DATE 25
DESIGN, AUTOMATION AND TEST IN EUROPE
THE EUROPEAN EVENT FOR ELECTRONIC SYSTEM DESIGN & TEST
31 MARCH – 2 APRIL 2025
LYON, FRANCE
CENTRE DE CONGRÈS DE LYON

Cute-Lock: Behavioral and Structural Multi-Key Logic Locking Using Time Base Keys

Kevin Lopez and **Amin Rezaei**

Contents

- **Introduction**
  - Fabless Manufacturing
  - Logic Locking
  - Motivation
- **Contribution**
  - Cute-Lock-Beh
  - Cute-Lock-Str
- **Conclusion**
  - Summary
  - Future Work
  - Acknowledgment

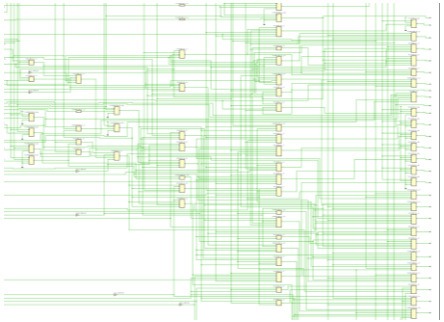
Introduction - Fabless Manufacturing

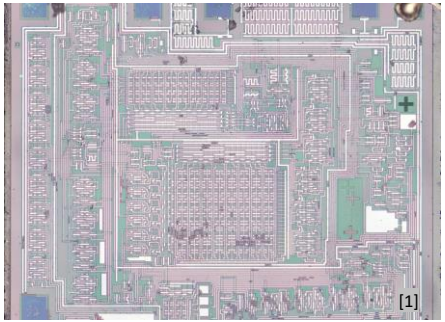
https://www.cirrus.com/company/quality/fabless-semiconductor-model/

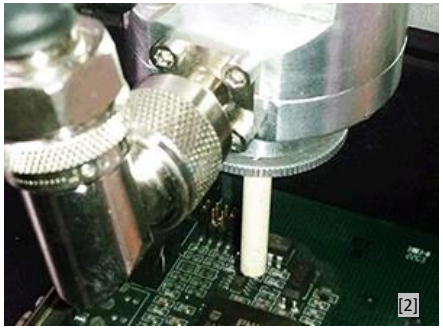
Introduction - Fabless Manufacturing

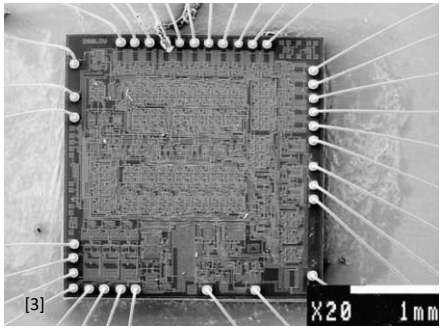
Threats

- Reverse Engineering
- Piracy
- Overproduction
- Counterfeiting









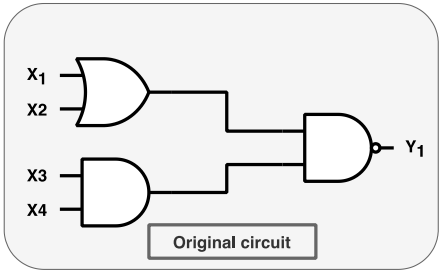
1 - http://www.righto.com/2021/12/reverse-engineering-tiny-1980s-chip.html
2 - https://www.ic-crack.com/reverse-engineering-ic-atmega644a-firmware/
3 - https://www.ic-crack.com/brute-force-chip-break-application/

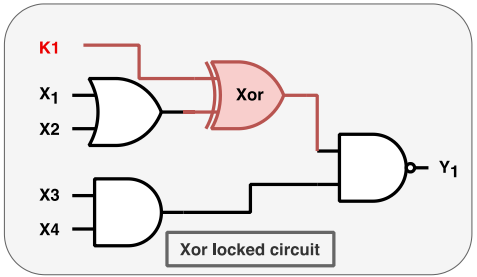
Introduction - Logic Locking

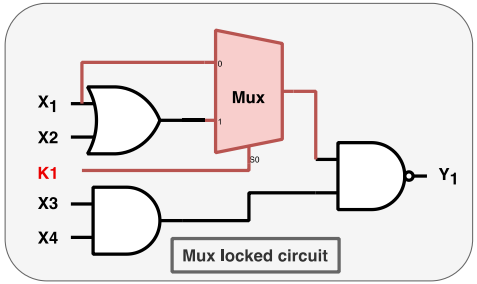
Combinational Logic Locking

- Two categories of combinational logic locking:
  - XOR-based[1]
  - MUX-based[2]

  - XOR Method K1=0
  - MUX Method K1=1





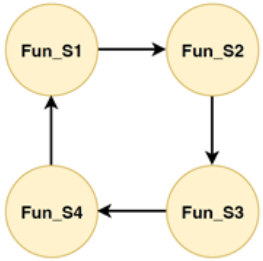


1 - J. A. Roy et al., "Ending piracy of integrated circuits," Computer, vol. 43, no. 10, pp. 30–38, 2010.
2- J. Rajendran et al., "Fault analysis-based logic encryption," IEEE Transactions on Computers, pp. 410-424, 2013.

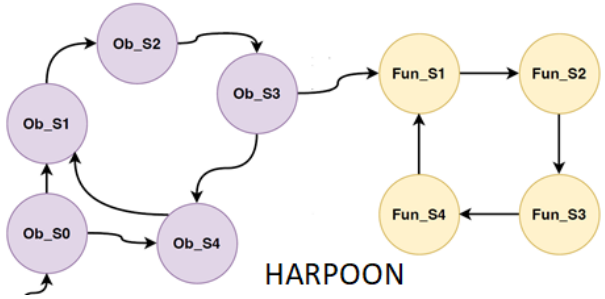
Introduction - Logic Locking

Sequential Logic Locking

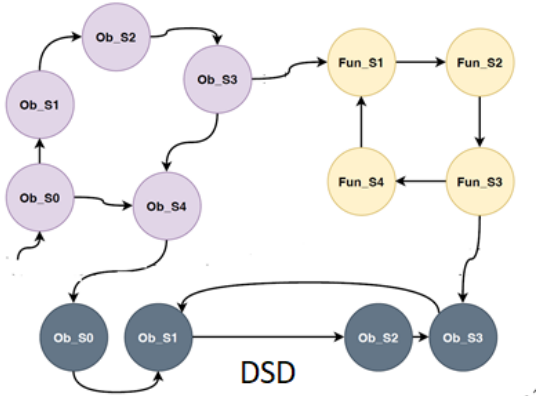
- Use state transition to lock the FSM.
- HARPOON[1]: Need to traverse through obfuscation states.
- DSD[2]: HARPOON + Adding black holes.



Original



HARPOON



DSD

1- R. Chakraborty et al., "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, 2009.
2- J. Dofe et al., "Novel dynamic state-deflection method for gate-level design obfuscation," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no. 2, pp. 273-285, 2018.

Introduction - Logic Locking

Oracle Guided Attacks

Goal: Determine the secret key used for logic locking

Attacker has:

- Locked netlist
- Functional IC

Attacker does:

- Compute the attack pattern from the locked netlist (DIPS);
- Apply them to IC;
- Find key values from responses.

Introduction - Logic Locking

The SAT Attack

- This attack[1] came out in 2015, breaking almost all the locking solutions.
- It works by removing multiple possible keys in one DIP.

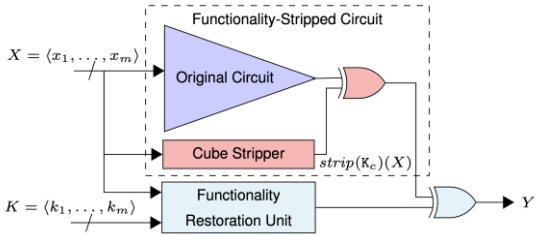
Functional Inputs	Key Inputs							
X[0:2]	K_0	K_1	K_2	K_3	K_4	K_5	K_6	K_7
X_0	0	1	1	1	1	0	0	0
X_1	0	1	1	1	1	0	0	0
X_2	0	1	1	1	1	0	0	0
X_3	0	1	0	0	1	0	1	1
X_4	1	1	0	1	0	0	0	0
X_5	1	1	0	1	0	0	1	0
X_6	1	1	0	1	0	0	1	0
X_7	1	1	1	0	0	0	1	1

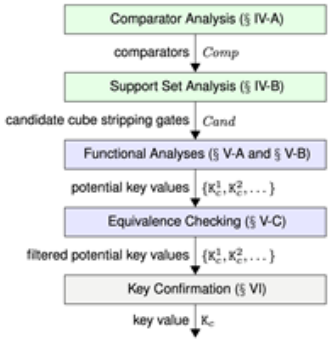
1- P. Subramanyan et al., "Evaluating the security of logic encryption algorithms," IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 137-143, 2015.

Introduction - Logic Locking

Structural Attack – FALL

- Categorize as a removal attack because of the aim to find the keys structurally.
- First, the FALL attack[1] attempts to find a list of potential keys using the structure of the circuit.
- Next, from a list of potential keys, the FALL attack attempts to find the correct key using a SAT solver.





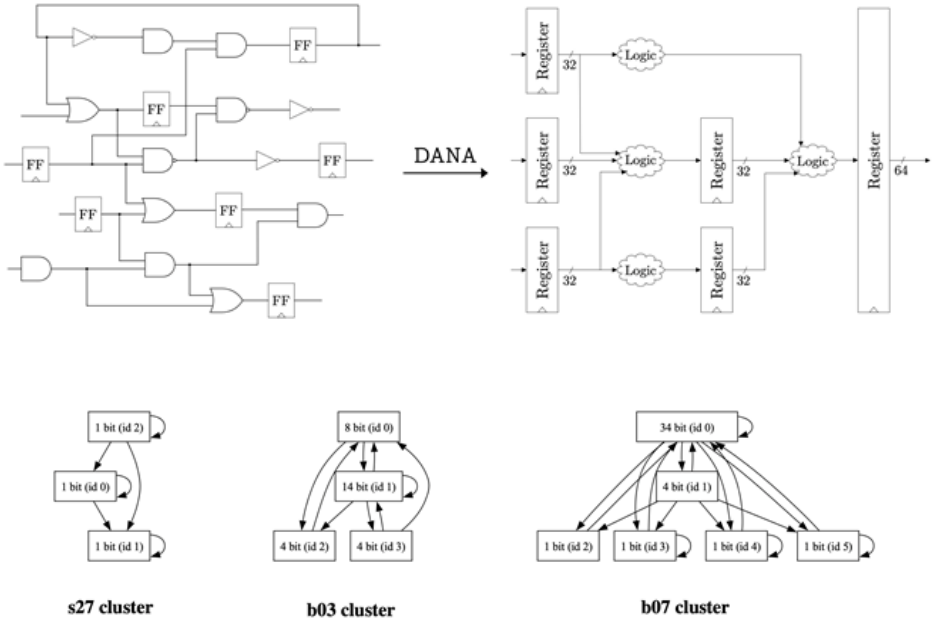
[1]

1- D. Sirone et al, "Functional Analysis Attacks on Logic Locking," Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 936-939, 2019.

Introduction - Logic Locking

Dataflow Attack – DANA

- The dataflow attack's main goal is to help attackers reverse engineer the netlist.
- DANA[1] aims to identify the high-level register structures from the unstructured sea of gates.
- DANA's output is a cluster of flip-flops.



1- N Albartus et al. "DANA Universal Dataflow Analysis for Gate-Level Netlist Reverse Engineering," IACR Transactions on Cryptographic Hardware and Embedded Systems, Vol. 4, 309-336, 2020.

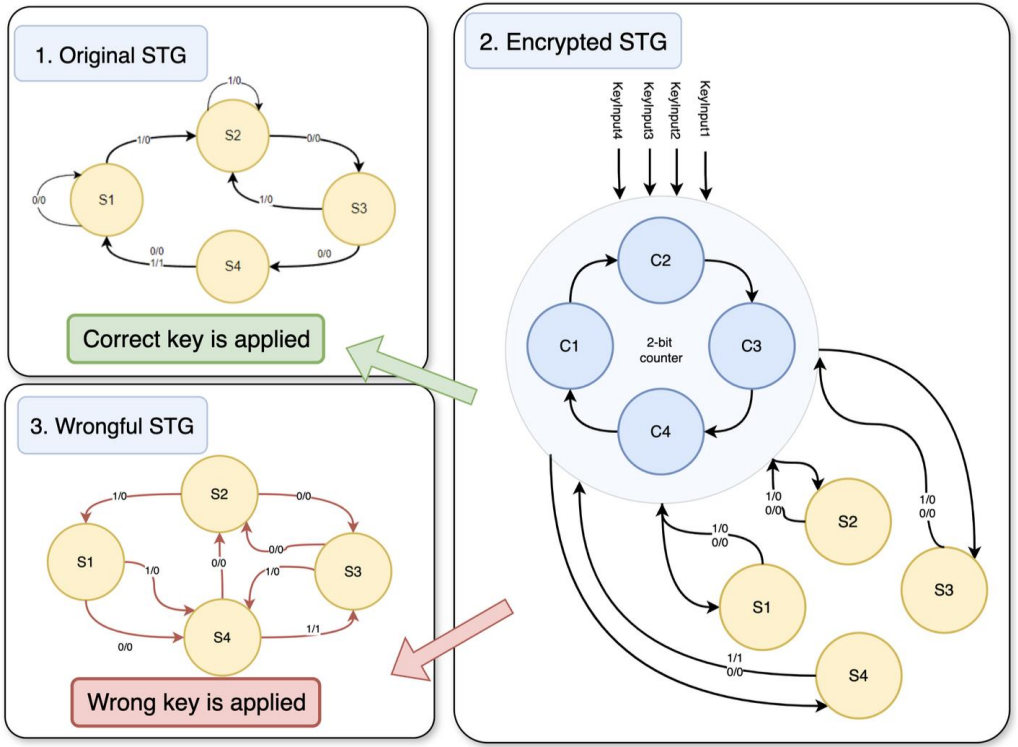
Motivation for Multi-Key Logic Locking

- A single key is vulnerable to current SAT attacks as well as removal and dataflow attacks.
  - It is impossible for advanced attacks to find all the correct keys in multi-key logic locking.
- Keys that should be applied at different clock cycles instead of one global key.
- The theoretical time complexity to break is exponential in relation to the number of keys and inputs.

Contribution - Cute-Lock-Beh

Cute-Lock-Beh

- **C**ounter-based m**ul**t**i**-k**e**y logic **lock**ing **beh**avioral solution to secure circuits against logic attacks in behavioral representation.
- Sequential logic locking Solution aimed to protect against oracle-guided SAT attacks.



Contribution - Cute-Lock-Beh

Cute-Lock-Beh Implementation

- Implementation of Cute-Lock-Beh requires only changes in the HDL code:
  - Add a counter;
  - Wrongful state transition when a wrong key is provided.

```
always @(negedge clk or posedge rst) begin
        if (rst == 1'b1)
                counter <= 0;
        else
                counter <= (counter >= 5) ? 0 : counter + 1;
end
```

```
begin
        if ( counter <= 1 ) begin
                if ( keyinput0 == 1'b1 && keyinput1 == 1'b1 )
                        pr_state = nx_state;
                else
                        pr_state = s3;
        end
        if ( counter > 1 && counter <= 3 ) begin
                if ( keyinput0 == 1'b1 && keyinput1 == 1'b0 )
                        pr_state = nx_state;
                else
                        pr_state = s2;
        end
        if ( counter > 3 && counter <= 5 ) begin
                if ( keyinput0 == 1'b0 && keyinput1 == 1'b1 )
                        pr_state = nx_state;
                else
                        pr_state = s1;
        end
end
```

Contribution - Cute-Lock-Beh

Cute-Lock-Beh Validation

- Validation was done using Xilinx Vivado.
- bcomp benchmark from the Synthezza suite with 19 key-bit values.
- When the correct key was provided, the output of the circuit was the same as the original (y_{ck}).
- When the wrong key was provided the, output was different compared to the original circuit (y_{wk}).
- Additionally, locking the benchmarks with constant keys allowed NEOS and RANE to find the correct key.

	Inputs	Outputs		
Time (ns)	x[7:0]	y[38:0]	$y_{ck}[38:0]$	$y_{wk}[38:0]$
0	0	0	0	0
60	2aaaa	0	0	400000000
100	3c3c3	2000002007	2000002007	00000000e
120	3c3c3	1800000002	1800000002	00000000e
160	2aaaa	0	0	00000000e
200	3c3c3	1800000002	1800000002	000071
220	2aaaa	400240	400240	91
240	2aaaa	0	0	91
260	2aaaa	0	0	91
280	2aaaa	0	0	2004
300	0	0	0	2004
320	0	0	0	0
340	0	0	0	0
360	0	0	0	0
380	0	20000002007	20000002007	e
380	3c3c3	1800000002	1800000002	e

[1] CNS, [2] x..x, [3] FAIL, [4] Equal [5] N/A

Contribution - Cute-Lock-Beh

Cute-Lock-Beh Against Oracle Guided Attacks

- Locked Synthezza benchmarks in Verilog.
- Converted .v to .bliff format, then converted to .bench format (Using Yosys and ABC).
- Run NEOS SAT attacks:
  - No attack was able to find the correct keys

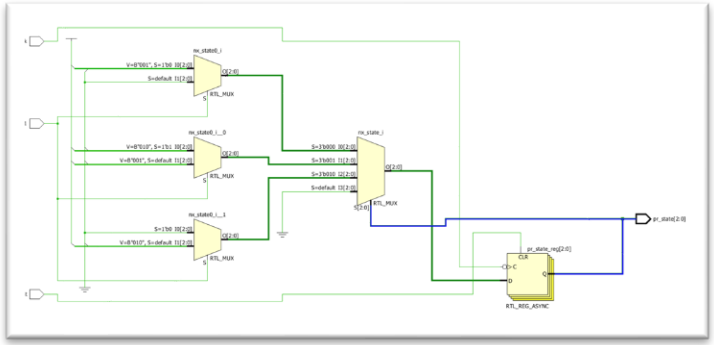
Benchmark and Locking Information				NEOS [56]		
Synthezza [62]	Circuit	# Keys (k)	Key Size (k_i)	BBO	INT	KC2
Small	bcomp	6	18	6m25.446s	0m0.885s	0m1.030s
	bech	6	18	6m4.845s	0m0.723s	0m0.838s
	bridge	5	16	3m28.614s	0m0.100s	0m0.182s
	cat	3	11	15m1.161s	0m0.772s	0m0.680s
	checker9	3	10	3m0.931s	0m0.842s	0m0.803s
	cpu	4	14	2m11.658s	0m0.732s	0m0.799s
	dmac	2	7	1m45.751s	0m0.623s	0m0.681s
	e10	3	10	3m17.832s	0m0.816s	0m1.033s
	e15	4	13	8m59.511s	0m1.361s	0m1.462s
	e16	4	13	7m50.966s	0m0.774s	0m0.918s
	e161	5	16	2m53.761s	0m0.731s	0m0.759s
	e17	2	8	15m0.543s	0m0.522s	0m0.607s
Medium	acdl	5	16	14m47.149s	0m0.641s	0m1.157s
	alf	0	31	0m0.180s	0m0.107s	0m0.469s
	amtz	7	23	14m9.747s	0m2.227s	0m2.727s
	ball	4	44	15m5.744s	0m1.162s	0m4.998s
	bens	7	21	15m3.290s	0m18.365s	0m19.804s
	berg	7	21	10m5.736s	0m1.730s	0m2.531s
	bib	7	21	15m3.795s	0m2.750s	0m3.324s
	big	6	18	11m14.492s	0m0.658s	0m1.163s
	bs	6	19	9m52.679s	0m0.600s	0m0.895s
	codec	2	4	15m2.282s	0m2.252s	0m2.032s
	codec1_2	9	28	15m4.756s	0m3.768s	0m4.005s
	cow	6	49	15m7.930s	0m1.225s	0m4.587s
	cyr	6	20	14m7.341s	0m2.375s	0m3.072s
	dav	6	18	15m3.939s	0m0.519s	0m1.019s
	doron	7	22	13m49.117s	0m2.854s	0m4.004s
Large	absurd	21	65	15m25.370s	0m40.360s	1m9.523s
	bulln	20	61	15m23.190s	1m19.553s	8m7.918s
	camel	19	59	16m29.513s	3m38.506s	14m34.180s
	exxm	15	47	15m52.984s	3m46.605s	3m43.372s
	lion	18	55	15m37.603s	1m31.160s	4m26.537s
	tiger	17	51	15m50.498s	0m37.245s	1m54.818s

[1] CNS, [2] x..x, [3] FAIL, [4] Equal [5] N/A

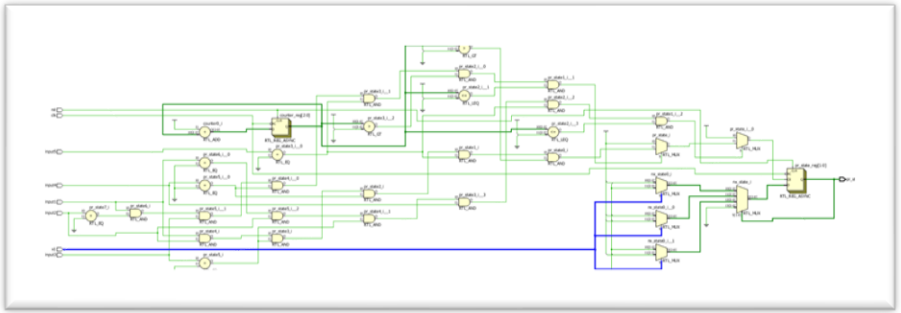
Problem with Cute-Lock-Beh

- High overhead.
- Encryption can be done in HDL (behavioral) level.
- What if the circuit is in structural or gate-level netlist form?
  - Exponential time to revert to state transition graph.

Original



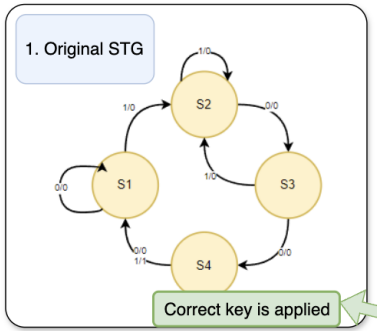
Locked



Contribution - Cute-Lock-Str

Cute-Lock-Str

- **C**ounter-based m**ul**t**i**-k**e**y logic **lock**ing **str**uctural solution to secure circuits against both logic and structural attacks in netlist representation.
- Improved version of Cute-lock-Beh aimed to have lower overhead and to be locked in structural level.
- Instead of transitioning to a random state upon providing a wrong key, it moves to a different state, predefined by existing state transition hardware.



2. STT

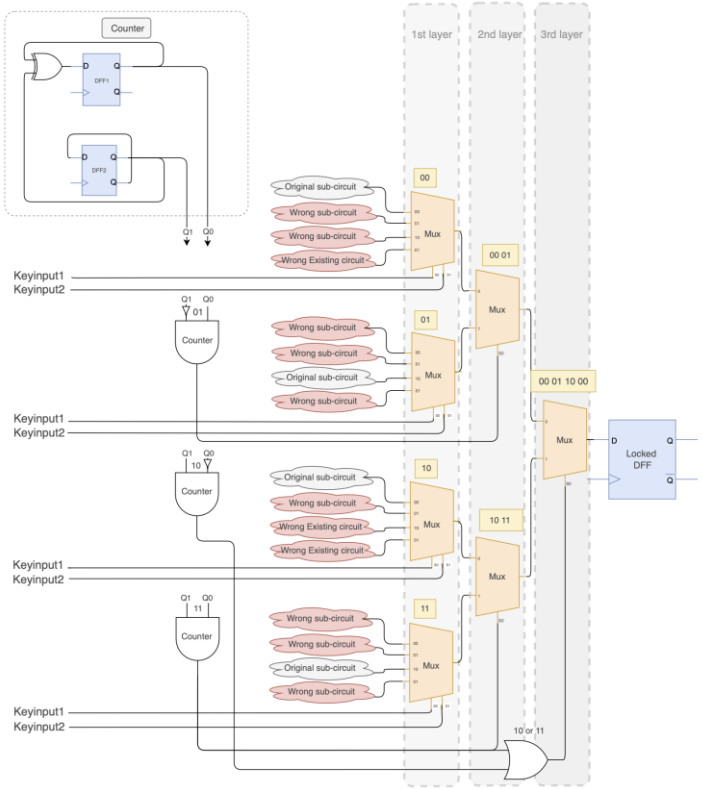
Circuit state transition table, when correct key and wrong key is applied

Input X	PS Q1	PS Q0	NS Q1+	NS Q0+	NS Q0+ *	Output Y
0	0	0	0	0	0	0
0	0	1	1	0	1	0
0	1	0	1	1	1	0
0	1	1	0	0	0	0
1	0	0	0	1	0	0
1	0	1	0	1	0	0
1	1	0	0	1	0	0
1	1	1	0	0	0	1

Contribution - Cute-Lock-Str

MUX Tree - Locking a Flip-Flop

- We use MUX trees to lock a flip-flop with multiple keys.
- 1st layer of MUXs is to verify that the correct key is provided.
- 2nd- m^{th} layer is to coordinate which key should be provided based on the counter inputs.



Contribution - Cute-Lock-Str

Cute-Lock-Str Validation

- Validation was done using Xilinx Vivado.
- Converted bench file to Verilog using ABC.
- s27 from ISCAS 89 is locked using the following keys: 1, 3, 2, 0.
- When the correct key was provided, the output of the circuit was the same as the original ($G17_{ck}$).
- When the wrong key was provided the, output was different compared to the original circuit ($G17_{wk}$).
- Additionally, locking the benchmarks with constant keys allowed NEOS and RANE to find the correct key.

	Inputs				Outputs		
Time (ns)	G0	G1	G2	G3	G17	$G17_{ck}$	$G17_{wk}$
0	0	1	0	1	x	x	x
20	1	0	1	0	1	1	1
40	1	1	0	0	1	1	1
60	1	1	1	0	1	1	1
80	0	1	0	1	1	1	1
100	1	0	1	0	1	1	1
120	0	0	0	0	1	1	1
140	1	1	1	1	1	1	0
160	0	0	1	1	0	0	0
180	1	0	0	1	0	0	1
200	0	1	1	0	0	0	0
220	0	1	1	1	0	0	1
240	1	1	0	1	1	1	1
260	0	0	0	1	1	1	1
280	1	0	1	1	1	1	0

[1] CNS, [2] x..x, [3] FAIL, [4] Equal [5] N/A

Contribution - Cute-Lock-Str

Cute-Lock-Str Against Oracle Guided Attacks

- Locked ISCAS 89, and ITC 99 benchmarks in .bench format.
- Run NEOS and RANE SAT attacks:
  - No attack was able to find the correct keys.
  - Some attacks failed.
  - Some did not report the correct key.
  - One attack timed out (20 hours).

	Benchmark and Locking Information			NEOS [56]			RANE [38]
	Circuit	# keys (k)	Key Size (k_i)	BBO	INT	KC2	RANE
ISCAS 89 [53]	s1196	4	14	1m20.096s	0m0.694s	0m0.753s	0m1.667s
	s13207	8	31	15m5.270s	0m15.520s	0m19.852s	0m35.909s
	s1488	2	8	1m37.663s	0m0.672s	0m0.723s	0m1.224s
	s15850	4	14	15m9.218s	0m12.460s	0m14.394s	0m20.279s
	s298	2	3	0m0.043s	0m0.474s	0m0.474s	0m0.798s
	s349	4	9	7m21.210s	0m0.672s	0m0.695s	0m0.964s
	s35932	8	35	15m5.694s	3m43.671s	4m9.463s	3m31.814s
	s510	8	19	0m35.772s	0m0.539s	0m0.540s	0m0.942s
	s5378	8	35	7m50.965s	0m1.287s	0m1.486s	0m3.591s
	s641	8	35	0m58.063s	0m0.804s	0m1.191s	0m1.326s
	s713	8	35	0m56.985s	0m0.624s	0m0.659s	0m1.234s
	s832	8	18	0m49.561s	0m0.563s	0m0.603s	0m1.080s
	s9234	8	19	15m4.725s	6h44m50s	7h56m45s	50m 6.04s
	s953	4	15	0m52.608s	0m0.826s	0m0.127s	2h6m4.59s
ITC 99 [64]	b01	2	2	0m0.296s	0m1.023s	0m0.882s	9m6.02s
	b02	2	2	0m0.143s	0m0.487s	0m0.653s	10m39.54s
	b03	2	4	15m0.528s	0m0.473s	0m0.653s	13m6.39s
	b04	4	11	0m52.426s	0m0.820s	0m0.194s	4h5m53.21s
	b05	2	2	0m0.153s	0m0.097s	0m0.089s	0m0.415s
	b06	2	1	0m0.151s	0m0.402s	0m0.165s	0m0.441s
	b07	2	2	0m0.163s	0m0.739s	0m0.863s	0m0.544s
	b08	4	9	0m14.811s	0m0.600s	0m0.186s	14m34.59s
	b09	2	1	0m0.250s	0m0.658s	0m0.698s	0m0.560s
	b10	4	11	0m16.103s	0m0.719s	0m0.206s	16m48.31s
	b11	2	7	1m36.385s	0m1.699s	0m0.256s	23m17.52s
	b12	2	5	16m20.762s	1m24.733s	0m0.261s	1h27m50.43s
	b14	8	32	15m3.473s	1m55.654s	0m1.083s	19m39.38s
	b15	16	36	14m3.219s	20m0.006s	0m4.006s	40m34.59s
	b17	16	37	17m1.496s	20m0.008s	20m0.011s	20h0m0.35s
	b18	16	37	0m0.320s	0m0.258s	0m0.252s	1h59m18.06s
	b19	8	24	0m0.538s	0m0.574s	0m0.752s	18h6m12.74s
	b20	8	32	15m6.045s	6m20.914s	0m4.988s	0m57.046s
	b21	8	32	15m11.598s	6m49.946s	0m5.389s	0m59.616s
	b22	8	32	15m24.620s	20m0.005s	2m41.473s	2m24.392s

[1] CNS, [2] x..x, [3] FAIL, [4] Equal [5] N/A

Contribution - Cute-Lock-Str

Cute-Lock-Str Against Structural and Dataflow Attacks

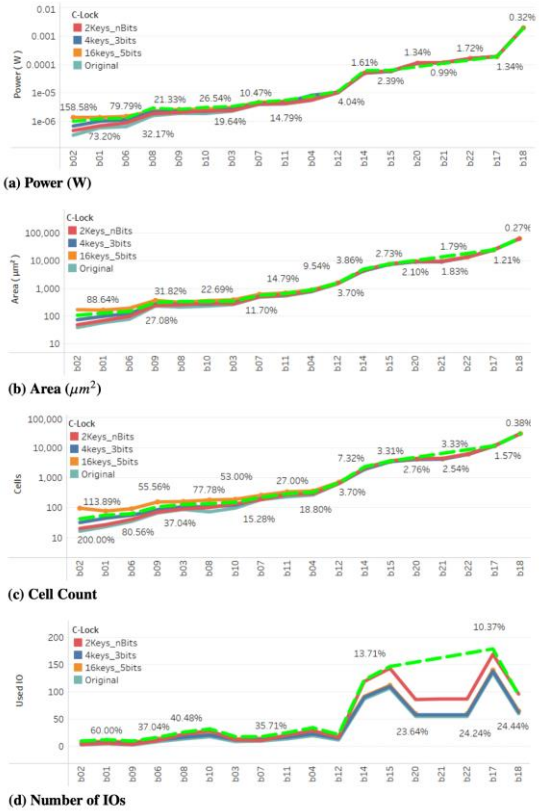
- We were able to get the NMI scores based on the original circuit.
- How similar the FF structure of the original circuit compared to the encrypted.
  - We got 0.45 percent average compared to the original circuit.
  - In the DANA paper, they got a 0.86 average.
- FALL structural attack was not able to get any of the key values: 0 candidates.

	DANA	FALL		
Circuit	NMI Score	Candidates	Keys	CPU Time (s)
b01	0.00	0	0	0.1
b02	0.00	0	0	0.1
b03	0.00	0	0	0.1
b04	0.00	0	0	0.1
b05	0.00	0	0	0.2
b06	0.00	0	0	0.1
b07	0.74	0	0	0.1
b08	0.99	0	0	0.1
b09	0.43	0	0	0.1
b10	0.00	0	0	0.1
b11	0.76	0	0	0.1
b12	0.99	0	0	0.1
b14	0.60	0	0	10.9
b15	0.89	0	0	16.8
b17	0.93	0	0	97.3
b18	0.93	0	0	1663.6
b19	0.50	0	0	3423.4
b20	0.56	0	0	23.4
b21	0.44	0	0	23.7
b22	0.39	0	0	46.4

Contribution - Cute-Lock-Str

Cute-Lock-Str Overhead

- Overhead values were calculated using the ITC 99 suite.
- 3 different configurations
  - 2keys_nBits
  - 4keys_3Bits
  - 16keys_5bits
- Compared to DK-Lock with configuration: n bits, 10 bits.
- Overhead becomes smaller as benchmarks become larger.



(a) Power (W)

(b) Area (μm^2)

(c) Cell Count

(d) Number of IOs

Conclusion - Summary

- **Cute-Lock-Beh multi-key logic locking solution:**
  - Evades SAT attacks.
  - Successfully locks circuits of any size.
  - Supports sequential locking at the HDL (behavioral) level.

- **Cute-Lock Str multi-key logic locking solution:**
  - Evades SAT attacks.
  - Prevents structural attacks and dataflow attacks.
  - Supports sequential locking at the netlist (structural) level.
  - Has low overhead.

Conclusion - Future Works

- **Adopt multi-key locking to combinational circuits using input encoding:**
  - K. Lopez and A. Rezaei, “K-Gate Lock: Multi-Key Logic Locking Using Input Encoding Against Oracle-Guided Attacks,” Recently presented in 30th Asia and South Pacific Design Automation Conference (ASP-DAC), 2025, Japan.

- **Broader applications of multi-key solutions beyond logic locking:**
  - Hardware trojan detection and mitigation

Conclusion - Acknowledgment

- This work is supported by the National Science Foundation under Award No. 2245247.

Award No. 2245247

- The source codes and created benchmarks are publicly available on our GitHub repository.

https://github.com/cars-lab-repo/Cute-Lock

Thank You!

Kevin Lopez

Amin Rezaei

https://github.com/cars-lab-repo/Cute-Lock