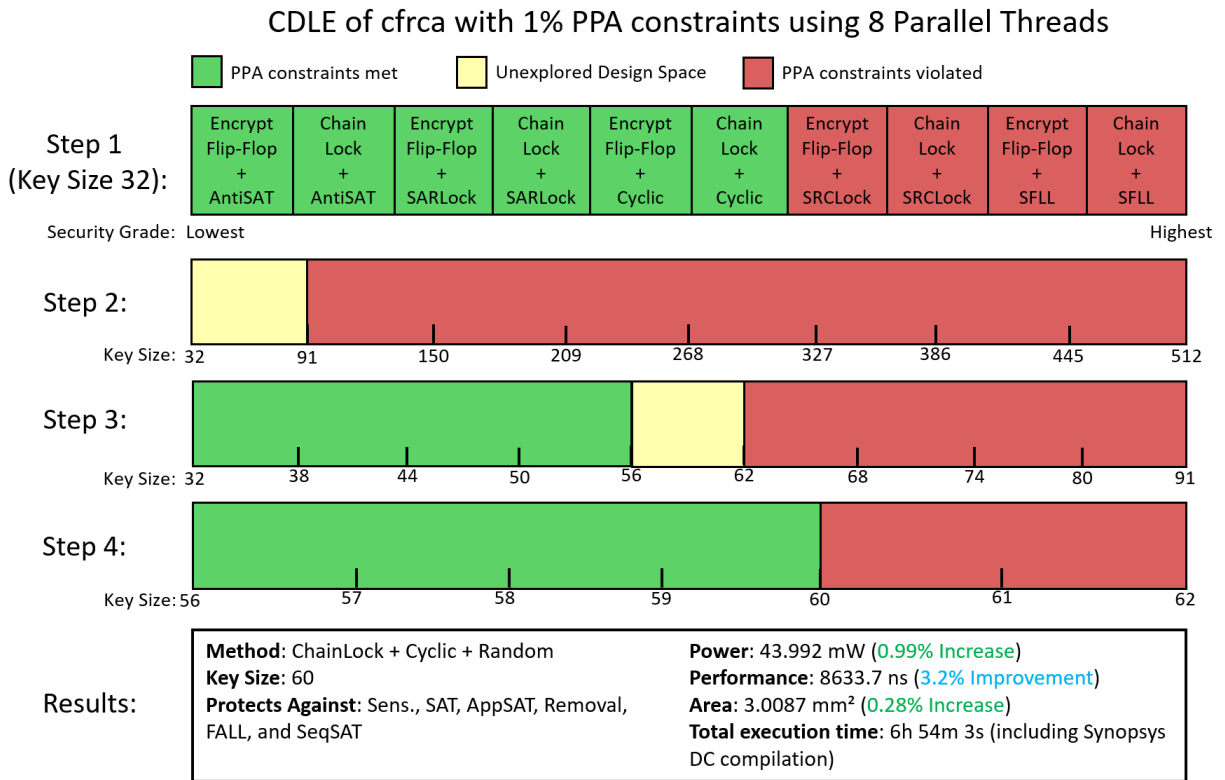


Figure A.9: CDLE Trajectory and result of cfrca with PPA constrained to 1% increases

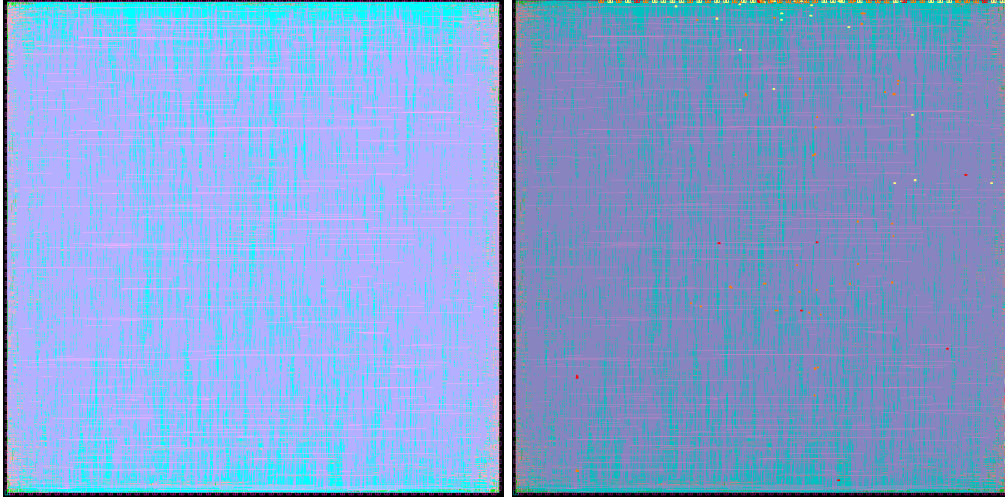


Figure A.10: Layout of encrypted cfrca after DC-only CDLE. On right, the key inputs and their respective key gates are highlighted, yellow corresponding to ChainLock, orange to Cyclic, and red to randomly placed gates.

- Original Power: 43.561 mW
- Original Performance: 8920.9 ns
- Original Area: 3.0003 mm²
- Final Power: 43.992 mW (0.99% Increase)
- Final Performance: 8633.7 ns (3.2% Improvement)
- Final Area: 3.0087 mm² (0.28% Increase)

The layout produced with the final encrypted design is shown in Figure A.10. The PPA estimated from this layout showed significantly reduced power and area usage, and much improved performance compared to the DC PPA estimation.

- Layout Power – 21.0 uW
- Layout Performance – 185.28 ns
- Layout Area – 1.8723 mm²

A.2.2 Performance Optimized 3-DES

This is a case of one of the larger designs of the test set by gate count, encrypted with very tight PPA constraints of 0.1%. The algorithm trajectory is shown in Figure A.11. As expected, an encryption strategy with weaker security metrics was selected in response. ChainLock+AntiSAT was chosen as the encryption scheme, which was rarely chosen in other experiments. The PPA constraints must be relatively strict, but not too strict, for this to happen. For large designs, this breakpoint is very close to 0. During the key size

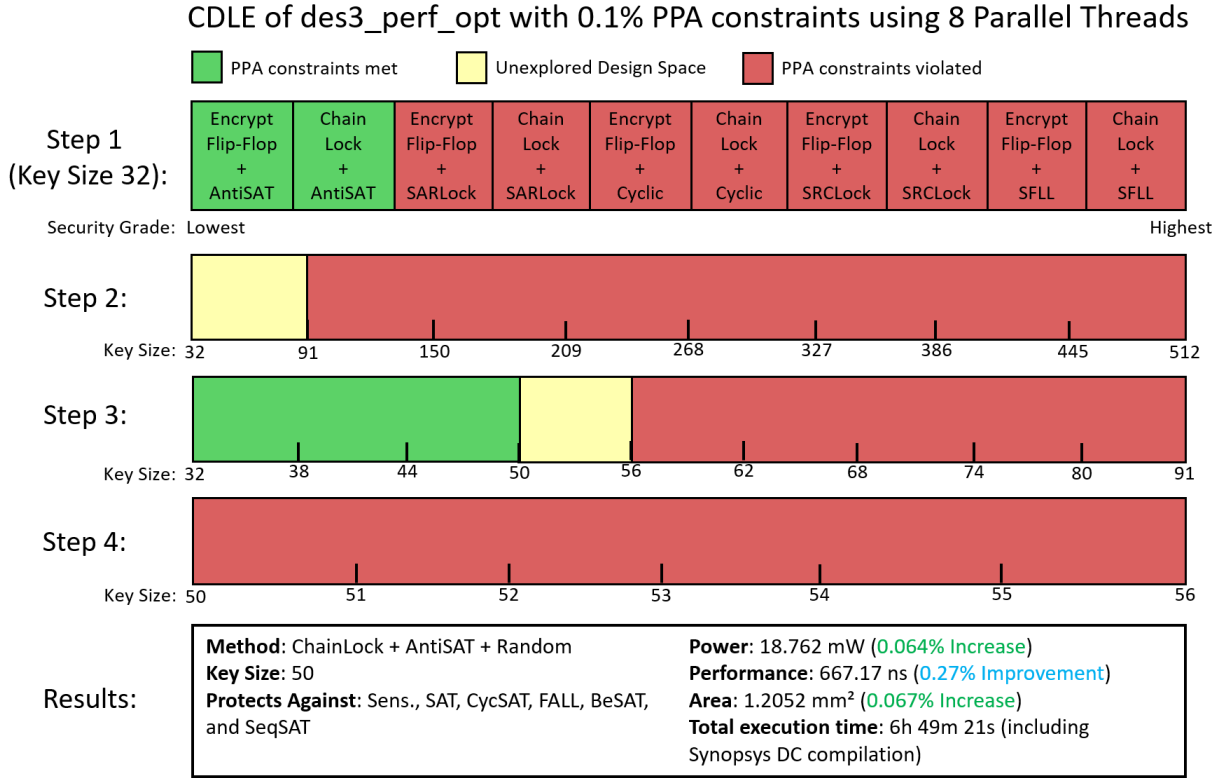


Figure A.11: CDLE Trajectory and result of des3_perf with PPA constrained to 0.1% increases

space search, 50 was the maximum key within the PPA constraints. The total execution time was 6h 49m 21s, and the PPA values estimated for the original and encrypted designs were:

- Original Power: 18.750 mW
- Final Power: 18.762 mW (0.064% Increase)
- Original Performance: 668.98 ns
- Final Performance: 667.17 ns (0.27% Improvement)
- Original Area: 1.2043 mm²
- Final Area: 1.2052 mm² (0.067% Increase)

The layout produced with the final encrypted design is shown in Figure A.12. The PPA estimated from this layout showed increases in power and area, and much improved performance compared to the DC PPA estimation.

- Layout Power – 28.600 mW
- Layout Performance – 165.29 ns
- Layout Area – 1.5225 mm²

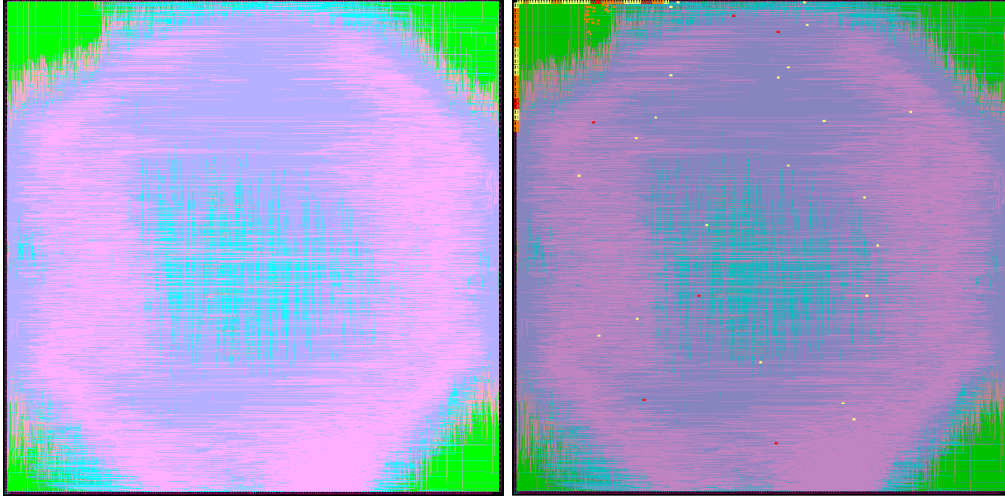


Figure A.12: Layout of encrypted des3_perf after DC-only CDLE. On right, the key inputs and their respective key gates are highlighted, yellow corresponding to ChainLock, orange to AntiSAT, and red to randomly placed gates.

A.2.3 SystemC DES

A comparison case study between this CDLE method using ICC and DC PPA estimations encrypts systemcdes with PPA constraints of 25%. The algorithm trajectory is shown in Figure A.13. The corresponding experiment using systemdex is Section 3.6.3 used ChainLock+SFLL at key size 256 for the best encryption strategy. In this case, the same encryption scheme was used, but at key size 226. The execution time with ICC estimation was 46m, while this case finished in 4m 36s. This is also considering that the ICC case executed in only two steps, choosing the maximum security encryption strategy quickly, while this case required 4 steps to close on a key size.

Like the case of des3_area, there was a large disparity between the DC and ICC PPA estimations for the unencrypted design, and therefore the encrypted design as well. For the unencrypted systemcdes, the power estimation is 61% smaller from DC compared to ICC, 416% slower in performance, and 49% smaller in area. The PPA values estimated during this experiment were:

- Original Power: 0.388 mW
- Final Power: 0.482 mW (24% Increase)
- Original Performance: 113 ns
- Final Performance: 115.7 ns (2.3% Increase)
- Original Area: 0.026 mm²
- Final Area: 0.0304 mm² (17% Increase)

The layout produced with the final encrypted design is shown in Figure A.14. The PPA estimated from this layout showed, as expected, a significant increase in power, increased area, and much improved performance compared to the DC PPA estimation.

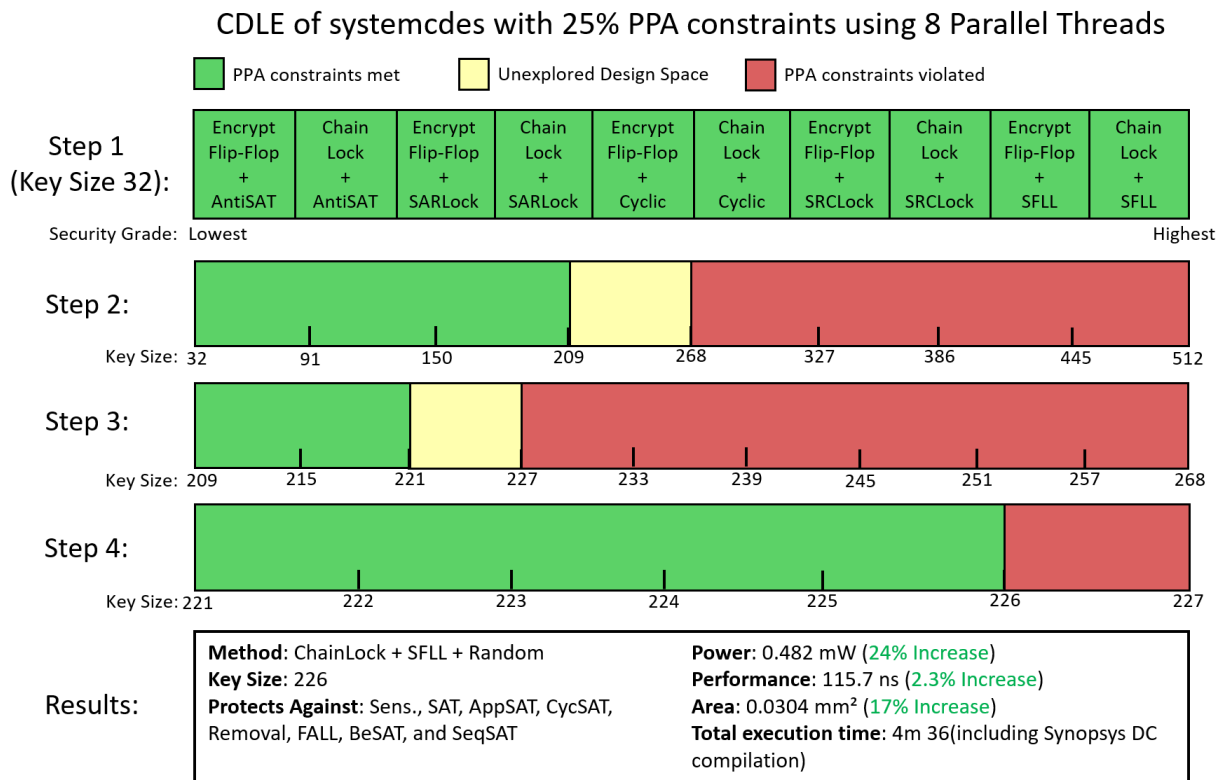


Figure A.13: CDLE Trajectory and result of systemcdes with PPA constrained to 25% increases

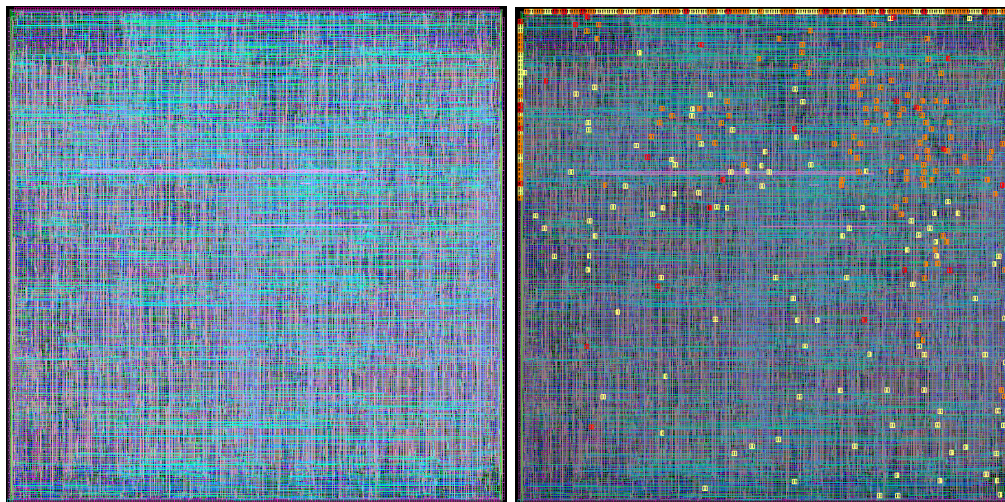


Figure A.14: Layout of encrypted systemcdes after DC-only CDLE. On right, the key inputs and their respective key gates are highlighted, yellow corresponding to ChainLock, orange to SFL, and red to randomly placed gates.

- Layout Power – 0.753 uW
- Layout Performance – 25.78 ns
- Layout Area – 0.03949 mm²