2023 Design, Automation & Test in Europe Conference (DATE 2023)

ObfusLock: An Efficient Obfuscated Locking Framework for Circuit IP Protection[†]

You Li[*], Guannan Zhao[*], Yunqi He, and Hai Zhou
Northwestern University, Evanston, USA
{you.li, gnzhao, yunqi.he}@u.northwestern.edu, haizhou@northwestern.edu

***Abstract*—With the rapid evolution of the IC supply chain, circuit IP protection has become a critical realistic issue for the semiconductor industry. One promising technique to resolve the issue is logic locking. It adds key inputs to the original circuit such that only authorized users can get the correct function, and it modifies the circuit to obfuscate it against structural analysis. However, there is a trilemma among locking, obfuscation, and efficiency within all existing logic locking methods that at most two of the objectives can be achieved. In this work, we propose ObfusLock, the first logic locking method that simultaneously achieves all three objectives: locking security, obfuscation safety, and locking efficiency. ObfusLock is based on solid mathematical proofs, incurs small overheads (<5% on average), and has passed experimental tests of various existing attacks.**

***Index Terms*—IP piracy, logic locking, SAT attack, hardware obfuscation, logic synthesis**

I. Introduction

The IC supply chain is susceptible to security and privacy challenges, including IP piracy, counterfeiting, reverse engineering, and the insertion of hardware Trojans. Various countermeasures such as hardware metering [1], split manufacturing [2], IC camouflaging [3], watermarking [4], and *logic locking* are developed to tackle these challenges. Among those, logic locking can defend from both reverse engineering and unauthorized activities within the supply chain. Logic locking techniques embed protection logic controlled by key bits to the gate-level netlist of the original circuit. The resulting circuit can operate adequately only in the presence of a correct key.

Early logic locking techniques [5], [6] insert additional key-controlled gates to the original circuit. They are susceptible to I/O attacks, such as the sensitization attack [7] and the SAT attack [8], [9]. In specific, the SAT attack incrementally prunes out incorrect keys and automatically solve for a correct key. In each iteration, it calls a SAT solver to find a distinguishing input pattern (DIP). The algorithm then queries an oracle circuit for the correct output pattern and adds the correct input/output pair as a constraint. It terminates when no more DIPs can be found.

To thwart I/O attacks, various single-flip defences [10], [11] were proposed. They add a dedicated logic to flip the primary outputs when an incorrect key is inserted. Single-flip defences defeat SAT attacks by ensuring that every DIP can prune out only a small set of incorrect keys, thus requiring an exponential number of SAT calls to find a correct key. Nevertheless, the existence of a unique flip node makes single-flip defences vulnerable to structural analysis on the encrypted netlist. Among others, the signal probability skewness (SPS) attack [12] locates the flip node by computing the skewness values for all nodes. Once identified, the flip node can be removed [13] or bypassed [14].

Double-flip defences are proposed subsequently. They aim at resisting both the SAT attacks and structural attacks. The original circuit is first flipped by the *corrupt unit*. It is then flipped by the key-protected *restoring unit*, whose functionality is identical to the corrupt unit when a correct key is applied. The original circuit is obfuscated with the corrupt unit, and the resultant circuit is referred to as the functionality stripped circuit. Launching a structural attack on the restoring unit alone does not fully recover the functionality of the original circuit. Due to obfuscation, identifying the critical node driven by the corrupt unit is non-trivial. Double-flip defences achieve resilience to SAT attacks in a similar way as single-flip defences.

Early double-flip defences corrupt the original circuit by inserting hard-coded structures [15], [16]. $\mathrm{SFLL_{hd}}$ [14] flips the output for any input patterns with a certain Hamming Distance from the key pattern. $\mathrm{SFLL_{flex}}$ [14] corrupts a set of user-specified input patterns, and the key bits consist of a lookup table that restores the circuit's functionality. $\mathrm{SFLL_{fault}}$ [17] and $\mathrm{SFLL_{rem}}$ [18] construct the functionality stripped circuit by injecting faults or removing nodes from the original circuit. Afterward, they utilize ATPG tools and equivalence checking tools to find all input cubes whose corresponding outputs deviate from the correct ones. These input cubes are referred to as protected input cubes. This way, they can build a lookup table to restore outputs for the protected input cubes.

A. Common Limitations in Existing Double-flip defences

- Several defences, including TTLock [15], $\mathrm{SFLL_{hd}}$ [14] and MCAS [19], have very specific designs. Knowing the details of the defences, attackers can devise dedicated algorithms to defeat them [9], [20]–[22]. Other defences are still vulnerable to machine learning attacks. GNNUnlock [23], OMLA [24] and SAIL [25] extract local structural characteristics of a node as learning features. They can discover critical nodes if the defence method exhibits deterministic structural patterns statistically.
- Attackers can still exploit structural vulnerabilities to defeat those defences that add additional logic as the corrupt unit [20], [26]. Re-synthesis is not effective on those defences even after correlations are added in between the corrupt unit and the original circuit. It is due to the fact that the behaviours of the two parts are fundamentally distinct. Furthermore, although re-synthesis techniques can usually mix up the flip nodes with the original circuit, equivalent nodes usually still exist. As long as the critical nodes exist, attackers can figure out a way to discover them. For instance, Valkyrie [27] disables two nodes simultaneously and checks if the resulting circuit has an equivalent behaviour to the oracle.
- The authors of [28] pointed out that almost all structural stripping techniques proposed so far can be easily discovered and recovered. Because a synthesized netlist is highly optimized and free of redundancy, injecting constant-0 or constant-1 faults to a node in the netlist or modifying a few cubes with a point function can always be observed with EDA tools.
- Fault-based defences, including $\mathrm{SFLL_{fault}}$ and $\mathrm{SFLL_{rem}}$, remove logic instead. While they are more robust to structural attacks, they have to find all affected input patterns of the injected faults to achieve a full restoration. The restoration process relies on testing and formal equivalence checking, and that takes several hours for circuit

[*] Equal contribution.

[†] This work is partially supported by the National Science Foundation under grants 2113704 and 2148177.

978-3-9819263-7-8/DATE23/© 2023 EDAA

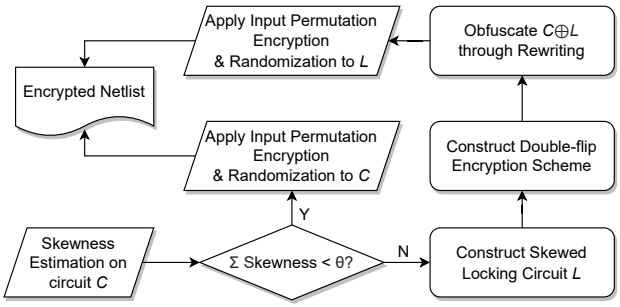


Fig. 1: An overview of ObfusLock.

benchmarks with a few thousand logic gates [17]. Moreover, these defences lack key efficiency: the length of the key is proportional to the number of protected input cubes. As a result, they can protect only one or a limited number of input cubes due to the cost of the tamper-proof memory.

B. Novelties and Contributions

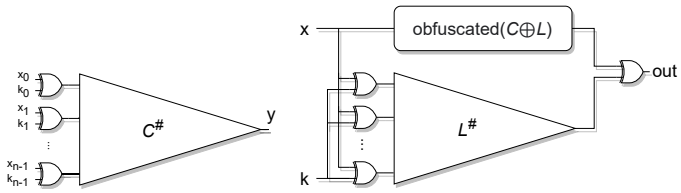
All the discussions above suggest the necessity of a new logic encryption scheme satisfying these requirements: *i)* has an exponential complexity against I/O attacks in key size; *ii)* produces fully randomized locking patterns; *iii)* eliminates critical nodes; *iv)* allows high flexibility in constructing the locking circuit to counter synthesis-based attacks; *v)* incurs minimum power, performance and area (PPA) overheads; and *vi)* is efficient in key size and runtime. As far as we are aware, none of the existing works can fulfill all these requirements.

ObfusLock addresses all the challenges in a comprehensive framework. It adopts input permutation encryption on skewed locking circuits to guarantee robustness against I/O attacks. The process of constructing a locking circuit is inherently randomized. It is also highly flexible so that a defender can first choose behaviour for the locking circuit and then implement that behaviour within the framework. ObfusLock utilizes structural rewriting to split and eliminate critical nodes. Because of logic sharing, PPA overheads incurred by logic locking are minimized. The overall workflow is systematic and it is efficient in execution time.

The workflow of ObfusLock is illustrated in Fig. 1. It starts by assessing the skewness of the original circuit C. If the skewness is within the desired range, ObfusLock applies input permutation encryption directly. Otherwise, ObfusLock incrementally constructs a highly skewed locking circuit L using the nodes in the original circuit as building components. The same locking circuit constitutes both the corrupt unit and the restoring unit in a double-flip scheme. Particularly, L is obfuscated and merged with C through a sequence of automatic rewriting processes, while C as the restoring unit is encrypted by key-controlled input permutation and randomization. ObfusLock has strong security guarantees against any known attacks on logic locking.

The contributions of this work are summarized as follows:

- We formally establish the security of input permutation encryption on skewed circuits.
- We develop a general scheme, ObfusLock, to lock any given netlist. In specific, we propose a method to construct a highly skewed locking circuit from the nodes in the original circuit. Such a locking circuit can be adapted into the double-flip locking framework.
- We present the rewriting rules and procedures to obfuscate the locking circuit with the original circuit with assurance.
- We conduct detailed analyses for ObfusLock to evaluate its security and overhead.



(a) (b)

Fig. 2: (a) the input permutation encryption; (b) the architecture of double-flip ObfusLock.

II. Threat Model

Our threat model is consistent with the previous works in logic encryption. The attacker is either an untrustworthy foundry or a reverse engineering house and thus has access to the encrypted netlist C_{enc}. Apart from that, the attacker can purchase a working chip from the open market as the oracle C_o. It can query the oracle by applying a specific input pattern and observing the corresponding output pattern. Nevertheless, the attacker cannot read the secret key as it is stored in a tamper-proof memory [29]. The objective of the attacker is to determine the secret key and thereby defeat logic locking.

III. Ideal Logic Locking for Skewed Circuits

A. Preliminaries

We denote as C the netlist of a combinational circuit or the combinational part of a sequential circuit. C implements a Boolean function $f(x) : \{0,1\}^m \rightarrow \{0,1\}^n$, where m, n are the lengths of the inputs and the outputs, respectively. The logic encrypted netlist $C_{enc}(x,k),\ k \in \{0,1\}^l$ is the encrypted version of C. A key k^* is correct if $\forall x \in \{0,1\}^m : C_{enc}(x,k^*) = C(x)$, and it is incorrect otherwise. The attacker's objective is to find a correct key and thus recover the functionality of the encrypted circuit.

The *on-set* of node p, denoted as f_p^1, is the set of all input patterns for which p evaluates to 1. Let h_p denote the smaller one between f_p^1 and its complement. The *skewness* of node p is defined as $h_p/2^m$, and p is said to be *highly skewed* if $h_p \ll 2^m$.

The *error matrix* of an encrypted circuit is given by

$$E(x_i,k_j) \triangleq C_{enc}(x_i,k_j) \neq C(x_i). \tag{1}$$

Each row in the matrix represents an input pattern from 0 to 2^m-1 and each column represents a key pattern from 0 to 2^l-1. The *error number* of x_i is defined as the number of 1s on the ith row. A key k_j is correct if and only if all elements are 0 on the jth column.

An input cube is either a full or a partial assignment to the input variables. It contains all input patterns satisfying the assignment.

B. Input Permutation Encryption for Skewed Circuits

Ideal logic locking can be achieved by placing key-controlled *XOR* gates to all primary inputs if the circuit has only one output bit and is highly skewed. Bubbles (inverters) are added at random to the primary inputs of C to randomize the key polarities. Logic simplifications are then performed to hide those bubbles and yield $C^\#$ [30]. The encrypted circuit is thus $C_{enc}(x,k) = C^\#(x \oplus k)$, as shown in Fig. 2(a). Fundamentally, a key represents a permutation in the input space, and a correct key restores the effects of all bubbles. The idea of adding *XOR* gates to the primary inputs has been mentioned in [31] but without any in-depth analyses or proofs. Besides, that work neither discusses how to ensure security when the original circuit does not satisfy the required conditions, nor does it provide a solution to control the overhead on industrial benchmarks. In this section, we formally establish the security of such a scheme. Section IV describes how this can facilitate the construction of a provably secure logic locking framework for general circuits.

1) SAT Attack Resilience: Launching an SAT attack on an encrypted circuit can be viewed as solving a covering problem on the error matrix. The objective of the attacker is to find a column with no errors. For this purpose, an exact attacking algorithm has to select a subset of rows, covering all columns with at least one error. An approximate attack [32] may terminate before all columns with an error have been covered and could return any keys whose corresponding columns have not yet been covered.

The error matrix of an input permutation encrypted circuit is given by the following lemma. We omit the proof due to space limit.

Lemma 1. *Let $f(x) : \{0,1\}^m \rightarrow \{0,1\}$ be a Boolean function, $|f^1| = M$. The error matrix of the encrypted function $f_{enc}(x,k) = f^{\#}(x \oplus k)$ has M rows each containing exactly $2^n - M$ errors, and $2^n - M$ rows each containing exactly M errors.*

A row is *zero-dominant* if most of its elements are 0s or *one-dominant* if most of them are 1s. Lemma 1 implies that at most $h = \min(M, 2^m - M)$ key patterns could be correct, as stated in the following lemma.

Lemma 2. *The number of correct keys is always equal to or less than h for the encrypted function $f_{enc}(x,k) = f^{\#}(x \oplus k)$.*

The following analysis assumes the underlying SAT solver has an equal chance to return any satisfiable assignments. A row could be selected if the x_i it corresponds to is distinguishing, *i.e.,* there exists both a 0 element and a 1 element on that row, and any previously selected rows do not cover the 1 element. The chance for a row to be selected is proportional to the number of remaining $(0,1)$ pairs on that row. Note that an attacker cannot determine if a row is one-dominant unless it knows $f(x_i)$ in advance.

Theorem 3. *If $|f^1| = M$ and $h = \min(M, 2^m - M) \ll 2^m$, the expected number of I/O queries for a SAT-based attacking algorithm to decrypt $f_{enc}(x,k) = f^{\#}(x \oplus k)$ is at least $(1/c) \cdot (2^m/h)$, where c is a small constant.*

Proof. One of the two conditions must be fulfilled before the attacker succeeds: *a)* select a one-dominant row; or *b)* select $2^m/h - 1$ zero-dominant rows. From Lemma 1, the probability for the attacker to select a one-dominant row is always less than $2h/2^m$ before the number of I/O queries reaches $2^m/2h$. Using $(1 - 2h/2^m)^{2^m/2h} \leq 1/e$, we derive $(1/2) \cdot (1 - 1/e) \cdot (2^m/h)$ as a lower bound of the expected I/O queries required to fulfill condition *a)*. □

2) Sensitization Attack Resilience: The sensitization attack [7] is another I/O attack that aims to decrease the number of effective key bits. It attempts to find input patterns that sensitize the targeted key bits to the output while muting the remaining key bits. Input permutation encryption resists sensitization attacks since all key bits are sensitized to the output regardless of the input pattern.

3) Approximate Attack Resilience: Several attacking algorithms [14], [32] query the oracle a finite number of times. Then they choose a key at random from those not yet proved to be incorrect. The following result can be proved similarly as Theorem 3.

Theorem 4. *Let $|f^1| = M$ and $h = \min(M, 2^m - M) \ll 2^m$. The probability that a randomly selected key decrypts $f_{enc}(x,k) = f^{\#}(x \oplus k)$ is less than $c \cdot r \cdot h/2^m$, where r is the number of queries to the oracle by an I/O attacking algorithm and c is a small constant.*

4) Bypass Attack Resilience: The bypass attack [14] adds an additional bypass unit to circumvent the protected input cubes that lead to errors. In this way, the circuit can still work properly when an incorrect key is applied. ObfusLock thwarts the bypass attack since *all* input patterns are protected by permutation.

IV. ObfusLock : A Universal Framework for Logic Encryption

A. Integrating Input Permutation to Logic Locking

As discussed in Section III, input permutation encryption is provably secure if the original circuit has a single primary output bit which is highly skewed. For a circuit that has multiple primary outputs, the sum of h for all outputs should not exceed a secure threshold.

What if the skewness level does not meet the threshold? In this case, ObfusLock constructs an additional locking circuit L that is highly skewed and has a single output. The high-level architecture of ObfusLock is shown in Fig. 2(b). It consists of the *obfuscated unit* $C \oplus L$ and the input permutation encrypted *restoring unit* $L^{\#}(x \oplus k)$. Similar to *SFLL* [33], this architecture resists structural attacks on the restoring unit because removing that unit cannot recover the functionality of the original circuit.

Consider the encrypted circuit $C \oplus L \oplus L^{\#}(x \oplus k)$ and the oracle circuit $C \oplus L \oplus L^{\#}(x \oplus k^*)$. Because the obfuscated unit parts in both circuits are always identical regardless of the keys, only the restoring unit part needs to be considered in analyzing any oracle-guided I/O attacks. Therefore, all security guarantees presented in Section III still hold for double-flip ObfusLock.

In the remainder of this section, we elaborate ObfusLock encryption procedures. We first discuss methods to estimate skewness values when they are exponentially small. We then propose a practical way to construct L incrementally. Finally, we present concrete resynthesis techniques that make ObfusLock structurally robust.

B. Estimating Skewness Values

How do we estimate the probability a node evaluates to 1, assuming every primary input has the same opportunity to be 0 or 1? One naive approach is to transform the netlist to an And-Inverter Graph, sorting all nodes into a topological order, and then compute skewness gate by gate. The skewness is the complement of its input for an inverter and the product of its inputs for an *AND* gate [12]. This *algebraic computation* method can lead to significant error, especially when the transitive fan-in cones of a gate's inputs overlap. Another naive way is to draw random input patterns and simulate the netlist. Because we desire at least $\epsilon < h/2^m$, the required sample size, $\mathcal{O}(1/\epsilon^2)$, is prohibitively large to achieve a reasonable confidence level.

To overcome the sample size problem, we propose Boolean multi-level splitting. The key insight is that a rare event could be divided into a sequence of consecutive common events, and applying random simulation on those common events separately requires only a moderate number of samples. Our method starts by reshaping the Boolean network to *maximize* its logic height. This can be achieved by reversely applying depth-oriented Boolean optimizations [34]. As a result, a long critical path B should exist for every primary output. We distinguish a set of nodes $p_1, \cdots, p_n$ along B, so that the skewness values of every pair of consecutive p nodes are close. The skewness value sk_i of node p_i can thus be computed recursively: $sk_i = Pr(p_i = 1 | p_{i-1} = 1) \cdot sk_{i-1} + Pr(p_i = 1 | p_{i-1} = 0) \cdot (1 - sk_{i-1})$. Then we use a SAT witness sampler [35] to accurately estimate the conditional probabilities in the above equation.

C. Constructing the Locking Circuit

ObfusLock builds a highly skewed locking circuit L with nodes in the original circuit. Using the same nodes can reduce area and power overheads and facilitate structural rewriting. It first executes algebraic skewness computation to find candidate nodes with high skewness values in the original circuit. Although inaccurate, algebraic computation is efficient in searching for a large number of nodes. It

then builds L incrementally. In each iteration, a pre-selected *operator* with a few nodes drawn from the candidates is tentatively attached to the front of the current critical path. Boolean multi-level splitting is initiated to estimate the gain in skewness value. If the gain is higher than the required level, the attachment is confirmed, and Boolean multi-level splitting is initiated again with a larger time budget to compute the new skewness value accurately. Otherwise, the process starts over with a decayed gain level. The construction is finished when the skewness value of the current L is below the secure threshold.

5) Synthesis-based Attack Resilience: Most logic locking techniques alter fixed patterns within the logic representations of the original circuits. The SPI attack [28] utilizes this fact to launch a structural attack on the critical node. After logic synthesis and optimization, the logic terms introduced by logic locking and those of the original circuit can be separated by a set of rules. The flexibility of ObfusLock enables the defender to first choose the logic terms that cannot be detected by the rules, and then construct a skewed locking circuit implementing those terms. An illustrative example is shown in Figure 3.

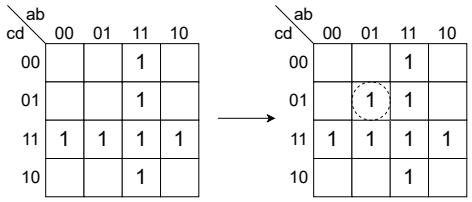


Fig. 3: A demonstration of how ObfusLock thwarts synthesis-based attacks. (a) The logic representation of the original circuit after logic optimization. $abcd$ are internal nodes on a cut within the transitive fan-in cone. (b) The circled element is changed by the defender, after which none of the original prime implicants are split from either 0 or 1 polarity. A skewed locking circuit L is then constructed to implement this change.

D. Logic Obfuscation through Structural Transformation

Structural attacks to logic locking aim to identify the critical nodes in the netlist. Attackers may launch static analysis (*e.g.,* gate-level analysis) or a combination of static and dynamic analysis (*e.g.,* signal activity analysis) to find the critical nodes. To protect sensitive information, logic obfuscation alters the netlist without changing its behaviour. It seems impossible to exhaust all possible structural analyses and prove an obfuscation method is definitely secure. However, an obfuscated netlist is structurally robust if it has a prohibitively large number of possible selections of critical nodes, and each selection needs to be validated dynamically. It is even more robust if no valid selections exist on the encrypted netlist.

For the obfuscated unit $C \oplus L$, the critical node is the root node of C or L. To launch a removal attack or a bypass attack, either the critical node itself or all nodes on a *cut* in the transitive fan-in cone need to be identified by the attacker. ObfusLock conducts a combination of structural and functional rewriting steps to obtain structural robustness. The advantages are threefold: *a) Decomposition:* the critical node and its activity is decomposed into multiple nodes; *b) Propagation:* the decomposed nodes are propagated and spread in the netlist; *c) Elimination:* the decomposed nodes are merged with or substituted by other nodes in the netlist.

The rewriting steps are summarized as follows. *i) Extended AIG Transformation:* the original circuit C is transformed to an And-Inverter Graph (AIG). Similarly, the locking circuit L is transformed to an extended AIG that contains Majority-of-three (*MAJ*) and *XOR* gates besides *AND* gates and inverters. *ii) Structural Reshaping:* Equations (2) - (4) illustrate the structural rewriting rules exploited by ObfusLock to decompose and propagate L. In these equations, f denotes the root node of C. f is XORed with the currently processed node in L, while a, b and c are the immediate inputs of the current node. Structural reshaping proceeds iteratively from the root node of L. Depending on the type of the current node, one equation among (2) - (4) is selected, and the LHS is replaced by the RHS. Such an iteration continues on the first term in the RHS. *iii) Structural Elimination:* the locking circuit L, as well as its activity, is distributed to the whole network after structural reshaping. However, there is a pitfall: node f is unchanged and can possibly be discovered by an attacker. Thus, we devise rewriting rules to further eliminate f. One example rule, given as equation (5), is applicable whenever the transitive fan-in cone of f contains the internal node $a\neg b$ where both a and b are certain nodes in C. We apply (5a)-(5b) repeatedly to propagate f down the AIG, and (5c) to get it finally eliminated. Elimination rules as such are made possible because L consists of nodes in C. *iv) Functional Rewriting:* it replaces local structures with equivalent precomputed structures. We use standard AIG functional rewriting [36] to remove traces produced in rewriting.

$$f \oplus ab = (f \oplus a) \oplus a\neg b, \tag{2}$$

$$f \oplus (a \oplus b) = (f \oplus a) \oplus b, \tag{3}$$

$$f \oplus \langle abc \rangle = \langle (f \oplus a)(f \oplus b)(f \oplus c) \rangle, \tag{4}$$

$$f \oplus ab = \begin{cases} \neg(f_0 \oplus ab), & (f = \neg f_0) \text{ (5a)} \\ (f_0 \oplus ab) f_1 \vee ab\neg f_1, & (f = f_0 f_1) \text{ (5b)} \\ a. & (f = a\neg b) \text{ (5c)} \end{cases}$$

The above equations are listed just for illustration purposes. Defenders can always add customized reshaping and elimination rules for further diversification. The whole rewriting procedure terminates when the number of applications of reshaping and elimination rules both reach user-specified thresholds or the whole L netlist has been traversed. Because of logic sharing, each application of a rewriting rule introduces at most a constant number of extra gates.

On large benchmarks, applying double-flip ObfusLock from primary inputs to primary outputs (as shown in Fig. 2(b)) can incur significant overhead. In this situation, we apply it only to a sub-circuit to mitigate this issue. To find a suitable sub-circuit, ObfusLock traverses backwards from a small set of primary outputs that have important functionalities. It aims to find a *cut*, such that *i)* the cut size is sufficiently large, and *ii)* the number of reachable patterns on the cut is exponential to the cut size. The second condition is crucial because an attacker may attempt to infer input patterns corresponding to the reachable patterns on the cut, thereby launching an SAT attack [37]. We use an approximate model counter [38] to track the number of reachable patterns on the current cut. Once the cut is settled down, the transitive fan-out cone of the cut is extracted as the sub-circuit.

Notice that the attacker can only obtain an oracle for the whole circuit. Even if the attacker can discover the cut of the sub-circuit, it needs to infer the corresponding input pattern from a pattern on the cut in every I/O attack iteration. This process will always introduce an extra cost. A defender can even encrypt the remaining part of the circuit to thwart such attempts.

6) Machine Learning Attack Resilience: These attacks [23], [25] extract local structural information of nodes as learning features. They discover critical nodes by revealing deterministic patterns caused by logic locking. However, ObfusLock applies structural transformations to the obfuscated unit globally. Moreover, it breaks any deterministic patterns or statistical relations by introducing randomness from four aspects: *i)* it enables high flexibility to choose internal nodes and operators in constructing L; *ii)* its bubble insertion and pushing

Bench. (#Nodes)	Skew.	#Keys (bits)	Run. (s)	SAT Atk. (s) sub.	SAT Atk. (s) whole	AppSAT Atk. (s) sub.	AppSAT Atk. (s) whole
s9234 (3677)	-11.4	14	1.23	162.9	TO	0.1	4.3
	-21.8	27	1.69	TO	TO	wrong	wrong
	-30.6	51	18.09	TO	TO	wrong	wrong
c7552 (4003)	-12.5	23	0.63	TO	TO	238.5	wrong
	-20.4	35	4.35	TO	TO	wrong	wrong
	-33.2	55	17.78	TO	TO	wrong	wrong
c6288 (4660)	-10.6	19	1.77	231.7	406.2	13.8	wrong
	-22.5	31	5.30	TO	TO	wrong	wrong
	-32.8	47	9.89	TO	TO	wrong	wrong
max (5907)	-10.4	16	0.63	TO	TO	172.1	wrong
	-20.6	35	3.90	TO	TO	wrong	wrong
	-30.0	54	7.19	TO	TO	wrong	wrong
s15850 (6820)	-12.0	19	0.42	TO	TO	wrong	wrong
	-21.6	33	4.45	TO	TO	wrong	wrong
	-32.5	51	2.63	TO	TO	wrong	wrong
b14 (10635)	-12.0	15	0.38	51.4	TO	245.3	wrong
	-24.0	33	2.16	TO	TO	wrong	wrong
	-30.0	38	1.65	TO	TO	wrong	wrong
	-54.0	125	33.56	TO	TO	wrong	wrong
s38417 (18781)	-10.0	16	0.82	0.9	TO	0.1	97.0
	-22.3	58	2.44	TO	TO	wrong	wrong
	-34.2	64	10.63	TO	TO	wrong	wrong
	-52.4	100	48.16	TO	TO	wrong	wrong
b20 (24292)	-12.0	20	0.43	0.2	103.8	10.9	175.4
	-21.0	21	2.61	TO	TO	wrong	wrong
	-30.0	31	7.94	TO	TO	wrong	wrong
	-50.0	99	35.30	TO	TO	wrong	wrong
s38584 (24296)	-10.0	15	1.37	4.1	76.0	12.8	95.0
	-22.9	35	1.51	TO	TO	wrong	wrong
	-32.0	51	17.52	TO	TO	wrong	wrong
	-57.4	126	32.70	TO	TO	wrong	wrong
square (39248)	-10.9	26	1.33	50.4	TO	55.5	wrong
	-20.2	37	7.06	TO	TO	wrong	wrong
	-31.5	63	22.98	TO	TO	wrong	wrong
	-50.2	148	60.01	TO	TO	wrong	wrong

TABLE I: Evaluation results (key efficiency, runtime, and resilience to I/O attacks) for ObfusLock at different skewness levels.

process is fully randomized; *iii)* it allows the defender to add structural reshaping and elimination rules at will; *iv)* it enables high flexibility to choose the sub-circuit to be encrypted.

V. Experimental Results

In this section, we present experimental results to demonstrate the effectiveness and the robustness of ObfusLock. All experiments are conducted on a Linux machine with a 3.2GHz CPU and 16GB of RAM. We evaluated ObfusLock on larger benchmark circuits from ISCAS'89 [39], ITC'99 [40] and EPFL [41]. Like most studies on logic locking [42], we assume attackers have scan chain access.

We notice that ObfusLock cannot be effectively applied to those circuits whose number of inputs is less than the desired skewness level. For instance, ObfusLock fails to find a locking circuit for $b09$ (serial to serial converter) and $b10$ (voting system) in ITC'99, both of which are deep sequential data pipelines with only a limited number of inputs. ObfusLock may be effectively applied to such circuits if unrolling is allowed.

In order to thoroughly evaluate ObfusLock, we extract the transitive fan-in cones for all outputs of all candidate benchmarks. We filter out a benchmark if less than 20% of its outputs have at least 30 inputs ($<$10,000 nodes) or 50 inputs ($\geq$10,000 nodes) within their transitive fan-in cones. However, as we will demonstrate later, this requirement is not needed in practice. Among the remaining, we select 10 benchmark circuits whose numbers of nodes are across the range of 3,000 to 40,000 to represent circuits of different sizes.

Encryption Cost: Table I shows the experiment results of ObfusLock encryption cost. Because different benchmark suites are in different formats (gate-level netlists and LUTs), we map all benchmark circuits to AIG and count the number of nodes. For every benchmark circuit, we create 3 locking circuits L with at least 10, 20 and 30 bits of skewness, respectively (-10.0 bits of skewness means the skewness value of L is 2^{-10}). For those benchmark circuits which have over 10,000 nodes, we also create a locking circuit with at least 50 bits of skewness. The execution time of ObfusLock is in the order of tens of seconds. The majority of the time is spent on estimating the gain in skewness for each additional operator and corresponding nodes. Therefore, to construct an L with a lower skewness value demands more execution time.

Security Analysis (I/O Attacks): We use SAT attack and AppSAT attack implementations in NEOS [32] to attack ObfusLock encrypted netlists. We choose NEOS because it integrates BDD sweeping to compress I/O constraints and is almost always faster than the original SAT attack implementation [8]. We set the timeout (TO) limit to be 3 hours for all attacks. Besides, we set the upper limit for AppSAT to be 2,048 iterations, after which it must return a key that is not proved to be incorrect. This limit is higher than what is used in the AppSAT paper [32].

We consider two attacking strategies. For the first strategy, an attacker targets the **whole** encrypted circuit as normal. For the second strategy, an attacker distinguishes and only targets the **sub**-circuit to reduce the burden on the SAT solver. In our experiments, we assume an attacker who adopts the second strategy *i)* always finds reachable patterns on the cut when launching an I/O attack, and *ii)* can deduce the pattern on the inputs from the pattern on the cut in no time. Because both of the assumptions are not realistically achievable for an attacker, the reported data in the **sub.** columns are the *lower* bounds of the real-world values.

As shown in Table I, for all the benchmark circuits, encryptions with 20 bits of skewness cannot be decrypted in the timeframe regardless of the sizes of the circuits. 30 bits of skewness or more is recommended to resist attackers who have greater capabilities.

Security Analysis (Structural Attacks): Structural attacks utilize internal structural and functional information of the encrypted circuit to recover the functionality of the original circuit. Most structural attacks aim to distinguish the critical nodes within the encrypted netlist and then isolate them or inject faults to these locations [12], [21], [26], [27], [37]. Naively, an attacker can search through the netlist to identify the critical nodes. Hence we start by running a combinational equivalence checker to validate whether the critical nodes, namely the root nodes of C and L, still exist after obfuscation. Our result verifies that all critical nodes are successfully eliminated in every encrypted circuit.

We further evaluate the structural robustness of ObfusLock against the state-of-the-art structural attacking tools, Valkyrie [27] and SPI attack [28]. When attacking double-flip defences, Valkyrie aims to find a pair of flipping node in the encrypted circuit, namely *perturb* and *restore*, such that when both nodes are replaced by constant faults, the encrypted circuit has the same behaviour as the original circuit. In our cases, perturb and restore are the root nodes of L and $L^{\#}$ respectively. We observe that while Valkyrie always finds the restore node quickly, which we leave as-is, it either exits without finding the obfuscated perturb node or exceeds the time limit.

We subsequently launch the SPI attack, which generates prime implicant tables (PITs) for a set of selected internal nodes given an encrypted circuit and infers the key therewith. The SPI attack always returns an incorrect key.

If an attacker cannot distinguish the critical nodes, it may attempt to distinguish a set of fan-in nodes of the obfuscated C, and then reconstruct C with these nodes. Common techniques to distinguish candidate nodes include skewness analysis [12], [37] and sensitization analysis [21], [26], [27], [37]. Fig. 4 demonstrates how ObfusLock

protects structural and functional traces against these two attacks on *s9234*. Fig. 4 (a) and Fig. 4 (b) show the statistics of skewness and the number of keys within its transitive fan-in cone for all nodes. It can be seen that the critical node $C \oplus L$ (marked in red) is an outlier. After structural transformation, this critical node no longer exists. We suppose that the attacker attempts to recover the functionality of C with nodes in the transitive fan-in cone of $C \oplus L$. Fig. 4 (c) and Fig. 4 (d) show the statistics after structural transformation. The attacker minimizes the complexity to recover the functionality of C, if it finds a cut between the inputs and the outputs, such that *i)* it consists of nodes that have equivalent counterparts in the original circuit C, and *ii)* it is the closest cut to the protected outputs. All nodes along the cut are marked in yellow. Because of structural transformation, it is almost unlikely to differentiate these nodes from others.

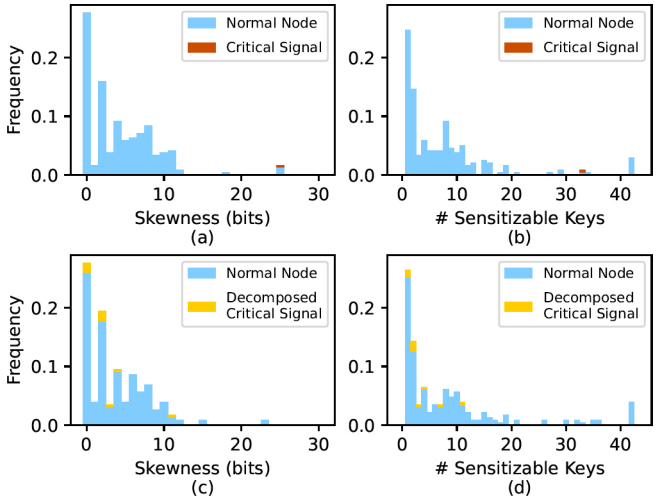


Fig. 4: Distributions of critical nodes before and after structural transformation.

Overhead Analysis: We use Cadence Genus and Innovus along with the NanGate 45nm Open Cell Library to measure the area and power overheads. We choose the typical RC corner and set the target clock period as $1ns$ for our analysis. As shown in Fig. 5, ObfusLock incurs average area overheads of 2.1%, 4.0%, 5.1% and 4.8% for netlists with 10, 20, 30 and 50 bits of skewness, respectively. The majority of the area overhead is due to the restoring unit. Facilitated with the sub-circuit approach, the area overhead is mostly correlated with the skewness level but almost unrelated to the size of the original circuit. It means that for a certain security level, the relative overhead of ObfusLock decreases as the size of the original circuit grows. ObfusLock records a 4.9% average power overhead for all the benchmarks. Delay overhead is almost negligible (0.07% on average).

VI. Conclusion

In this paper, we propose ObfusLock, a novel logic locking scheme that simultaneously achieves I/O attack resilience, structural attack resilience, locking efficiency and protection diversity. ObfusLock leverages skewness of nodes to construct a locking circuit and uses a set of rewriting rules to obfuscate it with the original circuit. Experimental results have confirmed the strong security guarantees provided by ObfusLock.

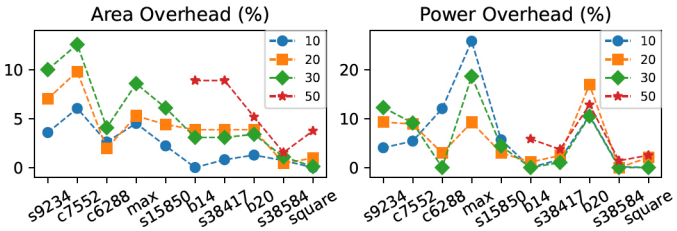


Fig. 5: Area and power overheads at different skewness levels (bits).

References

[1] Y. Alkabani and F. Koushanfar, "Active hardware metering for intellectual property protection and security." in *USENIX security symposium*, 2007, pp. 291–306.

[2] F. Imeson *et al.*, "Securing computer hardware using 3d integrated circuit ({IC}) technology and split manufacturing for obfuscation," in *USENIX Security 13*, 2013, pp. 495–510.

[3] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri, "Security analysis of integrated circuit camouflaging," in *CCS*, 2013, pp. 709–720.

[4] A. B. Kahng *et al.*, "Watermarking techniques for intellectual property protection," in *Proceedings of the 35th annual Design Automation Conference*, 1998, pp. 776–781.

[5] J. A. Roy, F. Koushanfar, and I. L. Markov, "Ending piracy of integrated circuits," *Computer*, vol. 43, no. 10, pp. 30–38, 2010.

[6] M. Yasin, J. J. Rajendran, O. Sinanoglu, and R. Karri, "On improving the security of logic locking," *TCAD*, vol. 35, no. 9, pp. 1411–1424, 2015.

[7] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security analysis of logic obfuscation," in *Proceedings of the 49th Annual Design Automation Conference*, 2012, pp. 83–89.

[8] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *HOST*. IEEE, 2015, pp. 137–143.

[9] Y. Shen and H. Zhou, "Double dip: Re-evaluating security of logic encryption algorithms," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*, 2017, pp. 179–184.

[10] M. Yasin, B. Mazumdar, J. J. Rajendran, and O. Sinanoglu, "Sarlock: Sat attack resistant logic locking," in *HOST*. IEEE, 2016, pp. 236–241.

[11] Y. Xie and A. Srivastava, "Mitigating sat attack on logic locking," in *International conference on cryptographic hardware and embedded systems*. Springer, 2016, pp. 127–146.

[12] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Security analysis of anti-sat," in *ASP-DAC*. IEEE, 2017, pp. 342–347.

[13] M. Li, K. Shamsi, T. Meade, Z. Zhao, B. Yu, Y. Jin, and D. Z. Pan, "Provably secure camouflaging strategy for ic protection," *TCAD*, vol. 38, no. 8, pp. 1399–1412, 2017.

[14] X. Xu *et al.*, "Novel bypass attack and bdd-based tradeoff analysis against all known logic locking attacks," in *CHES*. Springer, 2017, pp. 189–210.

[15] M. Yasin, A. Sengupta, B. C. Schafer, Y. Makris, O. Sinanoglu, and J. Rajendran, "What to lock? functional and parametric locking," in *GLSVLSI*, 2017, pp. 351–356.

[16] K. Shamsi, T. Meade, M. Li, D. Z. Pan, and Y. Jin, "On the approximation resiliency of logic locking and ic camouflaging schemes," *TIFS*, vol. 14, no. 2, pp. 347–359, 2018.

[17] A. Sengupta, M. Nabeel, M. Yasin, and O. Sinanoglu, "Atpg-based cost-effective, secure logic locking," in *2018 IEEE 36th VLSI Test Symposium (VTS)*. IEEE, 2018, pp. 1–6.

[18] A. Sengupta, M. Nabeel, N. Limaye, M. Ashraf, and O. Sinanoglu, "Truly stripping functionality for logic locking: A fault-based perspective," *TCAD*, vol. 39, no. 12, 2020.

[19] B. Shakya *et al.*, "Cas-lock: A security-corruptibility trade-off resilient logic locking scheme," *IACR Trans. on Cryptographic Hardware and Embedded Systems*, pp. 175–202, 2020.

[20] F. Yang, M. Tang, and O. Sinanoglu, "Stripped functionality logic locking with hamming distance-based restore unit (sfll-hd)–unlocked," *TIFS*, vol. 14, no. 10, pp. 2778–2786, 2019.

[21] A. Sengupta, N. Limaye, and O. Sinanoglu, "Breaking cas-lock and its variants by exploiting structural traces," *Cryptology ePrint Archive*, 2021.

[22] H. Zhou, Y. Shen, and A. Rezaei, "Vulnerability and remedy of stripped function logic locking." *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 139, 2019.

[23] L. Alrahis *et al.*, "Gnnunlock: Graph neural networks-based oracle-less unlocking scheme for provably secure logic locking," in *DATE*. IEEE, 2021, pp. 780–785.

[24] L. Alrahis, S. Patnaik, M. Shafique, and O. Sinanoglu, "Omla: An oracle-less machine learning-based attack on logic locking," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 3, pp. 1602–1606, 2021.

[25] P. Chakraborty, J. Cruz, and S. Bhunia, "Sail: Machine learning guided structural analysis attack on hardware obfuscation," in *AsianHOST*. IEEE, 2018, pp. 56–61.

[26] D. Sirone and P. Subramanyan, "Functional analysis attacks on logic locking," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 2514–2527, 2020.

[27] N. Limaye, S. Patnaik, and O. Sinanoglu, "Valkyrie: Vulnerability assessment tool and attack for provably-secure logic locking techniques," *TIFS*, vol. 17, pp. 744–759, 2022.

[28] Z. Han, M. Yasin, and J. J. Rajendran, "Does logic locking work with eda tools?" in *USENIX Security 21)*, 2021.

[29] R. Anderson, *Physical Tamper Resistance*. John Wiley & Sons, 2020.

[30] A. Jain and R. E. Bryant, "Inverter minimization in multi-level logic networks," in *ICCAD*. IEEE, 1993, pp. 462–465.

[31] H. Zhou *et al.*, "Resolving the trilemma in logic encryption," in *ICCAD*. IEEE, 2019.

[32] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "Appsat: Approximately deobfuscating integrated circuits," in *HOST*. IEEE, 2017, pp. 95–100.

[33] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. Rajendran, and O. Sinanoglu, "Provably-secure logic locking: From theory to practice," in *CCS*, 2017, pp. 1601–1618.

[34] J. Cortadella, "Timing-driven logic bi-decomposition," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 6, pp. 675–685, 2003.

[35] S. Chakraborty, D. J. Fremont, K. S. Meel, S. A. Seshia, and M. Y. Vardi, "On parallel scalable uniform sat witness generation," in *TACAS*. Springer, 2015, pp. 304–319.

[36] A. Mishchenko, S. Chatterjee, and R. Brayton, "Dag-aware aig rewriting: A fresh look at combinational logic synthesis," in *DAC*. IEEE, 2006, pp. 532–535.

[37] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Removal attacks on logic locking and camouflaging techniques," *TETC*, vol. 8, no. 2, pp. 517–532, 2017.

[38] M. Soos, S. Gocht, and K. S. Meel, "Tinted, detached, and lazy cnf-xor solving and its applications to counting and sampling," in *CAV*. Springer, 2020, pp. 463–484.

[39] F. Brglez, "A neural netlist of 10 combinational benchmark circuits," *Proc. IEEE ISCAS: Special Session on ATPG and Fault Simulation*, pp. 151–158, 1985.

[40] F. Corno, M. S. Reorda, and G. Squillero, "Rt-level itc'99 benchmarks and first atpg results," *IEEE Design & Test of computers*, vol. 17, no. 3, pp. 44–53, 2000.

[41] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The epfl combinational benchmark suite," in *IWLS*, 2015.

[42] H. M. Kamali, K. Z. Azar, F. Farahmandi, and M. Tehranipoor, "Advances in logic locking: Past, present, and prospects," *Cryptology ePrint Archive*, 2022.