StatSAT: A Boolean Satisfiability based Attack on Logic-Locked Probabilistic Circuits

Ankit Mondal, Michael Zuzak and Ankur Srivastava
Department of Electrical and Computer Engineering
University of Maryland College Park, Maryland 20742, USA
Email: amondal2@terpmail.umd.edu

***Abstract*—The outsourcing of chip designs for fabrication has raised concerns regarding the protection of Intellectual Property (IP) from an untrustworthy foundry. Logic locking is a design-for-security technique that has the potential to thwart attacks from such an adversary. On the other hand, the notions of approximate and probabilistic computing have been popularized due to their low energy consumption characteristics and their potential application in error-tolerant frameworks. Prior work has looked into and exposed the vulnerability of logic-locked circuits using concepts of Boolean Satisfiability (SAT), but mostly from the perspective of deterministic designs. Despite existing attack frameworks not being directly applicable, we show in this work that circuits exhibiting probabilistic behavior also face the same threat. We propose *StatSAT*, an attack methodology incorporating statistical techniques into the existing SAT attack, that can overcome the hurdles imposed by the probabilistic behavior. Our attack results show that the adversary is capable of unlocking the circuit to an extent good for all practical purposes.**

I. Introduction

The security of hardware designs has gained attention with the possibility of adversaries in the IC supply chain [1]. With the outsourcing of chip fabrication to offshore foundries becoming common, IC designers are increasingly interested in protecting the IP of their design from attackers in the fabrication unit. Manufactured circuits released into the market may be subject to reverse engineering (RE) with the intent of stealing sensitive information, and a foundry-based adversary may overproduce the design and sell illegal copies in the black market [2].

Logic locking (aka logic obfuscation) is a technique that aims to obfuscate the functionality of a design by adding additional logic gates and inputs (called keys) to its circuit [3, 4]. The circuit behaves correctly only when the correct key input is provided to it, otherwise it produces wrong outputs for certain primary inputs. Such a protection mechanism is expected to hide the original functionality of the design from an untrusted foundry and also prevent it from illegally overproducing the IC. The key input bits are stored in a tamper-proof memory after fabrication which is inaccessible to an attacker [5].

Early on, several works [6, 7] proposed different methods of performing logic locking to increase resilience against fault-sensitization and justification based attacks using automatic test pattern generation tools. More recently, attacks [8, 9] that leverage boolean satisfiability (SAT) to find the correct key have prompted researchers to investigate further into this field, leading to a flurry of defense techniques.

Relaxing the stringent requirements of functional correctness of circuits reduces the cost of manufacturing, verification and test [10]. Additionally, the energy benefits of such trade-offs have brought the concept of approximate and probabilistic/imprecise computing into the limelight. The rising popularity of big-data, Internet-of-Things and machine learning-based applications have widened the scope of such approximate computations owing to their error-tolerant nature [11].

The uncertainties of IC design at highly scaled-down technology nodes often lead to probabilistic behavior. This is due to a variety of factors, including reduced noise margin because of voltage scaling, which is an effective way to reduce the energy consumption of circuits, and larger process variations [10]. The errors arising from the use of probabilistic computing elements are dynamic and transient in nature [12, 13]. They have a certain probability of occurring and can occur anywhere in the circuit. Such non-deterministic deviation of a system's behavior from its specifications falls under probabilistic design/computing. Also, several non-CMOS emerging devices exhibit stochastic properties that can be exploited for realizing probabilistic systems [14].

In this work, we demonstrate how it is possible for an untrusted foundry to steal the IP of probabilistic chips potentially employed in systems wherein approximate/probabilistic computing is a sought-after framework. We develop an attack strategy based on Boolean Satisfiability (SAT) to find the key when such a circuit is logic-locked. The existing SAT attack [8, 9] is good for deterministic circuits but not for probabilistic ones, since the latter exhibits inconsistent behavior which can be detrimental to the progress of the attack. The work in [15] proposes a Probabilistic SAT (PSAT) attack to handle the situation. However its success is limited to low error rates and its scalability with the no. of circuit outputs tends to be poor. In this paper, we propose **StatSAT**, an attack that is applicable to circuits having higher error than what was considered by PSAT. Our contributions can be summarized as follows:

- We explain why and how our attack should differ from the Standard SAT attack when building the SAT solver's problem formulation.
- We propose statistical schemes for developing the SAT formulation to deal with the probabilistic behavior of the circuit. This includes a technique to model the error within the circuit during the progress of the attack.
- When the above techniques aren't sufficient to unlock the chip, we show how multiple SAT formulations can be used simultaneously to ensure the progress of the attack.
- We present attack results on standard benchmark circuits and demonstrate that it is possible for an attacker to recover at least a good enough key for unlocking the circuit in a feasible amount of time. We also make comparisons with PSAT and show that StatSAT produces better attack results.

Overall, this paper exposes the vulnerability of fault-tolerant chips to IP theft; their probabilistic behavior isn't much of a hurdle for an adversary.

II. Background and Related Work

A. Logic Locking

Logic locking is a design-for-security technique for protecting IP from adversaries based in an untrusted foundry, and not only from a malicious end-user [3]. It involves adding extra gates and inputs (keys) to the original netlist of a circuit to make it secure [6, 7]. This locked netlist would provide the desired functionality only if the correct key value, which is known to the IC designer, is loaded into the circuit. The untrusted foundry, although in possession of the locked netlist, does not possess the secret key to unlock the circuit. It thus cannot decipher the IP of the circuit and overproduce the chip (since the original functionality would be restored only with the correct key).

B. The SAT attack

The field of logic locking received a new lease of life when the concept of Boolean Satisfiability (SAT) was used to find the correct key [8, 9, 16]. **Threat model:** The SAT-based attack considers an attacker based in the foundry and requires them to have access to only

- The netlist of the locked circuit obtained through RE of the layout
- An activated/unlocked version of the chip (called the 'oracle') purchased from the open market which can be used as a black box.

The SAT attack essentially involves iteratively pruning the wrong keys with the help of the oracle. It finds in each iteration, using a SAT solver, an input pattern that can eliminate one or more wrong keys. The attack terminates when only the correct key K^* (and other equivalent keys[1], if any) remains in the search space.

Let us adopt the following notations for describing the attack:

- $C_L(X, K)$ - The response of the locked circuit with primary and key inputs X and K respectively.
- $C_O(X)$ - The correct output for input X obtained from the oracle.

The attack methodology is described next (illustrated in fig. 1).

- In the first iteration, a SAT solver is used to obtain an input X^1 which can produce different outputs (say Y_a^1 and Y_b^1) with 2 different keys (say K_a^1 and K_b^1 respectively). Since this input X^1 has the capability to distinguish between 2 keys, it is called a *distinguishing* input (DI).
- The same input X^1 is fed to the oracle to obtain its correct output Y^1, i.e. $C_O(X^1) = Y^1$. Then, (X^1, Y^1), which is known as a distinguishing input-output pair (DIP), is stored in the Conjunctive Normal Form (CNF) SAT formula of the locked circuit as a constraint to be satisfied in the future. The rationale behind obtaining the DI is that since it produces 2 different outputs (at most one of which can be correct) with 2 keys, it has the potential to eliminate at least one of those 2 keys and also all other keys which do not satisfy the corresponding I/O pair (X^1, Y^1).
- The m^{th} iteration of the attack produces a DI, say X^m, which again gives different outputs with 2 keys K_a^m and K_b^m. However, these keys have to be such that they satisfy all the previously obtained I/O pairs. That means $C_L(X^i, K_a^m) = Y^i = C_L(X^i, K_b^m)\ \forall\ i = 1 \ldots (m-1)$. Fig. 1 depicts the m^{th} iteration of the attack wherein DI X^m and distinguishing keys K_a^m and K_b^m have been found.
- This continues until the SAT solver is no longer able to find a DI, which means that all wrong keys have been eliminated with the previously found I/O pairs. At this point, the solver is asked to provide one key which satisfies all those DIPs, which is the correct key K^* for that locked netlist. This key satisfies $C_L(X, K^*) = C_O(X)\ \forall\ X$ in the input space.

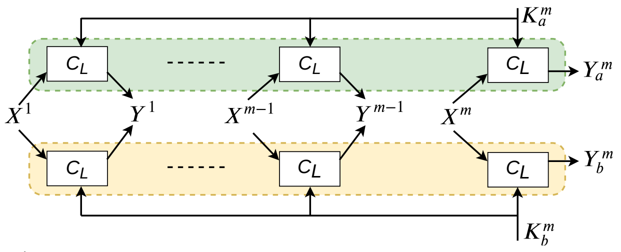

Fig. 1: The SAT attack. The green and yellow rectangles denote the CNF SAT formulas for 2 keys producing different outputs. By the m^{th} iteration, both of them have been fed with the constraint of satisfying the first $(m-1)$ I/O pairs. The m^{th} iteration produces (i) 2 keys which satisfy those pairs, and (ii) the DI X^m.

To counter the powerful SAT attack, several SAT-resilient techniques [17, 18] came up subsequently. They usually aimed for increasing the size of the distinguishing I/O set exponentially, thereby increasing the time complexity of the SAT attack. However, these defense mechanisms have also been subject to specialized attacks that seek to exploit their characteristics. Examples of such attacks include recovery of an approximate key [19], removal attack [20] and bypass attack [21]. Another way of making the SAT attack complex is to increase the time required per iteration by using SAT-hard instances, as was done in [22]. The cat and mouse game thus continues in this field, with the state-of-the-art locking techniques being Stripped Functionality Logic Locking (SFLL) [23, 24] and Full-Lock [22]. Several structural attacks [20, 25, 26, 27] have also been proposed that do not assume oracle access.

[1]Two keys are said to be equivalent if they induce the same overall logic function on the locked circuit

C. Probabilistic Computing

Approximate computing is a popular computational paradigm wherein the accuracy of computations is traded-off for lower energy consumption. One common example is the use of simplified logic for obtaining the less significant bits (LSBs) of a number. Probabilistic computing is a related notion, and can be considered as a subset of approximate computing. It relies on noisy components which always have a small probability of producing a faulty output [28]. Therefore, the entire computation is probabilistic rather than deterministic. However, the components are uncorrelated and thus the overall error rate of the system is usually within a tolerable limit. Such probabilistic designs have relevance in modern and future technologies [10]. With scaled voltages at smaller technology nodes, noise levels tend to be significant when compared to the energy difference between logic states [12]. Additionally, the impact of soft errors also becomes more prominent at smaller device dimensions [29].

Non-CMOS devices for Probabilistic Computing: With the integration of emerging non-CMOS devices into CMOS circuits, non-determinism in systems is usually inevitable due to their inherent randomness, and is often beneficial to and harnessed in the concerned application. An example is the work in [14] where a Giant Spin Hall Effect (GSHE) switch is used as a polymorphic gate that can be configured to realize any of the 16 functions of 2 boolean variables. The work in [14] was extended in [15] by moving away from deterministic computations to probabilistic ones by exploiting the inherent stochasticity of the GSHE switch. Such a stochastic/probabilistic gate would behave erroneously at certain times, in which case the output of the circuit built with such gates would not always be the same for a given input.

III. Attack Surface

In this work, we consider a circuit which is probabilistically approximate, as explained in sec. II-C. This means that the circuit not only produces approximate outputs but also behaves probabilistically - the same input, when applied repeatedly, does not elicit the same response. The reason behind such probabilistic behavior could either be noise from any source or the use of non-CMOS components behaving probabilistically. Operating circuits under scaled voltage conditions saves energy, and so does the use of non-CMOS devices. The resulting erroneous behavior at the outputs of the circuit could be acceptable for LSBs and irrelevant outputs.

We assume the threat model described in sec. II-B. In this work, we consider widely accepted locking techniques [6, 23] applied in a static/deterministic manner to a probabilistic circuit. We consider a circuit error model where each logic gate in the circuit has a certain error probability ϵ, which is the probability that the gate outputs the inverse of what it should have, given its inputs. Such an error model is realistic from the perspective of non-CMOS gates [15] and also dynamic errors [13]. Ideally, the attacker wishes to find the correct key corresponding to the locked circuit in the face of such uncertain behavior of the activated chip. We emphasize that the correct key is the one which will make the locked circuit produce the exact same input-output behavior as that of the oracle. Even if the attacker is unable to find the correct key, (s)he wants to find a key which can match the statistical behavior of the oracle as closely as possible.

Existing work: The work in [15], which proposed the probabilistic use of GSHE gates, also aims to find the correct key in a similar scenario. The authors first show that the probabilistic behavior of the gates in the oracle misguides the attacker (and hence a SAT solver) as (s)he may be tricked into using a wrong output for a distinguishing input. This leads to failure of the standard SAT attack (which was described in sec. II-B) even at barely significant error levels[2]. They then propose a modified version of the SAT attack, calling it Probabilistic SAT (PSAT), wherein the oracle is queried multiple times with a certain DI instead of just once. This is done to circumvent the inconsistency of the oracle's output. Then, if the most frequently occurring output pattern is dominant (see [15] for

[2]The approximate SAT attack AppSAT proposed in [19] also cannot be applied in our threat scenario because it too would require a deterministic oracle.

meaning of dominant), it is considered as the correct output[3], else one of the output patterns is sampled with a probability equal to its frequency of occurrence, and is chosen as the correct output. The PSAT produces better attack results than the conventional SAT.

In this work, we propose to improve upon the PSAT attack since the latter has the following limitations.

- The PSAT attack treats output patterns from the oracle as a whole, and always considers only a single correct pattern for any DI in the SAT-CNF formula. However, a probabilistic circuit (specifically, the activated chip with the attacker) may not produce the correct output with the highest frequency. For eg, if the output patterns 0110, 0010 and 0001 are produced by the oracle (for a given DI) 11, 7 and 2 times respectively, then 0110 will be considered as the correct output by PSAT since it is dominant, even if the correct output is 0010.
- Moreover, for a circuit with a large no. of outputs, the probability of appearance of the correct output may be exponentially small. As a result, PSAT fails to return any key when the no. of circuit outputs or the error levels in the circuit start to go up.

We devise an attack method **StatSAT**, and show that it not only has a higher success rate than PSAT (an attack is successful if it returns a key) but can also yield *the* correct key with high probability. In any case, StatSAT is capable of producing a key with which the statistical behavior of the now-unlocked circuit is very similar to that of the oracle.

IV. Attack formulation

The crux of the StatSAT attack is the same as that of the standard SAT attack described in sec. II-B - the SAT solver keeps finding DIs until it narrows down to the correct key. In this section, we describe the additional steps employed by our attack to overcome the obstacles presented by the probabilistic behavior of the circuit.

A. Deviation from the SAT attack

The standard SAT attack [8, 9] considers a deterministic oracle and queries it once in every iteration with a DI to obtain the corresponding correct output. In our work, we have an oracle which behaves probabilistically. Because its output is probabilistically approximate, and therefore inconsistent, the output obtained upon applying an input just once cannot be considered to be the correct output pattern for that input.

At this moment, let us define the *Bit Error Ratio* (BER) at an output of a circuit, for a given input, as the frequency/probability with which that output bit is erroneous for that input. In the context of our attack, erroneous means that the output of the probabilistic oracle differs from a deterministic version of it. For eg. consider that for an input X, a deterministic version of the oracle produces a '1' at the i^{th} output. Whereas our probabilistic oracle, when queried a large no. of times, produces '1' for 70% of the times and '0' for the rest 30%. Then the BER at that i^{th} output for input X is 0.3.

To get around such inconsistent behavior of the oracle, we apply the same input (DI) multiple times to it to get multiple output patterns, as was done in [15]. We, however, do not try to consider the oracle's output patterns as a whole, and instead take an average of all patterns for each of the circuit's outputs. Thus, for any DI, the oracle is queried not once but multiple times to obtain the *signal probabilities* of each of the output bits.

Stated mathematically, let N be the no. of outputs in the circuit. If the probabilistic oracle is sampled N_s times with an input X, let $Y^{(1)}$, $Y^{(2)}$ $\ldots Y^{(N_s)}$ be the observed binary output patterns, where each $Y^{(j)} \in \{0,1\}^N$. Then the signal probability at the i^{th} output of the oracle, with input X, is given as

$$P_i^Y = \langle Y_i \rangle = \frac{1}{N_s}\sum_{j=1}^{N_s}(Y^{(j)})_i \tag{1}$$

with every $P_i^Y \in [0,1]$ and $i = 1, 2, \ldots N$.

[3]Here (and throughout this paper) correct output pattern means what the oracle would have output if it was not a probabilistic circuit.

At this point, we argue that simply rounding all these signal probability values (to 0 or 1, whichever is closer) may not yield the correct output pattern corresponding to input X. One or more output bits may have higher than 50% BER in P^Y for a certain input depending on how gates within the circuit were sensitized [30].

Let us now discuss how the signal probabilities P_i^Y obtained from the oracle are provided to the SAT solver. The standard SAT attack uses responses from the oracle to constrain the satisfiability of the SAT-CNF expressions (sec. II-B). For any I/O pair (X, Y) fed to the solver, future iterations ensure that *all* bits of Y are satisfied (with input X) when DIs are obtained. However, in this work, we relax this constraint by not having to specify all the output bits. The prime purpose of this is to avoid having erroneous bits in the output patterns specified since these would either lead us to a wrong key or make the SAT formula unsatisfiable. How this is achieved is explained next.

B. Output Uncertainty

The signal probabilities P^Y of the oracle response tells us about the certainty of an output bit. Let us define the *uncertainty* U associated with the signal probabilities as

$$U_i = min(P_i^Y, 1 - P_i^Y) \qquad i = 1 \ldots N \tag{2}$$

Quite naturally, P_i^Y values close to 0.5 are associated with higher uncertainties in the output bit. Because a boolean SAT solver would accept only 0 or 1 as values of variables, the P_i^Y would have to be converted to a likely binary 0 or 1. But, we leave outputs with high uncertainties **unspecified** in the constraint because these evidently have a high BER. Thus, for any DI, the oracle output signal probability vector P^Y could be translated to a binary output vector Y as

$$Y_i = \begin{cases} round(P_i^Y) & \text{if } U_i \leq U_\lambda \\ \text{x (unspecified)} & \text{if } U_i > U_\lambda \end{cases} \tag{3}$$

where U_λ is a threshold value of uncertainty above which we refrain from specifying the rounded signal probability value as the (supposedly correct) binary output value.

Effect on search space of key: A DI and its corresponding output impose constraints on the possible correct key. In the standard SAT attack, each iteration trims down this solution space by eliminating a certain number (at least one) of wrong keys. In our StatSAT approach, not specifying one or more of the output bits reduces the no. of wrong keys eliminated in an iteration, with the possibility of no key being eliminated at all[4]. Thus, the solution space is not pruned as much as in the standard SAT attack in order to remain on the safe side and avoid eliminating the correct key from consideration.

C. Estimation of Output BERs with DIs

In the previous subsection, we proposed using the uncertainty values U as indicators of high output BERs to avoid providing wrong DIPs to the SAT solver. While this method is effective when the BER at a particular output is low or moderate (close to 0.5), it would not prevent from recording wrong outputs (Y) if the BER happens to be large. Specifically, if the BER at the i^{th} output is larger than $1 - U_\lambda$, then $U_i < U_\lambda$ and Y_i is set to $round(P_i^Y)$ (which would be wrong). For eg. if $P_i^Y = 0.3$, with BER at the i^{th} output being 0.7 (and not 0.3), and $U_\lambda = 0.35$, then $Y_i = round(P_i^Y) = 0$ will be wrongly passed off as the correct value. We shall now discuss how to prevent this kind of a situation by predicting the possibility of having a large BER at any output with a given DI.

Let us assume for now that the attacker is aware of the error probability ϵ_g at each logic gate in the circuit[5]. With this knowledge, the attacker can *roughly* estimate the BER at each of the outputs of the circuit with a given input using the concept of Boolean Difference Calculus [30]. It is not possible to estimate the BERs *exactly* without knowledge of the correct key.

Each iteration of the SAT attack prunes the keyspace such that the keys remaining in the solution space at the start of an iteration satisfy all of the

[4]We discuss later how to ensure that we make progress despite such a possibility.
[5]We'll relax this assumption in sec. V and show that it doesn't matter much

distinguishing I/O pairs of the previous iteration. These satisfying keys are likely to provide a better estimate of output BERs with the current DI than any key picked randomly from the entire keyspace. Therefore, we use the SAT solver to find a certain number of satisfying keys (or as many as remaining) and obtain the BERs separately for each of those keys. These BERs from satisfying keys are then averaged to get the estimated BER $E \in [0,1]^N$ for that DI.

Stated formally, let $(X^1, Y^1) \dots (X^{m-1}, Y^{m-1})$ be the distinguishing I/O pairs of the first $m-1$ iterations, and X^m be the m^{th} DI. We find a certain number of satisfying keys of these $m-1$ pairs - say N_{satis} (or less if that many aren't remaining in the solution space). Using those keys, the output BER E^m of the locked circuit with X^m is roughly estimated (using Boolean Difference Calculus [30]). This is used to flag the output bits as potentially having high error rates, and to decide whether or not to pass off the rounded signal probabilities to the CNF formula. Similar to earlier (eqn. 3), the translated binary output vector for DI X^m would finally be given as

$$Y_i^m = \begin{cases} round(P_i^Y) & \text{if } U_i \le U_\lambda \text{ and } E_i^m \le E_\lambda \\ \text{x (unspecified)} & \text{otherwise} \end{cases} \qquad (4)$$

where, if the estimated BER E_i^m for the i^{th} output crosses threshold E_λ, we do not specify its value irrespective of the corresponding signal probability P_i^Y. For eg. if the signal probability values for a 4-bit output are $P^Y = (0.85, 0.38, 0.20, 0.77)$ and BER estimate $E = (0.21, 0.28, 0.27, 0.34)$, then uncertainty $U = (0.15, 0.38, 0.2, 0.23)$. If $U_\lambda = 0.25, E_\lambda = 0.30$, then $Y =$ 1x0x is the output vector, with the 2^{nd} and 4^{th} bits unspecified, that would be provided to the SAT solver.

D. Multiple Instances of SAT CNF Formulas

Until now we focused on how we can prevent the CNF formula from recording wrong output bits for distinguishing input patterns, thereby holding back information that the SAT solver needs to trim down the keyspace. Doing so certainly keeps the correct key within the solution space. However, after a certain point of time (i.e. a certain number of SAT iterations), the SAT solver may no longer able to eliminate keys because of the unspecified bits in the output vectors of the DIs. This is evident from the **repetition of a DI** in 2 consecutive iterations. Thus if the DI in the $(m+1)^{th}$ iteration is same as that in the m^{th}, it implies that the unspecified output bits in Y^m likely hold the key to further elimination of "wrong keys". And so querying the oracle again with the same DI is unlikely to help since it would yield the same signal probability vector P^Y as in the m^{th} iteration, and hence the same binary output vector Y^m. Next, we describe how to get around this dead-end.

Duplication: Let us call the set of CNF formulas and the distinguishing I/O pairs collectively an *instance* of the SAT formulation. To proceed with the attack, we propose to create, at this stage, a duplicate (clone) of the SAT instance to be able to consider both possibilities of an unspecified bit. Thus, if the DI X^{m+1} is a repeat, we let one of the previously unspecified bits of the binary output vector Y^m be represented differently in Y^{m+1} by these 2 SAT instances - one considers a value of '0' for that bit, while the other considers a '1'. They differ only in that bit position of Y^{m+1} which we are now forced to specify, and are identical in all other respects (both hold the same

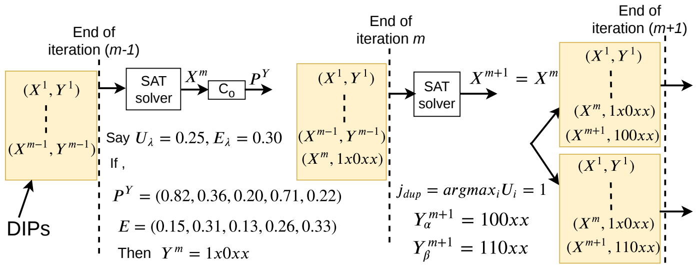

Fig. 2: Demo of duplication of SAT instances. The yellow rectangles represent the DIPs stored by the end of a certain iteration. Because $X^{m+1} = X^m$, the instance forks into 2 with Y^{m+1} being different only at bit position $j_{dup} = 1$ (as per eqn. 5).

I/O pairs of previous iterations, and also same values for the previously specified bits of Y^{m+1}). Quite obviously, since a bit position can have a correct value of either 0 or 1, it is guaranteed that one of these 2 instances continues to hold correct I/O pairs (if I/O pairs accumulated hitherto are all correct).

The choice of the specific unspecified output bit to specify during such a SAT duplication approach is made using the uncertainty U and estimated BER E. Among all such bits, we specify the one that has the largest value of uncertainty $U_i > U_\lambda$ since that output wire is likely to have a high value of error in general. If none of the U_i values are larger than U_λ, we choose the one which has the largest estimated BER E_i for the same reason as above. The original and duplicated instances thus differ in their Y^{m+1} at bit index j_{dup} given as

$$j_{dup} = \begin{cases} j = argmax_i\ (U_i) & \text{if } U_j > U_\lambda \\ argmax_i\ E_i & \text{otherwise} \end{cases} \qquad (5)$$

After the original SAT instance has duplicated into 2 instances, they continue with the attack finding their own DIs for the next iteration (which are most likely to be different[6]). These instances do not interact with each other in any way and do not exchange any information. Fig. 2 gives a demonstration of the aforementioned process of duplication.

Force Proceed: The 2 instances thus formed go on finding DIs, and whenever any of them gets stuck because of a repeated DI, it further duplicates itself as per the rule mentioned previously. We limit the total number of instances to a certain value, say N_{inst}, to avoid an explosion in the number of instances (that might happen if the error levels in the circuit are high) and thus to keep the attack's time complexity within a limit. Once the number of instances has reached the maximum allowed, it is not possible for any of them to duplicate itself upon encountering a repeated DI. In such a case, we forcefully proceed by specifying carefully just one of the unspecified bits of Y^m in the hope that it provides some/enough information to the SAT solver to trim down a few wrong keys and not repeat the DI again. Towards this, we specify that bit which had the least estimated BER (hence least risky) by rounding the observed signal probability. Thus if $X^{m+1} = X^m$, and the instance is not allowed to duplicate itself, then let

$$j_{fp} = argmin_i(E_i\ \mathbb{I}(Y_i^m = \text{x})) \qquad (6)$$

where indicator function $\mathbb{I}(f) = 1$ if f is *true*, and ∞ otherwise. Then, $Y_{j_{fp}}^{m+1} = round(P_{j_{fp}}^Y)$. For eg. if in fig. 2, there was no scope for duplication, $j_{fp} = 3$ since the 4^{th} bit has the least value (0.26) of E_i among unspecified bits. And so $Y^{m+1} =$1x01x.

Evaluation: Each of instances carries out the process of eliminating wrong keys on its own until it can't find any more DIs. It then returns one key which satisfies its set of recorded I/O pairs or states that the set cannot be satisfied by any key at all (UNSAT). When an instance becomes UNSAT, the space occupied by it is "freed-up" to allow other instances to duplicate themselves if necessary. After all the satisfiable instances return a key (there can be N_{inst} keys at most), these keys are then evaluated by comparing the response of the oracle with that of the locked circuit fitted with the obtained keys. This evaluation is sort of a check for correctness by looking for discrepancies in their responses.

Let $X_1 \dots X_j \dots X_{N_{eval}}$ be randomly chosen N_{eval} input vectors, and let P^{Y_j} and $P^{Y_j}(K)$ denote the output signal probabilities with input X_j of the oracle and of the unlocked circuit (with key K) respectively. Let $FM(K)$ denote the figure of merit of an obtained key K given as

$$FM(K) = \frac{1}{N} \sum_{i=1}^{N} \max_j |P^{Y_j} - P^{Y_j}(K)| \qquad (j = 1 \dots N_{eval}) \qquad (7)$$

The above equation takes the difference between P^Y and $P^Y(K)$ for N_{eval} input patterns, chooses the maximum difference for each of the N output indices and computes the average of these N maximums. The key with the *smallest* value of FM is chosen as the best key. The steps of the StatSAT attack are illustrated in fig. 3 for one SAT instance.

[6]It may be worth mentioning that the distinguishing input is one component of the attack that we do not control/modify in any way. It is always provided by the SAT solver in exactly the same way as is done in the standard SAT attack.

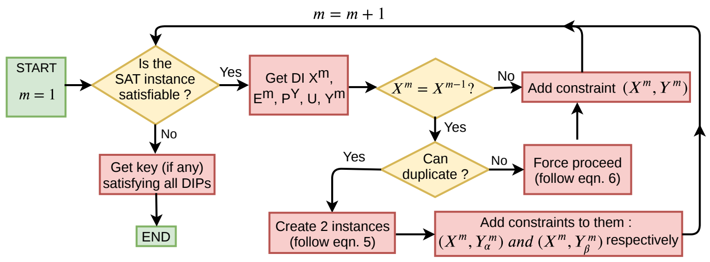
$m = m + 1$
START
$m = 1$
Is the SAT instance satisfiable ?
Yes
No
Get DI X^m, E^m, P^Y, U, Y^m
$X^m = X^{m-1}$?
No
Yes
Add constraint (X^m, Y^m)
Can duplicate ?
Yes
No
Force proceed (follow eqn. 6)
Get key (if any) satisfying all DIPs
END
Create 2 instances (follow eqn. 5)
Add constraints to them : (X^m, Y^m_α) and (X^m, Y^m_β) respectively

Fig. 3: StatSAT attack process for one SAT instance shown in a nutshell.

V. Attack Results

For running our attack, we rely on a C++-based SAT-attack framework (with Lingeling SAT solver) developed by [8, 15] and made open-source. This framework was modified to implement our StatSAT attack methodology[7]. Several benchmark circuits were used for evaluation from different sources as listed in Table I. Locking techniques SFLL-HD [23] and Strong Logic Locking (SLL) [6] were used for evaluationMost of the results are demonstrated on the assumption that the attacker happens to be aware of the exact value of gate error probability ϵ_g. Later when we relax this assumption and let the attacker estimate this value, we see that the differences aren't significant on an average.

Benchmark	Source	Inputs	Gates	Outputs
c3540	ISCAS85	50	1669	22
c7552	ISCAS85	207	3512	108
ex1010	MCNC	10	5066	10
seq	MCNC	41	3519	35
b14	ITC99	277	9767	299
b15	ITC99	485	8367	519

TABLE I: Benchmark circuits and their source

In our attack, N_{inst}, U_λ and E_λ are parameters that the attacker chooses before starting the attack. In addition to the figure of merit $FM(K)$ calculated in eqn. 7, we also obtain, for the best key K^*, a slightly different value that takes the average of the signal probability differences instead of the maximum. This is simply an average hamming distance in the signal probability domain and is given as

$$HD(K) = \frac{1}{N_{eval}} \sum_{j=1}^{N_{eval}} \frac{1}{N} ||P^{Y_j} - P^{Y_j}(K)||_1 \quad (j = 1 \ldots N_{eval}) \qquad (8)$$

where $||.||_1$ is the L1-norm. $HD(K)$ is a better statistical measure than $FM(K)$ for measuring the closeness of the behaviors of the oracle and the unlocked circuit since it averages over all N_{eval} patterns (the latter takes the maximum and is more suitable for measuring discrepancies while comparing amongst several keys).

We now present the results of our attacks for the circuits in table I from various perspectives.

(A) First we show how the no. of instances N_{inst} required to find *the* correct key increases with the gate-level error ϵ_g. Table II mentions the attack results for these circuits with their locking technique, for several values of ϵ_g, in terms of

- The average (over all outputs) and maximum BER of the oracle for each ϵ_g, calculated using 100 random input vectors.
- the (maximum) no. of instances N_{inst} and the no. of keys found ($|K|$)
- the quantity $HD(K^*)$, where K^* is the best key.

The gate-level error probability ϵ_g (labeled A, B, … for each circuit for future reference) was varied and N_{inst} was incremented (starting from 1) in powers of 2 until the correct key was found. For all benchmarks, the smallest value of ϵ_g mentioned in table II is such that values slightly larger than that required more than 1 instance. For eg. with the *c7552* circuit, error levels $\epsilon_g \leq 2.0\%$ did not require more than 1 instance; but a slightly higher value of $\epsilon_g = 2.25\%$ required $N_{inst} = 8$. Also, for each ϵ_g, we start the attack with parameters $U_\lambda = 0.25$, $E_\lambda = 0.30$. If the attack doesn't find a single key, we restart with lower values of one/both. From table II, we infer that it is possible to get the correct key even for high(er) values of ϵ_g by simply increasing the maximum number of SAT instances allowed. Note that ϵ_g values considered for most circuits yield avg. BERs that are significantly high.

[7]The code for StatSAT can be found on Github at https://github.com/ankit-mondal/StatSAT-attack

Bench+Lock	ϵ_g (in %)	Avg. BER	Max. BER	N_{inst}	$\|K\|$	$HD(K^*)$
c3540 SFLL-HD	1.25 (A)	0.241	0.834	1	1	0.0192
	1.50 (B)	0.269	0.852	8	7	0.0200
	1.75 (C)	0.286	0.865	16	16	0.0207
	2.00 (D)	0.302	0.850	64	64	0.0209
c7552 SFLL-HD	2.00 (A)	0.173	0.784	1	1	0.0122
	2.25 (B)	0.182	0.784	8	1	0.0125
	2.50 (C)	0.189	0.760	16	5	0.0125
	3.00 (D)	0.201	0.758	16	16	0.0128
seq SFLL-HD	6.0 (A)	0.263	0.628	1	1	0.0195
	7.0 (B)	0.277	0.610	4	4	0.0201
	8.0 (C)	0.300	0.628	4	4	0.0205
	9.0 (D)	0.313	0.630	32	31	0.0209
b14 SFLL-HD	0.50 (A)	0.0520	0.628	1	1	0.0087
	0.75 (B)	0.0668	0.724	4	1	0.0100
	0.80 (C)	0.0750	0.760	4	3	0.0103
	0.85 (D)	0.0759	0.776	16	16	0.0112
ex1010 SLL (253-bit key)	0.4 (A)	0.117	0.390	1	1	0.0158
	0.5 (B)	0.141	0.466	2	2	0.0171
	0.6 (C)	0.167	0.510	4	4	0.0187
b15 SFLL-HD	0.2 (A)	0.0200	0.378	1	1	0.00583
	0.4 (B)	0.0311	0.546	4	4	0.00796
	0.5 (C)	0.0489	0.580	8	1	0.00874
	0.6 (D)	0.0498	0.608	16	10	0.00939

TABLE II: Variation of N_{inst} required to find the correct key with error probability ϵ_g. It is only due to sampling error that $HD(K^*)$ is non-zero and increasing slightly with ϵ_g. With SFLL-HD locking, 16-bit keys were used. Attack parameters used were: $N_s = 500$, $N_{satis} = 100$, $N_{eval} = 2000$.

(B) Shown in fig. 4 is the number of iterations required by the SAT instance which found the correct key for the respective values of ϵ_g in table II. Accompanying those is the no. of iterations the standard SAT attack would take on a deterministic version of the circuit **only for the sake of comparison** (to show how much extra StatSAT needed). Note that standard SAT was not and should not be launched on the probabilistic circuits. Typically, iterations required by our attack is larger than that of the standard SAT, and tends to go up with ϵ_g because higher output BERs result in more unspecified bits in DIPs.

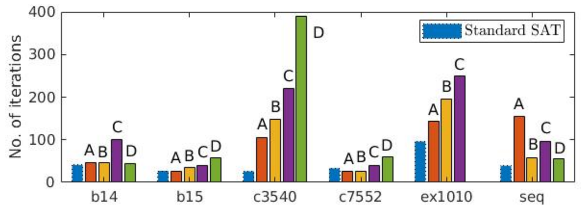

Fig. 4: No. of iterations of StatSAT and comparison with Standard SAT. Labels A, B, … at the top of each bar correspond to ϵ_g values in table II.

(C) Of more interest is perhaps the time taken to carry out the attack. Fig. 5 shows (i) T_{attack}, the time taken just for finding the keys (that is, not counting the time required for the evaluation step) for the same ϵ_g and (ii) T_{eval}, the time required for evaluation of *each* key found which depends on circuit size. T_{attack} of StatSAT is higher than that of the standard SAT attack because of all the extra steps that our attack requires and also of the presence of multiple instances.

(D) Next up, fig 6 depicts the trade-off between the total time spent

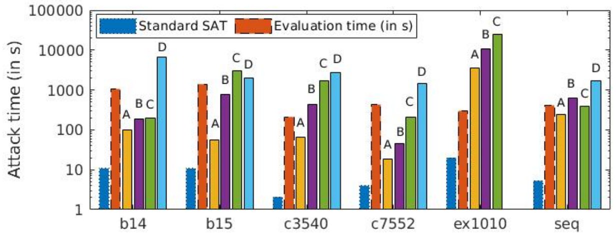

Fig. 5: T_{attack} for several ϵ_g and T_{eval} of each key compared with standard SAT attack time. Naturally, T_{eval} is proportional to N_{eval} and N_s.

on the attack ($= T_{attack} + |K|T_{eval}$) and the quality of the best key found in terms of $FM(K^*)$ for various circuits and fixed value of ϵ_g. And table III shows, for the same circuits, how $HD(K^*)$ varies with increasing N_{inst}. While both $FM(K^*)$ and $HD(K^*)$ typically reduce with increasing N_{inst}, the differences aren't significant, which implies that an attacker may choose to spend less time and find a "good enough" key rather than spend more time to find a better (or the correct) key.

A larger N_{inst} means that an instance has a higher chance of being able to duplicate whenever its DI repeats, ensuring with more likelihood that an erroneous bit is not forcefully recorded in the I/O pairs. Since not being able to duplicate implies that an unspecified bit will be rounded to either 0 or 1, which may not be the correct value (see explanation around eqn. 6). However, a larger N_{inst} has the downside of requiring more time to finish the attack.

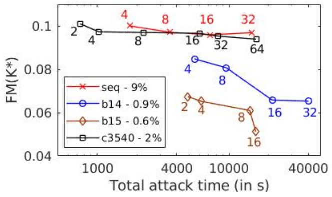

Fig. 6: Dependence of $FM(K^*)$ on total attack time which varies with N_{inst}. Numbers beside each data point are N_{inst}.

Circuit		c3540	b15	seq	b14
ϵ_g		2.0 %	0.6 %	9 %	0.9 %
N_{inst}	1	-	-	-	-
	2	0.0211	0.0107	-	-
	4	0.0210	0.0102	0.0209	0.0123
	8	0.0209	0.0099	0.0209	0.0123
	16	0.0210	**0.0094**	0.0209	0.0118
	32	0.0209		**0.0209**	0.0114

TABLE III: Variations in $HD(K^*)$ of unlocked circuit with N_{inst}. HD in **boldfont** implies that it corresponds to the correct key. A '-' indicates unsuccessful attack.

(E) Let us now relax the assumption that the attacker was aware of the value of ϵ_g. Now s/he has to guess an error, say ϵ'_g, before starting the attack for later estimating BERs during the attack (sec. IV-C). We perform this by comparing (using several random primary inputs) the output uncertainties of (i) the oracle and (ii) the locked circuit with several random key inputs with increasing values of ϵ'_g. When half of the output uncertainties become comparable with a certain ϵ'_g, we stop increasing ϵ'_g and choose that as our guess. Table IV mentions the estimated error probability ϵ'_g against the actual ϵ_g, and the resulting $HD(K^*)$ of the attack. These $HD(K^*)$ values are very close to the respective values in table II where it was assumed that the attacker had knowledge of ϵ_g. Because ϵ'_g was always less than ϵ_g, E_λ had to be reduced for successfully carrying out the attack. However, these results indicate that it is not necessary for an attacker to know (the exact value of) ϵ_g. What is more important in the attack is to estimate the output BERs relatively, because E_λ can be adjusted to include more/less of oracle's outputs within the limit of being potentially highly erroneous.

c3540			c7552			b14		
ϵ_g	ϵ'_g	$HD(K^*)$	ϵ_g	ϵ'_g	$HD(K^*)$	ϵ_g	ϵ'_g	$HD(K^*)$
1.25	0.869	**0.0194**	2.00	0.157	**0.0123**	0.50	0.244	**0.0086**
1.50	1.157	0.0201	2.25	0.192	0.0125	0.75	0.359	**0.0100**
1.75	1.374	0.0205	2.50	0.240	0.0126	0.80	0.396	0.0117
2.00	1.700	0.0208	3.00	0.420	0.0128	0.85	0.401	**0.0109**

TABLE IV: Attacker's guess (ϵ'_g) at error probability and resulting $HD(K^*)$. HD in **boldfont** implies that it corresponds to the correct key.

Comparison with PSAT: We observe the performance of the PSAT attack in [15] by launching it on some circuits and doing 20 runs of the attack. Table V mentions the no. of times in which PSAT ran till completion and found a key. On the contrary, StatSAT could always find the correct key for those circuits (compare with the corresponding results in table II). This proves the superiority of our StatSAT attack.

Circuit	c880 (32-bit key)			b15		c3540	b14	c7552
ϵ_g (in %)	1.0	1.5	2.0	0.1	0.2	1.25	0.5	2.0
PSAT Successful runs	20	5	0	20	0	0	0	0
StatSAT successful ?	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

TABLE V: No. of runs (out of 20) in which PSAT found correct key.

VI. Conclusion

This paper proposes StatSAT, an attack on logic-locked circuits, in an untrusted foundry setting, which are designed to behave probabilistically for application in the approximate/probabilistic computing paradigm. It highlights the drawbacks in an existing piece of work which attacks such circuits with the same threat model, and proposes techniques to overcome them. The attack algorithm developed manages to unlock protected circuits within a reasonable time-frame. This work thus shows that it is possible for an adversary based in an untrusted fab to produce counterfeit of ICs used for fault-tolerant applications. Future work would involve coming up with possible defenses against StatSAT.

Acknowledgement

This work was supported by the National Science Foundation (NSF) grant 1642424 and the Air Force Office of Scientific Research grant FA9550-14-1-0351.

References

[1] M. Rostami *et al.*, "A primer on hardware security: Models, methods, and metrics," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1283–1295, 2014.

[2] J. A. Roy *et al.*, "Epic: Ending piracy of integrated circuits," in *Proceedings of the conference on Design, automation and test in Europe*. ACM, 2008.

[3] S. Dupuis *et al.*, "Logic locking: A survey of proposed methods and evaluation metrics," *Journal of Electronic Testing*, pp. 1–19, 2019.

[4] A. Chakraborty *et al.*, "Keynote: A disquisition on logic locking," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 2019.

[5] P. Tuyls *et al.*, "Read-proof hardware from protective coatings," in *Intl. Workshop on Cryptographic Hardware and Embedded Systems*, 2006.

[6] J. Rajendran *et al.*, "Security analysis of logic obfuscation," in *Proceedings of the 49th Annual Design Automation Conference*. ACM, 2012, pp. 83–89.

[7] ——, "Fault analysis-based logic encryption," *IEEE Transactions on computers*, vol. 64, no. 2, pp. 410–424, 2013.

[8] P. Subramanyan *et al.*, "Evaluating the security of logic encryption algorithms," in *Int. Sym. on Hardware Oriented Security & Trust*. IEEE, 2015.

[9] M. El Massad *et al.*, "Integrated circuit (ic) decamouflaging: Reverse engineering camouflaged ics within minutes." in *NDSS*, 2015, pp. 1–14.

[10] K. V. Palem *et al.*, "Sustaining moore's law in embedded computing through probabilistic and approximate design: retrospects and prospects," in *Proceeding of the 2009 CASES*. ACM, 2009, pp. 1–10.

[11] C.-Y. Chen *et al.*, "Exploiting approximate computing for deep learning acceleration," in *Design, Automation & Test in Europe*. IEEE, 2018.

[12] Y. Li *et al.*, "Low-power noise-immune nanoscale circuit design using coding-based partial mrf method," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 8, pp. 2389–2398, 2018.

[13] T. Rejimon *et al.*, "Probabilistic error modeling for nano-domain logic circuits," *IEEE Trans. on VLSI Systems*, vol. 17, no. 1, pp. 55–65, 2008.

[14] S. Patnaik *et al.*, "Advancing hardware security using polymorphic and stochastic spin-hall effect devices," in *DATE*. IEEE, 2018, pp. 97–102.

[15] ——, "Spin-orbit torque devices for hardware security: From deterministic to probabilistic regime," *IEEE Trans. on Computer-Aided Design*, 2019.

[16] C. Yu *et al.*, "Incremental sat-based reverse engineering of camouflaged logic circuits," *IEEE TCAD*, vol. 36, no. 10, pp. 1647–1659, 2017.

[17] Y. Xie *et al.*, "Mitigating sat attack on logic locking," in *Intl. Conf. on Cryptographic Hardware and Embedded Systems*. Springer, 2016.

[18] M. Yasin *et al.*, "Sarlock: Sat attack resistant logic locking," in *Intl. Symp. on Hardware Oriented Security and Trust*. IEEE, 2016, pp. 236–241.

[19] K. Shamsi *et al.*, "Appsat: Approximately deobfuscating integrated circuits," in *2017 HOST*. IEEE, 2017, pp. 95–100.

[20] M. Yasin *et al.*, "Removal attacks on logic locking and camouflaging techniques," *IEEE Transactions on Emerging Topics in Computing*, 2017.

[21] X. Xu *et al.*, "Novel bypass attack and bdd-based tradeoff analysis against all known logic locking attacks," in *Intl. Conf. on CHES*. Springer, 2017.

[22] H. M. Kamali *et al.*, "Full-lock: Hard distributions of sat instances for obfuscating circuits using fully configurable logic and routing blocks," in *Proceedings of the Design Automation Conference*. ACM, 2019, p. 89.

[23] M. Yasin *et al.*, "Provably-secure logic locking: From theory to practice," in *SIGSAC Conf. on Computer and Communications Security*. ACM, 2017.

[24] A. Sengupta *et al.*, "Atpg-based cost-effective, secure logic locking," in *2018 IEEE 36th VLSI Test Symposium (VTS)*. IEEE, 2018, pp. 1–6.

[25] F. Yang *et al.*, "Stripped functionality logic locking with hamming distance based restore unit (sfll-hd)–unlocked," *IEEE TIFS*, 2019.

[26] P. Chakraborty *et al.*, "Sail: Machine learning guided structural analysis attack on hardware obfuscation," in *AsianHOST*. IEEE, 2018, pp. 56–61.

[27] D. Sirone *et al.*, "Functional analysis attacks on logic locking," in *Design, Automation & Test in Europe (DATE)*. IEEE, 2019, pp. 936–939.

[28] J. Han *et al.*, "Approximate computing: An emerging paradigm for energy-efficient design," in *18th European Test Symposium (ETS)*. IEEE, 2013.

[29] R. Xiao *et al.*, "Gate-level circuit reliability analysis: A survey," *VLSI Design*, vol. 2014, p. 4, 2014.

[30] N. Mohyuddin *et al.*, "Probabilistic error propagation in a logic circuit using the boolean difference calculus," in *Advanced Techniques in Logic Synthesis, Optimizations and Applications*. Springer, 2011, pp. 359–381.