Strong Anti-SAT: Secure and Effective Logic Locking

Yuntao Liu, Michael Zuzak, Yang Xie, Abhishek Chakraborty, Ankur Srivastava
{ytliu,mzuzak,yangxie,abhi1990,ankurs}@umd.edu
University of Maryland, College Park

ABSTRACT
Logic locking has been proposed as strong protection of intellectual property (IP) against security threats in the IC supply chain especially when the fabrication facility is untrusted. Such techniques use additional locking circuitry to inject incorrect behavior into the digital functionality when the key is incorrect. A family of attacks known as “SAT attacks” provides a strong mathematical formulation to find the correct key of locked circuits. Many conventional SAT-resilient logic locking schemes fail to inject sufficient error into the circuit when the key is incorrect: there are usually very few (or only one) input minterms that cause any error at the circuit output [18, 20–22]. The state-of-the-art stripped functionality logic locking (SFLL) [24] technique provides a wide spectrum of configurations which introduced a trade-off between security (i.e. SAT attack complexity) and effectiveness (i.e. the amount of error injected by a wrong key).

In this work, we prove that such a trade-off is universal among all logic locking techniques. In order to attain high effectiveness of locking without compromising security, we propose a novel secure and effective logic locking scheme, called Strong Anti-SAT (SAS). SAS has the following significant improvements over existing techniques. (1) We prove that SAS’s security against SAT attack is not compromised by increases in effectiveness. (2) In contrast to prior work which focused solely on the circuit-level locking impact, we integrate SAS-locked modules into an 80386 processor and show that SAS has a high application-level impact. (3) SAS’s hardware overhead is smaller than that of existing techniques.

KEYWORDS
Logic Locking, SAT Attack, IC Supply Chain Security

1 INTRODUCTION
Due to the increasing cost of maintaining IC foundries with advanced technology nodes, many chip designers have become fabless and outsource their fabrication to off-shore foundries. However, such foundries are not under the designer’s control which puts the security of the IC supply chain at risk. Untrusted foundries are capable of malicious activities including hardware Trojan insertion, piracy and counterfeiting, overbuilding, etc. Many design-for-trust techniques have been studied as countermeasures among which logic locking has been the most widely studied [3]. A logic locked circuit requires a secret key input and the correct key is kept by the designer and not known to the foundry. The functionality of the circuit is correct only if the key is correct. After the foundry manufactures the locked circuit and returns it to the designer, the correct key is applied to the circuit by connecting a tamper-proof memory containing the key to the key inputs. This process is called activation. Over the years, different types of logic locking mechanisms have been suggested. Initially, locking involved inserting XOR/XNOR gates in a synthesized design netlist [11]. Later, techniques based on VLSI testing principles have been outlined to improve logic locking schemes by manifesting high corruption at the output bits when an incorrect key is applied [9, 10].

The Boolean satisfiability-based attack, a.k.a. SAT attack [16] was a game changer and became the basis of many variants [4, 13, 14]. SAT provides a strong mathematical formulation to find the correct locking key of a logic locked IC which prunes out wrong keys in an iterative manner. In each iteration, an input (called the Distinguishing Input, or DI) is chosen by the SAT solver and all the wrong keys that corrupt the output of this DI are pruned out. All wrong keys are pruned out when no more DI can be found. Point function (PF)-based logic locking, including SARLock [21] and Anti-SAT [18, 20], force the number of SAT iterations to be exponential in the key size by pruning out only a very small number of wrong keys in each iteration. However, PF-based locking necessitates that there are very few (or only one) input minterms whose output is incorrect for each wrong key. Hence the overall error rate of the locked circuit with a wrong key is very small. This disadvantage is captured by approximate SAT attacks such as AppSAT [13] and Double-DIP [14]. These attack schemes are able to find an approximate key (approx-key) which makes the locked circuit behave correctly for most (but not all) of the input values.

More recently, Yasin et al. proposed stripped functionality logic locking (SFLL) which allows the designer to select a set of protected input patterns that are affected by almost all the wrong keys while other input patterns are affected by very few wrong keys [24]. However, when the number of protected patterns increases, SAT attacks need significantly fewer iterations to find the correct key. Essentially, SFLL creates a fundamental trade-off between security (i.e. SAT attack complexity) and effectiveness (i.e. the amount of error injected by a wrong key). This trade-off is problematic. On the one hand, if only very few input patterns are protected, a wrong key may not inject enough error into the circuit and useful work may still be done using the chip, rendering locking ineffective. On the other hand, having more protected input patterns will compromise the circuit’s security against SAT attack. Addressing this dilemma is the main theme of our paper.

We propose Strong Anti-SAT (SAS) to address the challenges in achieving high effectiveness without sacrificing security. SAS ensures that, given any wrong (including approximate) key, the error injected by locking circuitry will have a significant application-level impact. Additionally, SAS is provably resilient to SAT attacks (i.e. requiring exponential time). This is a substantial improvement over the limitations posed by SFLL. The contribution of this work is as follows.

(1) We prove the fundamental trade-off between security and effectiveness which is applicable to any logic locking scheme.

(2) We demonstrate the inability of existing locking techniques to secure hardware running real-world workloads due to such a trade-off. We show that, when the longest combinatorial path (i.e. the multiplier) in a 32-bit 80386 processor
2 BACKGROUND

2.1 Threat Model

Fig. 1 illustrates the threat model we consider which is consistent with the latest papers in the logic locking field [5, 13, 17–19, 21, 24]. The attacker can be either an untrusted foundry or an untrusted user who has the ability to reverse engineer the fabricated chip, obtaining the locked gate-level netlist. The attacker is considered to have the following resources:

1. The locked gate-level netlist of the circuit under attack. This can be obtained by reverse engineering the GDS-II file (which the foundry has) or a fabricated chip (which can be done by a capable end user).

2. An activated chip. The attacker is considered to own an activated chip (i.e. the one loaded with the correct key) since such a chip can be purchased from the open market.

In general, logic locking research does not assume that the attacker is able to insert probes into the activated circuit, i.e. to observe the intermediate values. This is because protection schemes (e.g. analog shield [8]) can counter probing attacks.

The Boolean satisfiability-based attack, a.k.a. SAT attack is a strong theoretical formulation to find the correct key of a locked circuit. In the context of the SAT attack, we use the Conjunctive Normal Form (CNF): 

\[ C(X, \beta, \gamma) \rightarrow C(\bar{X}, \beta, \gamma) \text{ to characterize Boolean satisfiability: } C(X, \beta, \gamma) = \text{TRUE if } X, \beta, \gamma \text{ satisfy } \gamma = F_L(\bar{X}, \beta), \text{ where } F_L \text{ stands for the Boolean functionality of the locked circuit. } C(X, \beta, \gamma) = \text{FALSE otherwise. } \]

SAT attacks run iteratively and prune out incorrect keys in every iteration. The attack consists of the following steps:

1. In the initial iteration, the attacker looks for a primary input, \( \bar{X}_1 \), and two keys, \( \bar{K}_\alpha \) and \( \bar{K}_\beta \), such that the locked circuit produces two different outputs \( \bar{Y}_\alpha \) and \( \bar{Y}_\beta \):

\[ C(\bar{X}_1, \bar{K}_\alpha, \bar{Y}_\alpha) \land C(\bar{X}_1, \bar{K}_\beta, \bar{Y}_\beta) \land (\bar{Y}_\alpha \neq \bar{Y}_\beta) \]  

\( \bar{X}_1 \) is called the Distinguishing Input (DI).

2. The DI, \( \bar{X}_1 \), is applied to the activated circuit (the oracle) and the output \( \bar{Y}_1 \) is recorded. Note that \( \bar{K}_\alpha, \bar{Y}_\alpha \), and \( \bar{K}_\beta, \bar{Y}_\beta \) are not recorded. Only the DI and its correct output are carried over to the following iterations.

3. In the \( i^{th} \) iteration, a new DI and a pair of keys, \( \bar{K}_\alpha \) and \( \bar{K}_\beta \), are found. The newly found \( \bar{K}_\alpha \) and \( \bar{K}_\beta \) should produce correct outputs for all the DIs found in previous iterations. To this end, we append a clause to Eq. (1):

\[ C(\bar{X}_i, \bar{K}_\alpha, \bar{Y}_\alpha) \land C(\bar{X}_i, \bar{K}_\beta, \bar{Y}_\beta) \land (\bar{Y}_\alpha \neq \bar{Y}_\beta) \]  

(2)

In this way, all the wrong keys that corrupt the output of previously found DIs (i.e. the output is different from that of the activated chip) are pruned out from the search space.

4. SAT solves Eq. (2) repeatedly until no more DI can be found, i.e. Eq. (2) is not satisfiable any more.

5. In this case, there is no more DI. The output of the SAT attack is a key \( \bar{K} \) that produces the same output as the activated circuit to all the DIs, which can be expressed using the following CNF:

\[ \bigwedge_{i=1}^{\lambda} C(\bar{X}_i, \bar{K}, \bar{Y}_i) \]  

(3)

where \( \lambda \) is the total number of SAT iterations.

2.2 Logic Locking

Multiple logic locking schemes have been proposed to thwart the SAT attack [18, 20, 21, 23, 24]. There are two ways to mitigate the SAT attack: one is to increase the time for each SAT iteration and the other is to increase the number of SAT iterations. The former requires either AES blocks [23] or reconfigurable logic [7], which is impractical for most circuits. The other approach is to exponentially increase the number of SAT iterations. This approach is also not perfect because a locking scheme must be rather ineffective to improve security. This is the case for Anti-SAT [18, 20], SARLock [21], and TTL [22]. All these techniques are vulnerable to the approximate SAT attacks (such as AppSAT [13] and Double-DIP [14]).

The state-of-the-art, stripped functionality logic locking (SPLL) [24], explores the trade-off between security and effectiveness. SPLL comprises of two parts: a functionality stripped circuit (FSC) and a restore unit (RU). The FSC is the original circuit with the functionality modified for a set of protected input cubes. The RU stores the key, compares the circuit’s input with the key, and outputs a restore vector which is XOR’ed with the FSC output. If the key is correct, the restore vector will fix the FSC’s output and the circuit will have correct output. There are two variants of SPLL: SPLL-HD and SPLL-flex. SPLL-HD has been successfully attacked by a functional analysis based attack [15]. As the latter remains secure, provides higher flexibility in selecting protected cubes, and is more relevant to SAS, we focus on SPLL-flex in this paper. An SPLL-flex configuration can be described using the number of protected cubes, \( c \), and the number of specified bits \( d \) of each cube, \( k \), denoted as SPLL-flex\(^{c \times k}d \). The authors of [24] derived the following characteristics of a circuit locked with SPLL-flex\(^{c \times k}d \): (1) the fraction of input minterms whose output will be corrupted by a wrong key (i.e. the “error rate” of a wrong key) is \( c \cdot 2^{-k} \); and (2) the probability that a SAT attack finds the correct key within \( q \) iterations is \( q \cdot 2^{\log_{c\cdot2}\cdot k} \). We illustrate this relationship in Fig. 4. As a higher SAT success probability...
indicates weaker security, SFLL inherently suffers from a trade-off between security and effectiveness.

![Diagram of SFLL's error rate](image)

**Figure 2:** SFLL’s error rate of wrong keys vs. the probability of SAT finding the correct key in 100 iterations

The rest of the paper is organized as follows. We show that SFLL’s trade-off makes it infeasible to secure real-world applications in Section 3. We then mathematically prove that the trade-off applies to all logic locking schemes in Section 4. In Section 5, SAS’s hardware structure is presented and its exponential SAT attack complexity is proved in theory. Section 6 shows the experimental results which demonstrate that when the same set of critical minterms are selected by SAS and SFLL, SAS achieves higher security than SFLL while maintaining similar application-level effectiveness. Section 7 concludes the paper.

## 3 INSUFFICIENCY OF SFLL

In this section, we investigate the application-level effectiveness of SFLL [24]. Specifically, we lock the multiplier within a 32-bit 80386 processor since it is the largest combinational component. The application-level impact is evaluated using the PARSEC Benchmark Suite [1]. In order to evaluate the application-level impact of a logic locking scheme, we modify the GEM5 [2] simulator so that error is injected into the locked processor module according to the hardware error profile due to the wrong key. In this way, the circuit-level error induced by an incorrect (including approximate) key can be evaluated at the application level. This framework is illustrated in Fig. 3 which is similar to the strategy used in [6, 25].

![Diagram of experimental framework](image)

**Figure 3:** Our experimental framework

SPLL allows the designer to explore the trade-off between effectiveness and security. We show that a “sweet spot” does not exist. In our experiments, we lock the multiplier with various SPLL configurations, each having a different level of security against SAT attack, quantified by the average SAT iterations to unlock (as the X axis in Fig. 4). The effectiveness of locking is evaluated by running the PARSEC benchmarks on the locked processors loaded with approximate keys. The percentage of PARSEC benchmark runs with an incorrect outcome is the effectiveness criterion of each locking configuration. The trade-off is illustrated in Fig. 4 from which we observe that the wrong keys’ impact decreases with the increase in SAT complexity. In order to have a visible accuracy drop for most benchmarks, the SPLL locked processor cannot endure more than roughly 1000 SAT iterations. Such a locking scheme is extremely vulnerable since 1000 SAT iterations can be fulfilled within minutes. Therefore, a logic locking scheme that ensures high application-level impact without sacrificing SAT complexity is needed.

![Diagram of security vs. effectiveness trade-off](image)

**Figure 4:** Security vs. effectiveness trade-off for SPLL-locked processor running PARSEC benchmarks.

## 4 LOGIC LOCKING’S UNIVERSAL TRADE-OFF

This section generalizes the trade-off of SPLL to all logic locking schemes. We start with definitions of concepts and then prove the relationship between security and effectiveness.

**Definition 4.1.** We say that a key \( \tilde{K} \) corrupts a primary input minterm \( \tilde{X} \) if and only if the locked circuit produces a different output to \( \tilde{X} \) from the original circuit’s output, i.e. \( F_L(\tilde{X}, \tilde{K}) \neq F(\tilde{X}) \).

**Definition 4.2.** The error rate \( \epsilon_{\tilde{K}} \) of a key \( \tilde{K} \) is the portion of primary input minterms corrupted by the key \( \tilde{K} \).

Note that \( \epsilon_{\tilde{K}} = 0 \) for any correct key. Let \( \mathcal{K}_K \) be the set of input minterms corrupted by \( \tilde{K} \). Then, \( \epsilon_{\tilde{K}} = \frac{|\mathcal{K}_K|}{2^n} \), where \( n \) is the number of bits in the primary input. We use \( \epsilon \) to denote the average error rate across all the keys. When the key is \( k \) bits long,

\[
\epsilon = \frac{1}{2^k} \sum_{\tilde{K} \in \{0,1\}^k} \epsilon_{\tilde{K}}
\]

**Definition 4.3.** The corruptibility \( Y_{\tilde{X}} \) of a primary input minterm \( \tilde{X} \) is the portion of wrong keys that corrupt this minterm.

Let \( \mathcal{K}_W \) be the set of wrong keys that corrupts the primary input minterm \( \tilde{X} \) and \( \mathcal{K}_W \) be the set of wrong keys. Then, \( Y_{\tilde{X}} = \frac{|\mathcal{K}_W|}{|\mathcal{K}_W|} \). Let \( \gamma \) denote the average corruptibility over all input minterms, i.e.

\[
\gamma = \frac{1}{2^n} \sum_{\tilde{X} \in \{0,1\}^n} Y_{\tilde{X}}
\]

**Theorem 4.4.** The average error rate of all wrong keys equals the average corruptibility of all input minterms, i.e. \( \epsilon = \gamma \).

See proof in Appendix A. Let \( \lambda \) be the number of SAT iterations that a SAT attacker needs to find the correct key.

**Theorem 4.5.** The expected number of SAT iterations \( \mathbb{E}[\lambda] \) is lower bounded by \( \frac{1}{\gamma} \).

Proof. In each SAT iteration, the average number of wrong keys pruned by the DI \( \tilde{X} \) is upper bounded by \( \frac{1}{|\mathcal{K}_W|} \) (because some of the wrong keys may have already pruned out by DIs of previous iterations). Therefore,

\[
\mathbb{E}[\lambda] \geq \frac{|\mathcal{K}_W|}{|\mathcal{K}_W|} = \frac{1}{\gamma}
\]

Hence proved.
Theorems 4.4 and 4.5 explicitly point out that there exists an inverse relationship between $\epsilon$ and the lower bound of $E[\lambda]$. This quantifies the trade-off between them. This trade-off applies to any logic locking scheme. Note that different input minterms may inject a different amount of error at the application level. By assigning higher corruptibility to a few minterms with high application-level impact, we can achieve high effectiveness while maintaining high security by keeping $\gamma$ low and $E[\lambda]$ high. This is the main intuition behind SAS.

5 SAS'S ARCHITECTURE AND PROPERTIES

In Sec. 3 and 4, we demonstrated that two competing objectives exist for all logic locking schemes:

(1) **Effectiveness:** Any incorrect key should have a high application-level error impact.

(2) **Security:** The complexity of determining the correct key via SAT attacks should be very high.

In this section, we introduce **Strong Anti-SAT (SAS)** logic locking scheme which aims to achieve both objectives simultaneously. SAS guarantees an exponential expected SAT solving time while having a large impact on the accuracy of real-world applications. In SAS, instead of uniformly distributing the error across all possible inputs, we identify certain input patterns which potentially have a higher impact on the overall application-level error. We call these inputs **critical minterms**. SAS is configured in such a way that any incorrect key corrupts any critical minterm. For the other minterms, the corruptibility is low.

5.1 The SAS Block

Let $M$ be the set of critical minterms and $m = |M|$ be the number of critical minterms. For the ease of implementation, we always choose $m$ to be a power of 2. The basic locking infrastructure is the SAS block which is illustrated in Fig. 5. The key $K$ of an $n$-bit SAS block consists of two $n$-bit sub-keys, $K_1$ and $K_2$. In order to describe the mechanism of the SAS locking scheme clearly, we use a reverse order and start our illustration from the output side.

![Figure 5: The architecture of SAS configuration 1 with the details of the SAS Block](image)

**Y_{SAS}** is the output of the SAS block. If $Y_{SAS} = 1$, a fault will be injected into the original circuit. $g$ is a function with an on-set-size of 1, i.e. only one input minterm will have output 1 and all others will have output 0. $g$ has the opposite functionality of $f$. A function block $T^i = H(X, K_i)$ is inserted before $g$ and $g$ and it works as follows. If $X$ is not a critical minterm, then $T^i = X$. In this case, only one combination of $K_i$ will make output 1, therefore $X$ has a low corruptibility. If $X$ is a critical minterm, then for a portion of $K_1$, $X$ is adjusted according to $K_1$ to obtain $T^i$ such that $g(T^i, K_i) = 1$ and hence the corruptibility is increased. $T^i = H(X, K_i)$ further ensures that the wrong keys that corrupt each critical minterm are mutually exclusive and evenly partition the set of wrong keys. More specifically, as the partitioning is based on the $K_1$ part of the key, we have the following.

### Table 1: Illustration of how $m$ critical minterms partition the set of wrong keys

<table>
<thead>
<tr>
<th>$\tilde{K}_i$ of wrong keys</th>
<th>$\tilde{k}_1$</th>
<th>$\tilde{k}_2$</th>
<th>$\tilde{k}_m$</th>
<th>$\tilde{k}_{m+1}$</th>
<th>$\tilde{k}_{m+2}$</th>
<th>$\tilde{k}_{2n}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>critical minterms</td>
<td>$X_1$</td>
<td>$X_2$</td>
<td>$\ldots$</td>
<td>$X_{m-1}$</td>
<td>$X_m$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>non-critical minterms</td>
<td>$X_{m+1}$</td>
<td>$X_{m+2}$</td>
<td>$\ldots$</td>
<td>$X_{m+n-1}$</td>
<td>$X_{m+n}$</td>
<td>$\ldots$</td>
</tr>
</tbody>
</table>

Let $K^i_X = \{\tilde{K}_1\} \cup \tilde{K}_2$ such that $(\tilde{K}_1, \tilde{K}_2) \in K^W, (\tilde{K}_1, \tilde{K}_2) \in K^X$. Then we have

$$\forall \tilde{X}_1, \tilde{X}_2 \in M, |K^i_{\tilde{X}_1}| = |K^i_{\tilde{X}_2}|, K^i_{\tilde{X}_1} \land K^i_{\tilde{X}_2} = 0 \bigcup_{\tilde{X} \in M} K^i_{\tilde{X}} = \mathbb{Z}^n_2 \quad (4)$$

where $n$ is the number of bits in $\tilde{X}$, $\tilde{K}_1$, and $\tilde{K}_2$. This effect is illustrated in Table 1. The 2 configurations of SAS will be introduced in the rest of this section.

5.2 Configuration 1: SAS with One SAS Block

This configuration is illustrated in Fig. 5. In this configuration, there is one SAS block. As the critical minterms evenly partition the set of wrong keys, the corruptibility of each critical minterm is $Y_C = \frac{1}{m}$. Below we derive the security (SAT attack complexity) of this configuration assuming that the SAT solver chooses a DI uniformly at random in each iteration. This is a common assumption [12, 22, 24]. The security is quantified using the expected number of SAT iterations $E[\lambda]$. To start with, we give 2 useful lemmas. The proofs are given in Appendices B and C.

**Lemma 5.1.** Let $D^i$ be the set of DIs that have been chosen in the first $i$ iterations and $X$ be a primary input minterm. If $K^i_X \subset \bigcup_{X \in D^i} K^i_{\tilde{X}}$, then $X$ cannot be the DI of any SAT iteration beyond $i$.

**Lemma 5.2.** For SAS Configuration 1, any critical minterm must exist in the set of DIs when SAT finishes: $X \in D^\lambda \forall X \in M$, where $\lambda$ is the total number of SAT iterations and $D^\lambda$ is the set of all DIs.

**Theorem 5.3.** The expected number of SAT iterations of SAS Configuration 1 is

$$E[\lambda] = \frac{2^n + m}{\frac{n}{2}} \quad (5)$$

**Proof.** The total number of SAT iterations equals the total number of DIs. DIs consist of critical minterms and non-critical minterms. By Lemma 5.2, all the critical minterms must be in the set of DIs for SAT to terminate. Therefore, we only need to find the expected number of non-critical minterms that are chosen as DIs. As illustrated in Table 1, $\forall X^i \notin M, \exists$ exactly one $\tilde{X} \in M$ such that $K^i_{\tilde{X}} \subset K^i_X$. By Lemma 5.1, if this $\tilde{X}$ is chosen as DI before $X^i$, then $X^i$ cannot be chosen in further iterations any more. In other words, $X^i$ will count towards the total number of iterations only when it is chosen before the critical minterm $\tilde{X}$. By our assumption that the DI is chosen uniformly at random in each iteration, $\tilde{X}$ has a probability of $\frac{1}{2}$ to be chosen as DI before $X$ is chosen. As this is true for any non-critical minterm, the expected number of SAT iterations is

$$E[\lambda] = \frac{1}{2}(2^n - m) + m = \frac{2^n + m}{2}.$$  

5.3 Configuration 2: Multiple SAS Blocks

In this configuration, we have $l$ SAS blocks as illustrated in Fig. 6. Each SAS block takes an $n$-bit primary input $\tilde{X}$, which
is shared among all the SAS blocks, and a 2\( n \)-bit key input. The output of each SAS block is XOR’ed with a wire in the original circuit. Therefore, a fault is injected into the original circuit if at least 1 SAS block has output 1. Let \( M^j \) be the set of critical minterms for the \( j \)th SAS block, \( j = 1, 2, \ldots, l \). For ease of implementation, we choose \( l \) also to be a power of 2 and \( l \leq m \).

The output of each SAS block is XOR’ed with a wire in the original circuit. Therefore, by Lemma 5.1, in order to include a critical minterm will be chosen as DI is present in each SAT iteration, the probability that each non-critical minterm is present in the set of DIs when SAT finishes: \( X \in D^\lambda \) \( \forall X \in M \), where \( \lambda \) is the total number of SAT iterations and \( D^\lambda \) is the set of all DIs.

The proof is given in Appendix D. Below, we will analyze the security of this configuration by deriving the expected number of SAT iterations.

**Theorem 5.5.** The expected number of SAT iterations of SAS Configuration 2 with \( l \) SAS blocks and \( m \) critical minterms is

\[
E[\lambda] = \frac{l \cdot 2^n + m}{l + 1}
\]

**Proof.** By Lemma 5.4, every critical minterm must count toward the total number of SAT iterations. Therefore, we only need to derive the expected number of non-critical minterms that are chosen as DIs.

For any non-critical minterm \( X' \not\in M \), in the \( j \)th SAS block, there exists exactly one critical minterm \( X^j \) such that the set of wrong keys that corrupt \( X^j \) in this SAS block, \( \mathcal{K}_{i,X^j} \), is a subset of the set of wrong keys that corrupt \( X' \). As the construction of the SAS block makes this true for any individual SAS block and the critical minterms for each SAS block are mutually exclusive, there are a total of \( l \) such critical minterms. When all of these critical minterms are chosen as DI, they will cover the entire set of wrong keys that corrupt \( X' \). Therefore, by Lemma 5.1, in order to include \( X' \) in the set of DIs, it must be selected before all \( l \) critical minterms are selected. This holds for any non-critical minterm.

By our assumption that the DIs are chosen uniformly at random in each SAT iteration, the probability that each non-critical minterm will be chosen as DI is \( \frac{l}{l+1} \). Therefore, the expected number of SAT iterations is

\[
E[\lambda] = \frac{1}{l+1}(2^n - m) + m = \frac{l}{l+1} \cdot \frac{2^n - m + m}{l+1}.
\]

The properties of both SAS configurations are summarized in Table 2.

**Table 2: Properties of the 2 SAS configurations**

<table>
<thead>
<tr>
<th>Configuration</th>
<th>( l )</th>
<th>( n )</th>
<th>( E[\lambda] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>( m )</td>
<td>( \frac{m}{l} )</td>
</tr>
<tr>
<td>2</td>
<td>1 \leq l \leq m</td>
<td>( \frac{m}{l} )</td>
<td>( \frac{2^n - m + m}{l+1} )</td>
</tr>
</tbody>
</table>

### 6 EXPERIMENTS

This section shows the experimental results of SAS as well as the comparison between SAS and SFLL. Recall that, as illustrated in Fig. 3, we obtain the gate-level netlists of a 32-bit 80386 processor by synthesizing the high-level description using Cadence RTL Compiler. Then we lock the netlist using various SAS configurations and SFLL-flex with the same set of critical minterms. Note that the critical minterms are selected from those that are present in each benchmark. The architecture-level simulation is conducted by a modified GEM5 [2] simulator where error is injected into the locked processor module according to the hardware error profile due to the wrong key. We conduct the following experiment to verify the security and effectiveness of SAS. We also compare SAS with SFLL.

#### 6.1 Security and Effectiveness

We first verify whether the security of SAS against SAT attack (i.e. the actual number of SAT iterations) matches what we have derived in the last section. The security of SAS and SFLL is also compared. We lock the multiplier in the 32-bit 80386 processor with SAS configurations 1 (\( l = 1 \)) and 2 (\( l = 2 \) and \( l = 4 \)) as well as SFLL. We choose \( m = 4 \) for each evaluated locking scheme.

The expected and actual numbers of SAT iterations to break various SAS configurations are given in Fig. 7. The theoretical and experimental results are consistent with each other and grow exponentially in the key length \( n \). In Fig. 8, we compare the actual SAT iterations of SAS and SFLL. In Fig. 8a, it can be observed that SAS’s SAT complexity is higher than that of SFLL by a roughly constant factor when \( m \) is fixed. Note that the same set of four critical minterms (\( m = 4 \)) are used for each locking scheme. In Fig. 8b, we vary the critical minterm count (\( m \)) from 4 to 32 and demonstrate its impact on the security of SAS and SFLL. While SAS configurations become stronger with more critical minterms, SFLL becomes weaker. Therefore, SAS is more secure against SAT attack and gives designers more flexibility when more critical minterms are needed.

We evaluate the effectiveness of SAS and SFLL at the application level using PARSEC [1] benchmarks. In our experiments, various numbers of critical minterms are locked. The same set of critical minterms are used for SAS and SFLL in each experiment. For SAS, we choose \( l = 1 \) when \( m = 1 \) and \( l = 2 \) when \( m \geq 2 \). Fig. 9 shows that both SAS and SFLL are effective at the application level. Considering that SAS’s security is not compromised with the increase in \( m \) as opposed to SFLL (as shown in Fig. 8b), SAS is a significant improvement over SFLL.

#### 6.2 Hardware Overhead and Summary

Now that we have demonstrated the security of SAS against SAT attack and its application-level effectiveness, we evaluate its hardware overhead and compare with existing logic locking methods. The hardware (i.e. chip area) overhead is estimated using the number of gates. The baseline case is a 32-bit multiplier without any logic locking. The details of each compared approach are as follows.
In this work, we investigate logic locking methodologies for securing real-world workloads. We motivate our work by demonstrating the insufficiency of the state-of-the-art logic locking scheme in securing such applications. We point out that this is due to the fundamental trade-off between security (SAT attack complexity) and effectiveness (error rate of wrong keys) of logic locking. We formally prove this trade-off. In order to address this dilemma, we propose Strong Anti-SAT (SAS) where a set of critical minterms are assigned higher corruptibility in order to ensure high application-level impact. Experimental results show that SAS secures processors against SAT attack by ensuring exponential SAT attack complexity and high application-level impact simultaneously given any wrong key. We also evaluate the hardware overhead of SAS and compare it with existing locking schemes where it is shown that SAS has lower hardware overhead.

ACKNOWLEDGMENTS
This work is supported by AFOSR MURI under Grant FA9550-14-1-0351 and Northrop Grumman Corporation and University of Maryland Seedling Grant.

REFERENCES
https://doi.org/10.1109/ISVLSI.2019.00103

A PROOF OF THEOREM 4.4
Recall that
\[ \epsilon = \frac{1}{|K^W|} \sum_{K \in K^W} \epsilon_K = \frac{1}{|K^W|} \sum_{K \in K^W} |X_K| = \frac{1}{2^n |K^W|} \sum_{K \in K^W} |X_K| \]
and
\[ \gamma = \frac{1}{2^n} \sum_{X \in \{0,1\}^n} Y_X = \frac{1}{2^n} \sum_{X \in \{0,1\}^n} |K_X| = \frac{1}{2^n |K^W|} \sum_{X \in \{0,1\}^n} |K_X| \]
Therefore, in order to prove \( \epsilon = \gamma \), we only need to prove
\[ \sum_{K \in K^W} |X_K| = \sum_{X \in \{0,1\}^n} |K_X| \] (8)
Let us consider the following bipartite graph \( G = (X, K^W, E) \) where \( X = \{0,1\}^n \) which is the set of all possible input minterms, \( K^W \) is the set of wrong keys, and \( E = \{(X, X) | X \in X \land K \in K^W, K \text{ corrupts } X\} \). Both sides of Eq. 8 denote the total number of elements in \( E \) and hence must be equal.

B PROOF OF LEMMA 5.1
Proof. Recall that Equation (2) gives the SAT formula for each SAT iteration:
\[ (X_1, K_\alpha, Y_\alpha) \land (X_1, K_\beta, Y_\beta) \land (Y_\alpha \neq Y_\beta) \]
\[ \land \bigwedge_{i=1}^{n-1} (X_i, \tilde{K}_\alpha, \tilde{Y}_i) \land (X_i, \tilde{K}_\beta, \tilde{Y}_i) \]
To satisfy the first line, at least one of \( \tilde{K}_\alpha \) and \( \tilde{K}_\beta \) must be a wrong key that corrupts \( X \). However, since any wrong key that corrupts \( X \) also corrupts at least 1 previously found DI, this wrong key cannot satisfy the second line. Hence such \( \tilde{X} \) cannot be the DI in future iterations.

C PROOF OF LEMMA 5.2
Proof. Recall that \( g \) has on-set size 1. Let \( \tilde{P} \) be the input that makes \( g(\tilde{P}) = 1 \). \( \forall X \in M \), let \( \tilde{K}_1 = X \oplus \tilde{P} \). Then, any \( \tilde{K} = (\tilde{K}_1, \tilde{K}_2) \in K^W \) is a wrong key that only corrupts \( X \). Therefore, \( \tilde{X} \) has to be chosen as a DI to prune out this wrong key.

D PROOF OF LEMMA 5.4
Proof. This is a natural extension of Lemma 5.2. Let \( \tilde{X} \) be a critical minterm and \( \tilde{X} \in M^r \). Recall that \( g \) has on-set size 1. Let \( \tilde{P} \) be the input that makes \( g(\tilde{P}) = 1 \). \( \forall X \in M^r \), let \( \tilde{K} = X \oplus \tilde{P} \). Then, let us consider the following wrong key \( \tilde{K} = (\tilde{K}_1, \tilde{K}_2, \ldots, \tilde{K}_i) \in K^W \) which is composed as follows: \( \tilde{K}_i = (k, \tilde{K}_i^C) \in K_i^C \) where \( K_i^C \) is the set of wrong keys for the \( i \)th SAS block. For any \( i = 1, 2, \ldots, l \) that \( i \neq j \), \( \tilde{K}_i \in K^C \) where \( K_i^C \) is the set of correct keys for the \( i \)th SAS block. Such a key \( \tilde{K} \) is a wrong key that only corrupts \( \tilde{X} \). Therefore, \( \tilde{X} \) has to be chosen as a DI to prune out this wrong key.