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

Kevin Lopez
Computer Engineering & Computer Science Department
California State University, Long Beach
Kevin.LopezChavez01@student.csulb.edu

Amin Rezaei
Computer Engineering & Computer Science Department
California State University, Long Beach
Amin.Rezaei@csulb.edu

ABSTRACT

Logic locking has emerged to prevent piracy and overproduction of integrated circuits ever since the split of the design house and manufacturing foundry was established. While there has been a lot of research using a single global key to lock the circuit, even the most sophisticated single-key locking methods have been shown to be vulnerable to powerful SAT-based oracle-guided attacks that can extract the correct key with the help of an activated chip bought off the market and the locked netlist leaked from the untrusted foundry. To address this challenge, we propose, implement, and evaluate a novel logic locking method called K-Gate Lock that encodes input patterns using multiple keys that are applied to one set of key inputs at different operational times. Our comprehensive experimental results confirm that using multiple keys will make the circuit secure against oracle-guided attacks and increase attacker efforts to an exponentially time-consuming brute force search. K-Gate Lock has reasonable power and performance overheads, making it a practical solution for real-world hardware intellectual property protection.

CCS CONCEPTS

• **Security and privacy → Security in hardware**.

KEYWORDS

Logic Locking, Logic Encryption, Logic Obfuscation, SAT Attack, Multi-Key Locking, Dynamic Locking, Input Encoding

ACM Reference Format:
Kevin Lopez and Amin Rezaei . 2025. K-Gate Lock: Multi-Key Logic Locking Using Input Encoding Against Oracle-Guided Attacks. In *30th Asia and South Pacific Design Automation Conference (ASPDAC '25), January 20–23, 2025, Tokyo, Japan.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3658617.3697764

1 INTRODUCTION

In this split of design and manufacturing, one company designs the digital design, while another handles the physical fabrication of the Integrated Circuit (IC). While this separation of tasks poses a threat to chip security, logic locking [1, 2] has emerged as a promising solution to prevent piracy and overproduction of hardware Intellectual Properties (IPs). Formally speaking, logic locking is the process of adding additional inputs to an IC, called key bits, to prevent the correct operation of the IC when the incorrect key is provided to the circuit. Traditionally, locking has been done using only one global key, which made it susceptible to the SAT-based attack [3] that extracts the key using an oracle (i.e., a working chip bought off the market) and a locked netlist leaked from an untrustworthy foundry. While there have been attempts to reduce the success of the SAT-based attack to brute-force [4, 5], sophisticated attacks have been proposed to find out the correct key of these methods.

We believe that the vulnerabilities associated with a single static key can be effectively mitigated through multi-key logic locking. In this paper, we introduce an advanced multi-key approach called **K-Gate Lock** where the inputs of each gate are encoded with different key values. To activate the circuit correctly, these values must be provided in the specific sequence used during the encoding process. In this paper, we present the following contributions:

- Proposing a robust multi-key logic locking based on input encoding, implemented fully in combinational logic;
- Implementing an efficient algorithm to lock a circuit with multiple user-defined keys with tunable time complexity;
- Generating more than 40 benchmarks based on the proposed method, measuring the overhead, and evaluating its security against state-of-the-art oracle-guided attacks.

2 BACKGROUND AND RELATED WORK

In this section, we first consider the evolution of logic locking techniques as well as oracle-guided attacks, and then review the existing efforts in multi-key logic locking.

2.1 Logic Locking Techniques

Initial techniques of logic locking rely on single-key schemes, primarily employing XOR-based and MUX-based mechanisms [1, 2]. In XOR-based logic locking, the key bits are matched with random inverters and buffers. Then, the XOR gates controlled by key bits are used to replace selected buffers and inverters. Additionally, MUX-based logic locking selects random wires and substitutes them with 2-1 MUXs whose inputs are real signals and random dummy ones, and selectors are the key bits. However, advancements in SAT solvers have been utilized to expose vulnerabilities in these methods [3], leading to the development of more robust techniques such as Anti-SAT [4], SAR-Lock [5], TT-Lock [6], CAS-lock [7], BLE [8], DLE [9], Full-Lock [10], Cross-Lock [11], HLock [12], TraceLL [13], TriLock [14], and others [15–27] that increase the time complexity of attacks. Obfus-Lock [28] is proposed to leverage the skewness of nodes to construct a locked circuit and obfuscate the circuit using re-write rules. Furthermore, a theoretical method has been proposed to achieve both high query complexity and key error rates

ASPDAC '25, January 20–23, 2025, Tokyo, Japan

ACM ISBN 979-8-4007-0635-6/25/01
https://doi.org/10.1145/3658617.3697764

based on quasi-universal circuits, including convolutional biased target circuits [29]. In addition, recently, a sequential obfuscation solution called STATION [30] has been proposed by leveraging disjoint encoding and combinational logic locking techniques. A comprehensive overhead and security analysis of state-of-the-art logic locking methods is also done in [31]. *Despite the mentioned efforts, single-key solutions remain susceptible once the key is compromised, endangering the entire security of the hardware IPs.*

2.2 Oracle-Guided Attacks

Boolean SAT solvers are used to reveal the correct key of logic-locked circuits using an oracle (i.e., an activated IC bought off the market) and a locked netlist to prune out the wrong key values [32–35]. The SAT-based attack [3] uses Distinguishing Input Patterns (DIPs) that are specifically designed to exploit the discrepancies between the locked circuit and the oracle by targeting and identifying incorrect key values. The more incorrect key values the SAT solver eliminates in one iteration, the faster the attack can find the correct key. Then on, each attack has been strategically designed to target a specific defense mechanism; for example, Double DIP [36] is used for attacking ICs locked with SAR-Lock [5], where using two DIPs instead of one helps find the correct key faster. AppSAT [37] uses an approximate flow to find the probably-approximate-correct key in Anti-SAT [4] method. Fa-SAT [38] inserts a single stuck-at fault at each signal of the locked circuit iteratively to find the correct key of BLE [8]. *The assumption in all the above attacks is that there is a single static key in the logic-locked circuit to be deciphered.*

2.3 Multi-Key Approaches

Recent works have brought the possibility of multi-key solutions. Specifically, DK-Lock [39] is a sequential locking method where one must provide two keys to a circuit; the first key is the activation key, which must be provided for a constant amount of time to activate the circuit, and then a functional key right after. DK-Lock may be susceptible to unrolling attacks [32] that can expand the key size to reverse the method back to a single-key solution. SLED [40] is another multi-key sequential solution but requires latches that operate on a clock, introducing additional complexity for combinational circuits. In addition, it depends on a seed value (i.e., a primary key) to operate, which can eventually be reduced to a single-key model since the attacker only needs to find out the seed value. *Both of the mentioned multi-key logic locking methods may still be reverted back to a single-key model and thus susceptible to traditional SAT-based oracle-guided attacks. In addition, they depend on sequential components to be implemented.*

Another multi-key logic locking solution, Gate-Lock [41], uses an approach focused on locking gates, resulting in circuits that are resilient to SAT attacks. In Gate-Lock, the truth table has a height of 2^{n+k} while our proposed method maintains the same input size height of 2^n and only locks the outcomes within the truth table that are true. This allowed us to implement a more efficient algorithm.

3 MULTI-KEY LOGIC LOCKING

In this section, after explaining the terminology, we discuss our proposed methods of locking a circuit with multiple keys; the first one locks the whole circuit, and it needs to generate a truth table for all the input combinations on the circuit, which may not be efficient in terms of space and time complexity. The second method, called **K-Gate Lock**, is a derivation but more optimized than the first one to focus on encoding the input combinations of the gates with specific key values. We also thoroughly discuss the implementation of the **K-Gate Lock** and evaluate the theoretical time complexity for any future oracle-guided attack to find the correct keys. An implementation example of **K-Gate Lock** on the c17 circuit from ISCAS 85 benchmarks [42] is shown in Figure 1. You may refer to Table 1 which contains the sequence of keys necessary to operate this locked circuit.

3.1 Terminology

In the context of **K-Gate Lock**, it is crucial to understand the terminology used to describe the various components and concepts:

n: The total number of inputs to the original circuit.
g: The maximum number of gates to be locked within a circuit.
k: The number of inputs to a gate is often called the level of locking.
gate key: Each gate in a locked circuit has a specific key that controls its operation based on the input combination.
key bit: The individual binary elements that constitute a key.
m: The total number of bits in a key, aggregated from all key bits associated with each locked gate.
keys: Our approach uses keys derived from gate key combinations, with the specific key depending on the input.

3.2 Locking the Whole Circuit

The brute force method of locking a combinational circuit using multiple keys requires expanding the logic table of all the circuit's possible input/output combinations. Multiple keys can be inserted into each input/output combination when all the inputs and outputs are expanded. The truth table will maintain a size of 2^n since it does not create every combination of keys; it only adds the desired keys to an input combination. This brute force approach to locking a whole circuit is impractical because it would lead to an exponential-size truth table, and the implementation of locking a circuit would be time-consuming. For example, the C432 circuit in ISCAS 85 produces a truth table of 68,719 million rows, which is extremely large for one of the smallest benchmarks in ISCAS 85 suite. This motivated us to introduce **K-Gate Lock**.

3.3 K-Gate Lock Algorithm

K-Gate Lock operates by locking specific gates within the circuit rather than the entire IC. This method utilizes the truth table of a gate or a more complex expression (i.e., a deep gate), encoding key bits directly into it. To operate the circuit, the user must provide a combination of inputs along with the corresponding keys in the correct sequence.

Now, we discuss the steps for locking a circuit based on the circuit example in Figure 1, which is locked at $k = 2$ with the key values of 01, 11, 10, and 11.

1. Inputs: The algorithm requires an input of the original circuit, unique keys, the maximum number of gates to lock (i.e., g), and a chosen level of gate locking (i.e., k). It is worth noting that the height of the truth table must be greater than the number of keys.

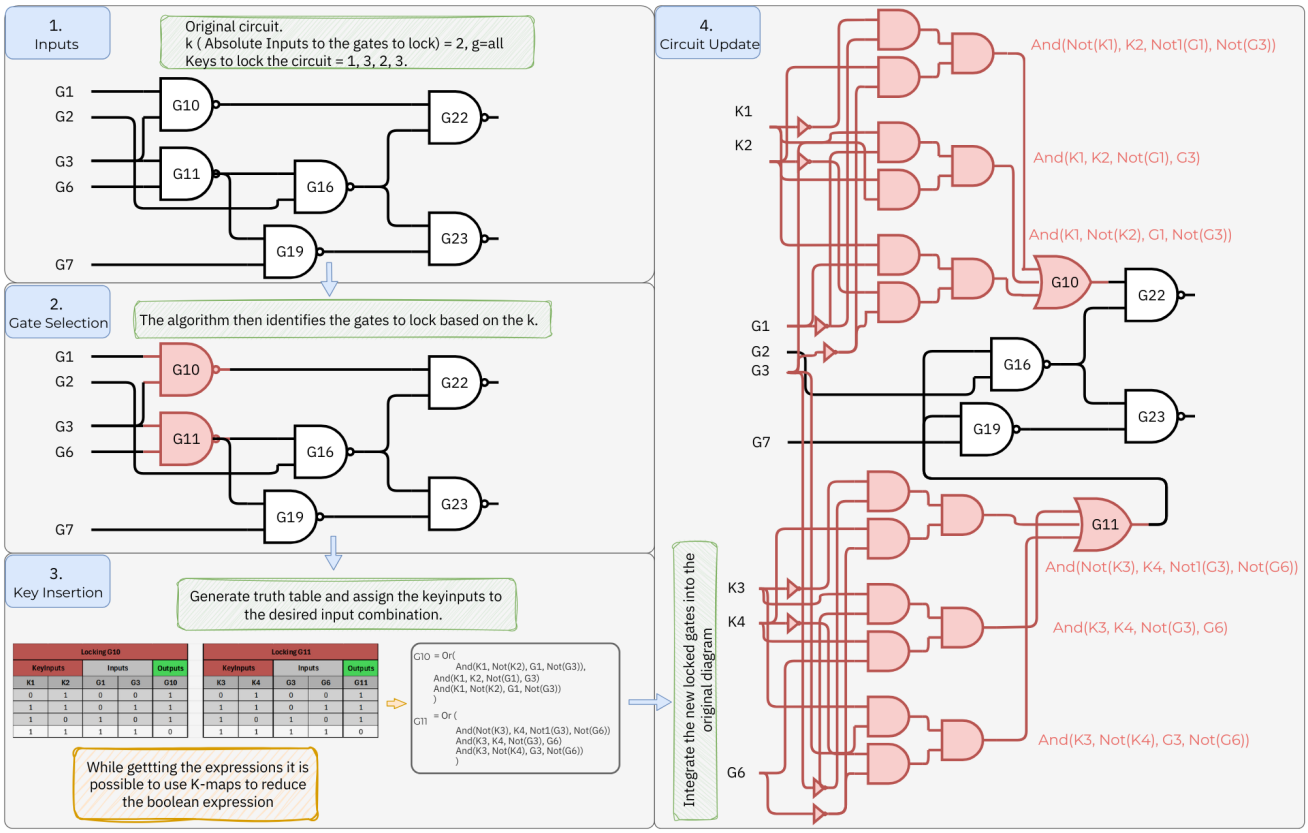


Figure 1: K-Gate Lock Example

2. Gate Selection: In the second phase of the process, the main task is determining which gates to lock, using the variable k as a guide. The gates needed to be locked are those that have the number of absolute inputs equal to the input value k. For example, while locking the C17 benchmark shown in Figure 1:

- G10 has two absolute inputs ($k = 2$), G1 and G3.
- G11 has two absolute inputs ($k = 2$), G3 and G6.
- G19 contains three absolute inputs ($k = 3$), which are G7, G3, and G6.
- G16 contains three absolute inputs ($k = 3$), which are G2, G3, and G6.

Since the initial constraint is to lock at $k = 2$, only G10 and G11 are selected.

3. Key Insertion: Up to this point, the algorithm has selected what gates to lock. In this step, the keys are inserted into the gate logic. For the C17 benchmark, as shown in Figure 1, truth tables are expanded for the expressions $\overline{(G1 \wedge G3)}$ and $\overline{(G3 \wedge G6)}$, and the key bits are added as defined in the initial contains.

For deeper gates (i.e., $k > 2$), it is necessary to have the absolute input to know when each key should be applied. It is also possible to maintain the original connections to the gate by using the inputs to the logic gate and adding them to the truth table. After the keys are added to the truth table, the expression can be extracted and simplified using any simplification method, like Karnaugh maps.

Table 1: Circuit Operation of Figure 1

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	1
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	0
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

4. Circuit Update: In this step, the algorithm updates the original circuit with the new locked gates. As shown by step 4 of Fig 1,

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: $gates\_k\_inputs \leftarrow get\_gates\_with\_k\_inputs(f(x), k, g)$
4: $h(x, k) \leftarrow f(x)$
5: **for** each gate in $gates\_k\_inputs$ **do**
6: $locked\_gate \leftarrow lock\_gate\_with\_key(gate, keys)$
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: $table \leftarrow generate\_logic\_table(gate.inputs, gate.output)$
5: $locked\_gate \leftarrow ()$
6: **for** each $(inputs, output)$ in $table$ **do**
7: **for** each key in $keys$ **do**
8: $logic\_combination \leftarrow ((key + inputs) \rightarrow output)$
9: $locked\_gate \leftarrow logic\_combination$
10: **end for**
11: **end for**
12: **return** $locked\_gate$
13: **End Function**

the locked G10 and G11 are placed instead of the original gates. Gates can be connected to the absolute inputs or the original inputs.

5. Circuit Operation: To operate the locked circuit, one must input the correct keys used to encode the gates. For the example shown in Figure 1, the truth table is shown in Table 1, where G1, G2, G3, G6, and G7 are the inputs and G22 and G23 are the outputs.

The **K-Gate lock** implementation is shown in Algorithms 1 and 2; it relies on two functions: one to lock an individual gate and another to lock the entire circuit. In other words, the main focus of Algorithm 1 is to find the gates to lock based on the given k and replace the gates with the locked ones generated from Algorithm 2. The main focus of Algorithm 2 is to lock an individual gate with a given set of dynamic keys. To attach the keys, it is necessary to generate the truth table of the gate's absolute inputs and then attach the given dynamic key.

3.4 Time Complexity

Now, we explain the time complexity of **K-Gate Lock**, which depends on the following:

Number of Gates to Lock ($\min\{\frac{n}{k}, g\}$)**:** This number represents how many gates the algorithm aims to lock, determined by g and k. The gates are chosen based on the level or absolute inputs they handle, as specified by k. This is demonstrated by line 3 of Algorithm 1, which identifies the gates to be locked.

Gate Locking Complexity (2^k)**:** The locking mechanism for each gate involves encoding the gate key into the gate's logic. This requires creating a truth table for the gate with all possible combinations of inputs, resulting in 2^k combinations. This step is shown in line 4 of Algorithm 2, which generates the truth table for the gate, and line 6, which encodes the gate key into the input combination.

Considering the above, the total time complexity can be represented as $O(\min\{\frac{n}{k}, g\} \times 2^k)$. It is practical to fix k at 2, leading to a linear time complexity for locking a circuit.

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

	k_1^1	...	$k_1^{\|key_1\|}$	....	k_g^1	...	$k_g^{\|keyg\|}$	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

3.5 Attack Analysis

For attack analysis, we show how the time complexity increases for SAT-based oracle-guided attacks to find the correct keys in a circuit locked with **K-Gate Lock**. We explore the idea of traditional single-key SAT solvers and future multi-key SAT attacks that are aware of the **K-Gate Lock** method.

Single-Key SAT Attack: Traditional SAT-based oracle-guided attacks are configured to find one correct key for a given circuit. Theoretically, such solvers are not suitable for finding multiple keys of the **K-Gate Lock** and in the best scenario, they will end up finding the first key of the sequence. We evaluate this with experimental results in Section 4.

Multi-Key SAT Attack: In this case, the attacker is aware the circuit is locked with **K-Gate lock** and needs to explore the keys for every input combination as follows:
I) All input combinations: 2^n possibilities (where n is the number of inputs) as shown by the height of Table 2.
II) All potential values for each key: 2^m possibilities (where m is the number of key bits). As shown in Table 2, the size of keys depends on the number of gates locked and the size of each gate key. The circuit designer has control over the total number of bits used and the number of gates. The m parameter that depends on the number of locked gates g and the key size for each gate is determined by the following equation:

$$m = \sum_{i=1}^{g} |key(i)|$$

Although current SAT-based attacks implement optimizations to prune several values of the global key, these optimizations cannot be applied to *K-Gate Lock* because it uses a different key at every DIP. Consequently, SAT-based attacks are forced to perform a brute-force search for every key, resulting in a time complexity of $O(2^{m+n})$.

4 EXPERIMENTAL RESULTS

We conduct experiments on a Windows 11 machine, which accesses Linux Ubuntu 22.04 via WSL2. The machine is a Ryzen 7940HS with 8 cores and 16 threads at 4.0 GHz and 32 GB of DDR5 RAM. The

Table 3: Attack Results on Benchmarks with Static Keys

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

source codes and created benchmarks of **K-Gate Lock** are publicly available on our GitHub repository[1].

4.1 Algorithm Validation

The algorithm validation is done in Python along with pyEDA [43], a tool used for electronic design automation. In this case, 2 gates of the c17 benchmark are locked with 4 different key combinations: 01, 11, 10, and 11. The truth tables are generated for the whole circuit shown in Table 1 along with adding the correct key values; in this case, the original circuit and the locked circuit output the same value when the correct keys are fed in.

Another set of tests is done using Netlist Encryption and Obfuscation Suite (NEOS) [44], where circuits from ISCAS 85 [42] are locked using multiple keys that remain the same value: 101, 101, 101. When multiple constant keys are provided, the SAT-based attack [3] is able to find out the correct key, meaning it is also possible to achieve the level of locking based on the traditional logic locking methods. The results are shown in Table 3.

4.2 Security Evaluation

The main objective of **K-Gate Lock** is to generate a locking mechanism that powerful SAT-based oracle-guided attacks will not be able to decrypt. We use combinational benchmarks of ISCAS 85 [42] and EPFL Benchmarks [45] as well as sequential benchmarks of ISCAS 89 [46]. Even though our proposed solution is based on combinational circuits, it is also possible to lock sequential circuits, locking portions of the circuit before including the flip-flops.

4.2.1 Three Dynamic Keys. Now, we perform SAT-based oracle-guided attacks against the benchmarks locked with **K-Gate Lock**. We use NEOS [44] and RANE [47] tools to run the attacks. The encryption for each circuit is done in *.bench* files with our Python implementation of **K-Gate Lock**.

The goal of this experiment is to demonstrate that with even a minimal number of keys, SAT-based attacks are unable to determine the correct keys. This limitation comes from their inherent design, which is to find only one key. The results are shown in Table 4 in which the circuits are locked with the following gate key values:

- b'011 - decimal value 3
- b'100 - decimal value 4
- b'101 - decimal value 5

[1]https://github.com/cars-lab-repo/KGL

Table 4: Attack Results on Benchmarks with 3 Small Dynamic Keys

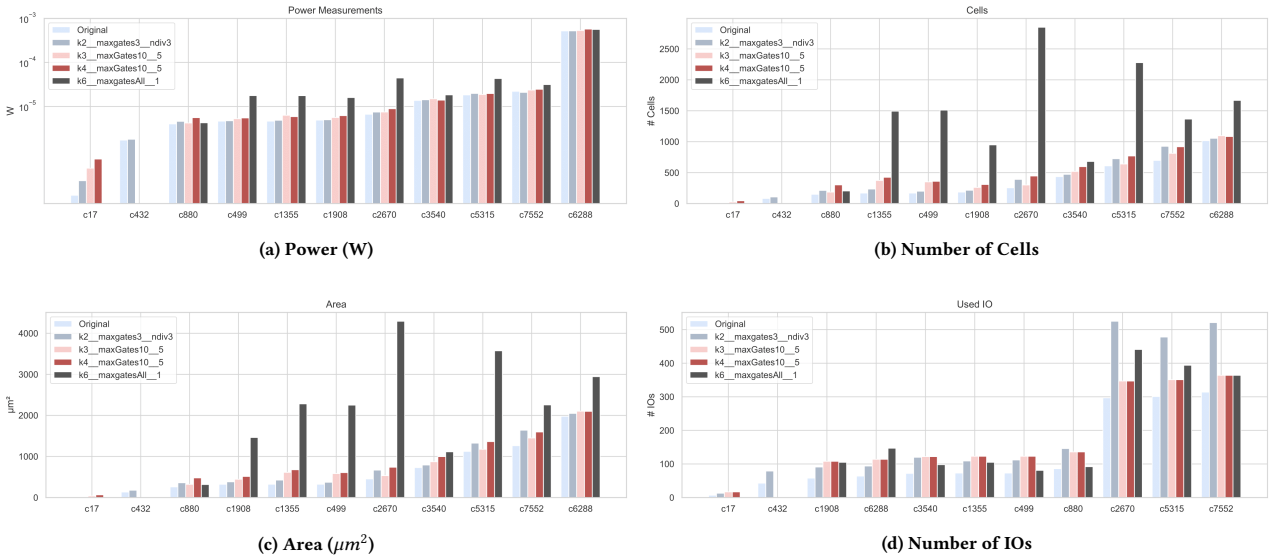
Benchmark	Gates	NEOS		RANE	
		Reported Key	Time (S)	Key Found	Time (S)
iscas85/c1355	3	100100100	0.0941482	CNS	1.27
iscas85/c17	2	100101	0.0132606	011100	0.08
iscas85/c1908	3	101011011	0.179116	101100101	0.69
iscas85/c3540	3	011011011	1.95018	011011011	0.75
iscas85/c432	3	011100100	0.0301723	011100100	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

Table 5: Attack Results on Benchmarks with Dynamic Key Sizes Scalable to the Input Sizes

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

In the tables, different colors are used to indicate specific conditions. The color light red[1] represents the "Condition Not Solvable" status. A deeper red[2] signifies a wrong key, while the darkest red[3] indicates that the attack failed. Finally, green[4] denotes that the correct key has been found.

[1] CNS, [2] x..x, [3] FAIL, [4] Equal



(a) Power (W) **(b) Number of Cells**

(c) Area (μm^2) **(d) Number of IOs**

Figure 2: Overhead Measurements

This experiment shows that SAT-based attacks are not able to find the sequences of the keys but only to find the first one. The key found by the SAT solvers is a combination of the keys for the locked gates, as we see in the first test for Table 4, the key value 100 is repeated three times, which is only one of the combinations of the keys that are used to lock the gates.

4.2.2 Dynamic Key based on Input Size. For the second security experiment shown in Table 5, we explore scaling key sizes to input sizes. Specifically, for ISCAS 85 benchmarks, the gate key size is calculated using $\lfloor \frac{n}{10} \rfloor$ with 10 gates locked, resulting in a floor value of 10 compared to the input size. In addition, we use a logarithmic scaling formula to deal with the high number of inputs in the EPFL benchmarks. While the EPFL benchmarks [45] are in *.blif* format, we use the ABC tool [48] to convert them to *.bench* files and perform the attack. The key values are generated randomly within the range dictated by the key input size for both benchmark suites. To simplify the testing process, we limit the number of gates locked to 10. This aims to approximate the size of the key as closely as possible to the input size. The experimental results highlight the challenges faced by SAT-based attacks in thwarting dynamic key locking.

4.3 Overhead Analysis

Now, we analyze the overhead of **K-Gate Lock**. The experimental setup utilizes Cadence Genus, using low mapping and optimization effort. Circuit locking is executed using *.bench* files and these are converted to Verilog with the ABC tool [48]. Power measurements, as shown in Figure 2, are performed on various ISCAS 85 [42] benchmarks with the following locking parameters.

- **Test Run 1** : $k = 2, g = 3, key\ bit = \frac{n}{3}$
- **Test Run 2** : $k = 3, g = 10, key\ bit = 5$
- **Test Run 3** : $k = 4, g = 10, key\ bit = 5$
- **Test Run 4** : $k = 6, g = all, key\ bit = 1$

Table 6: Average Overhead

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

As shown in Figure 2, power consumption does not increase much, and the area increases only slightly, correlated with the number of inputs and the gates locked. The missing data in the area, I/O measurements, along with power and temperature, means that the circuit is unable to lock any gates given the specific specifications, leading to the failure of the locked circuit creation. We calculated the average percentage increase using the sum of the whole test run and compared it with the original sum. These values are shown in Table 6. The highest jump in terms of power is 25.62% in *Test Run 4*, while the smallest jump is less than 1% in *Test Run 1*.

5 CONCLUSION

In this paper, we proposed a novel multi-key logic locking solution called **K-Gate Lock** that is based on input encoding and can be fully implemented using combinational logic without the need for state-holder components. Experimental results showed that **K-Gate Lock** is resilient against state-of-the-art SAT-based oracle-guided attacks with minimal overhead. This offers the potential of multi-key logic locking schemes for robust hardware IP protection with reasonable overhead.

ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation under Award No. 2245247.

REFERENCES

[1] Jarrod A. Roy, Farinaz Koushanfar, and Igor L. Markov. Ending piracy of integrated circuits. *Computer*, 43(10):30–38, 2010.

[2] Jeyavijayan Rajendran, Huan Zhang, Chi Zhang, Garrett S. Rose, Youngok Pino, Ozgur Sinanoglu, and Ramesh Karri. Fault analysis-based logic encryption. *IEEE Transactions on Computers*, 64(2):410–424, 2015.

[3] Pramod Subramanyan, Sayak Ray, and Sharad Malik. Evaluating the security of logic encryption algorithms. In *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 137–143, 2015.

[4] Yang Xie and Ankur Srivastava. Anti-sat: Mitigating sat attack on logic locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(2):199–207, 2019.

[5] Muhammad Yasin, Bodhisatwa Mazumdar, Jeyavijayan Rajendran, and Ozgur Sinanoglu. Sarlock: Sat attack resistant logic locking. In *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 236–241, 2016.

[6] Muhammad Yasin, Bodhisatwa Mazumdar, Jeyavijayan J V Rajendran, and Ozgur Sinanoglu. Ttlock: Tenacious and traceless logic locking. In *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 166–166, 2017.

[7] Bicky Shakya, Xiaolin Xu, Mark Tehranipoor, and Domenic Forte. Cas-lock: A security-corruptibility trade-off resilient logic locking scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(1):175–202, 2020.

[8] Amin Rezaei, Yuanqi Shen, and Hai Zhou. Rescuing logic encryption in post-sat era by locking & obfuscation. In *Design Automation & Test in Europe Conference & Exhibition (DATE)*, pages 13–18, 2020.

[9] Raheel Afsharmazayejani, Hossein Sayadi, and Amin Rezaei. Distributed logic encryption: Essential security requirements and low-overhead implementation. In *Proceedings of Great Lakes Symposium on VLSI (GLSVLSI)*, pages 127–131, 2022.

[10] Hadi Mardani Kamali, Kimia Zamiri Azar, Houman Homayoun, and Avesta Sasan. Full-lock: Hard distributions of sat instances for obfuscating circuits using fully configurable logic and routing blocks. In *Proceedings of Design Automation Conference (DAC)*, pages 1–6, 2019.

[11] Kaveh Shamsi, Meng Li, David Z. Pan, and Yier Jin. Cross-lock: Dense layout-level interconnect locking using cross-bar architectures. In *Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI)*, pages 147–152, 2018.

[12] Md Rafid Muttaki, Roshanak Mohammadivojdan, Mark Tehranipoor, and Farimah Farahmandi. Hlock: Locking ips at the high-level language. In *Design Automation Conference (DAC)*, pages 79–84, 2021.

[13] Michael Zuzak, Yuntao Liu, and Ankur Srivastava. Trace logic locking: Improving the parametric space of logic locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(8):1531–1544, 2021.

[14] Yuke Zhang, Yinghua Hu, Pierluigi Nuzzo, and Peter A. Beerel. Trilock: Ic protection with tunable corruptibility and resilience to sat and removal attacks. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1329–1334, 2022.

[15] Amin Rezaei, Ava Hedayatipour, Hossein Sayadi, Mehrdad Aliasgari, and Hai Zhou. Global attack and remedy on ic-specific logic encryption. In *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 145–148, 2022.

[16] Amin Rezaei, Jie Gu, and Hai Zhou. Hybrid memristor-cmos obfuscation against untrusted foundries. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 535–540, 2019.

[17] Amin Rezaei and Hai Zhou. Sequential logic encryption against model checking attack. In *Design Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1178–1181, 2021.

[18] Yeganeh Aghamohammadi and Amin Rezaei. Cola: Convolutional neural network model for secure low overhead logic locking assignment. In *Great Lakes Symposium on VLSI 2023 (GLSVLSI)*, pages 339–344, 2023.

[19] Amin Rezaei, You Li, Yuanqi Shen, Shuyu Kong, and Hai Zhou. Cycsat-unresolvable cyclic logic encryption using unreachable states. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, page 358–363, 2019.

[20] Amin Rezaei, Yuanqi Shen, Shuyu Kong, Jie Gu, and Hai Zhou. Cyclic locking and memristor-based obfuscation against cycsat and inside foundry attacks. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 85–90, 2018.

[21] Pei-Pei Chen, Xiang-Min Yang, Yu-Cheng He, Yung-Chih Chen, Yi-Ting Li, and Chun-Yao Wang. Looplock 3.0: A robust cyclic logic locking approach. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 594–599, 2024.

[22] Michaela Brunner, Tarik Ibrahimpasic, Bing Li, Grace Li Zhang, Ulf Schlichtmann, and Georg Sigl. Timing camouflage enabled state machine obfuscation. In *2022 IEEE Physical Assurance and Inspection of Electronics (PAINE)*, pages 1–7, 2022.

[23] Seetal Potluri, Aydin Aysu, and Akash Kumar. Seql: Secure scan-locking for ip protection. In *2020 21st International Symposium on Quality Electronic Design (ISQED)*, pages 7–13, 2020.

[24] Ujjwal Guin, Ziqi Zhou, and Adit Singh. Robust design-for-security architecture for enabling trust in ic manufacturing and test. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(5):818–830, 2018.

[25] Xiang-Min Yang, Pei-Pei Chen, Hsiao-Yu Chiang, Chia-Chun Lin, Yung-Chih Chen, and Chun-Yao Wang. Looplock 2.0: An enhanced cyclic logic locking approach. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 41(1):29–34, 2021.

[26] Subhajit Dutta Chowdhury, Gengyu Zhang, Yinghua Hu, and Pierluigi Nuzzo. Enhancing sat-attack resiliency and cost-effectiveness of reconfigurable-logic-based circuit obfuscation. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2021.

[27] Rajit Karmakar, Harshit Kumar, and Santanu Chattopadhyay. Efficient key-gate placement and dynamic scan obfuscation towards robust logic encryption. *IEEE Transactions on Emerging Topics in Computing*, 9(4):2109–2124, 2019.

[28] You Li, Guannan Zhao, Yunqi He, and Hai Zhou. Obfuslock: An efficient obfuscated locking framework for circuit ip protection†. In *2023 Design, Automation & Test in Europe Conference & Exhibition*, pages 1–6, 2023.

[29] Hai Zhou, Amin Rezaei, and Yuanqi Shen. Resolving the trilemma in logic encryption. In *International Conference on Computer Aided Design (ICCAD)*, pages 1–8, 2019.

[30] Zhaokun Han, Aneesh Dixit, Satwik Patnaik, and Jeyavijayan Rajendran. Station: State encoding-based attack-resilient sequential obfuscation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–1, 2024.

[31] Yeganeh Aghamohammadi and Amin Rezaei. Machine learning-based security evaluation and overhead analysis of logic locking. *Journal of Hardware and Systems Security*, 8:25–43, 2024.

[32] Amin Rezaei, Raheel Afsharmazayejani, and Jordan Maynard. Evaluating the security of efpga-based redaction algorithms. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–7, 2022.

[33] Yuanqi Shen, You Li, Shuyu Kong, Amin Rezaei, and Hai Zhou. Sigattack: New high-level sat-based attack on logic encryptions. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 940–943, 2019.

[34] Yuanqi Shen, Amin Rezaei, and Hai Zhou. Sat-based bit-flipping attack on logic encryptions. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 629–632, 2018.

[35] Yinghua Hu, Yuke Zhang, Kaixin Yang, Dake Chen, Peter A. Beerel, and Pierluigi Nuzzo. Fun-sat: Functional corruptibility-guided sat-based attack on sequential logic encryption. In *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 281–291, 2021.

[36] Yuanqi Shen and Hai Zhou. Double dip: re-evaluating security of logic encryption algorithms. In *Great Lakes Symposium on VLSI (GLSVLSI)*, pages 179–184, 2017.

[37] Kaveh Shamsi, Meng Li, Travis Meade, Zheng Zhao, David Z. Pan, and Yier Jin. App-sat: approximately deobfuscating integrated circuits. In *International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 95–100, 2017.

[38] Nimisha Limaye, Satwik Patnaik, and Ozgur Sinanoglu. Fa-sat: Fault-aided sat-based attack on compound logic locking techniques. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1166–1171, 2021.

[39] Jordan Maynard and Amin Rezaei. Dk lock: Dual key logic locking against oracle-guided attacks. In *2023 24th International Symposium on Quality Electronic Design (ISQED)*, pages 1–7, 2023.

[40] Yasaswy Kasarabada, Vaishali Muralidharan, and Ranga Vemuri. Sled: Sequential logic encryption using dynamic keys. In *2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 844–847, 2020.

[41] Vijaypal Singh Rathor, Munesh Singh, Kshira Sagar Sahoo, and Saraju P. Mohanty. Gatelock: Input-dependent key-based locked gates for sat resistant logic locking. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 32(2):361–372, 2024.

[42] Mark C. Hansen, Hakan Yalcin, and John P. Hayes. Unveiling the iscas-85 benchmarks: a case study in reverse engineering. *IEEE Design & Test of Computers*, 16(3):72–80, 1999.

[43] Chris Drake. PyEDA: Python electronic design automation. https://github.com/cjdrake/pyeda, 2023.

[44] Kaveh Shamsi. Attack tool and benchmarks. https://bitbucket.org/kavehshm/neos, 2019.

[45] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. The epfl combinational benchmark suite. *Integrated Systems Laboratory (LSI), EPFL, Switzerland*, 2015.

[46] Franc Brglez, David Bryan, and Krzysztof Kozminski. Combinational profiles of sequential benchmark circuits. In *1989 IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 3, pages 1929–1934, 1989.

[47] Shervin Roshanisefat, Hadi Mardani Kamali, Houman Homayoun, and Avesta Sasan. Rane: An open-source formal de-obfuscation attack for reverse engineering of logic encrypted circuits. *Great Lakes Symposium on VLSI*, 2021.

[48] Berkeley Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification. http://www.eecs.berkeley.edu/~alanmi/abc.