
EPiC Series in Computing
Volume 73, 2020, Pages 483–497
LPAR23. LPAR-23: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning


# Sensitivity Analysis of Locked Circuits

Joseph Sweeney[1], Marijn J. H. Heule[2], and Lawrence Pileggi[1]

[1] Department of Electrical and Computer Engineering
[2] Computer Science Department
Carnegie Mellon University, Pittsburgh PA 15213, USA
{joesweeney,marijn,pileggi}@cmu.edu

**Abstract**

Globalization of integrated circuits manufacturing has led to increased security concerns, notably theft of intellectual property. In response, logic locking techniques have been developed for protecting designs, but many of these techniques have been shown to be vulnerable to SAT-based attacks. In this paper, we explore the use of Boolean sensitivity to analyze these locked circuits. We show that in typical circuits there is an inverse relationship between input width and sensitivity. We then demonstrate the utility of this relationship for deobfuscating circuits locked with a class of "provably secure" logic locking techniques. We conclude with an example of how to resist this attack, although the resistance is shown to be highly circuit dependent.

## 1 Introduction

Due to prohibitively high research and development costs, only a few foundries are manufacturing integrated circuits (ICs) in advanced technology nodes. Consequently, many IC companies tend to operate fabless, relying on untrusted foundries to manufacture their designs. Once a circuit is sent for fabrication, the foundry gains full visibility of the intellectual property (IP) in netlist form with minimal effort, allowing IP theft. This threat undermines the significant cost associated with developing digital circuits and is a growing concern in the IC industry [6, 15, 12].

To combat IP theft, a variety of logic locking techniques have been developed. These techniques add programmable elements to the logic of an IC. When programmed incorrectly, the elements disrupt the circuit, obfuscating the true functionality. The key, which correctly programs the elements, is stored in an on-chip, tamper-proof memory. This key is set post-manufacture, so the correct functionality is never revealed to the untrusted foundry.

Early examples of logic locking techniques insert keyed exclusive-or (XOR) and multiplexer (MUX) gates to corrupt the next-state logic [10, 14]. Unfortunately, these methods have been largely broken using a variety of attacks, the most successful of which are miter-based SAT attacks [18]. Researchers have attempted to increase the difficulty of the miter-based attack by inserting resistant logic blocks into the locked circuit [22, 21]. These techniques reduce the number of keys ruled out per attack iteration, significantly increasing the overall execution

time; however, the logic blocks are susceptible to removal attacks since the circuitry is typically traceable through properties such as signal probability [22].

In response to the removal attacks, a new *strip-functionality* class of locking techniques has been developed. These techniques resist removal by stripping functionality from the circuit and re-establishing that functionality using the correct key. This class of technique currently includes TTLock [24], TTLock* [13], SFLL-HD [23], and SFLL-Flex [23]. Under this class of locking schemes, miter-based SAT attack resistance is maintained. Additionally, even though the locking circuitry can still be isolated and removed, the resulting circuit will still exhibit incorrect behavior.

In this paper, we explore the use of Boolean sensitivity in analyzing circuits locked with the strip-functionality class of techniques. Sensitivity is shown to be a powerful signal that can reveal flipped input patterns and thus the key to the locked circuit. Specifically, the contributions of this work are the following:

- Characterization of the sensitivities of a set of benchmark circuits
- Analysis of impact of strip-functionality class of locking techniques on sensitivity
- Development of a sensitivity-based SAT attack, detecting inputs with outlying sensitivity values
- An improved insertion technique for TTLock that mitigates this attack for certain circuits

# 2 Background

In this section, we present the most important background concepts related to this paper, including the attack model.

## 2.1 Digital Integrated Circuits

Digital ICs perform computations on Boolean-valued signals. Typically, they consist of interconnected logic gates and state elements that form a finite state machine (FSM). The interconnection of a circuit's gates and state is specified using a netlist. During *normal* operation, the FSM is evaluated periodically. Each cycle, the logic gates compute the current output values and the next values of the state elements based on the the current state and input values. During *test* mode, a scan-chain enables arbitrary reading and writing of a digital circuit's state. The scan-chain is a commonly used test infrastructure that forms a serial connection of all the circuit's state elements. This ability allows the logic of the circuit to be considered separately from the state elements; essential in testing a design, but a powerful attack vector for an adversary.

## 2.2 Attack Model

In the characterization of the security of a locking technique, an attack model is used to specify assumptions regarding the adversary's ability. In this paper and in the targeted class of techniques, it is assumed that the adversary has access to two artifacts: the locked circuit's netlist and an unlocked version of the circuit. The unlocked circuit has the correct key set in its tamper-proof memory, affording the attacker black-box access, commonly referred to as an oracle. These artifacts correspond to the access a foundry likely has when manufacturing a commercial design. The netlist can be easily reversed engineered from the design data and the

unlocked circuit can be obtained on the open market. It is also assumed that the adversary has access to the unlocked design's scan chains. While additional side-channel techniques may augment an attacker, they are considered outside the scope of the paper.

In general the problem being solved by the attacker is as follows. The attacker has a set of Boolean functions, obtained from the netlist. Each function, $f : \{0,1\}^{n+k} \rightarrow \{0,1\}$, has $n$ normal inputs and $k$ key inputs. The attacker also has an unlocked circuit to which an unknown, fixed key value is applied. The attacker can apply arbitrary inputs to the unlocked circuit, observing the corresponding outputs. The goal of the attacker is to obtain the key or a functionally equivalent version.

## 2.3 Brute Force Attack

Given the above attack model, a brute force attack establishes a baseline for the necessary key width of the circuit. This attack entails testing all possible values for the keys and inputs of a circuit. Each key-input combination is applied to the function. The output is compared to the unlocked circuit's output under the same input, ruling out keys when a difference is observed. Assuming a typical scan-chain frequency of 100MHz and a state size of 1000 bits, a query to the oracle can occur at a rate of 100kHz. With access to 1 unlocked circuit and 1 month of attack time, it is likely that the attacker can break any locking technique with under 38 bits of keys and inputs via brute force. A commonly assumed safe amount of key bits is 64.

## 2.4 Propositional Satisfiability

A successful approach to deal with hard combinatorial problems, such as finding the key of locked circuits, is to encode them into propositional logic and to solve the resulting propositional formulas with a satisfiability (SAT) solver. The performance of SAT solvers improved significantly in the last two decades and they are used for many applications in hardware and software verification [2, 9]. In recent years, SAT solvers have also been successfully applied to various attacks, such as hash collisions [17] and mathematical challenges [11].

A circuit can be encoded into propositional logic, specifically the conjunctive normal form (CNF) used by most SAT solvers, via the Tseytin transformation [19]. This transformation can take a circuit netlist and produce a set of clauses which, when collectively satisfied, will correspond to the original circuit's behavior.

## 2.5 Miter-Based SAT Attack

The above attack model enables the mounting of a more targeted, miter-based SAT attack. This attack uses the netlist and unlocked circuit to iteratively produce input-output (IO) relationships [18]. These relationships are used to rule out all keys that do not produce the same behavior, narrowing the space of possible circuit functionalities. The IO relationships are efficiently learned through a three-step procedure: **I.** First, a miter circuit is used to determine an input that is guaranteed to rule out at least a single key. A miter circuit consists of two copies of the original circuit with the inputs tied together, the key inputs kept separate, and the outputs connected to comparators. A diagram of the connections is shown in Fig. 1a. Additional key constraints, such as timing and loop breaking, can be conjuncted with the miter output. A SAT solver is used to find a setting of the shared input (I) and key inputs ($K_0, K_1$) such that the output of the miter circuit is logic 1. By construction, the solution to the SAT problem will have two different keys that, at that input value, disagree on the output value. The shared input value found by the solver is termed a differentiating input (DI). **II.** Next, as depicted in

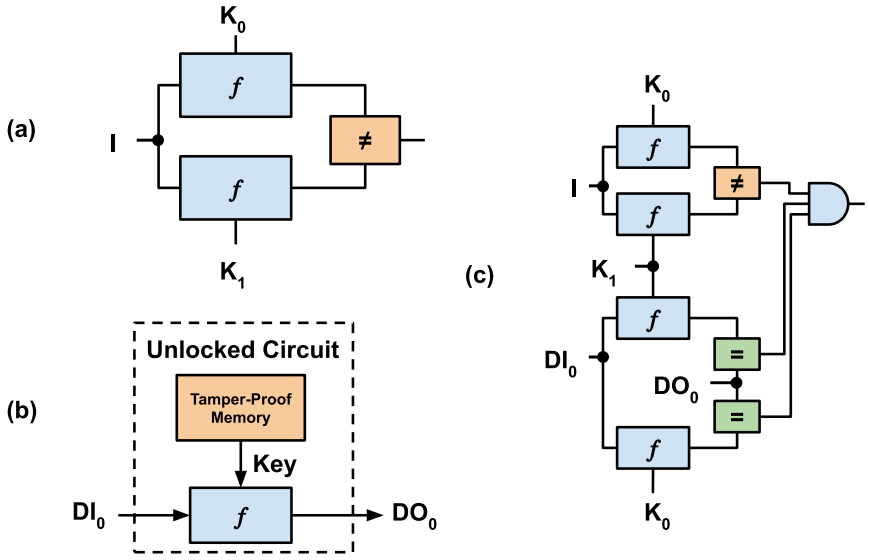


Figure 1: Miter-based SAT attack steps: (a) Miter circuit construction, (b) Unlocked (oracle) circuit produces correct IO functionality (c) Addition of learned IO constraint to miter circuit

Fig. 1b, the learned DI is applied to the oracle circuit to determine the differentiating output (DO), forming an input-output (IO) pair which the correct key must respect; any key that does not conform to this IO pair is incorrect. **III.** Finally, as shown in Fig. 1c, the IO pair is added as a constraint to the miter circuit for the next iteration. Now, any keys that satisfy the miter circuit will also satisfy the learned IO relationship. While each relationship is guaranteed to rule out at least one key, in practice, a larger portion of the key space is ruled out due to overlapping key functionalities at a given input. These steps repeat, adding more constraints until the miter circuit is unsatisfiable. At this point, any key that respects all learned IO relationships will be a functionally correct key.

## 2.6 Strip-Functionality Locking

Strip-functionality locking refers to a class of logic locking techniques that share a similar locking mechanism for directly defending against the miter-based SAT attack. This class includes TTLock, TTLock*, SFLL-HD, and SFLL-Flex. The class is characterized by securing a circuit through flipping a function's output for a small portion of the input space. The set of flipped inputs are referred to as protected inputs.

The generic structure of these techniques is shown in Fig. 2. The locked circuit consists of two layers, flip and restore. Both make use of a function, $P(I, K)$, that checks if an input, $I$, is part of the protected set determined by the key, $K$. Different values of $K$ will produce different protected sets.

The flip layer contains the original function, $f$, and a instance of $P$ with the key input hard coded at the correct value, $K^*$. The outputs of both functions are XOR'd together, inverting the original function for the protected inputs. We refer to the flip layer function as $f_{\text{flipped}}$; it is equivalent to $f$ except at the flipped values. The restore layer contains another instance of $P$, XOR'd with the output of the flip layer. When the key input, $K$, matches $K^*$, the

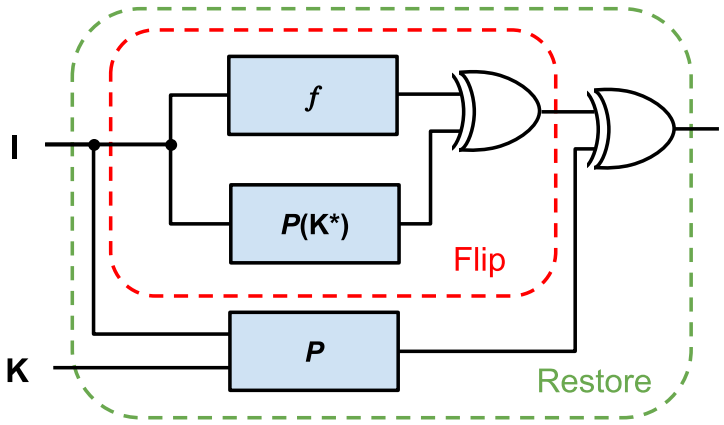


Figure 2: Underlying structure of strip-functionality locking

correct functionality of the original function is restored, each protected input value's output being flipped twice. We refer to the restore layer function as $f_{\text{locked}}$.

The whole circuit is synthesized together, mixing and reducing the logic from both layers along with $K^*$. In the given attack model, the adversary has access to this synthesized netlist. It is likely that the restore $P$ function remains intact as synthesis is unable to reduce the logic. However, the flip $P$ function and $K^*$ are usually combined with the logic of $f$ such that they are not recognizable via inspection.

These techniques resist the miter-based SAT attack because the overlap between the flipped input patterns for different keys is kept low. Thus, when an IO constraint is formed in the miter-based SAT attack, only a small number of keys are ruled out at once. While the exact scaling of the SAT-resistance depends on the specific technique, the class as a whole shows greatest miter-based SAT attack resistance when the number of protected inputs is minimal. Thus, there is an inherent trade-off between attack resistance and corruption of the circuit.

The specific techniques within the class are largely similar but each can be briefly described as following. TTLock is the original technique in this class; the method flips a single input pattern that is equal to the key. Thus in this incarnation, $P$ is a equality function. TTLock* is a version of TTLock which tries to mitigate any netlist-based attacks by converting the locked circuit to a reduced order Boolean decision diagram and resynthesizing. Effectively, this technique does a better job of mixing the $f$ with the flip layer's hard coded $K^*$ and $P$ function. SFLL-HD is a generalization of TTLock in which every input pattern a fixed Hamming distance from the key is flipped. In SFLL-HD, $P$ computes the Hamming distance between the key and input, then compares this to a fixed value to determine if the input is protected. Finally, SFLL-Flex stores a set of user-specified protected input patterns in a lookup table (LUT). In this case, $P$ is a function that is logic 0 for all inputs except the selected input values.

## 2.7 Boolean Sensitivity

Sensitivity is a simple complexity measure of a Boolean function [4]. Defined at a particular value in the input space, it is the number of inputs Hamming distance 1 from a particular input, for which the function produces a different value. Defined over the function, it is the maximum sensitivity of all inputs. We can specify each case more formally given a Boolean function, $f : \{0,1\}^n \rightarrow \{0,1\}$ and an input value, $x$. If $x_i$ represents the input value with the

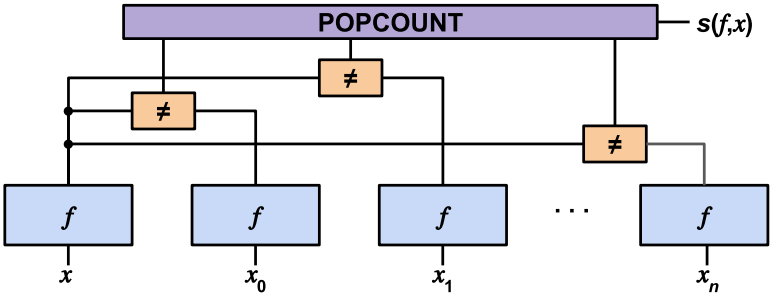


Figure 3: Circuit that determines sensitivity where $s(f, x)$ is the sensitivity of $f$ at a given input $x$ and $x_i$ represents $x$ with the $i$th bit flipped

$i$th bit flipped, the sensitivity of $f$ at $x$, $s(f, x)$, is the following:

$$s(f, x) = \frac{\sum_{i=1}^{n} f(x) \oplus f(x_i)}{n}$$

Thus, the sensitivity of $f$, $s(f)$, is:

$$s(f) = \max_{x} s(f, x)$$

Finally, we define the average sensitivity of $f$, $\bar{s}(f)$ as:

$$\bar{s}(f) = \frac{\sum_{x \in \{0,1\}^n} s(f, x)}{2^n}$$

# 3 Sensitivity Analysis

Since sensitivity is an easily computed metric, we can use it to efficiently analyze locked circuits. Ideally a secure locking scheme would leave no usable signal in the sensitivity domain. As we will show, this is not the case for strip-functionality locking.

## 3.1 Sensitivity of Benchmark Circuits

To understand the behavior of sensitivity, we consider a set of benchmark circuits [3], which are commonly used in the logic locking and circuit testing communities. Each circuit contains several Boolean functions. For each function, $f$, we estimate the average sensitivity across all inputs, $\bar{s}(f)$, and find the sensitivity of the function, $s(f)$.

The average sensitivities are determined by sampling a set of randomly selected values in the input space of each function. The sensitivity, $s(f, x)$, is evaluated at each input value by calculating the output value of the input value and all neighbors Hamming distance 1 away, summing the number of disagreements. This random input selection is the same method used to pick keys in the strip-functionality techniques, giving us an idea of the likely sensitivity of the picked inputs.

To find the sensitivity of each function, $s(f)$, we build a circuit that quantifies the sensitivity at a given input. This circuit, shown in Fig. 3, is made up of $n+1$ copies of the function, where

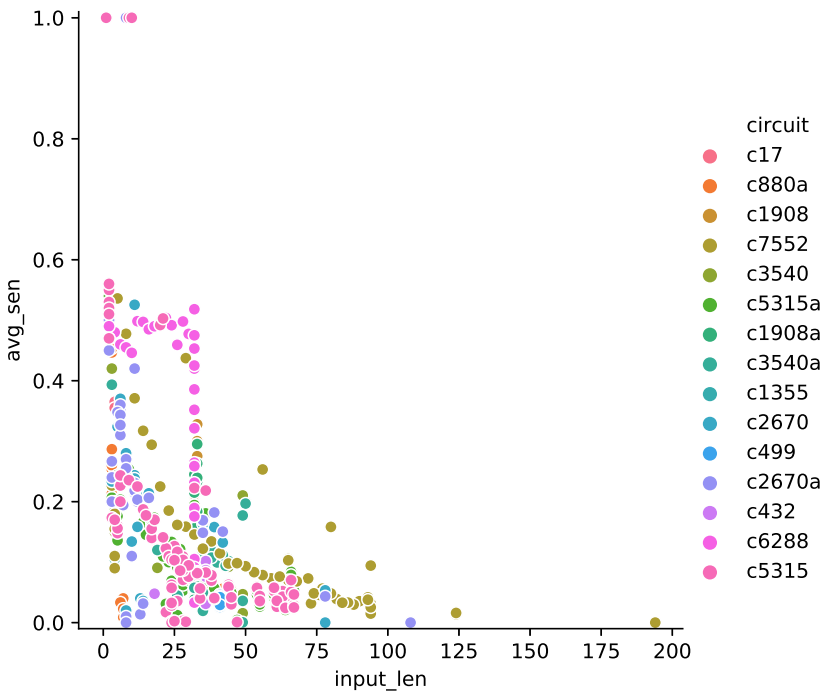


Figure 4: Average sensitivity, $\bar{s}(f)$, of benchmark circuits from 50 samples

$n$ is the width of the function's input. The inputs of the first copy of the function are tied to the $n$ additional copies in the following manner. If $x_i^j$ signifies the input of the $j$th function with the $i$th bit flipped:

$$x^i = x_i^0 : i \in (1...n)$$

The outputs are fed to comparators and subsequently a population count that determines the sensitivity at the input. The circuit is loaded into a SAT solver and the output sensitivity value is constrained to value, $s$. Starting from $n$, $s$ is decremented until a satisfying assignment can be found for the circuit. The first input found will have a sensitivity value, $s/n$, corresponding to the sensitivity value of the function.

The results of this analysis are shown in Fig. 4 and 5. A clear trend in the average sensitivities is seen in which the local sensitivity is inversely proportional to the input width. From a designer's perspective, this makes intuitive sense since specifying a complex function of many inputs is difficult. A notable outlier is c6288, a multiplier circuit, which maintains a high average local sensitivity for larger inputs. A similar scenario is seen for the function maximum sensitivities. The trend remains roughly the same, just shifted upwards. Here we have just captured the upper bound of the input sensitivities for each function.

## 3.2 Sensitivity of Strip-Functionality Circuits

Analyzing the effect of strip-functionality locking on a protected input's sensitivity, we see that the sensitivity is inverted. Consider a input value, $x_{\text{protected}}$, chosen at random to become a

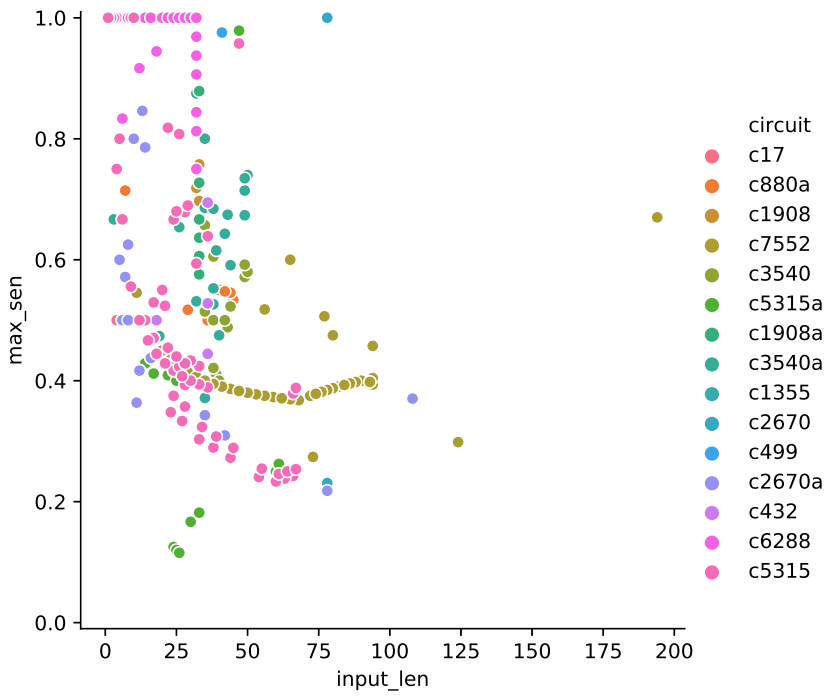


Figure 5: Sensitivity, $s(f)$, of benchmark circuits

protected input. In the original circuit, if the input has a low sensitivity, most inputs Hamming distance 1 away will agree. Looking back at Fig. 2, the locking procedure will invert the output value for this input value using the flip layer's $P$ function. This means that most of neighboring inputs now disagree with the protected input. Specifically, where $f$ is the original circuit and $f_{\text{flipped}}$ is the locked circuit without the restoration circuitry (ie. just the flip layer), the new sensitivity is:

$$s(f_{\text{flipped}}, x_{\text{protected}}) = 1 - s(f, x_{\text{protected}})$$

Thus, the protected input is moved to the opposite end of the sensitivity distribution. As established in Section 3.1, the average sensitivity for typical circuits decreases as the function's input width increases. This implies that at a sufficiently large input width, if an input is randomly selected as a protected pattern, the new sensitivity of this input will go from an *average low* to an *outlying high* value. Selecting inputs with large input width is motivated by increased brute force and miter-based SAT attack resistance. As the selection for the strip-functionality techniques does not consider the sensitivity of the protected pattern, it is likely that the protected input pattern will have a final input sensitivity that is an outlier in the high end of the distribution. In the next section, we show how this signal can be targeted by an attack.

**Algorithm 1:** Sensitivity-Based Attack

**Input:** $n$, $f$, $f_{\text{flipped}}$
**Output:** $x_{\text{protected}}$

```
1  sen := n;
2  block := ∅;
3  while sen > 0 do
4  |  CNF := (s(f_flipped, x) = sen) ∧ block;
5  |  if SAT[CNF] then
6  |  |  x_protected := SAT_ASSIGNMENT_x[CNF];
7  |  |  if f(x_protected) ≠ f_flipped(x_protected) then
8  |  |  |  return x_protected
9  |  |  end
10 |  |  block := block ∧ (x_protected ≠ x);
11 |  else
12 |  |  sen := sen − 1;
13 |  end
14 end
```

### 3.3 Sensitivity-Based Attack

Using the same sensitivity quantifying circuit from Fig. 3, we can build an attack algorithm that will detect inputs with high sensitivity. The first step in building such an attack is preprocessing the locked circuit. We find a function, $f_{\text{locked}}$, in the circuit which has the restoration unit in its fan-in. This can be done by tracing the key inputs through the circuitry. The restoration unit is disabled, creating a circuit functionally equivalent to $f_{\text{flipped}}$. In TTLock, TTLock*, and SFLL-Flex this entails adding constraints such that the input is not equal to the key. In SFLL-HD, the Hamming distance between the input and key must not be at the fixed value.

After obtaining $f_{\text{flipped}}$, we build the sensitivity quantifying circuit. We add a constraint setting the unnormalized sensitivity to the maximum value, $n$ (the input width of the function). This instance is put into a SAT solver searching for an input with this sensitivity level. The sensitivity is decremented until a satisfying input is found. The satisfying input is then applied to the oracle, $f$. If the output is the same as the simulated result from $f_{\text{flipped}}$, a constraint ruling out this input is added to a set of blocking clauses, *block*, and the process continues, searching for the next highest sensitivity input. If the output is different, it is a protected pattern. For TTLock and TTLock* this pattern is the key. For SFLL-Flex, this process must be repeated until all protected inputs in the LUT are found. Finally, for SFLL-HD, a total of three patterns with a mutual Hamming distance of twice the fixed value are found. The value of each key bit is then determined by the taking majority of the discovered protected inputs. The pseudo-code of the attack is listed in Algorithm 1.

## 4 Sensitivity Attack Resistant TTLock

For certain circuits, TTLock can be adapted such that the sensitivity-based attack is no longer effective. We demonstrate this process to show the limits of our attack method, however, we do not see this fixed TTLock as a viable locking method as the amount of output corruption is too small to be meaningful.

**Algorithm 2:** Sensitivity Attack Resistant TTLock

**Input:** set of functions $F$, set of input widths $N$, set of average sensitivities $A$
**Output:** optimal protected input $x_{\text{protected}}$, function $f$

```
b := 0;
while b < max(N) do
    for f ∈ F do
        n := N[f];
        a := A[f];
        CNF := |a − (1 − s(f, x))| ≤ b/n;
        if SAT[CNF] then
            x_protected := SAT_ASSIGNMENT_x[CNF];
            return x_protected, f
        end
    end
    b := b + 1;
end
```

Resisting the sensitivity-based attack can be achieved by selecting an input that, after flipping its output value, is not a sensitivity outlier. This means locking an input that will subsequently be moved to a dense part of the sensitivity distribution. Here, we implement an algorithm that targets the average sensitivity as the final value. Resistance to the sensitivity-based attack must be balanced with brute force and miter-based SAT attack resistance in which the function's input width determines the expected number of iterations. This entails locking an output function that has at least a given number of inputs. Thus from all output functions in the circuit with input width greater than the required value, we want to find the function, $f$, and input value, $x$, such that:

$$\arg\min_{x,f} |\bar{s}(f) - (1 - s(f, x))|$$

To find the optimal input, we first rule out all functions that do not meet the desired brute force and miter-based SAT attack resistance. For each function in the list of remaining functions, $f \in F$, we compute a mapping of input widths, $N : f \to n$, and of estimates of the average sensitivities, $A : f \to \bar{s}(f)$, using the same method from Section 3.1. We then search for the function and input pair which has a flipped sensitivity closest to the average value for the function. This is done by iteratively relaxing a bound, $b$, until a function is found that has an input with flipped sensitivity less than $b/n$ from the function's average sensitivity. After finding the optimal input, it is flipped following the original TTLock method. The pseudo-code of the algorithm to find the optimal protected input is shown in Algorithm 2.

Like TTLock, a sensitivity attack resistant version of SFLL-Flex can be easily created by repeating this process multiple times. However, SFLL-HD is harder to make resistant to the sensitivity-based attack. To avoid easy detection, all flipped inputs must have low sensitivity. Finding a set of inputs with this property and are all a fixed Hamming distance from a common value is unlikely.

Table 1: Sensitivity attack results for author-provided circuits using Cadence JasperGold

| Technique | Circuit | $N_{bits}$ | Time(s) | $N_{\text{iter}}$ |
|---|---|---|---|---|
| TTLock | c5315 | 32 | 3 | 1 |
| TTLock | c7552 | 32 | 3 | 1 |
| SFLL-HD | DFX | 256 (HD=32) | 584 | 3 |

Table 2: Sensitivity attack results for generated circuits using Cadence JasperGold. For SFLL-HD, HD $= N_{\text{bits}}/8$ and for SFLL-Flex, $N_{\text{patterns}} = N_{\text{bits}}/8$.

| | | TTLock | | TTLock* | | SFLL-HD | | SFLL-Flex | | TTLock-Sen | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Circuit | $N_{\text{bits}}$ | Time(s) | $N_{\text{iter}}$ | Time(s) | $N_{\text{iter}}$ | Time(s) | $N_{\text{iter}}$ | Time(s) | $N_{\text{iter}}$ | Time(s) | $N_{\text{iter}}$ |
| c499 | 32 | 1 | 2 | 70 | 6 | 2 | 3 | 8 | 25 | timeout | 3106 |
| c880 | 32 | 1 | 1 | 4 | 4 | 2 | 4 | 3 | 8 | timeout | 3480 |
| c1355 | 32 | 1 | 3 | 3849 | 181 | 2 | 3 | 5 | 14 | timeout | 3492 |
| c1908 | 32 | 1 | 1 | 18 | 2 | 3 | 4 | 3 | 6 | timeout | 3339 |
| c2670 | 32 | 1 | 1 | 6 | 1 | 2 | 3 | 2 | 4 | 68 | 62 |
| c2670 | 64 | 1 | 1 | 5 | 1 | 4 | 3 | 7 | 8 | 70 | 62 |
| c3540 | 32 | 2 | 2 | 24 | 1 | 2 | 3 | 2 | 6 | timeout | 2096 |
| c5315 | 32 | 2 | 3 | 1 | 1 | 2 | 3 | 2 | 5 | timeout | 3028 |
| c5315 | 64 | 4 | 9 | 3 | 1 | 4 | 3 | 10 | 14 | 24 | 26 |
| c7552 | 32 | 1 | 1 | 4 | 1 | 3 | 3 | 69 | 16 | timeout | 2700 |
| c7552 | 64 | 2 | 1 | 19 | 1 | 3 | 3 | 4 | 4 | timeout | 2568 |
| c7552 | 128 | 4 | 1 | 16 | 1 | 14 | 3 | 9 | 8 | timeout | 1670 |

# 5 Attack Results

To assess the strength of our sensitivity-based attack against strip-functionality locking as well as our modified version of TTLock, we ran three experiments. First using a commercially available tool, Cadence JasperGold, we demonstrate the attack's applicability on a set of locked benchmarks provided by the authors of the respective techniques from the strip-functionality class. To further validate these results, we then extend this analysis to an additional set of generated locked circuits. Finally, we implement the attack using open source tools and repeat the analysis on the generated circuits.

All attacks are run using a 64GB, 24-core, 2.2GHz machine. JasperGold, the commercial tool, is a formal verification suite that uses a parallel execution strategy attempting to find a solution employing several different solvers at once. In this case, our attack algorithm is implemented in TCL, a widely adopted scripting language used in digital IC design tools. Our open source flow, uses the CaDiCaL SAT solver [1] and Python to implement the attack algorithm. For all experiments, we limit each run to a timeout of 4 hours. The implementation of this flow can be found in our repository[1].

In Table 1, we present the results of our attack on the circuits provided by the authors. For each circuit we show the number of bits used to lock it, the overall time to execute the attack, and the number of iterations (the number of inputs checked in the oracle). As seen, all circuits are broken, most in seconds, the largest in minutes. This is significantly faster than the expected miter-based SAT attack time which scales exponentially in the width of the key. As

[1] https://github.com/jpsety/sensitivity_attack

Table 3: Sensitivity attack results for generated circuits using the SAT solver CaDiCaL. For SFLL-HD, HD = $N_{\text{bits}}/8$ and for SFLL-Flex, $N_{\text{patterns}} = N_{\text{bits}}/8$.

| | | TTLock | | TTLock* | | SFLL-HD | | SFLL-Flex | | TTLock-Sen | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Circuit | $N_{\text{bits}}$ | Time(s) | $N_{\text{iter}}$ | Time(s) | $N_{\text{iter}}$ | Time(s) | $N_{\text{iter}}$ | Time(s) | $N_{\text{iter}}$ | Time(s) | $N_{\text{iter}}$ |
| c432 | 32 | 0.20 | 1 | 0.15 | 1 | 0.67 | 3 | 0.63 | 4 | timeout | 11018 |
| c499 | 32 | 0.06 | 2 | 26.64 | 1 | 0.36 | 3 | 57.14 | 918 | timeout | 12243 |
| c880 | 32 | 0.07 | 1 | 0.14 | 1 | 0.67 | 3 | 0.37 | 4 | timeout | 13159 |
| c1355 | 32 | 65.21 | 989 | 1271.27 | 377 | 0.59 | 3 | 425.36 | 2982 | timeout | 11938 |
| c1908 | 32 | 0.13 | 1 | 0.67 | 1 | 1.08 | 3 | 0.46 | 4 | timeout | 17602 |
| c2670 | 32 | 0.05 | 1 | 0.05 | 1 | 0.43 | 3 | 0.19 | 4 | 1.03 | 1 |
| c2670 | 64 | 0.06 | 1 | 0.10 | 1 | 1.27 | 3 | 0.49 | 8 | 4.49 | 1 |
| c3540 | 32 | 0.34 | 2 | 9.55 | 1 | 0.69 | 3 | 0.27 | 4 | timeout | 3442 |
| c5315 | 32 | 0.19 | 3 | 0.04 | 1 | 1.58 | 3 | 0.31 | 4 | timeout | 13670 |
| c5315 | 64 | 0.12 | 1 | 0.07 | 1 | 2.82 | 3 | 0.99 | 8 | 2.81 | 1 |
| c7552 | 32 | 0.05 | 1 | 0.10 | 1 | 0.38 | 3 | 0.16 | 4 | timeout | 10079 |
| c7552 | 64 | 0.07 | 1 | 0.72 | 1 | 2.11 | 3 | 0.53 | 8 | timeout | 12286 |
| c7552 | 128 | 0.08 | 1 | 0.08 | 1 | 5.78 | 3 | 2.06 | 16 | timeout | 7667 |

sensitivity is not considered in these locking schemes, in all cases the protected inputs are the highest sensitivity inputs keeping the required number of iterations small.

For the next experiments, we implement the strip-functionality locking techniques and lock a commonly used set of benchmark circuits [3]. We generate locked circuits starting at 32 key bits, doubling the count until the circuit no longer has a function with at least that input width. We set both the Hamming distance used in SFLL-HD and the number of patterns used in SFLL-Flex to $N_{\text{bits}}/8$. Since the protected inputs are chosen randomly and thus will likely have high sensitivity after being flipped, the impact of these parameter choices will likely have negligible effect on the attack result. Immediately clear from the locking procedure is that the smaller circuits don't contain a Boolean function with input width large enough to provide adequate security against a brute force attack; only three of the tested circuits can scale to 64 bits. However, since the functions with lower input width are likely to have higher average sensitivities and thus a lower chance of a protected input being an outlier, these circuits should be the most resistant to our attack.

The commercial tool attack results for the generated locked circuits shown in Table 2. We are able to deobfuscate all circuits locked with previous strip-functionality methods. With the exception of a few outliers, all protected inputs are found in the several attack iterations and in seconds of run time. The results for the sensitivity attack resistant version are mixed depending on the circuit and the number of bits. Two notable examples are c2670 and c5315. The first circuit, c2670, has very few high-sensitivity inputs; thus, when it is locked with our flow, the protected input is still in the highest portion of the sensitivity distribution. The second circuit, c5315, has a suitable protected input for the 32-bit locking, but not for the 64-bit. Therefore, in this circuit there is a distinct tradeoff between sensitivity attack and miter-based SAT attack resistance. In general, it is clear that the resistance of this locking technique is highly circuit dependent.

Finally, in Table 3, we show the open source flow results for the generated circuits. The results are similar to the commercial flow, however the execution times are lower. All previous strip-functionality circuits are again deobfuscated and the sensitivity attack resistant TTLock circuits have matching results. The open source flow is able to explore a greater portion of the input space in the allotted time.

# 6 Discussion

The strip-functionality class of techniques was developed in response to the miter-based SAT attack. For previous locking schemes, the miter-based attack ruled out large portions of the key space with each learned IO pair. Strip-functionality locking minimizes the number of keys ruled out per IO pair by minimizing the number of protected inputs per key. As the authors of these techniques have noted, the small number of protected inputs decreases the amount of corruption in the circuit under an incorrect key. The trade-off is unsatisfying: the attacker has a harder time searching for the incorrect inputs, but the probability of those incorrect inputs being encountered reduces just as much.

Even further, we have shown that Boolean sensitivity is a strong signal to detect the few protected inputs. Under the right lens, what may seem to be an undetectable change becomes obvious. Thus, this class of techniques has a tradeoff between miter-based SAT attack resistance and sensitivity attack resistance. The result is a narrowed set of situations under which strip-functionality locking is applicable.

We provide a means to find inputs that will not be easily detected with the sensitivity attack via the sensitivity attack resistant TTLock. However, the locked circuits must have high sensitivity nodes that can be moved into denser areas of the sensitivity distribution, making the security of the technique highly circuit dependent. Notable examples that have this property are cryptographic circuits and algebraic circuits. While both classes of circuits will likely have many inputs that can be hidden from the sensitivity attack, it should be noted that these classes are highly regular structures that may be subject to different types of analysis.

Furthermore, if our version of TTLock or a similar construction is used, it would be prudent to more accurately characterize the sensitivity distribution before selecting inputs to lock. This could be done using approximate model counting techniques [16] on the sensitivity circuit. Additionally, it should be noted that the sensitivity attack speed can likely be substantially improved via parallelism and utilization of AllSAT solvers. Finally, we make no claims as to the security of our proposed technique against other attack methods.

# 7 Conclusion

In this paper, we have used Boolean sensitivity to analyze circuits obfuscated with a class of logic locking techniques. We show that if the locking procedure does not consider the sensitivity during insertion, the locked circuit will likely be easily deobfuscated by our attack. We present a scheme to avoid such attacks, however the resulting resistance will be highly circuit dependent.

In future work we want to further explore how to make circuits resistant to SAT-based attacks. Many families of small formulas are known to be hard for today's solvers ranging from uniform random formulas [5] to theoretical challenges and instances designed to obstruct state-of-the-art solver techniques [7, 20]. Most small hard formulas are unsatisfiable, but some are also satisfiable [8], which is required for circuit locking. We consider it an important challenge to determine whether small formulas that are hard for SAT solvers can be turned into an effective locking method.

**Acknowledgments.** We thank the reviewers for their comments that helped improve the paper. This work was supported in part by the Defense Advanced Research Projects Agency under contract FA8750-17-1-0059 "Obfuscated Manufacturing for GPS (OMG)", Honeywell Federal Manufacturing & Technologies, LLC under contract A023646, and NSF grant CCF-1813993.

# References


[1] Armin Biere. Cadical, lingeling, plingeling, treengeling and yalsat entering the sat competition 2018. In *Proceedings of SAT Competition 2018*, pages 13–14, 2018.

[2] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In *TACAS '99: Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, pages 193–207, London, UK, 1999. Springer-Verlag.

[3] Franc Brglez and Hideo Fujiwara. A neutral netlist of 10 combinational benchmark circuits and a targeted translator in fortran. *Special session on ATPG and fault simulation, Proc. IEEE International Symposium on Circuits and Systems*, pages 663–698, 06 1985.

[4] Nadia Creignou and Hervé Daudé. Sensitivity of Boolean formulas. *European Journal of Combinatorics*, 34(5):793–805, 7 2013.

[5] John Franco and Marvin Paull. Probabilistic analysis of the davis putnam procedure for solving the satisfiability problem. *Discrete Applied Mathematics*, 5(1):77 – 87, 1983.

[6] Ujjwal Guin, Ziqi Zhou, and Adit Singh. A novel design-for-security (DFS) architecture to prevent unauthorized IC overproduction. In *Proceedings of the IEEE VLSI Test Symposium*, 2017.

[7] Armin Haken. The intractability of resolution. *Theor. Comput. Sci.*, 39:297–308, 1985.

[8] Edward A. Hirsch. Sat local search algorithms: Worst-case study. *Journal of Automated Reasoning*, 24(1):127–143, Feb 2000.

[9] Franjo Ivančić, Zijiang Yang, Malay K. Ganai, Aarti Gupta, and Pranav Ashar. Efficient sat-based bounded model checking for software verification. *Theoretical Computer Science*, 404(3):256 – 274, 2008. International Symposium on Leveraging Applications of Formal Methods (ISoLA 2004).

[10] F. Koushanfar J. A. Roy and I. L. Markov. EPIC: Ending Piracy of Integrated Circuits. *2008 Design, Automation and Test in Europe*, 2008.

[11] Boris Konev and Alexei Lisitsa. A sat attack on the erdős discrepancy conjecture. In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing – SAT 2014*, pages 219–226, Cham, 2014. Springer International Publishing.

[12] Farinaz Koushanfar, Gang Qu, and Miodrag Potkonjak. Intellectual property metering. In *International Workshop on Information Hiding*, pages 81–95. Springer, 2001.

[13] Mohamed El Massad. *On the Foundations of Integrated Circuit Intellectual Property.* PhD thesis, New York University, 2019.

[14] 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.

[15] M. Rostami, F. Koushanfar, and R. Karri. A primer on hardware security: Models, methods, and metrics. *Proceedings of the IEEE*, 102(8):1283–1295, 2014.

[16] Mate Soos and Kuldeep S Meel. Bird: Engineering an efficient cnf-xor sat solver and its applications to approximate model counting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1592–1599, 2019.

[17] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. The first collision for full sha-1. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, pages 570–596, Cham, 2017. Springer International Publishing.

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

[19] Grigori S Tseitin. On the complexity of derivation in propositional calculus. In *Automation of reasoning*, pages 466–483. Springer, 1983.

[20] Alasdair Urquhart. The complexity of propositional proofs. *The Bulletin of Symbolic Logic*,

1(4):425–467, December 1995.

[21] Muhammad Yasin, Bodhisatwa Mazumdar, Jeyavijayan J V Rajendran, and Ozgur Sinanoglu. SARLock: SAT attack resistant logic locking. *Proceedings of the 2016 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2016*, pages 236–241, 2016.

[22] Muhammad Yasin, Bodhisatwa Mazumdar, Ozgur Sinanoglu, and Jeyavijayan Rajendran. Security analysis of anti-sat. In *2017 22nd Asia and South Pacific Design Automation Conference, ASP-DAC 2017*, pages 342–347. Institute of Electrical and Electronics Engineers Inc., 2 2017.

[23] Muhammad Yasin, Abhrajit Sengupta, Mohammed Thari Nabeel, Mohammed Ashraf, Jeyavijayan (JV) Rajendran, and Ozgur Sinanoglu. Provably-secure logic locking: From theory to practice. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 1601–1618, New York, NY, USA, 2017. Association for Computing Machinery.

[24] Muhammad Yasin, Abhrajit Sengupta, Benjamin Carrion Schafer, Yiorgos Makris, Ozgur Sinanoglu, and Jeyavijayan (JV) Rajendran. What to Lock? Functional and Parametric Locking. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*, GLSVLSI '17, page 351–356, New York, NY, USA, 2017. Association for Computing Machinery.