

**Enhancing the Cybersecurity Root of Trust Through Hardware Layer
Logic Obfuscation**

A Thesis

Submitted to the Faculty

of

Drexel University

by

Kyle Juretus

in partial fulfillment of

the requirements for the degree

of

Doctor of Philosophy

July 2020

Supervised

by

Professor Ioannis Savidis





GRADUATE THESIS/DISSERTATION APPROVAL FORM AND SIGNATURE PAGE

Instructions: This form must be completed by all master's and doctoral students with a thesis or dissertation requirement. Please type or print clearly as this form MUST be included as page 1 of your thesis or dissertation via electronic submission to ProQuest. All theses and dissertations must be formatted according to the University and department/program requirements. **Reminder:** It is the responsibility of the student to submit any/all edits requested by the Examining Committee to the Faculty Mentor or Supervising Professor for final approval and signature via the Graduate Program Completion Form.

Type: ☐ Master's Thesis ☒ PhD/Doctoral Thesis or Dissertation
Thesis or Dissertation Title: Enhancing the Cybersecurity Root of Trust Through Hardware Layer Logic Obfuscation

Author's Name: Kyle Juretus
Month and Year: July 2020

The signatures below certify that this thesis / dissertation (circle one) is complete and approved by the Examining Committee.

Committee Chairperson's Name: Dr. Baris Taskin
Title: Professor
Department: ECE
Institution (if other than Drexel University):

Signature: Taskin, Baris
Digitally signed by Taskin, Baris
Date: 2020.07.29 12:00:30 -04'00'

Committee Member's Name: Dr. Nagarajan Kandasamy
Title: Professor
Department: ECE
Institution (if other than Drexel University):

Signature: Naga Kandasamy
Digitally signed by Naga Kandasamy
DN: cn=Naga Kandasamy, o=Drexel University, ou=ECE,
email=nkandasamy@drexel.edu, c=US
Date: 2020.07.30 15:43:19 -04'00'

Committee Member's Name: Dr. Jeremy Johnson
Title: Professor and Dept. Head
Department: Computer Science
Institution (if other than Drexel University):

Signature: JRL
Digitally signed by Jeremy R Johnson
DN: cn=Jeremy R Johnson, o=Drexel University, ou=Computer
Science, email=jjohnson@cs.drexel.edu, c=US
Date: 2020.07.29 12:33:45 -04'00'

Committee Member's Name: Dr. Ioannis Savidis
Title: Associate Professor
Department: ECE
Institution (if other than Drexel University):

Signature: Ioannis Savidis
Digitally signed by Ioannis Savidis
Date: 2020.07.29 12:09:05 -04'00'

Committee Member's Name: Dr. Kapil Dandekar
Title: E. Warren Colehower Professor
Department: ECE
Institution (if other than Drexel University):

Signature: Kapil Dandekar
Digitally signed by member: EAT7F7A7E-FE8A-49C4-
A7B5-8A4AAAZ7389F C4681398-9299-431C-
B05B-2CE5C0C76D54
Date: 2020.07.30 15:50:42 -04'00'

Committee Member's Name:
Title:
Department:
Institution (if other than Drexel University):

Signature:

ProQuest Number:28095110

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent on the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



ProQuest 28095110

Published by ProQuest LLC (2020). Copyright of the Dissertation is held by the Author.

All Rights Reserved.

This work is protected against unauthorized copying under Title 17, United States Code
Microform Edition © ProQuest LLC.

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

© Copyright by Kyle Juretus 2020

All Rights Reserved

Acknowledgments

First and foremost, I would be remiss if I did not express my sincerest gratitude to my family and friends. Those closest to me, including my parents, aunts, uncles, family friends, and grandmother, have always instilled a desire to learn and an inquisitive nature. My soon to be wife has also been very supportive throughout my graduate studies. Having an understanding partner who encourages your educational aspirations and comprehends the triumphs and turmoils of a Ph.D. program has been greatly appreciated.

In addition, I would also like to acknowledge all my amazing mentors and professors at Drexel University, including a few who I have spent extended time with. I would like to first recognize Dr. Arye Rosen, who was my instructor for an introductory honors course during my freshman year at Drexel University. Dr. Rosen encouraged me to apply for the Students Tackling Advanced Research (STAR) program, which introduced me to university level research and opened many doors for me in the future. During my undergraduate studies, I had the privilege of having extended contact with Dr. Kandasamy, Dr. Taskin, and Dr. Hempstead in various capacities. I would like to sincerely thank all three of them for all the guidance and knowledge they provided me over the years. I deeply appreciate that both Dr. Taskin and Dr. Kandasamy are also serving on my Ph.D. committee.

I want to express my personal gratitude and thanks to my advisor, Dr. Savidis, for

approaching me to join his research group during my senior year as an undergrad at Drexel University. I have developed considerably in both a professional and personal sense during my time in the Ph.D. program. I will forever be appreciative for your support, guidance, encouragement, and time throughout the research, educational, and teaching process.

During my graduate studies, I also had the good fortune of working with Dr. Dandekar in collaboration to increase the security of wireless communications. I acquired a lot of valuable lessons from Dr. Dandekar throughout that process and appreciate having Dr. Dandekar as part of my defense committee. I would also like to thank Dr. Johnson for being on my candidacy, proposal, and defense committees. Your feedback and willingness to be on the various committees throughout my Ph.D. studies at Drexel is greatly appreciated.

During my Ph.D. studies at Drexel University, I was fortunate enough to receive the National Defense Science and Engineering Graduate (NDSEG) fellowship, which was sponsored by Air Force Research Laboratory (AFRL). I would like to thank all those involved with the NDSEG fellowship for providing me with support to explore the research of my choosing.

Having excellent colleagues and collaborators to work with throughout my Ph.D. has also been a great benefit. Our research and lighthearted conversations were greatly valued over the years.

Table of Contents

List of Tables	ix
List of Figures	xviii
Abstract	xxxiii
1 Introduction	1
1.1 The Evolution of the Integrated Circuit	2
1.1.1 Drivers of Integrated Circuit Development	4
1.2 Introduction to Hardware Security	8
1.2.1 Security Vulnerabilities of a Horizontal IC Design Flow	9
1.2.2 Countermeasures to Security Threats	12
1.3 Thesis Outline	16
2 Logic Obfuscation Techniques	21
2.1 Split Manufacturing	21
2.2 IC Camouflaging	24
2.3 Protecting Against a Diverse Threat-Model	26
2.4 In-Cone Logic Locking	27
2.4.1 In-Cone Logic Locking Methodologies	27
2.4.2 In-Cone Logic Locking Gate Selection Algorithms	30

2.5	Comparison of Logic Obfuscation Methodologies	45
3	Logic Locking Attack Vectors	49
3.1	Hill Climbing Attack	51
3.2	Sensitization Attack	53
3.3	SAT-Attack	59
3.3.1	Partial Break Algorithm	66
4	Out-of-Cone Logic Locking	69
4.1	Methodologies that Implement Unmodified Logic Cones	70
4.1.1	Anti-SAT	71
4.1.2	SAR-Lock	73
4.1.3	Vulnerabilities of Unmodified Logic Cone Methodologies	74
4.2	Modified Logic Cone Methodologies	76
4.2.1	SFLL-HD	78
4.2.2	DTL	83
4.2.3	SFLL-Flex	85
4.2.4	Structural Attacks on Modified Logic Cones	87
5	Sequential Logic Locking	93
5.1	Introduction to Sequential Logic	93
5.2	Sequential Logic Locking Methodologies	98

5.2.1	FSM Partitioning	98
5.2.2	Vulnerabilities of FSM Partitioning	102
5.2.3	Introducing Incorrect State Transitions into the FSM	105
5.2.4	Vulnerabilities of Incorrect Transitions within an FSM	108
6	Reduced Overhead In-Cone Logic Locking	112
6.1	Gate Level Logic Locking	115
6.1.1	Stack-based Topology	116
6.1.2	Transmission Gate Topology	118
6.2	Key Expansion	120
6.2.1	Key Expansion Topologies	121
6.3	Evaluation of Power, Area, and Performance	124
7	Characterization of In-Cone Logic Locking	127
7.1	Avoiding SAT Solver Specific Security	128
7.1.1	Effect of Variable Ordering on the SAT Attack	128
7.1.2	Effect of Initial DIPs on the SAT Attack	133
7.1.3	Effect of Phase Heuristic on the SAT Attack	134
7.2	Effect of Logic Structure on SAT Attack Resiliency	135
7.2.1	Analysis of Gate Heterogeneity	137
7.2.2	Analysis of Logical Reconvergence	139
7.2.3	Analysis of the Number of Keys per Node	142

7.3	Managing the Number of Correct Keys	144
7.4	Overview of MFFC Gate Selection	148
7.4.1	Determination of MFFCs	150
7.4.2	Weighting of the MFFCs	151
7.4.3	XOR-based Insertion	153
7.4.4	Heterogeneous LUT Based Insertion	153
7.4.5	2x1 MUX Based Insertion	156
7.5	Results of MFFC Based Gate Selection	157
7.5.1	Comparison with SFLL	165
7.6	Discussion of Trends in Logic Locking	167
8	Synthesis of Hidden State Transitions	169
8.1	Preventing Scan Enable During Startup	170
8.2	Security Overview Against Activation Sequence Extraction	171
8.3	Hidden State Transitions	172
8.3.1	Frequency Based Hidden State Transitions	174
8.3.2	Clock Gating Based Hidden State Transitions	176
8.4	Synthesis of Hidden State Transitions	178
8.4.1	Identification and Grouping of FSMs	180
8.4.2	Modification of State Transition Logic	180
8.4.3	Generation of Hidden State Mapping Function	183

	viii
8.4.4 Determine Hidden State Transition	185
8.4.5 Flip Outputs of Altered Registers	189
8.4.6 Generation of Key Transition	192
8.5 Results	193
9 Corruption Adaptable Logic Locking	203
9.1 CORALL	204
9.1.1 SAT Attack Resilience of CORALL	209
9.1.2 Simulation of benchmark circuits	212
9.2 Structural Attack Resilience of CORALL	221
9.2.1 Results of Simulation	233
10 Conclusions	236
11 Future Work	240
11.1 Evaluation of the Security of Logic Obfuscation Methodologies	240
11.2 Validation of the Security of Tamper Proof Memory	242
11.3 Efficient Subtractive Logic Cone Modification Algorithms	243
Bibliography	244
Vita	260

List of Tables

1.1	Evolution of Intel microprocessors from 1971 to 2017 [22, 28, 36, 38, 42, 44].	7
2.1	Overview of protection split manufacturing, IC camouflaging, and logic locking provide for the design, fabrication, test and validation, and end user stages of the IC design flow. The protection against each threat vector is rated from None, meaning no protection is provided, to High, meaning significant protection is achievable against that threat vector.	46
4.1	The complete key space for the Anti-SAT flip function shown in Fig. 4.2 applied to the original logic cone of an OR gate with inputs A and B.	72
4.2	The reduced key space for the Anti-SAT flip function shown in Fig. 4.2 applied to the original logic cone of an OR gate with inputs A and B.	72
4.3	Key space of a two input OR gate logic cone locked with SAR-Lock [98] with the logic mask occurring for input pattern 10.	74
4.4	Truth table of a two input AND gate locked with SFLL-HD ⁰ and a correct key of K1.	80
4.5	Truth table of a two input AND gate locked with SFLL-HD ¹ and a correct key of K1 or K2.	80

4.6	Key space of the DTL logic tree with all AND gates within the logic tree [107].	84
4.7	Key space of the DTL logic tree with a single NAND gate within the first layer of the tree logic [107].	85
4.8	Key space of SFLL-Flex for a two input OR gate with two protected cubes.	87
4.9	SFLL-HD ¹ key space for a three input function with a correct key of 010. The protected cube patterns are shown in red and the key value is shown in green.	90
5.1	Comparison of the vulnerability of different sequential logic locking methodologies to an unauthorized activation executed through fault-injection or FSM extraction. The reverse engineering of an IC is also considered through structural/removal and scan chain access attacks.	110
6.1	Analysis of the propagation delay and area of standard cells from a 180 nm fabrication process.	112
6.2	Analysis of the average and leakage power of standard cells from a 180 nm fabrication process.	113
6.3	Analysis of the propagation delay, power, and area of XOR based logic locking. Per-gate overheads are provided as percent increases over the standard cell values listed in Tables 6.1 and 6.2.	113

6.4	Analysis of the propagation delay, power, and area of 4x1 MUX based logic locking. Per-gate overheads are provided as percent increases over the standard cell values in Tables 6.1 and 6.2.	115
6.5	Analysis of the average leakage power of the XOR and LUT based logic locking methods. Per-gate overheads are provided as percent increases over the standard cell values listed in Table 6.2.	115
6.6	Propagation delay, power, and area analysis of the AND/OR implemented with stack-based logic locking. Percent improvements over XOR and LUT logic locking are listed.	124
6.7	Propagation delay, power, and area analysis of the NAND/AND implemented with transmission gate logic locking. Percent improvements over XOR and LUT logic locking are listed.	124
6.8	Propagation delay, power, and area analysis of the NOR/OR implemented with transmission gate logic locking. Percent improvements over XOR and LUT logic locking are listed.	125
6.9	Analysis of the average leakage power of the stack-based AND/OR and transmission gate based AND/NAND and OR/NOR topologies. Percent improvement over the XOR based topology are listed.	125

7.1	Minimum and average number of iterations to complete the SAT attack with and without all 0/1 initial constraints for the tree structure of Fig. 7.1b.	134
7.2	Analysis of the number of correct keys for four gate selection methodologies and benchmark circuits. The area allocated for logic locking is provided in parenthesis.	144
7.3	Area, power, and performance overheads of the XOR, LUT, and MUX obfuscation techniques implementing the MFFC and gate controllability based gate selection methodologies for the benchmark circuits apex2, c432, c499, c880, ex5, i7, and i9.	161
7.4	Area, power, and performance overheads of the dac12, iolts14, rnd, toc13xor and toc13mux logic locking gate selection methodologies for the apex2, c432, c499, c880, ex5, i7, and i9 benchmark circuits.	162
7.5	The minimum percentage and average percentage improvement in the minimum number of cubes, minimum number of iterations, and minimum CPU time over 100 variable orderings to complete the SAT attack is provided for the XOR, LUT, and MUX based obfuscation techniques implementing the MFFC and gate controllability based logic locking gate selection methodology for a 128 bit key as compared to the dac12, iolts14, rnd, toc13xor, and tox13mux gate selection methodologies.	163

7.6	Average percentage of corrupted outputs of the apex2, c432, c499, c880, ex5, i7, and i9 benchmark circuits for 1,000 random input patterns. 100 random key sequences are applied to each of the 1,000 random input patterns.	166
7.7	Analysis of the average number of iterations and output corruption percentage for $h=k/4$, $k/3$, and $k/2$ when implementing SFLL.	168
8.1	Results from the analysis of the ISCAS'89 benchmark circuits including 1) the number of flip-flops in the obfuscated FSM, 2) the number of inputs to the FSM being obfuscated, 3) the hidden state transition random walk score (random walk score without hidden state transitions), 4) an estimate of the average number of paths to determine the activation sequence from (8.1), 5) the number of iterations to generate the hidden state transition, and 6) the runtime to obfuscate the circuit.	190
8.2	Area, power, and performance analysis of the ISCAS'89 benchmark circuits with and without sequential obfuscation implemented with hidden state transitions. The percentage overhead in area, power, and performance is also provided for each benchmark circuit.	194

8.3	Additional overhead in area, power, and performance beyond the overheads reported in Table 8.1 for 1 to 5 modified minterms C within the state transition logic. A timing overhead of 0% implies that the performance of the circuit is constrained by the state transition logic and not the correction circuitry.	197
8.4	Analysis of execution of the SAT attack with a set timeout of 5 days for the ISCAS'89 sequential benchmark circuits. All benchmark circuits include two protected cubes. The number of iterations and the total number of SAT cubes are determined at the moment a timeout TO occurs.	201
8.5	Comparison of different sequential logic locking methodologies against various attack vectors, including 1) oracle guided attacks, 2) structural attacks that allow for the bypass of the added security features, 3) extraction of the transition from obfuscated to functional mode, and 4) extraction of the combinational state transition logic through reverse engineering. A ✓ means the technique is not vulnerable to the attack vector and a ✗ means the methodology is susceptible to the attack vector.	202

9.1	Output corruption from the modified logic cones of the b17, b18, s38584, and s35932 benchmark circuits. The secured circuits include a 20 input flip function and are evaluated for 1,000 random input patterns, with each random input pattern evaluated for 100 random key values.	214
9.2	Analysis of the power, performance, and area overhead of implementing SFLL-HD and SFLL-Flex on the b17, b18, c432, c1908, s35932, and s38584 benchmark circuits. SFLL-Flex is evaluated for 2 and 4 protected cubes c and a flip function with 5 inputs ($n = 5$). All evaluated circuits are limited to a maximum of 20 inputs to the flip function. .	219
9.3	Analysis of the power, performance, and area overhead of the CORALL architecture for the b17, b18, c432, c1908, s35932, and s38584 benchmark circuits. CORALL is evaluated for 2 and 4 protected input cubes c and LUT sizes of 2 and 4 inputs. Only a single constrained input is utilized for the generation of the CORALL circuits. All evaluated circuits are limited to a maximum of 20 inputs to the flip function. .	220

9.4	Output corruption of the modified logic cone for the CORALL iterative cofactor protected cube selection algorithm and SFLL-HD ⁰ . The number of inputs to the secured cone of the CORALL architecture is provided. The corruption is measured as the number of errors per second, assuming an IC operating at 1 GHz.	225
9.5	Comparison of the resilience of SFLL-HD, SFLL-Flex, and CORALL to the structural attacks proposed in [108–110, 151]. A ✓ denotes the technique is resilient to the attack vector and a ✗ denotes the methodology is susceptible to the attack vector. A ✓* indicates the topology is partially resilient to the attack vector.	229
9.6	SAT attack analysis of CORALL implemented with two protected cubes and a LUT size of two inputs. A TO designates a timeout, which was set to five days (432,000 seconds) for the analysis. The number of needed key bits is determined by utilizing (9.1) and the number of inputs to the logic cone from Table 9.4.	230

9.7	Overhead in area, power, and performance of implementing CORALL on a subset of ISCAS'85, MCNC, ISCAS'89, and ITC'99 benchmark circuits when utilizing the algorithm that selects protected cubes through iterative cofactors. The number of inputs and the number of gates are provided for each benchmark circuit. The runtime of the iterative cofactor algorithm is also listed for each benchmark circuit.	231
-----	--	-----

List of Figures

- 1.1 Overview of the typical computing stack. 2
- 1.2 The (a) first point contact transistor developed by Bardeen, Brattain, and Shockley at Bell Laboratories in 1947 [2, 3] and (b) the first integrated circuit developed by Jack Kilby at Texas Instruments in 1958 [3]. 3
- 1.3 The evolution of the IC from the Fairchild integrated circuits of the 1960’s (a)-(c), to early Intel processors including the (d) 4004, (e) 8008, and (f) 80386, to the Intel Pentium IV (g), and finally to the recent (h) Intel i7 Coffee Lake 6-core processor [13–17]. 5
- 1.4 Number of transistors in select Intel processors from 1971 (Intel 4004) to 2017 (Core i7-8700k Coffeelake) [22–42] and the 2019 AMD Epyc Rome processor [43]. 6
- 1.5 Typical IC development, manufacturing, and test flow along with associated threats at each stage. 9
- 1.6 Input dependent current noise generated by a NAND gate on the power distribution network [77]. 13

1.7	Increased voltage noise on the power rail caused by activation of a hardware modification when a) the non-Trojan NAND switches from 1-0, and b) both the Trojan and non-Trojan NAND switch from 1-0. .	14
2.1	Partitioning of an IC for split manufacturing. The a) cross sectional view and b) top view of a partitioned circuit are shown [83].	22
2.2	Application of the placement perturbation algorithm where (a) depicts the original unmodified placement, and (b) gate A and gate B are swapped. Dashed lines represent the missing connections to the BEOL and red lines indicate the gates with the closest proximity to A and B, respectively. Wirelength is further reduced from 70% overhead by the placement shown in (c) by applying a smaller perturbation to gates A and B [85].	23
2.3	IC camouflaging of a gate with dummy contacts, where a depiction of a) the top view of standard NAND and NOR gates and camouflaged NAND/NOR gates, and b) the cross section of dummy and true contacts are provided [86].	25
2.4	In-cone logic locking methodologies, including (a) XOR-based [88], (b) MUX-based [89], and (c) LUT based [90].	28
2.5	Number of combinations P that an attacker must consider when a key of size N is 128 for different values of incorrect bits Q [89].	31

2.6	Hamming distance of the outputs of ISCAS benchmark circuits when applying the correct key and a random incorrect key for a) random insertion and b) fault analysis-based insertion of XOR gates, and c) random insertion and d) fault analysis-based insertion of MUX gates [89].	37
2.7	Analysis of Hamming distance as a function of the percentage of gates encrypted in a circuit [90].	40
2.8	Impact of α on the amount of information learned [90].	41
2.9	Example of mutable and non-mutable key gates. Key gates K1 and K2 are pairwise secure. K3 is mutable and can be sensitized to the output.	42
3.1	Example of transformation of (a) original logic function to a (b) logic locked functional representation.	50
3.2	Example of a dominating key gate K2. The key gate K1 must be muted since the gate is part of the transitive fan-in cone of K2. Applying a logic 0 to I0 or a logic 1 to I2 mutes the effect of K1, which allows for the determination of K2 at OUT0.	55
3.3	Example of a sequentially mutable circuit. The key gate K1 must be muted first since K2 cannot be muted. Applying a logic 1 to I5 mutes K1 and allows for the determination of K2 at OUT0 [91].	56

3.4	Miter circuit with replicated A and B versions of the reverse engineered netlist that includes obfuscated gates. Corresponding outputs between the A and B versions of the circuit are XORed. Each XORed signal is then passed to an OR gate to check for any differentiating output. . .	61
3.5	Encrypted circuit with $(k_1, k_2) = (1, 1)$ [96].	64
3.6	Analysis of the execution time (left) and number of input/output combinations needed to decrypt the circuit (right). Each color represents a different locking technique and each marker represents a different allowed percentage overhead [96].	65
3.7	Box plot of the runtime of Algorithm 7 [96].	66
4.1	Generalized circuit topology to defend against the SAT attack utilized in [98–101]. Signals represented by X , which are extracted from the inputs to the logical cone or the internal signals of the logical cone, are combined with key signals K to generate a function that controls the assertion of the flip function.	70
4.2	The flip function of the Anti-SAT [99] technique where the two functional complements g and $\bar{g}$ are ANDed together. For the given circuit, the AND gate implements g and the NAND gate $\bar{g}$	71

4.3	Logic diagram of SAR-Lock, which utilizes a comparator to compare an applied key value and the primary inputs of an IC. A mask function then prevents a logic flip when the correct key is applied [98].	73
4.4	Block diagram of the SFLL-HD logic locking technique, where the perturb unit is shown within the dotted box and with the correct key labeled as K^* . The restore unit corrects any modifications introduced by the perturb unit.	79
4.5	The diversified logic tree (DTL) (a) alterations to the logic tree and (b) the high level corrupt and correct DTL topology [107].	83
4.6	SFLL-Flex topology with 3 inputs connected to the LUT restore unit and three flip function vectors serving three outputs [101].	86
4.7	The (a) unmodified ISCAS'85 c17 benchmark and (b) the c17 benchmark with a stuck-at-1 (sa1) fault applied to N11.	91
5.1	The (a) truth table of a 2-bit counter with an enable (EN) and inputs A and B , (b) the combinational logic implementing the circuit, and (c) the unrolled combinational circuit.	94
5.2	The (a) circuit symbol of a D flip-flop and (b) the corresponding truth table representation of the functionality of the D flip-flop.	95
5.3	(a) A 2-bit counter with an enable bit represented with two D flip-flops and (b) the corresponding state transition graph (STG) representation.	96

5.4	Representation of (a) the mux-based scan circuitry implemented to output the internal state of an IC, where the scan enable (SE), clock (CLK), scan in (SI), and scan out (SO) signals are applied, and (b) the activation pattern of the scan chain. When SE is high, the flip-flops form a scan chain and sequentially output the internal state of the IC through the SO pin. Activation of the scan chain begins with SE set to logic high to scan in and store a specific data pattern to the registers. SE is then transitioned to logic low to capture the circuit response through the combinational logic at the functional frequency of the circuit, and then switched back to logic high again to scan out the test response.	97
5.5	Modified FSM with an enabling sequence of P0, P1, and P2 required to enter normal circuit operation [115].	99
5.6	The (a) modification cell structure and (b) control circuit diagram for HARPOON [115].	100
5.7	Partitioned FSM methodology that generates a code word when transitioning through the states of the obfuscated mode. Transitioning from state 2 is dependent on the value of the code word [117].	101
5.8	Modified FSM with black hole states that never return to the original state transition graph (STG) [122].	103

5.9	A depiction of a (a) modified FSM with replicated states and RUB to alter the state transition, and (b) block diagram representation of the RUB [126]. The key bits are utilized to correct the output of the RUB to ensure correct transition.	106
5.10	Modified FSM to increase the number of unrollings required to successfully execute a SAT attack when scan chain access is restricted [124].	108
5.11	The (a) STG of an FSM implementing dynamic state-deflections as proposed in [125] and (b) the circuit modifications that allow for dynamic state deflections.	109
6.1	Stack-based topology implementing a NAND or NOR gate depending on the value of KEY0.	117
6.2	Transmission gate based topology implementing an AND/NAND depending on the values of KEY0 and KEY1.	119
6.3	Key expansion utilizing net KE as a standard key input.	121
6.4	Key expansion topology with two NAND/AND gate level logic locking gates.	122
6.5	Controllable path insertion to mask key expansion.	123
7.1	AND-tree topologies with a) an input at every level, and b) inputs at the level farthest from the output.	129

- 7.2 Simulation of the gate tree structure from Fig. 7.1a with 1,000 random netlist orders quantifying the number of iterations required for the SAT attack to decrypt the tree. All nets within the tree are encrypted for the simulation. The vertical line represents the netlist order used in [96].129
- 7.3 Simulation of the gate tree structure from Fig. 7.1b with 1,000 random netlist orders quantifying the number of iterations required for the SAT attack to decrypt the tree. All nets within the tree are encrypted for the simulation. The vertical line represents the netlist order used in [96].131
- 7.4 SAT solver propagating a constant logic 1 at the output of the AND-tree. After a key gate is reached, the solver has an increased degree of freedom when selecting input logic values. 133
- 7.5 Simulation of the gate tree structure of Fig. 7.1b with 1,000 random netlist orders. The number of iterations required for the SAT attack to decrypt the tree with a default SAT solver phase of true is quantified, with the vertical line representing the netlist order used in [96]. All nets within the tree are encrypted for the simulation. 136

7.6	Analysis of each gate type starting as a homogeneous tree (probability of dominant gate = 1) and then decreasing the probability of selecting a dominant gate to 0.1 (0.25 represents an AND, OR, NAND, and NOR selection probability of 25%). XOR based logic locking is applied to 25% of the circuit inputs.	138
7.7	Analysis of the effect of reconvergence on the number of SAT iterations required to decrypt the key for a tree structure consisting of gates of the same type. The largest minimum number of iterations across the 1,000 configurations of the benchmark circuits for each gate type and reconvergence ratio is listed above each data point.	140
7.8	Characterization of the effect the number of keys applied at a given netlist location has on a) the number of iterations and b) the run time required to complete the SAT attack.	142
7.9	Example of a buffer/inverter chain topology that must be accounted for when selecting nets to secure. Selection of nets C, D, or E results in an increase in the number of functional keys.	145
7.10	AND gate topology with dependent inputs limiting the possible functional outputs, which results in multiple correct key values when both inputs are secured.	147

7.11	Example circuit that includes pairwise secure key gates K1, K2, and K3 [91]. The pairwise security of the key bits prevents the key sensitization attack proposed in [129].	148
7.12	An example of an MFFC for net N23. Gates contained within the MFFC converge to node N23. Dashed edges represent graph edges where potential XOR/XNOR locking is possible while still being contained within the MFFC.	150
7.13	Number of required iterations of the SAT attack for in-cone logic locking gate selection methodologies when securing 5% of the gates in the netlist. Number of keys shown in parenthesis next to benchmark name. Evaluations are completed with 1,000 random variable orders and the default phase heuristic of Lingeling [127], logic θ , and logic 1	158
7.14	Number of required iterations of the SAT attack for in-cone logic locking gate selection methodologies when when securing 25% of the gates in the netlist. Number of keys shown in parenthesis next to benchmark name. Evaluations are completed with 1,000 random variable orders and the default phase heuristic of Lingeling [127], logic θ , and logic 1	160
8.1	A count register C is compared with a desired count X to determine the number of transitions that have occurred, which allows entrance to scan mode if correct.	171

8.2	Original state machine with single input X	173
8.3	State machine with hidden state transitions (HST) from $S3$ to $S4$, $S6$ to $S8$, and $S5$ to $S7$ that activate when a strategic timing glitch is applied. A single input X is applied to the circuit.	174
8.4	Simulation of the state machine with (a) no frequency based state transition, and (b) a temporary increase in clock frequency to enable a hidden state transition to $S4$	175
8.5	Circuit topologies to implement logic based hidden state transitions: (a) Key controlled gating of the clock signal, (b) clock signal gated with an AND gate, where the second input of the AND gate is the output of an XOR gate included to implement clock gating, and (c) clock signal gated with an AND/OR [143, 144] and with an XOR integrated to implement clock gating.	177
8.6	Variability analysis of the arrival time of $KEY0$ for a hidden state transition implemented on register $S[0]$ in the FSM shown in Fig. 8.3. The minimum and maximum arrival times for the $KEY0$ transition are shown, as well results from analysis of the slow-slow (SS) and fast-fast (FF) corners, which indicates proper circuit functionality with a 30% variation in the arrival time of $KEY0$	179

- 8.7 Primary steps to transform an unobfuscated circuit into a circuit that includes a secure partitioned FSM with a hidden state transition. The procedural steps that include a letter are further described in Section 8.4.181
- 8.8 Modified FSM that requires a temporally dependent key. Dotted lines represent incorrect edges that are corrected by asserting $K0$ or $K1$ in state $S5$ (correct transitions are shown as thick red lines). 182
- 8.9 Example of a hidden state mapping function where (a) depicts the logical remapping of the states in the form of a truth-table and (b) is a schematic of the circuit that implements the hidden state mapping function of an FSM. 184
- 8.10 Procedural steps to generate the hidden state transition values. The steps include: 1) selection of the functional mode state, 2) flipping of a randomly chosen non hidden state (HS) bit, and 3) random selection of the bits not flipped in step two. The state bits that are modified during each step are highlighted in gray. 185

8.11	Unrolling the combinational logic of the state machine. The ending state of the hidden state transition in the functional partition is m , the starting state of the hidden transition is $m - 1$, and states $m - 2$ and $m - 3$ are previous states that are constrained to not equal states m or $m - 1$. The f_0 to f_n inputs represent the flip signals determined by the SAT solver to meet the required constraints.	188
8.12	Logical structure of the correction circuitry.	191
8.13	Histogram of circuit timing (ns) for 100 instances of the s35932, s526, s15850, and s298 benchmark circuits.	199
9.1	Truth tables of a two input AND gate locked with a) SFLL-HD ⁰ with a correct key of K1, b) SFLL-HD ¹ with a correct key of K1 and K2, c) a LUT based correction circuit with a correct key of K1, and d) CORALL $n2l1c1$ with a correct key of $K1$	206
9.2	Implementation of the flip function circuit of CORALL. The first layer consists of l -input LUTs, where l is two for the example shown, with the inputs to the modified cone provided to the LUTs. The outputs of the LUTs are inputs to an AND tree, which forms a single protected cube. Multiple protected cubes are then provided as inputs to an OR gate, which generates the flip signal to correct any errors in the modified logic cone.	207

9.3	Number of keys required for a standard LUT based logic locking topology and the CORALL methodology as the number of primary inputs to the circuit is increased from 1 to 12. The CORALL architecture is evaluated for LUT sizes of 2, 3, 4, 6, and 8 inputs.	208
9.4	Number of unique columns of the key space truth table as the number of protected cubes c is increased for CORALL and SFLL-Flex when $n = 4$. The l size of CORALL is varied from 1 to 3.	211
9.5	Comparison of the SFLL-HD, SFLL-Flex, and proposed CORALL out-of-cone logic locking techniques when the modified logic cone requires a high corruption of outputs as compared to an activated IC. The techniques are evaluated for circuit implementations of the flip function with 5, 10, 15, and 20 inputs. The number of cubes c and LUT-size l are varied for CORALL. The SFLL-HD architecture is evaluated for varying Hamming distances computed based on the number of inputs n . The SFLL-Flex architecture is analyzed for a varying number of cubes c and number of inputs per cube n	213

9.6	Number of SAT attack iterations required to decrypt the ISCAS'85 c1908 benchmark circuit when the circuit is locked with a 10 input flip function using the SFLL-HD and CORALL methodologies. Each secured circuit is evaluated for 500 netlist orders [148] to characterize the variation in the number of iterations. The SFLL-HD technique is evaluated for Hamming distances of 0, $n/4$, $n/3$, and $n/2$, where n is the number of inputs to the circuit of the flip function. The CORALL methodology is evaluated for $c2l1f1$, $c2l2f5$, $c2l2f7$, $c2l2f9$, and $c5l3f9$, where c represents the number of cubes, l the LUT size, and f the number of free inputs.	216
9.7	Three different perturb functions generated with the developed algorithm that selects protected cubes through iterative cofactors. Note the topological differences between the three generated circuits and the variance in the support set of the functions.	224
9.8	Procedural stages of the modifications to the original logic cone to include the logic of the perturb unit shown in Fig. 9.7a. The schematics depict (a) the unmodified c17 circuit, (b) the modified logic cone before re-synthesis, and (c) the modified logic cone after re-synthesis.	224

Abstract

Enhancing the Cybersecurity Root of Trust Through
Hardware Layer Logic Obfuscation

Kyle Juretus

Supervised by Professor Ioannis Savidis

While the security of software has garnered significant attention in the cybersecurity landscape, the expanding utilization of untrusted third parties within the hardware supply chain raises concern over the security of the hardware layer. This dissertation focuses on 1) developing low-cost logic locking solutions, 2) incorporating logic locking into the electronic design automation (EDA) flow, and 3) enhancing the security of logic locking against oracle guided attacks.

When an oracle guided attack is not part of the threat space, the main constraint limiting the application of in-cone logic locking techniques is the overhead in area, power, and performance needed to add the locking elements to the circuit. Gate level logic locking techniques are presented that reduce the overhead of logic locking, which results in a 34.54% increase in performance, 41.50% reduction in power, and a 43.58% reduction in area as compared to XOR-based logic locking methodologies.

When oracle guided attacks are considered as part of the threat model, the security of in-cone logic locking techniques is significantly reduced. The security of in-cone logic locking against oracle guided attacks is, therefore, characterized. The analysis is then utilized to develop three gate selection algorithms that result in an increase

of the average number of iterations to complete an oracle guided attack by 61.8% when securing 5% of the logic gates of a netlist. The results demonstrate that while the security of in-cone logic locking techniques is improved, oracle guided attacks still prove effective due to the logical masking of key gates within the circuit.

To increase the security against oracle guided attacks, a sequential logic locking methodology utilizing hidden state transitions is developed. By reducing the information leakage of the structure of the circuit added to secure the IC, the key space for an adversary to search greatly increases. The larger search space results in an increase in the average expected time to determine the key from 48 hours to 1.64×10^{12} years for the ISCAS'89 s15850 benchmark circuit.

A methodology to increase the output corruption of out-of-cone logic locking techniques while maintaining a high level of security against oracle guided attacks is also developed. The methodology results in an increase in the number of iterations required to complete a SAT attack for a 20 input circuit implementing a flip function by 34.41x over SFLL-HD $n/4$ and 82.36x over SFLL-Flex.

Overall, the methodologies presented in this dissertation increase the security of logic locking while limiting the impact on power, performance, area, and design time. The presented work, therefore, allows for the addition of security as a design constraint, which provides a more secure cybersecurity landscape.

Chapter 1: Introduction

“I have always wished that my computer would be as easy to use as my telephone. My wish has come true. I no longer know how to use my telephone.”

– Bjarne Stroustrup

Technology continues to rapidly penetrate society, which, as Dr. Stroustrup suggests, has created more complex telephones. In addition to an increase in the difficulty of utilizing technology, the increased complexity of technology has also resulted in various security vulnerabilities [1]. The vulnerabilities receiving most of the attention are typically in the software layers of the computing stack, as shown in Fig 1.1. However, the evolution of both the integrated circuit and the manufacturing process has led to increased concern in the security of the hardware layers. Since the hardware layer provides the foundation for computing, security vulnerabilities in the hardware pose significant concern across all higher layers of the stack.

The hardware layer heavily relies on integrated circuits (ICs), which are ubiquitous in society today. Traditional uses of ICs, such as computers and cell phones, along with more non-traditional applications such as smart refrigerators and lightbulbs, provide entry points for adversaries to exploit. Ensuring the security of ICs, and therefore the hardware layer, is vital to a secure technological landscape.

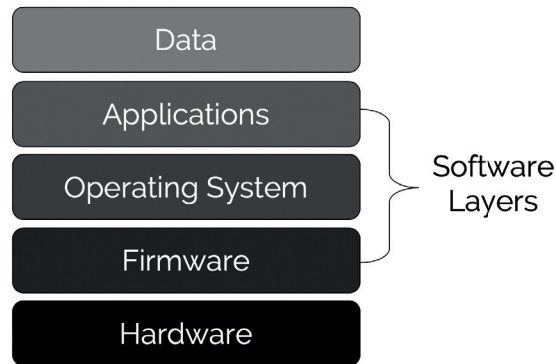


Figure 1.1: Overview of the typical computing stack.

The chapter begins with an introduction to ICs, providing background into the evolution of IC technology and drivers for the growth of the industry. The security vulnerabilities of ICs are then introduced, along with an overview of countermeasures that secure an IC.

1.1 The Evolution of the Integrated Circuit

The primary building block of all ICs is the transistor, where the fundamental functionality of the transistor is simplified to that of a switch. All complex functionality of an IC, such as for computer and cell phone processors, is the on/off control of these very fast switches.

The first transistor, a point contact transistor shown in Fig. 1.2a, is most commonly attributed to John Bardeen, Walter Brattain, and William Shockley at Bell Laboratories in 1947 [2–4]. The more common bipolar junction transistor was introduced just a year later by William Shockley in 1948 [2, 3, 5]. The point contact

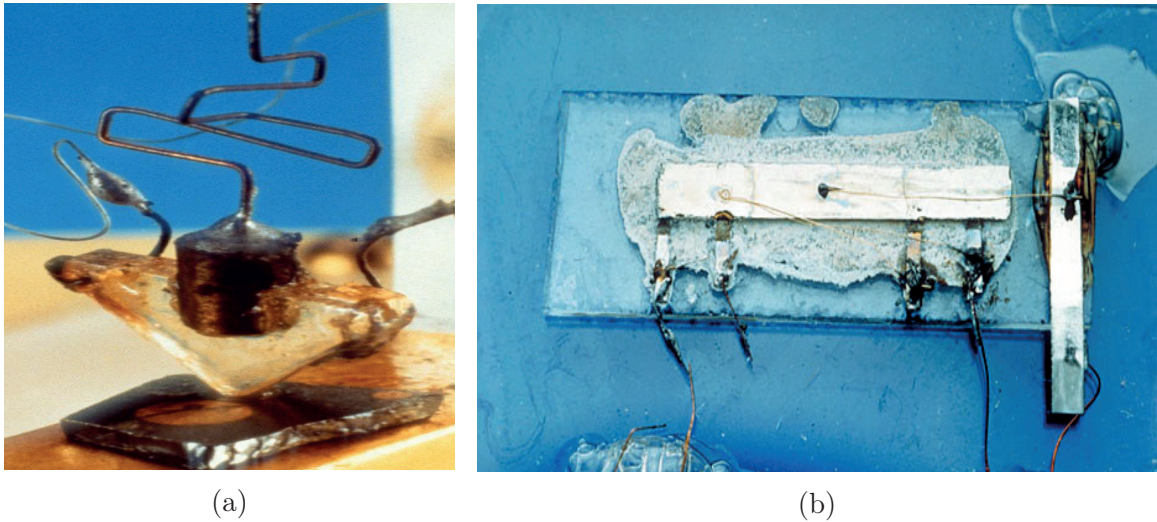


Figure 1.2: The (a) first point contact transistor developed by Bardeen, Brattain, and Shockley at Bell Laboratories in 1947 [2, 3] and (b) the first integrated circuit developed by Jack Kilby at Texas Instruments in 1958 [3].

transistor shown in Fig. 1.2a demonstrates that while the transistor was a monumental achievement, the size of the original device and the required wiring prevented the integration of many such transistors to form a single circuit.

As an employee of Texas instruments in 1958, Jack Kilby changed the paradigm of separately fabricating transistors by utilizing a single piece of material to integrate multiple devices [6–8]. Placing multiple devices on a single substrate credited Kilby with the production of the first IC. While the IC developed by Jack Kilby integrated multiple devices, external interconnects were needed to transmit signals between the transistors, as shown in Fig. 1.2b. In addition to Jack Kilby, Robert Noyce concurrently developed an IC at Fairchild Semiconductor that was publicly disclosed a few months after Kilby’s initial IC in 1959 [6–9]. The IC developed by Robert Noyce

integrated both the transistors and the interconnects within a single substrate to form a monolithic process, which is more similar to the fabrication process utilized to produce modern ICs [9].

1.1.1 Drivers of Integrated Circuit Development

Fairchild offered early commercial ICs in the 1960s, three of which are shown in Figs. 1.3a, Fig. 1.3b, and Fig. 1.3c. The small number of large size transistors in early ICs, which is evident as individual transistors are visible in the microphotographs, along with the cost to fabricate, limited the viable commercial applications. Only customers with critical size and weight constraints, such as the NASA Apollo Guidance Computer [10–12], were able to justify the cost. As process technology advanced, decreasing development cost and increasing functionality provided a catalyst for the commercial adaptation of ICs.

As the IC industry continued to advance, there was a near doubling of the transistor count each one to two years, which is often referred to as Moore’s Law, after Gordon E. Moore, who published a paper in the mid 1960’s describing the trend of increasing transistor count [18–21]. The trend is evident from the data plotted in Fig. 1.4, where there is a clear exponential increase in transistor count from 1971 to 2019.

The exponential growth in the number of transistors allowed for the integration of

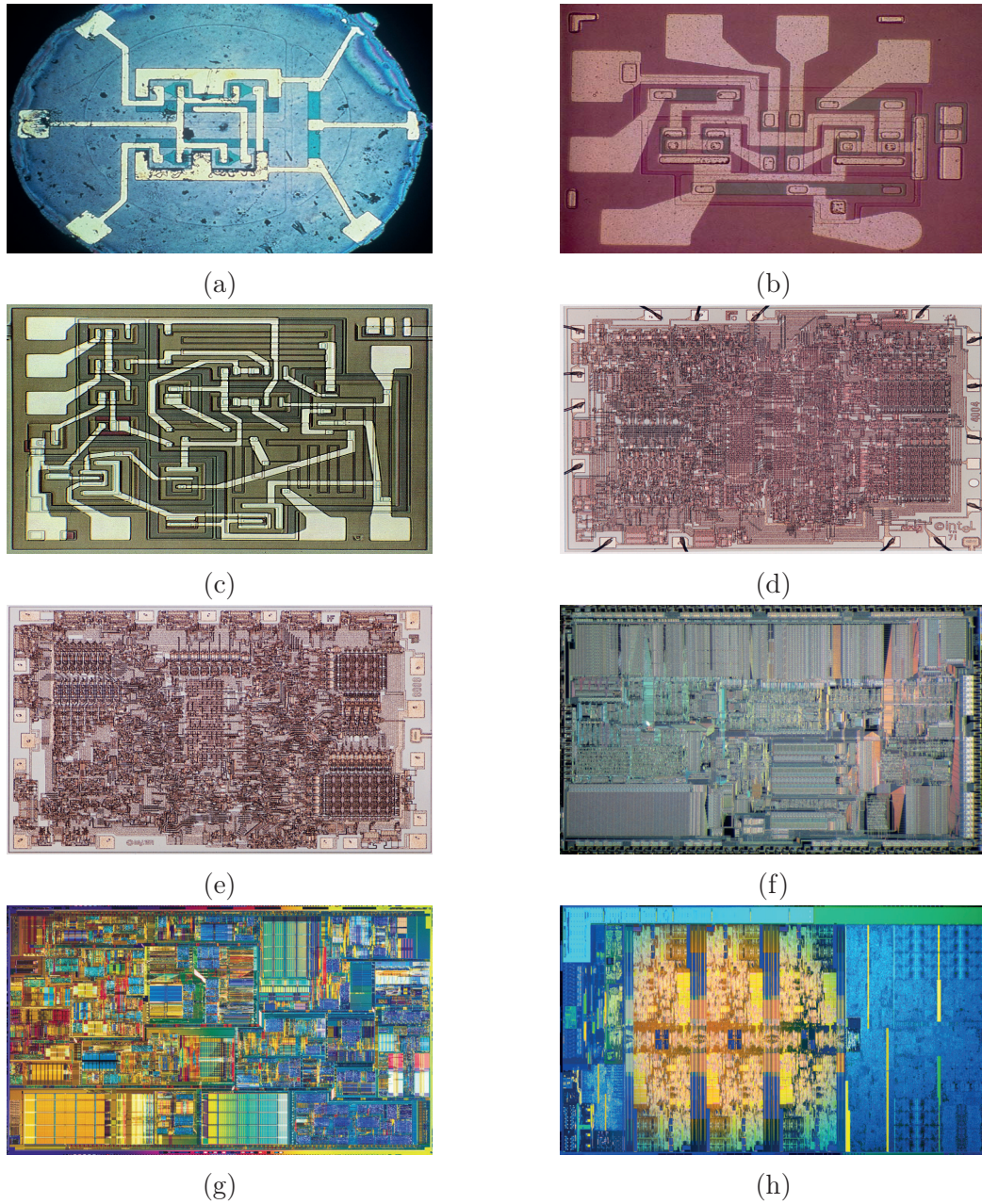


Figure 1.3: The evolution of the IC from the Fairchild integrated circuits of the 1960's (a)-(c), to early Intel processors including the (d) 4004, (e) 8008, and (f) 80386, to the Intel Pentium IV (g), and finally to the recent (h) Intel i7 Coffee Lake 6-core processor [13–17].

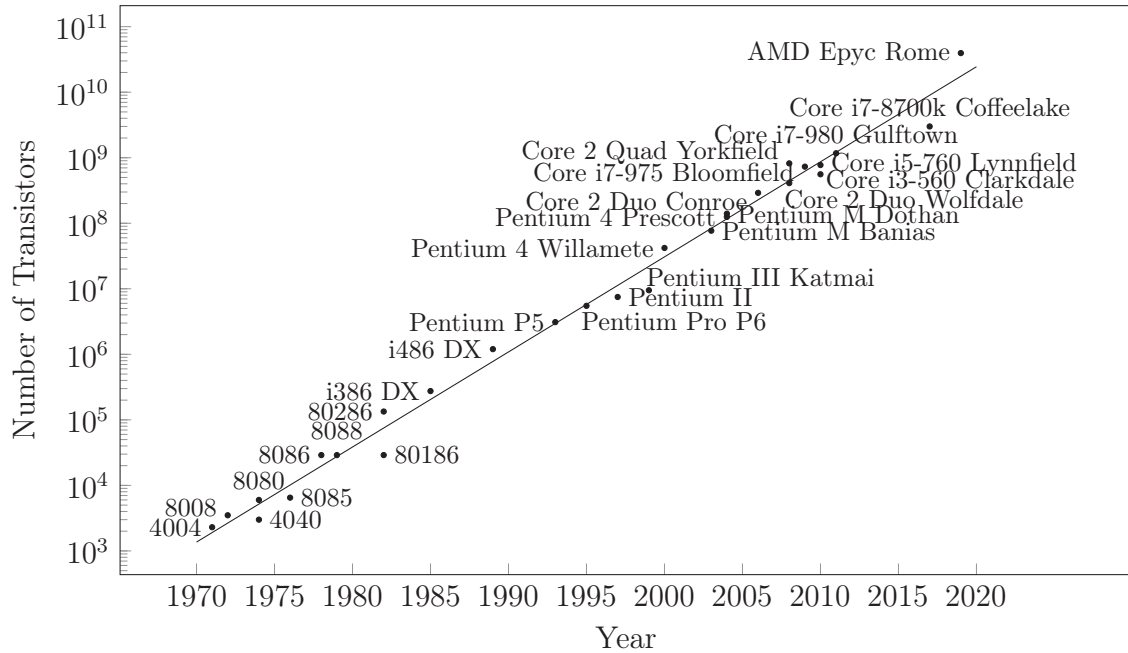


Figure 1.4: Number of transistors in select Intel processors from 1971 (Intel 4004) to 2017 (Core i7-8700k Coffeelake) [22–42] and the 2019 AMD Epyc Rome processor [43].

greater functionality, which resulted in an increase in the commercial market for ICs.

At a high level, the visual depiction of the increasing device density is evident from the ICs shown in Fig. 1.3, where only system level blocks are discernible in modern processors, such as shown in Fig. 1.3h, as compared to early ICs where individual transistors are discernable.

In addition to the number of transistors on an IC, the scaling of the fabrication process of the transistor provides significant benefits to other IC design criteria as well. One criterion that improved significantly is performance, as evident by comparing the 4700 MHz clock frequency of the Intel Coffee Lake Core i7-8700k 0.014

Table 1.1: Evolution of Intel microprocessors from 1971 to 2017 [22, 28, 36, 38, 42, 44].

Processor	Year	Process Technology (um)	Power Consumed (Watts)	Supply Voltage (Volts)	Maximum Clock Frequency (MHz)	Number of Cores
4004	1971	10	0.6	15	0.108	1
80186	1982	2	3	5	20	1
Pentium 4 Willamette	2000	0.18	75.3	1.75	1600	1
Core 2 Duo Conroe	2006	0.065	65	0.85-1.5	3000	2
Core i7-975 Bloomfield	2009	0.045	130	0.8-1.3	3700	4
Core i7-8700k Coffee Lake	2017	0.014	95	1.385-1.437	4700	6

μm process to the 20 MHz clock of the Intel 80186 fabricated in a 2 μm process, as listed in Table 1.1. In addition, until approximately 2006, process scaling allowed for transistors to not only become faster but also consume less power and occupy less area for each subsequent technology generation, which is often referred to as Dennard scaling [45]. The ability to offer better performance, power, area, and reduced cost drastically increased demand for ICs, signifying the main drivers of the semiconductor industry.

After 2006, novel circuit and architectural paradigms were introduced to further improve the performance and functionality of the IC. One such shift was an increase in the number of IC processor cores per die, with the early Intel Core 2 Duo processor containing two cores, while the more recent Intel i7-8700k includes six processing cores, as listed in Table 1.1. The additional transistor count of more modern ICs, such

as the i7-8700k, also resulted in increased application specific functionality, including the integration of a graphical processing unit (GPU), advanced encryption standard (AES) accelerator, and random number generator [44]. The focus on power, performance, area, cost, and functionality has undoubtedly lead to significant technological progress, however, the lack of security as a design objective leaves ICs vulnerable to a variety of threats.

1.2 Introduction to Hardware Security

The optimization of the power, performance, and area of an integrated circuit is becoming increasingly challenging to achieve as 1) scaling of transistor technology is approaching physical limits, or in other words, the ability to reduce the size of a transistor is becoming a challenge and 2) the complexity of ICs is increasing. As the functional complexity of an IC increases, design firms must ensure that the development cost of an IC is minimized in terms of both time and resources. Meeting strict time constraints requires a shift from a once vertical manufacturing process, where the entire IC design, fabrication, and packaging was completed by a single entity, to a horizontal process where multiple third-parties are involved in the design, fabrication, and test of an IC [46].

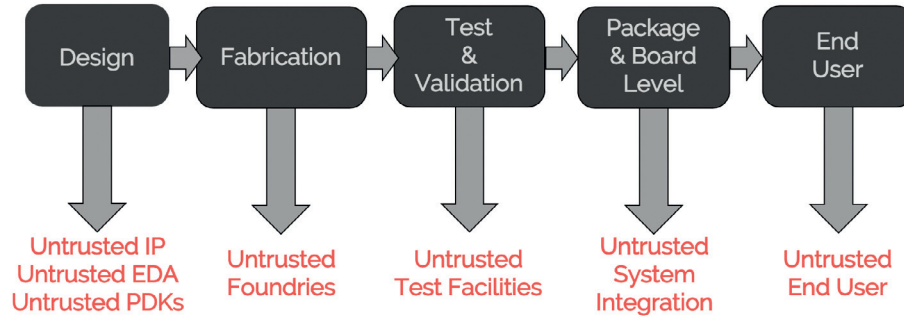


Figure 1.5: Typical IC development, manufacturing, and test flow along with associated threats at each stage.

1.2.1 Security Vulnerabilities of a Horizontal IC Design Flow

One third-party entity within the IC design and manufacturing process is the fabrication facility, which was formed due to the prohibitive multi-billion dollar investment required to establish an advanced fabrication node [47]. Another common strategy when developing modern ICs is to utilize third-party intellectual property (IP) in order to integrate advanced functionality without significantly increasing the IC design time [46]. In addition to third-party foundries and IP, third party tools for electronic design automation (EDA) and third party testing facilities are utilized. Finally, the ICs are potentially sold to untrusted end-users, which introduces a variety of security threats to an IC.

A horizontal IC design and fabrication flow is vulnerable to IP theft, counterfeiting and overproduction of ICs, and the insertion of harmful circuit modifications (hardware Trojans). The threat models require varying levels of access and knowledge

of the IC. In order to steal IP, an adversary must have access to an IC and be able to reverse engineer the netlist of the circuit, where the netlist describes the types of logic gates and the connections between gates. Counterfeiting an IC is distinct from IP theft as an adversary is not required to possess the netlist. The use of the IC while lacking possession of the netlist is referred to as a black-box attack [48]. Having physical access to the IC allows for the counterfeiting and resale of a circuit. The threat of overproduction describes an adversary that produces excess ICs, with the extra ICs sold for additional profit. In the case of overproduction, an adversary must have the complete design specification in order to fabricate the circuit. The insertion of hardware Trojans also requires that an adversary gain knowledge on the design of the circuit by either reverse engineering the netlist or through physical experimentation and analysis.

An overview of the various untrusted third-parties within the IC design, manufacturing, and test stages is shown in Fig. 1.5. One of the foremost threats currently facing the IC industry is the use of third-party fabrication facilities, as these facilities have access to the IC design and possess the required tools and knowledge to reverse engineer the IC from the provided masks used to produce the IC [49]. Having access to the photolithographic masks of an IC allows an untrusted foundry to steal IP, counterfeit and overproduce ICs, and insert hardware Trojans into the circuit. Another critical threat model to consider is an untrusted end-user, as IP theft and

counterfeiting are viable vulnerabilities when direct access to a fully functional IC and image based reverse engineering are available [49]. Untrusted packing and board level testing facilities are of a similar threat as an untrusted end-user, as a system is vulnerable to the replacement of the original ICs with cheaper or malicious components prior to delivery of the fully packaged circuit. The design stage, as shown in Fig. 1.5, includes unique challenges as third party EDA tools, IP, and process development kits (PDKs) utilized for the efficient design of an IC often contain black box components or algorithms that are used by a trusted IC designer. The lack of access to details regarding a component, tool, or process results in increased challenges of determining if any malicious functionality is included.

At the very least, the threats pose a serious concern in terms of financial loss. The estimated percentage of the IC semiconductor market effected by counterfeiting is between 7% and 10%, equating to an estimated financial loss of 22 billion dollars in 2014 [50]. The monetary concerns are overshadowed by the potential harm caused by an IC that does not function to design specifications. An IC that fails to meet the original specifications results in increased system failure rates, produces logical errors, and potentially includes hardware Trojans embedded in the circuit. Hardware Trojans provide additional challenges as many different attack vectors exist, including denial of service, stealing information, and/or corrupting functionality [51]. The challenge in security then becomes protecting against the wide variety of possible threats.

1.2.2 Countermeasures to Security Threats

There are a variety of countermeasures that have been proposed for the threats described in Section 1.2.1. Overall, the countermeasures are divided into two categories: 1) attack detection and 2) attack prevention. Techniques for detection are based on either on-chip or off-chip characterization to determine if a malicious modification was made to a circuit. Techniques for prevention typically modify the structure or topology of an IC to increase the difficulty an adversary faces to execute an attack.

1.2.2.a Detection

Both hardware Trojan detection and IP watermarking fall into the category of detection. There are a variety of methods to detect foreign hardware or activity, including adding circuits to increase the switching activity of components [52–54], utilizing path delays [55–57], utilizing built in self-test [58, 59], and using side-channel analysis [60–76].

The most prominent form of hardware Trojan detection is side-channel analysis, where power, delay, electromagnetic (EM), and/or thermal information is utilized to detect abnormalities in circuit functionality. For example, consider the noise signature generated by a NAND gate as shown in Fig. 1.6. The difference in the voltage as the various combinations of inputs are applied to the NAND gate provides a side-channel that allows for the prediction of the executing operation of the circuit. Any

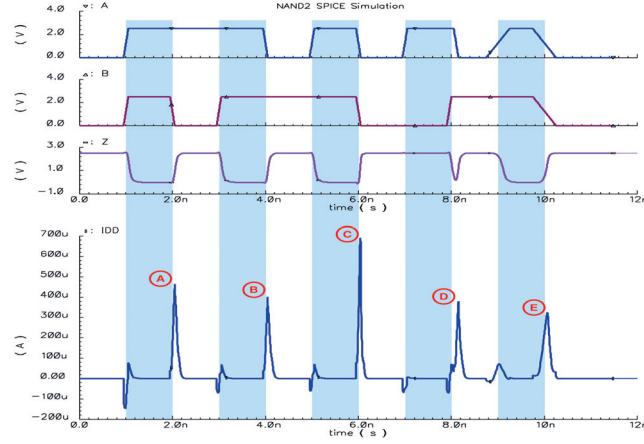


Figure 1.6: Input dependent current noise generated by a NAND gate on the power distribution network [77].

malicious modification to the IC potentially alters the noise signature, which allows for the detection of the malicious component. For example, the effects of a hardware Trojan integrated with a NAND gate are shown in Fig. 1.7. The transition of both the Hardware Trojan and NAND gate from logic one to logic zero reveals the insertion of malicious functionality.

While the example in Fig. 1.7 allows for easy detection of the hardware Trojan, the original circuit and the Trojan circuit are of equal size. When the original circuit is significantly larger than the Trojan circuit, the difference in voltage noise on the power network due to the activity of the Trojan becomes much more difficult to detect. Referring to Table 1.1, modern ICs include billions of transistors, whereas a hardware Trojan is able to maliciously alter an IC with only a single transistor [78]. The problem is akin to finding a needle in a haystack, as abnormalities in a given signature are not easily detected. Process, voltage, and temperature variations

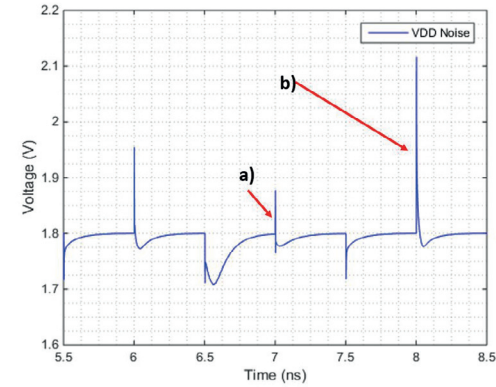


Figure 1.7: Increased voltage noise on the power rail caused by activation of a hardware modification when a) the non-Trojan NAND switches from 1-0, and b) both the Trojan and non-Trojan NAND switch from 1-0.

also increase the difficulty of detecting a hardware Trojan as the Trojan free circuit response also varies over time and for each IC produced.

IP watermarking introduces a signature to an IC that is ideally not detected by an untrusted foundry, but at the very least is difficult to physically remove from the IC [79]. The two categories of watermarking techniques are described as 1) constraint-based watermarks and 2) finite state machine (FSM) watermarks. A generic constraint based watermarking technique is introduced in [80]. The technique is not limited to a specific circuit parameter, but rather parameters such as clock frequency, restrictions to the functionality of logic gates, or placement of logic gates allow for designs to be generated with a unique signature. Since no specific imposed constraint on the circuit is required, the added constraint is difficult for an adversary to determine.

Another method of inserting a watermark into a circuit is through modifications to the FSM. The work described in [81, 82] both present methods that create a

unique identifier based on added sequences to an FSM. The work in [82] implements a watermark utilizing the unused transitions of a state machine graph (STG) and inserting a new sequence through the state space based on the unused transitions. In [81], a unique string is generated that serves as an identifier of the circuit. The STG is then modified to include a transition to a specific state when the identification sequence is applied [79]. Watermarking techniques provide a way to detect if IP is stolen, counterfeited, or overproduced, but do not offer any means to prevent such threats.

1.2.2.b Prevention

The alternative to detection is to modify the IC to prevent a set of threats. Techniques that alter a circuit to prevent security vulnerabilities predominantly fall into the category of obfuscation. Techniques that obfuscate a circuit mask portions of the functional and structural information of an IC from an adversary. Possible circuit properties to obfuscate include the connections between logical gates and the functionality of the logic gates, with an overview of split manufacturing, IC camouflaging, and logic locking techniques provided in Chapter 2. By obfuscating portions of the circuit, IP theft, IC counterfeiting and overproduction, and hardware Trojan insertion become more difficult to execute. The remainder of the thesis focuses on attack prevention methodologies due to the potential to protect against a wide variety of threat vectors.

1.3 Thesis Outline

The primary focus of this research effort is the development of secure and low-cost, in terms of overhead in power, performance, and area, logic locking methodologies. Logic obfuscation is introduced in Chapter 2, with a discussion of split-manufacturing, IC camouflaging, and logic locking. The content described in Chapter 2 emphasizes in-cone logic locking methodologies as well as techniques to optimally select locations to place logic locked gates or interconnects within the circuit. An overview of attack vectors on logic locked circuits is presented in Chapter 3 with a discussion on sensitization attacks, hill climbing attacks, and satisfiability (SAT) solver based oracle guided attacks.

An introduction to out-of-cone logic locking techniques is provided in Chapter 4. The out-of-cone techniques aim to address the vulnerabilities of in-cone logic locking to the attacks described in Chapter 3 by adding a circuit feedback path that strategically corrupts, or corrects, the original primary outputs of the IC. Attacks on out-of-cone logic locking techniques are also described in Chapter 4.

Background on sequential logic locking is provided in Chapter 5. Techniques that partition the state machine into an obfuscated and functional mode are described. In addition, methods that generate incorrect transitions within the original state machine are discussed. Attack vectors on sequential logic locking techniques are also presented in Chapter 5.

Reducing the overhead of in-cone logic locking techniques through the development of novel logic structures that allow for increased performance, reduced power, and reduced area overhead as compared to XOR or look-up table (LUT) based in-cone logic locking methodologies are discussed in Chapter 6. In addition, a technique to expand the search space of a key sequence is described, which increases the difficulty for an adversary to extract the key. For a locked AND gate, a reduction in power of 41.50%, an estimated reduction in area of 43.58%, and an increase in performance of 34.54% is achieved when using the developed gate level logic locking technique instead of the LUT-based locking technique described in Chapter 2. The developed logical structures reduce the cost of implementing in-cone logic locking, but do not address the SAT based attack vectors described in Chapter 3.

A characterization of methods that increase the resiliency of in-cone logic locking against SAT based attacks is provided in Chapter 7. The chapter begins with an analysis of the parameters of the SAT solver that impact security and a characterization of the effect netlist topology has on the overall security of the circuit. The analysis of SAT solver parameters and logic structure is used to develop three novel logic locking gate selection algorithms based on maximum fanout free cones (MFFCs) and gate controllability. Applying the MFFC algorithms with XOR, LUT, and 2x1 MUX based obfuscation resulted in an average increase of, respectively, 61.8%, 123.6%, and 38.5% in the minimum number of iterations required to complete the

SAT attack across 1,000 different variable orderings of the netlist while applying the locking techniques to 5% of the gates of the circuit. In addition, the resiliency to the SAT attack and the corruption of outputs when applying the developed algorithms is compared with out-of-cone locking techniques, with results indicating that in-cone techniques are only able to provide greater security for limited cases. The topologies that are better suited for in-cone methods include circuits with a limited number of inputs and with logic cones of large logical depth.

A methodology to increase the security of sequential logic locking is presented in Chapter 8. Specifically, hidden state transitions and logic cone modifications are utilized to further enhance the security of sequentially locked circuits. An algorithm to insert hidden transitions and logic modifications into a netlist is developed that results in an average overhead of 6.05% in area, 10.30% in power, and 8.97% in performance across all of the ISCAS'89 sequential benchmark circuits. To modify the logic cone with two altered minterms, the average increase in area and power, beyond what is needed for the implementation of hidden state transitions, is 26.46% and 30.30%, respectively, with no additional overhead in performance. The developed algorithm that implements hidden state transitions for partitioned state machines is the only current algorithm that protects against extraction of the transition between the obfuscated and functional mode of the FSM, which results in an increase in the number of paths an adversary must search to determine the key sequence that

activates the circuit.

A novel out-of-cone logic locking methodology described as CORruption Adaptable Logic Locking (CORALL) is presented in Chapter 9. CORALL provides increased resilience to the SAT attack for circuits that require a large corruption of outputs between the original logic cone and the modified logic cone of an obfuscated version of the circuit. The CORALL architecture results in an increase in the number of iterations required to complete a SAT attack for a 20 input circuit implementing a flip function by 34.41x over SFLL-HD $n/4$ and 82.36x over SFLL-Flex. CORALL provides a large increase in the resilience against the SAT attack while also increasing the percentage of output corruption by at least 40% over SFLL-HD $n/4$ and SFLL-Flex. The improvement in the resilience of a circuit to the SAT attack is provided with similar overheads in area and timing as compared to SFLL-HD. A 20% reduction in area and a 2.14% increase in performance on average was observed as compared to SFLL-HD when implementing the CORALL architecture utilizing four protected cubes and a LUT size of four. An algorithm that generates protected cubes by applying iterative cofactors is also presented, which increases the resilience of the circuit to the execution of structural attacks on the modified logic cone. The varying implementations of the logic structures of the perturb unit and the mapping of the logic of the perturb unit into the LUTs of the CORALL architecture provide protection against all known structural attacks.

A summary of the primary contributions of this dissertation is provided in Chapter 10. The primary goal of this research effort is the development of secure and low-cost logic locking methodologies. The dissertation describes methods to generate low overhead logic locking gates, characterize in-cone logic locking against the SAT attack, and develop sequential and combinational logic locking techniques that are highly resilient to SAT attacks while minimizing the overhead in power, performance, and area when implementing the techniques on an unobfuscated IC. Therefore, the material presented in this dissertation provides design insight to properly secure integrated circuits, which is vital to secure the cybersecurity root of trust.

Future research directions are provided in Chapter 11. Research directions into evaluating the security of logic locking, validating the security of tamper proof memory, and developing efficient algorithms to apply subtractive logic cone modifications are discussed. Future work on the proposed areas will provide more secure and accessible logic locking methodologies.

Chapter 2: Logic Obfuscation Techniques

The overarching goal of obfuscation is to limit the amount of information recoverable by an adversary. A comparable scenario is removing pieces from a puzzle to obscure the image. If too few pieces are removed, or the pieces are of a unique shape, an adversary is able to recover the puzzle image. Therefore, logic obfuscation techniques must carefully consider which portions of the circuit are to be obscured and if the added security features leak information to an adversary. This chapter provides an overview of the methodologies to obfuscate an IC, including 1) split manufacturing, 2) IC camouflaging, and 3) logic locking. The associated threat models and security limitations of the techniques are also discussed. The chapter concludes with a comparison of the provided security and circuit overhead when implementing the three obfuscation methodologies.

2.1 Split Manufacturing

Split manufacturing divides the fabrication of an IC into two portions, one of which is completed by a trusted foundry while the other is completed by an untrusted third-party foundry. A cross sectional view of an IC divided into front end of the line (FEOL) layers and back end of the line (BEOL) layers is depicted in Fig. 2.1a.

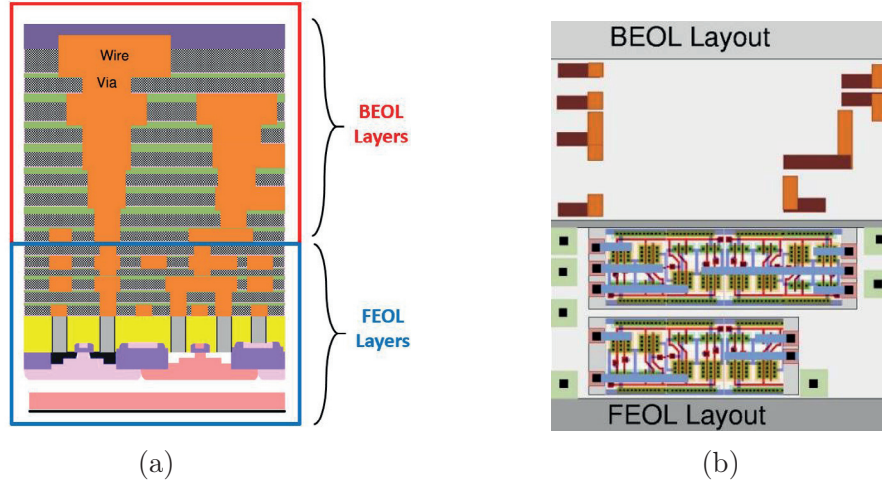


Figure 2.1: Partitioning of an IC for split manufacturing. The a) cross sectional view and b) top view of a partitioned circuit are shown [83].

The FEOL layers include the advanced process technology features, such as the transistors and often metal layers 1 and 2, while the BEOL layers are fabricated in older process nodes and include higher level metal layers. Allowing the untrusted foundry to only produce the FEOL obfuscates information about connections between logical elements within the IC, which prevents an untrusted foundry from knowing the entire functionality of the circuit.

The primary security threat with split manufacturing is the amount of information provided by the FEOL layers [83], which an untrusted foundry has access to. The information leakage primarily occurs due to the proximity between connected components, which is utilized in [83] and [84] to determine the missing connections of the BEOL. The proximity of connected components is the result of EDA design tools optimizing the area, power and performance of the circuit.

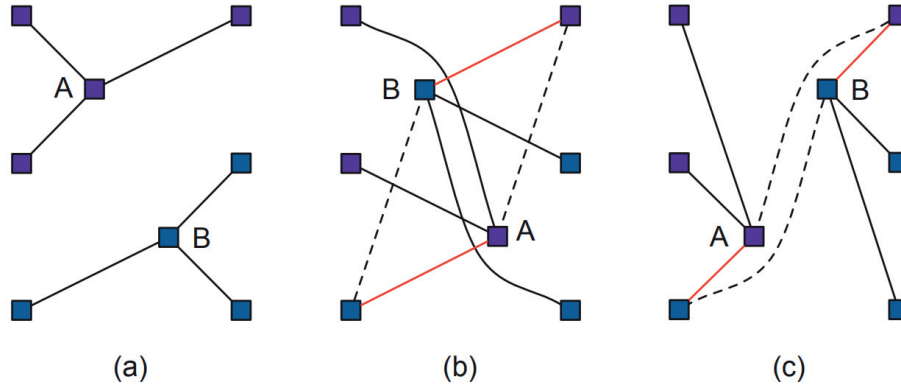


Figure 2.2: Application of the placement perturbation algorithm where (a) depicts the original unmodified placement, and (b) gate A and gate B are swapped. Dashed lines represent the missing connections to the BEOL and red lines indicate the gates with the closest proximity to A and B, respectively. Wirelength is further reduced from 70% overhead by the placement shown in (c) by applying a smaller perturbation to gates A and B [85].

One source of information leakage is the physical direction of the unconnected interconnects in the FEOL. An unconnected vertical or horizontal interconnect provides unintended information regarding the direction to search for the correct connection point of the dangling interconnect. Creating only horizontal or vertical connections in a given metal layer, such as shown in Fig. 2.1b, is one potential approach to reduce the amount of information leaked regarding the connections of the interconnects.

The proximity of the unconnected interconnect to the devices in the FEOL also provides an adversary with information since one objective of EDA algorithms is to minimize wirelength. To reduce the amount of information leaked due to the proximity of devices, a placement perturbation algorithm is utilized in [85] that shifts interconnects such that the nearest FEOL devices and corresponding wires are not

always connected through the BEOL metal layers. An example of applying the technique is shown in Fig. 2.2, where the original connection shown in Fig. 2.2a is modified such that gate A is closer to the FEOL connection location meant for node B as shown in Fig. 2.2b. To further minimize wire length, the gate placement shown in Fig. 2.2c is applied, which reduces the wirelength from 70%, as produced by the gate placement shown in Fig. 2.2b, to 30% [85]. Preventing the overhead in wirelength due to the false connections is also important as the adversary is able to deduce the operating frequency of the IC from the clock source. Therefore, any connections that result in a timing violation or cyclical logic connection within the IC allow an adversary to eliminate possible connections [85].

2.2 IC Camouflaging

IC camouflaging aims to obfuscate a circuit by inserting gates that exhibit the same structure as multiple different logical functions. Therefore, an adversary reverse engineering an IC observes a cell that is classified as a set of functions, rather than one specific gate function. For example, consider the NAND and NOR gates, where each is depicted in Fig. 2.3a, designed to appear identical to the other. An adversary is not privy to whether the correct functionality is that of a NAND or NOR for the given gate placement.

One methodology to camouflage an IC is to utilize dummy contacts, as shown in

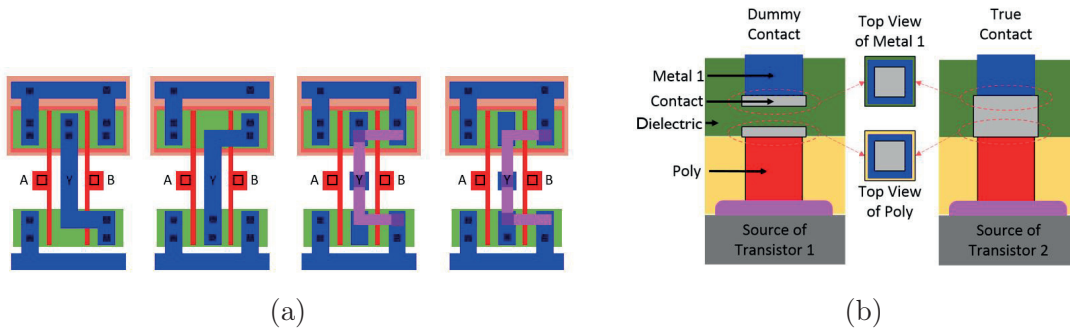


Figure 2.3: IC camouflaging of a gate with dummy contacts, where a depiction of a) the top view of standard NAND and NOR gates and camouflaged NAND/NOR gates, and b) the cross section of dummy and true contacts are provided [86].

Fig. 2.3b. When delayering the IC, an adversary only observes the top view of the metal and poly layers. Assuming the limited resolution of the delayering process, an adversary is not able to discern between a true contact and a dummy contact.

The primary limitation of IC camouflaging is due to the difference in physical appearance between camouflaged and non-camouflaged gates. Since camouflaged gates result in significant overhead in power, performance, and area, only a subset of the gates of an IC are camouflaged [86]. An adversary is, therefore, able to replace the camouflaged gates with a choice between the set of possible standard gates. An oracle-guided attack utilizing a satisfiability (SAT) solver is, therefore, possible [87], which are discussed in greater detail in Chapter 3.

2.3 Protecting Against a Diverse Threat-Model

As discussed in Section 1.2.1, the threat space for hardware security is diverse. An untrusted foundry, testing facility, and end-user all are potential security vulnerabilities. Split manufacturing protects against a malicious foundry, however, there is no means to thwart a malicious end-user. As such, an end-user has the ability to reverse engineer and counterfeit an IC without any additional obstacles. IC camouflaging protects against an untrusted end-user, but the foundry must be aware of the true functionality of each camouflaged cell, or else the produced circuit will not function as intended. Therefore, an untrusted foundry is able to steal and over-produce ICs. In addition, the end-user is able to counterfeit the IC as the functionality of the circuit is not impacted by the implementation of camouflaging.

An obfuscation methodology that provides security against IP theft, IC counterfeiting and overproduction, and hardware Trojan insertion must, therefore, follow certain criteria: 1) The foundry must not have access to the entire design specification, and 2) the circuit cannot be used as a black-box. If the foundry does not have the entire design specification, the difficulty to reverse engineer the circuit and/or insert Trojans increases significantly. If the circuit cannot be used as a black-box, the possibility to overproduce and counterfeit the IC is substantially lower as the correct functionality is not achieved without input from a trusted party.

2.4 In-Cone Logic Locking

Logic locking requires the application of a key to locked logical gates to allow for the correct operation of the IC. The key is not known by the untrusted foundry or end-user without being released by the designer. Requiring a key within the IC to activate correct functionality satisfies the security requirements described in Section 2.3, as the integrated circuit is not completely released to the foundry and black-box usage is not possible without the key. Logic locking, therefore, provides protection against IP theft, IC counterfeiting and overproduction, and hardware Trojan insertion. An overview of the different methodologies, including strategies to optimally select nodes to apply logic locking, is discussed in this section.

2.4.1 In-Cone Logic Locking Methodologies

The three primary in-cone logic locking methodologies are XOR-based [88], multiplexer (MUX)-based [89], and look-up table (LUT) based [90]. While the implementation of the three techniques varies, all three methodologies require a key as input that, when incorrect, results in the erroneous functionality of the IC.

2.4.1.a XOR-Based Logic Locking

XOR-based logic locking is implemented by inserting an XOR/XNOR gate after a selected gate within a netlist, with a key bit as the second input to the XOR gate. An

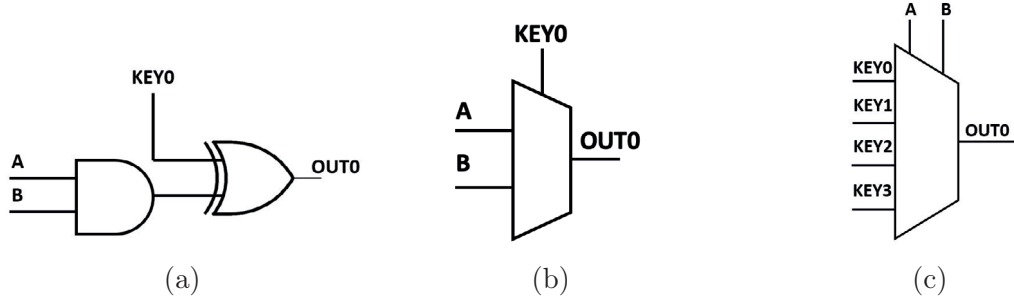


Figure 2.4: In-cone logic locking methodologies, including (a) XOR-based [88], (b) MUX-based [89], and (c) LUT based [90].

example of locking an AND gate with an XOR gate is depicted in Fig. 2.4a. When KEY0 is 0, the XOR gate acts as a buffer, whereas when KEY0 is 1 the XOR gate acts as an inverter [88]. The KEY0 input to the XOR gate then controls the output OUT0, which permits correct functionality only when the correct key input, logic 0, is applied. The same concept is applicable to an XNOR with only a simple inversion to the key input of the gate.

One concern with logic locking based on XOR/XNOR gates is that the correct key is known when only an XOR/XNOR gate is inserted [89]. For example, observe in Fig. 2.4a that the correct functionality of the AND gate is achieved when KEY0 is set to 0. To prevent an adversary from determining the key bit, an inverter is added before or after the key gate to further obscure the key. The goal is to bubble push the inverter away from the key gate so that an adversary is unaware of the addition of the inverter [89].

2.4.1.b MUX-Based Logic Locking

MUX-based logic locking utilizes the select inputs of a MUX as the key inputs, as shown in Fig. 2.4b for a 2x1 MUX. The A input in Fig. 2.4b is then the original net and B is a “false” input from another gate in the circuit. The inherent benefit of the 2x1 MUX-based locking technique as compared to the XOR-based technique is that no additional inverters are needed to mask the key bits [89]. The challenge of implementing the 2x1 MUX-based logic locking technique is selecting the false input, as inputs to the MUX must be logically different from one another a majority of the time to be effective. If the inputs to the MUX are of the same value for a high percentage of the time, then a circuit with an incorrect key will not often differ from an activated circuit with the correct key applied.

2.4.1.c LUT-Based Logic Locking

LUT based logic locking replaces a gate within the original netlist with the 4x1 MUX structure shown in Fig. 2.4c, which allows for the implementation of any 2-input function. The logical function is implemented by setting the key inputs to target values. For example, if KEY0, KEY1, and KEY2 are set to logic 0, and KEY3 is set to logic 1, then an AND gate is realized.

The LUT-based methodology provides two significant benefits over the XOR and MUX-based methodologies. First, the key space is larger as each locked gate requires

more than one key bit. The larger key space results in an increase in the number of key combinations an adversary must attempt to deduce the correct key combination. Second, the original gates in the circuit are replaced by the reconfigurable LUTs [90]. Replacing the original gates also offers a distinct advantage over XOR-based logic locking, as a portion of the gates are replaced with unknown functions, in contrast to XOR-based locking which adds an XOR gate to the circuit, but does not remove the original gate.

The cost for the larger key space comes in the form of a greater overhead in power, performance, and area. Given a target overhead in power, area, and performance, XOR-based logic locking allows for the insertion of a greater number of gates into a circuit, which obfuscates a larger portion of the IC. An additional consideration is that the original gate and a LUT that includes an incorrect key often share some common functionality, which decreases the relative corruption of the output when comparing between an activated and locked IC.

2.4.2 In-Cone Logic Locking Gate Selection Algorithms

In addition to the utilized logic locking technique, the selection of the nets within an IC to lock is also critical. There are various gate selection algorithms utilized for in-cone logic locking including 1) randomly selecting the nets within the circuit based on the number of gates or overhead budget [88], 2) selecting gates to reach a desired

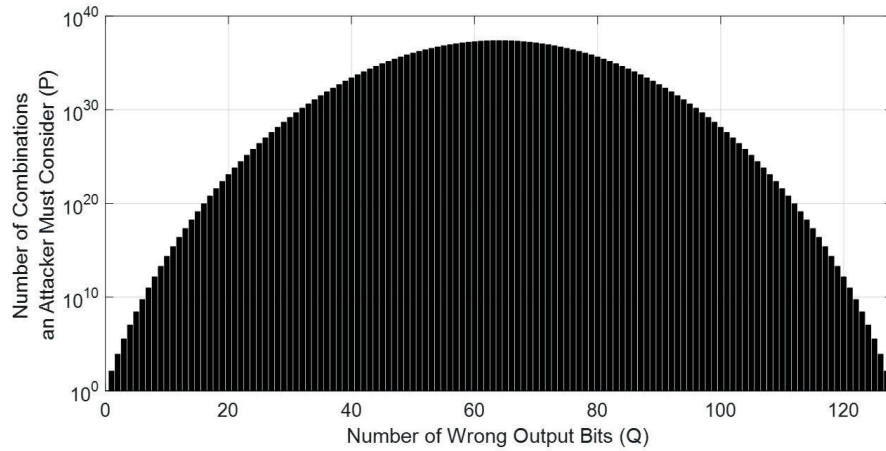


Figure 2.5: Number of combinations P that an attacker must consider when a key of size N is 128 for different values of incorrect bits Q [89].

level of output corruption [89, 90], and 3) selecting gates to limit key sensitization attacks [91].

2.4.2.a Fault Analysis-Based XOR Gate Selection

The criteria to select which gates to apply locking on with the XOR/XNOR methodology described in [89] is based on choosing locations that produce a 50% Hamming distance. The Hamming distance, as relating to logic obfuscation, is the number of bits that differ between the correct circuit output and an output when an incorrect key is applied. The methodology is such that the ambiguity an attacker sees at the output of an IC is maximized. If the Hamming distance between an incorrect output and a correct output is zero, then the attacker sees no ambiguity between a locked and un-locked IC. Similarly, when the Hamming distance is equal to the

number of output bits, the attacker is able to flip all bits to produce the correct output.

A combinatorial example, $\binom{N}{Q}$, is shown in Fig. 2.5, where N is the number of outputs and Q is the number of incorrect outputs. The maximum number of combinations occurs when $Q = N/2$, which is equivalent to a 50% Hamming distance, and, therefore, motivates corrupting 50% of all outputs for every input pattern [89].

A fault analysis-based methodology is proposed in [89] to satisfy such a metric, and is based on three key properties: 1) Fault excitation, 2) fault propagation, and 3) fault masking.

1) Fault excitation: When applying an incorrect key, an incorrect output is generated at the location of the key gate. The incorrect output is modeled as a stuck-at-0 (s-a-0) or stuck-at-1 (s-a-1) fault. For the obfuscated AND gate shown in Fig. 2.4a, applying the inputs $A = 0$, $B = 1$, and $KEY0=1$, the locked gate produces an output of 1. The correct operation of the circuit results in an output of 0 when $A = 0$ and $B = 1$, which implies that the incorrect output is modeled as a s-a-1 fault.

2) Fault propagation: In addition to generating a fault at the output of a locked gate, the observability of the fault at the output of the IC is also important. A fault does not propagate if a controlling input is applied to a down-stream gate. For example, consider an OR gate with an input A connected to a fault from a locked gate and input B connected to a non-obfuscated node. If input B is 1, the output

of the OR gate is 1 independent of input A . The fault no longer propagates towards the outputs of the circuit.

3) Fault masking: An extension to fault propagation is fault masking, which considers the effect of applying multiple incorrect key bits on a given output bit. One fault from a locked gate, therefore, potentially masks the fault from another locked gate. For example, consider the AND gate shown in Fig. 2.4a, where input A is also locked with an XOR-gate along with the output. If the key applied to the A input and $KEY0$ at the output are incorrect and input B is a logic 1, then the faults produced by the two XOR gates cancel each other out.

A fault simulation tool is utilized to model the effect of faults, but the primary challenge is that the tools are only capable of modeling a single stuck-at-fault. The limitation of the fault model is addressed by implementing a greedy and iterative based insertion method that inserts one key gate at a time [89]. After the first iteration, the key bits are set to random incorrect values to limit the effect of fault masking.

A fault impact criteria is introduced in [89] to evaluate the effect of locking a single gate, which is provided as (2.1). The $N_O P_0$ term represents the number of patterns that detect the s-a-0 fault, while $N_O O_0$ represents the number of output bits altered by the fault. The effect of a s-a-1 fault is also accounted for in (2.1).

$$Fault\ Impact = (N_O P_0 * N_O O_0 + N_O P_1 * N_O O_1) \quad (2.1)$$

The fault impact criteria is then applied, which is a sub-routine of Algorithm 1. There are two primary phases to the algorithm. The first is the location selection, which computes the impact of the fault for each gate in the netlist and selects the gate with the highest fault impact. The second, a modification phase, inserts inverters into the circuit based on a generated random number and replaces certain XOR gates with XNORs.

In addition to the XOR based locking methodology, a fault-analysis selection algorithm for 2x1 MUXes is also introduced in [89]. In order to ensure the true and false inputs to the 2x1 MUX differ for a significant percentage of the input patterns of the circuit, a contradiction metric, as given by (2.2), is introduced.

$$Contradiction\ Metric = (P_{0,true} * P_{1,false}) + (P_{1,true} * P_{0,false}) \quad (2.2)$$

The probability of a 0 or 1 on the true input is represented as, respectively, $P_{0,true}$ and $P_{1,true}$. Algorithm 1 is modified to include the contradiction metric, which results in Algorithm 2. The execution of the algorithm, to determine the false input for a single locked gate, now requires a computation of the contradiction metric for each net in the circuit that does not form a combinational loop with the net being locked.

Algorithm 1: Fault analysis-based insertion of XOR/XNOR gates [89]

Input: Netlist, KeySize, EncryptionKey
Output: Encrypted Netlist
// Location Selection Phase
for $i \leftarrow 1$ **to** $KeySize$ **do**
 foreach $gate_j \in Netlist$ **do**
 Compute FaultImpact;
 end
 Select the gate with highest FaultImpact;
 Insert XOR gate and update the Netlist;
 Apply the Test Patterns;
end
// Modification Phase
Generate R; *// a random number*
foreach $bit\ i \in R$ **do**
 if $i == 1$ **then**
 Insert an inverter at the o/p of the corresponding key-gate;
 end
end
GateType = EncryptionKey $\oplus$ R;
foreach $bit\ i \in R$ **do**
 if $i == 1$ **then**
 Replace the XOR key-gate with an XNOR key-gate;
 end
end

The computation of the contradiction method adds $O(n)$ time complexity [89] to an already time consuming algorithm. For the 2x1 MUX obfuscation technique, a random Boolean value is chosen and the true and false inputs are flipped accordingly during the modification phase of the gate selection algorithm to switch the value of the key input.

The fault analysis-based insertion method achieves a Hamming distance of 50% at the outputs in much less time than a random insertion methodology, as indicated

Algorithm 2: Fault analysis-based insertion of MUXes [89]

Input: Netlist, KeySize, EncryptionKey
Output: Encrypted Netlist
// Location Selection Phase
for $i \leftarrow 1$ **to** $KeySize$ **do**
 foreach $gate\ j \in Netlist$ **do**
 Compute FaultImpact;
 end
 Select the gate with the highest FaultImpact as the *true* wire;
 List of false wires = Φ ;
 foreach $wire\ j \in Netlist$ *and* $j \neq true\ wire$ **do**
 if $CombinationalLoop(j, true\ wire) == False$ **then**
 ListOfFalseWires = ListofFalseWires $\cup j$;
 Compute Contradiction metric;
 end
 end
 Select the wire with the highest contradiction metric as false wire;
 Insert MUX and update the Netlist;
 Apply Test Patterns;
end
// Modification Phase
foreach $bit\ i \in EncryptionKey$ **do**
 if $i == 1$ **then**
 Connect *true* and *false* wires to the second and first inputs of the MUX;
 end
end

by the results shown in Fig. 2.6. The results from the analysis of random insertion of XOR and MUX gates is shown, respectively, in Fig. 2.6a and Fig. 2.6c. The results from the characterization of the fault analysis-based insertion methodology are shown in Fig. 2.6b and Fig. 2.6d for insertion of XOR and MUX gates, respectively. Note the increasing challenge of achieving a Hamming distance of 50% for the 2x1 MUX logic locking methodology.

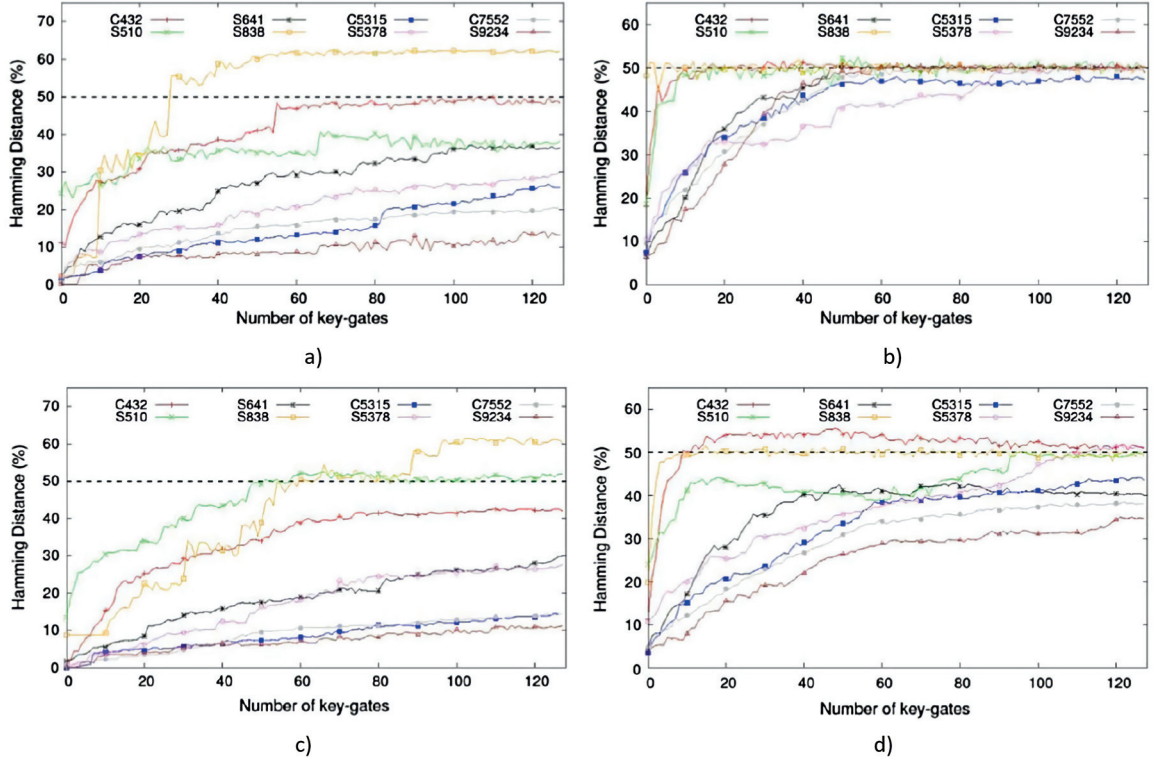


Figure 2.6: Hamming distance of the outputs of ISCAS benchmark circuits when applying the correct key and a random incorrect key for a) random insertion and b) fault analysis-based insertion of XOR gates, and c) random insertion and d) fault analysis-based insertion of MUX gates [89].

Reducing the number of key gates to achieve a 50% Hamming distance results in improved overheads in power, area, and performance when securing an IC with logic locking. The average overhead in power of randomly inserting XOR gates to achieve a 50% Hamming distance is 46.8% as compared to an unobfuscated equivalent circuit, while the overhead in power when applying the fault analysis-based method is 25.9% [89]. Similarly, the overhead in delay of the random XOR based method is 56.9%, while logic locking implemented with the fault analysis-based methodology results

in a 46.3% penalty in delay. Finally, the overhead in area of the random insertion method on the ISCAS benchmark circuits is 80.3%, while the fault analysis-based technique requires 65.3% greater area [89].

2.4.2.b Observability Don't Care Gate Selection

The work in [90] also utilizes the 50% Hamming distance criteria described in Section 2.4.2.a. However, the gate selection algorithm provided in [90] differs from the selection criteria described in [89].

The selection algorithm described in [90] relies on node observability. Observability is a measure of the ability to discern the digital value of an internal node at a primary output and is evaluated with observability don't care (ODC) values [90]. Controllability is also considered in the algorithm, which is a measure of the ability to control the value of a node from the primary inputs of the circuit. In addition, cut height is used in [90] as a heuristic metric to select nodes, which is a measure of the shortest simple height or nodal distance (number of gates in a logical chain) from a primary input to a given gate.

The gate selection algorithm is provided as Algorithm 3. First, the ODC values and height of each node in the logical network are calculated. Once the ODC values are calculated, the nodes are sorted into topological order and the various logical cuts are applied. A node is added to the cut and the children are then trimmed from the

Algorithm 3: ODC cut based selection heuristic [90]

```

forall  $n_i \in nodes$  do
    Calculate ODC set of  $n_i$ ;
    Calculate height of  $n_i$ ;
end
forall  $n_i \in |nodes|$  in topological order do
    Add  $n_i$  to cut;
    Trim children of  $n_i$ ;
    Cutscore[ $n_i$ ] =
         $\alpha \times \left| \frac{max\ Height - 2 \times cutHeight}{max\ Height} \right| + (1 - \alpha) \times \left( \frac{cutODCSum}{max\ ODCSum} \right)$ ;
end
    Select best cut C;
forall  $n_i \in C$  do
    Tag  $n_i$ ;
end

```

given node. The cutscore is then computed, which includes a weighting term α to favor either the ODC or cut height. A component of the cutscore is calculated using the $2 \times cutHeight$ criteria, as nodes in the middle of the logical network are favored for better controllability and observability [90]. When selecting a gate, moving farther from a primary input reduces the controllability of a node, while moving away from a primary output reduces the observability.

After calculating the cutscore for each node, the cut with the best score is selected. The nodes are then marked for use during the technology mapping portion of the modified synthesis flow [90], completed with MVSIS, an open source tool from UC Berkeley.

Similar to [89], the heuristic based selection methodology achieves a 50% Hamming

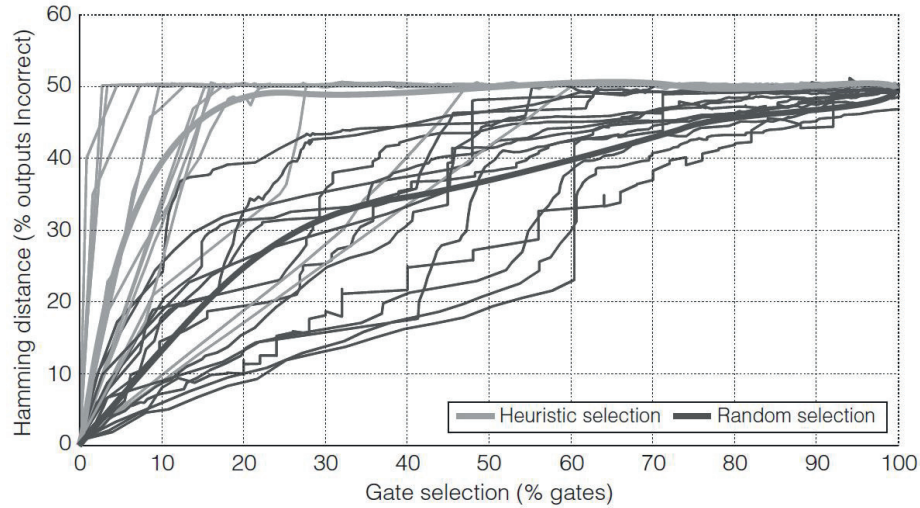


Figure 2.7: Analysis of Hamming distance as a function of the percentage of gates encrypted in a circuit [90].

distance at the outputs much quicker than the heuristic that applies random gate selection, as shown in Fig. 2.7. On average, about 8% of the gates in the circuit were encrypted to achieve a single cut, which effectively produces a Hamming distance close to 50%. The k2 benchmark circuit (1201 gates) required the largest overhead, as measured by the number of selected gates, with approximately 20% of gates being locked [90].

The security against an attack where the oracle is altered one key bit at a time is also analyzed. The attack is performed by applying input patterns to the circuit and observing the corresponding output patterns. The number of correct outputs is then used to configure the inputs to a LUT. The process is repeated for each LUT within the circuit. The analysis provides an evaluation of the value(s) of α that result

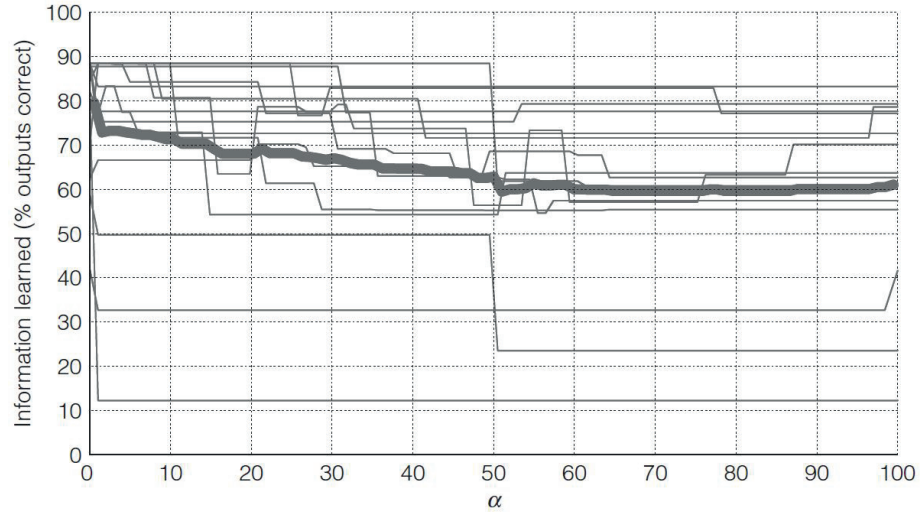


Figure 2.8: Impact of α on the amount of information learned [90].

in the most secure circuit. The results of the analysis are shown in Fig. 2.8, and indicate that the minimum average amount of information learned (thick line) occurs at an α of approximately 0.5. The described attack applies principles of both a key sensitization attack and hill climbing attacks, both of which are discussed in detail in Chapter 3.

2.4.2.c Strong Logic Locking Gate Selection

In contrast to the fault-analysis and observability don't care gate selection techniques that target a 50% Hamming distance, the strong logic locking technique proposed in [91] aims to prevent the sensitization of the keys to the outputs of an IC. The key sensitization attack introduced in [91] demonstrates that key extraction is possible even when gate selection for logic locking is judiciously applied. Greater

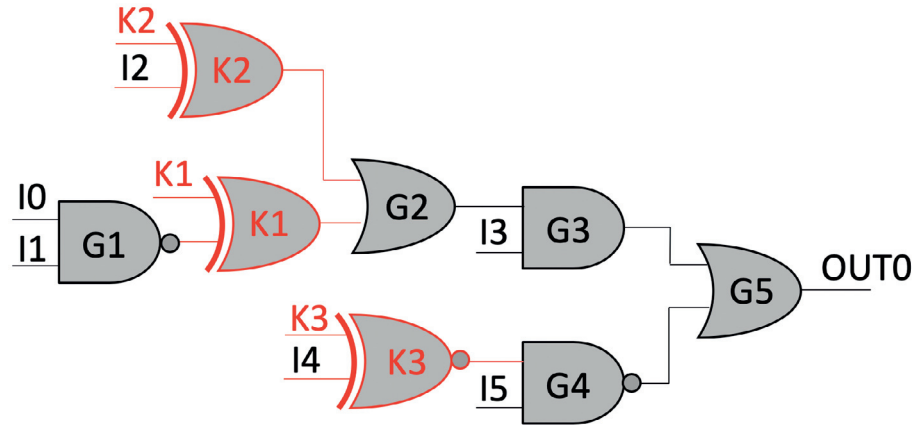


Figure 2.9: Example of mutable and non-mutable key gates. Key gates K1 and K2 are pairwise secure. K3 is mutable and can be sensitized to the output.

detail regarding the key sensitization attack is provided in Chapter 3.2.

To prevent a key sensitization attack, the algorithm proposed in [91] generates non-mutable key dependencies between the inserted key gates. A pair of non-mutable key gates implies that one of the two key gates is not logically masked by a given input pattern. An example is shown in Fig. 2.9, where key gates K0 and K1 are pairwise secure, whereas K3 is sensitized to the output. To sensitize K1 to the output, the other input to G2 must be known. However, since I2 is secured by K2, an adversary is unable to determine the internal value of the second input to G2, which makes K2 non-mutable with regard to K1.

The logic locking gate K3 is mutable, as setting input I5 to logic 0 removes any dependence on K3 at the outputs of the IC. In addition, by setting I3 to logic 0 and I5 to logic 1, OUT0 is only dependent on K3 and I4, which allows an adversary to

determine the correct value of K3. By creating pairwise security and avoiding mutable key gates, the strong logic locking technique minimizes the amount of information leaked though a key sensitization attack [91].

The algorithm utilized to generate high levels of pairwise security is provided as Algorithm 4. The algorithm begins with an initialization phase by calling the *Initialize()* function. The *Initialize()* function greedily chooses the node with the highest convergence score, which is a measure of the number of converging nodes at a given gate [91].

After initialization, groups of gates are generated that are pairwise secure, which are referred to as cliques [91]. Pairwise security is checked first by Theorem 1, which states:

If node n1 and node n2 are pairwise secure, node n2 and node n3 are pairwise secure, and n2 is a dominating key gate (more detail provided in Chapter 3.2) of n1, then n1 and n3 are pairwise secure as well [91].

The utilization of Theorem 1 eliminates unnecessary computation by an automated test pattern generation (ATPG) tool, which is required if Theorem 1 is not satisfied. If the algorithm is unable to insert a pairwise secure key gate with the currently inserted key gates, the algorithm returns to the initialization phase to start a new clique.

The overheads in delay and power are dependent on whether or not the critical

Algorithm 4: Strong Locking Locking [91]

Input: Original netlist, KeySize
Output: Locked Netlist
 $SET_{KG} = \{\}$; KeyGateCount = 0;
 $CLIQ_{cur} = \{\}$; cliqueCount = 0;
Initialize();
while $KeyGateCount \leq KeySize$ **do**
 Find the set C of new candidates for pairwise secure locations that have not been marked as processed;
 foreach $Gate_i \in C$ **do**
 if $Gate_i$ is dominated by any key-gate in $CLIQ_{cur}$ and $Gate_i$ is pairwise secure **then**
 Insert a key-gate at the output of $Gate_i$;
 Add $(Gate_i, cliqueCount)$ to SET_{KG} ; Add $Gate_i$ to $CLIQ_{cur}$;
 KeyGateCount += 1; Mark $Gate_i$ as processed;
 break;
 else
 foreach $KG_j \in CLIQ_{cur}$ **do**
 if $Gate_i$ and KG_j are pairwise secure using ATPG **then**
 mark $Gate_i$ and KG_j as pairwise secure;
 else
 break;
 end
 if $Gate_i$ is marked pairwise secure with all key-gates in SET_{KG} **then**
 Insert a key-gate at the output of $Gate_i$;
 Add $(Gate_i, cliqueCount)$ to SET_{KG} ; Add $Gate_i$ to $CLIQ_{cur}$;
 KeyGateCount += 1;
 break;
 else
 Mark $Gate_i$ as insecure; Mark $Gate_i$ as processed;
 end
 end
 if all gates $\in C$ are insecure **then**
 $KG_{new} = Initialize()$;
 if $KG_{new} = \{\}$ **then**
 break; // Terminate the algorithm
 end

path was secured. For example, when securing the ISCAS'89 s444 benchmark circuit, a 42% overhead in delay is incurred when utilizing strong logic locking whereas only a 3% overhead in delay is incurred when utilizing fault analysis based insertion [91]. However, for the c432 ISCAS'85 benchmark circuit, a 30% overhead in delay is incurred when implementing strong logic locking while the fault analysis based gate selection algorithm results in a 45% increase in area. The resulting overhead in area due to the strong logic locking methodology is similar to that incurred when implementing the fault analysis MUX-based technique, where the average area overhead for strong logic locking is 2.94% and the average area overhead of the fault analysis MUX-based gate selection methodology is 2%, when 1% of the gates of a circuit are locked [91].

2.5 Comparison of Logic Obfuscation Methodologies

Split manufacturing, IC camouflaging, and logic locking represent three different methodologies to obfuscate an IC. The overall objective of the three methodologies is to remove critical design information available to an adversary. However, the differences in the implementation of each technique results in dissimilar threat models and overheads in power, performance, and area.

Splitting the IC design and manufacturing flow into stages that include design, fabrication, test and validation, and the end user, all three obfuscation techniques are vulnerable to threats within at least one stage of the flow as listed in Table 2.1.

Table 2.1: Overview of protection split manufacturing, IC camouflaging, and logic locking provide for the design, fabrication, test and validation, and end user stages of the IC design flow. The protection against each threat vector is rated from None, meaning no protection is provided, to High, meaning significant protection is achievable against that threat vector.

Threat Vector	Obfuscation Type	Design Stage			
		Design	Fabrication	Test & Validation	End User
Malicious Insertion (Hardware Trojan)	SM	None	Low	Low	Low
	IC CAMO	None	Low	Low	Low
	Logic Locking	None	Low	Low	Low
IC Counterfeiting	SM	N/A	High	None	None
	IC CAMO	N/A	None	None	None
	Logic Locking	N/A	High	High	High
IP Theft	SM	N/A	High	None	None
	IC CAMO	N/A	None	High	High
	Logic Locking	N/A	High	High	High
Overproduction	SM	N/A	High	N/A	N/A
	IC CAMO	N/A	None	N/A	N/A
	Logic Locking	N/A	High	N/A	N/A

Note that none of the three obfuscation methodologies protect against the insertion of hardware Trojans in the IC design phase. The vulnerability is due to the opaqueness of using untrusted third parties in the design of a circuit including for IP integration, EDA tools, and PDKs. Since the resulting IC fundamentally relies on the untrusted features, adding any obfuscation measures does not provide additional security against malicious modifications or use of untrusted third parties.

After the design stage of the IC manufacturing flow, the logic obfuscation methodologies offer some protection against the insertion of malicious circuitry as less is

known regarding the correct operation of the circuit once obfuscation is added to an IC. However, the level of protection against hardware Trojan insertion is described as low in Table 2.1 since an adversary is still able to add malicious components to an IC, even if modifications to the specific functionality of the circuit are more difficult to implement.

In the fabrication stage of the IC flow, IC camouflaging does not offer protection from an untrusted fabrication facility as listed in Table 2.1. The requirement of photolithographic masks for the production of the IC limits the protection against an untrusted fabrication facility. Since the fabrication facility has the masks, the true functionality for each of the obfuscated cells is known. Split manufacturing and logic locking limit the amount of circuit information provided to a fabrication facility by, respectively, 1) splitting the fabrication of a circuit between an untrusted and trusted facility and 2) requiring a key to activate the correct functionality of the IC.

After fabrication, when the IC is undergoing testing or is in the field (end user), split manufacturing is vulnerable to counterfeiting and IP theft as all design information is available to an adversary that reverse engineers the circuit. IC camouflaging prevents IP theft since the camouflaged cells are unknown, but is vulnerable to counterfeiting as the IC can be used as a black box, which allows an adversary to fabricate a correctly operating circuit without knowledge of the internal functionality. Logic locking protects against both counterfeiting and IP theft at both the test and end

user stages by requiring a key to set the correct functionality of the circuit. Since neither the test facility nor end user has the correct key without acquiring access from the IC designer, logic locking protects against both threat vectors. Note that the protection against counterfeiting is dependent on the life-cycle of the key. If the key is set when activated and is not removed when the IC is stripped from a printed circuit board (PCB), then counterfeiting is still possible.

While logic locking offers protection against the widest variety of threat vectors, the implementation of the technique results in the greatest overhead in power, performance, and area (PPA) of the three obfuscation methodologies. The additional overhead is due to the required modifications/additions to the standard cells and the logic to store and apply the key. The keys can be applied off chip, but additional I/O overhead results. IC camouflaging typically requires less overhead than logic locking as there is no key storage circuitry, while the modification to the standard cells results in similar overhead in PPA to logic locking. Split manufacturing provides the lowest overhead since standard cells remain unchanged and no key logic is needed. However, the resulting overhead from split manufacturing is due to the increased length of interconnects required to implement the technique. The longer interconnects are a result of avoiding dangling interconnects and proximity leakage as described in Section 2.1. The choice of obfuscation methodology is then dependent on the threat vectors addressed and the tolerable overhead in PPA.

Chapter 3: Logic Locking Attack Vectors

The introduction to logic locking provided in Chapter 2 demonstrates the implementation of key dependencies added to a circuit. Introducing a key dependency changes the logical functionality of a circuit from that represented by the truth table shown in Fig. 3.1a to the functionality given by Fig. 3.1b, where the set of possible keys each represents a different possible function at the output of the IC. When an adversary is unaware of the correct output of the IC for a given input, determining the key becomes very difficult. The determination of the key is possible through 1) bias in the algorithm implementing a given security technique, such as a skew in the amount of key values that produce a given output, or 2) structural/topological leakage of information regarding the key, such as only using XOR gates for a logic 0 key.

While the key is difficult to determine when no input-output pairs are known, the assumption that an adversary has access to an activated IC with the correct key applied is reasonable. For example, an adversary is able to legitimately purchase an activated IC to gain access to the correct input-output pairs. Access to an activated IC with the correct key applied is referred to as oracle access, which provides black box responses to input queries. For example, if an adversary applied input 010 to the logical function shown in Fig. 3.1b, then an oracle response of 0 eliminates three

A	B	C	OUT
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

(a)

A	B	C	K0	K1	K2	K3	K4	K5	K6	K7
0	0	0	0	1	0	1	0	1	0	0
0	0	1	0	0	0	1	1	1	0	0
0	1	0	1	1	0	0	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	1	1	0	0	1	1	1	0
1	0	1	1	1	1	1	0	0	0	0
1	1	0	0	0	0	0	1	1	1	1
1	1	1	0	1	0	1	1	0	0	0

(b)

Figure 3.1: Example of transformation of (a) original logic function to a (b) logic locked functional representation.

possible keys. The focus of this chapter is to characterize three different oracle guided attacks, specifically, 1) hill climbing, 2) key sensitization, and 3) SAT solver based attacks, and to analyze the benefits and drawbacks of each attack from the perspective of an attacker.

3.1 Hill Climbing Attack

Hill climbing is an optimization algorithm that iteratively improves a solution by 1) generating possible solutions, 2) evaluating those solutions, and 3) applying the solution with the best score. The steps of the algorithm are then repeated until the desired solution is found or a pre-determined number of iterations is completed.

The hill climbing attack proposed in [92] utilizes random test vectors to iteratively improve the effective error rate of the circuit. The pseudocode for the hill climbing attack is provided as Algorithm 5. The initial step sets each key bit to a random value. The response of the circuit to a random bit is then evaluated with *test()*, and the bit value (0 or 1) that produces the fewest differences when compared with the oracle is kept. If there are no differences, then the circuit is evaluated with the *correct()* function, which assesses the given key against a set number of oracle responses. If the applied key and oracle match for the set number of responses, then the key is deemed correct and returned. If the correct function returns that the key is incorrect, but no differences were determined for the provided key bit flip, then the algorithm resets all key bits [92].

In the context of a hill climbing attack on logic locking, the correctness of an applied key is evaluated based on the number of errors that are produced when comparing against the output from an oracle IC. However, the security of logic locking is dependent on the number of input combinations to the circuit, where the total number

Algorithm 5: Hill Climbing Attack [92]

Input: Locked Circuit: *ckt*, Patterns: *patterns*,
 Expected Test Response: *resp*
Output: $CK \in \{0, 1\}^n$
found = false;
while *found* **do**
 $CK \leftarrow \text{list random}\{0, 1\}$;
 foreach *random ck in CK* **do**
 $\text{resp1} = \text{test}(\text{ckt}(CK), \text{patterns})$;
 $\text{diffs1} = \text{diff}(\text{resp1}, \text{resp})$;
 // flip one random bit
 $\text{ck} = !\text{ck}$;
 $\text{resp2} = \text{test}(\text{ckt}(CK), \text{patterns})$;
 $\text{diffs} = \text{diff}(\text{resp2}, \text{resp})$;
 if $\text{diffs1} < \text{diffs}$ **then**
 // original value of random bit
 $\text{ck} = !\text{ck}$;
 $\text{diffs} = \text{diffs1}$;
 if $!\text{diffs}$ **then**
 if $\text{correct}(\text{ckt})$ **then**
 $\text{found} = \text{true}$;
 break
 end
end

is large enough that an adversary is unable to apply all possible input combinations in a reasonable amount of time. For example, if there are 64 inputs to a circuit, then exhaustively searching the number of possible inputs results in 2^{64} combinations. If the input patterns are applied at a rate of 10 GHz, then approximately 58 and a half years are required to verify all input combinations on a single CPU.

Since the number of possible input-output combinations is so large, the hill climbing attack is only able to explore a small portion of the possible input-output combinations for a given key. While the search potentially returns a key with zero errors on the evaluated input-output combinations, there is no guarantee that the correct key has been returned. As the number of inputs increases, the attack searches a smaller percentage of all possible input combinations to the IC, which results in a greater likelihood that a local minima is obtained when applying an incorrect key as the analyzed subset of input-output pairs do not show any error. The hill climbing attack is able to return an approximate key that evaluates well on a sub-space of the input-output responses of the IC, but there is no formal bound on the number of test patterns that are required to guarantee a correct key for a given circuit.

3.2 Sensitization Attack

The sensitization attack described in [91] generates testing patterns that propagate the value of specific key bits to the outputs of the IC. The targeting of a single key bit at a time is in contrast to the hill climbing attack described in Section 3.1, where an entire key vector is applied and evaluated each iteration of the algorithm. The advantage of the key sensitization attack is that each iteration results in the correct determination of a single key bit, which provides partial information regarding the circuit even if all key bits cannot be resolved.

The key sensitization attack targets four types of logical interference between key

gates, which includes 1) runs of key gates, 2) isolated key gates, 3) dominating key gates, and 4) convergent key gates. Interference based on runs of key gates, which are multiple key gates in a row, results in a simplification to the circuit and a reduction in the number of effective keys. The simplification results from the functional equivalence of a run of key gates to a buffer or inverter [91]. Therefore, any number of consecutive key gates collapses to two possible functions, which permits the replacement of a run of key gates by a single XOR key gate.

An isolated key gate describes the logical structure where the output of a key gate is propagated to an output of the IC. An isolated key gate, therefore, results when no other key gates with unknown key values exist in the logical path from the output of the given key gate to a primary output of the circuit. An example of an isolated key gate is shown in Fig. 2.9 that includes a key bit K3, since setting input I3 to logic 0 and input I5 to logic 1 allows an adversary to propagate the output of K3 to OUT0. The output of the oracle IC for the given input test pattern then reveals the correct value of K3.

A dominating key gate describes the logical structure where another key gate exists in the transitive fan-in cone of the key gate under evaluation. In order to determine the bit value of the target key gate, the other key gate within the transitive fan-in cone of the target gate must be muted, which implies that the applied input pattern is independent of the other key gate. Consider the example shown in Fig. 3.2, where K2

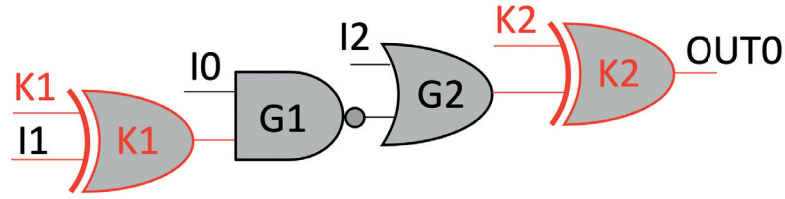


Figure 3.2: Example of a dominating key gate K2. The key gate K1 must be muted since the gate is part of the transitive fan-in cone of K2. Applying a logic 0 to I0 or a logic 1 to I2 mutes the effect of K1, which allows for the determination of K2 at OUT0.

is a dominating key gate with respect to K1. The value of K1 is muted by applying a logic 0 to I0 or a logic 1 to I2. After muting the value of K1, then the value of OUT0 is only dependent on K2.

Lastly, convergent key gates merge at a downstream node within the circuit. Just like dominating key gates, one of the key gates must be muted in order for the other key gate to be sensitized to an output of the IC. If both key gates are independently mutable, which allows for the determination of the value of the other key gate, then the gates are termed concurrently mutable. If the key gates must be muted in a certain order, then the two key gates are termed sequentially mutable [91]. An example of a sequentially mutable circuit is shown in Fig. 3.3, where both K1 and K2 converge at G4. K2 cannot be muted in this example. K1 must, therefore, be muted first to determine the value of both key gates. Once K1 is muted by setting I5 to logic 1, then the value of K2 is determined. K1 is then ascertained after setting K2 to the determined value.

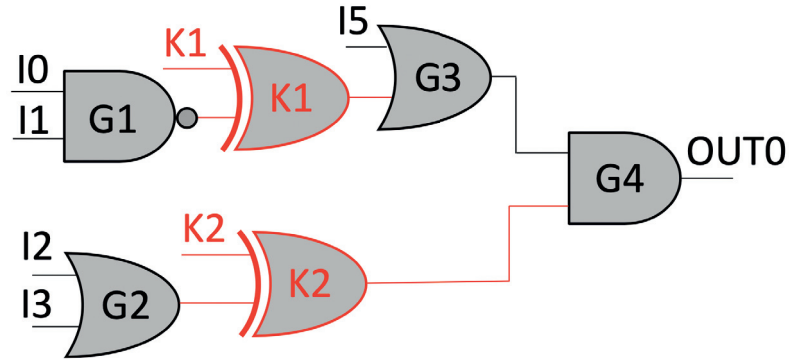


Figure 3.3: Example of a sequentially mutable circuit. The key gate K1 must be muted first since K2 cannot be muted. Applying a logic 1 to I5 mutes K1 and allows for the determination of K2 at OUT0 [91].

The pseudocode of the key sensitization attack that applies the four interference rules is provided as Algorithm 6. Prior to the execution of the primary loop of the algorithm, the check for *runs of key gates* is completed. If any runs of key gates are found, the run is replaced by a single functionally equivalent key gate and the netlist is updated.

For isolating, dominating, and convergent key gates, the attack relies on the generation of input patterns that are able to determine a given key bit. The algorithm utilizes automatic test pattern generation (ATPG) to produce the patterns, where the unknown key values are treated as don't cares (logic X's). More information regarding ATPG tools and algorithms is provided in [93]. The attack first prioritizes *isolated key gates* within the loop, since determining the sensitization pattern of isolated key gates is the most straightforward as no other key gates must be muted. If any key

Algorithm 6: Key Sensitization Attack [91]

Input: Locked netlist, Functional IC, Key-inputs
Output: Original Netlist
 Determine *Runs of Keys*;
 Replace them with XOR gates;
 Update Netlist;
for *the remaining keys* **do**
 foreach *Isolated Key* **do**
 Compute and apply sensitization pattern;
 Determine *Key-bits* and update Netlist;
 end
 foreach *concurrent || sequential key* **do**
 if *there exists a golden pattern* **then**
 Apply the golden pattern;
 Determine *Key-bits*, Update Netlist, Break;
 ApplyBruteForce(), Break;
 end
 foreach *Non-mutable Key* **do**
 ApplyBruteForce(), Break;
 end
 end
end

 ApplyBruteForce();
foreach *possible key combination* **do**
 Generate random input patterns;
 Simulate the patterns and obtain the outputs OP_{sim} ;
 Apply the patterns on IC and obtain the outputs OP_{exe} ;
 if $OP_{sim} == OP_{exe}$ **then**
 Valid Key = current key combination;
 Update netlist;
end

gates are found to satisfy the criteria of an isolated key gate, the netlist is updated by replacing the given key gate with a buffer or inverter depending on the correct value of the key. Since the netlist is updated after each iteration a key is determined, the type of interference of the key gates potentially changes, which implies that the

classification of the interference for each gate type must be redetermined before the execution of the next iteration [91]. Concurrent and sequential gates are then evaluated, where the netlist is examined for an input pattern that exposes the value of the key bit. An ATPG test pattern that determines the key bits for concurrent or sequential gates is termed a golden pattern. If a golden pattern is not found, then a brute force attack is executed to determine the given key bit. The remaining key bits that are not determined fall under the category of non-mutable key gates, which also require the execution of a brute force attack to ascertain the correct value.

As previously mentioned, the key sensitization attack provides a guarantee that the determined key bit values are correct, which is a benefit over the hill climbing attack discussed in Section 3.1. However, there are many cases where non-mutable key gates exist within a design. Under such scenarios, applying a brute force attack to determine the key values quickly becomes inefficient as the number of inputs is increased. The key sensitization attack is then of most use when only a few non-mutable gates exist and the fan-in cone for the non-mutable gates has a limited number of inputs. However, by judiciously selecting the key gates using the strong logic locking gate selection algorithm described in Section 2.4.2.c, quite a few non-mutable key gates exist, which results in an inefficient key sensitization attack.

3.3 SAT-Attack

The example of the logic locked functional space shown in Fig. 3.1b indicates that applying a single input pattern to an oracle IC prunes more than a single key. The hill climbing attack evaluates a subset of the key space at a given time, which implies that only certain columns are evaluated for the applied test inputs to estimate an error rate. Therefore, the evaluated inputs are only used to constrain the current keys under evaluation, which implies that the inputs are not optimally utilized to constrain the key space.

The key sensitization attack discussed in Section 3.2 utilizes test patterns generated with an ATPG tool to find input patterns that determine the value of key bits. Each one of the applied input values effectively constrains the key space by a factor of 2 since the value of the key bit is constrained to either logic 0 or logic 1. When test patterns are found that sensitize a single key bit to an output, such an attack is efficient since the number of input patterns required to determine the correct key grows linearly with key size. However, the requirement of propagating the key bit to a primary output becomes challenging, as discussed in Section 3.2, since key gate insertion algorithms are able to create a large percentage of non-mutable key gates that prevent the execution of an efficient attack.

Satisfiability (SAT) based attacks differ as the attack simply selects an input pattern and uses the oracle response as a constraint for the next iteration. Any key

value that does not satisfy the constraint is not a part of the solution space for future iterations, which allows for an efficient use of each input pattern. Unlike the key sensitization attack, SAT based attacks look for the set of generated constraints to be satisfied as opposed to the logical propagation of a single key gate to an output of the circuit. Therefore, groups of key sequences are able to influence the output, and while a single key bit is not determined in a given iteration, the attack effectively constrains the functional space.

The backbone of SAT based attacks are SAT solvers, as the name of the attack suggests. Boolean SAT solvers function by attempting to prove the existence of a solution to a given problem, rather than searching for an optimal solution. SAT solvers rely on a set of clauses, which collectively form the constraints applied to the solver. For example, a set of clauses in conjunctive normal form (CNF) for a SAT solver to solve are provided in (3.1). For the given example, one possible solution that satisfies the clauses is $A = 0$, $B = 1$, and $C = 1$. However, switching B to logic 0 also satisfies the formulated clauses. More information regarding the efficiency of SAT solvers resolving such constraints is available in [94] and [95].

$$(A \vee \bar{B} \vee C) \wedge (B \vee C) \wedge (\bar{A}) \quad (3.1)$$

The SAT attack presented in [96] relies on a miter circuit as the original problem formulation provided to the SAT solver. A miter circuit logically represents two copies

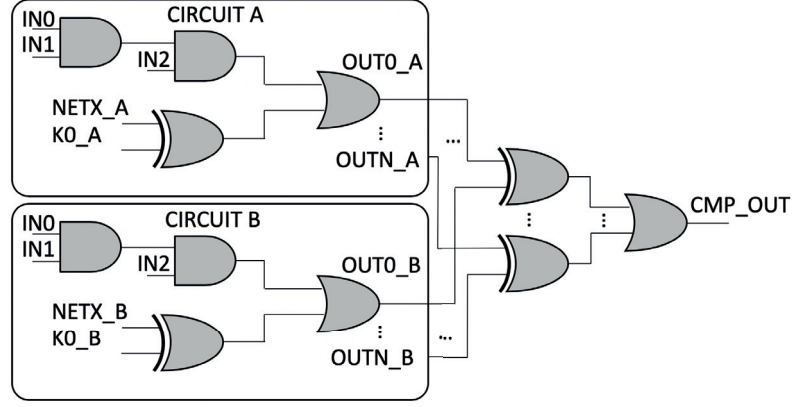


Figure 3.4: Miter circuit with replicated A and B versions of the reverse engineered netlist that includes obfuscated gates. Corresponding outputs between the A and B versions of the circuit are XORed. Each XORed signal is then passed to an OR gate to check for any differentiating output.

of a locked reverse engineered circuit with each output of the two copies XORed for equivalence checking. The outputs from the XORs are then connected to an OR gate, as shown in Fig. 3.4. The logical representation of the miter circuit is efficiently converted into conjunctive normal form (CNF) via a Tseytin transformation [97], where the Tseytin transformation of an AND gate with inputs A and B and output C is provided as (3.2). The CNF representation of the circuit is then provided to a SAT solver.

$$(\bar{A} \vee \bar{B} \vee C) \wedge (A \vee \bar{C}) \wedge (B \vee \bar{C}) \quad (3.2)$$

For the miter circuit to produce an output of logic one, there must be a difference in the logical output between the two copies of the circuit. Since only the key values

Algorithm 7: SAT Attack Algorithm [96]

Input: C and $eval$
Output: $\vec{K}_C$
 $i := 1;$
 $F_1 = C(\vec{X}, \vec{K}_1, \vec{Y}_1) \wedge C(\vec{X}, \vec{K}_2, \vec{Y}_2);$
while $sat[F_i \wedge (\vec{Y}_1 \neq \vec{Y}_2)]$ **do**
 $\vec{X}_i^d := sat_assignment_{\vec{X}}[F_i \wedge (\vec{Y}_1 \neq \vec{Y}_2)];$
 $\vec{Y}_i^d := eval(\vec{X}_i^d);$
 $F_{i+1} := F_i \wedge C(\vec{X}_i^d, \vec{K}_1, \vec{Y}_i^d) \wedge C(\vec{X}_i^d, \vec{K}_2, \vec{Y}_i^d);$
 $i := i + 1;$
end
 $\vec{K}_C := sat_assignment_{\vec{K}_1}(F_i)$

differ between the two copies of the locked circuit, the applied pattern represents an input that is guaranteed to reduce the key space by at least one key sequence. This is true as at least one key value differs from the oracle response. The generated input pattern is referred to as a distinguishing input pattern (DIP) [96].

The pseudocode of the SAT attack is provided as Algorithm 7. The notation utilized to describe the combinational logic space is $C(\vec{X}, \vec{K}, \vec{Y}) \subseteq \mathbb{B}^{M+L+N}$, where $\mathbb{B}$ is the binary domain of $\{0, 1\}$, $\vec{X} \in \mathbb{B}^M$ represents the M primary inputs, $\vec{Y} \in \mathbb{B}^N$ represents the N primary outputs, and $\vec{K} \in \mathbb{B}^L$ represents the L key inputs.

The primary steps of the algorithm include 1) generation of a distinguishing input pattern (DIP), 2) evaluation of the DIP on an oracle IC, and 3) adding the constraints generated from the oracle response to the SAT model of the circuit. As previously described, the generation of a DIP, $\vec{X}_i^d := sat_assignment_{\vec{X}}[F_i \wedge (\vec{Y}_1 \neq \vec{Y}_2)]$, is

completed through the use of a miter circuit of the locked netlist obtained by reverse engineering the IC. The generated DIP values are then provided to an activated IC as inputs, $\vec{Y}_i^d := eval(\vec{X}_i^d)$, where the unlocked IC is utilized to determine the correct outputs of the circuit for the given input pattern.

The applied inputs and resulting outputs generated from the oracle IC are then propagated through the CNF equivalent representation of the locked IC netlist, leaving a set of logical constraints that are appended to the original SAT formulation, $F_{i+1} := F_i \wedge C(\vec{X}_i^d, \vec{K}_1, \vec{Y}_i^d) \wedge C(\vec{X}_i^d, \vec{K}_2, \vec{Y}_i^d)$. The process of 1) DIP generation, 2) evaluation on an oracle IC, and 3) adding input/output constraints to the SAT model is repeated until no further generation of DIPs is possible.

An example of the algorithm executing on the circuit shown in Fig. 3.5 is provided. Evaluating the output y for the input vector $\vec{X}_1^d = (1,0,1)$ results in a logic one. Adding the new input/output condition to the SAT solver eliminates $(k_1, k_2) = (0,1)$ as a valid key combination since $y = 0$ when evaluating $C(\vec{X}_1^d, \vec{K}_1, \vec{Y}_1^d)$. The next iteration of the algorithm selects $\vec{X}_2^d = (0,0,0)$, for example, that evaluates to $y = 0$, which forces k_2 to 1 and leaves $(k_1, k_2) = (1,1)$ as the only valid key combination. One important result of the algorithm is that the actual key does not need to be observed at a primary output. Rather, differences in correct and incorrect outputs, independent of the value, result in the discarding of incorrect keys. The ability to trim the keyspace with constraints generated through groups of key gates increases

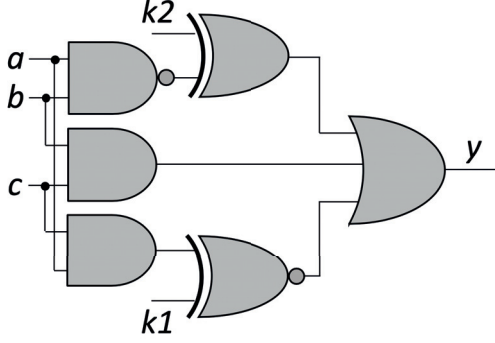


Figure 3.5: Encrypted circuit with $(k_1, k_2) = (1, 1)$ [96].

the efficiency of SAT based attacks over key sensitization attacks, as no brute force attacks are required.

The efficiency of the SAT attack is evident through the results shown in Fig. 3.6, where six in-cone logic locking methodologies are evaluated. The evaluated logic methodologies include random insertion (RND) [88], fault-analysis XOR (ToC'13/xor) and MUX (ToC'13/mux) based insertion [89], observability don't care selection LUT-based insertion (DTC'10/LUT) [90], strong logic locking (DAC'12) [91], and a technique to increase the switching probability of nets to facilitate hardware Trojan detection (IOLTS'14) [54]. Background material providing an overview of the RND, ToC'13/xor, ToC'13/mux, DTC'10/LUT, and DAC'12 gate selection strategies is provided in Section 2.4.2. The allowed overhead in area, as given by the number of additional gates added to the original circuit, was set to 5%, 10%, 25%, and 50%. DTC'10/LUT does not provide a means to limit the overhead in the area of the circuit, so only a single circuit topology was implemented, and the overhead was not

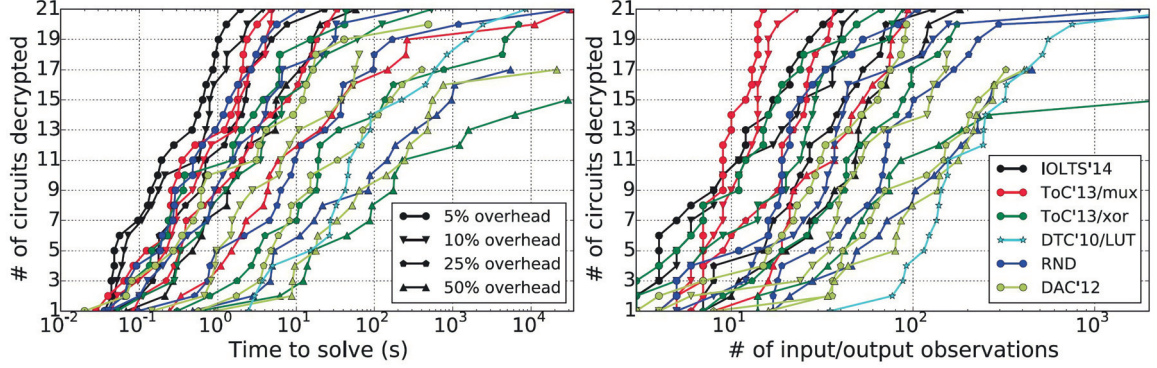


Figure 3.6: Analysis of the execution time (left) and number of input/output combinations needed to decrypt the circuit (right). Each color represents a different locking technique and each marker represents a different allowed percentage overhead [96].

analyzed. The ISCAS'85 and Microelectronics Center of North Carolina (MCNC) benchmark circuit suites were utilized for evaluation.

A SAT attack executed on the ToC'13/XOR and DAC'12 logic locking techniques resulted in the longest execution time to determine the key when the overhead in area was set to 50%. However, the increase in time to decrypt the circuit is not very significant over random placement (RND). The LUT based technique (DTC'10/LUT) requires more time to decrypt than other locking methodologies. However, the comparison with other techniques is not equivalent as the overhead in area and the number of applied key bits is not known for the LUT based technique. When analyzing the number of input-output pairs (DIPs) required to determine the logic locking key, the DTC'10/LUT based technique provides the strongest results as measured by the minimum number of DIPs required to determine the key. As the state space of the LUT is

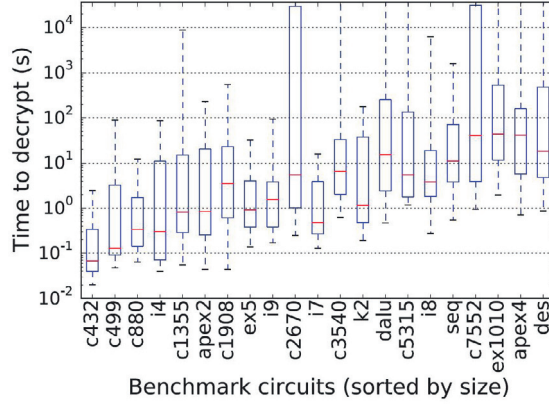


Figure 3.7: Box plot of the runtime of Algorithm 7 [96].

larger than the XOR gate, the number of DIPs is expected to increase. The primary concern from the results shown in Fig. 3.6 is that 95% of the evaluated benchmark circuits were decrypted within 10 hours.

The box plot shown in Fig. 3.7 is provided as a means to analyze and compare the security of various benchmark circuits [96]. The c2670 benchmark circuit provides interesting results, as c2670 is relatively small, 1193 gates, but offers one of the higher levels of security as measured by *time to decrypt*. The AND-tree structure within the circuit provides additional security against the SAT attack [96], which is discussed in further detail in Chapter 7.

3.3.1 Partial Break Algorithm

In addition to the algorithm that attempts to fully decrypt a circuit, an algorithm that partially-breaks the key is also introduced in [96]. Instead of attempting to

Algorithm 8: Algorithm to partially decrypt a key value with SAT analysis using an ILP formulation [96].

Input: C and $eval$

Output: $\vec{K}_C$

$C_{slice} := findSliceUsingILP(C);$

$\{(\vec{X}_1^d, \vec{Y}_1^d), \dots, (\vec{X}_\lambda^d, \vec{Y}_\lambda^d)\} := decrypt(C_{slice}, eval);$

$G := \forall_{i=1}^\lambda C(\vec{X}_i^d, \vec{K}, \vec{Y}_i^d);$

$\vec{K}_C := findBackbones(G)$

determine the key across the entire circuit, a sub-set of the circuit, called a slice, is utilized. By applying the algorithm on a slice, the primary memory needed to decrypt the IC is reduced, which is a bottleneck for large circuits [96]. A slice of the circuit must 1) contain the transitive fanin cone of each included output, 2) have a transitive fanout cone of at least one key value, and 3) be limited to a fraction P of the entire circuit, where a P of 0.3 was applied in [96]. A transitive fanin cone includes all previous gates that connect to the inputs of the given node in a circuit, and a transitive fanout cone includes all the nodes that are connected through the outputs of a given node in a circuit.

An integer linear program (ILP) is formulated to maximize the number of key inputs contained within the slice [96]. The determination of the slice by applying integer linear programming (ILP) is the first step of pseudocode described by Algorithm 8. The resulting slice is then provided to the decrypt algorithm, previously formulated in Algorithm 7, which provides a set of distinguishing input-output combinations.

The input-output constraints, given by G , define required conditions for the correct key. However, simply abiding by the constraints formed in G does not guarantee the returned key decrypts the entire circuit. As a result, a set of backbones, key values $k_i \in \vec{K}$ that must be true in G ($G \implies k_i$), are determined to obtain individual key bits. Such a case is observed for the example circuit shown in Fig. 3.5 when applying the decrypt algorithm. For the given circuit, when $(a,b,c) = (0,0,0)$, the output y evaluates to 0, which leaves $k_2 = 1$ as the only satisfying input. The assertion must then hold for the entire circuit as well, implying that a portion of the correct logic locking key was determined.

Chapter 4: Out-of-Cone Logic Locking

The attack vectors on logic locked circuits described in Chapter 3 demonstrate that an oracle guided attack is able to prune the key space. The efficiency of the pruning of the key space is directly related to the number of keys eliminated from the search space for each executed iteration of the attack. When utilizing the in-cone logic locking methodologies discussed in Chapter 2, oracle guided attacks tend to be efficient as sufficiently limiting the number of keys eliminated for each DIP is not easily achieved.

Instead of inserting the key gates within the logic cone, as in-cone logic locking methodologies do, this chapter examines alternative techniques that control the pattern of corruption induced by the insertion of logic locking. The typical circuit resembles a structure similar to that shown in Fig. 4.1, where the flip function ensures that corruption at the *OUT* net is strategically generated to prevent a single DIP from significantly pruning the key space. Since the flip function exists outside of the original logic cone, such methods are referred to as out-of-cone logic locking methodologies [98–102]. By limiting the number of keys that produce an incorrect response for each input to the circuit, attacks such as the SAT attack [96] are no longer efficient. Each iteration of the SAT attack produces a DIP that guarantees that at least one key is eliminated from the key space. However, the attack is unaware

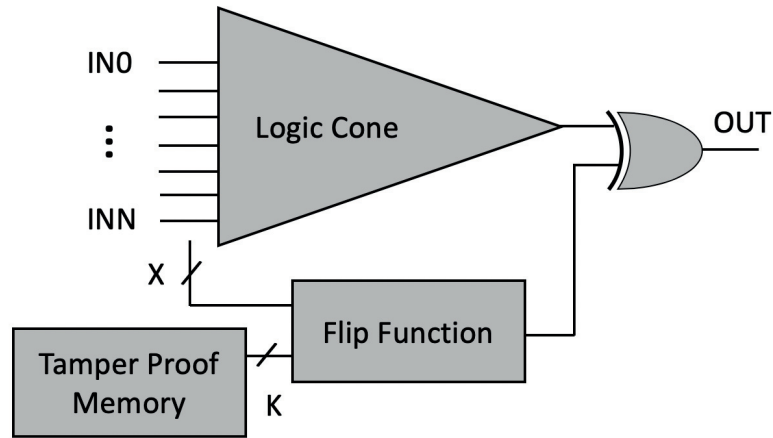


Figure 4.1: Generalized circuit topology to defend against the SAT attack utilized in [98–101]. Signals represented by X , which are extracted from the inputs to the logical cone or the internal signals of the logical cone, are combined with key signals K to generate a function that controls the assertion of the flip function.

of whether the generated DIP eliminates one or all but one of the keys, which implies a reduced SAT attack efficiency if a given locking technique results in the elimination of as close to one key per iteration as possible. The control provided by out-of-cone architectures regarding the number of keys eliminated for each DIP is discussed in this chapter. While the controlled corruption of out-of-cone logic locking methodologies provides resilience to SAT attacks, the techniques also result in novel vulnerabilities, which are also discussed in this chapter.

4.1 Methodologies that Implement Unmodified Logic Cones

The general operation of the flip-function shown in Fig. 4.1 is that for each input pattern, the output is corrupted for specific key combinations. Two flip function

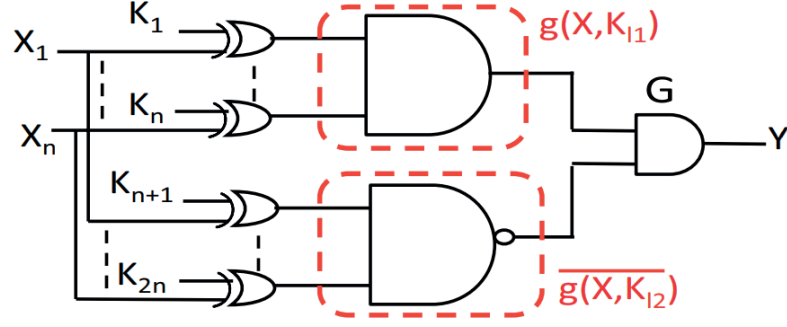


Figure 4.2: The flip function of the Anti-SAT [99] technique where the two functional complements g and $\bar{g}$ are ANDed together. For the given circuit, the AND gate implements g and the NAND gate $\bar{g}$.

techniques are evaluated: 1) Anti-SAT [99] and 2) SAR-Lock [98]. Both methodologies include a flip-function that generates controlled corruption for each input-output pair, but neither technique modifies the original logic cone. Vulnerabilities of each technique are discussed after describing the implementation of each methodology.

4.1.1 Anti-SAT

The Anti-SAT methodology described in [99] utilizes two complemented functions, referred to as g and $\bar{g}$, to generate a flip function. An example of Anti-SAT implemented on a simple circuit is shown in Fig. 4.2, where the g and $\bar{g}$ functions are AND and NAND, respectively. Since g and $\bar{g}$ are complements of each other, no incorrect output responses are observed when the applied key values are the same between g and $\bar{g}$. For example, for the circuit shown in Table 4.1, K0, K5, K10, and K15 all produce the correct circuit response, which corresponds to when the key to g and $\bar{g}$

Table 4.1: The complete key space for the Anti-SAT flip function shown in Fig. 4.2 applied to the original logic cone of an OR gate with inputs A and B.

A	B	K0	K1	K2	K3	K4	K5	K6	K7	K8	K9	K10	K11	K12	K13	K14	K15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
0	1	1	1	1	1	1	1	1	1	0	0	1	0	1	1	1	1
1	0	1	1	1	1	0	1	0	0	1	1	1	1	1	1	1	1
1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1

Table 4.2: The reduced key space for the Anti-SAT flip function shown in Fig. 4.2 applied to the original logic cone of an OR gate with inputs A and B.

A	B	K0	K1	K4	K8	K12
0	0	0	0	0	0	1
0	1	1	1	1	0	1
1	0	1	1	0	1	1
1	1	1	0	1	1	1

match. However, when the keys to g and $\bar{g}$ do not match, one input response is corrupted at the output of the function. To characterize the resilience of the technique to the SAT attack, the key space given by Table 4.1 is reduced to that of Table 4.2 by only retaining the unique columns of the table. Only unique columns are retained as any DIP constraint that removes a single key also eliminates any key that results in duplicate functionality, which implies no security is gained through having multiple keys that result in the same functionality. The reduced key space shown in Table 4.2 demonstrates that each input response only results in a single incorrect output response, which forces an adversary to apply every input response to extract the correct key. The resilience to oracle guided attacks then matches a brute force attack.

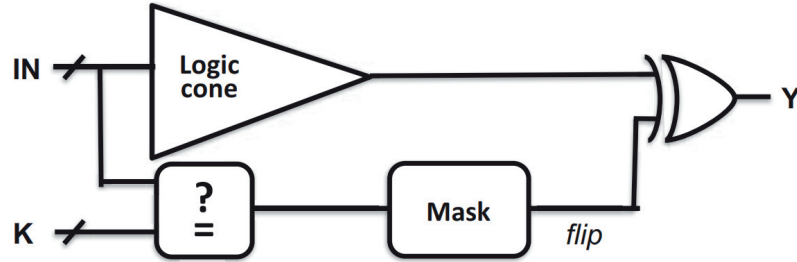


Figure 4.3: Logic diagram of SAR-Lock, which utilizes a comparator to compare an applied key value and the primary inputs of an IC. A mask function then prevents a logic flip when the correct key is applied [98].

4.1.2 SAR-Lock

The SAR-Lock technique [98] utilizes a comparator and a logic masking block to create a point function structure as shown in Fig. 4.3. No error is observed in the IC when the correct key is applied, whereas an incorrect key results in error for a single applied input pattern. In order to ensure no error is observed when applying the correct key, the mask sub-block must guarantee that the output of the flip function is 0 when the correct key is set. While the implementation of the mask function is not described in detail in [98], a second comparator with a set key applied as an input represents a circuit that masks the effect of the original comparator. The only input that then does not result in a flipped output for a given applied key is the masked response. The functional representation of the circuit is then described as $O = F(I) \oplus ((I == K) \oplus (I == K_s))$, where O is the output(s) of the circuit, K is the key inputs, and K_s is the correct key input [103].

The resulting key space from implementing SAR-Lock on a two input OR gate is

Table 4.3: Key space of a two input OR gate logic cone locked with SAR-Lock [98] with the logic mask occurring for input pattern 10.

A	B	K0	K1	K2	K3
0	0	1	0	0	0
0	1	1	0	1	1
1	0	1	1	1	1
1	1	1	1	1	0

provided in Table 4.3. Note that SAR-Lock provides the same property as Anti-SAT, where only a single key is eliminated for each applied DIP. The use of the mask does eliminate the evaluation of a single DIP, which provides a slightly less SAT resilient topology. However, as the number of input responses increases, the reduction in the number of DIPs by only one is insignificant when characterizing the SAT resilience of the circuit. Note that SAR-Lock achieves a similar key space without the addition of the non-unique columns that were observed in Anti-SAT, which allows SAR-Lock to provide SAT attack resilience with fewer keys.

4.1.3 Vulnerabilities of Unmodified Logic Cone Methodologies

While the Anti-SAT [99] and SAR-Lock [98] techniques provide significant resilience against the SAT attack, the introduction of a flip function to the circuit results in additional security concerns that include 1) removal attacks [103] and 2) bypass attacks [104].

Removal attacks [103] utilize the set structure of the flip function added to control the corruption when an incorrect key is applied. One type of removal attack targets the logical properties of the flip functions utilized in Anti-SAT and SAR-Lock. Both Anti-SAT and SAR-Lock, as implemented on the circuits shown in, respectively, Fig. 4.2 and Fig. 4.3, utilize AND and NAND gates within the flip function circuitry. As AND and NAND gates include only a single input condition that leads to, respectively, a logic one or logic zero, the probability of a logic zero or logic one at the output of the flip function is highly skewed. The skewed probability is utilized in the signal probability skew (SPS) attack [103] to first determine the flip function and then to remove the circuit implementing the flip function. Once the flip circuit is removed, the original logic cone is left, which implies that an oracle IC is no longer required to determine the logic locking key.

The structure of the circuit implementing the flip function also reveals information that allows for the removal of the block. For example, the connections to the memory elements storing the logic locking key further limits the key space an adversary must search. The limited logic connected to the memory elements, therefore, allows for the discernment of the circuit implementing the flip function from the original logic cone. In addition, the original logic cone is XORed with the flip function at a primary output of the circuit. If the synthesis tool does not sufficiently integrate the two logic cones, which is highly probable, then an adversary is able to directly isolate the

original logic cone and the flip function logic. The ability to partition the two logic structures allows for the removal of the circuit implementing the flip function.

Bypass attacks [104] to circumvent the errors induced by the flip function when applying an incorrect key have also been proposed. For example, consider the truth table of the key space after implementing SAR-Lock, which is provided in Table 4.3. Consider the case where two random keys are applied to the miter circuit of the SAT attack, for example K0 and K3. Each of the two keys only differs from the correct key for only one input response, which implies that the miter circuit either returns input pattern $AB = 00$ or input pattern $AB = 11$ as the DIP. The single oracle response can then be used to correct either K0 or K3. For example, assume the input pattern 00 is returned. In this case, the oracle response is a 0, meaning that K0 is incorrect. By adding a logic mask to prevent the logical flip of K0 when 00 is applied, K0 now functions as a correct key.

4.2 Modified Logic Cone Methodologies

To address the vulnerabilities of unmodified logic cone techniques to removal and bypass attacks, modified logic cone methodologies modify the original logic cone to provide incorrect functionality when the flip function is removed. Early techniques apply in-cone logic locking to modify and secure the logic cone, while the flip function provides the resilience to the SAT attack [98]. However, multiple attack vectors

including 1) AppSAT [105] and 2) Double DIP [106], were developed to extract the in-cone logic locking key without the need to determine the key used for the out-of-cone logic locking.

AppSAT [105] exploits the fact that out-of-cone techniques provide low output corruption rates in comparison to in-cone logic locking techniques. AppSAT allows the user to set the error threshold that is calculated after every n iterations of the SAT attack, where n is also user defined [105]. If the calculated error rate, which consists of the evaluation of x random patterns, falls below a set error threshold, then the attack stops. The early stop criteria of the attack prevents the generation of the low error rate DIPs induced by the flip function. Since the error threshold, n , and x are all set by the user, there is no guarantee the correct key utilized for the in-cone logic locking is returned. However, execution of the attack typically results in the return of the correct key as shown in [105].

The method developed in [106] described as Double DIP instead forces DIPs to eliminate at least two keys for each iteration of the SAT solver, which bypasses the artificial corruption of the outputs generated by the flip function and targets the key of the in-cone logic locking elements. For example, consider the key space described by the truth table provided in Table 4.3, where the flip function generates only a single error per input pattern. Adding in-cone logic locking increases the number of corrupted outputs for each input pattern. Double DIP then targets the patterns with

increased corruption, which allows for extraction of information regarding the in-cone key while avoiding any inefficiencies due to the flip function when executing the SAT attack.

Methodologies that modify the logic cone directly without the use of key gates are discussed in this section. For example, if the input pattern 00 of the original circuit produces a logic 0 at the output under normal operation, changing the output to a logic 1 requires modifications to the original functionality of the circuit. The flip function then corrects the output of the circuit back to a logic 0 when a correct key is applied. For this scenario, attacks such as AppSAT and Double DIP do not apply as the in-cone logic locking techniques no longer dominate the error rate in the outputs. Three different methodologies are examined in this section that modify the original logic cone, which include 1) stripped functionality logic locking (SFLL)-HD, 2) diversified tree logic (DTL), and 3) SFLL-Flex. The structural vulnerabilities of each approach are also analyzed.

4.2.1 SFLL-HD

The topology of the circuit implementing SFLL-HD, where HD stands for Hamming distance, is shown in Fig. 4.4. The original logic cone is XORed with a perturb unit, which is shown in dashed lines in Fig. 4.4. For SFLL-HD, the perturb unit is a Hamming distance checker that includes a set key. Any input to the circuit that is a

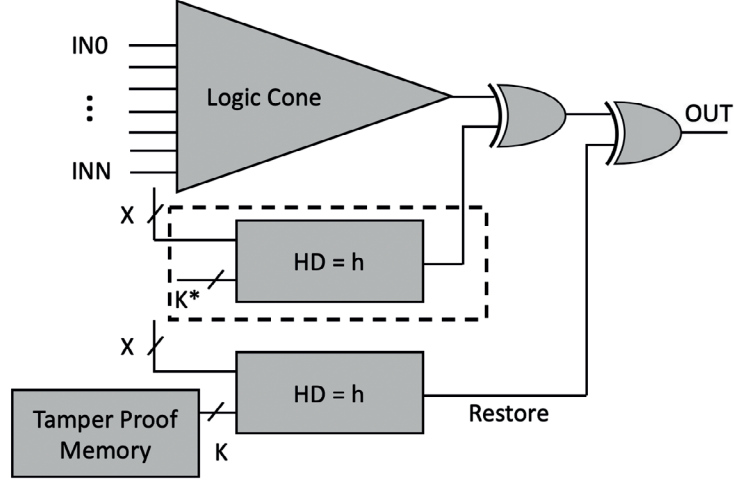


Figure 4.4: Block diagram of the SFLL-HD logic locking technique, where the perturb unit is shown within the dotted box and with the correct key labeled as K^* . The restore unit corrects any modifications introduced by the perturb unit.

specified Hamming distance away from the key value results in an incorrect response at an output of the IC. After inserting the Hamming distance checker, the circuit is re-synthesized to integrate the Hamming distance checker with the original logic cone. The same Hamming distance circuit utilized as the perturb unit is also used as the flip function, which is now referred to as a restore unit [101] since the original functionality is restored with the correct key.

The truth table indicating correct and incorrect output responses for two functional representations of an AND gate locked with SFLL-HD^0 and SFLL-HD^1 are provided as Table 4.4 and Table 4.5, respectively. When locked with SFLL-HD^0 , all of the input patterns produce a single error except for the protected cube of $AB = 01$. Therefore, if an adversary applies $AB = 01$ as a DIP, the correct key is determined.

Table 4.4: Truth table of a two input AND gate locked with SFLL-HD⁰ and a correct key of K1.

A	B	K0	K1	K2	K3
0	0	X	✓	✓	✓
0	1	X	✓	X	X
1	0	✓	✓	X	✓
1	1	✓	✓	✓	X

Table 4.5: Truth table of a two input AND gate locked with SFLL-HD¹ and a correct key of K1 or K2.

A	B	K0	K1	K2	K3
0	0	X	✓	✓	X
0	1	X	✓	✓	X
1	0	X	✓	✓	X
1	1	X	✓	✓	X

The security of SFLL is then dependent on the size of the input space, since a large number of possible input combinations provides a low probability of picking the protected cube as a DIP.

Modifying the locking technique to implement SFLL-HD¹ increases the amount of errors to two for each applied input pattern to the circuit, as shown from the data listed in Table 4.5. Since the size of the key space does not change, the resilience against a SAT attack is reduced. Therefore, there is a trade-off between the resilience

provided against the SAT attack and the resulting input-output corruption generated.

The SAT attack resilience of SFLL-HD is formally defined by the number of protected cubes c , the number of key bits k , and the number of queries q the adversary performs on an oracle IC. The provided security of SFLL-HD is estimated as given by (4.1) [101]. As c increases, the probability of an adversary finding a protected input cube also increases, which results in a decrease in the average number of queries that must be completed to determine the key bits. When the Hamming distance parameter is set to a low value, which results in a small c , the number of calls to the SAT solver, on average, becomes large for a sufficient key size k .

$$\frac{c}{2^k} + \frac{c}{2^k - 1} \dots + \frac{c}{2^k - q} \approx \frac{c * q}{2^k} \quad (4.1)$$

The strongest resiliency against the SAT attack occurs when the Hamming distance parameter is set to 0, as the number of protected cubes is 1. The number of iterations to determine the protected input cube is then estimated as approximately uniform, as calculated by (4.1). In addition, non-protected input cubes only eliminate a single key for each DIP, which results in SFLL-HD⁰ providing a high resiliency to the average SAT attack execution. However, the number of corrupted outputs when comparing an unlocked and locked IC is very low, with only two of 2^n input patterns corrupted when an incorrect key is applied to the locked IC, where n is the number of inputs. Low levels of corruption of the input output pairs cause concern as an

adversary essentially possesses a working IC, especially if the IC is tolerant to rare errors.

Increasing the value of the Hamming distance parameter of SFLL-HD results in greater corruption of the circuit input output pairs as the number of protected cubes expands to k choose h $\binom{k}{h}$, where h is the selected Hamming distance and k is the number of key bits. Instead of a single incorrect output, each non-protected cube now yields an incorrect output for any key that is a Hamming distance h away from the applied input vector. The protected cubes now produce correct responses for $\binom{k}{h}$ combinations of keys, instead of the single correct key provided by SFLL-HD⁰. The total corruption of the outputs of the circuit is now given by (4.2), where $\binom{k}{h}/2^k$ defines the corruption produced by a non-protected cube and $(2^k - \binom{k}{h})/2^k$ denotes the corruption provided by a protected cube. While the corruption increases for larger HD values, DIPs generated by the SAT attack for non-protected cubes now eliminate a greater number of keys, which results in greater efficiency when executing a SAT attack.

$$\frac{\binom{k}{h}}{2^k} + \frac{(2^k - \binom{k}{h})}{2^k} \quad (4.2)$$

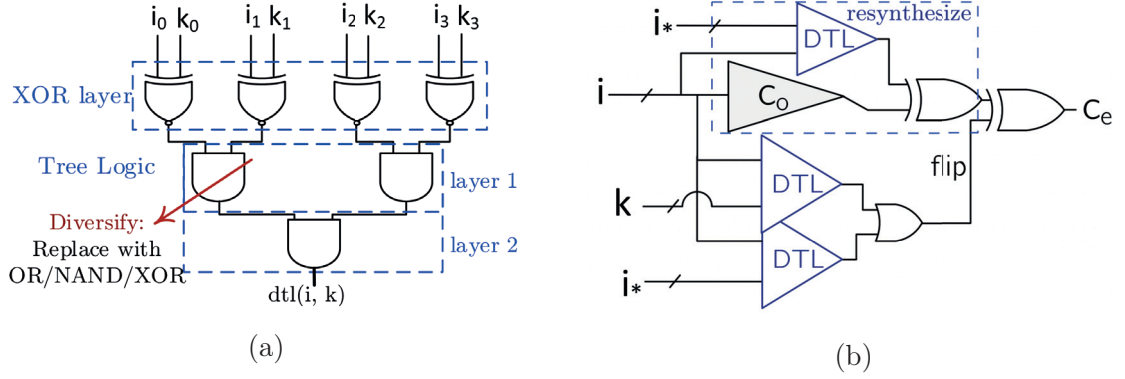


Figure 4.5: The diversified logic tree (DTL) (a) alterations to the logic tree and (b) the high level corrupt and correct DTL topology [107].

4.2.2 DTL

The diversified tree logic (DTL) method proposed in [107] utilizes the comparator based restore unit of SFLL-HD⁰, but replaces select gates within the tree logic to generate increased corruption at the outputs of the circuit. The modifications to the tree logic and the overall structure of the methodology are shown in Fig. 4.5a. Similar to the SFLL-HD methodology, the DTL topology shown in Fig. 4.5b utilizes a tree structure with a set key to perturb the original logic cone. However, the DTL tree structure with a set key is also a portion of the restore unit. Since the restore unit is easier to identify due to the topological structure of the key logic, the adversary is able to remove the DTL structure with the unknown key and still achieve the correct circuit functionality. To avoid removal based attacks, the DTL tree structure with the correct set key in the restore unit must be removed, leaving the DTL tree structure with an unknown key to serve as the entire restore unit.

Table 4.6: Key space of the DTL logic tree with all AND gates within the logic tree [107].

		$k[0, 7]$															
$i[0, 7]$		1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
		0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
		0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
		0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
		0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
		0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
		0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
		0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
		0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
		0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
		0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

The logic space of a DTL tree implemented as a standard AND-tree topology is shown in Table 4.6, and the resulting logic space when a single gate of the AND-tree is replaced with a NAND gate is shown in Table 4.7. Note that the logic space does not include the effects of a protected cube, which inverts the logic 0s and logic 1s for each protected input pattern. Just like SFLL-HD, the dimension of the key space is limited to a maximum of 2^n , where n is the number of inputs to the circuit. Since the space is limited to 2^n , modifying the tree topology with gate functions other than AND reduces the SAT resilience of the circuit while increasing the number of errors observed at the output of the IC.

Table 4.7: Key space of the DTL logic tree with a single NAND gate within the first layer of the tree logic [107].

		$k [0, 7]$															
$i [0, 7]$		1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
		1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
		1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
		0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
		0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
		0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0
		0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0
		0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0
		0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0
		0	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0
		0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0
		0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0
		0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
		0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1
		0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1
		0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

4.2.3 SFLL-Flex

In addition to SFLL-HD, SFLL-Flex is also introduced in [101], which modifies the restore unit to a look-up table (LUT). The altered topology implementing SFLL-Flex is shown in Fig. 4.6. The metric evaluating the security provided by SFLL-Flex as described in [101] estimates the amount of iterations required to determine a single protected cube. When only analyzing the security of a single protected cube, the estimated security of SFLL-Flex simplifies to that of SFLL-HD given by (4.1).

Estimating the security of SFLL-Flex by measuring the number of iterations to

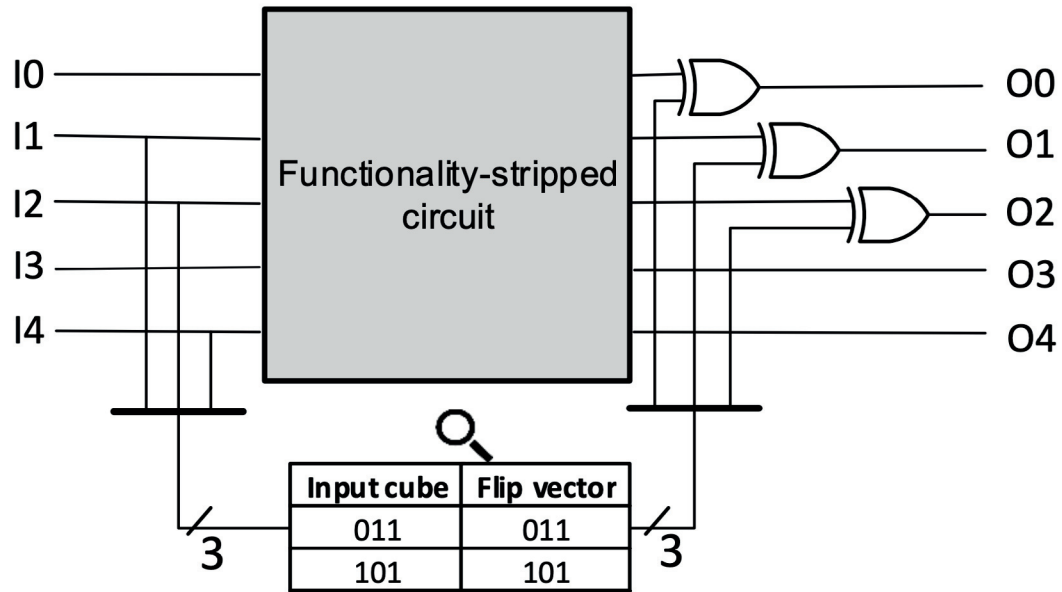


Figure 4.6: SFLL-Flex topology with 3 inputs connected to the LUT restore unit and three flip function vectors serving three outputs [101].

determine all of the protected cubes no longer simplifies to that of SFLL-HD. The primary cause of the disparity in the security of the two techniques is due to the resulting differences in the key spaces of the two methodologies. Unlike SFLL-HD and DTL, adding an additional protected cube increases the number of key bits applied to the circuit, which in turn increases the size of the key space. Since the key space expands beyond 2^n , where n is the number of inputs, the SAT resilience does not degrade when adding a protected cube, which is observed from the key space produced by SFLL-Flex for two protected cubes provided in Table 4.8.

While SFLL-Flex provides SAT resilience when the number of protected cubes c is increased, the overhead in power, performance, and area of adding a single protected

Table 4.8: Key space of SFLL-Flex for a two input OR gate with two protected cubes.

A	B	K0	K1	K2	K3	K4	K5	K6	K7	K8	K9
0	0	0	0	0	1	1	0	0	1	0	1
0	1	1	1	0	1	1	0	1	1	0	1
1	0	1	0	1	1	1	1	0	0	0	1
1	1	0	1	1	1	0	0	0	1	1	1

cube c at a time becomes prohibitive if attempting to increase the output corruption of the circuit when an incorrect key is applied. The alternative means of increasing the input-output pair corruption for SFLL-Flex is to reduce the amount of key bits k protecting each cube. While decreasing k allows for an efficient means of increasing the output corruption, the SAT resilience of the circuit then decreases as the key space an adversary must search is much smaller. For example, if requiring 50% corruption of the input-output pairs of a logic cone, the key space k reduces to one. An adversary is, therefore, easily able to brute force the combinations of the protected cube, as a cube with only a single input dependency has two possible values.

4.2.4 Structural Attacks on Modified Logic Cones

There are a variety of proposed structural attacks to determine the key of a logic locked IC [108–110] that target the fixed and predictable structure of the modifications made to the original, unobfuscated, logic cones of the circuit. The majority of the structural attacks are developed for the SFLL-HD^h topology. The current structural

attacks are capable of identifying the perturb unit and extracting the hard coded key without access to an oracle IC when resynthesis of the original logic cone to integrate the perturb unit does not produce significant topological mixing.

The work described in [108] proposes two different structural attacks. The first structural attack is based on the unnateness of the perturb unit of SFL- HD^0 , where the perturb unit simplifies to a single comparator. A Boolean function is described as positive unate with regard to x if switching x from 0 to 1, while leaving all other variables unchanged, never results in the function switching from 1 to 0. A negative unate function behaves exactly the opposite, where x transitioning from 0 to 1 never results in the function switching from 0 to 1 when all other variables are held constant. The perturb unit of SFL- HD^0 is a comparator, which consists of XNOR gates on the first layer of logic feeding into an AND-tree, which is shown in Fig. 4.5a, where each gate of the tree implements the same logical function. For a given key applied to the comparator, each input to the AND-tree is either in the positive (buffered) or negative (inverted) form. For example, consider a three input function with inputs A , B , and C and a protected cube of $ABC = 101$. The logic of the perturb unit is then $\bar{A} \wedge B \wedge \bar{C}$. When $A = 1$, the output of the perturb unit is logic 0, which implies that the function never switches from 0 to 1 when A transitions from 0 to 1, making A a negative unate. Following a similar analysis, B is positive unate and C is negative unate. Whether the input variable is negative or positive unate then also leaks the

value of the key bit, which allows for the extraction of the protected cube. Out of the 20 circuits analyzed in [108], the protected cube was extracted from 18.

In addition, an attack on SFL- HD^h that utilizes properties of the Hamming distance checker is proposed in [108]. The attack exploits the property of protected cubes being a Hamming distance h away from the key. Therefore, two different protected cubes are $2h$ apart from each other since both inputs are h distance away from the key and cannot equal one another [108]. Any bits where the two protected input cubes agree must, therefore, represent a portion of the correct key [108]. The work described in [109] also applies the $2h$ property to determine the key within $i - 1$ queries once the first protected cube is found, where i corresponds to the number of primary inputs. Once the attack finds the first protected cube, the $2h$ input patterns from that input pattern are utilized as DIPs until all the correct key bits are determined. As an example, consider the key space produced by SFL- HD^1 with a key of 010 as provided in Table 4.9. If the protected cube pattern of 011 is initially determined, then the adversary is able to search the patterns 000, 101, and 110 that are $2h$ distance from 011. Examining any two of the three $2h$ patterns then reveals the correct key, since the constraints limit the SAT solver to picking K2 in Table 4.9.

A functional reverse engineering approach is utilized in [110] instead of the separation in the Hamming distance. The BSIM tool described in [111] is applied to identify comparators and adders within the circuit, which constitute components of

Table 4.9: SFLL-HD¹ key space for a three input function with a correct key of 010. The protected cube patterns are shown in red and the key value is shown in green.

A	B	C	K0	K1	K2	K3	K4	K5	K6	K7
0	0	0	X	✓	✓	X	X	X	✓	X
0	0	1	X	✓	✓	X	✓	X	✓	✓
0	1	0	X	✓	✓	X	✓	✓	X	✓
0	1	1	X	✓	✓	X	X	X	X	✓
1	0	0	X	✓	✓	✓	✓	X	X	✓
1	0	1	✓	X	✓	✓	X	✓	✓	X
1	1	0	X	X	✓	X	✓	X	X	✓
1	1	1	✓	✓	✓	X	✓	X	X	✓

the circuit that implement the Hamming distance perturb and restore unit, allowing for a greatly reduced search space for the perturb unit logic.

Structural attacks are prevented using cryptographic techniques including one-way functions and an AES block as described in [109] and [87], respectively. Both one-way functions and AES increase the difficulty of determining the input conditions of the circuit from the output constraints, but do not integrate well with the original logic cone. Therefore, modified approaches such as described in [110] allow for identification of the perturb unit, which is then simply removed, exposing the original logic cone to an adversary.

The work described in [112] prevents structural attacks by intentionally inducing a stuck-at fault into the original logic circuit. An example of applying a stuck-at-1 (sa1) fault to net N11 of the ISCAS'85 c17 benchmark is shown in Fig. 4.7. The

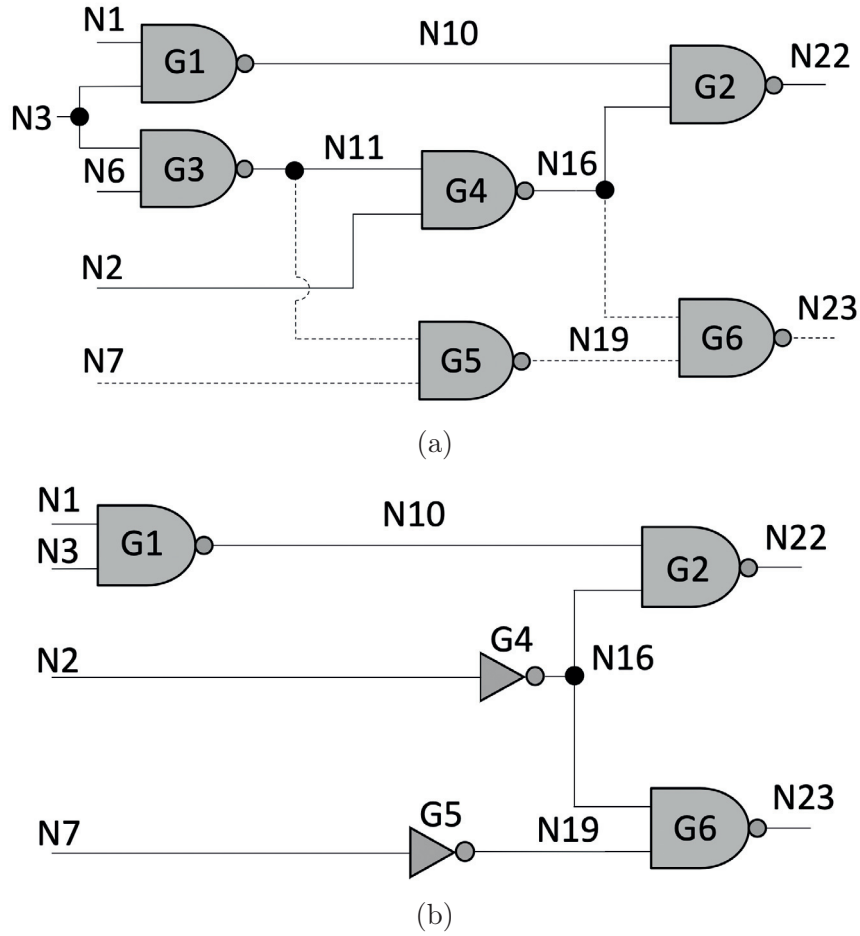


Figure 4.7: The (a) unmodified ISCAS'85 c17 benchmark and (b) the c17 benchmark with a stuck-at-1 (sa1) fault applied to N11.

introduction of the stuck-at fault guarantees modification to the original logic cone without requiring re-synthesis to implement the changes to the original circuit. Such a technique is referred to as subtractive, as logic is removed from the original logic cone rather than adding modifying elements that are integrated through resynthesis. The primary drawback of the SFLL-Fault methodology is the run-time required to select a fault that is correctable with a limited flip function circuit, as over an hour

was required per fault on the evaluated benchmark circuits [112]. The limitation is illustrated from the circuit shown in Fig. 4.7b, where applying the sa1 fault results in the simplification of G4 and G5 to inverters and the elimination of G3 and N6. Finding a location to insert the fault then requires the trial of many nets within a circuit until one with few protected cubes is determined. The modification of additive measures to protect against structural attacks while limiting the overhead in design time is discussed in Chapter 9.

Chapter 5: Sequential Logic Locking

While the in-cone logic locking techniques described in Section 2.4 modify the combinational logic, the methods discussed in this chapter lock the sequential logic of an IC. Sequential logic locking techniques target a specific sequence of states or the current state of a finite state machine (FSM) to alter the behavior of the circuit. The chapter begins with an introduction to sequential logic and continues with a discussion of sequential logic locking techniques and vulnerabilities.

5.1 Introduction to Sequential Logic

Traditional combinational logic depends only on the input applied to the circuit, where applying a specific input pattern results in a corresponding output from the circuit. However, many functions are not efficiently represented without feedback from internal signals of the given function. For example, consider the combinational circuit shown in Fig. 5.1b that outputs the next value from 0 to 3 when the enable EN signal is high and outputs 0 when 3 ($AB = 11$) is applied to the inputs of the circuit. When the EN signal is low, the circuit outputs the current values of A and B to A_OUT and B_OUT, respectively. However, the circuit representation shown in Fig. 5.1b can only output the next value. For example, applying $AB = 00$ when $EN = 1$ only outputs 01. If the combinational circuit had to transition between

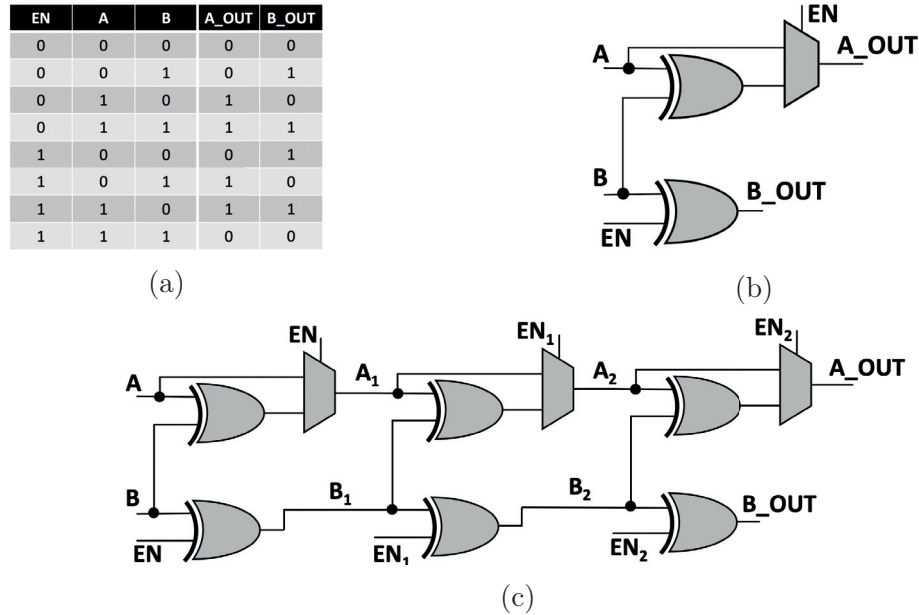


Figure 5.1: The (a) truth table of a 2-bit counter with an enable (EN) and inputs A and B , (b) the combinational logic implementing the circuit, and (c) the unrolled combinational circuit.

multiple values, such as from $AB = 00$ to $AB = 11$, the circuit must be replicated twice to accommodate the transitions, as shown in Fig. 5.1c, which is referred to as an unrolling of the circuit. The need to replicate the circuit results in significant overhead in area and power. Furthermore, the circuit shown in Fig. 5.1c also requires additional inputs and outputs to provide adequate control of the functionality. For example, to produce an output after one, two, or three executions of the circuit, the A_1 , A_2 , A_OUT , B_1 , B_2 , and B_OUT signals all need to be accessible at the outputs of the circuit.

To create a feedback path within the circuit, a flip-flop is utilized to provide a

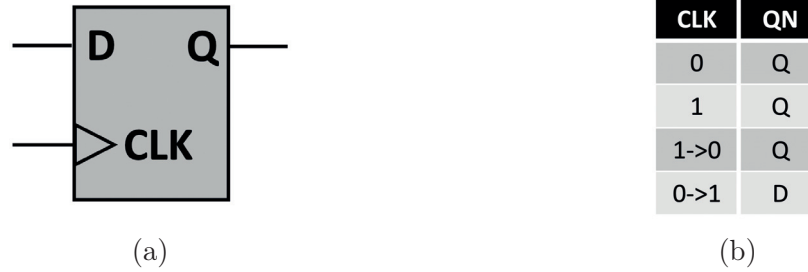


Figure 5.2: The (a) circuit symbol of a D flip-flop and (b) the corresponding truth table representation of the functionality of the D flip-flop.

sequential dependency. The truth table and circuit symbol of a D flip-flop are shown in Fig. 5.2. In the case of a D flip-flop, the state is only updated on the rising edge of a clock CLK signal. The insertion of flip-flops into a circuit removes the need to unroll the combinational logic, which results in the circuit shown in Fig. 5.3a. Not requiring the unrolling of a circuit results in less combinational logic, which reduces the power and area needed to replicate the functionality of the circuit. Another benefit of implementing the sequential circuit shown in Fig. 5.3a over the unrolled combinational circuit shown in Fig. 5.1c is that the sequential circuit functions for any number of time steps, whereas the unrolled combinational circuit provides an output for only three iterations of the clock.

A function, such as that shown in Fig. 5.3a, that relies on the internal state of a circuit is termed a finite state machine (FSM). In addition to the truth table representation of the function shown in Fig. 5.1, an FSM is also typically represented as a state transition diagram or graph. The state transition graph (STG) for the

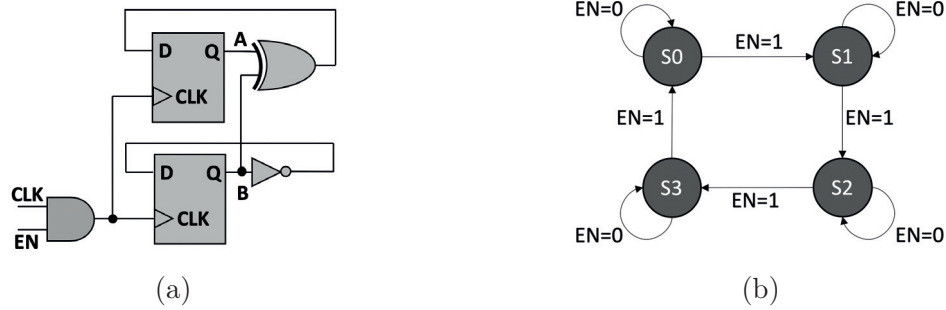


Figure 5.3: (a) A 2-bit counter with an enable bit represented with two D flip-flops and (b) the corresponding state transition graph (STG) representation.

state counter circuit is shown in Fig. 5.3b.

The challenge of including an internal circuit state is that the state is not inherently observable or controllable through the primary inputs and outputs of the IC, which results in challenges when testing the functionality of the circuit. A scan chain structure is, therefore, inserted to improve the observability and controllability of the internal states of the circuit. One implementation of a scan chain utilizes a multiplexer (MUX) based architecture [113], as shown in Fig. 5.4a.

The inserted scan chain allows for the control and subsequent observation of the internal states of the circuit using a shift register like structure. The typical activation pattern of a scan chain is shown in Fig. 5.4b. The scan enable (SE) signal is first applied high to set the flip-flops to shift register mode. Data is then read in from the scan input (SI) port shown in Fig. 5.4a. A slower frequency is applied in test mode to limit the power consumption and control the temperature of the IC as the greater switching activity during test mode results in as much as a 2x increase in the nominal

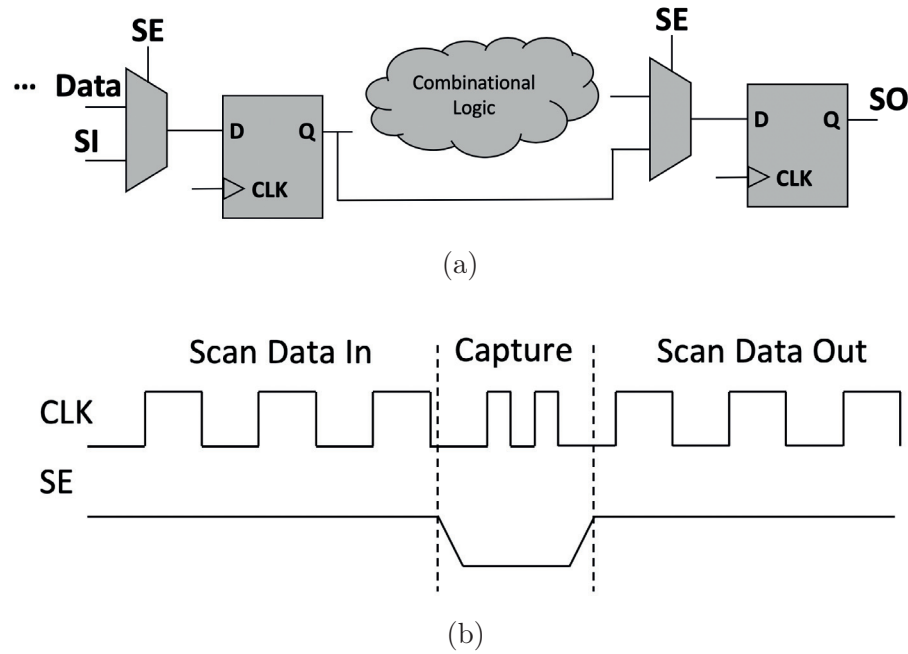


Figure 5.4: Representation of (a) the mux-based scan circuitry implemented to output the internal state of an IC, where the scan enable (SE), clock (CLK), scan in (SI), and scan out (SO) signals are applied, and (b) the activation pattern of the scan chain. When SE is high, the flip-flops form a scan chain and sequentially output the internal state of the IC through the SO pin. Activation of the scan chain begins with SE set to logic high to scan in and store a specific data pattern to the registers. SE is then transitioned to logic low to capture the circuit response through the combinational logic at the functional frequency of the circuit, and then switched back to logic high again to scan out the test response.

power consumption [114]. The SE signal is then set to low, with the clock (CLK) returning to a functional frequency to capture the at-speed response of the IC for the input vector provided through the scan chain. After the response is captured, the SE signal is once again set to high while the clock signal is returned to the frequency used for test, which allows for the readout of the captured response through the scan out (SO) port.

5.2 Sequential Logic Locking Methodologies

The traversal through an FSM is dependent on the input, starting state, and state transition logic. Finding a single sequence out of the many possible sequences within the FSM, therefore, results in a large search space, which provides an opportunity to enhance the security of the circuit. In addition, the fundamental component of the SAT attack discussed in Chapter 3.3, specifically the SAT solver, struggles with feedback paths within the logic as the solution does not converge to a single value. Multiple methodologies to capitalize on the potential security provided by sequential dependencies have been developed, with methods falling into two primary categories: 1) FSM partitioning, and 2) application of incorrect state transitions within the original FSM. Prior work in both categories is discussed in this chapter, along with vulnerabilities of each methodology.

5.2.1 FSM Partitioning

When utilizing a key sequence to activate an IC, the adversary must be limited to observe only partially correct circuit functionality before the key sequence is applied. A method that allows for differences in circuit functionality based on an applied key sequence splits the FSM into an obfuscated state space and a functional state space. The division of the state space into obfuscated and functional modes is referred to as FSM partitioning, where only the correct key sequence transitions the circuit from

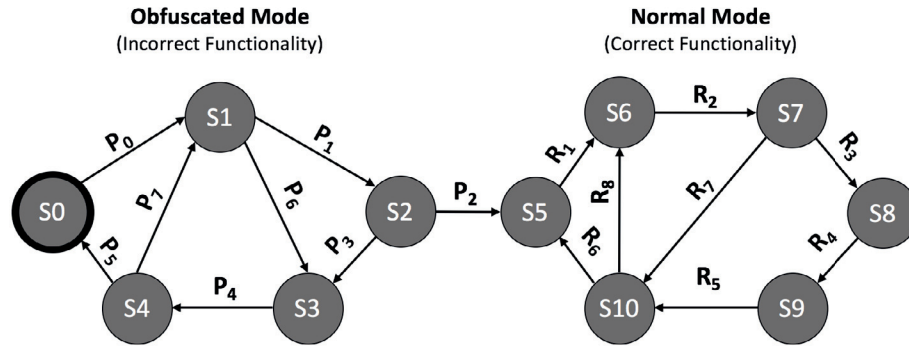


Figure 5.5: Modified FSM with an enabling sequence of P0, P1, and P2 required to enter normal circuit operation [115].

the obfuscated state space to the functional state space. Varying methodologies to create partitioned state spaces are described in this section.

5.2.1.a HARPOON

An example of FSM partitioning is shown in Fig. 5.5, where correct operation is only achieved after applying the key sequence of P0, P1, and P2. The depicted partitioning is implemented by applying the HARPOON sequential logic locking technique [115], which utilizes modifying cells to alter the combinational logic response when in obfuscated mode. The logical structure of a modification cell is shown in Fig. 5.6a, which alters the function based on the application of an enable signal. When the enable signal is high, the output function of the modification cell (f_{mod}) is altered to $\bar{f}g$, which prevents an adversary from achieving functional operation until the correct start-up sequence is applied, as shown in Fig. 5.5.

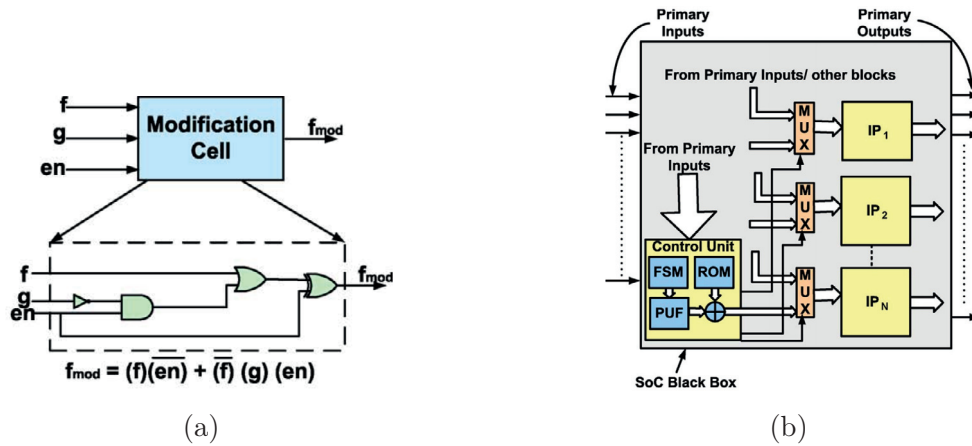


Figure 5.6: The (a) modification cell structure and (b) control circuit diagram for HARPOON [115].

The control of the obfuscated FSM is performed by the logic structure shown in Fig. 5.6b, which combines the initial FSM state and the key bits stored in a ROM. The output of the FSM is provided as input to a physically unclonable function (PUF), which utilizes process variations present in each circuit to generate a unique signature for each IC. Background material on PUFs and the various types of PUF structures is provided in [116]. The output of the control circuit then activates the IP of the circuit by correctly configuring the modification functions to operate within the functional mode of the partitioned FSM.

5.2.1.b Interlocking Obfuscation

Another partitioned FSM scheme is proposed in [117] that alters the functional mode to rely on a code word as shown in Fig. 5.7. The path through the obfuscated

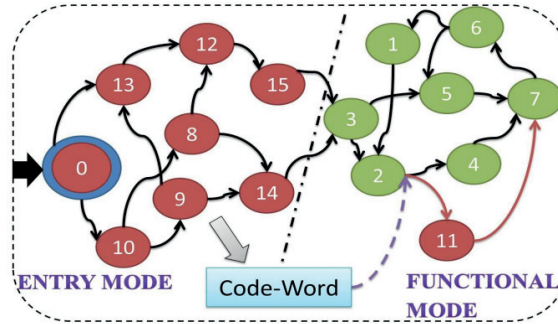


Figure 5.7: Partitioned FSM methodology that generates a code word when transitioning through the states of the obfuscated mode. Transitioning from state 2 is dependent on the value of the code word [117].

mode, the entry mode in Fig. 5.7, determines the value of the code word. The addition of the obfuscated mode and the code word is performed at the register-transfer level (RTL), which forces the logic synthesis tools to implement the functionality of the modified FSM as a logical representation. For example, a circuit with four state registers and a single input includes modified RTL that sets the code word to 0100 if the current state is equal to 0010 and the input is equal to 1. Each state transition modifies the code word to a specific value, and if the circuit enters the functional mode without the correct code word then an incorrect transition still exists within the functional mode FSM. The incorrect transition is generated by XORing the code word with the state value, which is equivalent to XOR based in-cone logic locking.

5.2.2 Vulnerabilities of FSM Partitioning

Methods that implement partitioned FSMs are potentially vulnerable to FSM extraction [118], fault-injection attacks [119–121], and structural leakage from the inserted security logic elements. Techniques that execute FSM extraction explore the state space through analysis and characterization of the state transition logic between the registers of the FSM. As the size of the FSM increases, analysis of the possible transitions becomes exponentially more computationally expensive, which reduces the vulnerability of the FSM to extraction.

Execution of a fault-injection attack without knowledge of the FSM state transitions is possible, as the objective is to transition to the activated mode of the partitioned FSM. However, the adversary must have knowledge of the states that fall within the functional and obfuscated state spaces of the partitioned FSM. An adversary is able to determine the starting state of the activated circuit by resetting the IC in scan mode and reading out the state. Glitching the IC into the specific starting state from the obfuscated state space, however, becomes progressively more difficult as the number of states increases. In addition, even if an adversary is able to glitch into the starting state of the IC, the activation key sequence has not been determined or applied, which implies that the correct functionality of the circuit is only observable until the IC is returned to the obfuscated state after a power-off event.

Fault-injection attacks are also thwarted by introducing *black hole* states to an

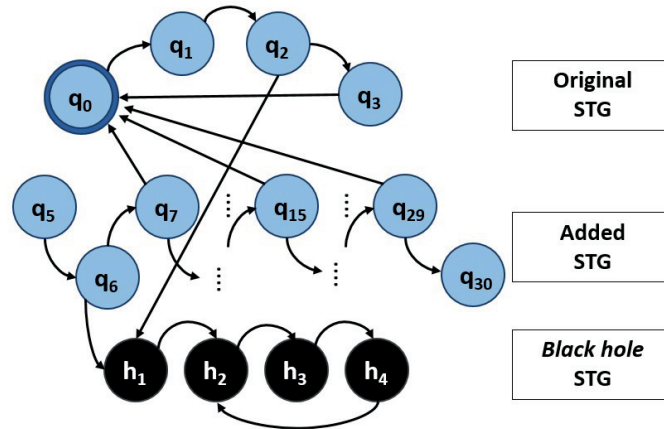


Figure 5.8: Modified FSM with black hole states that never return to the original state transition graph (STG) [122].

FSM, as shown in Fig. 5.8, which are never functionally exited, or *gray hole* states, which are difficult to functionally exit [122]. If an adversary glitches into a *black hole* or *gray hole* state, the circuit does not function as intended. The inclusion of separate keys for the obfuscated and functional modes of the partitioned state machine is also possible, where even if an adversary is able to transfer into the functional mode, applying an incorrect key in the functional mode transitions the FSM back into the obfuscated mode [123].

The technique described in [117] prevents fault-injection attacks by requiring a code word for correct operation when in the functional operating mode of the circuit. The code is determined by the sequence applied during the obfuscated operating mode, which implies that if the correct code is not entered when bypassing the obfuscated mode of the FSM with a fault, then the functional mode does not result in

proper operation. When considering full scan chain access and access to an oracle IC, the code word for the technique described in [117] is vulnerable to a SAT attack, as the key dependency is equivalent to XOR based in-cone logic locking. Additional background on SAT based oracle guided attacks is available in Section 3.3. The determination of the code word provides the adversary with the activation key sequence needed to reach the functional state space.

When an adversary has reverse engineered the IC, partitioned state machines require additional security against attacks that leverage the logical structure of the circuit. For example, in the case of HARPOON [115], the interconnects that modify the FSM are determined from the connections to the physically unclonable function (PUF) and the read only memory (ROM). The ability to determine the added interconnects that connect to the logic modification cells provides an adversary with the information needed to remove the circuitry. In addition, the determination of the state transition from the obfuscated mode to the functional mode is possible when the starting state only includes a small number of logical connections. For example, consider the example FSM shown in Fig. 5.5, where only $S10$ and $S2$ logically transition to the starting state $S5$. Since $S10$ and $S5$ both transition to $S6$, the only possible transition from the obfuscated mode to the functional mode is $S2$ to $S5$.

5.2.3 Introducing Incorrect State Transitions into the FSM

As opposed to generating distinct partitions separating the obfuscated and active states of an IC, as was discussed in Section 5.2.1, the work described in [124–126] implement modifications to the functionality of the original FSM. The FSM then includes a mix of correct STG transitions as well as incorrect transitions that are only taken when an erroneous key is applied. An overview of techniques that modify transitions within the STG is provided in this section, as well as a discussion of vulnerabilities.

5.2.3.a Remote Activation of ICs

The work described in [126] alters the STG through state replication to prevent an unauthorized user from gaining correct functionality of the IC. The modifications to the circuit are shown in Fig. 5.9a, where an unclonable random unique block (RUB) is inserted into the circuit. The topology of the RUB is shown in Fig. 5.9b, where delay elements and arbiters are utilized to effect the value of the output bits. The variations in delay across different ICs results in the RUB producing different outputs, which in turn generates unintended state transitions within an IC. For example, if both outputs from a SEL element shown in Fig. 5.9b have the same delay, then the arbiter theoretically fluctuates between choosing each delay line. However, process variations within the IC results in one line having more delay than the other, which

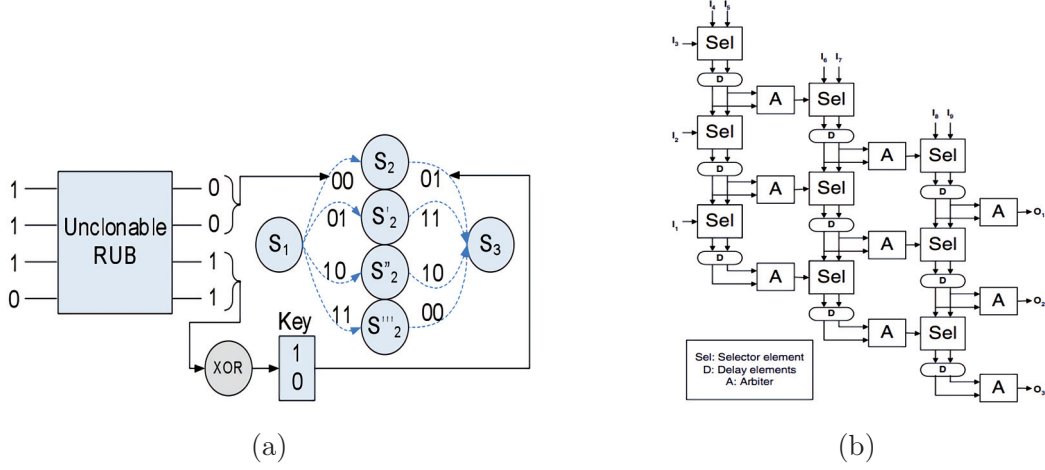


Figure 5.9: A depiction of a (a) modified FSM with replicated states and RUB to alter the state transition, and (b) block diagram representation of the RUB [126]. The key bits are utilized to correct the output of the RUB to ensure correct transition.

results in a more consistent output from the arbiter. The path taken through the RUB is then dependent on the individual variations in the paths of the IC. In order to generate the correct state transition within the IC, the output of the RUB block is XORed with the key value, as shown in Fig. 5.9a.

5.2.3.b Revisiting Sequential Logic Obfuscation

A technique that forces an adversary to unroll the FSM by an additional number of cycles is described in [124]. As discussed in the introduction to sequential logic, each unrolled instance adds new logic and input signals, which expands the number of constraints required by the SAT solver. The increase in the number of constraints results in an increase in the time to solve the expression, which makes the SAT attack

less efficient. The number of times the circuit has to be unrolled is then defined as the minimum depth between the reset state and the locked state [124]. An example FSM is shown in Fig. 5.10 that requires two time steps to transition from the reset state rst to state s_3 . Note that if the starting state is set to rst , then unrolling to allow for a two time step window does not provide information on o_3 as the shortest path from rst to o_3 requires three time steps. Since the adversary is unaware of the number of time steps required to access each of the FSM states, the recovery of the correct FSM by an adversary is, therefore, very difficult. Rather, the SAT attack only guarantees that the states that are reached within the amount of unrolled time steps exhibit the correct response. The work described in [124] does not provide details on the modifications required to the state transition logic that introduce incorrect transitions. As previously discussed, if the state values are conditionally XORed, then the key is vulnerable to the SAT attack. In addition, unrolling is only required if access to the scan chain of an IC is restricted. Otherwise, modification to the starting state of the IC by an adversary allows all states to be reached within a single time step.

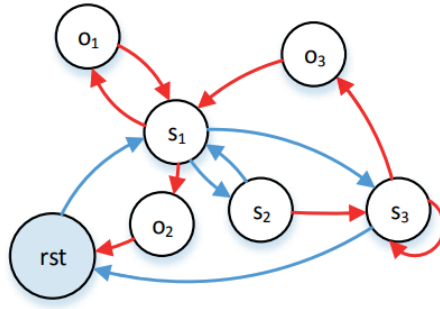


Figure 5.10: Modified FSM to increase the number of unrollings required to successfully execute a SAT attack when scan chain access is restricted [124].

5.2.3.c Dynamic State-Deflection

The work in [125] utilizes dynamic state deflection, which transitions into a *black hole* state cluster if an incorrect key vector is applied. The STG of an FSM utilizing dynamic state deflections is shown in Fig. 5.11a, where each state transition is dependent on the applied key. When an incorrect key is applied to the FSM, a mapping function *MapFunc* and rotation function *RotateFunc* are applied to dynamically alter the state bits. The *MapFunc* and *RotateFunc* determine the value of the *FlipReg* and *FlipOut* vectors, which flip the register values or output values, respectively, with an XOR gate, as shown in Fig. 5.11b.

5.2.4 Vulnerabilities of Incorrect Transitions within an FSM

Under a threat model where all register values are accessible through the scan chain, all of the sequential logic locking techniques that introduce incorrect state

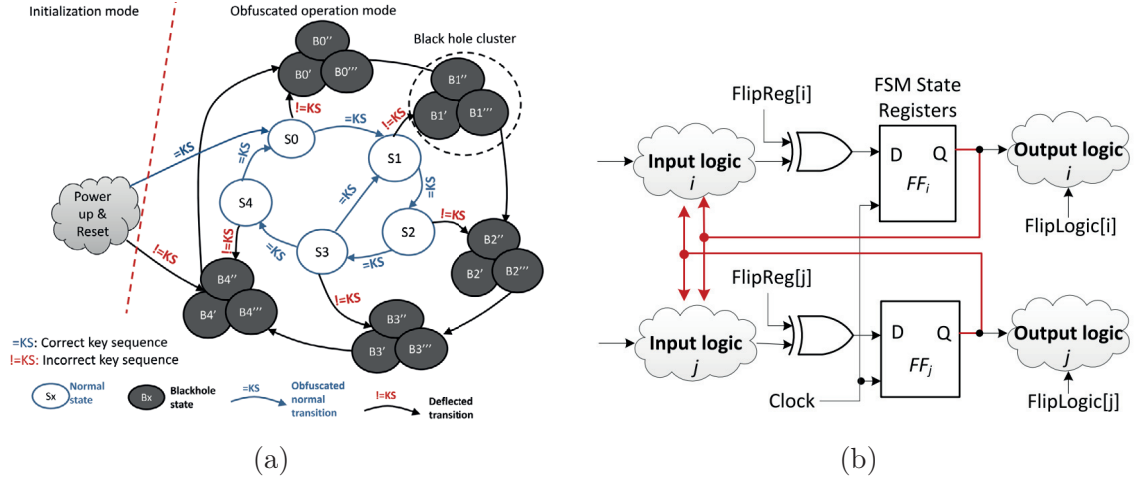


Figure 5.11: The (a) STG of an FSM implementing dynamic state-deflections as proposed in [125] and (b) the circuit modifications that allow for dynamic state deflections.

transitions are vulnerable to SAT based attacks. The vulnerability arises as the circuit is effectively only secured through combinational locking, with the in-cone key dependencies efficiently determined by SAT based attacks due to the logical masking that results from the distinguishing input patterns (DIPs), which is discussed in greater detail in Chapter 7.

In addition, the techniques described in [125] and [126] primarily account for unauthorized activation assuming a black box model. However, under a threat model where the IC is reverse engineered, the techniques are vulnerable to structural attacks, where an adversary is able to identify the memory locations containing the key and determine the added circuitry for sequential locking. In [126], the RUB circuitry may be identified and removed, which allows an adversary to determine the key with a

Table 5.1: Comparison of the vulnerability of different sequential logic locking methodologies to an unauthorized activation executed through fault-injection or FSM extraction. The reverse engineering of an IC is also considered through structural/removal and scan chain access attacks.

Locking Methodology	Unauthorized Activation		Reverse Engineering	
	Fault Injection	FSM Extraction	Removal/Structural	Scan Chain
HARPOON [115]	Medium	Medium	High	High
Code Word [117]	Low	Low	Medium	High
Remote Activation [126]	Low	Low	High	High
Revisiting Sequential Logic Obfuscation [124]	Low	Low	Low	High
Dynamic State Deflection [125]	Low	Low	High	Medium

SAT attack. Similarly, the *FlipReg* signals described in [125] and shown in Fig. 5.11b are vulnerable to structural attack, since the signals must be set to logic 0 for correct functionality. The *FlipLogic* signals are then determined by a SAT attack as the signals are generated with in-cone logic locking.

Depending on the implementation of the modifications of the FSM described in [124], the implemented changes are potentially vulnerable to structural attack. As discussed, if the original transition is XORed with a key, then effective execution of a SAT attack is possible when access to the scan chain is available. In addition, modifications to the FSM made by comparing the state values and adding logic that is dependent on the activation state are possible. If the added logic does not integrate well with the original logic cone, then the identification and removal of the added logic exposes the original state transition logic to the adversary.

A summary of the vulnerabilities of various sequential logic locking methodologies that are based on partitioned FSMs and incorrect transitions is provided in

Table 5.1. All of the developed sequential logic locking techniques provide resilience against unauthorized activation when the adversary is not able to reverse engineer the netlist. When utilizing HARPOON [115], fault injection attacks are a concern as no modifications are made to the functional FSM. In addition, when the FSM is small, extraction of the FSM is possible. However, executing an FSM extraction attack becomes less effective as the size of the FSM increases. Since [117] modifies the original FSM, fault injection and FSM extraction will not recover the correct functionality of the FSM.

While the discussed sequential logic locking techniques assist in preventing unauthorized activation, the techniques either exhibit structural vulnerabilities when the threat of reverse engineering is considered, or the added security elements are neutralized through scan chain access. The development of a sequential logic locking methodology that reduces the structural information of a circuit leaked through reverse engineering is described in Chapter 8 . In addition, the technique reduces the amount of information provided through the scan chain.

Chapter 6: Reduced Overhead In-Cone Logic Locking

While the attacks against in-cone logic locking described in Chapter 3 demonstrate serious vulnerabilities, the attacks are only pertinent when an oracle activated IC is available to an adversary. Many threat models, including those considered for military ICs, may never be exposed to the public or untrusted third parties in an activated state. For threat models where an oracle IC is not available, the gate selection algorithms discussed in Chapter 2 are applicable to meet the given security requirements.

The primary limitation of applying logic locking techniques when the threat of an adversary gaining access to an oracle IC is not pertinent is the resulting overhead in power, area, and performance of implementing the security features. For example, the performance and area of standard un-obfuscated gates are listed in Table 6.1. In addition, both the average power and average leakage power of the un-obfuscated standard cell gates are provided in Table 6.2.

Table 6.1: Analysis of the propagation delay and area of standard cells from a 180 nm fabrication process.

Standard Cell	Propagation Delay (ps)	Area (μm^2)
AND	69.79	30.73
NAND	36.71	23.25
OR	92.90	30.73
NOR	42.09	23.25
XOR	91.23	45.69
XNOR	106.7	41.62

Table 6.2: Analysis of the average and leakage power of standard cells from a 180 nm fabrication process.

Standard Cell	Average Power (nW)	Average Leakage Power (pW)
AND	70.40	192.5
NAND	72.67	149.7
OR	64.52	252.3
NOR	96.85	193.6
XOR	148.0	455.3
XNOR	204.6	527.6

Table 6.3: Analysis of the propagation delay, power, and area of XOR based logic locking. Per-gate overheads are provided as percent increases over the standard cell values listed in Tables 6.1 and 6.2.

Logic Cell	Prop. Delay (ps)	Power (nW)	Area (μm^2)
AND	151.3 (116.8%)	150.4 (113.6%)	63.73 (107.4%)
NAND	127.6 (247.6%)	142.2 (95.68%)	63.73 (174.1%)
OR	157.5 (69.54%)	174.6 (170.6%)	63.73 (107.4%)
NOR	134.3 (219.1%)	168.5 (73.98%)	63.73 (174.1%)
XOR	181.8 (99.27%)	219.3 (48.18%)	84.50 (84.94%)
XNOR	201.3 (88.66%)	226.3 (10.61%)	84.50 (101.4%)
Average	140.2%	85.45%	124.9%

The circuits presented are implemented in a 180 nm technology. The power and performance were determined by matching the drive strength of the gates to ensure a fair analysis between the standard cells and the equivalent logic locking topologies. The following simulation characteristics were applied:

1. All simulations were completed with a load capacitance of 5 fF.
2. The worst case propagation delay is determined from the low to high and high to low transient delays.

3. The area was obtained from a layout of the implemented topologies.
4. The duration and input transitions are held constant for each gate to measure average power.
5. The highest average leakage power is reported from all possible input combinations to a gate.

Note that for the analysis of the overhead of the XOR and LUT logic locking techniques, no inverters were added after the XOR logic locking gates and no memory elements were used for the LUT based approach. The values provided for the per-gate overhead of each method are, therefore, highly optimistic as compared to implementations that include the inverters and memory.

The per-gate overheads in power, performance, and area of the XOR based logic locking technique are listed in Table 6.3. Similarly, the overheads due to implementation of the LUT based logic locking method are listed in Table 6.4. The leakage power for both the XOR and LUT logic locking techniques is provided in Table 6.5.

The large overheads in power, area, and performance indicated by the results listed in Tables 6.3, 6.4, and 6.5 limit the use of the XOR and LUT logic locking techniques in many IC applications. In addition, the large per-gate overhead in propagation delay of implementing in-cone logic locking limits the acceptable signal paths to place gates without decreasing performance. The total number of locked gates placed in a circuit is limited by power and area constraints as well. As ICs must

Table 6.4: Analysis of the propagation delay, power, and area of 4x1 MUX based logic locking. Per-gate overheads are provided as percent increases over the standard cell values in Tables 6.1 and 6.2.

Logic Cell	Prop. Delay (ps)	Power (nW)	Area (μm^2)
AND	122.5 (75.53%)	146.0 (107.4%)	90.58 (194.8%)
NAND	124.6 (239.4%)	157.0 (116.0%)	90.58 (289.6%)
OR	120.8 (30.03%)	142.2 (120.4%)	90.58 (194.8%)
NOR	126.8 (201.3%)	158.8 (63.96%)	90.58 (289.6%)
XOR	124.0 (35.92%)	191.8 (29.59%)	90.58 (98.25%)
XNOR	124.6 (16.78%)	191.7 (6.310%)	90.58 (115.9%)
Average	99.82%	73.95%	197.2%

Table 6.5: Analysis of the average leakage power of the XOR and LUT based logic locking methods. Per-gate overheads are provided as percent increases over the standard cell values listed in Table 6.2.

Logic Cell	XOR Leakage (pW)	LUT Leakage (pW)
AND	673.8 (250.0%)	895.8 (365.3%)
NAND	581.0 (288.1%)	841.8 (462.3%)
OR	661.5 (162.2%)	863.4 (242.2%)
NOR	648.9 (235.2%)	973.7 (402.9%)

meet performance, power, and area constraints, it is, therefore, imperative to reduce the per-gate overhead of logic locking techniques.

6.1 Gate Level Logic Locking

The motivation for gate level logic locking is due to the large performance, power, and area overheads associated with the XOR (see Fig. 2.4a) and LUT (see Fig. 2.4b) based logic locking topologies. Consider a standard, non-locked, AND gate as a way

to demonstrate the inefficiencies of current logic locking methodologies. The XOR method shown in Fig. 2.4a adds an additional XOR gate after the AND gate to mask OUT0 with KEY0. However, the data must now propagate through both the AND and XOR gates, resulting in a reduction in performance. The added XOR gate also contributes power and area over the original non-secured AND gate.

The LUT based approach replaces the AND gate with a 4x1 MUX structure as shown in Fig. 2.4b, which is used to implement any two input function. While the per-gate security is enhanced, the performance, power, and area are negatively affected by the additional transistors required to implement a given function. Two novel gate level logic locking methodologies are described in this section that enhance the security of a gate while also reducing the power, performance, and area overheads as compared to both the LUT and XOR techniques.

6.1.1 Stack-based Topology

The first topology is termed *stack-based* (see Fig. 6.1) as portions, or stacks, of logic are turned off/on depending on the key input. The topology depicted in Fig. 6.1 functions as either a NAND or NOR gate depending on the value of KEY0. When KEY0 is 0, the PMOS stack is activated allowing the gate to behave as a NAND. When KEY0 is set to 1, the NMOS stack is activated and the gate behaves as a NOR. Placing the KEY0 transistors on the OUT net of the gate reduces the

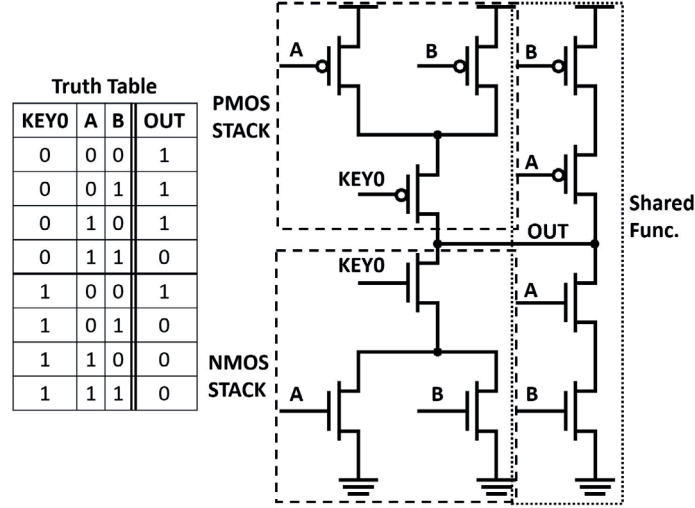


Figure 6.1: Stack-based topology implementing a NAND or NOR gate depending on the value of KEY0.

capacitance connected to OUT by essentially disconnecting one of the logic stacks during execution. A smaller output capacitance reduces the performance and power overhead of implementing stack based logic locking.

The NAND/NOR stack-based topology has two important characteristics: 1) The ability to share common input-output combinations between implemented logical functions, and 2) implementations that do not require negated inputs. For example, the NAND and NOR gates have the same logical output when inputs A and B are both 0 or 1, which permits shared functionality as indicated by the fine dashed box (*shared func.*) shown in Fig. 6.1. The shared logic does not require a key transistor as the NAND and NOR produce the same output for both the 00 and 11 input combinations. The ability to forgo the key transistor reduces the area, power, and

performance overheads of utilizing the stack-based approach.

The second characteristic of the NAND/NOR topology, is not requiring negated inputs of A and B , which removes two additional inverters from the circuit and further reduces the overhead of the methodology. The negated inputs are required if a NAND/AND stack-based topology is implemented instead. The negated logic is needed as the same input combinations must either turn on a PMOS or NMOS stack depending on the key.

While both characteristics of the stack-based topology allow for the reduction in the overhead in performance, power, and area of the NAND/NOR gate, sharing the two common output cases when inputs A and B are both 0 or 1 reduces the percentage of inputs that produce incorrect outputs when an incorrect key is applied. Both characteristics that reduce the overhead must be ignored to achieve complete output corruption when an incorrect key is entered, which results in a larger overhead in power and performance.

6.1.2 Transmission Gate Topology

The transmission gate topology shown in Fig. 6.2 is better suited to completely corrupt the outputs of the implemented circuit functions. The key value is passed through the transmission gates, eliminating the additional key transistors required in the stack-based topology. The removal of the key transistors reduces the overhead

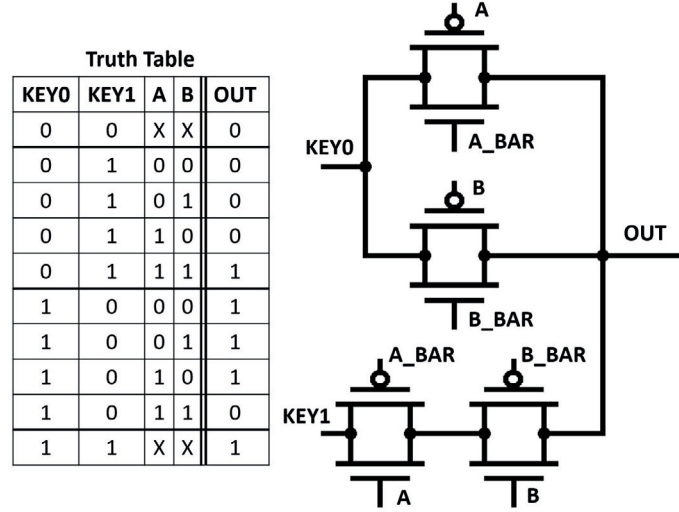


Figure 6.2: Transmission gate based topology implementing an AND/NAND depending on the values of KEY0 and KEY1.

in the performance, power, and area, making the transmission gate topology better suited to replicate the functionality of the XOR based logic locking method.

The transmission gate topology shown in Fig. 6.2 implements either an AND or a NAND gate depending on the values of KEY0 and KEY1. When KEY0 is 0 and KEY1 is 1, the topology behaves as an AND gate. When KEY0 is 1 and KEY1 is 0, the topology behaves as a NAND gate. Note that if KEY0 and KEY1 are 0 the AND/NAND topology produces a constant low on OUT. Similarly, a constant high output is achieved when KEY0 and KEY1 are 1. The use cases for the two constant outputs are described in Section 6.2.

The transmission gate topology is modified slightly to implement a locked OR/NOR. For the AND/NAND topology, if logic low is applied to either input A or input B the

value of KEY0 is passed to OUT. Similarly, when input A and input B are both logic high, the value of KEY1 is passed to OUT. A logic locked OR/NOR gate is achieved by passing the value of KEY0 when input A or B is logic high, and the value of KEY1 when inputs A and B are both low.

While the implementation of a NAND, AND, OR, or NOR gate with transmission gate logic provides a means to reduce the power, performance, and area overhead, locking an XOR or XNOR gate does not include the same logic simplifications. The only simplification that is possible for the XOR or XNOR gates is to eliminate one of the key inputs by tying the KEY1 and KEY2 inputs in Fig. 2.4b together. The reduction in the number of key values is beneficial when routing congestion is a concern.

6.2 Key Expansion

The transmission gate topology shown in Fig. 6.2 has the ability to output a constant logic low or a constant logic high independent of the inputs A and B . The number of key combinations an adversary must consider effectively doubles when utilizing the constant outputs of the transmission gate topology, while still achieving the reduction in overhead provided by gate level logic locking. The ability to utilize the constant outputs to expand the key search space of an adversary is coined key expansion. Circuit topologies that utilize the constant low and constant high output

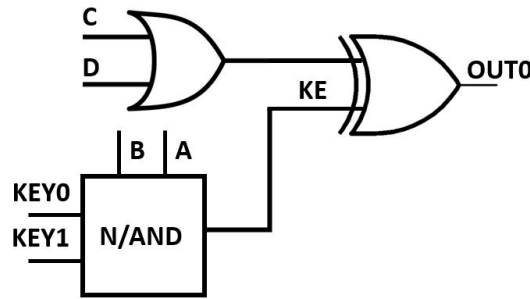


Figure 6.3: Key expansion utilizing net KE as a standard key input.

of the transmission gate logic locking family to force an adversary to consider a larger key search space are provided in this section.

6.2.1 Key Expansion Topologies

Key expansion is successfully included into a circuit when the expanded key appears to be a standard case of logic locking. As a result, one secured gate must not be directly connected to the key input of another secured gate. The first topology that locks the input keys is shown in Fig. 6.3, where the net labeled KE represents the key expanded net. For the topology shown in Fig. 6.3, A and B are inputted from nearby nets, providing the appearance that the locked gate is a NAND or AND. However, when KEY0 is equal to KEY1, a constant value is generated on KE, essentially replicating the XOR based logic locking technique. The topology shown in Fig. 6.3 also functions when the XOR gate is replaced by another standard gate such as an AND or OR. For the case that the XOR is replaced with an OR, if a controlling value is

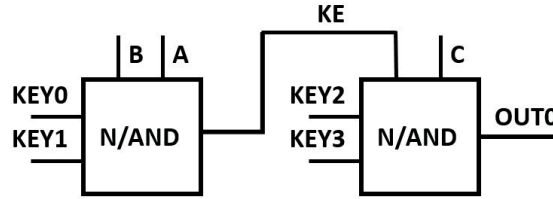


Figure 6.4: Key expansion topology with two NAND/AND gate level logic locking gates.

generated on KE, then OUT0 is a corrupted version of the output of the OR gate.

Another topology that allows for key expansion is shown in Fig. 6.4. The output of a gate level logic locking instance is connected to one of the logical inputs of another locked instance. The topology appears to be a standard logical connection within the circuit, however, key expansion is implemented by holding the KE net constant. When KE is held at a constant high, OUT0 is either C , the inversion of C , a constant 0, or a constant 1 depending on the values of KEY2 and KEY3. The AND/NAND structure no longer functions as an AND/NAND gate, forcing an adversary to explore more key input combinations.

The topology shown in Fig. 6.5 is an expanded use case of the topology in Fig. 6.4. Instead of directly connecting the output of one locked gate to another, a controllable path within the circuit is used to disguise the use of key expansion. The modified circuit makes it more difficult for an adversary to associate the given topology with key expansion.

While the presented key expansion topologies present opportunities to utilize the

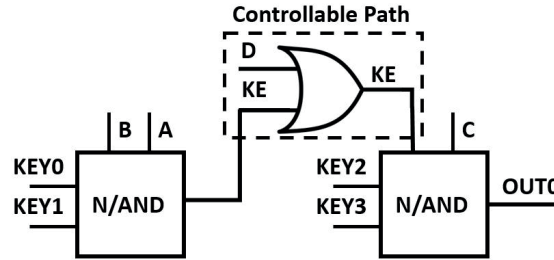


Figure 6.5: Controllable path insertion to mask key expansion.

constant output cases of the transmission gate logic locking, the overheads of the topologies are larger than a standard locked gate. As such, the ideal circuit candidate is one that already includes the budget for the implementation of the XOR or LUT based logic locked techniques. For example, the c432 ISCAS benchmark implemented with the fault-based XOR logic locking technique has a performance overhead of approximately 10%, a power overhead of 55%, and an area overhead of 57% [89]. Although the impact on performance requires further investigation, the use of gate level logic locking reduces the power overhead to an estimated 32.3% and the area overhead to 45.7% for the same implementation. The 22.7% difference in the power overhead and 11.3% difference in the area overhead are available for key expansion. The reduced overhead of gate level logic locking permits the addition of key expansion to further increase the key combinations an adversary must search while maintaining a similar overhead budget to current methodologies.

Table 6.6: Propagation delay, power, and area analysis of the AND/OR implemented with stack-based logic locking. Percent improvements over XOR and LUT logic locking are listed.

Comp.	Logic Cell	Prop. Delay (ps)	Power (nW)	Area (μm^2)
XOR	AND	119.8 (20.82%)	80.72 (46.33%)	41.62 (34.77%)
	OR	116.7 (25.90%)	79.87 (54.26%)	
LUT	AND	119.8 (2.204%)	80.72 (44.71%)	41.62 (54.05%)
	OR	116.7 (3.394%)	79.87 (43.83%)	

Table 6.7: Propagation delay, power, and area analysis of the NAND/AND implemented with transmission gate logic locking. Percent improvements over XOR and LUT logic locking are listed.

Comp.	Logic Cell	Prop. Delay (ps)	Power (nW)	Area (μm^2)
XOR	AND	80.19 (46.99%)	85.41 (43.21%)	51.10 (19.81%)
	NAND	98.54 (22.77%)	101.0 (28.97%)	
LUT	AND	80.19 (34.54%)	85.41 (41.50%)	51.10 (43.58%)
	NAND	98.54 (20.91%)	101.0 (35.67%)	

6.3 Evaluation of Power, Area, and Performance

The improvements in the power, propagation delay, and area for both the stack and transmission gate topologies are described in this section. The drive strengths of the locked cells were matched with those of the XOR and LUT based logic locking techniques. In order to match the drive strength, an inverter was added to the output of both the stack and transmission gate topologies. The functionality of the stack-based design is therefore an AND/OR gate as opposed to a NAND/NOR. The improvement in performance is described as a reduction in propagation delay. An analysis of the average power and average leakage power is provided for each gate.

Table 6.8: Propagation delay, power, and area analysis of the NOR/OR implemented with transmission gate logic locking. Percent improvements over XOR and LUT logic locking are listed.

Comp.	Logic Cell	Prop. Delay (ps)	Power (nW)	Area (μm^2)
XOR	OR	86.99 (44.77%)	91.44 (47.63%)	51.10 (19.81%)
	NOR	96.77 (27.94%)	88.00 (47.78%)	
LUT	OR	86.99 (27.99%)	91.44 (35.70%)	51.10 (43.58%)
	NOR	96.77 (23.68%)	88.00 (44.58%)	

Table 6.9: Analysis of the average leakage power of the stack-based AND/OR and transmission gate based AND/NAND and OR/NOR topologies. Percent improvement over the XOR based topology are listed.

Logic Cell	Stack (pW)	Trans. Gate (pW)
AND	226.8 (66.34%)	405.1 (39.88%)
NAND	-	371.6 (36.04%)
OR	181.2 (68.81%)	528.2 (20.15%)
NOR	-	494.6 (23.78%)

The area improvement is based on an area estimate determined from a layout of the locked gates as compared with layouts of the XOR and LUT locked methods. The improvements over the XOR and LUT based logic locking techniques are listed in Tables 6.6, 6.7, and 6.8 for, respectively, an AND/OR, NAND/AND, and NOR/OR. The average leakage power of each of the topologies and the percentage improvement over the XOR based technique is listed in Table 6.9. A comparison with the LUT based technique results in larger improvements, as the LUT leakage power provided in Table 6.5 is greater than the XOR based logic locking.

The results demonstrate a substantial reduction in the performance, power, and

area overheads associated with locking a gate. The reductions allow for an expanded key search space to further deter adversaries without increasing the overheads of current logic locking methods. In addition, the lower per-gate logic locking overheads allow for the placement of locked gates closer to the critical timing path within a circuit.

Chapter 7: Characterization of In-Cone Logic Locking Resiliency Against the SAT Attack

When the IC design and manufacturing threat model includes susceptibility to oracle guided attacks, as discussed in Chapter 3, gate selection of in-cone logic locking techniques is of greater importance. The modifications to logic locking gate selection algorithms that allow for increased resilience against the SAT attack and whether the increased resilience provides sufficient protection against oracle guided attacks are described in this chapter.

In order to provide a more robust analysis of security, the chapter begins with an analysis of SAT solver parameters that effect the measurement of SAT attack security, and an analysis of the impact logic structure has on the efficacy of the SAT attack is provided. The analysis of in-cone logic locking is utilized to develop a gate selection strategy based on maximum fanout free cones (MFFCs) and gate controllability for the insertion of XOR gates, heterogeneously sized LUTs, and 2x1 MUXes. Guiding principles to limit the generation of superfluous keys are also provided. The developed in-cone MFFC based selection algorithms are compared to the out-of-cone stripped functionality logic locking (SFLL) technique [101], discussed in Chapter 4, with regard to SAT security and corruption of circuit outputs.

7.1 Avoiding SAT Solver Specific Security

Each developed SAT solver includes differences that effect the solution time and number of iterations required to solve a given problem. This section analyzes variable ordering, initial conditions, and phase heuristics to characterize the variability in the execution of the SAT attack. The presented analysis is not a methodology to increase the security of in-cone logic locking against the SAT attack, but rather to prevent security solutions that target a specific implementation of a SAT solver.

The homogeneous tree structures shown in Fig. 7.1 are the fundamental logical structures used in the characterization of in-cone logic locking techniques as the AND-tree was shown to provide increased resiliency against the SAT attack [96]. An analysis of variable ordering, and, therefore, the clause order provided to the SAT solver is provided in Section 7.1.1. The effect of the applied initial DIPs on the number of iterations required to execute the SAT attack is described in Section 7.1.2. The three phase heuristics of 1) all true, 2) all false, and 3) the advanced heuristic included with Lingeling are analyzed in Section 7.1.3 to characterize the effect that the initial phase of the solver has on the SAT attack.

7.1.1 Effect of Variable Ordering on the SAT Attack

The C++ implementation of the SAT attack described in [96] topologically sorts the circuit, which results in a variable ordering of nodes at an equal logical level

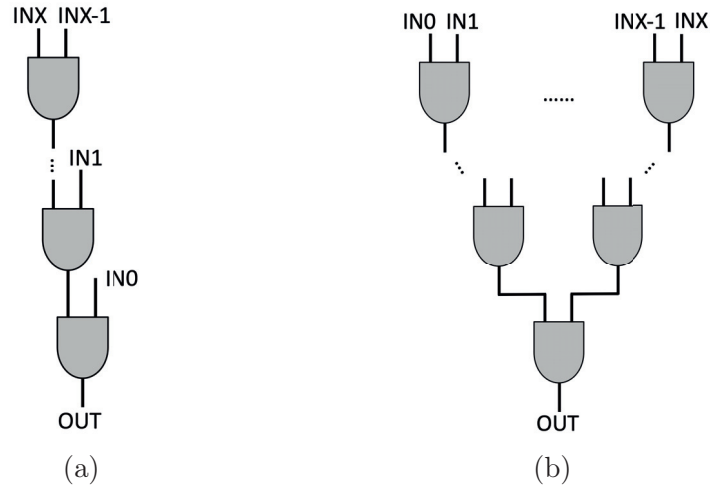


Figure 7.1: AND-tree topologies with a) an input at every level, and b) inputs at the level farthest from the output.

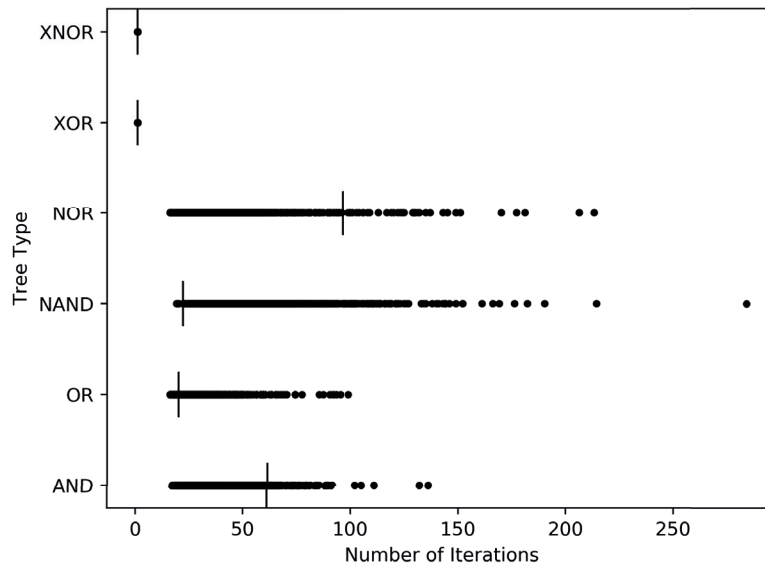


Figure 7.2: Simulation of the gate tree structure from Fig. 7.1a with 1,000 random netlist orders quantifying the number of iterations required for the SAT attack to decrypt the tree. All nets within the tree are encrypted for the simulation. The vertical line represents the netlist order used in [96].

dependent on the structure of the netlist. Randomly re-writing the original netlist, therefore, generates multiple variable orderings. Note that the topology of the circuit is never altered, simply the structural order of the netlist is rearranged. The structure in Fig. 7.1a with 16 inputs is implemented to analyze netlist reordering, where the tree of AND gates is linearly connected. All nodes except for the output are logic locked with an XOR, requiring a key gate at every input and internal net.

From the initial netlist, 1,000 random variable orderings of the linear AND-tree structure shown in Fig. 7.1a are generated and provided as inputs to the SAT solver. The results of executing the SAT attack on the reordered netlist are provided in Fig. 7.2, with the unaltered netlist order shown as a vertical line. OR, NOR, NAND, XOR, and XNOR trees are also analyzed, providing a quantified distribution of the number of iterations of the SAT attack required to decrypt the keys as a function of the variable ordering of the netlist.

For the linear tree structure shown in Fig. 7.1a, the results shown in Fig. 7.2 indicate NAND and NOR trees require more iterations on average to complete the SAT attack than the AND and OR tree structures. The increase in the average number of iterations is due to 1) the distribution of output values being biased towards either logic 0 or logic 1 when applying the secured NAND and NOR tree structures and 2) the SAT attack applying initial DIPs of all logic 0 and all logic 1. The initial DIPs of all logic 0 and all logic 1 do not significantly constrain the key space for the

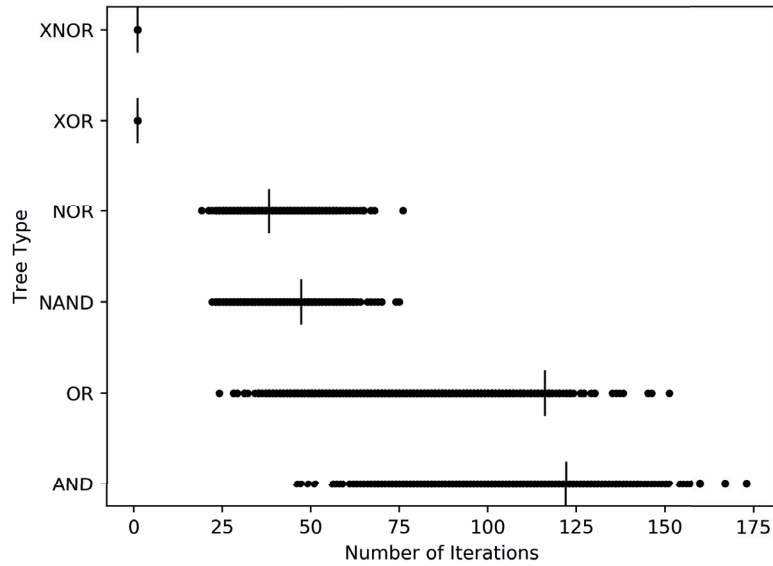


Figure 7.3: Simulation of the gate tree structure from Fig. 7.1b with 1,000 random netlist orders quantifying the number of iterations required for the SAT attack to decrypt the tree. All nets within the tree are encrypted for the simulation. The vertical line represents the netlist order used in [96].

NAND and NOR tree topologies as much as the key space is constrained for the AND and OR trees. As the linear NAND/NOR based tree structure shown in Fig. 7.1a is vulnerable to a more limited number of significantly constraining DIPs, the average number of iterations required to execute the SAT attack increases. However, the minimum number of iterations across all gates remains relatively constant as the attack selects the more constraining DIPs in earlier iterations.

The pyramid tree structure in Fig. 7.1b is also analyzed for the same six standard gate types with 1,000 random netlist orderings of each, with results provided

in Fig. 7.3. The AND and OR trees yield much stronger security on average than the NAND and NOR tree topologies, exactly opposite from the results provided in Fig. 7.2 for the tree topology of Fig. 7.1a. The small AND-tree topology shown in Fig. 7.4 illustrates the cause of the increase in the number of iterations when executing the SAT attack. An output of 1 from the AND-tree forces all internal nodes of the tree to logic 1 until an XOR key gate is reached, at which point the SAT solver has an increased degree of freedom. The solver is now allowed to select either a logic 0 or 1 for each input of the XOR gate. The increase in the degrees of freedom result in a decrease in the likelihood of the SAT solver generating an input of all ones as a DIP. Any DIP that is not all ones results in a variety of key combinations that generate a zero at the output of the IC as the AND tree is significantly skewed towards a zero output. Similarly, a DIP of all logic zeros is less likely to be generated for an OR-tree, which allows for the masking of incorrect key sequences and accounts for the increase in the number of iterations provided by OR-tree structures as compared to NOR, NAND, XNOR, and XOR gates, as indicated by results shown in Fig. 7.3.

Independent of the utilized circuit topology, the variation in the distribution of the number of SAT attack iterations due to variable ordering is an important consideration. Without accounting for the variation, the level of perceived circuit security does not necessarily match the true security provided by the implemented obfuscation technique.

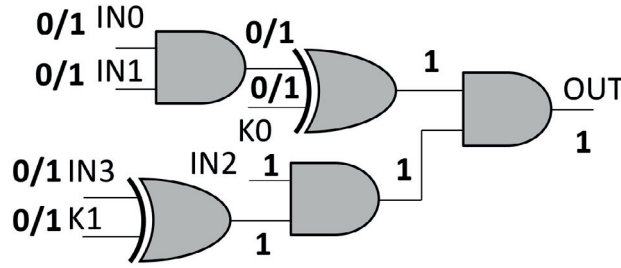


Figure 7.4: SAT solver propagating a constant logic 1 at the output of the AND-tree. After a key gate is reached, the solver has an increased degree of freedom when selecting input logic values.

7.1.2 Effect of Initial DIPs on the SAT Attack

Before generating any DIPs with the miter circuit, the SAT attack, as described in [96], applies the initial input constraints of all logic 0 and all logic 1 . An analysis of the number of iterations of the SAT attack with and without the initial constraints of all logic 0 and all logic 1 inputs is performed with results listed in Table 7.1. Each gate type is analyzed with 1,000 random variable orderings of the gate structure shown in Fig. 7.1b with 8 inputs to the tree. A significant difference is seen in the number of iterations to complete the SAT attack when comparing the results with and without the initial constraints of logic 0 and logic 1 . The results listed in Table 7.1 indicate a 443% and a 636% increase in the number of SAT attack iterations on average for, respectively, the AND and OR trees when the initial constraints are removed. Similar to variable ordering, the variation in the number of SAT attack iterations due to initial constraints limits the assessment of the security of an IC when executing the SAT attack only once.

Table 7.1: Minimum and average number of iterations to complete the SAT attack with and without all 0/1 initial constraints for the tree structure of Fig. 7.1b.

Gate Type	Number of SAT Attack Iterations			
	0/1 Constraints		No Constraints	
	Min.	Average	Min.	Average
AND	12	24.26	15	131.7
OR	9	17.63	10	129.8
NAND	9	16.77	9	22.59
NOR	8	14.02	8	15.63
XOR	1	1.00	2	2.00
XNOR	1	1.00	2	2.00

7.1.3 Effect of Phase Heuristic on the SAT Attack

SAT solvers employ varying phase heuristics to select between a true and false assignment of a given variable. To analyze the effect of the phase heuristic on the efficacy of a SAT attack, initial default phases of 1) all true, 2) all false, and 3) an advanced phase heuristic included with the Lingeling SAT solver [127] are applied to the circuit topology shown in Fig. 7.1b, where the pyramid structure includes 16 inputs for each gate type. Applying a default phase of true to all inputs results in the OR-tree topology requiring more iterations on average than an AND-tree structure, as indicated by the results provided in Fig. 7.5. The increase in the number of iterations of the SAT attack for an OR-tree with a default phase of true is explained in Section 7.1.1 through the example structure shown in Fig. 7.4. Defaulting to a phase of true, or logic 1, results in a lower probability of the SAT solver selecting a DIP of all zeros, which masks incorrect key information throughout the pyramid

structure. As information regarding the circuit is masked, the number of iterations increases on average as compared to an AND-tree structure.

As the AND-tree structure of the ISCAS'85 c2670 benchmark circuit was presented as the primary factor that resulted in an increase in the security of the circuit in [96], other researchers applied an AND-tree topology to increase the security of novel obfuscation techniques [128]. However, the analysis provided in this paper demonstrates that utilizing a single tree topology leads to vulnerabilities as an adversary is able to simply adjust the phase heuristic to more efficiently attack the implemented obfuscation technique. The analysis provided in Section 7.1 indicates that the use of a specific circuit function to increase security provides limited gains as an adversary is able to 1) transform the circuit to shift controllable inputs towards the outputs of the IC, 2) add specific DIPs to the SAT formulation, and/or 3) alter the phase heuristic of the solver to reduce the number of iterations required to determine the key.

7.2 Effect of Logic Structure on SAT Attack Resiliency

The effects of gate heterogeneity, logical reconvergence, and number of keys per node are analyzed as each impacts the efficacy of the SAT attack. The benefits of using an AND-tree structure to secure combinational logic are described in [96] when the relatively smaller ISCAS'85 c2670 benchmark circuit was one of the more challenging to decrypt. Finding similar logical structures in a netlist that exhibit increased

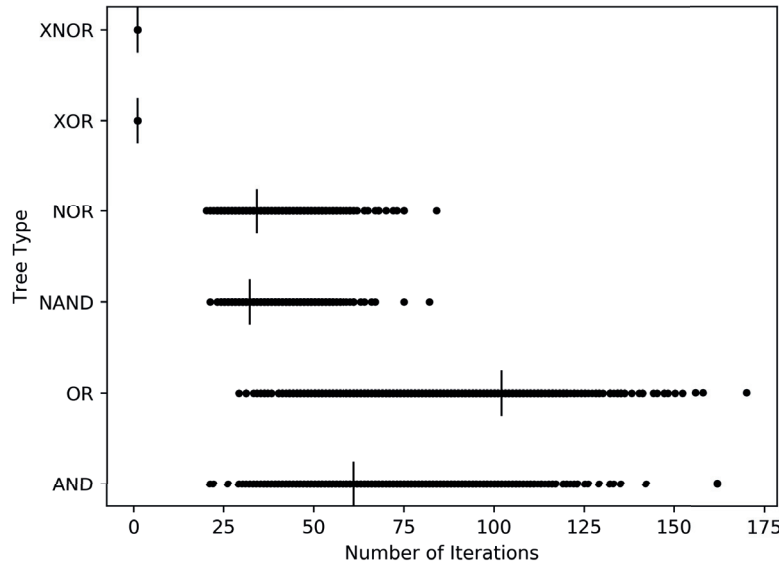


Figure 7.5: Simulation of the gate tree structure of Fig. 7.1b with 1,000 random netlist orders. The number of iterations required for the SAT attack to decrypt the tree with a default SAT solver phase of true is quantified, with the vertical line representing the netlist order used in [96]. All nets within the tree are encrypted for the simulation.

resiliency to SAT attacks is, therefore, an important step to secure in-cone logic locking. The study of gate heterogeneity, where the logical structure of the circuit limits the exposure of key sequences during a SAT attack, is described in Section 7.2.1. Logical reconvergence is examined in Section 7.2.2 to further characterize the effect of the circuit structure on the ability to secure the logic cone. Lastly, the effect on the security provided against the SAT attack due to the number of keys per node is analyzed in Section 7.2.3.

7.2.1 Analysis of Gate Heterogeneity

The topology shown in Fig. 7.1b is utilized to characterize the effect gate heterogeneity has on the resiliency of the circuit against the SAT attack. The topology includes 16 inputs with a varying combination of distributions of logic gates. The gate types varied from a homogeneous structure of AND, OR, NOR, and NAND gates to a distribution where the gate type under evaluation occupied 10% of all the gates in the netlist and the remaining three gate types occupied 30% each. In order to limit the effect due to random placement of key gates, fixed locations representing 25% of the inputs of the netlist are locked. Each circuit topology is evaluated for 100 random gate types selected for each gate distribution. The 100 randomly generated circuits are analyzed for 1,000 random variable orderings to quantify the level of security provided by the logical structure of the circuit against the SAT attack. In order to avoid the formation of homogeneous paths, connected gates are not permitted to be of the same type unless all other gate types had already met or exceeded the target selection percentage for the given netlist. The average number of iterations required to execute the SAT attack for the 100 randomly generated netlists across the 1,000 variable orderings is shown in Fig. 7.6.

Both the homogeneous AND and OR trees provide a higher min and average number of iterations than the other gate types and gate distributions as shown in Fig. 7.6. Using more heterogeneous gate structures decreases the number of iterations

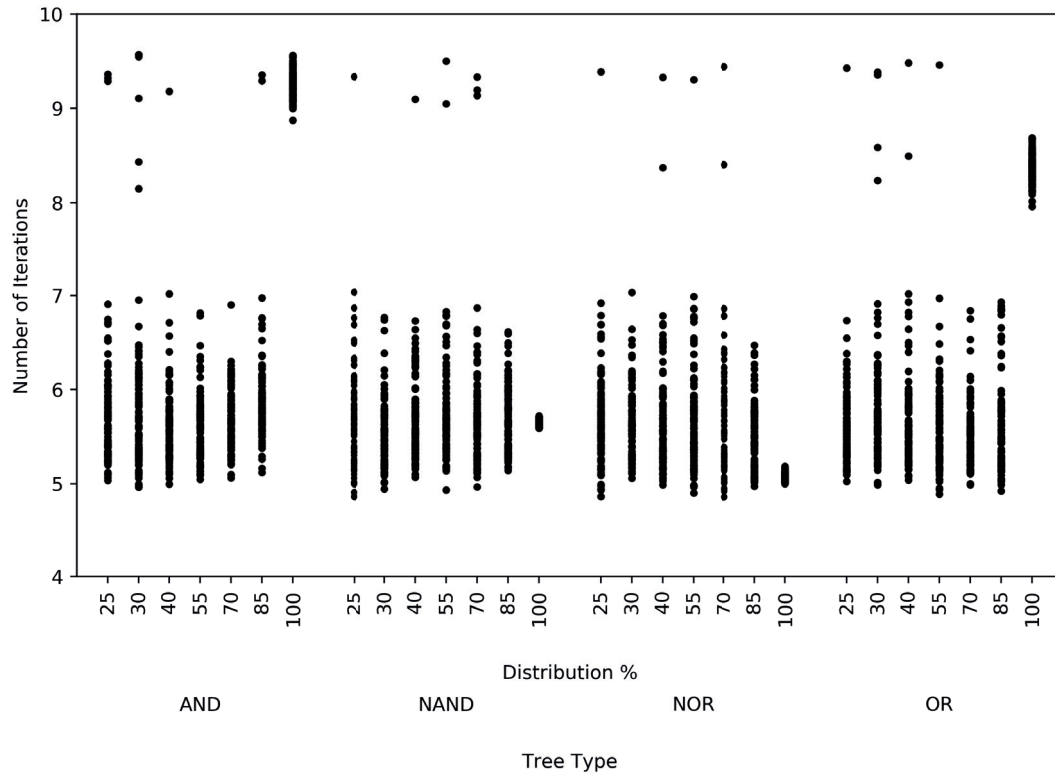


Figure 7.6: Analysis of each gate type starting as a homogeneous tree (probability of dominant gate = 1) and then decreasing the probability of selecting a dominant gate to 0.1 (0.25 represents an AND, OR, NAND, and NOR selection probability of 25%). XOR based logic locking is applied to 25% of the circuit inputs.

required to complete the SAT attack on average. However, there are cases where the number of iterations increases significantly above the average. Analyzing the structure of the benchmark circuits that require a larger number of iterations indicates that the circuit topologies limit the degree of freedom of the SAT solver, which results in an increase in the number of iterations. Therefore, the circuits that are most secure include a high disparity in the probability of the output being either logic 0

or logic 1. The increased degrees of freedom for the SAT solver to generate DIPs then favors an input pattern that produces the probabilistically favored output (i.e. 0 for an AND gate and 1 for an OR gate) even in cases where an incorrect key is applied, decreasing the efficiency of the SAT attack. For example, the benchmark circuit requiring the highest average number of iterations has an output probability of 95.67% when producing a logic 0 and 4.33% when generating a logic 1 with a topology consisting of 55% OR gates and 15% AND, NAND, and NOR gates. The benchmark circuit with the lowest average number of iterations for a topology with the same percentage of gate types has an output probability of 59.1% when producing a logic 0 and 40.9% when generating a logic 1. The analysis demonstrates that homogeneous tree structures are not the only circuit topologies providing increased protection against the SAT attack. However, the likelihood of randomly generating a heterogeneous circuit topology that sufficiently skews the output probability is low. The controllability of the gates, or the disparity between the probability of a logic 0 and 1, therefore, serves as an important quantifiable parameter to determine the optimal locations to secure in the netlist.

7.2.2 Analysis of Logical Reconvergence

Logical reconvergence, where the fanout from a gate subsequently converges at a downstream gate of the circuit, is evaluated to characterize the effect on the efficacy

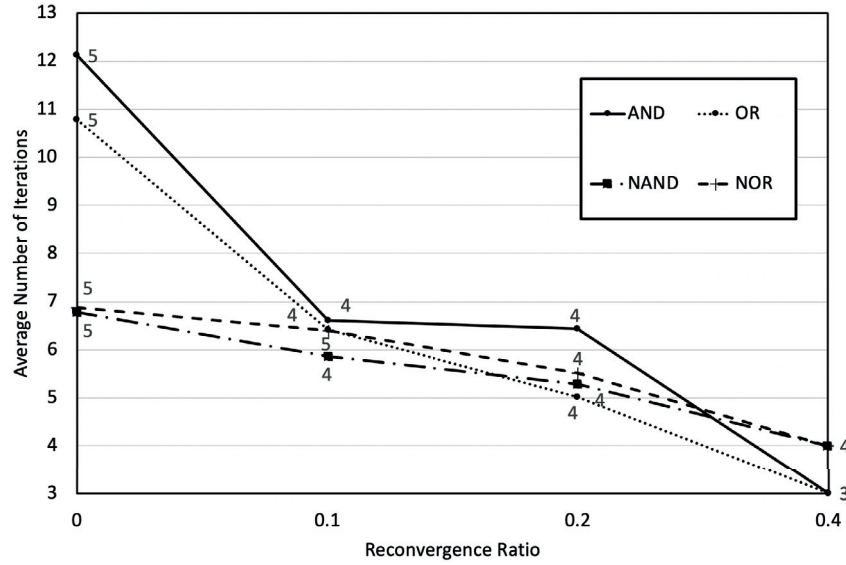


Figure 7.7: Analysis of the effect of reconvergence on the number of SAT iterations required to decrypt the key for a tree structure consisting of gates of the same type. The largest minimum number of iterations across the 1,000 configurations of the benchmark circuits for each gate type and reconvergence ratio is listed above each data point.

of the SAT attack. Reconvergence of logic is controlled probabilistically by limiting the number of available nodes that connect to the next level of logic. As n number of 2-input gates require $2 * n$ inputs at a given logic level l , limiting the number of gates providing inputs to logic level l to less than $2 * n$ requires logical reconvergence within the netlist.

The reconvergence ratio is used to represent the reduction in the number of inputs applied to logic level l . The maximum number of inputs is set to $2 * n$, which indicates a reconvergence ratio of 0. As an example, a reconvergence ratio of 0.4 implies $2 * n * (1 - 0.4)$ inputs are applied to logic level l . The results characterizing the

effect on the average number of iterations of the SAT solver due to probabilistically controlling the logical reconvergence are shown in Fig. 7.7, where the reconvergence ratio is swept from 0.0 to 0.4. The analysis consists of generating 1,000 random implementations of logic locked circuits, each while assuming homogeneous gate types, a maximum number of inputs of 16, and a 25% overhead in area. Each of the 1,000 generated implementations is then subjected to 100 random variable orderings to further characterize the effect on the efficacy of the SAT attack, as discussed in Section 7.1.1.

The results shown in Fig. 7.7 indicate that both the average number and the largest minimum number of iterations (shown above each data point) decrease for all gate types as the reconvergence ratio increases. The reduction in the number of iterations to complete the SAT attack implies an increased challenge in masking key information and, therefore, hiding the structure of the circuit as the reconvergence ratio increases. As the level of logical fanout increases, incorrect key information spreads across a larger segment of the circuit, resulting in a lower probability of masking the applied key bit(s) at the output(s) of the IC.

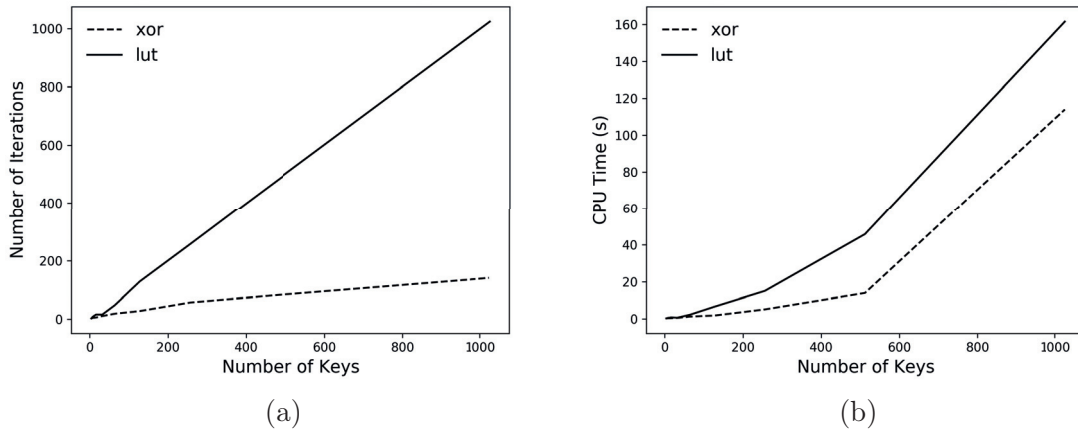


Figure 7.8: Characterization of the effect the number of keys applied at a given netlist location has on a) the number of iterations and b) the run time required to complete the SAT attack.

7.2.3 Analysis of the Number of Keys per Node

The analysis of gate heterogeneity and logical reconvergence demonstrates that the structure of a circuit has an impact on the resiliency against a SAT attack. In-cone logic locking techniques, for the same overhead in area, either secure more nodes of the circuit or implement additional key bits at a single node. The analysis is completed by replacing a gate(s) within the ISCAS'85 c5315 circuit netlist with a LUT, where the number of keys determines the size of the LUT and the number of required LUT select lines is $\log_2(\text{number of keys})$. Once the number of select lines to the LUT increases beyond the total number of inputs to a functional block, false inputs are added to the LUT. The false inputs are generated from nearby nets within the circuit and have no effect on the output of the LUT.

LUT insertion at a single node is compared with random insertion of XOR gates at multiple nodes within a ISCAS'85 c5315 benchmark circuit, with results shown in Fig. 7.8. The number of keys is linearly correlated with the number of iterations required to complete the SAT attack when replacing a gate, as shown in Fig. 7.8a. The number of iterations increases at a much faster rate for LUT based logic locking, as compared to random XOR insertion. The SAT attack has a linear correlation with the number of keys at a given net as a single DIP only exposes a single key bit of the LUT. In addition to the linear increase in the number of iterations, the run time of the SAT attack increases exponentially due to the exponential increase in the number of variables provided to the SAT solver. The random XOR insertion matches more closely to LUT based insertion with regard to CPU run time as the SAT attack spends more time searching for a satisfying assignment within the formulated miter circuit. The trade-off between securing more nodes and increasing the number of keys per node also depends on the allocated overhead in performance, power, and area. Inserting a LUT requires the use of k 2-input MUXes, where k is the total number of key bits, which results in a large power, performance, and area overhead within a small section of the netlist.

Table 7.2: Analysis of the number of correct keys for four gate selection methodologies and benchmark circuits. The area allocated for logic locking is provided in parenthesis.

Benchmark	[91]	[89] MUX	[89] XOR	[88]
c432 (10%)	2	1	1	1
c499 (10%)	4 (5%)	2	1	32
c880 (5%)	2	2	2	2
i4 (5%)	128	-	-	1

7.3 Managing the Number of Correct Keys

In addition to effecting the ability to secure against the SAT attack, the logical structure of the circuit also generates conditions that lead to undesired working keys for in-cone logic locking. To demonstrate, a set of four benchmark circuits are secured and the key space of each is exhaustively searched to determine all functional keys. The number of correct keys generated for four different gate selection algorithms is listed in Table 7.2. For the i4 benchmark circuit, 128 functional keys are present when the logic locking algorithm described in [91] is implemented with a permitted overhead in area of 5%.

The work in [91] highlights the importance of avoiding the selection of consecutive gates for logic locking as the selected gates effectively function as a single security gate. Three new rules are, therefore, proposed to control the generation of functional keys for in-cone logic locking, and are described as 1) buffer/inverter separation, 2) insertion on XOR/XNOR gates, and 3) reconvergent fanout checks.

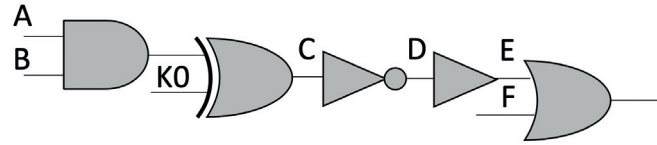


Figure 7.9: Example of a buffer/inverter chain topology that must be accounted for when selecting nets to secure. Selection of nets C, D, or E results in an increase in the number of functional keys.

7.3.0.a Buffer/inverter separation

The first rule accounts for the presence of buffers or inverters within the netlist. The circuit topology of concern is shown in Fig. 7.9, where a key gate is already placed within the circuit and the selection algorithm is determining which net to secure next. If the algorithm selects nets C, D, or E in Fig. 7.9, the number of generated functional keys increases as the resulting topology is essentially the same as a series of key gates.

To prevent the selection and insertion of secure gates on nets of a buffer/inverter chain, the netlist is traversed from the input(s) and the output(s) of a placed key gate with buffers and inverters marked until a non-buffer or non-inverter is reached. Referring to Fig. 7.9, the nets C, D, and E are marked with a penalty term, alerting the selection algorithm that inserting a key gate at these locations creates multiple functional keys.

7.3.0.b Insertion on selected XOR/XNOR gates

The concern with inserting key gates before or after an XOR/XNOR gate is the symmetric nature of the function. Securing both inputs of a two-input XOR/XNOR results in a functional key when the two key bits are correct and when the correct key is inverted. Netlists containing higher percentages of XOR/XNOR gates are, therefore, susceptible to many working keys, which explains the results shown in Figs. 7.2 and 7.3, where both the XOR and XNOR gates require the least number of iterations to decrypt the circuit with a SAT attack. To limit the selection of logic locking gates to a single net connected to a single input or the output of the XOR/XNOR gate, a penalty term is applied to the remaining inputs and the output of the XOR/XNOR gate after the insertion of a key gate. For example, if input A of a two input XOR gate is secured, both input B and the output of that XOR gate are penalized to avoid the generation of undesired working keys.

7.3.0.c Reconvergent fanout check

Another important consideration is the reconvergence of fanouts within the netlist as inputs to a gate generated from a common source potentially reduce the total number of input combinations possible at the given gate. An example of such a circuit is shown in Fig. 7.10, where the inputs to the AND gate never equal one another (i.e. 00 or 11). The reduction in the possible input combinations of the gate

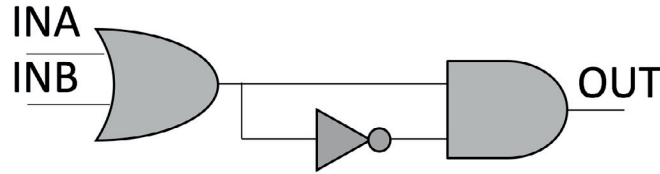


Figure 7.10: AND gate topology with dependent inputs limiting the possible functional outputs, which results in multiple correct key values when both inputs are secured.

results in the generation of undesired correct keys when both inputs are secured.

The simplest solution is to only allow the locking of a single input of each gate. However, such a solution results in a large reduction in the possible nets to secure within the circuit. Tracking and simulating the location and the effect of each reconvergent fanout is computationally expensive. Therefore, a trade-off between increasing computational complexity and not limiting the nets available when selecting gates for logic locking is required to efficiently determine the potential for an undesired increase in the number of functional keys. After a key gate is inserted, the netlist is traversed from the added secure gate to the inputs of the circuit, with each visited node marked. Each of the remaining unsecured inputs to the target gate are then traversed to the inputs of the circuit. If any of the marked nodes are encountered during the traversal, a penalty is applied to those specific inputs of the gate.

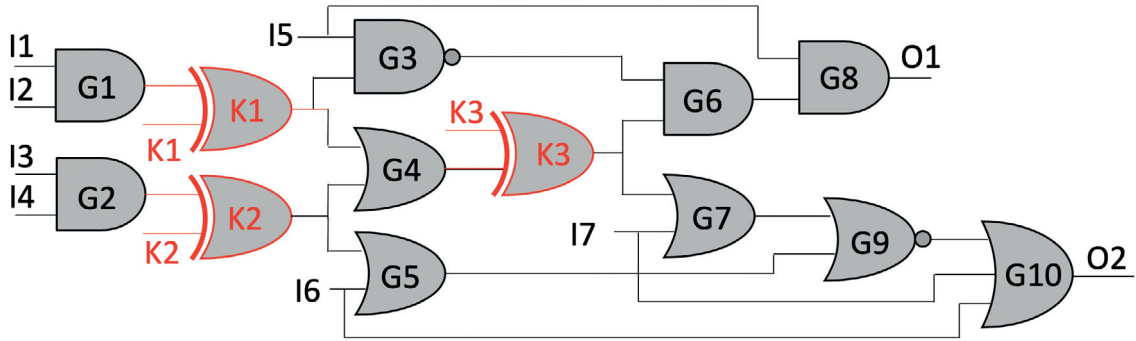


Figure 7.11: Example circuit that includes pairwise secure key gates K1, K2, and K3 [91]. The pairwise security of the key bits prevents the key sensitization attack proposed in [129].

7.4 Overview of MFFC Gate Selection

The circuit shown in Fig. 7.11 includes three pairwise secure gates to prevent the key sensitization attack [91, 129]. While the key gates cannot be muted to sensitize the key bits to individual outputs, consider the case where $I1 = 0$, $I2 = 1$, and $I5 = 1$. The keyspace is now significantly constrained as setting $K1 = 1$ results in an incorrect output for the majority of the applied keys. The heavily constrained DIP sequences for the circuit shown in Fig. 7.11 motivate the need for a novel gate selection strategy. The fanout of incorrect key information is a primary factor in the efficient determination of the key, as shown in Section 7.2.2.

Increasing the resiliency of a circuit that implements in-cone logic locking in part relies on controlling the amount and location of leaked information that the SAT attack uses to generate DIPs. To ensure that the fanout of logical nets does not create

unintended leakage channels for the SAT attack to utilize, the concept of maximum fanout free cones (MFFCs) is developed. An MFFC is a cone of logic where a net within the MFFC converges to a given output net [130]. As an example, consider the ISCAS'85 benchmark c17 circuit shown in Fig. 7.12, where the MFFC of net N23 includes all gates contained within the dashed box. Since the fanout of G4 includes a logical path that does not converge to N23, it is not part of the MFFC. In addition, edges that converge at net N23 are potential locations to utilize for XOR/XNOR locking as all leaked information must propagate through N23. For the circuit shown in Fig. 7.12, the edges $N16 \rightarrow G6$ and $N11 \rightarrow G5$ are, therefore, added to the list of eligible locations for the insertion of a locking gate. Returning to the example shown in Fig. 7.11, if K1 is inserted only on the edge $G1 \rightarrow G4$ and K2 is inserted only on the edge $G2 \rightarrow G4$, the three gates are now pairwise secure and within a MFFC. The previous scenario, where $I1 = 0$, $I2 = 1$, and $I5 = 1$, is no longer dependent on any key values, forcing the SAT attack to generate DIPs that are reliant on all three key bits. Generating constraints that depend on a greater number of key bits increases the circuit resiliency against the SAT attack as more DIPs are needed to determine the key sequence.

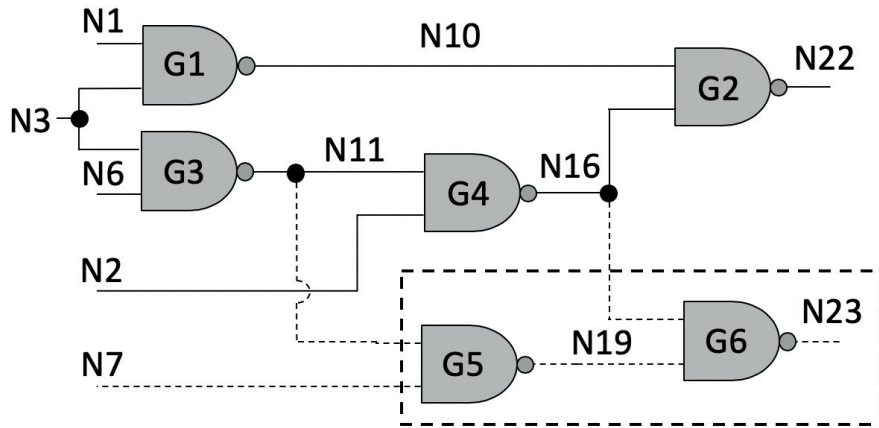


Figure 7.12: An example of an MFFC for net N23. Gates contained within the MFFC converge to node N23. Dashed edges represent graph edges where potential XOR/XNOR locking is possible while still being contained within the MFFC.

7.4.1 Determination of MFFCs

To determine the MFFC, a single node is selected and Algorithm 9 is executed for the list of directly connected input nodes. When all outputs from a given node are connected to nodes within the MFFC, the given node is added to the MFFC. The algorithm is then applied recursively to input nodes that fit the MFFC criteria, termed *next_lvl_gates* in Algorithm 9. Edges that fanout to a node contained within the MFFC are added to the MFFC for the current node evaluated by Algorithm 9 (G6 in the example circuit shown in Fig. 7.12), but the starting node of the edge is not added to *next_lvl_gates* unless all of the fanout paths from the node are also contained within the MFFC. Nodes and/or edges are traversed and added to the MFFC in topological order. Although not guaranteed pairwise secure, the generated

Algorithm 9: Determination of MFFC

```

Input: Logic level gates level_gates,
MFFC of Node N list,  $M_N$ 
next_lvl_gates = [ ];
for gate in level_gates do
    fanout_free = True;
    tmp_edges = [ ];
    foreach gate_output in gate_outputs do
        if gate_output in mffc_nodes then
            /* Add  $gate \rightarrow gate\_output$  to tmp_edges */
        else
            fanout_free = False;
        end
    if fanout_free then
        /* Add node to MFFC */
        /* Add gate_inputs of gate to next_lvl_gates */
    end
if next_lvl_gates not empty then
    /* Recursively call function with next_lvl_nodes */

```

order provides inherent pairwise security. The pairwise security due to the generated MFFC protects against a sensitization attack as a brute force attack is required with complexity of approximately 2^k , where k is the total number of key bits. Therefore, the SAT attack remains the dominant attack vector. If, however, non-mutable edges are needed, then the algorithm in [91] can be applied to the MFFC nodes.

7.4.2 Weighting of the MFFCs

Once the MFFC of each node is calculated, the MFFCs are sorted by a weighted sum of controllability to determine the MFFC best suited for the insertion of logic locking gates. The measure of gate controllability is used due to the results described

in Section 7.2.1, which demonstrated that skewed gate probabilities result in an increase in the resiliency of the circuit against the SAT attack. The controllability is heuristically determined by estimating the probability of a logic 1 and logic 0 at each node and then taking the absolute difference between the calculated probabilities. The controllability values for the nodes in an MFFC are averaged together and multiplied by the total number of nodes in the MFFC, or multiplied by the target number of key gates to insert if the total number of gates fit within the entire MFFC.

Algorithm 10: Insertion of Logic Locking Gates

Input: Number of Gates to Insert *num_gates*,
Node MFFCs *mffcs*
while *keys_inserted* < *num_gates* **do**
 if *use_mux* **then**
 enc_nodes = *get_mux_enc_node(mffcs)*;
 num_inserted = *insert_mux(enc_nodes)*;
 else if *remaining_keys* > 4 and *use_luts* **then**
 lut_netlist = *get_lut_enc_nodes(mffcs)*;
 num_inserted = *insert_lut(lut_netlist)*;
 else
 enc_node = *get_xor_enc_node(mffcs)*;
 num_inserted = *insert_xor(enc_node)*;
 keys_inserted += *num_keys_inserted*;
 end
 /* Determination of XOR Node */
 /* Weighted MFFC selected and reversed so nodes close to inputs
 are selected */
 for *elem* in *reversed_mffc* **do**
 /* Check elem has not been consumed and is not marked */
 if *elem* in *netlist_gates* and *not_marked* **then**
 /* Select Node for locking */
 end
 end

7.4.3 XOR-based Insertion

After the weighted MFFCs are determined, Algorithm 10 is executed to iteratively insert XOR/XNOR gates. The algorithm greedily chooses nets to insert an XOR/XNOR based on the order of the weighted MFFCs. The MFFC with the highest score is then iterated through in reverse order, as nets closest to the inputs of the circuit are added last to the MFFC. The edges and nodes within the MFFC are examined for marks, which represent a previous selection of the edge and/or a penalty applied to avoid the generation of multiple working keys. The process is repeated until the target number of key gates are inserted into the netlist.

7.4.4 Heterogeneous LUT Based Insertion

In addition to a gate insertion strategy for XOR based logic locking, a LUT based insertion strategy is described that allows for the placement of heterogeneously sized LUTs into the circuit netlist. The LUT based algorithm provides a means to increase the number of keys per node, which was shown to increase the number of iterations and execution time of the SAT attack, as described in Section 7.2.3. The maximum allowed LUT size is set by the *max_expansion* variable in Algorithm 11, which is limited to six inputs in this work.

When LUT based selection is enabled and the number of keys available is greater than four, the execution of Algorithm 10 determines the best candidate locations

Algorithm 11: Determination of LUT Candidate Gates

Input: MFFC of Node *node_mffc*, List of Node MFFCs *mffcs*
/ Access node from node_mffc */*
for *gi* in *get_gate_inputs(node)* **do**
 if *not_marked* and *gi* in *node_mffc* **then**
 / Add to candidate nodes */*
 num_expansion = get_num_expansion(netlist, lut_nodes);
 if *num_expansion < max_expansion* **then**
 get_expansion_candidates(mffcs[gi])
 end

within the netlist to insert a LUT. The process begins by examining the MFFC with the highest weight for each node in reverse order. However, in contrast to XOR-based insertion, the LUT based algorithm must examine fan-in gates as possible additions to the LUT structure.

Execution of Algorithm 11 provides a list of candidate gates to include in a functionally equivalent LUT for each gate within the highest weighted MFFC. The MFFC of a given node is also utilized in the algorithm to ensure that gates do not include fanouts that require gate replication, avoiding additional overhead when inserting the LUT. When executing Algorithm 11, the inputs to the current selected gate are analyzed. The gate is considered a candidate for inclusion in the LUT if no prior mark for selection is included or no applied penalty is present. The number of expanded nodes are then updated to ensure the number of inputs to the LUT has not exceeded six. The algorithm is recursively called for all potential candidate gates, allowing for the inclusion of additional gates into the logically equivalent LUT. As buffers/inverters

within the MFFC of a node do not expand the number of inputs, the buffers/inverters are possible candidate gates for inclusion in the logically equivalent LUT, which allows for additional obfuscation of the original netlist without requiring an increase in the size of the LUT.

The determination of candidate LUT nodes is repeated for each of the non-marked nodes within the weighted MFFC, one of which is selected as the insertion point of the LUT. Once the insertion node for the LUT is selected, the list of candidate gates generated from the execution of Algorithm 11 is utilized to include additional gates in the LUT until 1) all consumable gates are exhausted, 2) the *max_expansion* limit is reached, or 3) the limit on the number of key bits is reached. The netlist of the LUT is then provided to Algorithm 10, which splices the netlist of the LUT into the logic structure of the original netlist and removes the original gates that are obfuscated by the LUT. For this work, the LUT insertion procedure is repeated until the target number of key bits (a user defined parameter) is reached or the remaining number of key bits is less than four (the minimum number of key bits needed for a 2-input LUT representation of a gate), at which point the algorithm utilizes XOR/XNOR based logic locking to reach the required number of key bits.

Algorithm 12: Determination of MUX Inputs

```

Input: List of Nodes in MFFCs mffc_nodes
max_skew = 0; max_node = None;
/* Determine node with max skew */
for node in mffc_nodes do
    if not_marked then
        prob_skew = get_prob_skew(netlist, node);
        if prob_skew > max_skew then
            max_skew = prob_skew;
            max_node = node;
    end
/* Find node with close skew */
max_node_lvl = lvl(max_node);
min_skew = 1.0; obs_node = None;
for node in mffc_nodes do
    if not_marked and (lvl(node) < max_lvl_node) then
        skew_diff = get_skew_diff(netlist, max_node, node);
        if skew_diff < skew_min then
            skew_min = skew_diff;
            obs_node = node;
    end
/* mark max_node */
return max_node, obs_node;

```

7.4.5 2x1 MUX Based Insertion

A 2x1 MUX based interconnect logic locking algorithm is also developed, as described by Algorithm 12. A node within an MFFC is first determined that produces the maximum probability skew, as described by results provided in Section 7.2.1. The second input to the MUX is then selected from nodes that are 1) at lower levels than the *max_node* in the topological order and 2) provide a minimum difference in the probability of the node outputting a logic 0 when subtracting the probability

of logic 0 produced by *max_node*. The constraint on topological order prevents the generation of cycles within the netlist. The difference in the probability between the two nodes is minimized to reduce the ability of the SAT attack to determine DIPs that reveal the key of the MUX. Once selected, the *max_node* is marked to prevent a repeated selection of the same node. A MUX is inserted into the netlist at the target node, with the process described by Algorithm 12 repeated until the requisite number of MUXes are added.

7.5 Results of MFFC Based Gate Selection

The gate selection algorithm is applied to the ISCAS'85 benchmark circuits and to the combinational circuits from the Microelectronics Center of North Carolina (MCNC), as described in [96]. Characterization is performed based on selecting a fraction of the nodes to obfuscate, with an analysis of locking 5%, 10%, 25%, and 50% of the nodes performed. The number of implemented key bits remains the same across all the obfuscation techniques. The developed MFFC and gate controllability based XOR, LUT, and MUX based obfuscation techniques are compared with gate selection methodologies for in-cone logic locking that include *iolts14* [131], *dac12* [91], *rnd* [88], *toc13xor* [89], and *toc13mux* [89].

The analysis is performed on 1,000 random variable orderings of the netlist and for the Lingeling [127], false, and true phase heuristics. All simulations were completed

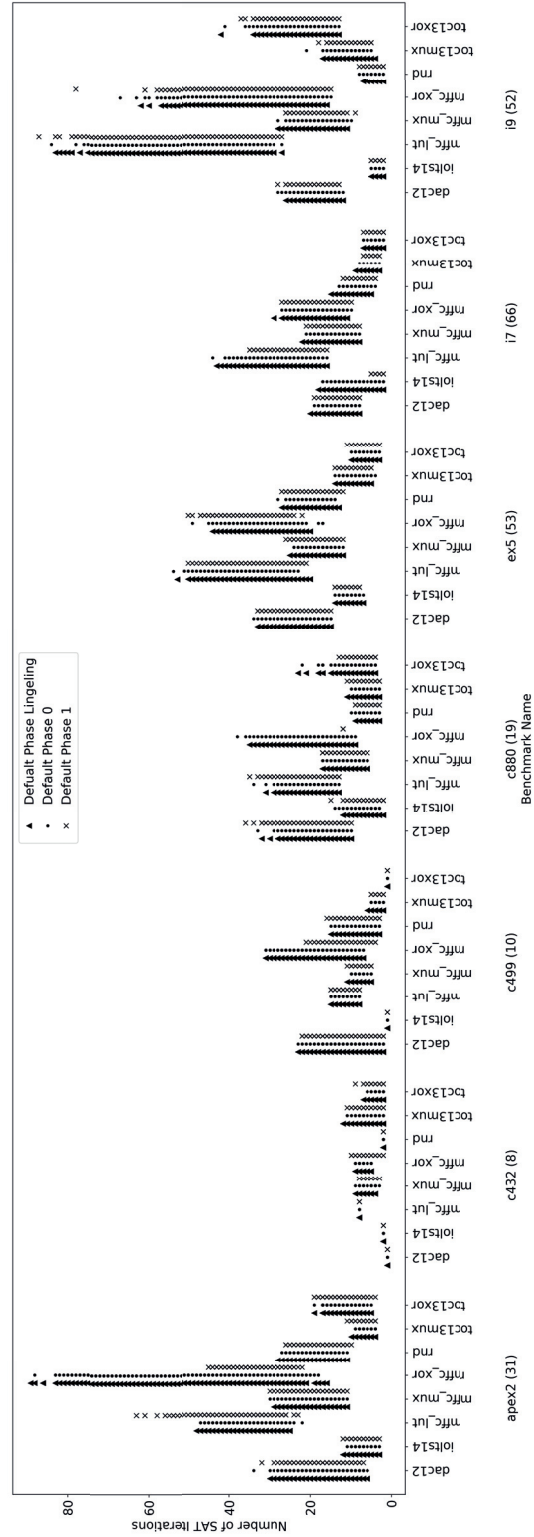


Figure 7.13: Number of required iterations of the SAT attack for in-cone logic locking gate selection methodologies when securing 5% of the gates in the netlist. Number of keys shown in parenthesis next to benchmark name. Evaluations are completed with 1,000 random variable orders and the default phase heuristic of Lingeling [127], logic 0, and logic 1.

on a Xeon E52687W CPU running at 3 GHz with 95 GB of RAM. The execution time of the SAT attack is limited to a maximum of 24 hours (86,400 seconds).

The number of SAT iterations for each gate selection methodology when securing 5% and 25% of the gates is provided in Fig. 7.13 and Fig. 7.14, respectively. The results indicate that the developed MFFC based selection algorithm significantly increases *the minimum number of iterations* required to complete the SAT attack for most of the evaluated benchmark circuits. The exceptions are the c880 benchmark circuit with 5% of the gates secured in the netlist and the i9 benchmark circuit with 25% of the gates secured, where the minimum number of iterations is at most 20.8% lower than the dac12 [91] methodology. In all cases, the LUT based selection algorithm meets or further increases *the minimum number of iterations* required to execute the SAT attack. Overall, the XOR MFFC gate selection strategy increases the minimum number of iterations to complete the SAT attack by 61.8% and increases, on average, the mean number of iterations by 80.1% across all benchmark circuits. The XOR based selection strategy results in a 196.2% increase in the average CPU run time, with a 119.6% increase in the minimum run time. The LUT MFFC based selection methodology increases the minimum number of iterations to complete the SAT attack by 123.6% and increases, on average, the mean number of iterations by 82.8% across all benchmark circuits. The LUT based selection strategy results in a 88.2% increase in the average CPU run time, with a 72.4% increase in the minimum

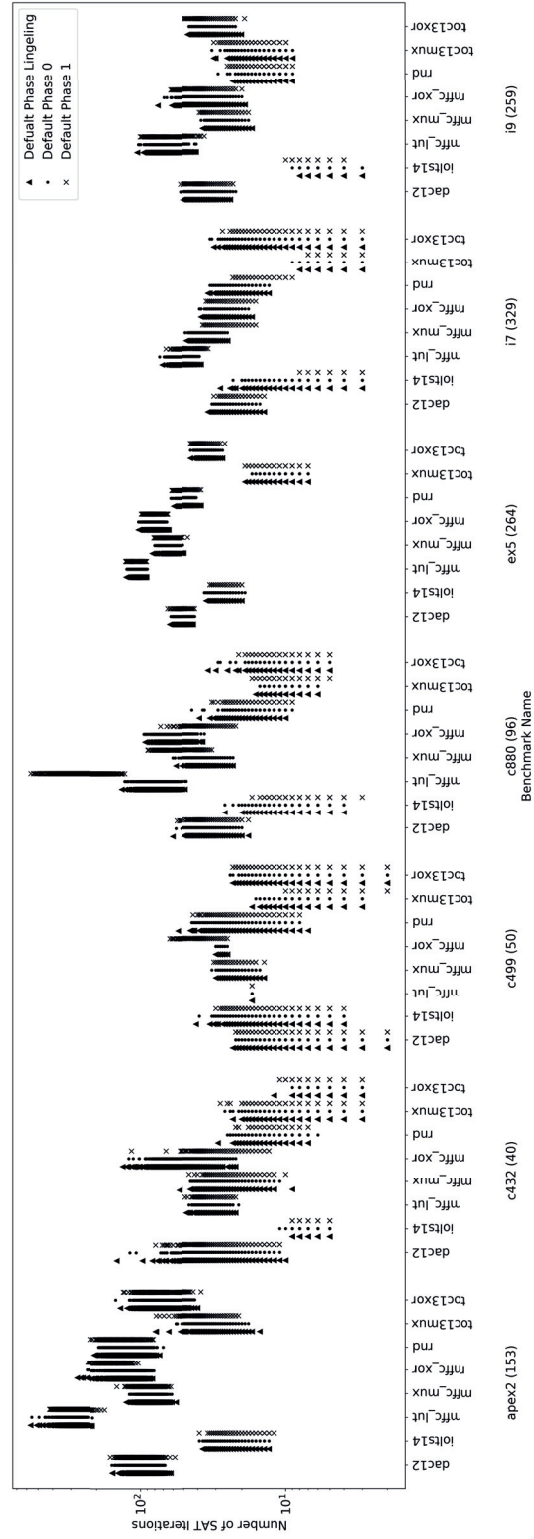


Figure 7.14: Number of required iterations of the SAT attack for in-cone logic locking gate selection methodologies when securing 25% of the gates in the netlist. Number of keys shown in parenthesis next to benchmark name. Evaluations are completed with 1,000 random variable orders and the default phase heuristic of Lingeling [127], logic 0, and logic 1.

Table 7.3: Area, power, and performance overheads of the XOR, LUT, and MUX obfuscation techniques implementing the MFFC and gate controllability based gate selection methodologies for the benchmark circuits apex2, c432, c499, c880, ex5, i7, and i9.

Benchmark	mffc_xor			mffc_lut			mffc_mux		
	Area (%)	Power (%)	Timing (%)	Area (%)	Power (%)	Timing (%)	Area (%)	Power (%)	Timing (%)
apex2	30.9	15.5	43.1	22.7	14.0	5.2	33.5	18.5	77.7
c432	23.1	12.8	9.3	21.4	5.4	18.7	14.6	15.3	20.4
c499	6.7	9.2	35.9	4.5	0.6	16.5	7.5	14.8	17.7
c880	19.3	10.1	8.4	16.5	13.6	4.1	17.7	14.6	13.0
ex5	35.0	4.4	51.6	22.5	5.0	37.1	35.7	19.6	116.7
i7	39.0	13.0	65.4	26.1	0.7	52.3	40.5	14.9	28.5
i9	25.9	75.4	108.7	18.9	38.5	52.8	28.7	26.6	112.7

Table 7.4: Area, power, and performance overheads of the dac12, iolts14, rnd, toc13xor and toc13mux logic locking gate selection methodologies for the apex2, c432, c499, c880, ex5, i7, and i9 benchmark circuits.

Benchmark	dac12			iolts14			rnd			toc13xor			toc13mux		
	Area (%)	Power (%)	Timing (%)	Area (%)	Power (%)	Timing (%)	Area (%)	Power (%)	Timing (%)	Area (%)	Power (%)	Timing (%)	Area (%)	Power (%)	Timing (%)
apex2	29.7	14.4	3.7	8.6	5.2	8.6	41.3	40.7	23.4	31.2	19.0	42.0	25.7	4.9	54.6
c432	18.1	8.6	21.0	23.1	3.8	22.4	21.2	8.8	35.1	31.4	25.2	43.3	37.3	34.9	51.3
c499	6.1	6.6	21.0	1.7	-3.7	3.6	9.0	1.7	39.5	8.4	18.6	33.1	7.6	6.1	19.0
c880	18.5	6.1	2.3	5.5	4.2	4.8	15.6	6.9	9.1	16.9	13.5	22.5	21.4	19.2	38.6
ex5	47.4	27.7	67.7	10.4	1.2	5.9	64.3	52.0	41.4	73.6	77.1	87.6	75.6	41.5	303.8
i7	60.0	11.8	103.1	22.4	1.8	1.5	60.1	32.1	79.2	62.5	32.4	116.2	55.2	51.4	99.2
i9	19.4	64.4	55.0	4.8	-63.2	27.5	31.7	67.9	44.1	23.6	174.4	101.7	27.1	14.2	203.5

Table 7.5: The minimum percentage and average percentage improvement in the minimum number of cubes, minimum number of iterations, and minimum CPU time over 100 variable orderings to complete the SAT attack is provided for the XOR, LUT, and MUX based obfuscation techniques implementing the MFFC and gate controllability based logic locking gate selection methodologies for a 128 bit key as compared to the dac12, iolts14, rnd, toc13xor, and tox13mux gate selection methodologies.

Benchmark	Num. Gates	mffc_xor			mffc_lut			mffc_mux		
		Num. Cubes	Num. Iterations	CPU Time (s)	Num. Cubes	Num. Iterations	CPU Time (s)	Num. Cubes	Num. Iterations	CPU Time (s)
c7552	3512	8.91E+05 Min: 54.04% Avg: 423.7%	46 Min: 39.29% Avg: 447.6%	8.67 Min: 8.44% Avg: 598.3%	8.87E+05 Min: 53.35% Avg: 421.4%	47 Min: 42.42% Avg: 459.4%	4.65 Min: -41.90% Avg: 269.2%	7.38E+05 Min: 54.04% Avg: 334.0%	38 Min: 15.15% Avg: 352.3%	2.53 Min: -68.30% Avg: 101.3%
c5315	2307	4.89E+05 Min: 93.48% Avg: 263.2%	35 Min: 59.09% Avg: 235.3%	7.04 Min: 5.28% Avg: 648.7%	5.91E+05 Min: 133.60% Avg: 338.6%	44 Min: 100.0% Avg: 321.5%	5.01 Min: -25.0% Avg: 433.0%	5.57E+06 Min: 93.48% Avg: 313.6%	40 Min: 81.81% Avg: 283.2%	3.13 Min: -53.30% Avg: 232.2%
ex1010	5066	2.39E+06 Min: 67.78% Avg: 679.5%	93 Min: 69.09% Avg: 874.8%	10.03 Min: 71.70% Avg: 547.0%	1.74E+06 Min: 22.17% Avg: 467.5%	69 Min: 25.45% Avg: 623.3%	6.08 Min: 186.0% Avg: 292.2%	1.48E+06 Min: 67.78% Avg: 381.1%	57 Min: 3.63% Avg: 497.5%	4.91 Min: 10.01% Avg: 216.5%
des	6473	1.59E+06 Min: 127.1% Avg: 697.1%	49 Min: 133.3% Avg: 926.6%	5.6 Min: 137.60% Avg: 362.1%	1.34E+06 Min: 92.04% Avg: 547.1%	42 Min: 100.0% Avg: 780.0%	3.97 Min: 68.60% Avg: 227.8%	1.43E+06 Min: 127.1% Avg: 616.7%	44 Min: 109.52% Avg: 821.9%	4.9 Min: 107.80% Avg: 304.2%

run time. The 2x1 MUX MFFC based selection methodology increases the minimum number of iterations to complete the SAT attack by 38.2% and increases, on average, the mean number of iterations by 23.1% across all benchmark circuits. The MUX based selection strategy results in a 26.6% increase in the average CPU run time, with a 17.4% increase in the minimum run time. The power, performance, and area overheads of the apex2, c432, c499, c880, ex5, i7, and i9 benchmark circuits are provided in Table 7.3 for the XOR, LUT, and MUX MFFC based gate selection algorithms and in Table 7.4 for the dac12, iolts14, rnd, toc13xor, and toc13mux methodologies. All techniques were synthesized using Synopsys DC Compiler in a 180 nm technology, with the toggle rate of the key inputs set to 0%. Overall, the overheads for all of the techniques are large, with a few benchmark circuits for each technique resulting in single digit overheads in area, power, and/or timing. One exception was the power overhead after implementing the iolts14 selection methodology, which inserts AND and OR gates into the netlist to increase the switching probability of nets, which enhances the detection of Trojans for a given key. However, as the correct key was unknown when performing the power analysis, the switching activity of the circuit was modified, which resulted in low or reduced power consumption for iolts14. While the overheads for the techniques are significant, the primary optimization criteria was security. A reduction in the overhead is possible by implementing a gate selection methodology that applies a multi-objective optimization algorithm.

In addition to the benchmark circuits characterized by results included in Fig. 7.13, an analysis of the largest benchmark circuits from ISCAS'85 and MCNC is performed, with the results listed in Table 7.5 for a key size of 128 bits. The minimum number of cubes, number of iterations, and CPU time for the XOR, LUT, and MUX MFFC based gate selection methodologies across 100 variable orderings are listed. The minimum percentage and average percentage increase for the XOR, LUT, and MUX based techniques when compared with `dac12`, `toc13xor`, `tox13mux`, `rnd`, and `iolts14` is also provided. Shorter CPU run times are required for the LUT and MUX based techniques as each iteration is generally shorter than XOR based methods. The increased iteration time for the XOR techniques may be an artifact of using a non-incremental SAT attack [132]. Another consideration is that the iteration time increases when the oracle model is not provided to the SAT attack, forcing an adversary to query the IC and wait for the corresponding response, which often increases the iteration time.

7.5.1 Comparison with SFLL

The corruption of the primary output signals of the obfuscated benchmark circuits is analyzed, with results listed in Table 7.6 for 1,000 random input patterns and 100 random key patterns applied to each random input. As expected, the `toc13xor` and `toc13mux` techniques provide the highest percentage of corruption, as the techniques were developed to produce a 50% Hamming distance as compared with the original

Table 7.6: Average percentage of corrupted outputs of the apex2, c432, c499, c880, ex5, i7, and i9 benchmark circuits for 1,000 random input patterns. 100 random key sequences are applied to each of the 1,000 random input patterns.

Security Technique	apex2	c432	c499	c880	ex5	i7	i9
dac12	12.68	17.85	0.78	3.32	6.31	47.30	40.74
iolts14	2.61	30.18	18.70	13.52	18.33	11.96	38.57
mffc_xor	0.09	13.34	3.17	1.97	0.39	2.23	20.36
mffc_lut	0.13	11.22	3.08	1.82	0.29	1.91	28.07
mffc_mux	0.03	6.00	1.04	1.54	0.69	2.87	23.00
rnd	7.05	21.09	6.03	14.70	12.67	27.58	43.31
toc13mux	68.08	37.35	14.99	15.88	16.86	37.51	44.37
toc13xor	50.14	45.24	27.96	19.08	19.08	48.70	43.90

outputs of the circuit. In addition, implementation of the MFFC based techniques results in a lower percentage of corruption as gate controllability is used as an insertion metric, which favors less observable gates within the netlist.

The implemented SLL technique that includes Hamming distance error correction circuitry is $k - \log_2 \binom{k}{h}$ secure against the SAT attack, where k is the total number of key bits and h is the target Hamming distance of the primary outputs [101]. The probability of a successful attack is, therefore, $q/2^{k - \log_2 \binom{k}{h}}$, with q number of queries made to the circuit [101]. The number of altered minterms is $\binom{k}{h}$, while the number of non-altered minterms is $(2^n - \binom{k}{h})$, where n is the total number of inputs. The corruption of the outputs of the circuit for a non-altered minterm is $\binom{k}{h}/2^k$, while an altered minterm results in $(2^k - \binom{k}{h})/2^k$ corrupted outputs. The number of iterations for a success rate of 50% and the corresponding percentage of corrupted outputs are

listed in Table 7.7 for $h = k/4$, $k/3$, and $k/2$. The data indicates that for lower k values and increased levels of output corruption, in-cone techniques are capable of outperforming out-of-cone techniques. The analysis is completed by applying the SAT attack, and not the more recently developed dominant attack vector for SFLL described in [108], where properties of the Hamming distance checking circuitry are exploited.

7.6 Discussion of Trends in Logic Locking

While out-of-cone techniques such as SFLL provide provable security against the SAT attack when the number of keys is large, the data listed in Table 7.7 indicates that the security is not adequate when the number of key bits is limited and the desired level of corruption of the primary outputs is high. In-cone techniques, such as the MFFC based methods developed in this paper, outperform SFLL for a low number of key bits and for circuit conditions that require a higher level of output corruption. However, even though the resiliency against the SAT attack increases, execution of the SAT attack still determines the correct key within tens of seconds. The concern is that even for large circuits, scan chain access partitions the IC into smaller blocks that provide an easy means to attack obfuscated circuits. Scan chain obfuscation, such as the work proposed in [133–140], is consequently vital to the effective implementation of logic locking techniques. However, many of the methods

Table 7.7: Analysis of the average number of iterations and output corruption percentage for $h=k/4$, $k/3$, and $k/2$ when implementing SFLl.

Benchmark	# Keys	Average Iterations			Corruption		
		k/4	k/3	k/2	k/4	k/3	k/2
apex2	30	263.71	17.87	3.46	0.38	5.44	24.72
c432	8	4.57	4.57	1.83	19.48	19.48	39.73
c499	10	11.38	4.27	2.03	8.40	20.69	37.11
c880	19	67.63	9.66	2.84	1.47	9.81	29.03
ex5	53	5352.55	139.39	4.63	0.02	0.71	19.28
i7	66	43129.06	202.51	5.11	2E-3	0.49	17.65
i9	52	3546.07	102.61	4.54	0.03	0.97	19.60

that obfuscate the scan chain do not secure the logic between the registers, which allows for the reverse engineering of the IC. Modifications to the logic cones between registers is, therefore, needed, while accounting for the analysis and results presented in this paper since attacks without scan chain access exist [141]. Any novel technique must ensure obfuscation of both the scan chain and logic cone, with selection of the specific logic locking implementation dependent on the number of cone inputs, desired level of output corruption, and the cone structure, as indicated by the results in this paper.

Chapter 8: Synthesis of Hidden State Transitions for Sequential Logic Locking

The analysis of in-cone logic locking SAT attack resilience, and out-of-cone techniques for circuits with limited inputs described in Chapter 7 demonstrates that alternative security techniques are required to adequately protect against a threat model that includes oracle guided attacks. The partitioned state machine model described in Chapter 5 is one potential measure to limit oracle guided attacks, as the activated and locked IC are in different state spaces. Due to the different state spaces of the activated and locked ICs, the oracle response does not reveal information regarding the activation sequence of the IC. However, as discussed in Chapter 5, sequential logic locking techniques are vulnerable to one or more attack vectors, including structural attacks and the determination of the transition from obfuscated mode to functional mode. In this chapter, the security of sequential logic locking is enhanced through a novel partitioned state machine methodology that 1) does not allow an adversary to observe the start-up sequence of the IC in obfuscated mode, 2) does not allow an adversary to determine the transition from obfuscated mode to functional mode by monitoring and attacking the combinational logic within the FSM, and 3) reduces the observable structural information of the added security elements in the circuit.

8.1 Preventing Scan Enable During Startup

If an adversary is able to restart the circuit and switch to scan mode after each clock cycle, the key sequence is vulnerable to extraction, which limits the security provided by the applied logic locking technique. To prevent access to scan mode when the key sequence is applied, the circuit shown in Fig. 8.1 is utilized. The circuit checks that the number of applied transition signals to the sub-block matches the desired count, ensuring that the activation sequence has completed before scan mode is enabled. If an adversary inserts additional glitches to try to bypass the counter, the circuit transitions into a state outside of the startup sequence that generates an incorrect key response, which results in either the circuit remaining in obfuscated mode or starting the IC in an incorrect state. The utilization of a hidden state function, as described in Section 8.4.3, permits the designer to determine if the current state is in functional or obfuscated mode before allowing scan chain access, which further protects against adversarial attacks. The observability of the primary outputs is also an important consideration as attacks such as described in [141] and [142] do not require scan chain access. If the protected FSM is observable at the primary outputs, the outputs must also be masked during the start-up sequence to prevent an adversary from determining state transitions by monitoring the outputs of the circuit. As an example, an AND gate with a control signal from the comparator shown in Fig. 8.1 prevents the outputs from switching until the startup sequence completes.

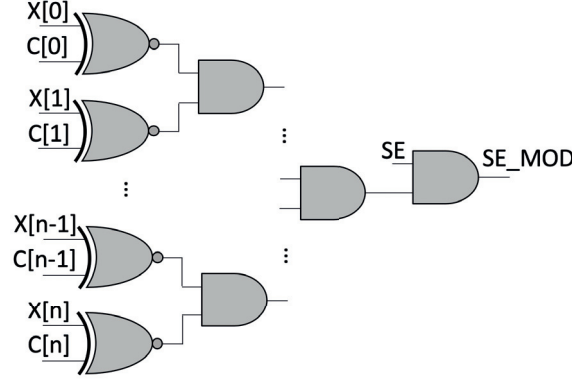


Figure 8.1: A count register C is compared with a desired count X to determine the number of transitions that have occurred, which allows entrance to scan mode if correct.

The functional testing of an IC is minimally impacted as scan chain access is only blocked during the application of the key sequence. Assuming that the testing facility is not trusted, the IC is provided for testing without the correct key. The scan chain becomes accessible after the IC is powered on and the test facility waits the required number of cycles for the secure key sequence to complete. After the scan chain is accessible, the testing facility is able to apply any sequence to the IC to verify the functionality of the device.

8.2 Security Overview Against Activation Sequence Extraction

Since access to an oracle IC only provides information regarding the state transitions of a circuit operating in the functional mode, the SAT attack no longer applies to the circuit topology developed in this article as the key sequence is not observable due to the included partitioned state machine. The adversary is, therefore, tasked

with finding a path from the starting obfuscated state to the ending obfuscated state in s cycles, where s is the number of cycles set by the k -bit length key stored in tamper proof memory or deduced from the run-time beginning from start-up until activation. The adversary is essentially performing a depth first search of the FSM, attempting each activation sequence. Assuming the FSM has n inputs, f state flip-flops, and that a transition to each state exists, an adversary must search $2^{\min(n,f)*s}$ paths. However, the likelihood that each state transitions to every other state is low. Therefore, the number of paths is weighted by the α term, which results in a search space of $[\alpha * 2^{\min(n,f)}]^s$, where α is between 0 and 1. Each path is assumed to have a uniform chance of being the activation sequence, which results in the expected number of paths an adversary must traverse given by (8.1). An overview of the security against SAT based attacks when functional mode FSM modifications are implemented is provided in Section 8.4.5.

$$E[X] = \sum_{i=1}^{[\alpha * 2^{\min(n,f)}]^s} \frac{i}{[\alpha * 2^{\min(n,f)}]^s} = \frac{[\alpha * 2^{\min(n,f)}]^s}{2} \quad (8.1)$$

8.3 Hidden State Transitions

A methodology to eliminate the logical connection between the obfuscated mode and functional mode is provided in this section. The transition of the state machine

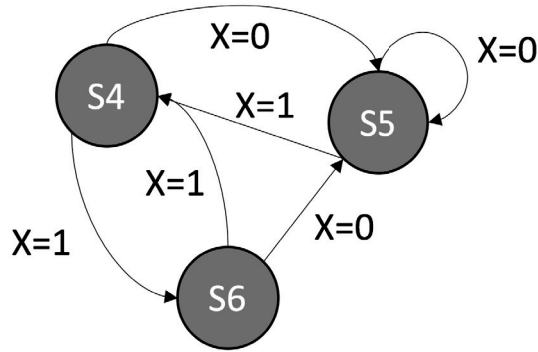


Figure 8.2: Original state machine with single input X .

once the logical representation in the state transition logic has been removed is referred to as a hidden state transition. Two different methods to generate hidden state transitions are described, which include 1) altering the clock frequency and 2) inserting logic to gate the clock signal, where both produce a strategic timing glitch in the circuit. There are two primary security advantages that hidden state transitions add over traditional logic based transitions. The first is the ability to increase the number of paths the adversary must search. Since an adversary is unaware of whether or not a hidden state transition occurs during a given clock cycle, the number of possible paths increases, which corresponds to an increase in the α term of (8.1). The second benefit is that the additional paths are not dependent on modifications to the original state transition function. The amount of structural information on the circuit corresponding to the execution of the hidden state transition is, therefore, limited.

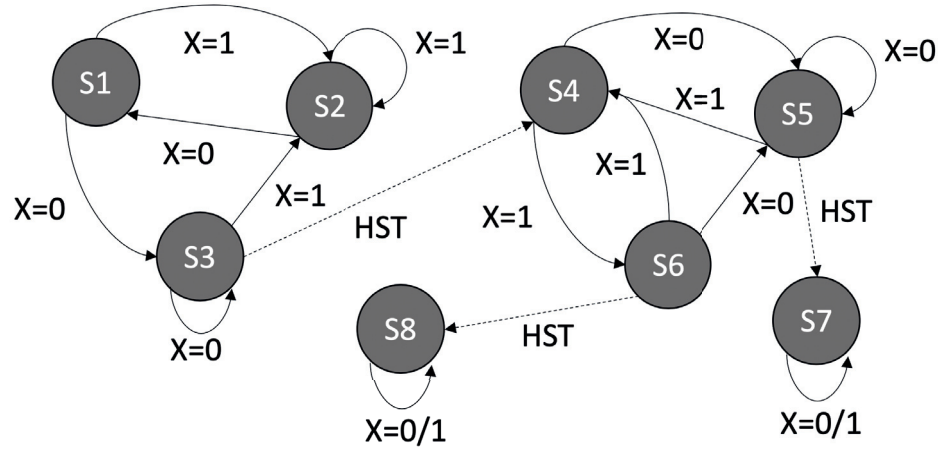


Figure 8.3: State machine with hidden state transitions (HST) from $S3$ to $S4$, $S6$ to $S8$, and $S5$ to $S7$ that activate when a strategic timing glitch is applied. A single input X is applied to the circuit.

8.3.1 Frequency Based Hidden State Transitions

To demonstrate a frequency based hidden state transition, consider the original state machine shown in Fig. 8.2. The desired transformation of the original FSM results in a structure similar to that depicted in Fig. 8.3. The hidden state transitions are shown as dotted lines in Fig. 8.3, with the primary hidden state transition from the obfuscated FSM to the functional FSM occurring from $S3$ to $S4$. The transition from $S3$ to $S4$ is dependent on a modification (increase for this example) to the clock frequency, which prevents the state transition logic from propagating the correct logical value to a register of the FSM.

To demonstrate the execution of hidden state transitions within a circuit, the FSM shown in Fig. 8.3 was implemented and simulated using SPICE. The results are shown

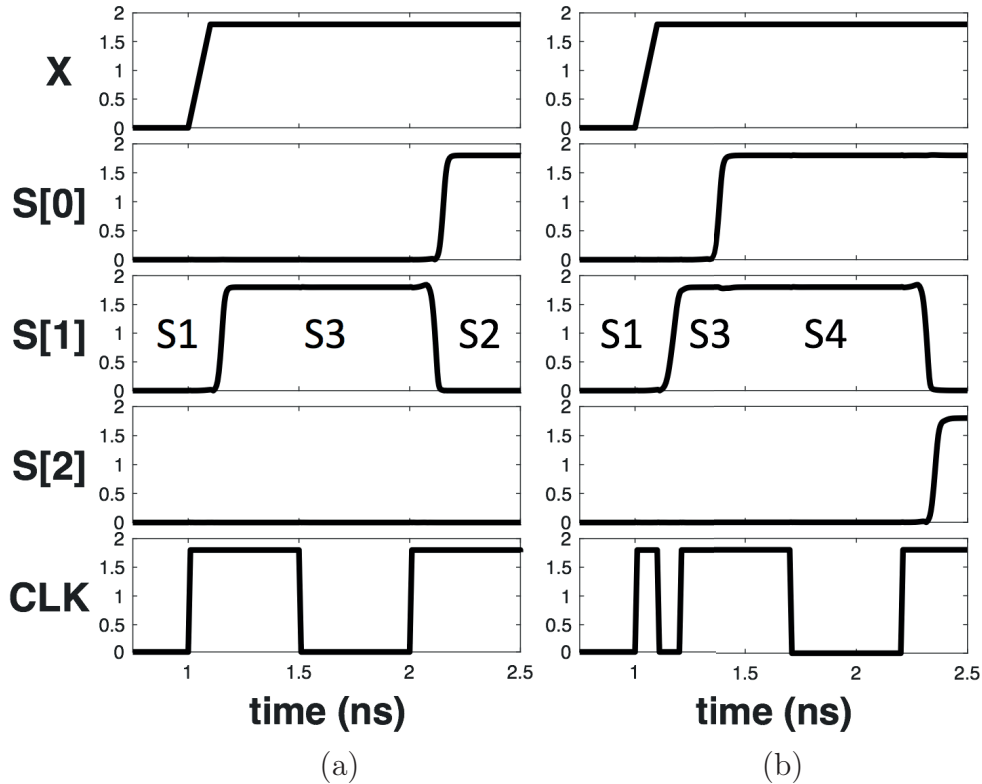


Figure 8.4: Simulation of the state machine with (a) no frequency based state transition, and (b) a temporary increase in clock frequency to enable a hidden state transition to S_4 .

in Fig. 8.4b, which indicate that while in S_3 and a pulse corresponding to a frequency of 5 GHz is applied, the circuit transitions to S_4 (S_1 to S_3 to S_4). However, when the pulse is not applied, the circuit transitions to S_2 on the next clock edge, as shown in Fig. 8.4a. By applying a clock pulse corresponding to a frequency of 5 GHz, the state transition logic for register S[0] does not have enough time to switch to logic 1. Instead, S[0] is held at logic 0, which places the FSM in the functional mode.

The timing slack of the path chosen for insertion of the hidden state transition determines the amount of variation that is tolerable by the circuit. If the path provides

minimal timing slack, meaning that the propagation of the logic value requires the majority of the clock cycle, then the execution of the hidden state transition is possible at any time prior to the propagation of the signal(s) through the combinational logic. However, if the logical path includes significant slack, the arrival time of the key signal used to execute the hidden state transition is constrained to a shorter time interval. Path timing analysis is, therefore, required to ensure that the hidden state transitions occur for all scenarios while accounting for process, voltage, and temperature (PVT) variations.

8.3.2 Clock Gating Based Hidden State Transitions

In addition to relying on changes to the applied clock frequency, hidden state transitions are also implemented by adding logic to the clock port of the registers to generate controlled glitches. Three different topologies that modify the clock signal based on the applied key are shown in Fig. 8.5. The first topology shown in Fig. 8.5a allows the key signal to gate the clock signal directly. However, a different topology is needed if the circuit already requires clock gating for power management, which includes an XOR connected to the control signal to allow for the execution of the clock gating technique as shown in Fig. 8.5b. As a result, the state of a given register is held when $KEY0$ is 0. If clock pulses are required, then the circuit topology shown in Fig. 8.5c is utilized, which implements an AND/OR gate [143, 144] that allows for

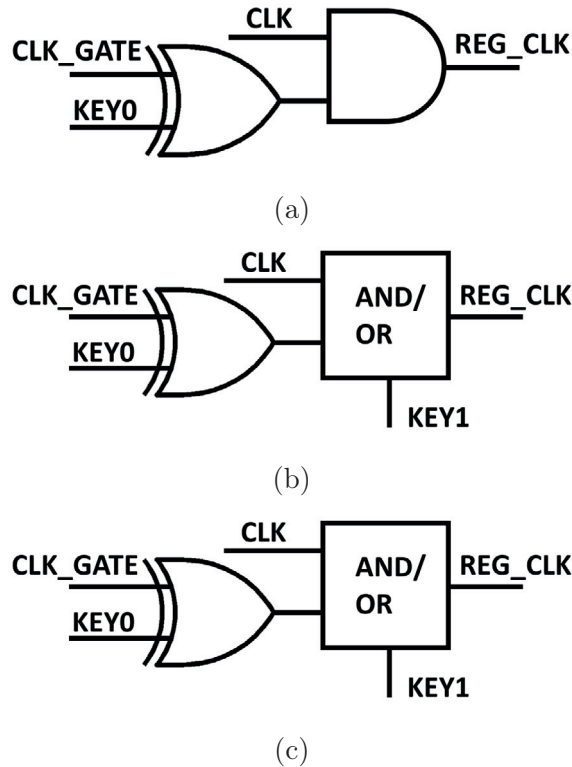


Figure 8.5: Circuit topologies to implement logic based hidden state transitions: (a) Key controlled gating of the clock signal, (b) clock signal gated with an AND gate, where the second input of the AND gate is the output of an XOR gate included to implement clock gating, and (c) clock signal gated with an AND/OR [143, 144] and with an XOR integrated to implement clock gating.

the insertion of clock pulses when setting *KEY1* to 1.

To analyze the acceptable variation in the arrival time of the key signal when executing the hidden state transition, the FSM shown in Fig. 8.3 is utilized. The circuit shown in Fig. 8.5a is connected to the clock input of register S[0] and the arrival time of *KEY0* is analyzed. The results of the sweep of the arrival time of *KEY0* are shown in Fig. 8.6, which indicate a large range of acceptable arrival times of *KEY0*. The minimum arrival time is determined by the hold time of the register

at the start of the current clock cycle, as shown at time $t = 0.25$ ns in Fig. 8.6. The maximum acceptable arrival time is determined by the hold time of the register for the next clock cycle, as shown at time $t = 2.25$ ns in Fig. 8.6. The *KEY0* signal was modeled with a 30% variation in arrival time and for both the slow-slow (SS) and fast-fast (FF) corners. The results indicate that a significant portion of the clock cycle is available for execution of the hidden state transition when *KEY0* is applied on the falling edge of the clock signal.

8.4 Synthesis of Hidden State Transitions

In Chapter 5, structural attacks on sequential logic locking methods are described, that allow for the identification of the circuitry added to implement a given security technique by determining the connections to the tamper proof memory. Discerning the added structure(s) allows an adversary to determine the enable signals used in [115] or the flip signals used in [125]. The determination of the modification function further exacerbates security concerns as value assignment is static after a correct key is applied to the circuit. An adversary knows that the signals are set to logic 0 for correct operation, which provides a means to bypass/remove the circuitry or reduce the search space to determine the correct activation pattern. In order to reduce structural leakage from partitioned FSMs, the modified FSM must 1) be integrated within the original FSM and 2) not apply a static signal to the hidden state registers.

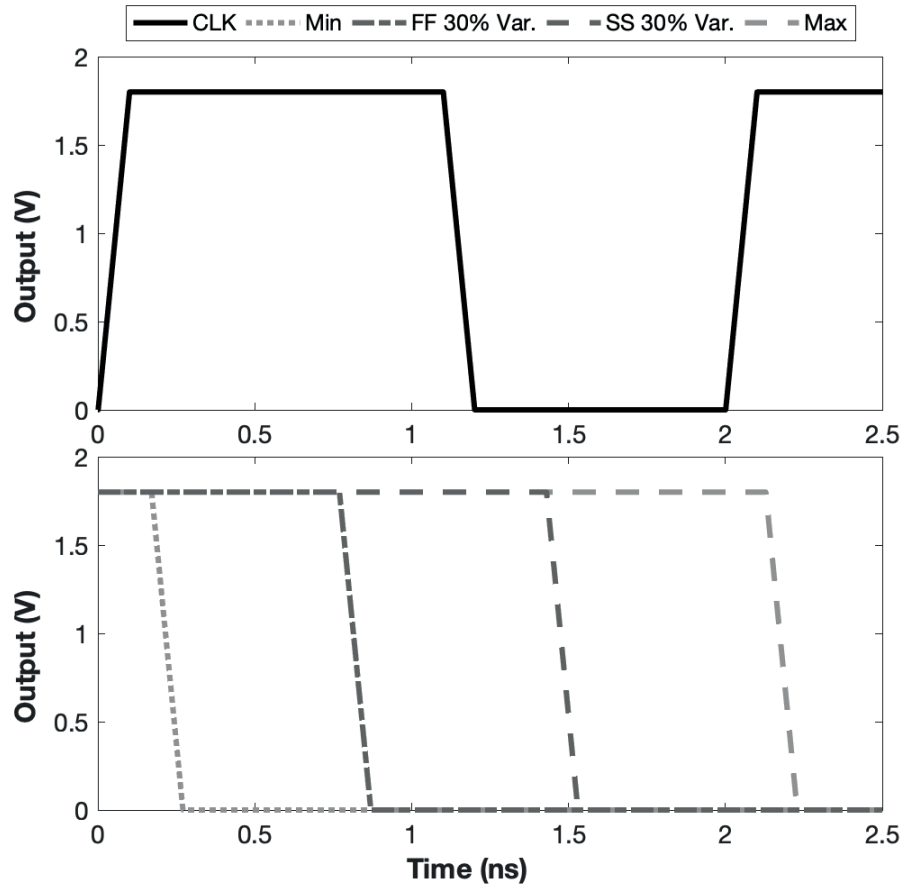


Figure 8.6: Variability analysis of the arrival time of *KEY0* for a hidden state transition implemented on register *S[0]* in the FSM shown in Fig. 8.3. The minimum and maximum arrival times for the *KEY0* transition are shown, as well results from analysis of the slow-slow (SS) and fast-fast (FF) corners, which indicates proper circuit functionality with a 30% variation in the arrival time of *KEY0*.

An algorithm is, therefore, developed to incorporate hidden state transitions into an FSM while avoiding static register values when inserting the additional components to the state machine. A flowchart of the primary steps of the algorithm, as shown in Fig. 8.7, begins with an unobfuscated FSM and upon completion results in a secure FSM.

8.4.1 Identification and Grouping of FSMs

The method described in [145] is utilized to identify and group the state registers of a netlist into FSMs. All registers are first examined to identify if any paths exist that form a closed loop, where the output from a given register returns to the input of the same register. The list of potential state registers is further filtered by grouping memory elements that share enable and clock gating signals [145]. The potential state registers are then grouped by examining the netlist for strongly connected components [146], where each sub-group includes a subset of the potential state registers that share state transition logic. The final outcome is a set of groupings of FSMs within the original netlist.

The worst case time complexity, $O(|V| + |E|)$, occurs when the algorithm must explore the entire set of vertices $|V|$ and edges $|E|$ of the directed acyclic graph (DAG) representing the netlist of the circuit. If no FSM groupings are found, the algorithm terminates before any other stages of the algorithm are executed.

8.4.2 Modification of State Transition Logic

While a partitioned state machine allows for secure activation and protection against IC counterfeiting and overproduction, structural attacks, as pertaining to IP theft, are still a concern. A possibility exists that an adversary is able to identify the mapping function of the hidden state transition and determine the added hidden

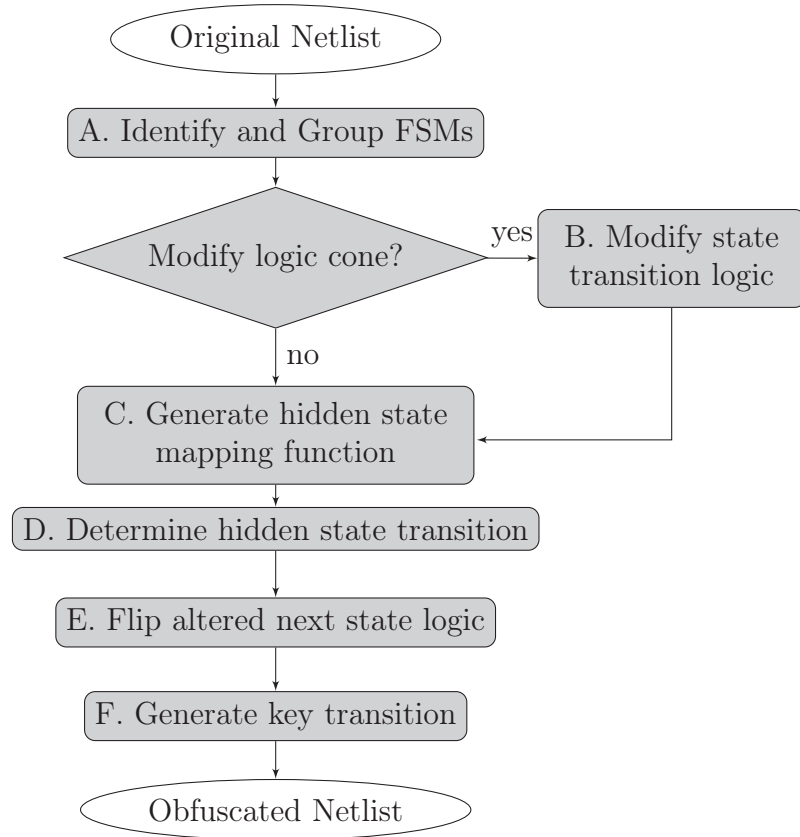


Figure 8.7: Primary steps to transform an unobfuscated circuit into a circuit that includes a secure partitioned FSM with a hidden state transition. The procedural steps that include a letter are further described in Section 8.4.

state registers if the modifications to the state mapping are not integrated within the original state transition logic when synthesizing the netlist. If IP theft is a concern, then modification of the original state transition logic is needed. Consider the example FSM shown in Fig. 8.2. The transition from $S5$ to $S4$ when $X = 1$ is changed to the transition $S5$ to $S6$, and the transition from $S5$ to $S5$ when $X = 0$ is changed to $S5$ to $S4$. Modifying the conditions of the state transitions results in the altered state

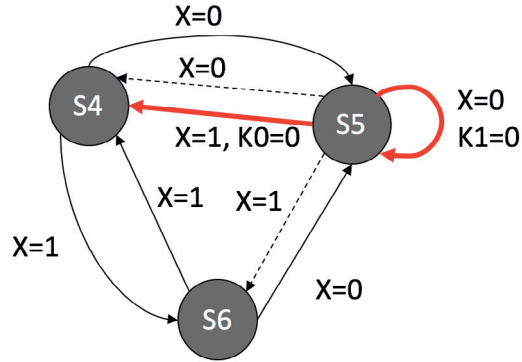


Figure 8.8: Modified FSM that requires a temporally dependent key. Dotted lines represent incorrect edges that are corrected by asserting $K0$ or $K1$ in state $S5$ (correct transitions are shown as thick red lines).

machine shown in Fig. 8.8.

The correct functionality when in state $S5$ is achieved by setting $K0$ to 0 when X is 1 and $K1$ to 0 when X is 0, as shown by the thick red transition arrows in Fig. 8.8. The required circuitry to execute the necessary corrections outside of the original logic cone is described in Section 8.4.5. The in-cone modifications include either an XOR if the function is altered in both the active and obfuscated mode, or the circuits shown in Fig. 8.5 and described in Section 8.3 if there is no modification required for the hidden state transition. When using the circuits shown in Fig. 8.5, the corresponding combinational logic path does not incur an additional delay for the correction of the state mapping, which results in lower overhead in performance.

Each protected cube added to the FSM requires the addition of approximately $3 * n$ vertices to the netlist represented by a DAG, where n is the number of inputs

to the FSM. Since the addition of a vertex to the data structure of the adjacency list graph is $O(1)$, the worst case time complexity of adding the protected cubes is $O(c * n)$, where c is the number of added cubes. However, the number of cubes added is typically small as each additional protected cube incurs significant overhead in power consumption and area, which results in n as the dominant term. Therefore, the time complexity of adding the additional cubes is typically $O(n)$.

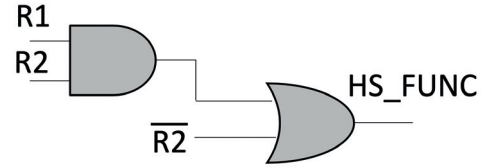
8.4.3 Generation of Hidden State Mapping Function

To avoid a static mapping of the hidden state register when the correct key sequence is applied, a novel mapping function of the hidden states is developed to ensure that the added hidden state register periodically switches between logic 0 and logic 1. The primary challenge is remapping the original states with the new hidden state bit. Applying a mapping of the old state to each new state becomes prohibitively costly in terms of run-time as the number of original states increases, with each additional register resulting in an increase in the total number of new states by a factor of 2. Instead, the use of a random logic function as the hidden state mapping results in a significant reduction in the time to re-map the states into the new state space. However, applying a purely random function results in a large increase in the area and power of the circuit and potentially degrades the performance as well.

To avoid a significant degradation in the quality of results (QoR) of the obfuscated

HS	R1	R2	X
1	0	0	0
1	0	0	1
0	0	1	0
0	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

(a)



(b)

Figure 8.9: Example of a hidden state mapping function where (a) depicts the logical remapping of the states in the form of a truth-table and (b) is a schematic of the circuit that implements the hidden state mapping function of an FSM.

FSM, the hidden state mapping function is derived from an internal net within the combinational logic used for the state transition, or is a function of the state registers of the FSM. The hidden states are then derived by XORing the hidden state mapping function. An example of the re-mapping of the state machine shown in Fig. 8.2 is provided in Fig. 8.9, where the hidden state function is $(R1 \cdot R2) + \overline{R2}$. In addition, a random subset of the outputs of the original FSM are flipped to mask the original transitions of the FSM. The modification of the outputs of the original FSM is further described in Sections 8.4.2 and 8.4.5.

In the case where an internal net is utilized as the hidden state mapping function, the complexity is $O(1)$. If, however a random function implements the hidden state mapping, then the complexity becomes $O(n)$, where n is the total number of registers

1) Determine Ending State in Functional Mode

HS[3]	S[2]	S[1]	S[0]
0	0	0	0

2) Flip Random Non-HS State Bit

0	0	1	0
---	---	---	---

3) Generate Random Non-Flipped State Bits

0	1	0	1
---	---	---	---

Figure 8.10: Procedural steps to generate the hidden state transition values. The steps include: 1) selection of the functional mode state, 2) flipping of a randomly chosen non hidden state (HS) bit, and 3) random selection of the bits not flipped in step two. The state bits that are modified during each step are highlighted in gray.

included in the selected FSM. The $O(n)$ bound is a result of the random combination of the sub-set of registers chosen for the hidden state mapping function, which requires the random selection of a pair of registers until no further register pairs exist.

8.4.4 Determine Hidden State Transition

Once the hidden state mapping function is embedded into the FSM, the hidden state transition is generated via the pseudocode provided as Algorithm 13. To generate the hidden transition, at least one register is chosen as the hidden state register. For example, consider the process shown in Fig. 8.10, where the starting state in functional mode is *0000* and the most significant state bit HS(3) is the hidden state bit. Another random state is generated with a one-bit Hamming distance, which, for

Algorithm 13: Determination of hidden state transition

```

Input: FSM netlist netlist;
Hidden state function info hs_func_info;
/* Determine random obfuscated end state */
hst_end = get_hst_end(netlist, hs_func_info);
/* Determine obfuscated start based on end state */
hst_start = get_hst_start(netlist, hst_end);
/* Cut netlist at state registers */
fsm_netlist = get_fsm_netlist(netlist);
/* Add flip signals to fsm_netlist */
add_flip_signals(fsm_netlist);
/* Unroll fsm_netlist 2x */
for state_io in fsm_netlist do
    add_state_constraints(fsm_netlist, state_io);
end
is_sat = satisfy_constraints(fsm_netlist);
if is_sat then
    /* Extract states from SAT model */
    return flipped_dffs, hst_start, hst_end;
else
    return None;

```

the given example, is *0010*. The generated state serves as the ending state of the obfuscated FSM when the hidden state transition does not occur. The second least significant bit (LSB) of the starting state of the obfuscated partition must, therefore, equal the second most LSB of the starting state of the original FSM, but may differ in value for any other bit location. In the case of the given example shown in Fig. 8.10, assume the generated state is *0101*. Now, when transitioning from state *0101* to *0010*, if the second least significant bit is held low during capture, the FSM transitions into *0000* instead of *0010*.

Once the desired states of the hidden transition and the starting state of the

functional FSM are generated, a logical transition from state 0101 to 0010 does not exist within the FSM state transition logic. Note that the starting state of the functional FSM does not have to coincide with the ending state of the hidden transition. To allow for such transitions, an XOR gate is added at the end of the combinational logic path for each state register, with the other input to the XOR taken from the primary inputs of the original logic circuit, as shown in Fig. 8.11. In addition, the states chosen for the hidden state transition must be reachable from other states of the FSM. To ensure the states are reachable, the state transition logic is unrolled twice (more state unrolling is possible), as shown in Fig. 8.11, where the desired output of state m is the value of the ending state of the hidden state transition in the functional partition, and the desired output of state $m - 1$ is the value of the starting state of the hidden state transition in the obfuscated partition. The only constraints for the states prior to $m - 1$ are that the state values do not equal the starting or ending states of the hidden state transition, and that the state value of $m - 3$ does not equal the state value of $m - 2$. The constraints of the FSM are provided to a SAT solver, which, when successfully completed, validates the inclusion of a given hidden state transition and determines if the output of a given register must be flipped. If the SAT solver returns UNSAT, the current hidden state transition is discarded and a new hidden state transition is generated and validated by the solver, with the process repeating until a valid transition is identified or the maximum number

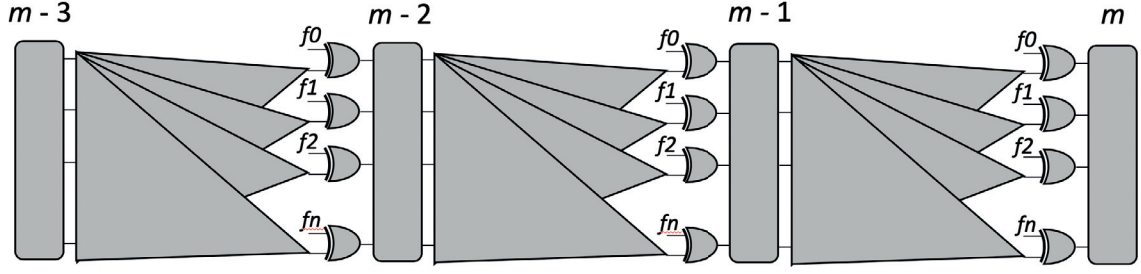


Figure 8.11: Unrolling the combinational logic of the state machine. The ending state of the hidden state transition in the functional partition is m , the starting state of the hidden transition is $m - 1$, and states $m - 2$ and $m - 3$ are previous states that are constrained to not equal states m or $m - 1$. The f_0 to f_n inputs represent the flip signals determined by the SAT solver to meet the required constraints.

of attempts, as set by the designer, is reached. Reaching the maximum number of attempts indicates that the generated hidden state mapping function is not well suited for modification with a hidden state transition. The selection of a hidden state mapping function is, therefore, restarted to generate a hidden state transition with a new set of constraints.

The runtime for the generation of the hidden state transition is bounded by the use of the SAT solver, which is an *NP-complete* problem [147]. The generation of an UNSAT condition for a given set of constraints of the hidden state transition results in the formulation of a new set of constraints. In this work, the number of restarts is limited to 500; however, the limit is user specified. If during the generation of the hidden state transition an UNSAT condition occurs and the number of restarts has been exhausted, the algorithm terminates without generating the modified FSM.

8.4.5 Flip Outputs of Altered Registers

A list of the outputs of registers that must be flipped in either the obfuscated or functional mode is generated as a result of executing subroutines of the hidden state transition algorithm described in Section 8.4.2 and Section 8.4.4. If the state transition logic is not modified, then the output of the function generated as described in Section 8.4.3 is utilized as the flip signal of the altered registers. The outputs of the modified registers are then simply connected to an XOR gate, with the other input to the gate connected to the flip signal.

If, however, the original state transition logic is modified, then the flip signal is generated from the hidden state function using the inputs to the FSM. The logical structure of the correction circuit is shown in Fig. 8.12, where the current state of the circuit is compared to a key value to determine if the output of the cone must be flipped or not. The correction circuit serves the function of a look-up-table (LUT) for the modified I/O pairs of the original circuit. Assuming that every register is accessible via the scan chain in functional mode, the sequentially locked circuit is now equivalent to a combinational structure vulnerable to an oracle guided attack. The estimated security against the SAT attack is then equivalent to SFLL-Flex [101], with each DIP given a uniform probability of $c/2^n$ to select a protected cube, where c is the number of cubes and n is the number of inputs to the logic cone. The probability of selecting a protected input cube is then given by $1/2^{n-\log_2 c}$, with the expectation

Table 8.1: Results from the analysis of the ISCAS'89 benchmark circuits including 1) the number of flip-flops in the obfuscated FSM, 2) the number of inputs to the FSM being obfuscated, 3) the hidden state transition random walk score (random walk score without hidden state transitions), 4) an estimate of the average number of paths to determine the activation sequence from (8.1), 5) the number of iterations to generate the hidden state transition, and 6) the runtime to obfuscate the circuit.

Benchmark	Number of FSM FFs	Number of Inputs	Random Walk Score (α)	Avg. Number of Paths	Number of HST Iterations	Runtime (s)
s27	4	4	35.5 (26.3)	5.87E+11	1	0.18
s13207	89	580	29.2 (27.3)	9.04E+25	1	13.08
s1423	51	41	25.86 (19.5)	1.13E+16	1	0.77
s1488	7	8	3.18 (2.0)	7.97E+04	9	5.48
s15850	99	484	18.24 (14.4)	5.78E+28	1	20.47
s298	11	7	5.54 (5.0)	5.60E+05	14	1.6
s344	16	9	16.36 (13.8)	3.51E+09	2	0.52
s35932	3	1473	16.00 (14.5)	6.40E-01	1	20.98
s382	14	11	4.7 (2.9)	7.15E+09	3	0.66
s38417	76	1589	3.0 (2.2)	1.13E+21	2	37.44
s38584	138	1321	8.4 (5.6)	1.46E+40	1	12.98
s386	7	7	2.44 (1.6)	1.66E+04	20	2.71
s400	14	11	4.68 (2.9)	6.99E+09	10	1.54
s420	17	19	3.3 (2.0)	4.24E+12	173	25.06
s444	14	11	4.52 (2.9)	5.85E+09	5	1.02
s510	7	19	2.54 (1.6)	1.66E+02	2	0.7
s526	14	11	3.08 (2.9)	8.21E+08	3	0.82
s641	16	37	56.64 (51.9)	5.43E+10	1	0.55
s713	16	37	59.24 (42.2)	6.05E+10	1	0.59
s820	6	18	8.84 (6.2)	5.18E+03	1	0.89
s832	6	18	8.16 (5.6)	3.36E+03	2	1.1
s838	33	35	3.26 (2.3)	3.98E+15	390	109.24
s9234	55	188	27.62 (23.9)	4.98E+15	3	11.85
s953	7	16	2.22 (1.4)	1.67E+02	5	2.08

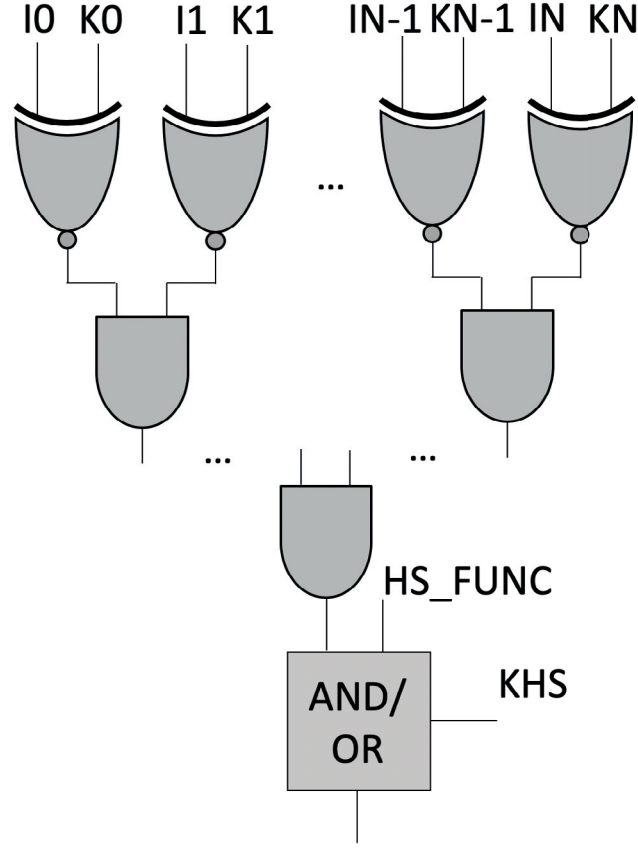


Figure 8.12: Logical structure of the correction circuitry.

for the number of DIPs required given as (8.2). Note that the metric is conservative, as the determination of a single cube is used to evaluate the security of the circuit. An adversary must, however, determine all c protected cubes to extract the key.

$$E[X] = \sum_{i=1}^{2^{n-\log_2 c}} \frac{i}{2^{n-\log_2 c}} = \frac{2^{n-\log_2 c}}{2} \quad (8.2)$$

As discussed in Section 8.4.2, the time complexity of modifying the netlist to include the protected cubes is $O(n)$. In addition to the circuitry required to add the

protected cubes, up to m XOR gates are needed to allow for logical modifications to the outputs of the registers, where m is the total number of registers. Since the addition of each XOR gate is $O(1)$, the time complexity to add the XOR gates to flip the output of the registers is bounded to $O(m)$.

8.4.6 Generation of Key Transition

The key is generated based on a user defined number of key bits k . The algorithm iteratively executes to produce a sequence of transitions through the FSM beginning from the starting state of the hidden state transition to the starting state of the key sequence, which is equivalent to the starting state of the obfuscated partition. The netlist is first unrolled and a constraint on the input state of the circuit is applied to the SAT solver, which requires that the current state and the returned state from the solver are both reachable. The satisfying conditions of the previous state are determined using a SAT solver and are added to a list containing the states of the key sequence. In addition, the generated state is added to a hash table to track the number of times the generated state is visited. If the frequency of visits to a given state in successive iterations of the SAT solver is greater than a set threshold (set to two in this paper), then an additional constraint is added to prevent the solver from returning to the given state. If the solver is unable to generate a solution with the additional constraint on the number of visits, then the constraint is removed and the

solver is executed again. If the solver is still unable to produce a valid solution, then the algorithm exits without generating a key sequence.

Generating the key transition(s) is also bounded by the complexity of executing the SAT solver, which is an *NP-complete* problem [147]. If an UNSAT condition is produced by the SAT solver, the generation phase of the hidden state transition algorithm described in Section VI-D is executed again. If a key is not generated for a second time, the algorithm terminates and restarts the key generation process for another FSM to be secured. In this paper, the number of attempts to generate a key is limited to four; however, the number is user defined. If the maximum number of attempts to generate a key is reached, the algorithm exits without securing a single FSM.

8.5 Results

The algorithm described in Section 8.4 is executed on the ISCAS'89 sequential benchmark circuits. The number of flip-flops within the modified FSM and the total number of inputs to the circuit, which affect the potential security of an FSM, are listed in Table 8.1. A majority of the modified FSMs include at most 15 flip-flops, which are feasibly recoverable with FSM extraction tools, or even brute force analysis.

To estimate the α term from (8.1), an analysis utilizing a random walk is performed for each FSM. The α term is determined by starting with 50 random states

Table 8.2: Area, power, and performance analysis of the ISCAS’89 benchmark circuits with and without sequential obfuscation implemented with hidden state transitions. The percentage overhead in area, power, and performance is also provided for each benchmark circuit.

Benchmark	Original			Obfuscated			Overhead		
	Area (μm^2)	Power (mW)	Timing (ns)	Area (μm^2)	Power (mW)	Timing (ns)	Area (%)	Power (%)	Timing (%)
s13207	3.95E+04	6.87	1.85	4.00E+04	7.39	1.87	1.41	7.10	1.07
s1423	1.04E+04	1.82	1.85	1.06E+04	1.85	1.85	1.99	1.76	0.00
s1488	6.18E+03	1.17	1.85	6.27E+03	1.18	1.87	1.45	0.93	1.07
s15850	5.13E+04	9.70	1.86	5.15E+04	9.72	1.86	0.44	0.20	0.00
s27	2.86E+02	0.03	0.49	4.83E+02	0.06	0.65	40.73	43.78	24.62
s298	1.65E+03	0.57	1.10	1.87E+03	0.65	1.23	11.89	12.54	10.57
s344	1.98E+03	0.35	1.16	2.18E+03	0.37	1.28	9.19	6.38	9.38
s35932	1.75E+05	55.54	0.70	1.77E+05	55.87	0.75	0.65	0.59	6.67
s382	2.31E+03	0.43	1.90	2.50E+03	0.49	1.92	7.41	11.80	1.04
s38417	1.83E+05	48.90	1.85	1.85E+05	53.24	1.87	0.80	8.15	1.07
s38584	1.60E+05	48.68	1.86	1.60E+05	50.41	1.86	0.27	3.43	0.00
s386	1.64E+03	0.32	0.96	1.86E+03	0.34	1.32	12.16	6.64	27.27
s400	2.38E+03	0.51	1.60	2.58E+03	0.56	1.84	7.90	8.80	13.04
s420	2.21E+03	0.38	0.77	2.35E+03	0.45	0.92	6.11	14.78	16.30
s444	2.38E+03	0.52	1.53	2.56E+03	0.61	1.70	6.95	15.45	10.00
s510	2.76E+03	0.64	1.50	2.93E+03	0.66	1.68	5.93	2.08	10.71
s526	2.40E+03	0.46	1.54	2.54E+03	0.48	1.72	5.51	4.42	10.47
s641	2.87E+03	0.47	1.86	3.09E+03	0.49	1.86	7.21	4.23	0.00
s713	2.86E+03	0.47	1.85	3.07E+03	0.48	1.94	6.77	1.58	4.64
s820	3.12E+03	0.48	1.45	3.27E+03	0.48	1.73	4.40	1.32	16.18
s832	3.13E+03	0.48	1.69	3.43E+03	0.49	1.76	8.69	1.84	3.98
s838	4.48E+03	0.67	0.85	4.66E+03	0.72	1.07	3.81	5.87	20.56
s9234	1.61E+04	2.20	1.86	1.74E+04	2.80	1.86	7.52	21.50	0.00
s953	5.18E+03	0.86	0.72	5.39E+03	0.87	0.80	3.86	1.58	10.00
Average	-	-	-	-	-	-	6.79	7.78	8.28

and applying 100 random input sequences, where each input sequence must each be different. The average percentage of unique paths over the 100 walks per state is listed in Table 8.1, with most circuit benchmarks producing a random walk score of less than 10%. Results estimating the α term are provided in Table 8.1 for circuits that include FSMs modified with a hidden state transition as well as for circuits that include no hidden state transitions. In most cases, the FSMs with a hidden state transition produce an α term that is multiple percentage points greater than FSMs

with no hidden state transitions, which indicates that an adversary must search extra paths for the correct start-up sequence. For example, a change in α from 2.9% to 4.7% for the s382 benchmark circuit results in an increase in the average number of paths to search from 6.03E+08 to 7.15E+09. A further increase in the α term is possible by adding the circuit shown in Fig. 8.5a to multiple registers, rather than a single register as was done to produce the results listed in Table 8.1. Applying the hidden state circuitry to all 14 state registers of the s382 benchmark circuit increases the α term to 35.46%.

The estimate of the average number of paths an adversary must traverse obtained from computing (8.1) indicates that circuits with a small number of both flip-flops and inputs only require the analysis of thousands of paths to correctly decrypt. However, when the number of both flip-flops and is sufficiently increased, the estimated average number of paths to analyze becomes computationally expensive, as shown for the s15850 benchmark circuit, which requires the analysis of an estimated 5.78E+28 paths to properly determine the obfuscated activation sequence. In order to enhance the security of netlists comprised of smaller FSMs, or FSMs with low unique path scores given by (8.1), multiple FSMs must be grouped together within the IC. However, some of the small benchmark circuits such as s27 only contain a single FSM. The small circuits are, therefore, not optimal for the implementation of sequential or combinational logic locking.

The runtime of the algorithm to determine and implement hidden state transitions is also provided in Table 8.1, with the algorithm completing for most of the large ISCAS'89 benchmark circuits in the order of 10's of seconds. The longest runtime was 109.242 seconds for the s838 benchmark circuit, which required 390 restarts of the algorithm to determine a hidden state transition, as described in Section 8.4.4. The difficulty of finding a satisfying condition of the constraint for the hidden state transition only occurs for a few benchmark circuits, with most only requiring a handful of iterations to successfully determine a hidden state transition that does not stem from an unreachable state.

The unobfuscated and obfuscated ISCAS'89 sequential benchmark circuits are characterized for area, power, and performance, with results provided in Table 8.2. The characterization of the ISCAS'89 benchmark circuits is performed on a 180 nm IBM technology node using Synopsys DC compiler. The circuit shown in Fig. 8.5a that executes the hidden state transition is included in the analysis of the overhead in power, area, and performance. However, the overheads due to the tamper proof memory are not considered as is common when analyzing logic locking [88, 89, 101]. The circuits that resulted in the most significant overheads in area and power once obfuscated are the smallest benchmark circuits, which includes s27 and s526. As the benchmark circuits increase in size, the area and power overheads of implementing the sequential locking technique for a majority of the obfuscated circuits is less than

Table 8.3: Additional overhead in area, power, and performance beyond the overheads reported in Table 8.1 for 1 to 5 modified minterms C within the state transition logic. A timing overhead of 0% implies that the performance of the circuit is constrained by the state transition logic and not the correction circuitry.

Benchmark	$C=1$			$C=2$			$C=3$			$C=4$			$C=5$		
	Area (%)	Power (%)	Timing (%)	Area (%)	Power (%)	Timing (%)	Area (%)	Power (%)	Timing (%)	Area (%)	Power (%)	Timing (%)	Area (%)	Power (%)	Timing (%)
s27	26.40	48.22	0.00	57.60	113.67	0.00	85.60	156.32	0.00	128.80	277.56	0.00	142.40	279.89	0.00
s13207	3.66	4.62	0.00	7.35	9.51	0.00	11.02	14.62	0.00	14.67	19.47	0.00	18.59	27.76	0.00
s1423	4.58	3.70	0.00	9.16	7.75	0.00	13.74	11.58	0.00	18.25	16.14	0.00	22.95	20.15	0.00
s1488	3.83	2.81	0.00	7.56	5.10	0.00	11.33	7.70	0.00	15.00	10.77	0.00	19.00	13.22	0.00
s15850	4.37	4.16	0.00	8.75	8.52	0.00	13.44	13.00	0.00	17.95	18.43	0.00	22.75	23.43	0.00
s298	5.58	3.16	0.00	13.95	10.00	0.00	18.45	9.88	0.00	27.90	19.53	0.00	30.69	17.37	0.00
s344	15.62	16.44	0.00	31.64	31.32	0.00	47.46	48.95	0.00	62.89	65.02	0.00	80.47	84.05	0.00
s349	15.62	16.44	0.00	31.64	31.32	0.00	47.46	48.95	0.00	62.89	65.02	0.00	80.47	84.05	0.00
s35932	0.61	7.27	0.00	1.23	15.46	0.00	1.87	23.28	0.00	2.52	33.02	0.00	3.15	44.58	0.00
s382	4.15	3.32	0.00	10.37	10.49	0.00	13.72	10.37	0.00	20.73	20.50	0.00	22.81	18.23	0.00
s38417	0.46	5.83	0.00	0.92	12.25	0.00	1.38	18.09	0.00	1.84	24.20	0.00	2.32	32.26	0.00
s38584	0.75	8.09	0.00	1.51	17.32	0.00	2.31	26.85	0.00	3.09	36.36	0.00	3.87	45.73	0.00
s386	14.08	13.20	0.00	29.11	26.11	0.00	43.66	39.30	0.00	58.45	54.98	0.00	74.88	69.88	0.00
s400	4.09	3.31	0.00	10.22	10.46	0.00	13.52	10.33	0.00	20.44	20.43	0.00	17.61	15.45	0.00
s420	28.88	27.48	0.00	57.60	55.86	0.00	86.64	89.81	0.00	115.19	120.03	0.00	146.91	150.60	19.74
s444	4.09	3.21	0.00	10.22	10.13	0.00	13.52	10.01	0.00	20.44	19.79	0.00	22.48	17.60	0.00
s510	21.93	19.39	0.00	43.73	39.44	0.00	65.78	63.42	0.00	88.21	86.45	0.00	111.53	106.13	0.00
s526	3.52	1.71	0.00	8.80	5.39	0.00	11.64	5.33	0.00	17.59	10.54	0.00	19.35	9.37	0.00
s641	40.26	45.89	0.00	80.65	97.50	0.00	122.06	146.81	0.00	163.48	207.56	0.00	206.71	282.47	0.00
s713	40.26	45.90	0.00	80.65	97.54	0.00	122.06	146.87	0.00	163.48	207.64	0.00	206.71	282.59	0.00
s820	17.55	13.77	0.00	34.99	28.80	0.00	52.65	46.34	0.00	69.99	61.80	0.00	88.08	76.80	0.00
s832	19.52	23.64	0.00	38.92	49.43	0.00	58.55	79.54	0.00	77.83	106.07	0.00	97.95	131.81	0.00
s838	25.98	28.85	0.00	52.04	61.37	0.00	78.77	92.32	3.66	107.49	132.61	10.98	133.39	173.23	15.85
s9234	7.30	9.09	0.00	14.71	19.56	0.00	22.42	29.55	0.00	30.00	42.61	0.00	37.55	51.17	0.00
s953	9.13	12.47	0.00	18.26	26.13	0.00	27.40	39.12	0.00	36.40	54.39	0.00	45.79	67.85	0.00
Average	12.89	14.07	0.00	26.46	30.30	0.00	39.46	44.94	0.15	53.82	65.68	0.44	66.34	80.36	1.42

5%, with most of the large benchmark circuits (i.e. s13207, s15850, and s38584) resulting in power and area overheads of less than 1%. A larger variation in the delay overhead is observed across the benchmark circuits, as the delay is dependent on 1) the depth of the combinational logic of the modified FSM and 2) whether the critical path is included as one of the flipped outputs as described in Section 8.4.5. Rerunning the algorithm and selecting a different hidden state transition, therefore, results in a different overhead in delay, as indicated by the results shown in Fig. 8.13. The variation in the delay due to the selection of the path through the state transition logic is characterized by running the hidden state transition algorithm on the s35932, s526, s15850, and s298 benchmark circuits 100 times.

The additional overhead in area, power, and performance required when the original logic cone is modified to secure against structural attack is listed in Table 8.3. The overheads are analyzed for 1 to 5 cubes C , where the protected cubes are minterms of the given circuit. A lower overhead in power, performance, and area is obtained when using non-minterm protected cubes of the circuit. In addition, the corruption of the outputs increases and attacks such as described in [108] become more difficult to execute. For example, consider the s1423 benchmark circuit, which has 41 primary inputs. The output corruption when a single protected cube is utilized is 9.1E-11%. However, if the protected cube constrains only 20 inputs, the corruption increases to 9.54E-5%, but the resilience to a SAT attack, as given by (8.2), is reduced to an

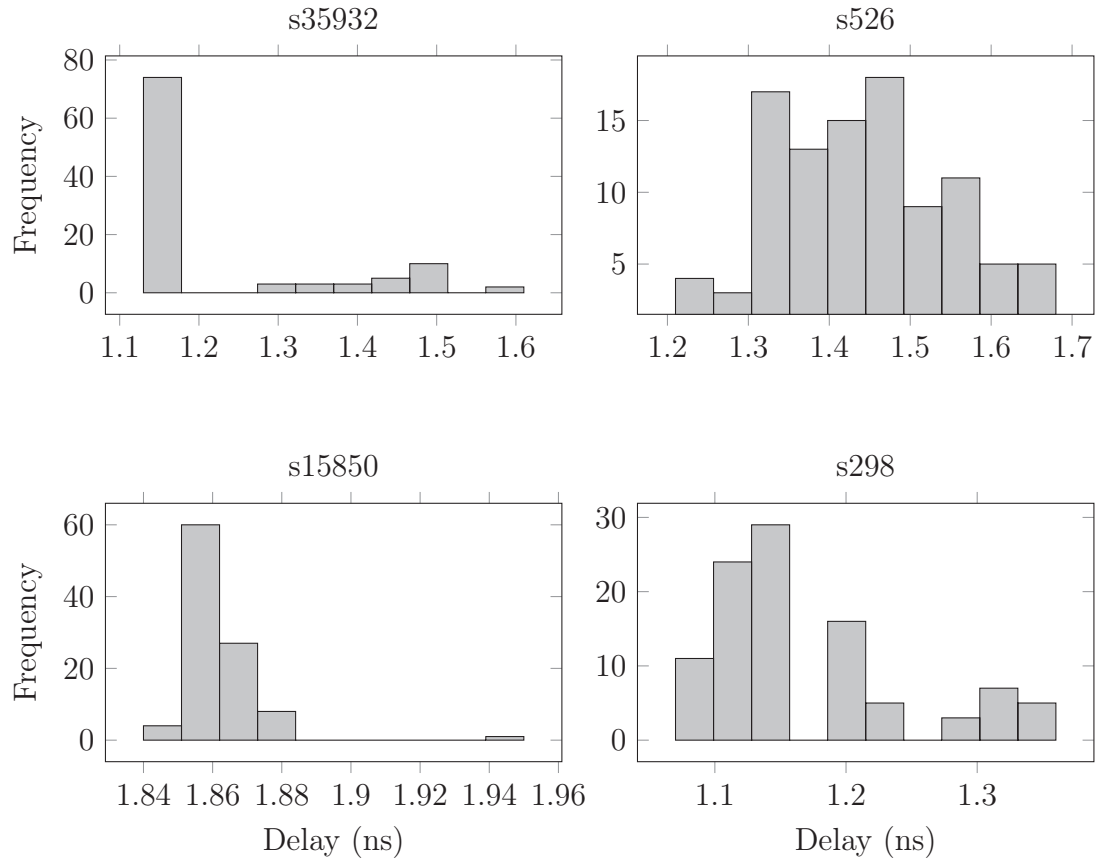


Figure 8.13: Histogram of circuit timing (ns) for 100 instances of the s35932, s526, s15850, and s298 benchmark circuits.

average of $5.24\text{E}+5$ iterations from $1.1\text{E}+12$ iterations. For larger FSMs, the trade-off between output corruption and provided security is acceptable. However, for smaller FSMs, the resulting decrease in the security of the circuit may not be sufficient to protect against the primary attack vectors including SAT.

Implementing a secure partitioned state machine to protect against unauthorized activation and modified logic to protect against reverse engineering forces an adversary to conduct two separate attacks. The adversary must determine the activation

sequence, with a measure of the provided security given by (8.1). In addition, the adversary must utilize an oracle guided attack to determine the modifications to the functional mode FSM, with the measure of the provided security given by (8.2). The analysis of the resilience of all circuits to the SAT attack was conducted on a Xeon E52687W CPU running at 3 GHz with 95 GB of RAM, with a timeout set to 5 days (432,000 seconds). The results of the analysis of the resilience of the secured circuits to the SAT attack are listed in Table 8.4 for the case when two minterm cubes are protected. Over 75% of the executing SAT attacks time out in 5 days, with the exception of the benchmark circuits that have a small number of inputs, as listed in Table 8.1. The results also correlate with the estimated number of required attack iterations given by (8.2), which further demonstrates strong resilience against SAT based attacks. Note that the execution of the SAT attack on s38584 returned the correct key despite a large security expectation as calculated by (8.2). The SAT attack was re-run with 5 different netlist orderings, as described in [148]. However, all but one trial resulted in a decryption of the key sequence as the locked cone has a large fanout but a shallow logic depth, which provides the SAT attack with increased probability of finding the modified logic cubes.

A comparison of the security of circuit obfuscating methodologies against various threats is provided in Table 8.5. The proposed hidden state transition (HST)

Table 8.4: Analysis of execution of the SAT attack with a set timeout of 5 days for the ISCAS'89 sequential benchmark circuits. All benchmark circuits include two protected cubes. The number of iterations and the total number of SAT cubes are determined at the moment a timeout TO occurs.

Benchmark	# of Iterations	# of Cubes	CPU Time(s)
s13207	3.33E+04	7.40E+08	TO
s1423	6.94E+04	3.02E+08	TO
s1488	4.83E+03	2.55E+07	1913.23
s15850	4.47E+04	8.85E+08	TO
s27	6.70E+01	1.28E+04	0.11
s298	1.19E+05	9.82E+07	140143.00
s344	1.30E+05	1.74E+08	278422.00
s349	1.71E+05	2.42E+08	TO
s35932	1.91E+04	1.77E+09	TO
s382	1.46E+05	1.42E+08	TO
s38417	2.16E+04	8.81E+08	TO
s38584	1.97E+04	2.15E+09	309432.00
s386	1.62E+03	2.13E+06	35.50
s400	1.70E+05	1.78E+08	TO
s420	1.36E+05	2.18E+08	TO
s444	1.95E+05	1.93E+08	TO
s510	1.09E+05	2.49E+08	TO
s526	1.38E+05	1.60E+08	TO
s641	1.01E+05	2.08E+08	TO
s713	1.34E+05	2.77E+08	TO
s820	1.33E+05	3.18E+08	TO
s832	1.06E+05	2.55E+08	TO
s838	1.12E+05	3.50E+08	TO
s9234	2.87E+04	6.76E+08	TO

Table 8.5: Comparison of different sequential logic locking methodologies against various attack vectors, including 1) oracle guided attacks, 2) structural attacks that allow for the bypass of the added security features, 3) extraction of the transition from obfuscated to functional mode, and 4) extraction of the combinational state transition logic through reverse engineering. A ✓ means the technique is not vulnerable to the attack vector and a ✗ means the methodology is susceptible to the attack vector.

Obfuscation Methodology	Oracle Guided Attacks	Structural Attacks	Extraction of Obfuscated to Functional Mode Transition	Extraction of State Transition Logic
[115]	✓	✗	✗	✗
[117]	✗	✓	✗	✓
[124]	✗	✓	N/A	✓
[125]	✗	✗	N/A	✗
HST	✓	✓	✓	✗
HST-MOD CONE	✓	✓	✓	✓

algorithm is the only partitioned state machine algorithm that protects against extraction of the obfuscated to functional mode transition, which results in an increase in the number of paths an adversary must search to determine the key sequence that activates the circuit. In addition, HST with modifications to the logic cone protects against extraction of the state transition logic.

Chapter 9: Increased Output Corruption and Structural Attack

Resilience for SAT Attack Secure Logic Locking

While out-of-cone logic locking techniques provide greater resilience to the SAT attack, the techniques discussed in Chapter 4 do not provide the same security guarantees when high corruption of the input-output pairs is required for an incorrect key. The concern with limited output corruption is that an adversary has access to an IC that produces very few incorrect input-output pairs. The rare errors allow an adversary to sell an IC without applying the correct key. An unknowing customer might never realize that the circuit is functioning incorrectly, or may have the error occur during critical circuit functionality. Therefore, the generation of significant input-output pair corruption is of importance when implementing out-of-cone logic techniques, while also maintaining sufficient resilience to the SAT attack and minimizing any overhead in power, performance, and area.

In addition, as discussed in Chapter 4, the circuitry added to modify the original logic cone typically leaks information regarding the key through the structure of the logic. Many of the developed structural attacks allow for the extraction of the key without an oracle IC. While techniques exist to minimize the structural leakage through subtractive measures [112], the overhead in design time to modify the original cone is large.

A novel out-of-cone logic locking methodology is presented in this chapter, which is termed CORruption Adaptable Logic Locking (CORALL). The technique provides improved SAT attack resilience while also increasing the number of corrupted input-output pairs. The chapter begins with an overview of the CORALL methodology and continues with a discussion of the resilience of the technique to both SAT and structural attacks. In addition, a logic cone modification algorithm that relies on logical cofactors is described.

9.1 CORALL

The developed CORALL architecture addresses limitations to current techniques by utilizing an expanded key space that allows for increased output corruption while providing high SAT attack resilience. The expansion is denoted as an increase in the number of unique columns of the truth table representing the key space, which implies more functions for an oracle guided attack to constrain. For example, consider the output corruption tables shown in Fig. 9.1 of the flip function circuitry of a two-input AND gate implemented with SFLL-HD⁰, SFLL-HD¹, and a LUT. The modification of the logic cone when implementing SFLL-HD⁰ on a two-input AND flips the output of the circuit when $A = 0$ and $B = 1$, which results in a modified cone with an output corruption probability of 25%. In comparison, 50% of the cubes, $AB = 00$ and $AB = 11$, are corrupted when implementing the AND with SFLL-HD¹. However,

as discussed in Section 4.2.1, less DIPs are required to execute a successful SAT attack on the obfuscated AND gate, with an average of two DIPs required for SFLL-HD⁰ and only one DIP for SFLL-HD¹. As opposed to the SFLL-HD implementations, the LUT flip function shown in Fig. 9.1c implements up to 2^4 possible functions. Since there are 2^4 possible functions, four iterations of the SAT attack are required to decrypt the key independent of the corruption applied to the modified logic cone. In general, a LUT with n select lines generates 2^{2^n} unique columns. Since each input pattern applied to the LUT generates an equally balanced logic one and logic zero in the key space, as shown in Fig. 9.1c, the key space is cut in half each iteration. The 2^{2^n} unique columns, therefore, force an oracle guided attack to apply 2^n input patterns to determine the correct key.

While a LUT allows for the maximum amount of unique columns, and therefore, the largest number of functions for an adversary to search, the use of a LUT as the flip-function is limited by the significant overhead in power, performance, and area required to implement the circuit. The objective of CORALL is to generate a large number of unique columns in the key space without resulting in a large overhead in power, performance, and area.

The CORALL architecture is described as $\text{CORALL}(n, l, c)$, where n defines a subset of the circuit inputs, l is the number of LUT select lines, and c is the number of protected cubes. The circuit structure when l equals 2 is shown in Fig. 9.2, where

A	B	K0	K1	K2	K3
0	0	X	✓	✓	✓
0	1	X	✓	X	X
1	0	✓	✓	X	✓
1	1	✓	✓	✓	X

(a)

A	B	K0	K1	K2	K3
0	0	X	✓	✓	X
0	1	X	✓	✓	X
1	0	X	✓	✓	X
1	1	X	✓	✓	X

(b)

A	B	K0	K1	K2	K3	K4	K5	K6	K7	K8	K9	K10	K11	K12	K13	K14	K15
0	0	✓	✓	✓	✓	✓	✓	✓	✓	X	X	X	X	X	X	X	X
0	1	✓	✓	✓	✓	X	X	X	X	✓	✓	✓	✓	X	X	X	X
1	0	✓	✓	X	X	✓	✓	X	X	✓	✓	X	X	✓	✓	X	X
1	1	X	✓	X	✓	X	✓	X	✓	X	✓	X	✓	X	✓	X	✓

(c)

A	B	K0	K1	K2	K3	K4	K5	K6	K7	K8	K9
0	0	✓	✓	✓	✓	✓	X	X	✓	X	X
0	1	✓	✓	✓	✓	X	✓	X	X	✓	X
1	0	✓	✓	X	X	✓	✓	✓	✓	X	X
1	1	X	✓	X	✓	X	X	X	✓	X	✓

(d)

Figure 9.1: Truth tables of a two input AND gate locked with a) SFLL-HD⁰ with a correct key of K1, b) SFLL-HD¹ with a correct key of K1 and K2, c) a LUT based correction circuit with a correct key of K1, and d) CORALL $n2/lc1$ with a correct key of K1.

each logical input of the circuit is connected to a select line of a two input LUT that requires four key bits and the output of each LUT is then connected to an AND tree. The structure is easily modified to account for multiple protected cubes, where each output of a protected cube is connected to an OR gate, as shown in Fig. 9.2. In addition, the size of the LUTs is adaptable. For example, setting l to 4 is possible,

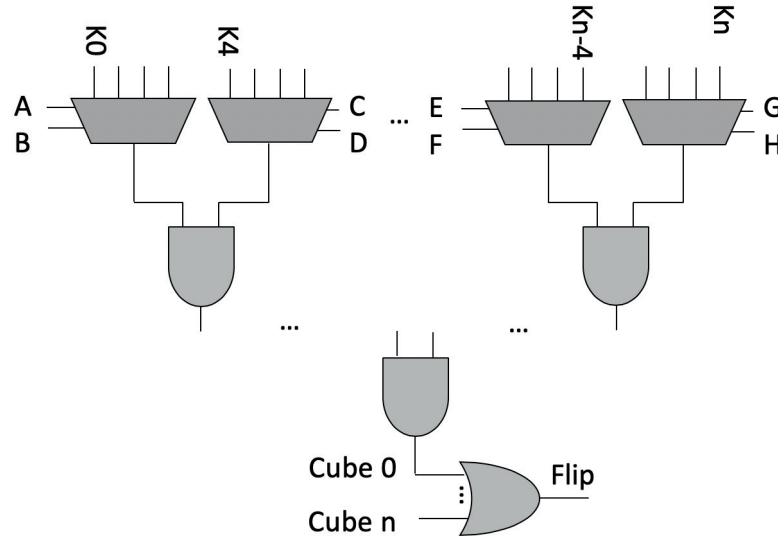


Figure 9.2: Implementation of the flip function circuit of CORALL. The first layer consists of l -input LUTs, where l is two for the example shown, with the inputs to the modified cone provided to the LUTs. The outputs of the LUTs are inputs to an AND tree, which forms a single protected cube. Multiple protected cubes are then provided as inputs to an OR gate, which generates the flip signal to correct any errors in the modified logic cone.

which combines inputs A , B , C , and D into a single LUT instead of utilizing two LUTs to group inputs A and B and inputs C and D when l equals 2. Rather than requiring an exponential increase in the number of keys, as needed for a standard LUT topology, the CORALL architecture only requires a linear increase in the key size. The number of keys required for circuits with 1 to 12 inputs is provided in Fig. 9.3, where a linear increase in the number of keys for an l of 2, 3, 4, 6, and 8 inputs is shown for the CORALL architecture. For example, securing a circuit with inputs A to H ($n = 8$) requires 2^8 key inputs for a standard LUT topology as compared to 16 key bit inputs for CORALL $n8l2c1$, which significantly reduces the

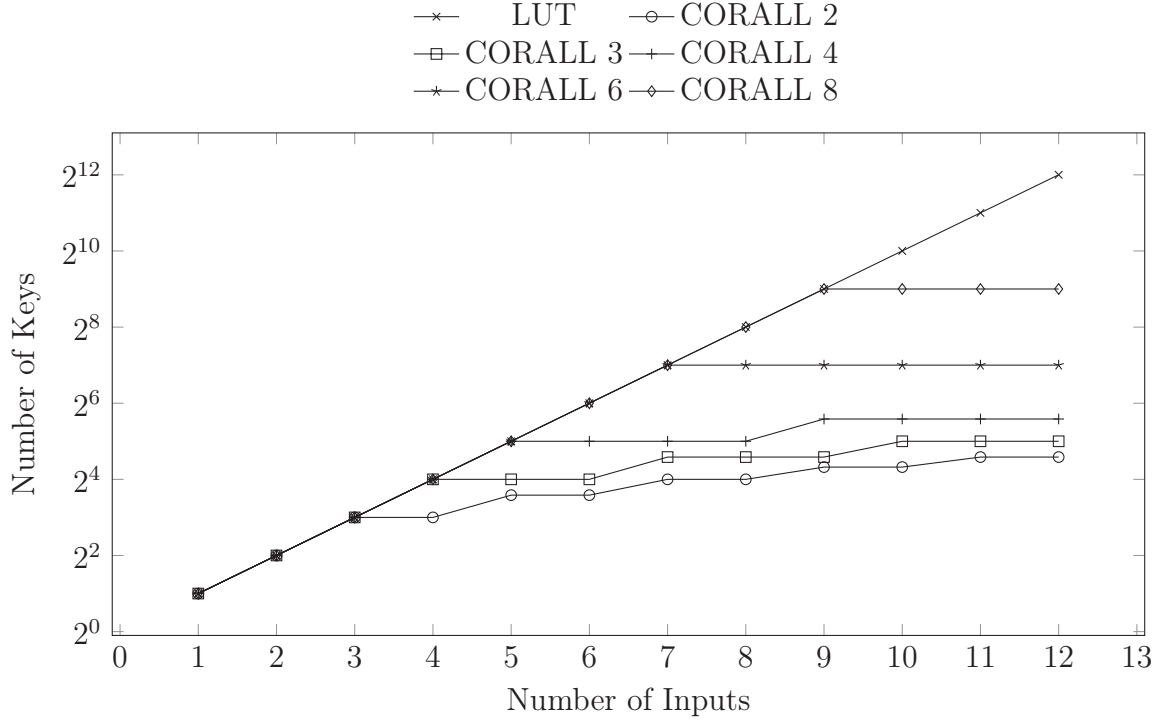


Figure 9.3: Number of keys required for a standard LUT based logic locking topology and the CORALL methodology as the number of primary inputs to the circuit is increased from 1 to 12. The CORALL architecture is evaluated for LUT sizes of 2, 3, 4, 6, and 8 inputs.

area and power overhead when l is less than the total number of inputs to a given logic cone. The number of keys required to implement CORALL is given by (9.1), where $num_lut = \lceil \frac{n}{l} \rceil$.

$$c * \sum_{i=1}^{num_lut} 2^{l_i} \quad (9.1)$$

9.1.1 SAT Attack Resilience of CORALL

When the corruption of the modified logic cone equals $1/2^n$, the resilience of CORALL $n11c1$ to a SAT attack is nearly identical to SFLL-HD⁰ as the functionality of CORALL inherently contains the point flip function produced by SFLL-HD⁰. The only difference is that CORALL results in an increase in the minimum number of iterations required by the SAT attack to return the correct key since guessing the protected cube does not eliminate all but one key value as is the case with SFLL-HD⁰. For example, consider the resulting key space when securing the input pattern $AB = 11$ with CORALL $n211c1$, which is shown in Fig. 9.1d. Even when the DIP $AB = 11$ is selected, multiple correct functions still exist. In the case of CORALL $n211c1$, three DIPs are required to determine the correct functionality of the circuit as the technique generates 10 out of a possible 16 functions. Note that when the correct constraints (check marks) and incorrect constraints (X's), as shown in Fig. 9.1d, are roughly evenly distributed, the number of iterations of the SAT attack is approximated by $\log_2(u)$, where u is the number of unique columns. For the approximation of $\log_2(u)$ to hold, the remaining valid columns for each iteration must also have an approximately even distribution of correct and incorrect output possibilities. If, for example, a DIP of $AB = 10$ was chosen after $AB = 00$ in Fig. 9.1d, there are still four correct and two incorrect possible functions remaining, which implies that the adversary is only able to eliminate two of the six possible key values.

As the amount of output corruption is increased, which is quantified by comparing the outputs of the original and modified logic cones, the number of unique columns in the truth table of the key space dictates the extent to which the CORALL topology is resilient to an oracle guided attack. An increased number of unique columns provides greater resilience to the SAT attack as an added constraint to the SAT solver when a protected cube is chosen as a DIP is the logical AND of the key bits exposed by the LUTs. For example, the circuit shown in Fig. 9.2 with $c = 1$ results in the constraint $K0 \cdot K4 \cdot (Kn - 4) \cdot Kn$ when the DIP input of A to H equals 00001111 produces an output of 1 from the flip function. The extreme case occurs when the modified logic cone is an inversion function of the original logic cone, which results in a constant one at the output of the flip function. In the case where the modified logic cone is completely corrupted and $c = 1$, which implies a single protected cube, the CORALL architecture requires a minimum number of SAT iterations of 2^l , where l is the number of LUT select lines. The minimum number of iterations is a result of revealing a single key bit of each LUT for each determined protected minterm pattern. Increasing l results in a greater minimum number of iterations required to fully execute an oracle guided attack. The greater number of unique columns added to the truth table of the key space by increasing l , therefore, provides an improved resilience to oracle guided attacks, with the number of unique columns when $c = 1$ given by $(2^{2^l} - 1)^{n/l} + 1$.

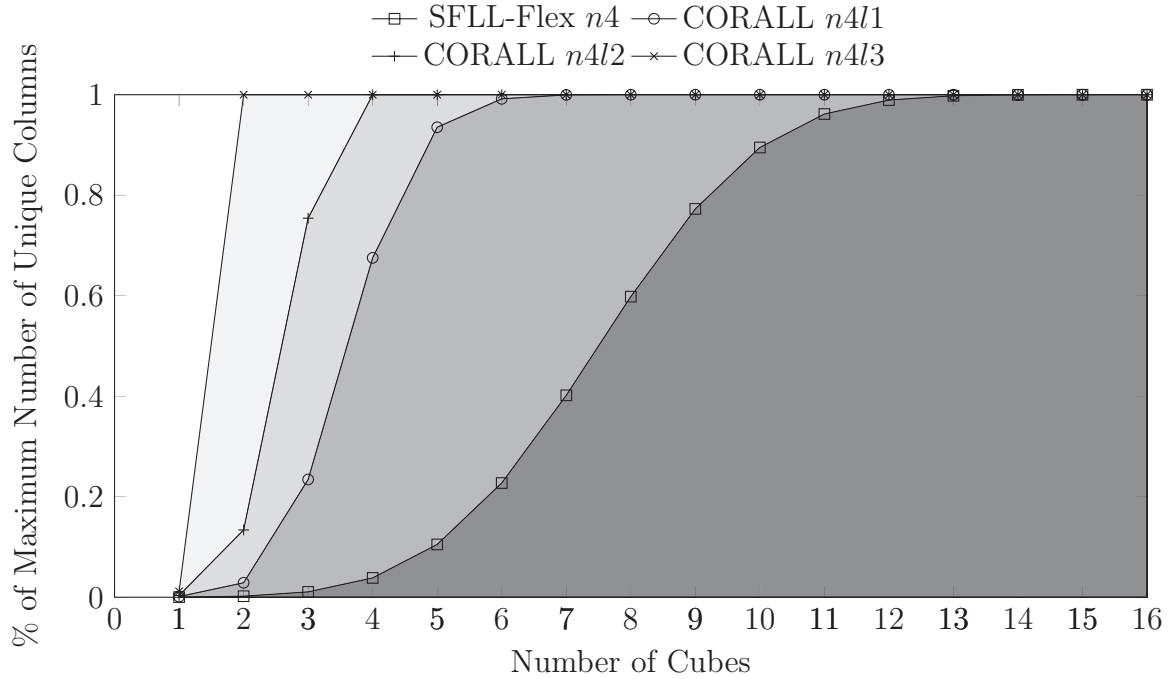


Figure 9.4: Number of unique columns of the key space truth table as the number of protected cubes c is increased for CORALL and SFLF-Flex when $n = 4$. The l size of CORALL is varied from 1 to 3.

The number of unique columns is also increased by providing a greater number of cubes c to the CORALL circuitry. Whereas the number of unique columns for SFLF-Flex is given by $\sum_{i=1}^c \binom{n}{i}$, CORALL generates more unique columns as multiple minterms are flipped per protected cube. The number of unique columns for CORALL and SFLF-Flex when the number of inputs to the flip function equals four is provided in Fig. 9.4. The results shown in Fig. 9.4 indicate that CORALL provides a much faster increase in the number of unique columns than SFLF-flex, generating SAT attack resilience with increased modified logic cone error rates without requiring the large overhead of additional cubes needed with SFLF-flex. Using the notation defined

in [107], the CORALL circuit is able to achieve Best-Possible Approximation-Resilient Locking (BPARL) with regard to oracle guided attacks as the hypothesis space, which equals the number of unique columns, exponentially increases until the space saturates at 2^{2^n} columns. As the required level of output corruption for a given modified logic cone decreases, the value of both the c and l parameters is reduced as the SAT attack resilience of SFLl given by (4.1) applies.

9.1.2 Simulation of benchmark circuits

The SFLl-HD, SFLl-Flex, and the proposed CORALL architectures are compared for 5, 10, 15, and 20 input flip functions implemented for the b17 and b18 ITC'99 [149] benchmark circuits and the s38584 and s35932 ISCAS'89 [150] benchmark circuits, with results provided in Fig. 9.5. The CORALL architecture is evaluated for 2 and 4 protected input cubes c and for 2 and 4 inputs to the LUT l . The number of constrained inputs for the analysis of CORALL is 1, which implies that $n - 1$ inputs are free, where n is the total number of inputs to the flip function. Constraining only a single input per cube results in modified logic cones that produce a large amount of output corruption as compared to an activated IC.

The analysis of SFLl-HD with results as shown in Fig. 9.5 is completed for Hamming distances of $n/4$, $n/3$, and $n/2$ to ensure significant output corruption within the modified logic cone. The analysis of SFLl-Flex for all trials is completed with 2

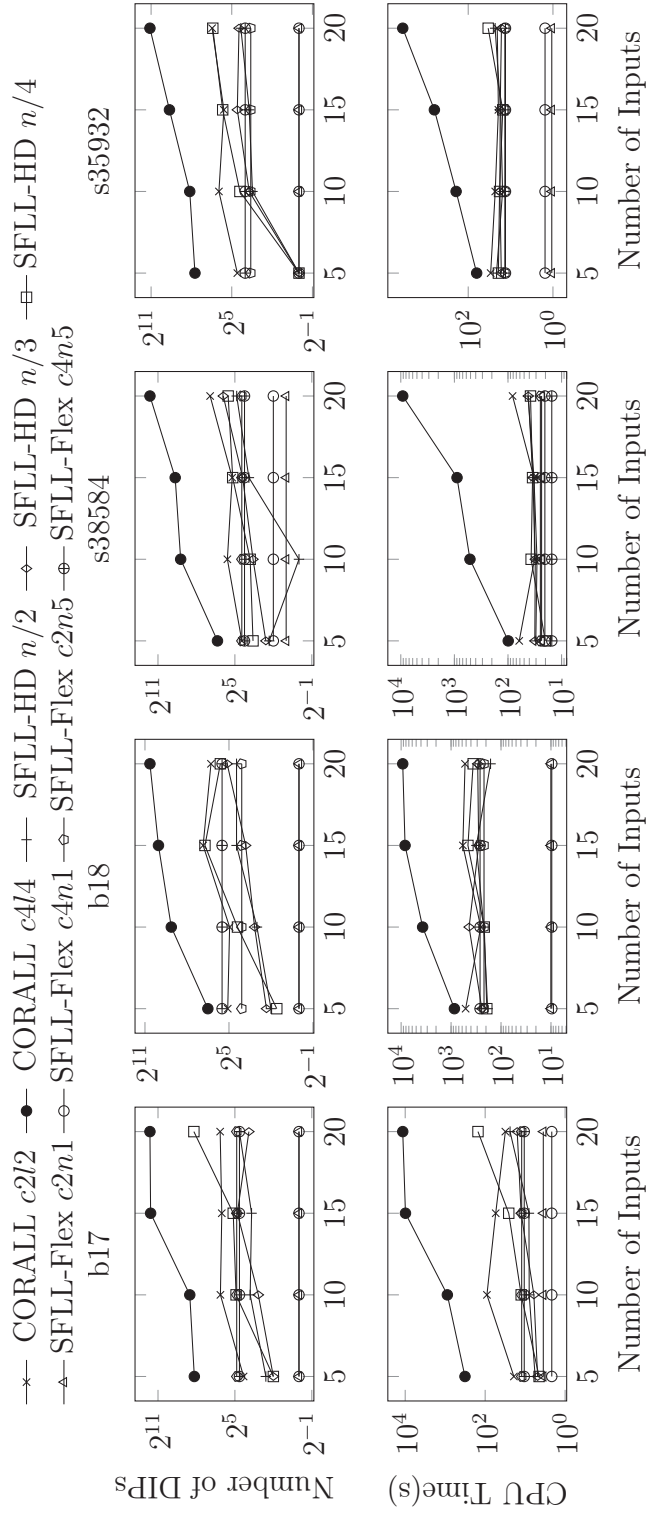


Figure 9.5: Comparison of the SPLL-HD, SPLL-Flex, and proposed CORALL out-of-cone logic locking techniques when the modified logic cone requires a high corruption of outputs as compared to an activated IC. The techniques are evaluated for circuit implementations of the flip function with 5, 10, 15, and 20 inputs. The number of cubes c and LUT-size l are varied for CORALL. The SPLL-HD architecture is evaluated for varying Hamming distances computed based on the number of inputs n . The SPLL-Flex architecture is analyzed for a varying number of cubes c and number of inputs per cube n .

Table 9.1: Output corruption from the modified logic cones of the b17, b18, s38584, and s35932 benchmark circuits. The secured circuits include a 20 input flip function and are evaluated for 1,000 random input patterns, with each random input pattern evaluated for 100 random key values.

Methodology	Benchmark			
	b17	b18	s38584	s35932
CORALL <i>c4l4f19</i>	83.60%	82.90%	80.80%	84.70%
CORALL <i>c2l2f19</i>	76.30%	73.00%	73.60%	74.30%
SFLL HD <i>n/3</i>	6.80%	7.90%	6.70%	6.90%
SFLL HD <i>n/2</i>	29.60%	31.60%	28.60%	30.40%
SFLL HD <i>n/4</i>	3.20%	3.30%	2.50%	3.30%
SFLL Flex <i>c2n5</i>	10.10%	10.60%	13.20%	11.70%
SFLL Flex <i>c4n5</i>	21.70%	20.20%	22.00%	20.70%

and 4 protected cubes c and 1 and 5 inputs n for each cube.

The results shown in Fig. 9.5 confirm that CORALL significantly outperforms SFLL-HD and SFLL-Flex in both the number of iterations of the SAT solver and the CPU execution time required to complete the SAT attack. When implementing a 20 input flip function, CORALL *c4l4*, on average, outperforms SFLL-HD *n/4* by 34.41x and SFLL-Flex by 82.36x in the number of iterations required to execute a successful SAT attack. The execution time of the SAT attack on circuits secured by CORALL is, on average, 111.96x that of SFLL-HD and 430.05x that of SFLL-Flex for a flip function with 20 inputs.

In addition to increasing the number of SAT attack iterations, CORALL also provides greater corruption of outputs as compared to SFLL-HD and SFLL-Flex. The

estimated error rate (output corruption) for the CORALL, SFLL-HD, and SFLL-Flex techniques is analyzed for a 20 input circuit implementation of the flip function with results listed in Table 9.1. The analysis is performed by comparing the outputs of an activated and locked IC for 1,000 random input patterns. For each random input pattern, 100 random key values are applied to the locked IC. The results listed in Table 9.1 indicate that the CORALL architecture produces a minimum corruption of 73% of the total input-output pairs. The maximum output corruption observed for SFLL-HD and SFLL-Flex is, respectively, 31.60% and 21.70%. CORALL, therefore, significantly improves the resilience of a circuit to a SAT attack as compared to both SFLL-HD and SFLL-Flex for cases where a large corruption of the primary outputs is required.

An additional consideration when characterizing the resilience of a circuit to the SAT attack is the variation in the number of iterations required to execute the attack, which is analyzed on the c1908 ISCAS'85 benchmark circuit. The c1908 circuit is evaluated for 500 random netlist orders after applying the SFLL-HD and CORALL techniques with a 10 input circuit of the flip function. Results of the analysis are provided in Fig. 9.6. Varying the order of the netlist results in variation in the number of iterations required to successfully execute the SAT attack [148].

SFLL-HD⁰ and CORALL *c2l2f1* are also characterized for a single free input $f = 1$ as shown in Fig. 9.6, which represents the case where the provided output

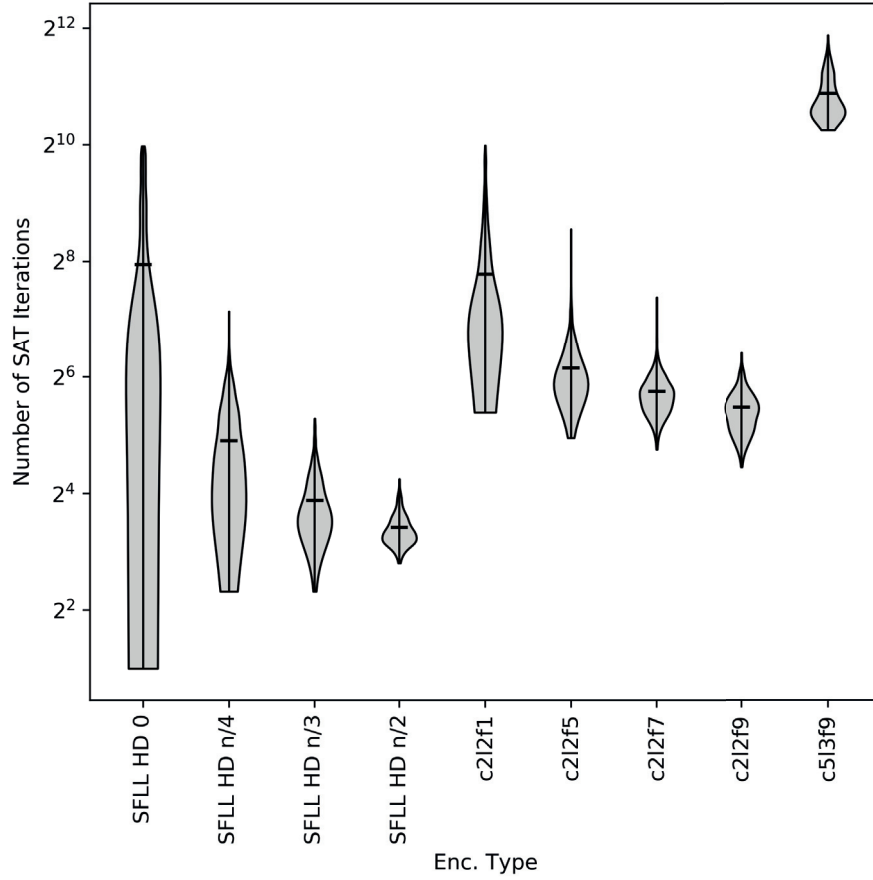


Figure 9.6: Number of SAT attack iterations required to decrypt the ISCAS’85 c1908 benchmark circuit when the circuit is locked with a 10 input flip function using the SFLl-HD and CORALL methodologies. Each secured circuit is evaluated for 500 netlist orders [148] to characterize the variation in the number of iterations. The SFLl-HD technique is evaluated for Hamming distances of 0, $n/4$, $n/3$, and $n/2$, where n is the number of inputs to the circuit of the flip function. The CORALL methodology is evaluated for $c2l1f1$, $c2l2f5$, $c2l2f7$, $c2l2f9$, and $c5l3f9$, where c represents the number of cubes, l the LUT size, and f the number of free inputs.

corruption is limited for each methodology. Note that the free inputs f are a subset of n and represent the inputs a protected cube is independent of. The results shown in Fig. 9.6 indicate that the mean number of iterations produced by SFLl-HD⁰ is

slightly greater than CORALL $c2l2f1$. However, the minimum number of iterations for SFLL-HD⁰ is much lower, and the distribution is uniform across much of the range of the number of iterations required to complete the SAT attack. The uniform nature of SFLL-HD⁰ matches the expected numerical results given by (4.1). A much larger minimum number of iterations is required for CORALL $c2l2f1$, which matches the results from the analysis provided in Section 9.1.1. CORALL, therefore, provides nearly equivalent resilience to the SAT attack as that of SFLL-HD⁰ for cases that require a low output corruption while also requiring a much larger minimum number of iterations of the SAT attack.

As the Hamming distance parameter of SFLL is increased, the mean number of iterations required to execute the SAT attack is reduced significantly as indicated by the results shown in Fig. 9.6. However, the minimum number of iterations needed to successfully execute the SAT attack increases. In contrast, the number of iterations to complete the SAT attack decreases as the number of free inputs increase for CORALL. However, the mean and minimum number of iterations are greater for CORALL than SFLL-HD. Note that the resilience of CORALL to the SAT attack increases as the number of cubes c and the LUT size l increase. Increasing the number of cubes to 5 and the LUT size to 3 while 9 of 10 inputs are unconstrained to produce greater output corruption results in an increase in the number of iterations to over 2^{10} . The trade-off for increasing the number of cubes and LUT size is the additional overhead

in power, performance, and area.

The overhead in power, performance, and area to implement the SFLL-HD and SFLL-Flex techniques with a flip function of 20 inputs is listed in Table 9.2, while the characterization of the overheads for CORALL are listed in Table 9.3. The listed data indicates that for the large ITC'99 (b17 and b18) and ISCAS'89 (s35932 and s38584) benchmark circuits, the implementations of all three techniques result in low overheads, with the power, performance, and area increasing by no more than 10% as compared to the unobfuscated version of the circuit. Applying the three techniques on all of the large benchmark circuits results in no more than a 3% increase in area, with both SFLL-Flex and CORALL providing further reductions in occupied area after re-synthesis of the modified logic cone. Implementations of all three techniques result in large overheads in area for the small ISCAS'85 benchmark circuits, where the c1908 and c432 circuits increase in area by over 100%. On average, a 20% reduction in area and a 2.14% increase in performance was observed as compared to SFLL-HD when implementing the CORALL architecture utilizing four protected cubes c and a LUT size l of four. The largest variation occurs in the timing overhead, as different cones are secured by each technique for each benchmark circuit.

The results listed in Table 9.3 indicate that the implementation of CORALL results in similar overheads in power, performance, and area as SFLL while providing a significantly higher resilience to the SAT attack for circuits that require a large

Table 9.2: Analysis of the power, performance, and area overhead of implementing SFLI-HD and SFLI-Flex on the b17, b18, c432, c1908, s35932, and s38584 benchmark circuits. SFLI-Flex is evaluated for 2 and 4 protected cubes c and a flip function with 5 inputs ($n = 5$). All evaluated circuits are limited to a maximum of 20 inputs to the flip function.

Benchmark	Original			SFLI-HD						SFLI-Flex					
	Area (μm^2)	Power (mW)	Timing (ns)	Area (%)	Power (%)	Timing (%)	Area (%)	Power (%)	Timing (%)	c2n5			c4n5		
b17	2.47E+05	102.81	23.73	1.15	5.15	5.65	-0.62	-1.28	2.44	-0.42	-1.54	2.44	-0.42	-1.54	2.44
b18	7.11E+05	402.93	27.67	0.43	1.89	4.26	-0.06	0.79	2.35	0.05	0.98	2.02	0.05	0.98	2.02
c432	1.71E+03	0.48	3.85	249.01	659.26	24.42	30.77	22.92	20.26	55.82	40.09	13.25	55.82	40.09	13.25
c1908	4.15E+03	2.12	4.64	95.83	145.82	8.41	10.44	10.41	3.88	22.60	17.22	-1.29	22.60	17.22	-1.29
s35932	9.72E+04	38.04	3.04	4.47	11.18	114.47	0.54	1.01	-0.33	1.04	1.75	0.66	1.04	1.75	0.66
s38584	1.18E+05	40.56	5.22	4.25	2.11	24.71	0.93	-7.91	1.53	1.31	-7.76	5.94	1.31	-7.76	5.94

Table 9.3: Analysis of the power, performance, and area overhead of the CORALL architecture for the b17, b18, c432, c1908, s35932, and s38584 benchmark circuits. CORALL is evaluated for 2 and 4 protected input cubes c and LUT sizes of 2 and 4 inputs. Only a single constrained input is utilized for the generation of the CORALL circuits. All evaluated circuits are limited to a maximum of 20 inputs to the flip function.

Benchmark	CORALL					
	c2l2			c4l4		
	Area (%)	Power (%)	Timing (%)	Area (%)	Power (%)	Timing (%)
b17	-2.12	-1.70	3.37	0.14	1.06	7.12
b18	-2.50	-0.59	12.94	-1.30	0.68	9.65
c432	117.80	103.21	7.79	494.95	530.18	23.90
c1908	46.64	36.56	-0.65	203.99	152.38	10.56
s35932	2.33	0.46	-3.95	8.62	8.26	-3.95
s38584	0.81	-6.83	8.05	6.88	8.86	3.64

percentage of corruption between the activated and locked IC. In addition, CORALL provides greater flexibility than SFLL, as the number of cubes and LUT size are adaptable parameters that tune between the security of a circuit, as measured by the number of iterations of the SAT attack, and the overhead in area, power, and performance of implementing CORALL on the circuit. For example, the timing overhead of the c432 benchmark circuit secured with CORALL *c4l4* is 23.90%, as listed in Table 9.3. As a four input LUT contributes to the delay of the flip function, decreasing the LUT size to three inputs reduces the logical depth of the critical path of the circuit implementing the flip function, which reduces the overhead in delay to 0%. If increased resilience to the SAT attack is required, an additional cube is added instead to limit the impact on the timing of the circuit.

9.2 Structural Attack Resilience of CORALL

Increasing the resilience of out-of-cone logic locking techniques to structural attacks requires 1) better mixing of the original logic cone with the perturb function and 2) non-identical modifications to the logic cone that prevent identification and exploitation of the logical structure of the perturb function due to the possession of knowledge of the logic locking algorithm. The first modification to the locking mechanism of the circuit when implementing CORALL is the elimination of the XOR gate in the perturb unit, which reduces the complexity of integrating the perturb function with the original logic cone. To remove the XOR, the algorithm must determine a set of input constraints that result in a constant logic 0 or logic 1 output from the original logic cone. The output of the circuit is now flipped with standard OR and AND gates, depending on the desired output of the modified cone.

$$f_{x_2} = f(x_0, x_1, 1, x_3, \dots, x_n) \quad (9.2)$$

$$f_{\bar{x}_2} = f(x_0, x_1, 0, x_3, \dots, x_n) \quad (9.3)$$

In order to determine the modified logic cubes for CORALL, an approach based on iterative cofactors is applied. For a Boolean function, the positive cofactor of a variable x_2 is given by (9.2), while the negative cofactor is given by (9.3). The positive cofactor sets x_2 to logic 1 and returns the remaining logic function, while the negative cofactor sets x_2 to logic 0 and, similarly, returns the remaining logic function.

Algorithm 14: Protected Cube Generation

```

Input: FSM netlist netlist, LUT Size lut_size,
Common Probability common_prob,
Number of Cubes num_cubes;
curr_func = netlist;
num_cone_inputs = length(curr_func.inputs);
flip_funcs = [];
for num_cubes do
  while num_cone_inputs > lut_size do
    if common_inputs then
      /* Set cf_var, cf_val from common inputs */
    else
      /* Choose random cf_var and cf_val */
      func_cfs = get_cfs(curr_func, cf_var);
      if common_inputs then
        curr_func = func_cfs[cf_val];
      else if num_ins_cf0 < num_ins_cf1 then
        curr_func = func_cfs[0];
      else if num_ins_cf1 < num_ins_cf0 then
        curr_func = func_cfs[1];
      else
        curr_func = func_cfs[cf_val];
      /* Add set input to constraints */
      if rand(0, 1) <= common_prob then
        /* Append to common inputs */
      if num_cone_inputs <= 2 * lut_size then
        /* Check for 2-LUT function criteria */
      end
      flip_funcs.append(curr_func)
    end
  end
  /* Add correction circuitry for flip_funcs */

```

The formulaic description of utilizing the iterative cofactor to determine the modified logic cubes is provided as Algorithm 14. The while loop included in Algorithm 14 is the primary pseudo-code describing the generation of the protected cubes. The algorithm begins by checking if there are previous common inputs to utilize as the

branching variable for the cofactor. If not, a random variable and logic value are generated for the branching of the cofactor. The positive (f_x) and negative ($f_{\bar{x}}$) cofactors are determined for the given cofactor input variable cf_var . If the branching direction is already chosen by the common input, then the respective positive or negative cofactor is selected as the current function $curr_func$. Otherwise, the number of inputs to the positive and negative cofactor functions are considered for the branching direction. The polarity of the cofactor with the smallest support set is chosen as the $curr_func$. Utilizing the cofactor with the smallest support set reduces the number of branches taken, which results in fewer constrained inputs and, therefore, the generation of more input-output pairs with corrupted outputs. If, however, the objective is to lower the output corruption, then the cofactor with the largest support set is selected. If the positive and negative cofactors include an equal number of inputs, then the cofactor polarity is chosen at random. The set variable and the polarity of the given set variable are added to a hash map of the constrained inputs, which are utilized when generating the logic of the perturb unit.

The computational complexity, as given by the runtime of executing Algorithm 14, is bounded by the cofactor operation and the number of iterations of the while loop required to determine the protected cube. In the worst case, the cofactor algorithm needs to update the entire graph structure to a constant value, which is bounded by $O(|V| * |E|)$, where $|V|$ is the set of graph vertices and $|E|$ the set of graph edges.

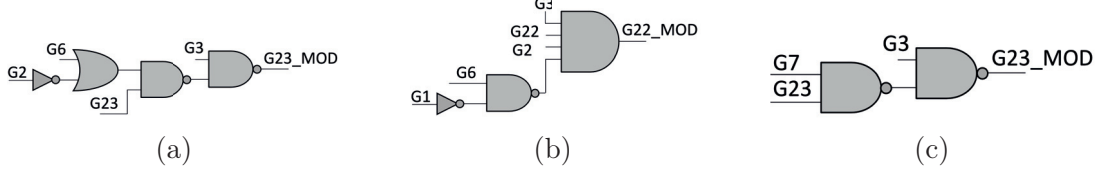


Figure 9.7: Three different perturb functions generated with the developed algorithm that selects protected cubes through iterative cofactors. Note the topological differences between the three generated circuits and the variance in the support set of the functions.

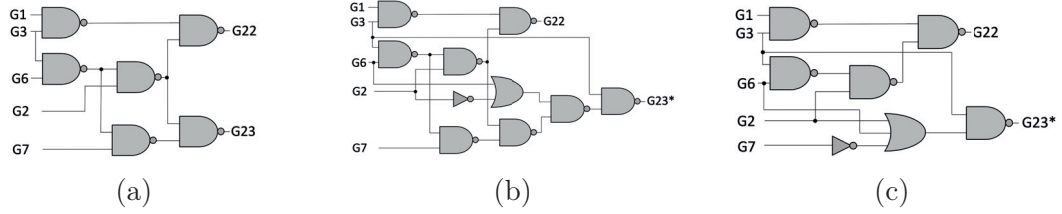


Figure 9.8: Procedural stages of the modifications to the original logic cone to include the logic of the perturb unit shown in Fig. 9.7a. The schematics depict (a) the unmodified c17 circuit, (b) the modified logic cone before re-synthesis, and (c) the modified logic cone after re-synthesis.

However, in most cases, the cofactor operation does not have to update the entire graph. Rather, the cofactor operation typically only updates a small portion of the graph corresponding to an average runtime complexity on the order of $O(\log(|V| * |E|))$. The number of iterations is limited to n , where n is the number of primary inputs to the circuit. The total runtime complexity of the operation is then bounded by $O(n * \log(|V| * |E|))$. Note that the number of iterations of the algorithm and the cofactor complexity are correlated. As a result, a cofactor iteration that significantly modifies the graph also eliminates a subset of the n input variables, which reduces the total number of executed iterations of the cofactor algorithm.

To produce increased variation in the logical structure of the generated perturb

Table 9.4: Output corruption of the modified logic cone for the CORALL iterative cofactor protected cube selection algorithm and SFLL-HD⁰. The number of inputs to the secured cone of the CORALL architecture is provided. The corruption is measured as the number of errors per second, assuming an IC operating at 1 GHz.

Benchmark	Number of Cone Inputs	CORALL Corruption	SFLL-HD ⁰ Corruption
ex5	7	6.51E+08	3.97E+06
apex4	9	4.97E+08	9.80E+05
ex1010	10	9.10E+07	4.89E+05
c1908	33	1.00E+06	5.82E-02
c432	36	4.00E+04	7.28E-03
apex2	36	1.28E+08	7.28E-03
c499	41	1.80E+00	2.27E-04
seq	38	4.80E+08	1.82E-03
c1355	41	1.00E+04	2.27E-04
k2	29	7.00E+04	9.31E-01
c3540	50	4.00E+06	4.44E-07
c880	36	2.20E+07	7.28E-03
dalu	27	1.10E+05	3.73E+00
i9	13	4.10E+07	6.11E+04
i8	17	2.20E+07	3.81E+03
c5315	67	7.00E+06	3.39E-12
i4	47	4.00E+06	3.55E-06
i7	7	2.35E+08	3.97E+06
c7552	80	1.43E+03	4.14E-16
c2670	107	8.40E+07	3.08E-24
des	19	8.60E+07	9.54E+02
b17	150	9.54E+02	2.80E-36
s38584	49	6.00E+07	8.88E-07
s35932	14	5.00E+07	3.05E+04
b18	141	4.77E+02	8.35E-44

unit, a term described as *common probability* is introduced to the algorithm. The common probability is the likelihood that a variable from a previous protected input cube is applied again on another cube, which results in the generation of non-comparator based logical functions and a significantly large amount of variation within the logic of the perturb unit. The resulting variation increases the difficulty of successfully executing a structural attack on the modified cone. Three logical topologies of the perturb unit generated using the iterative cofactor technique on the c17 ISCAS'85 benchmark circuit are shown in Fig. 9.7. The variation in the logic of the perturb unit provides increased difficulty to an adversary examining the netlist to determine the structure of a given perturb unit. The procedural stages of implementing the modifications to the logic cone that integrate the perturb circuit shown in Fig. 9.7a with the ISCAS'85 c17 benchmark circuit are shown in Fig. 9.8. The modified logic cone with no re-synthesis is shown in Fig. 9.8b, which is the worst case structural topology as the additional perturb logic is not well integrated with the original cone. However, the similarity of the logic of the perturb unit to the original logic cone of the circuit increases the difficulty of detecting the perturb unit, even when re-synthesis does not produce an optimal mixing of the perturb logic with the original logic cone. The re-synthesized logic cone of c17 including the perturb unit shown in Fig. 9.7a is depicted in Fig. 9.8c, which demonstrates that re-synthesis of the modified logic cone results in a logical topology that is more difficult to sub-divide and extract.

In addition, for each iteration of the while loop, a check is performed to determine if the number of remaining inputs to the cube-generation function is less than $2 * lut_size$. If the number of inputs is less than $2 * lut_size$, the algorithm explores the feasibility of mapping the remaining logical function to two LUTs connected with an AND gate. The remaining inputs must allow for independent groupings of literals. As an example, for a function with literals $ABCD$, one possible grouping of the literals is AB and CD , which are then mapped to two 2-input LUTs. For the standard CORALL architecture, independent groups with last level functions of AND/NOR are searched for first, as AND/NORs are readily mapped to two LUTs followed by an AND gate. If more flexibility in the logical topology of the circuit is needed, the first level of AND gates connected to the LUTs, as shown in Fig. 9.2, are converted to AND/OR gates [143]. The conversion to AND/OR gates allows for the use of AND, OR, NAND, and NOR gates as last level logic.

If the number of remaining inputs to the cube-generation function is greater than $2 * lut_size$, then the algorithm continues to iterate until the remaining cube-generation function fits into a single LUT, or the remaining function is a constant logic 0 or logic 1. The generated function is then added to a list of flip functions (*flip_funcs*) and the algorithm is repeated until the target number of cubes is reached. By utilizing a LUT to capture some of the logic of the perturb unit, even if modifications to the logic cone are identifiable, the adversary does not possess the

necessary information required to determine the key.

The iterative cofactor is utilized until the logic of the protected cube is mapped to a LUT, is a constant logic 0, or is a constant logic 1 to ensure that the generated perturb function is of similar logical depth to the original logic cone. The goal is to avoid large differences in the depth of logic between the original circuit and the perturb unit. For example, if the logic of the perturb unit is only reliant on a single input, then the output corruption is high but the logic of the perturb unit is discernible from the original logic cone. In that case, the perturb unit does not integrate with the original logic cone during re-synthesis as the logical path only has a depth of one. In contrast, if all inputs to the logical cone are constrained within the perturb unit, the adversary is able to isolate the perturb unit by the logical structure of the circuit. While applying the iterative cofactor of the logical cone reduces the disparity in the logical depth between the logic of the perturb unit and the logic of the original cone, additional constraints on the inputs reduce the corruption of the outputs of the circuit. For an IC operating at 1 GHz, an error rate of one corrupted output every second deems a device unusable. The corruption of the outputs of the benchmark circuits modified using the iterative cofactor based algorithm is listed in Table 9.4, where all benchmark circuits secured with CORALL produced more than $4.77\text{E}+02$ errors per second when locked. The error rate for SFLL-HD⁰ is much lower, which provides an adversary with a much more functional IC when an incorrect key is applied. CORALL

Table 9.5: Comparison of the resilience of SFLL-HD, SFLL-Flex, and CORALL to the structural attacks proposed in [108–110, 151]. A ✓ denotes the technique is resilient to the attack vector and a ✗ denotes the methodology is susceptible to the attack vector. A ✓* indicates the topology is partially resilient to the attack vector.

Attack Vector	Logic Locking Methodology		
	SFLL-HD [101]	SFLL-Flex [101]	CORALL
[108] Unateness	✗	✓*	✓
[108] 2H	✗	✓	✓
[109]	✗	✓	✓
[110]	✗	✓*	✓

allows for additional corruption by adding more protected cubes and/or increasing the size of the LUTs. However, both increasing the LUT size and/or the number of cubes results in an increase in the power, performance, and area overheads. CORALL is also capable of utilizing cubes with less constraints, which generate greater output corruption. The trade-off is increased risk to structural attack if the re-synthesis of the circuit does not integrate the perturb unit into the original logic cone.

By 1) replacing the XOR gate as the output of the perturb unit, 2) generating variation in the structure of the perturb logic, and 3) placing the logic of the perturb unit in the top level LUTs of the CORALL architecture, CORALL protects against all current structural attacks. The unateness check described in [108] no longer applies as the perturb unit is not required to utilize all inputs of the logic cone. The properties of the Hamming distance checker exploited in [108] and [109] also no longer apply as the perturb unit does not utilize a Hamming distance checker and does not have a set logical structure. The mixing of multiple correlated comparators of varying lengths by

Table 9.6: SAT attack analysis of CORALL implemented with two protected cubes and a LUT size of two inputs. A TO designates a timeout, which was set to five days (432,000 seconds) for the analysis. The number of needed key bits is determined by utilizing (9.1) and the number of inputs to the logic cone from Table 9.4.

Benchmark	# of Keys	# of Iterations	# of Cubes	CPU Time(s)
apex2	144	1.38E+05	3.67E+08	TO
apex4	34	7.00E+02	1.52E+07	146.992
b17	586	5.69E+04	1.20E+09	TO
b18	688	4.27E+04	1.92E+09	TO
c1355	162	7.51E+04	3.52E+08	TO
c1908	130	7.96E+04	3.21E+08	TO
c2670	426	4.17E+04	4.02E+08	TO
c3540	200	7.30E+04	6.31E+08	TO
c432	144	1.02E+05	1.93E+08	TO
c499	162	9.66E+04	4.51E+08	TO
c5315	266	4.07E+04	6.15E+08	TO
c7552	320	4.73E+04	8.28E+08	TO
c880	144	9.11E+04	3.43E+08	TO
dalv	106	5.36E+04	4.83E+08	TO
des	74	3.76E+04	1.24E+09	TO
ex1010	40	6.62E+02	1.37E+07	69.1274
ex5	26	1.86E+02	6.19E+05	1.50104
i4	186	1.20E+05	3.23E+08	TO
i7	26	6.20E+01	3.20E+05	1.0908
i8	66	6.33E+04	5.98E+08	TO
i9	50	1.09E+04	6.77E+07	8144.68
k2	114	7.69E+04	6.42E+08	TO
s35932	56	1.78E+04	1.66E+09	TO
s38584	194	1.94E+04	2.12E+09	TO
seq	152	5.07E+04	7.19E+08	TO

Table 9.7: Overhead in area, power, and performance of implementing CORALL on a subset of ISCAS'85, MCNC, ISCAS'89, and ITC'99 benchmark circuits when utilizing the algorithm that selects protected cubes through iterative cofactors. The number of inputs and the number of gates are provided for each benchmark circuit. The runtime of the iterative cofactor algorithm is also listed for each benchmark circuit.

Benchmark	Number of Inputs	Number of Gates	Original			CORALL				
			Area (μm^2)	Power (mW)	Timing (ns)	Area (%)	Power (%)	Timing (%)	Runtime (s)	Runtime (s)
i4	192	120	2.03E+03	0.29	1.19	347.04	567.22	62.18		0.01
c432	36	160	1.71E+03	0.48	3.85	319.12	259.87	22.86		0.11
c499	41	202	3.90E+03	2.42	2.78	172.49	79.58	20.14		0.03
ex5	8	256	4.64E+03	1.30	1.97	20.03	19.09	3.55		0.02
c880	60	383	3.59E+03	0.89	4.49	42.08	50.17	-4.68		0.05
apex4	9	438	3.00E+04	11.46	4.08	-1.74	-0.25	-0.74		0.07
i7	199	471	5.08E+03	1.53	1.33	25.98	13.50	34.59		0.03
i9	88	522	5.61E+03	1.83	2.61	29.83	20.24	7.66		0.04
c1355	41	546	3.90E+03	2.43	2.78	171.14	99.92	22.30		9.68
ex1010	10	810	2.94E+04	11.11	3.63	-1.24	-0.54	-2.20		0.05
c1908	33	880	4.15E+03	2.12	4.64	118.78	83.43	29.74		4.96
apex2	39	1035	2.78E+03	0.59	2.84	182.70	208.16	-1.41		0.01
c2670	233	1193	5.79E+03	2.44	3.79	28.22	7.93	-2.37		16.46
k2	45	1201	1.04E+04	1.54	4.07	29.93	58.78	1.23		0.07
seq	41	1459	1.87E+04	5.59	3.38	21.34	23.41	16.86		0.05
c3540	50	1669	1.01E+04	4.42	6.63	53.99	29.58	-11.46		6.90
dalv	75	1697	7.49E+03	1.67	5.51	74.81	81.02	-29.40		0.26
i8	133	1831	8.55E+03	1.78	3.01	22.76	34.84	11.63		0.01
c5315	178	2307	1.37E+04	7.13	4.70	55.42	30.70	4.89		3.80
c7552	207	3512	1.67E+04	8.48	8.62	13.54	13.70	4.18		0.02
des	256	4000	3.48E+04	19.03	3.76	2.26	0.91	4.26		0.01
s38584	1464	11448	1.18E+05	40.56	5.22	0.23	-0.02	-2.11		6.56
s35932	1763	12204	9.72E+04	38.04	3.04	-2.84	-6.05	-20.07		5.91
b17	1452	30777	2.47E+05	102.81	23.73	-0.79	-1.43	-0.17		66.75
b18	3357	111241	7.11E+05	402.93	27.67	-0.29	1.24	4.92		20.86

executing CORALL results in the generation of logic structures similar to the original logic cone, which prevents the execution of the functional reverse engineering attack described in [110]. Even if a comparator is utilized, the support set of the comparator is unknown, which forces the BSIM tool to examine a majority of the logic within the original cone of the IC. With CORALL, removing the topological distinction between the original logic cone and the perturb logic significantly increases the computational cost of a BSIM attack. A summary of the vulnerabilities of SFLI-HD, SFLI-Flex, and CORALL to structural attacks is provided in Table 9.5. While the SFLI-Flex technique is not described as providing cubes of variable length, the flip function does allow for modifications to the size of cubes. Therefore, SFLI-Flex offers partial protection against the unateness attack described in [108] and the functional reverse engineering attack described in [110]. Full protection against both attacks is not possible as the circuit implementing the flip function reveals the variable length of the cubes, which permits the unateness and functional reverse engineering attacks to search a smaller subset of the original logic cone.

The primary challenge in performing a structural attack on CORALL is verifying that the original cone has been restored. Consider the worst case scenario where the added logic is found within the netlist, which is equivalent to extracting the added logic from Fig. 9.8b. While other techniques produce a static structural signature of the perturb unit, CORALL yields no such distinct signature. Without a set structural

signature, the only means for an adversary to verify that the original logic cone has been extracted is to apply an oracle guided attack.

9.2.1 Results of Simulation

The analysis of circuit resilience against the SAT attack is performed for two protected cubes and a LUT size of two inputs when utilizing the developed algorithm that selects protected cubes using iterative cofactors, with results as listed in Table 9.6. The SAT attack was performed on a Xeon E52687W CPU running at 3 GHz and with 95 GB of RAM. A timeout of 5 days (432,000 seconds) was set for all circuits when executing the SAT attack. The results listed in Table 9.6 further support the expectation that larger cone sizes, as listed in Table 9.4 for each benchmark circuit, provide greater resilience to the SAT attack as discussed in Section 9.1.1. In addition, the output corruption is lower for the algorithm that selects protected cubes using iterative cofactors as compared to the results shown in Table 9.1, which also increases the resilience against the SAT attack as the probability of generating a DIP that causes the flip function to evaluate to logic 1 on the activated IC decreases. The circuits that the SAT attack decrypted the key within 5 days include a small number of inputs to the logic cone, such as the ex1010 benchmark circuit that contains only 10 inputs. Other than the four circuits that were expected to fail due to smaller size, all other circuits timed out after 5 days of executing the SAT attack.

In order to increase the resilience of a circuit to the SAT attack when the number of inputs to a single cone is small but the total number of inputs to the circuit is much larger, as is the case for the i9 benchmark circuit that includes 88 total inputs, securing multiple cones with a single implementation of the CORALL circuit is possible by applying protected cubes from both cones. For merged cones, the logic that interfaces between the two or more cones is prioritized. Combining cones allows for greater resilience against the SAT attack. For example, the key for i7 was determined in 1.09 seconds when executing the SAT attack as the cone included only 7 inputs. However, a larger number of inputs to the flip function is generated by combining three of the smaller cones of the i7 benchmark circuit, which results in a cone with 11 total inputs. The execution time of the SAT attack increases to 192.48 seconds from 1.09 seconds, and the number of iterations of the SAT solver increases to 2048 from 32. The cost of combining the multiple cubes is an increase in the logic of the flip function, which is a trade-off that must be considered.

The resilience against the SAT attack is also improved by increasing the number of cubes and the LUT size of the CORALL architecture, as discussed in Section 9.1.1. The cost of increasing the number of cubes and/or the LUT size is an increase in the power consumption and area of the circuit as well as a potential drop in performance.

The overhead in power, performance, and area (PPA) to secure the ISCAS'85, MCNC, and ISCAS'89 benchmark circuits by CORALL with two protected cubes

and a LUT size of two inputs is provided in Table 9.7. The analysis of the benchmark circuits indicates that the overhead in power, performance, and area of a given topology is highly dependent on the original size of the circuit being secured. For example, the i4 circuit includes a total of 120 gates, which results in a greater percentage of the total gates added to implement the CORALL architecture and a corresponding increase in area of 347.04%. However, for the s38584 circuit that includes 11,448 gates, the overhead in area is only 0.23%. The size of the circuit must, therefore, be considered when choosing a logic locking implementation. The CORALL architecture provides a similar overhead in PPA to SFLL-HD as indicated by the results listed in Table 9.2.

In addition, certain benchmark circuits including apex4 and ex1010 resulted in reductions in the area and/or power consumption and improvements in the performance over the original unobfuscated circuit. The improvements are due to a significant modification to the original logic cone and indicate a strong integration of the logic of the perturb unit and the original logic of the circuit.

The runtime of executing the algorithm that generates protected cubes through iterative cofactors for each benchmark circuit is also listed in Table 9.7. All of the benchmarks complete within approximately one minute or less, with most of the benchmarks requiring less than one second. The low runtimes demonstrate the usability of the algorithm to quickly secure a circuit netlist.

Chapter 10: Conclusions

The expanding utilization of untrusted third parties within the design and manufacturing flow raises concern over the security of underlying hardware. Ensuring a secure computing stack requires that the hardware layer provides a secure foundation. The work in this dissertation furthers the design of secure hardware by 1) developing low-cost logic locking solutions, 2) incorporating logic locking into the electronic design automation (EDA) flow, and 3) improving the security of logic locking against oracle guided attacks, where such attacks utilize an activated IC with the correct key already applied to generate input-output constraints that are used to efficiently determine the logic locking key.

Low cost in-cone logic locking techniques are developed in Chapter 6 that utilize novel logic structures that provide increased performance, reduced power consumption, and reduced area as compared to XOR or look-up table (LUT) based in-cone logic locking methodologies. For a locked AND gate, a reduction in power of 41.50%, an estimated reduction in area of 43.58%, and an increase in performance of 34.54% is achieved when applying the developed gate level logic locking technique instead of the LUT-based locking technique described in Chapter 2. The developed logical structures reduce the cost of implementing in-cone logic locking, but are developed for a threat model that does not consider oracle guided attacks.

When oracle guided attacks are considered as part of the threat model, the security of in-cone logic locking techniques is significantly reduced. The security of in-cone logic locking against oracle guided attacks is, therefore, characterized through analysis of SAT solver parameters and logic structure in Chapter 7. The results from the characterization are then utilized to develop three gate selection algorithms for logic locking based on maximum fanout free cones (MFFCs). The use of MFFCs resulted in an increase of the average number of iterations to complete an oracle guided attack by 61.8% when securing 5% of the logic gates of a netlist. The results demonstrate that while the security of in-cone logic locking techniques is improved, oracle guided attacks still prove effective due to the logical masking of key gates within the circuit.

Two methodologies are then developed to increase the resilience of a circuit to oracle guided attacks. The first methodology, presented in Chapter 8, utilizes sequential logic locking to partition the state space into obfuscated and activated modes. As the oracle IC only reveals information on the activated state space, the oracle provides limited information to utilize in the extraction of the activation sequence of the circuit. Hidden state transitions are developed to limit the logical and structural information of a reverse engineered IC, which increases the expected time to determine the key to 1.64×10^{12} years for the ISCAS'89 s15850 benchmark circuit from an initial maximum decryption time of 48 hours [152]. Algorithms to modify the original state machine and insert hidden state transitions are also developed, which,

when executed, result in an average overhead of 6.05% in area, 10.30% in power, and 8.97% in performance across all of the ISCAS'89 sequential benchmark circuits.

The second methodology to increase security against oracle guided attacks is presented in Chapter 9. The developed CORALL technique increases the output corruption of out-of-cone logic locking techniques while maintaining a high level of security against oracle guided attacks. The CORALL architecture results in an increase in the number of iterations required to complete a SAT attack for a 20 input circuit implementing a flip function with high input-output pair corruption by 34.41x over SFLL-HD $n/4$ and 82.36x over SFLL-Flex. Even while significantly increasing the security of a circuit when high output corruption is required, the developed technique provides a 20% reduction in area and a 2.14% increase in performance on average as compared to the state of the art. In addition, an algorithm based on logical cofactors is developed to reduce the structural signatures of the implemented logic cone modifications, which results in an increase in the security of out-of-cone logic locking methodologies. The runtime to secure the ISCAS'89 sequential benchmark circuits with CORALL was 66 seconds at the maximum, which demonstrates that the developed techniques provide a significant opportunity to secure hardware without significant cost in computational resources.

Overall, the work presented in this dissertation provides enhanced logic locking security while also minimizing the effect on power, performance, and area as compared

to an unobfuscated IC. The developed algorithms to automate the implementation of the techniques on an unobfuscated IC minimize the cost in design time and allow for increased access to secure hardware. Therefore, the presented work allows for the addition of security as a design objective along with power, performance, and area to further secure the cyber landscape.

Chapter 11: Future Work

The hardware security techniques developed in this dissertation provide increased security of the hardware layer without drastically increasing the overhead in power, performance, area, or design time. There are, however, research directions that have the potential to further secure the circuit landscape, and provide a broader utilization of secure ICs. Future research directions discussed in this chapter include 1) the evaluation of the security provided by obfuscating techniques, 2) validation of tamper proof memory, and 3) subtractive logic cone modification algorithms.

11.1 Evaluation of the Security of Logic Obfuscation Methodologies

The evaluation of techniques to secure hardware largely relies on empirical validation. For example, the measure of the security provided against the SAT attack typically involves executing the attack with a single implementation of the technique across multiple benchmark circuits. However, a variety of parameters, including the implementation of the SAT solver, effect the number of DIPs required to determine the key of a locked IC, as demonstrated in Chapter 7.

The development of mathematical estimates of the number of DIPs to execute a SAT attack described in [101] is a positive step towards reducing the reliance on empirical measures. However, the quantified measure described is *average* security.

The distribution of SFL- HD^0 is uniform, with the key just as likely to be determined in the first iteration as in the 2^n iteration, where n is the number of inputs. The expected security of $2^n/2$ iterations might not be acceptable for some applications that require a guarantee of the minimum number of iterations.

The analysis described in Chapter 9 provides an estimate of the distributions to determine the correct key as a measure of the expected security when implementing CORALL and SFL. However, the evaluation is limited to small circuits as the number of possible evaluation paths is large (32,768 possible paths for a circuit with four inputs). Future work estimating the distribution of determining the key after applying a given security technique must provide efficient and flexible definitions of security for a variety of use cases. Work such as described in [153] moves towards determining probability distributions and evaluating methods under exhaustive exploration at a small scale to develop closed form equations of the security provided by a given technique. However, complex logic spaces like CORALL become difficult to evaluate and to develop closed form equations for. Research directions into sampling the attack space and evaluating the security provided with less reliance on empirical evidence are then vital to moving logic locking towards wide spread use.

11.2 Validation of the Security of Tamper Proof Memory

All logic locking methodologies rely on memory storage for the key. The key is stored in either off-chip or on-chip memory, with different threats to consider for each methodology. Since bringing the key from an off-chip source to an on-chip location is potentially vulnerable to attack, many techniques assume a tamper proof on-chip memory to store the key. However, the design and validation of the security of the tamper proof memory is not provided.

Future research on the ability to securely store a key on-chip and withstand a variety of key retrieval attack vectors is vital for logic locking to progress towards commercial and military use. For example, the work described in [154] highlights probing attacks on registers and minimal mask modification attacks to extract the key value. Such attacks bypass any provable security provided against oracle or structural attacks. Research into tamper proof memory compilers that allow for the generation and use of tamper proof memory for logic locking is, therefore, vital to provide the assurance and protection against the wide-variety of threats currently possible. The development of secure memory requires significant physical validation as well as modeling in the logical domain to ensure the value of the key is not propagated to any accessible locations within the IC.

11.3 Efficient Subtractive Logic Cone Modification Algorithms

The research presented in Chapter 9 reduces structural signatures of additive logic cone modifications by utilizing an iterative cofactor algorithm. While the technique provides increased security against structural attacks, no formal guarantees exist that the differences between the perturb unit and the original logic cone do not allow an adversary to determine the key.

Measures of subtractive logic cone modifications, such as described in [112], provide an increased level of confidence that minimal distinctions exist between the original logic cone and the perturb unit. However, utilizing subtractive modification algorithms currently requires significant runtime to determine an optimal logic net to modify, with stuck-at-faults utilized in [112]. The long runtime is correlated with the limited nets within the IC that produce a small enough set of errors to fit into an SFL-Flex restore unit, which biases the algorithm towards a small percentage of nets within the circuit. Research into techniques that combine the benefits of the cofactor approach with techniques implementing subtractive modifications will further increase the security of the circuit while limiting the computational cost.

Bibliography

- [1] M. Corporation, “Common Weakness Enumeration,” https://cwe.mitre.org/top25/archive/2019/2019_cwe_top25.html, September 2019.
- [2] C. Weiner, “How the transistor emerged,” *IEEE Spectrum*, Vol. 10, No. 1, pp. 24-33, Jan 1973.
- [3] W. F. Brinkman, D. E. Haggan, and W. W. Troutman, “A History of the Invention of the Transistor and Where it Will Lead Us,” *IEEE Journal of Solid-State Circuits*, Vol. 32, No. 12, pp. 1858-1865, Dec 1997.
- [4] J. Bardeen and W. H. Brattain, “The Transistor, a Semi-Conductor Triode,” *Physical Review*, Vol. 74, pp. 230–231, Jul 1948.
- [5] W. Shockley, “The Theory of P-N Junctions in Semiconductors and P-N Junction Transistors,” *Bell System Technical Journal*, Vol. 28, No. 3, pp. 435-489, 1949.
- [6] C. Pirtle, *Engineering the World: Stories from the First 75 Years of Texas Instruments*, Ph.D. Dissertation, Southern Methodist University, Dallas, Texas, 2005.
- [7] T. R. Reid, *The Chip: How Two Americans Invented the Microchip and Launched a Revolution*. Random House, 2001.
- [8] L. Berlin, *Man Behind the Microchip: Robert Noyce and the Invention of Silicon Valley*. Oxford University Press, 2005.
- [9] N. Noyce, “Microelectronics,” *Scientific American*, Vol. 237, No. 3, pp. 62-69, September 1977.
- [10] E. C. Hall, “Integrated Circuits in the Apollo Guidance Computer,” pp. 1-24, 1963.
- [11] E. C. Hall, “A Case History of the AGC Integrated Logic Circuit,” *Apollo The International Magazine Of Art And Antiques*, pp. 1-40, 1965.

- [12] L. D. Hanley, J. Partridge, and E. C. Hall, "The Application of Failure Analysis in Procuring and Screening of Integrated Circuits," Massachusetts Institute of Technology, Technical Report, October 1965.
- [13] *Intel 4004 Image*, The Register. [Online]. Available: http://regmedia.co.uk/2011/11/14/4004_die_large.jpg
- [14] *Intel 8008 Image*, The Register. [Online]. Available: http://regmedia.co.uk/2011/11/14/8008_die_large.jpg
- [15] *Intel 80386 Image*, The Register. [Online]. Available: http://regmedia.co.uk/2011/11/14/80386_die_large.jpg
- [16] *Intel Pentium IV Willamette Image*, Hardware Secrets. [Online]. Available: https://www.hardwaresecrets.com/all-pentium-4-models/die_willamette1/
- [17] *Intel Hexa Core Coffee Lake Die*, WikiChip. [Online]. Available: [https://en.wikichip.org/wiki/File:coffee_lake_die_\(hexa_core\).png](https://en.wikichip.org/wiki/File:coffee_lake_die_(hexa_core).png)
- [18] G. Moore, "The Future of Integrated Electronics," Fairchild Semiconductor Internal Publication, 1964.
- [19] G. Moore, "Cramming More Components onto Integrated Circuits," *Electronics Magazine*, Vol. 38, No. 8, pp. 114-117, April 1965.
- [20] G. Moore, "Cramming More Components onto Integrated Circuits," *Proceedings of the IEEE*, Vol. 86, No. 1, pp. 82-85, January 1998.
- [21] G. E. Moore, *Progress in Digital Integrated Electronics*, 2006, Chapter 6, pp. 61-66.
- [22] *4004: Single Chip 4-Bit P-Channel Microprocessor*, Intel Corporation, March 1971.
- [23] *4040: Single Chip 4-Bit P-Channel Microprocessor*, Intel Corporation, March 1974.
- [24] *8008: 8-Bit Parallel Central Processor Unit*, Intel Corporation, November 1972.

- [25] *8085AH/8085AH-2/8085AH-1: 8-Bit HMOS Microprocessors*, Intel Corporation, September 1987.
- [26] *8086: 16-Bit HMOS Microprocessors*, Intel Corporation, September 1990.
- [27] *8088: 8-Bit HMOS Microprocessor*, Intel Corporation, August 1990.
- [28] *80186/80188: High-Integration 16-Bit Microprocessors*, Intel Corporation, November 1994.
- [29] *80C286/883: High Performance Microprocessor with Memory Management and Protection*, Intel Corporation, September 1988.
- [30] *Intel386 EX Embedded Microprocessor*, Intel Corporation, October 1998.
- [31] *Embedded Intel486 SX Processor*, Intel Corporation, August 2004.
- [32] *Pentium Processor at iCOMP Inter 610-75MHz*, Intel Corporation, June 1997.
- [33] *Pentium Pro Processor with 1 MB L2 Cache at 200 MHz*, Intel Corporation, January 1998.
- [34] *Pentium II Processor at 233MHz, 266MHz, 300MHz and 333MHz*, Intel Corporation, January 1998.
- [35] *Pentium III Processor for the SC242 at 450MHz to 1.0GHz*, Intel Corporation, July 2002.
- [36] *Pentium IV Processor on 180nm Process*, Intel Corporation, November 2000.
- [37] *Pentium IV Processor on 90nm Process*, Intel Corporation, February 2005.
- [38] *Intel® Core™2 Extreme Processor X8600 and Intel® Core™2 Duo Desktop Processor E6000 and E4000 Sequences*, Intel Corporation, October 2007.
- [39] *Intel® Core™2 Extreme Quad-Core Mobile Processor, Intel® Core™2 Quad Mobile Processor, Intel® Core™2 Extreme Mobile Processor, Intel® Core™2 Duo Mobile Processor, Intel® Core™2 Solo Mobile Processor and Intel® Celeron® Processor on 45-nm Process*, Intel Corporation, November 2008.

- [40] *Intel® Core™2 Duo Processor E8000 and E7000 Series*, Intel Corporation, June 2009.
- [41] *Intel® Core™ i5-600, i3-500 Desktop Processor Series, Intel® Pentium® Desktop Processor 6000 Series*, Intel Corporation, January 2011.
- [42] *Intel® Core™ i7-900 Desktop Processor Extreme Edition Series on 32-nm Process*, Intel Corporation, June 2010.
- [43] *AMD EPYC 7002 Series Processors: A New Standard for the Modern Data-center*, AMD Corporation, February 2020.
- [44] **th and 9th Generation Intel® Core™ Processor Families*, Intel Corporation, April 2019.
- [45] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc, "Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions," *IEEE Journal of Solid-State Circuits*, Vol. 9, No. 5, pp. 256-268, October 1974.
- [46] IARPA, "Trusted Integrated Chips (TIC) Program," IARPA-BAA-11-09, October 2011.
- [47] DigiTimes, "Trends in the Global IC Design Service Market," <http://www.digitimes.com/news/a20120313RS400.html?chid=2>, March 2012.
- [48] E. Cauer, W. Mathis, and R. Pauli, "Life and Work of Wilhelm Cauer (1900 - 1945)," *Proceedings of the International Symposium on Mathematical Theory of Networks and Systems*, pp. 1-10, June 2000.
- [49] R. Torrance and D. James, "The State-of-the-Art in Semiconductor Reverse Engineering," *Proceedings of the IEEE Design Automation Conference*, pp. 333 - 338, June 2011.
- [50] M. Pecht and S. Tiku., "Bogus! Electronic Manufacturing and Consumers Confront a Rising Tide of Counterfeit Electronics," *IEEE Spectrum*, Vol. 43, No. 5, pp. 37-46, May 2006.

- [51] M. Tehranipoor and F. Koushanfar, "A Survey of Hardware Trojan Taxonomy and Detection," *IEEE Design and Test of Computers*, Vol. 27, No. 1, pp. 10-25, February 2010.
- [52] H. Salmani and M. Tehranipoor, "Layout-Aware Switching Activity Localization to Enhance Hardware Trojan Detection," *IEEE Transactions on Information Forensics and Security*, Vol. 7, No. 1, pp. 76-87, February 2012.
- [53] H. Salmani, M. Tehranipoor, and J. Plusquellic, "New Design Strategy for Improving Hardware Trojan Detection and Reducing Trojan Activation Time," *Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust*, pp. 66-73, July 2009.
- [54] S. Dupuis, P. S. Ba, G. Di Natale, M. L. Flottes, and B. Rouzeyre, "A Novel Hardware Logic Encryption Technique for Thwarting Illegal Overproduction and Hardware Trojans," *Proceedings of the IEEE International On-Line Testing Symposium*, pp. 49-54, July 2014.
- [55] X. Zhang and M. Tehranipoor, "RON: An On-Chip Ring Oscillator Network for Hardware Trojan Detection," *Proceedings of the IEEE Design, Automation Test in Europe Conference*, pp. 1-6, March 2011.
- [56] J. Li and J. Lach, "At-Speed Delay Characterization for IC Authentication and Trojan Horse Detection," *Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust*, pp. 8-14, June 2008.
- [57] P. Kumar and R. Srinivasan, "Detection of Hardware Trojan in SEA using Path Delay," *Proceedings of the IEEE Students' Conference on Electrical, Electronics and Computer Science*, pp. 1-6, March 2014.
- [58] A. Davoodi, M. Li, and M. Tehranipoor, "A Sensor-Assisted Self-Authentication Framework for Hardware Trojan Detection," *IEEE Transactions on Design and Test*, Vol. 30, No. 5, pp. 74-82, October 2013.
- [59] K. Xiao, D. Forte, and M. Tehranipoor, "Efficient and Secure Split Manufacturing via Obfuscated Built-in Self-Authentication," *Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust*, pp. 14-19, May 2015.

- [60] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar, "Trojan Detection using IC Fingerprinting," *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 296-310, May 2007.
- [61] C. Bao, D. Forte, and A. Srivastava, "Temperature Tracking: Towards Robust Run-time Detection of Hardware Trojans," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 34, No. 10, pp. 1577-1585, Oct. 2015.
- [62] S. Narasimhan, W. Yueh, X. Wang, S. Mukhopadhyay, and S. Bhunia, "Improving IC Security Against Trojan Attacks Through Integration of Security Monitors," *IEEE Design and Test of Computers*, Vol. 29, No. 5, pp. 37-46, October 2012.
- [63] J. Aarestad, D. Acharyya, R. Rad, and J. Plusquellic, "Detecting Trojans Through Leakage Current Analysis Using Multiple Supply Pad I_{DDQS} ," *IEEE Transactions on Information Forensics and Security*, Vol. 5, No. 4, pp. 893-904, December 2010.
- [64] B. Hou, C. He, L. Wang, Y. En, and S. Xie, "Hardware Trojan Detection via Current Measurement: A Method Immune to Process Variation Effects," *Proceedings of the International Conference on Reliability, Maintainability and Safety*, pp. 1039-1042, August 2014.
- [65] S. Narasimhan, D. Du, R. Chakraborty, S. Paul, F. Wolff, C. Papachristou, K. Roy, and S. Bhunia, "Hardware Trojan Detection by Multiple-Parameter Side-Channel Analysis," *IEEE Transactions on Computers*, Vol. 62, No. 11, pp. 2183-2195, November 2013.
- [66] N. Gunti and K. Lingasubramanian, "Efficient Static Power Based Side Channel Analysis for Hardware Trojan Detection using Controllable Sleep Transistors," *SoutheastCon 2015*, pp. 1-6, April 2015.
- [67] Y. Cao, C. Chang, and S. Chen, "A Cluster-Based Distributed Active Current Sensing Circuit for Hardware Trojan Detection," *IEEE Transactions on Information Forensics and Security*, Vol. 9, No. 12, pp. 2220-2231, December 2014.

- [68] B. Liu and R. Sandhu, "Fingerprint-Based Detection and Diagnosis of Malicious Programs in Hardware," *IEEE Transactions on Reliability*, Vol. 64, No. 3, pp. 1068-1077, Sept. 2015.
- [69] A. Nowroz, K. Hu, F. Koushanfar, and S. Reda, "Novel Techniques for High-Sensitivity Hardware Trojan Detection Using Thermal and Power Maps," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 33, No. 12, pp. 1792-1805, December 2014.
- [70] C. He, B. Hou, L. Wang, Y. En, and S. Xie, "A Novel Hardware Trojan Detection Method Based on Side-Channel Analysis and PCA Algorithm," *Proceedings of the International Conference on Reliability, Maintainability and Safety*, pp. 1043-1046, August 2014.
- [71] Y. Liu, K. Huang, and Y. Makris, "Hardware Trojan Detection Through Golden Chip-Free Statistical Side-Channel Fingerprinting," *Proceedings of the ACM/EDAC/IEEE Design Automation Conference*, pp. 1-6, June 2014.
- [72] O. Soll, T. Korak, M. Muehlberghuber, and M. Hutter, "EM-Based Detection of Hardware Trojans on FPGAs," *Proceedings of the IEEE International Symposium on Hardware-Oriented Security and Trust*, pp. 84-87, May 2014.
- [73] X. Ngo, I. Exurville, S. Bhasin, J. Danger, S. Guilley, Z. Najm, J. Rigaud, and B. Robisson, "Hardware Trojan Detection by Delay and Electromagnetic Measurements," *Proceedings of the Design, Automation and Test in Europe Conference Exhibition*, pp. 782-787, March 2015.
- [74] F. Lodhi, S. Hasan, O. Hasan, and F. Awwad, "Hardware Trojan Detection in Soft Error Tolerant Macro Synchronous Micro Asynchronous (MSMA) Pipeline," *Proceedings of the IEEE International Midwest Symposium on Circuits and Systems*, pp. 659-662, August 2014.
- [75] S. Narasimhan, X. Wang, D. Du, R. Chakraborty, and S. Bhunia, "TeSR: A Robust Temporal Self-Referencing Approach for Hardware Trojan Detection," *IEEE International Symposium on Hardware-Oriented Security and Trust*, pp. 71-74, June 2011.

- [76] D. Werner, K. Juretus, I. Savidis, and M. Hempstead, "Machine Learning on the Thermal Side-Channel: Analysis of Accelerator-Rich Architectures," *Proceedings of the IEEE International Conference on Computer Design*, pp. 83-91, October 2018.
- [77] F. K. Gurkaynak, *GALS System Design: Side Channel Attack Secure Cryptographic Accelerators*, Ph.D. Dissertation, ETH Zurich, Zurich, Switzerland, December 2006.
- [78] K. Yang, M. Hicks, Q. Dong, T. Austin, and D. Sylvester, "A2: Analog Malicious Hardware," *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 18-37, May 2016.
- [79] A. Abdel-Hamid, S. Tahar, and E. Aboulhamid, "IP Watermarking Techniques: Survey and Comparison," *Proceedings of the IEEE International Workshop on System-on-Chip for Real-Time Applications*, pp. 60 - 65, July 2003.
- [80] A. Kahng, D. Kirovski, S. Mantik, M. Potkonjak, and J. Wong, "Copy Detection for Intellectual Property Protection of VLSI Designs," *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 600 - 604, November 1999.
- [81] A. Oliveira, "Techniques for the Creation of Digital Watermarks in Sequential Circuit Designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 20, No. 9, pp. 410-424, September 2001.
- [82] A. Oliveira, "Watermarking-based Copyright Protection of Sequential Functions," *IEEE Journal of Solid-State Circuits*, Vol. 35, No. 3, pp. 434 - 440, March 2000.
- [83] J. Rajendran, O. Sinanoglu, and R. Karri, "Is Split Manufacturing Secure?" *Design, Automation Test in Europe Conference Exhibition*, pp. 1259-1264, March 2013.
- [84] J. Magaña, D. Shi, and A. Davoodi, "Are Proximity Attacks a Threat to the Security of Split Manufacturing of Integrated Circuits?" *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 90:1–90:7, November 2016.

- [85] Y. Wang, P. Chen, J. Hu, and J. J. Rajendran, "The Cat and Mouse in Split Manufacturing," *Proceedings of the IEEE/ACM Design Automation Conference*, pp. 165:1–165:6, June 2016.
- [86] J. Rajendran, O. Sinanoglu, and R. Karri, "Regaining Trust in VLSI Design: Design-for-Trust Techniques," *Proceedings of the IEEE*, Vol. 102, No. 8, pp. 1266-1282, July 2014.
- [87] M. El Massad, S. Garg, and M. Tripunitara, "The SAT Attack on IC Camouflaging: Impact and Potential Countermeasures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1-15, 2019.
- [88] J. Roy, F. Koushanfar, and I. Markov, "EPIC: Ending Piracy of Integrated Circuits," *Proceedings of the IEEE/ACM Design, Automation and Test in Europe Conference*, pp. 1069-1074, October 2008.
- [89] J. Rajendran, H. Zhang, C. Zhang, G. Rose, Y. Pino, O. Sinanoglu, and R. Karri, "Fault Analysis-Based Logic Encryption," *IEEE Transactions on Computers*, Vol. 64, No. 2, pp. 410-424, February 2015.
- [90] A. Baumgarten, A. Tyagi, and J. Zambreno, "Preventing IC Piracy Using Reconfigurable Logic Barriers," *IEEE Design and Test of Computers*, Vol. 27, No. 1, pp. 66-75, February 2010.
- [91] M. Yasin, J. Rajendran, O. Sinanoglu, and R. Karri, "On Improving the Security of Logic Locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 35, No. 9, pp. 1411-1424, September 2016.
- [92] S. M. Plaza and I. L. Markov, "Solving the Third-Shift Problem in IC Piracy With Test-Aware Logic Locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 34, No. 6, pp. 961-971, June 2015.
- [93] M. Bushnell and V. Agrawal, *Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits*. Kluwer Academic, 2002.
- [94] A. Biere, A. Biere, M. Heule, H. van Maaren, and T. Walsh, *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

- [95] U. Schöning and J. Torán, *The Satisfiability Problem: Algorithms and Analyses*, ser. Mathematics for applications. Lehmanns Media, 2013.
- [96] P. Subramanyan, S. Ray, and S. Malik, “Evaluating the Security of Logic Encryption Algorithms,” *Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust*, pp. 137-143, May 2015.
- [97] G. S. Tseytin, “On the Complexity of Derivations in the Propositional Calculus,” Vol. II, pp. 115-125, Consultants Bureau, New York, London, 1968.
- [98] M. Yasin, B. Mazumdar, J. Rajendran, and O. Sinanoglu, “SARLock: SAT Attack Resistant Logic Locking,” *Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust*, pp. 236-241, May 2016.
- [99] Y. Xie and A. Srivastava, “Mitigating SAT Attack on Logic Locking,” *Proceedings of the International Conference on Cryptographic Hardware and Embedded Systems*, pp. 127-146, June 2016.
- [100] M. Li, K. Shamsi, T. Meade, Z. Zhao, B. Yu, Y. Jin, and D. Z. Pan, “Provably Secure Camouflaging Strategy for IC Protection,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 38, No. 8, pp. 1399-1412, August 2019.
- [101] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. Rajendran, and O. Sinanoglu, “Provably-Secure Logic Locking: From Theory To Practice,” *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pp. 1601–1618, November 2017.
- [102] K. Shamsi, T. Meade, M. Li, D. Z. Pan, and Y. Jin, “On the Approximation Resiliency of Logic Locking and IC Camouflaging Schemes,” *IEEE Transactions on Information Forensics and Security*, Vol. 14, No. 2, pp. 347-359, February 2019.
- [103] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, “Removal Attacks on Logic Locking and Camouflaging Techniques,” *IEEE Transactions on Emerging Topics in Computing*, August 2017.

- [104] X. Xu, B. Shakya, M. Tehranipoor, and D. Forte, “Novel Bypass Attack and BDD-based Tradeoff Analysis Against All Known Logic Locking Attacks,” *Proceedings of the Cryptographic Hardware and Embedded Systems Conference*, Vol. 10529, pp. 189–210, 2017.
- [105] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, “AppSAT: Approximately Deobfuscating Integrated Circuits,” *Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust*, pp. 95–100, May 2017.
- [106] Y. Shen and H. Zhou, “Double DIP: Re-Evaluating Security of Logic Encryption Algorithms,” *Proceedings of the Great Lakes Symposium on VLSI*, pp. 179–184, May 2017.
- [107] K. Shamsi, D. Z. Pan, and Y. Jin, “On the Impossibility of Approximation-Resilient Circuit Locking,” *Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust*, pp. 161–170, May 2019.
- [108] D. Sirone and P. Subramanyan, “Functional Analysis Attacks on Logic Locking,” *Proceedings of the IEEE Design, Automation and Test in Europe Conference*, pp. 936–939, March 2019.
- [109] H. Zhou, Y. Shen, and A. Rezaei, “Vulnerability and Remedy of Stripped Function Logic Locking,” Cryptology ePrint Archive, Report 2019/139, 2019, <https://eprint.iacr.org/2019/139>.
- [110] L. Alrahis, M. Yasin, H. Saleh, B. Mohammad, and M. Al-Qutayri, “Functional Reverse Engineering on SAT-Attack Resilient Logic Locking,” *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 1–5, May 2019.
- [111] P. Subramanyan, N. Tsiskaridze, K. Pasricha, D. Reisman, A. Susnea, and S. Malik, “Reverse Engineering Digital Circuits Using Functional Analysis,” *Proceedings of the IEEE Design, Automation and Test in Europe Conference Exhibition*, pp. 1277–1280, March 2013.
- [112] A. Sengupta, M. Nabeel, M. Yasin, and O. Sinanoglu, “ATPG-based Cost-effective, Secure Logic Locking,” *Proceedings of the IEEE VLSI Test Symposium*, pp. 1–6, April 2018.

- [113] M. J. Y. Williams and J. B. Angell, "Enhancing Testability of Large-Scale Integrated Circuits via Test Points and Additional Logic," *IEEE Transactions on Computers*, Vol. C-22, No. 1, pp. 46-60, January 1973.
- [114] Y. Zorian, "A Distributed BIST Control Scheme for Complex VLSI Devices," *Proceedings of the IEEE VLSI Test Symposium*, pp. 4-9, April 1993.
- [115] S. Chakraborty and S. Bhunia, "HARPOON: An Obfuscation-Based SoC Design Methodology for Hardware Protection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 28, No. 10, pp. 1493-1502, October 2009.
- [116] R. Maes, *Physically Unclonable Functions: Constructions, Properties and Applications*. Springer-Verlag Berlin Heidelberg, 2013.
- [117] A. R. Desai, M. S. Hsiao, C. Wang, L. Nazhandali, and S. Hall, "Interlocking Obfuscation for Anti-tamper Hardware," *Proceedings of the Cyber Security and Information Intelligence Research Workshop*, pp. 8:1-8:4, January 2013.
- [118] T. Meade, S. Zhang, and Y. Jin, "Netlist Reverse Engineering for High-level Functionality Reconstruction," *Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 655-660, January 2016.
- [119] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache, "Fault Injection Attacks on Cryptographic Devices: Theory, Practice, and Countermeasures," *Proceedings of the IEEE*, Vol. 100, No. 11, pp. 3056-3076, November 2012.
- [120] C. H. Kim and J. J. Quisquater, "Faults, Injection Methods, and Fault Attacks," *IEEE Design and Test of Computers*, Vol. 24, No. 6, pp. 544-545, November 2007.
- [121] K. Rothbart, U. Neffe, C. Steger, R. Weiss, E. Rieger, and A. Muehlberger, "High Level Fault Injection for Attack Simulation in Smart Cards," *Proceedings of the Asian Test Symposium*, pp. 118-121, November 2004.
- [122] Y. Alkabani and K. Farinaz, "Active Hardware Metering for Intellectual Property Protection and Security," *Proceedings of the USENIX Security Symposium*, pp. 291-306, August 2007.

- [123] K. Juretus and I. Savidis, "Time Domain Sequential Locking for Increased Security," *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 1-5, May 2018.
- [124] T. Meade, Z. Zhao, S. Zhang, D. Pan, and Y. Jin, "Revisit Sequential Logic Obfuscation: Attacks and Defenses," *Proceedings of the IEEE International Conference on Circuits and Systems*, pp. 1367-1370, May 2017.
- [125] J. Dofe and Q. Yu, "Novel Dynamic State-Deflection Method for Gate-Level Design Obfuscation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. PP, No. 99, pp. 1-13, April 2017.
- [126] Y. Alkabani, F. Koushanfar, and M. Potkonjak, "Remote Activation of ICs for Piracy Prevention and Digital Right Management," *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 674-677, November 2007.
- [127] A. Niemetz, M. Preiner, and A. Biere, "Technical Report 15/1," *Boolector at the SMT competition 2015*, June 2015.
- [128] K. Shamsi, W. Wen, and Y. Jin, "Hardware Security Challenges Beyond CMOS: Attacks and Remedies," *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, pp. 200-205, July 2016.
- [129] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security Analysis of Logic Obfuscation," *Proceedings of the ACM/EDAC/IEEE Design Automation Conference*, pp. 83-89, June 2012.
- [130] J. Cong and Y. Ding, "On Area/Depth Trade-off in LUT-based FPGA Technology Mapping," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 2, No. 2, pp. 137-148, June 1994.
- [131] S. Dupuis, P. S. Ba, G. D. Natale, M. L. Flottes, and B. Rouzeyre, "A Novel Hardware Logic Encryption Technique for Thwarting Illegal Overproduction and Hardware Trojans," *Proceeding of the IEEE International On-Line Testing Symposium*, pp. 49-54, July 2014.

- [132] C. Yu, X. Zhang, D. Liu, M. Ciesielski, and D. Holcomb, "Incremental SAT-Based Reverse Engineering of Camouflaged Logic Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 36, No. 10, pp. 1647-1659, October 2017.
- [133] G. Sengar, D. Mukhopadhyay, and D. R. Chowdhury, "Secured Flipped Scan-Chain Model for Crypto-Architecture," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 26, No. 11, pp. 2080-2084, November 2007.
- [134] Y. Atobe, Y. Shi, M. Yanagisawa, and N. Togawa, "Dynamically Changeable Secure Scan Architecture Against Scan-based Side Channel Attack," *Proceedings of the IEEE International SoC Design Conference*, pp. 155-158, November 2012.
- [135] R. Karmakar, S. Chattopadhyay, and R. Kapur, "Encrypt Flip-Flop: A Novel Logic Encryption Technique For Sequential Circuits," *The Computing Research Repository*, Vol. abs/1801.04961, 2018.
- [136] J. Lee, M. Tehranipoor, and J. Plusquellic, "A Low-cost Solution for Protecting IPs Against Scan-based Side-channel Attacks," *Proceedings of the IEEE VLSI Test Symposium*, pp. 1-6, April 2006.
- [137] M. A. Razzaq, V. Singh, and A. Singh, "SSTKR: Secure and Testable Scan Design Through Test Key Randomization," *Proceedings of the IEEE Asian Test Symposium*, pp. 60-65, November 2011.
- [138] S. Paul, R. S. Chakraborty, and S. Bhunia, "VIm-Scan: A Low Overhead Scan Design Approach for Protection of Secret Key in Scan-Based Secure Chips," *Proceedings of the IEEE VLSI Test Symposium*, pp. 455-460, May 2007.
- [139] X. Wang, D. Zhang, M. He, D. Su, and M. Tehranipoor, "Secure Scan and Test Using Obfuscation Throughout Supply Chain," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 37, No. 9, pp. 1867-1880, September 2018.

- [140] U. Guin, Z. Zhou, and A. Singh, "Robust Design-for-Security Architecture for Enabling Trust in IC Manufacturing and Test," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 26, No. 5, pp. 818-830, May 2018.
- [141] M. E. Massad, S. Garg, and M. Tripunitara, "Reverse engineering camouflaged sequential circuits without scan access," *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 33-40, November 2017.
- [142] K. Shamsi, M. Li, D. Z. Pan, and Y. Jin, "KC2: Key-Condition Crunching for Fast Sequential Circuit Deobfuscation," *Proceedings of the IEEE Design, Automation Test in Europe Conference Exhibition*, pp. 534-539, March 2019.
- [143] K. Juretus and I. Savidis, "Reduced Overhead Gate Level Logic Encryption," *Proceedings of the IEEE/ACM Great Lakes Symposium on VLSI*, pp. 15-20, May 2016.
- [144] K. Juretus and I. Savidis, "Reducing Logic Encryption Overhead Through Gate Level Key Insertion," *Proceedings of the IEEE International Conference on Circuits and Systems*, pp. 1714-1717, May 2016.
- [145] Y. Shi, C. W. Ting, B. Gwee, and Y. Ren, "A highly efficient method for extracting FSMs from flattened gate-level netlist," *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 2610-2613, May 2010.
- [146] R. Tarjan, "Depth-First Search and Linear Graph Algorithms," *SIAM Journal on Computing*, Vol. 1, No. 2, pp. 146-160, 1972.
- [147] S. Cook, "The Complexity of Theorem-Proving Procedures," *Proceedings of the ACM Symposium on Theory of Computing*, p. 151-158, 1971.
- [148] K. Juretus and I. Savidis, "Characterization of In-Cone Logic Locking Resiliency Against the SAT Attack," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1-14, 2019.
- [149] F. Corno, M. Reorda, and G. Squillero, "RT-level ITC'99 benchmarks and first ATPG results," *IEEE Transactions on Design and Test of Computers*, Vol. 17, No. 3, pp. 44-53, Jul 2000.

- [150] F. Brglez, D. Bryan, and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 1929-1934, May 1989.
- [151] Y. Shen, Y. Li, S. Kong, A. Rezaei, and H. Zhou, "SigAttack: New High-level SAT-based Attack on Logic Encryptions," *Proceedings of the IEEE Design, Automation Test in Europe Conference*, pp. 940-943, March 2019.
- [152] D. Duvalsaint, Z. Liu, A. Ravikumar, and R. D. Blanton, "Characterization of Locked Sequential Circuits via ATPG," *Proceedings of the IEEE International Test Conference in Asia*, pp. 97-102, 2019.
- [153] S. Phatharodom, N. Kandasamy, and I. Savidis, "Closed-form Estimation of SAT-Attack Resilience," *Proceedings of the Government Microcircuit Applications & Critical Technology Conference*, p. PP, March 2020.
- [154] S. Engels, M. Hoffmann, and C. Paar, "The End of Logic Locking? A Critical View on the Security of Logic Locking," Cryptology ePrint Archive, Report 2019/796, 2019, <https://eprint.iacr.org/2019/796>.

Vita



Kyle Juretus received his Bachelor of Science in Computer and Electrical Engineering in 2014 and his Master of Science in Computer Engineering in 2016, all at Drexel University. He received his Ph.D. in Electrical Engineering from Drexel University in 2020. Kyle was a National Defense Science and Engineering Graduate (NDSEG) fellow from 2016-2019. His research focuses on creating secure and efficient integrated circuits (ICs) through the development of circuit level techniques to prevent intellectual property theft and counterfeiting, mitigate side-channel leakage of integrated circuit designs, and improve electronic design automation methodologies.

Honors and Awards

Nihat Bilgutay Fellowship	2019
Koerner Family Fellowship	2018
ASEE NDSEG Fellowship Awardee	2016
ASEE SMART Fellowship Awardee	2016
NSF GRFP Fellowship Honorable Mention	2016
Harry L. Brown Jr. Fellowship	2015
Blasi Family Endowed Award	2014
Harry E. Muchnic Scholarship	2011 - 2013
AJ Drexel Academic Scholarship	2009 - 2014

Publications

Journal Publications

M. Jacovic, **K. Juretus**, N. Kandasamy, I. Savidis, and K. R. Dandekar, "Physical Layer Encryption for Wireless OFDM Communication Systems," *Journal of*

Hardware and Systems Security, Vol. PP, July 2020.

K. Juretus and I. Savidis, "Increased Output Corruption and Structural Attack Resiliency for SAT Attack Secure Logic Locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. PP, June 2020.

K. Juretus and I. Savidis, "Synthesis of Hidden State Transitions for Sequential Logic Locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. PP, June 2020.

K. Juretus and I. Savidis, "Characterization of In-Cone Logic Locking Resiliency Against the SAT Attack," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 39, No. 8, pp. 1607-1620, Aug 2020.

J. Chacko, **K. Juretus**, M. Jacovic, C. Sahin, N. Kandasamy, I. Savidis, and K. R. Dandekar, "Securing Wireless Communication via Hardware-Based Packet Obfuscation," *Journal of Hardware and Systems Security*, Vol. 3, No. 3, pp. 261-272, May 2019.

Conference Publications

K. Juretus and I. Savidis, "Reducing Logic Locking Key Leakage Through the Scan Chain," *Proceedings of the International Symposium on Circuits and Systems*, [accepted].

V. Rao, **K. Juretus** and I. Savidis, "Security Vulnerabilities of Obfuscated Analog Circuits," *Proceedings of the International Symposium on Circuits and Systems*, [accepted].

K. Juretus and I. Savidis, "Increasing the SAT Attack Resiliency of In-Cone Logic Locking," *Proceedings of the International Symposium on Circuits and Systems*, pp. 1-5, May 2019.

K. Juretus, V. Rao, and I. Savidis, "Securing Analog Mixed-Signal Integrated Circuits Through Shared Dependencies," *Proceedings of the IEEE/ACM International Great Lakes Symposium on VLSI*, pp. 483-488, May 2019.

K. Juretus and I. Savidis, "Importance of Multi-parameter SAT Attack Exploration for Integrated Circuit Security," *Proceedings of the IEEE International Asia Pacific Conference on Circuits and Systems*, pp. 366-369, October 2018.

D. Werner, **K. Juretus**, I. Savidis, Mark Hempstead, "Machine Learning on the Thermal Side-Channel: Analysis of Accelerator-Rich Architectures," *Proceedings of the IEEE International Conference on Computer Design*, pp. 83-91, October 2018.

K. Juretus and I. Savidis, "Time Domain Sequential Locking for Increased Security," *Proceedings of the International Symposium on Circuits and Systems*, pp. 1-5, May 2018.

K. Juretus and I. Savidis, "Enhanced Circuit Security Through Hidden State Transitions," *Proceedings of the Government Microcircuit Applications & Critical Technology Conference*, pp. 781-784, March 2018.

J. Chacko, **K. Juretus**, M. Jacovic, C. Sahin, N. Kandasamy, I. Savidis, and K. Dandekar, "Physical Gate Based Preamble Obfuscation for Securing Wireless Communication," *Proceedings of the International Conference on Computing, Networking and Communications*, pp. 293-297, January 2017.

K. Juretus and I. Savidis, "Reducing logic encryption overhead through gate level key insertion," *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 1747-1717, May 2016.

K. Juretus and I. Savidis, "Reduced Overhead Gate Level Logic Encryption," *Proceedings of the IEEE/ACM International Great Lakes Symposium on VLSI*, pp. 15-20, May 2016.

K. Juretus and I. Savidis, "Low Overhead Gate Level Logic Encryption," *Proceedings of the Government Microcircuit Applications & Critical Technology Conference*, pp. 455-459, March 2016.

Patents

K. Juretus, V. Rao, and I. Savidis, "Securing Analog Mixed-Signal Integrated Circuits Through Shared Dependencies," US Patent Application No. 62,839,858.

J. Chacko, K. Juretus, M. Jacovic, C. Sahin, N. Kandasamy, I. Savidis, and K. Dandekar, "Physical Layer Key Based Interleaving for Secure Wireless Communication," US Patent Application No. 62,515,369.

J. Chacko, K. Juretus, M. Jacovic, C. Sahin, N. Kandasamy, I. Savidis, and K. Dandekar, "Physical Layer Key Based Pilot Encryption for Secure Wireless Communication," US Patent Application No. 62,517,359.

V. Rao, I. Savidis, and K. Juretus, "Protecting Analog Circuits with Parameter Biasing Obfuscation," US Patent Application No. 15,918,278.

J. Chacko, K. Juretus, M. Jacovic, C. Sahin, N. Kandasamy, I. Savidis, and K. Dandekar, "Physical Gate Based Preamble Obfuscation for Securing Wireless Communication," US Patent Application No. 18,132,796.

K. Juretus and I. Savidis, "Low Overhead Gate Level Logic Encryption," US Patent Application No. 62,245,155.

