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

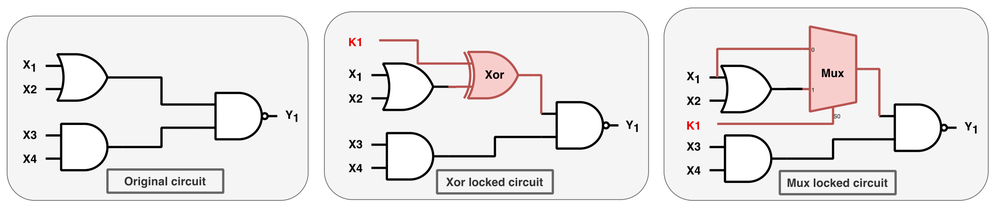
Kevin Lopez and Amin Rezaei

Introduction

The outsourcing of semiconductor manufacturing raises security risks, such as piracy and overproduction of hardware intellectual property. To overcome this challenge, logic locking has emerged to lock a given circuit using additional key bits. While single-key logic locking approaches have demonstrated serious vulnerability to a wide range of attacks, multikey solutions, if carefully designed, can provide a reliable defense against not only oracle-guided logic attacks but also removal and dataflow attacks. In this paper, using time-based keys, we propose, implement, and evaluate a family of secure multi-key logic locking algorithms called Cute-Lock that can be applied both in RTL-level behavioral and netlist-level structural representations of sequential circuits. Our extensive experimental results under a diverse range of attacks confirm that, compared to vulnerable state-of-the-art methods, employing the Cute-Lock family drives attacking attempts to a dead end without additional overhead.

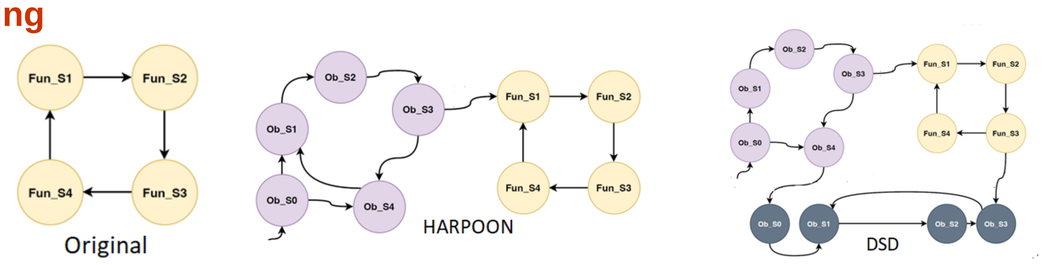
We present the following contributions.
• Proposing Cute-Lock-Beh, a **c**ounter-based m**u**l**t**i-k**e**y logic locking **beh**avioral solution to secure circuits against logic attacks in RTL-level representation;
• Proposing Cute-Lock-Str, a counter-based m**u**l**t**i-k**e**y logic locking **str**uctural solution to secure circuits against both logic and structural attacks in the gate-level netlist;

Logic Locking



Combinational Logic Locking

Two categories of combinational logic locking:
XOR-based[1] K1=0
MUX-based[2] K1=1

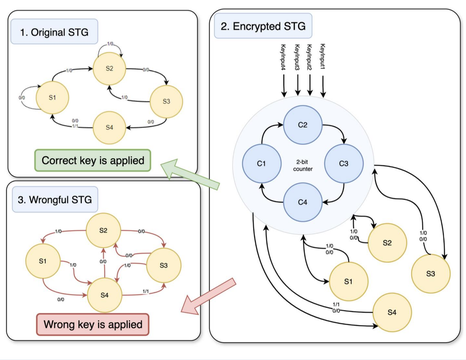


Sequential Logic Locking

Use state transition to lock the FSM.
HARPOON[3]: Need to traverse through obfuscation states.
DSD[4]: HARPOON + Adding black holes.

Cute-Lock-Beh

- Counter-based multi-key logic locking behavioral solution to secure circuits against logic attacks in behavioral representation.
- Sequential logic locking Solution aimed to protect against oracle-guided SAT attacks.



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
```

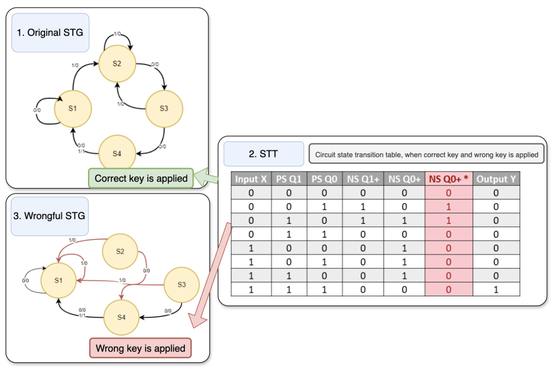
```
module simple (clk,rst, x1, pr_state);
input clk, rst, x1;
parameter s1=0, s2=1, s3=2;
output reg[2:0] pr_state;
reg[2:0] nx_state;

always@ (negedge clk or posedge rst )
        begin
        if ( rst == 1'b1 )
                pr_state <= s1;
        else
                pr_state <= nx_state;
end

always@ ( pr_state or x1 )
        begin
        case ( pr_state )
        s1:  nx_state = ( x1 == 1'b0 ) ? s2 : s1;
        s2 : nx_state = ( x1 == 1'b1 ) ? s3 : s2;
        s3 : nx_state = ( x1 == 1'b0 ) ? s1 : s3;
        default: nx_state = s1;
        endcase
        end
endmodule
```

Cute-Lock-Str

- Counter-based multi-key logic locking structural 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.

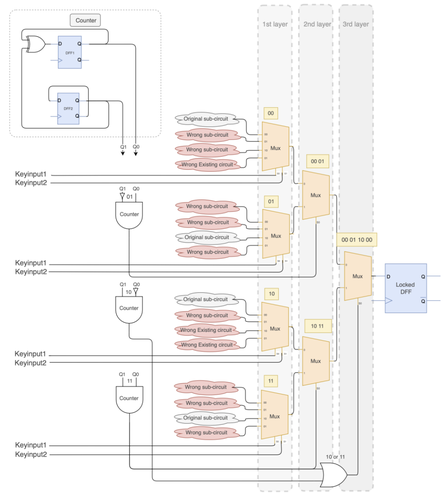


Input X	PS Q1	PS Q0	NS Q1+	NS Q0+	NS Q0+ *	Output Y
0	0	0	0	0	0	0
0	0	1	1	0	1	0
0	1	0	1	1	1	0
0	1	1	0	0	0	0
1	0	0	0	1	0	0
1	0	1	0	1	0	0
1	1	0	0	1	0	0
1	1	1	0	0	0	1

Cute-Lock-Str Implementation

MUX Tree - Locking a Flip-Flop

MUX trees are used to lock flip-flops 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.



Experimental Results

Cute-Lock-Beh 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_s)	BBO	INT	KC2
Small	bcomp	6	18	6m25.446s	0m0.885s	0m1.030s
	bech	6	18	6m4.845s	0m0.772s	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	0m0.772s	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	10m3.736s	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	0m0.608s	0m0.809s
	codec	2	4	15m2.282s	0m2.252s	0m2.032s
	codec1_2	9	28	15m4.756s	0m3.768s	0m4.005s
	cow	6	49	15m7.930s	0m1.325s	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	0m40.366s	1m9.523s
	bulln	20	61	15m23.190s	1m19.553s	8m7.918s
	camel	19	59	16m29.513s	3m38.506s	14m34.180s
	exam	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

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_s)	BBO	INT	KC2	RANE
ISCAS'89 [63]	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	1m57.663s	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	0m56.983s	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	0m0.230s	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	1h27m50.41s
	b14	8	32	15m3.473s	1m55.654s	0m1.083s	19m39.38s
	b15	16	36	14m3.219s	20m0.006s	0m4.096s	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	0m0.874s	0m0.752s	180m12.74s
	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

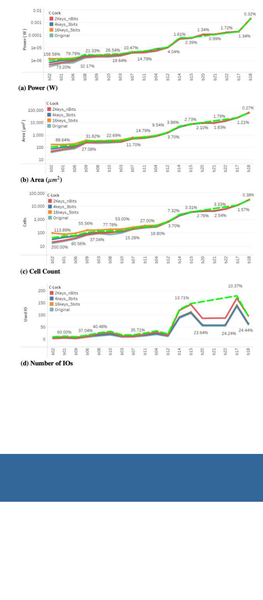
Cute-Lock-Str Against Structural and Dataflow Attacks

- 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

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.



Conclusion

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.

In addition, **Cute-Lock Str** is a multi-key logic locking solution that:
- Prevents structural attacks and dataflow attacks.
- Supports sequential locking at the netlist (structural) level.
- Has low overhead.

References

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.
3 - 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.
4 - 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.