K-Gate Lock: Multi-Key Logic Locking Using Input Encoding Against Oracle-Guided Attacks

30th Asia and South Pacific Design Automation Conference
Jan. 20-23, 2025
Tokyo Odaiba Miraikan, Japan

Kevin Lopez and Amin Rezaei

Computer Architecture, Reliability, and Security Laboratory (CARS-Lab)

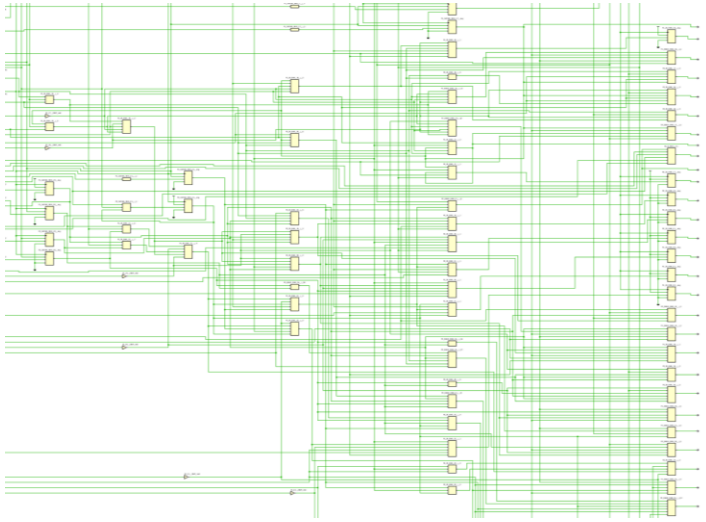
Contents

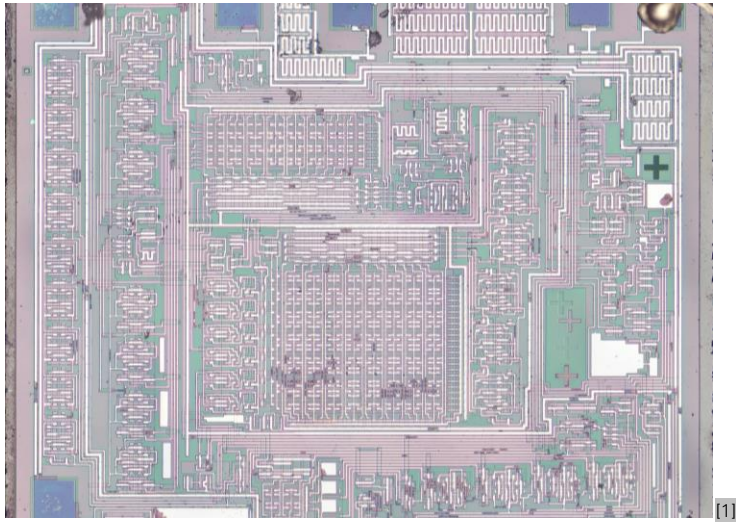
Fabless Manufacturing

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

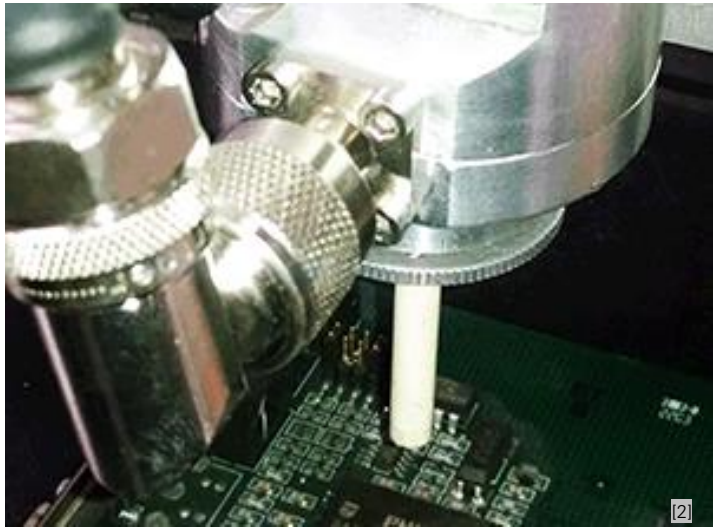
Threats

- Reverse Engineering
- Piracy
- Overproduction
- Counterfeiting

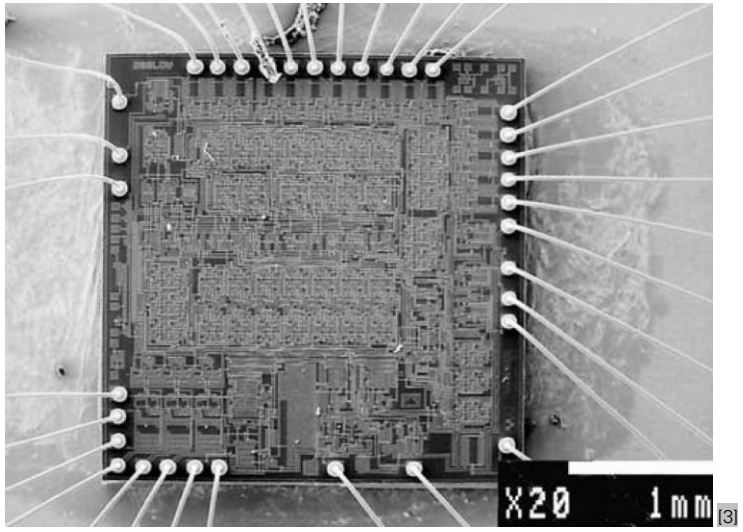




[1]



[2]



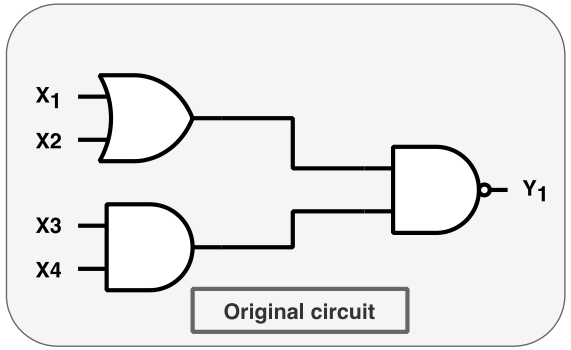
[3]

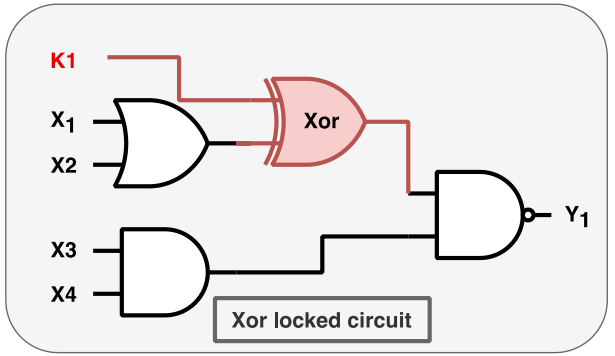
1 - http://www.righto.com/2021/12/reverse-engineering-tiny-1980s-chip.html
2 - https://www.ic-crack.com/reverse-engineering-ic-atmega644a-firmware/
3 - https://www.ic-crack.com/brute-force-chip-break-application/

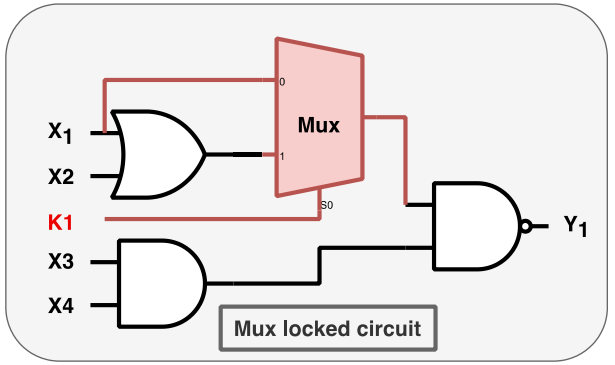
Combinational Logic Locking

- Two categories of combinational logic locking
  - XOR-based
  - MUX-based

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







Oracle Guided Attacks

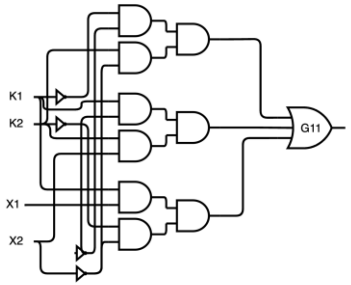
Goal: Determine the secret key used for logic locking

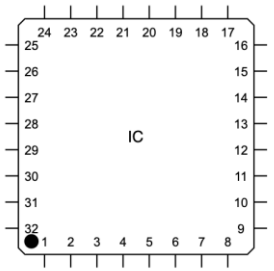
Attacker has:

- Locked netlist
- Functional IC

Attacker does:

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





The SAT Attack

- This attack came out in 2015, breaking almost all the locking solutions.
- Evaluating the security of logic encryption algorithms by P. Subramanyan, S. Ray and S. Malik
- 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

Advanced Logic Locking Solutions

- Anti-SAT, SARLock, TTLock
- Increase the SAT attack to exponential time.
  - SAT can only eliminate one wrong key per DIP
- There are many attacks against advanced single key logic locking

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

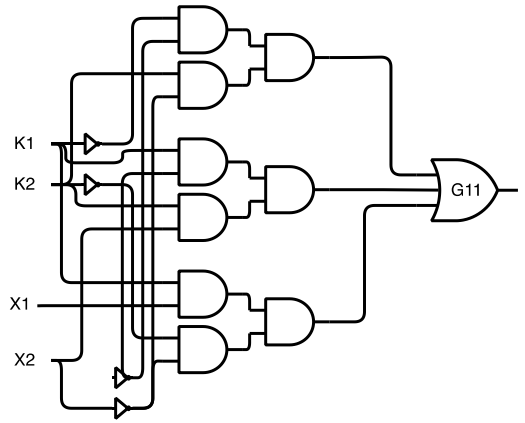
Motivation for Multi-Key Logic Locking

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

Multi-Key Logic Locking Solution 1

- Brute Force Input Encoding
  - Expands the truth table and inserts input key bits in every positive input combination.
  - It is not feasible for larger circuits because it takes exponential time
    - Python implementation fails to lock medium-sized circuits.
  - $O(2^n)$, where n is the number of inputs to the circuit.
  - C432 circuit in ISCAS 85 produces a truth table of 68,719 million rows (64 GB RAM was not enough)
    - 36 inputs

Inputs					Output
X_1	X_2	..	..	X_n	Y
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..
X_1^n	..	..	..	X_n^n	Y^n



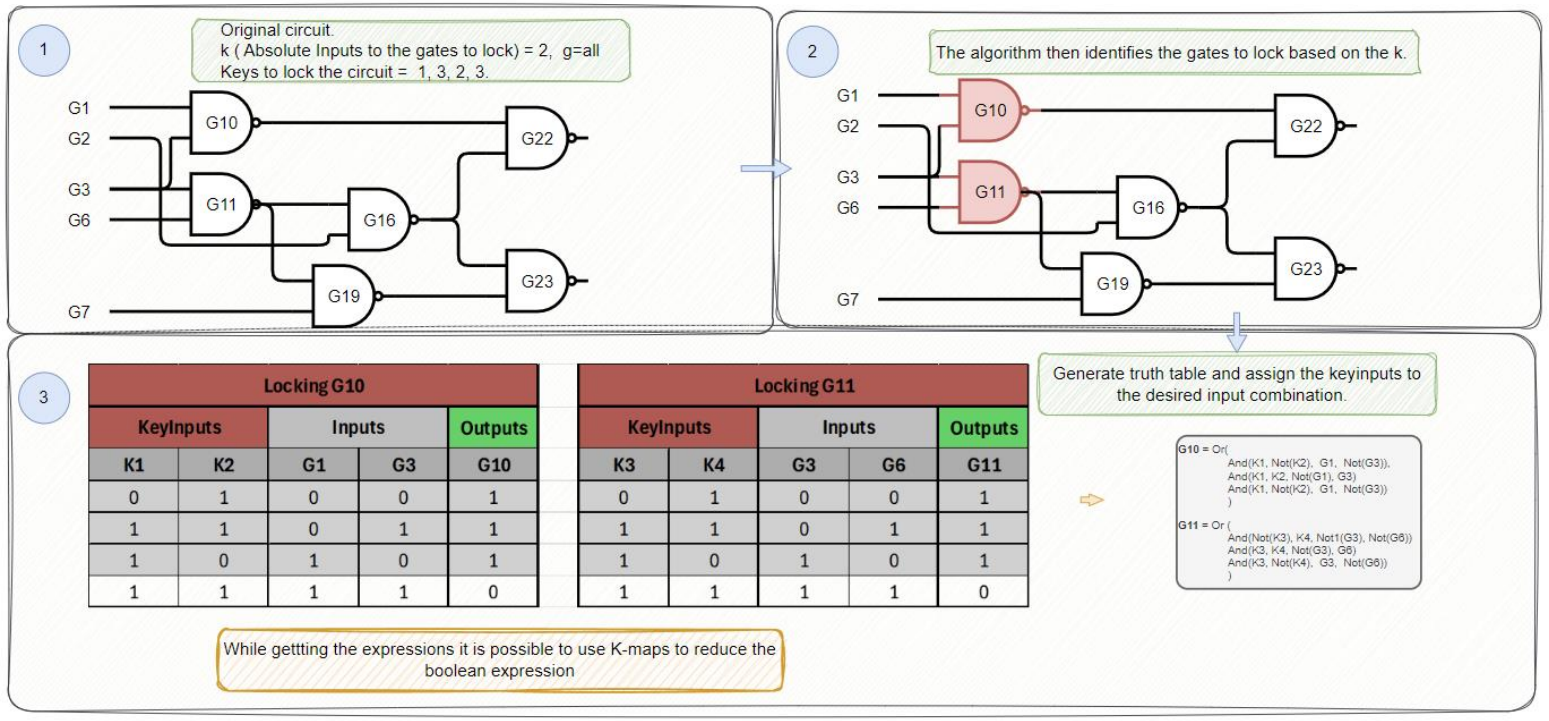
2 input circuit (NAND gate)

Multi-Key Logic Locking Solution 2

- K-Gate Lock
  - We expand the truth table for portions (gate or subcircuit) instead of the whole circuit.
  - Expands the truth table for a given level k gate and inserts input key bits in every positive input combination.
  - To limit the number of gates to lock, we added g, an upper bound on the gates locked.
  - Much lower time complexity, it can lock every circuit.
    - $O\left(\min\left(\frac{n}{k}, g\right) \times 2^k\right)$ vs 2^n
      - Where k is the number of inputs to the gate,
      - g is the maximum number of gates to lock.
      - n is the number of inputs to the circuit.

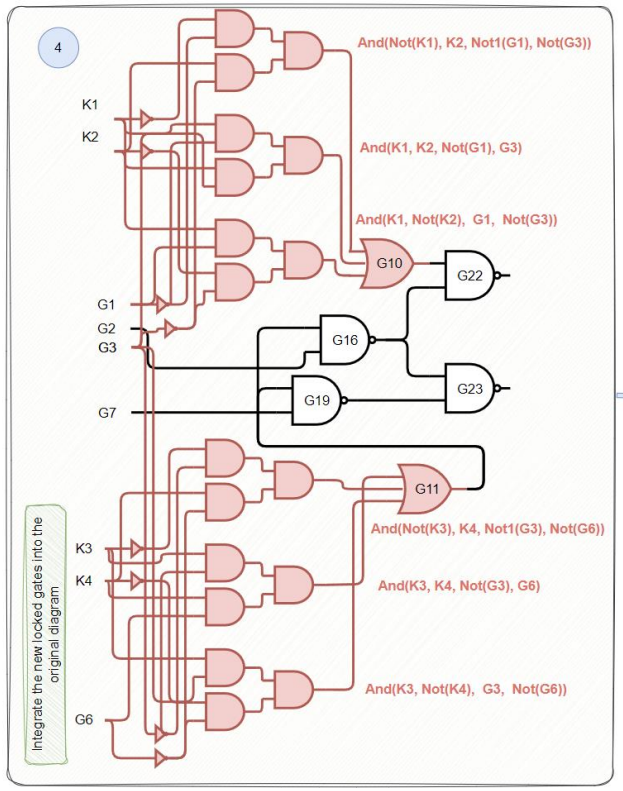
K-Gate Logic Locking

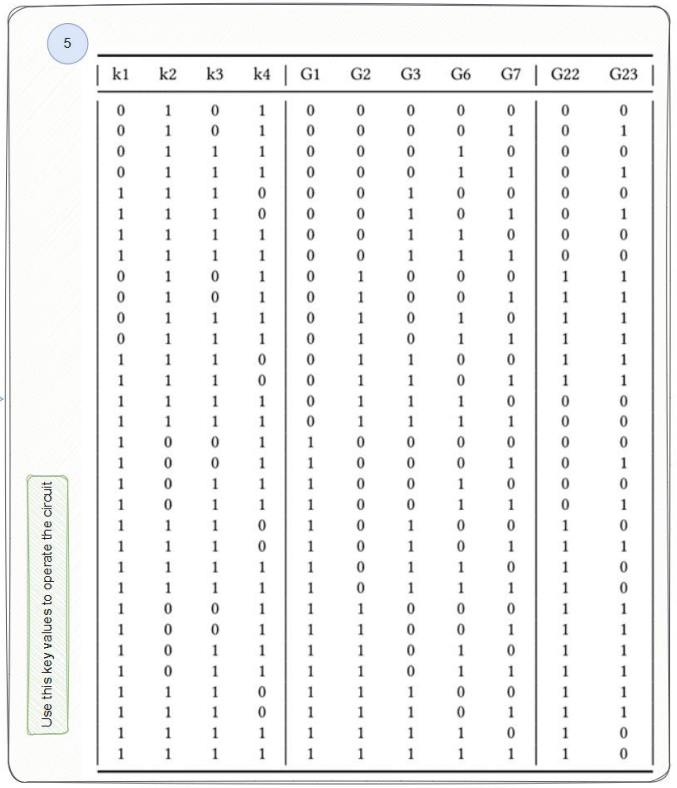
- Step 1 – Receive the inputs
  - $f(x)$ Original circuit
  - k = 2 (absolute inputs)
  - g = all (maximum number of gates)
  - $keys$ = 1,3,2,3
    - 01
    - 11
    - 10
    - 11
- Step 2 – Find all gates with k inputs
- Step 3 – Generate truth tables and insert key inputs



K-Gate Logic Locking

- Step 4 – Integrate the locked gates into the circuit.
- Step 5 – To use the circuit, use the table shown.





k1	k2	k3	k4	G1	G2	G3	G6	G7	G22	G23
0	1	0	1	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	1	0	1
0	1	1	1	0	0	0	1	0	0	0
0	1	1	1	0	0	0	1	1	0	1
1	1	1	0	0	0	1	0	0	0	0
1	1	1	0	0	0	1	0	1	0	1
1	1	1	1	0	0	1	1	0	0	0
1	1	1	1	0	0	1	1	1	0	0
0	1	0	1	0	1	0	0	0	1	1
0	1	0	1	0	1	0	0	1	1	1
0	1	1	1	0	1	0	1	0	1	1
0	1	1	1	0	1	0	1	1	1	1
1	1	1	0	0	1	1	0	0	1	1
1	1	1	0	0	1	1	0	1	1	1
1	1	1	1	0	1	1	1	0	0	0
1	1	1	1	0	1	1	1	1	0	0
1	0	0	1	1	0	0	0	0	0	0
1	0	0	1	1	0	0	0	1	0	1
1	0	1	1	1	0	0	1	0	0	0
1	0	1	1	1	0	0	1	1	0	1
1	1	1	0	1	0	1	0	0	1	0
1	1	1	0	1	0	1	0	1	1	1
1	1	1	1	1	0	1	1	0	1	0
1	1	1	1	1	0	1	1	1	1	0
1	0	0	1	1	1	0	0	0	1	1
1	0	0	1	1	1	0	0	1	1	1
1	0	1	1	1	1	0	1	0	1	1
1	0	1	1	1	1	0	1	1	1	1
1	1	1	0	1	1	1	0	0	1	1
1	1	1	0	1	1	1	0	1	1	1
1	1	1	1	1	1	1	1	0	1	0
1	1	1	1	1	1	1	1	1	1	0

Pseudo Implementation

- The implementation of K-Gate logic locking uses 2 functions
  - The main algorithm that locks the whole circuits
  - A function used to lock a gate.
    - Goes through each output and input combination and locks it with the provided keys.

Algorithm 1 Circuit Locking

```
1: Input: Circuit f(x), keys, g(max gates to lock), level k
   (default k = 2)
2: Output: Locked circuit h(x, k)
3: gates_k_inputs ← get_gates_with_k_inputs(f(x), k, g)

4: h(x, k) ← f(x)
5: for each gate in gates_k_inputs do
6:     locked_gate ← lock_gate_with_key(gate, keys)
7:     h(x, k).replace(gate, locked_gate)
8: end for
9: return h(x, k)
```

Algorithm 2 Gate Locking

```
1: Function lock_gate_with_key(gate, keys)
2: Input: gate, keys
3: Output: locked_gate
4: table ← generate_logic_table(gate.inputs, gate.output)

5: locked_gate ← ()
6: for each (inputs, output) in table do
7:     for each key in keys do
8:         logic_combination ← ((key + inputs) → output)
9:         locked_gate ← logic_combination
10:    end for
11: end for
12: return locked_gate
13: End Function
```

Oracle Guided Attack Results

- Algorithm validation: Constant keys - SAT attacks can identify the correct key and confirm that same circuit operates as the oracle.
- Attack 1: 3 keys, 3 bit – SAT attack fails to find keys
- Attack 2: n keys, k bits – SAT attack fails to find keys

Algorithm Validation

Benchmark	Gates	Time (S)	Reported Key
iscas85/c1355	3	0.08091	101101101
iscas85/c17	2	0.01133	101101
iscas85/c1908	3	0.1377	101101101
iscas85/c3540	3	1.576	101101101
iscas85/c432	3	0.03317	101101101
iscas85/c499	3	0.04950	101101101
iscas85/c5315	3	0.9743	101101101
iscas85/c7552	3	1.942	101101101
iscas85/c880	3	0.06435	101101101

Attack 1

Benchmark	Gates	NEOS		RANE	
		Reported Key	Time (S)	Key Found	Time (S)
iscas85/c1355	3	100100100	0.0941482	CNS	1.27
iscas85/c17	2	100101	0.0132606	011100	0.08
iscas85/c1908	3	101011011	0.179116	101100101	0.69
iscas85/c3540	3	011011011	1.95018	011011011	0.75
iscas85/c432	3	011100100	0.0301723	011100100	0.14
iscas85/c499	3	101101101	0.052921	100100101	0.36
iscas85/c5315	3	011011101	0.462587	011011100	1.20
iscas85/c6288	3	100100101	0.383951	101101100	2.86
iscas85/c7552	3	011100100	1.95771	011011011	1.39
iscas85/c880	3	101011100	0.0654231	101011100	0.25
iscas89/s1196	3	011101011	0.081063	011101011	0.48
iscas89/s15850	3	110011011	0.398465	000011101	55.53
iscas89/s5378	3	CNS	0.398465	010000011	11.03
iscas89/s641	3	111000011	0.040502	000000010	4.71
iscas89/s713	3	101100101	0.040502	000000101	4.22
iscas89/s832	3	010010010	0.342658	010010011	1.53s

Attack 2

Benchmark	Key Size	NEOS Time (S)	RANE Time (S)
iscas85/c1355	40	0.104842	3.08
iscas85/c17	2	0.0322511	0.05
iscas85/c1908	30	0.12339	0.47
iscas85/c3540	50	0.149944	0.52
iscas85/c432	30	0.164	0.14
iscas85/c499	40	0.151909	0.45
iscas85/c5315	170	2.14995	3.46
iscas85/c6288	30	0.894312	2.77
iscas85/c7552	200	1.90459	1.32
iscas85/c880	60	0.123312	0.22
EPFL/adder	60	0.18	FAIL
EPFL/bar	50	0.69	FAIL
EPFL/div	50	461.62	FAIL
EPFL/hyp	60	688.26	FAIL
EPFL/log2	20	25.73	FAIL
EPFL/max	80	1.46	FAIL
EPFL/multiplier	50	550.99	FAIL
EPFL/sin	20	2.19	FAIL
EPFL/sqrt	50	7.88	FAIL
EPFL/adder_depth_2023	60	0.9180	FAIL
EPFL/arbiter_depth_2022	60	0.9101	FAIL
EPFL/bar_depth_2015	50	12.12	FAIL
EPFL/cavlc_depth_2022	10	0.2943	FAIL
EPFL/div_depth_2023	50	152.8	FAIL
EPFL/adder_size_2022	60	1.224	FAIL
EPFL/arbiter_size_2023	60	0.2725	FAIL
EPFL/bar_size_2015	50	0.2725	FAIL
EPFL/cavlc_size_2023	50	0.63578	FAIL
EPFL/div_size_2023	50	83.98	FAIL

Possible SAT Attack If Knows About K-Gate lock

- Future SAT attacks would need to explore the keys for every input combination as follows:
  - I) All input combinations: 2^n possibilities (where n is the number of inputs)
  - II) All potential values for each key: 2^m possibilities (where m is the total number of key bits).
    - Each key is made up of the sum of all gate keys $m = \sum_{i=1}^{g} |key(i)|$
  - SAT-based attacks are forced to perform a brute-force search for every key, resulting in a time complexity of $O(2^m \times 2^n) \Rightarrow O(2^{m+n})$

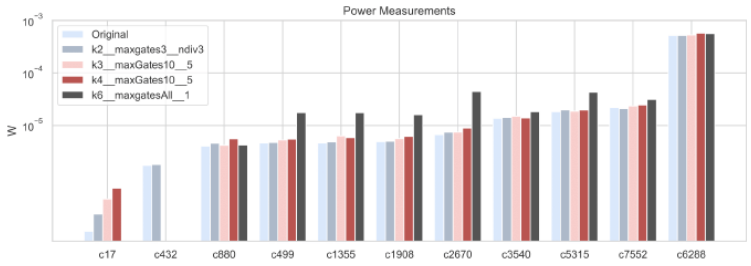
Number of Keys for the Height of 2^n and g Gates

	k_1^1	...	$k_1^{\|key_1\|}$	....	k_g^1	...	$k_g^{\|key_g\|}$	inp_1	...	inp_n
key 1	x	...	x	....	x	...	x	0	...	0
key 2	x	...	x	....	x	...	x	0	...	1
key 3	x	...	x	....	x	...	x	0	...	0
.	.	...	.	....	.	...	.	.	...	.
.	.	...	.	....	.	...	.	.	...	.
key 2^n	x	...	x	....	x	...	x	1	...	1

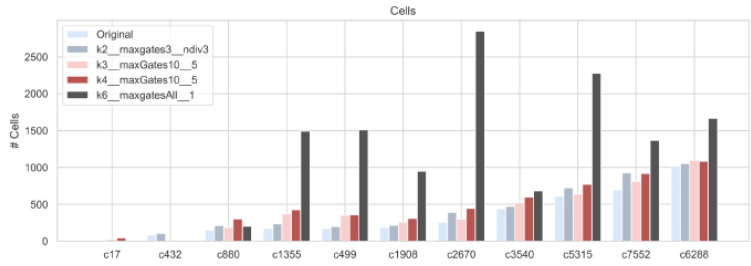
Overhead Measurements

- Cadence Genus - 45nm library
- The highest overhead in terms of power is ~26%, and the Area is ~198%.
- The lowest is around ~0.45% percent for power and ~19% for the Area.
- The IO increased based on the added keys.

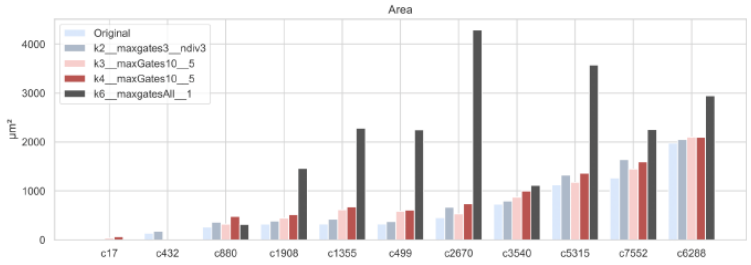
Test Name	Power %	Cells %	Area %	I/O %
K=2, g=3, key bit=n/3	0.45	20.40	18.63	64.84
K=3, g=10, key bit=5 bits	2.93	23.07	19.52	33.14
K=4, g=10, key bit=5 bits	10.45	41.33	33.95	33.14
K=6, g=All, key bit=1 bit	25.62	246.33	197.89	35.23



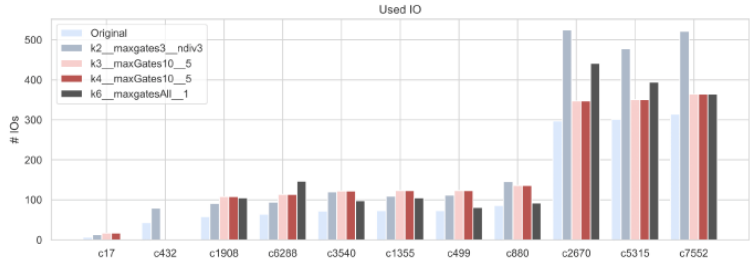
(a) Power (W)



(b) Number of Cells



(c) Area (μm^2)



(d) Number of IOs

Summary

- K-Gate Logic locking is a solution that:
  - Resists SAT attacks. Has the potential to resist white-box (probing) attacks too.
  - Successfully locks circuits of any size.
  - Depends only on combinational logic.
  - Maintains a reasonable overhead.

Future Work

- Expand multi-key locking to sequential circuits using time base keys.
  - Stay tuned: K. Lopez and A. Rezaei, “Cute-Lock: Behavioral and Structural Multi-Key Logic Locking Using Time Base Keys,” Accepted in Proceedings of 28th Design, Automation & Test in Europe Conference & Exhibition (DATE), 2025, France.

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/KGL

Thank you

Kevin Lopez

Amin Rezaei

https://github.com/cars-lab-repo/KGL