

On Oracle-Guided Random Circuit Learning via Stochastic Boolean Satisfiability

Shakil Ahmed, Kaveh Shamsi

Department of Electrical & Computer Engineering

University of Texas at Dallas, Richardson, USA

Email: {shakil.ahmed, kaveh.shamsi}@utdallas.edu

Abstract—Oracle-guided circuit learning (OGCL) or deobfuscation is the problem of recovering a set of secret key bits from a keyed circuit with the help of queries to a black-box functional oracle of the circuit. This problem has various applications in hardware security, such as in security analysis of obfuscation schemes, side-channel analysis, reverse engineering for trust, and Trojan detection. Boolean satisfiability (SAT)-based algorithms have been used here extensively. In this paper, we explore the adjacent problem of random circuit learning (OGRCL), which is the CL problem when the keyed circuit has an additional set of uncontrollable/unobservable random inputs with known probabilities. This can find applications in deobfuscation of probabilistic circuits, deobfuscation in the presence of noise, side-channel attacks in the presence of noise, optimal random circuit synthesis, and so on. We show for the first time that Boolean stochastic satisfiability (SSAT), which is a generalization of SAT to computing the probability of a given Boolean formula, can be used to devise generic random circuit learning procedures. We implement our proposed algorithms using modern SSAT solvers and showcase their superiority relative to a traditional black-box optimization approach on a set of benchmark circuits.

Index Terms—Circuit Deobfuscation, Circuit Learning, Stochastic Boolean Satisfiability, Random Circuit Learning

I. INTRODUCTION

Given a Boolean circuit $c_e(k, x)$, trying to recover the secret key k_* via oracle black-box queries on x to $c_e(k_*, \cdot)$ is a problem that has been studied over the decades, initially in computational learning theory [1]–[3]. The problem, which we deem oracle-guided circuit learning (OGCL), has appeared in various contexts in hardware security [4]–[9]. Logic obfuscation schemes are techniques that integrated circuit (IC) designers can use to obscure their design from untrusted elements in today’s globalized IC supply chain [10]–[21]. These techniques introduce ambiguity into circuits from an attacker’s viewpoint, which can be modeled in $c_e(k, x)$, and with access to a functional IC as an oracle, the security of the obfuscation scheme reduces to the difficulty of the OGCL procedure. Other forms of ambiguity, such as those from incomplete reverse engineering, side-channel analysis [22], or Trojan localization [23], [24], can all be modeled and attacked similarly via OGCL.

Over the years, various generic algorithms for OGCL have been proposed, where the most notable is the family of Boolean satisfiability (SAT)-based OGCL algorithms [25], [26]. These procedures use a modern SAT-solver to find input patterns on which querying the oracle is informative regarding the key (called discriminating input patterns or DIPs), querying them on the oracle, then reinforcing the input-output observation back

into the SAT solver and repeating the process. This refines the key hypothesis until a provably correct key is found.

In this paper, we define and explore the problem of generic oracle-guided random circuit learning (OGRCL). Here, the keyed circuit is $c_e(k, x, r)$, where k are secret keys as before, but we have additional random inputs r which are random Boolean variables with (Bernoulli) probabilities known to the attacker. We utilize, for the first time to the best of our knowledge, the framework of stochastic boolean satisfiability (SSAT) towards this goal. We deliver the following contributions that are novel to the best of our knowledge:

- We formalize the generic OGRCL problem and show how several other problems in this domain can be reduced to it. We explore how a traditional SAT-based OGCL procedure can run into trouble when working with a circuit with random inputs. We explore the theoretical aspects of the problem, for instance, how the nondeterministic nature of queries can make the formal termination with a provably correct key that SAT-OGCL offers difficult, unless the circuit has sufficiently many points on which it behaves deterministically.
- We make the connection between SSAT and OGRCL. We provide an SSAT-based procedure that, similar to the SAT-based OGCL, can mine for key-informative queries, collect the responses, and find the key that best satisfies the observations all via a series of SSAT calls. We show how this can achieve formal termination when theoretically possible.
- We discover how the runtime for this fully-interactive SSAT OGRCL loop is explosive in the circuit size due to the query mining step being difficult for off-the-shelf SSAT solvers. We hence propose a less-interactive N-queries 1-SSAT call approach that tries to find high-quality queries (using other means instead of N costly SSAT calls), query them, then call the SSAT solver in the end to get a set of best candidate keys.
- We implement our algorithms in software and report runtime data on simulated models, showing our procedures beating simulated-annealing style black-box optimization loops when given the same number of queries and runtime budget. We show that curating the queries affects the final key accuracy. The paper is organized as follows. Section II presents preliminaries, Section III OGRCL via Stochastic SAT, Section IV experiments, and Section V the conclusion.

II. PRELIMINARIES

Generic Adaptive Oracle-Guided Circuit Learning (OGCL). In the (combinational) circuit learning problem, we

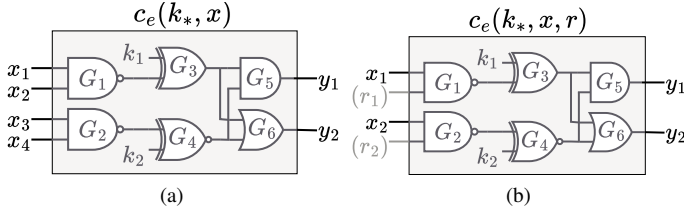


Fig. 1: (a) OGCL: learning the key bits k_1, k_2 via black-box queries to $c_e((k_1, k_2), x)$; (b) OGRCL: the same except the circuit $c_e((k_1, k_2), x, (r_1, r_2))$ has hidden random uncontrollable inputs r_1, r_2 that are updated randomly in each query.

are given a general digital (combinational) circuit $c_e(k, x) : K \times I \mapsto O$ where x are controllable inputs, k are secret (key) uncontrollable inputs with $I = \{0, 1\}^n, O = \{0, 1\}^m, K = \{0, 1\}^l$ being the $n/m/l$ -bit input/output/key space respectively.

The learner/attacker can make adaptively chosen queries to a black-box oracle of $c_e(k_*, x) \equiv c_o(x)$. The goal is to recover some key k_+ such that $c_e(k_+, x) = c_e(k_*, x)$ for all (or $1 - \epsilon$ of) inputs $x \in I$, where k_* is the hidden correct secret key. k_+ here is a key hypothesis that produces a functionality $c_e(k_+, x)$ that is equivalent to (or an ϵ -approximation of) $c_e(k_*, x)$.

SAT-based OGCL. In [25], [26], a generic SAT-based OGCL procedure was proposed. The procedure is as follows: A miter circuit $M \equiv [c_e(k_1, x) \neq c_e(k_2, x)]$ is formed, converted to a conjunctive normal form (CNF) SAT formula, then passed to a SAT solver. If the formula is satisfiable, the solver will return an input \hat{x} and two keys \hat{k}_1, \hat{k}_2 . Here \hat{x} is called a discriminating-input-pattern (DIP) as it produces different outputs under the two keys \hat{k}_1, \hat{k}_2 . As such, querying this input pattern on the functional oracle ($\hat{y} = c_o(\hat{x})$) is guaranteed to eliminate at least one incorrect key. The input-output (IO) constraint is reinforced back into the solver by conjoining ($\hat{y} = c_o(\hat{x}) = c_e(\hat{k}_1, \hat{x}) = c_e(\hat{k}_2, \hat{x})$) to the miter M . This process is repeated until the miter becomes unsatisfiable (UNSAT) under the constraints, at which point a provably functionally correct key may be extracted from the constraints alone (see Algorithm 1).

Algorithm 1: Given oracle access to $c_o(x) \equiv c_e(k_*, x)$ and circuit $c_e(k, x)$, returns a functionally correct key k_+ .

```

1 Function SATOGCL ( $c_e, c_o$  as black-box) :
2    $F \leftarrow true, M \leftarrow [c_e(k_1, x) \neq c_e(k_2, x)]$ 
3   while  $F \wedge M$  is solvable do
4      $\hat{x}, \hat{k}_1, \hat{k}_2 \leftarrow \text{SAT}(F \wedge M)$ 
5      $\hat{y} \leftarrow c_o(\hat{x})$ 
6      $F \leftarrow F \wedge (c_e(k_1, \hat{x}) = \hat{y}) \wedge (c_e(k_2, \hat{x}) = \hat{y})$ 
7   satisfy  $F$  with  $\hat{k}_1$  and  $\hat{k}_2$ 
8   return  $\hat{k}_1$  as a correct key  $k_+$ 

```

Stochastic SAT (SSAT). Stochastic satisfiability (SSAT) is a generalization of the classical Boolean satisfiability (SAT) that incorporates probabilistic quantification over Boolean variables. In SSAT, variables are quantified either existentially (\exists) or randomly (\mathfrak{R}), where a randomly quantified variable is associated with a probability value indicating the likelihood of it being assigned true. A typical SSAT formula has the form

$Q_1 x_1 Q_2 x_2 \cdots Q_n x_n \cdot \phi(x_1, \dots, x_n)$, where each Q_i is either \exists or \mathfrak{R}^{p_i} (random variable with probability p_i), and ϕ is a propositional formula in conjunctive normal form (CNF). The solution to the SSAT problem is an assignment (strategy) to the existentially quantified variables that maximizes the probability of ϕ being true over the random variables. SSAT provides a natural framework for reasoning under uncertainty and has applications in probabilistic planning, model checking, fair machine learning, and quantitative verification [27]–[30].

Stochastic SAT Solving. SSAT with interleaved quantifiers is a PSPACE-complete problem. As such, it is generally harder than SAT (which is NP-complete), and more closely related to quantified Boolean formula (QBF) satisfiability. Various techniques have been developed over the years for SSAT solving. ErSSAT [31] uses techniques from QBF solving, such as clause selection in conjunction with model counting. Model counting (#SAT) is the problem of counting the number of solutions to a CNF SAT formula, which is typically done using recursive procedures, binary decision diagrams (BDDs) [32], and so on. SharpSSAT [33] (the solver we use in this paper) belongs to a family of SSAT solvers based on a Davis–Putnam–Logemann–Loveland (DPLL)-style recursive procedure that branches on variables (set x_i to 0 and 1 in some CNF fragment), trying to compute the probability of the two resulting CNF components that result by recursing until trivial components are encountered, and then backtracking to the top of the recursion tree. This is paired with caching the probability of components that have already been solved, in addition to conflict-driven clause learning (CDCL) for avoiding unsatisfiable branches and non-chronological backtracking.

Our proposed procedures here are agnostic to the underlying SSAT solver algorithm, so any solver with support for 2 levels of interleaved quantification (Exist-Random or ER) can be used. Note that SSAT solving is an active research area that will continue to advance, improving the power of the procedures we present here.

III. ORACLE-GUIDED RANDOM CIRCUIT LEARNING VIA STOCHASTIC SATISFIABILITY

A. Formalism

We first formally define the OGRCL problem, similar to the OGCL procedure, and mention how a couple of related problems in the literature, i.e., probabilistic circuit learning and nondeterministic oracles (such as an erroneous power oracle), can be reduced to special cases of OGRCL.

Oracle-Guided Random Circuit Learning (OGRCL). We define OGRCL as follows: we are given a generic Boolean (combinational) circuit $c_e(k, x, r) : K \times I \times R \mapsto O$ where x are controllable inputs, k are secret (key) uncontrollable inputs, r are uncontrollable/unobservable random inputs with $I = \{0, 1\}^n, O = \{0, 1\}^m, K = \{0, 1\}^l, R = \{0, 1\}^s$ being the $n/m/l/s$ -bit input/output/key/random space respectively. The learner can make queries to the function $c_e(k_*, x, r)$ as an oracle, but each random input r_i hidden from the learner will be sampled from a Bernoulli distribution with parameter/probability $p(r_i)$ on each query inside the oracle (see Figure

1). The goal of the learner is to find a key k_ϵ such that $\mathbb{P}_{x \sim I, r \sim R}[c_e(k_\epsilon, x, r) \neq c_e(k_*, x, r)] < \epsilon$.

Notice that in the definition, the success criteria for the learner is made probabilistic with ϵ being the disagreement probability taken over all controllable and random inputs. There are alternative ways to define success here, but $\epsilon = 0$ here captures true random circuit equivalence. This is possible for an oracle-guided attacker to land on or to provably achieve in special cases. Also note that the correct key k_* by definition satisfies this perfect key criterion.

Probabilistic Circuit Learning (PCL). In [34], the authors explored the question of circuit deobfuscation of probabilistic circuits. Here, the target circuit is an otherwise normal Boolean combinational circuit $c_e(k, x)$ except for the fact that every wire in the circuit on each query to the circuit has a small probability p_f of flipping. For this, they proposed a SAT-based procedure that involved repeating queries, finding unstable ones, and then branching on the output for those queries being either 0 or 1, creating up to some threshold of parallel instances of the SAT attack that are, in the end, studied to find the best key.

PCL can be modeled as a special case of OGRCL. We can take the circuit $c_e(k, x)$ and XOR each of its internal wires with a random input $w'_i = w_i \oplus r_i$ where the probability of the random input $p(r_i)$ matches the flip probability of the probabilistic circuit p_f . This does, in practice, introduce a large number of random inputs to the circuit, and hence, specialized PCL solvers in the low p_f probability regime may be better suited than a generic OGRCL scheme that we explore here.

Nondeterministic Oracles. In [35], the authors proposed a locking scheme that introduces randomness into the circuit output when queried in scan-chain mode. This can confuse a conventional SAT-based OGCL attacker, as the output queries are not deterministically aligned with the circuit SAT conditions in the procedure. Nondeterministic oracles of this sort can also be modeled as special cases of OGRCL by simply XORing the output wires of the circuit $c_e(k, x)$ with random inputs where $p(r_i)$ matches the probability of a random flip in the oracle's output. This reduction can be done with a smaller number of random inputs compared to the PCL-OGRCL reduction. Note that if the probability of the oracle flipping on any query is perfectly 0.5, then the output of the oracle becomes random and hence not informative for an OG attack. This will effectively eliminate the oracle and reduce the attack to an oracle-less regime. Additionally, side-channel attacks that run OGCL on power models [22] but for which the side-channel queries may be noisy [36] can be modeled the same way via OGRCL.

The above reductions indicate that OGRCL is a problem worth exploring in its own right. Note that the formalism is highly expressive, allowing for any Boolean integration of random Boolean inputs with a circuit at arbitrary locations to be described. However, it is important to keep in mind that a generic procedure may be outperformed by a specialized procedure focused on speeding up special cases such as PCL.

B. SAT-based OGCL against Random Circuits

We begin now to sketch an OGRCL procedure by first seeing how the baseline SAT-based deterministic OGCL may proceed

under a $c_e(k, x, r)$ random circuit. This will be useful for later constructing SSAT-based routines.

The DIP Miter. The SAT-based OGCL algorithm will start by constructing a miter: $M = [c_e(k_1, x, r) \neq c_e(k_2, x, r)]$, which is meant to search for inputs that create disagreement at the output for different keys. The question is what to do with the additional r random inputs. If they are tied together in the miter, solving this condition will return $\hat{x}, \hat{r}, \hat{k}_1, \hat{k}_2$, which concretizes r . This means that for some choice of the random inputs, the input \hat{x} is a DIP. However, for other choices of r it may not be. Here, the DIP loses some of its meaning. Not sharing the r inputs in the miter will make things worse.

Input-Output Constraints. If we ignore the issue with \hat{x} not being a DIP under all choices of r , and proceed with querying \hat{x} on the oracle, we obtain $\hat{y}^1 = c_e(k_*, \hat{x}, r^1)$. This output, however, may change if we retry the query. If we make this query $q \rightarrow \infty$ many times, the output varies, yet its expectation converges to the total probability $\mathbb{E}[\hat{y}] = \mathbb{P}_{r \sim R}[c_e(k_*, \hat{x}, r)]$.

If we try to reinforce one of these \hat{y} query observations back into the solver, we obtain the condition $F \equiv [c_e(k_1, \hat{x}, r_1) = c_e(k_2, \hat{x}, r_2) = \hat{y}]$. Again, we have the question of what to do with the r_1, r_2 variables here. Since we do not know the exact value for these random inputs that were sampled randomly inside the black-box, we cannot fix them. If we leave them free, this gives the SAT-solver extra degrees of freedom, which can veer it off course when it comes to finding the correct key.

Key Elimination. We can also explore here the key elimination that the SAT-based OGCL guarantees. A conventional DIP, \hat{x} , is guaranteed to eliminate at least one key. But in the OGRCL, a query may not deterministically eliminate a key. If $c_e(k_1, \hat{x}, r_1) = 0$ and $c_e(k_1, \hat{x}, r_2) = 1$, i.e., there exist two different random input choices that produce different outputs under the same key, then querying the oracle on this input cannot strictly eliminate one of the keys as wrong. So, based on the randomness, both query outcomes are possible, and the key cannot be ruled out without resorting to statistical arguments.

C. Baseline SSAT-based OGRCL

We present our first baseline procedure here with a simple high-level idea: simply lift the (Circuit-)SAT formulations/calls in the SAT-based OGCL procedure into SSAT formulations/calls for the SSAT-based OGRCL procedure.

SSAT OGRCL Miter. We begin with the miter. By lifting to SSAT (with some extra work), we get:

$$MPM \equiv (\exists k_1 \exists k_2 \exists x \forall r [c_e(k_1, x, r) \neq c_e(k_2, x, r)])$$

Solving this miter via (Tseiting transform of the circuit c_e expressions to) SSAT corresponds to searching for a pair of keys \hat{k}_1, \hat{k}_2 , and an input pattern \hat{x} that will maximize the probability of disagreement at the output under the two keys over the shared random inputs r . We can hence call this a most-probable discriminating input pattern (MPDIP) (see Figure 2b).

Input-Output Constraints. When an MPDIP x_q is found, if we query it on the oracle once, we obtain a non-deterministic $y_q = c_e(k_*, x_q, r_q)$. What we want from an IO constraint here is to find a key k for which $\mathbb{P}_{r \sim R}[c_e(k, x_q, r)]$ is as close as possible to the expectation value of y_q on the oracle. Trying

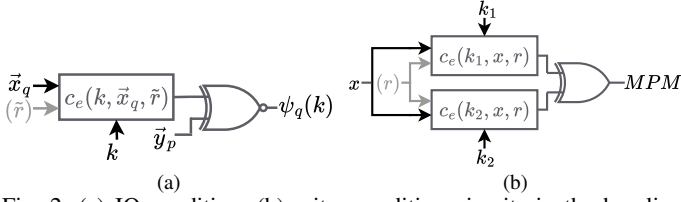


Fig. 2: (a) IO condition, (b) miter condition circuits in the baseline SSAT OGRCL procedure.

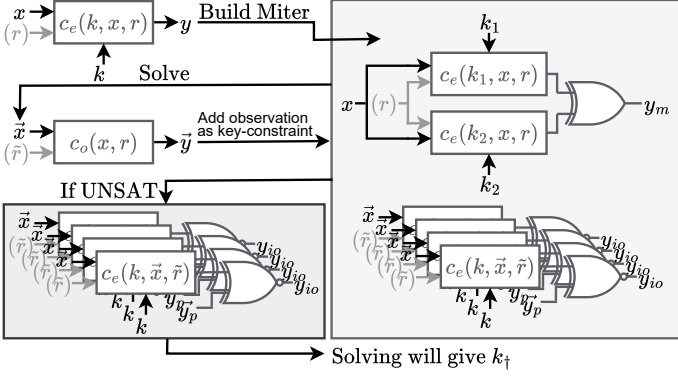


Fig. 3: SSAT-based random circuit learning flow.

to force a dependent variable to have as close as possible a probability to a target probability is not natively supported by SSAT. While such an SSAT extension is a topic of our future work, here we settle on an approximate solution which is to simply take the majority vote of y_q on some number of repeated queries (as seen in Figure 2a) and using it as the IO constraint target: $\psi_q(k) = \exists k \forall r [c_e(k, x_q, r) = (\text{mean}(y_q) > 0.5)]$.

Multiple of the above conditions can be ANDed together. The statistics of each $\psi_q(k)$ will be independent if the random inputs are not shared in each $\psi_q(k)$, and the probability of their AND will be equivalent to the product of their probabilities. As such, maximizing the probability of the AND of all of them should return a key that tries to maximize the probability of each IO constraint being satisfied, and hence general agreement with the oracle, which is the end goal here.

We can pair the above miter in a loop with the IO conditions (as seen in Figure 3), same as SAT-based OGCL, to build an SSAT-based procedure for OGRCL. Note that, unlike in SAT-based OGCL, the loop may not terminate (unless in special cases discussed later), and hence the user needs to manage the runtime via query limits based on the desired level of effort.

D. Speeding up SSAT-based OGRCL

The above raw baseline procedure is a generic (approximate) OGRCL procedure. However, in our experiments, we found that it struggles with larger (than tens of gates) circuits, as the numerous intermediate SSAT calls tend to have explosive runtimes. While we suspect that this may be due to internal dynamics in the SSAT solver that we aim to explore more in future work, here we utilize a few techniques to get around this and speed up the procedure as we discuss herein.

The Less-Interactive Approach. Note that the baseline procedure is fully interactive, i.e., it mines a single MPDIP query via an expensive SSAT call, then performs the query and extends the IO constraints, then repeats all of that on

every query. Therefore, one strategy to speed things up here is to collect highly informative queries through other less computationally demanding means and then use the SSAT solver in the end, once, to find a good candidate key. This N-query 1-SSAT call approach is less interactive. We explain some of these alternative query mining schemes herein.

Deterministic Discriminating Input Patterns. The MPDIP miter ($\exists k_1 \exists k_2 \exists x \forall r [c_e(k_1, x, r) \neq c_e(k_2, x, r)]$) searches for an input that maximizes the probability of disagreement at the output for two different keys over the random inputs. If the SSAT solver returns 1 as the probability, that means that an input pattern has been found for which the disagreement is certain. We call these deterministic DDIPs here. It is possible to show that these only occur when the randomness is prevented from reaching the output on these input/key combinations. We can let the SSAT solver try to extract as many of these DDIPs initially, even non-interactively, by running the SSAT solver on the miter, banning the found solution if the probability is 1, then repeating until no more can be found. In practice, most larger circuits have few of these, but on smaller circuits, they exist. The inclusion of these DDIPs in the IO constraint can provably remove keys, and it can lead to an UNSAT-miter true termination (provably correct key recovery).

MPDIP Miter Optimized Input Patterns. Instead of performing a complete SSAT call to extract one MPDIP query, we can let the miter search until a user-defined computational resource budget is met. This would return the input pattern with the highest probability found during the search. While this query may be less informative than the full MPDIP, it still has a higher probability of satisfying the miter. Additionally, it can eliminate a substantial number of incorrect keys while being computationally inexpensive. This type of resource-efficient SSAT call can also be beneficial for solving a costly key condition $\psi(k, r)$, especially in the case of large circuits.

Heuristic Metric Optimized Input Patterns. In generic circuit learning, the concept of discriminating-input-pattern (DIP) is inspired by the traditional principle of uncertainty sampling in machine learning [37], which says that queries that create the most uncertainty in the learner's hypothesis should be the most informative ones. Utilizing this principle, we can derive a heuristic fitness metric for a given query in the context of random circuit learning, which is similar to entropy in information theory. If a query is absolutely uninformative about the key space, it eliminates the dependency of the output on the key. On the other hand, a highly informative query makes the output highly sensitive to the key. This sensitivity can be captured via the output signal probability over choices of the key. An output signal probability closer to 0.5 will represent a higher uncertainty input sample. For a large number of keys n_k , we can define the probability of an output y_i being true as $p_{y_i} = \frac{1}{n_k} \sum_{n_k} y_i$ and the entropy associated with it as $H_i = -p_{y_i} \log(p_{y_i}) - (1-p_{y_i}) \log(1-p_{y_i})$. The total entropy is calculated as $H_{total} = \sum_i H_i$. So, we aim to find out an input pattern for which this H_{total} is maximized. This formula can be treated as a black-box optimization problem. The objective function is the total entropy, and the input pattern \hat{x} is the

optimization space. Black-box optimization tools like simulated annealing, the genetic algorithm, etc. [38], can be utilized for this purpose. Our implementation uses simulated annealing.

Algorithm 2: Given q total iterations, q_m MPDIP queries, q_r query repetitions, t extracted keys, oracle access to $c_e(k_*, \cdot, r)$, and random circuit $c_e(k, x, r)$, returns key k_{\dagger} . `IndQuery` here is the independent query mining strategy, i.e., random, heuristic, or MPDIP miter-optimized queries.

```

1 Function SSATOGRL( $c_e, c_o$  as black-box) :
2    $\psi \leftarrow true, i \leftarrow 0$ 
3    $MPM \leftarrow \exists k_1 \exists k_2 \exists x, \forall r [c_e(k_1, x, r) \neq c_e(k_2, x, r)]$ 
4   while  $\psi \wedge MPM$  is solvable and  $i < q_m$  do
5      $\hat{x}, \hat{k}_0, \hat{k}_1 \leftarrow SSAT(\psi \wedge MPM)$ 
6      $\hat{y} \leftarrow \text{maj}_{j \in [1, q_r]}(c_e(k_*, \hat{x}, r_j))$ 
7      $\psi_i(k, r) \leftarrow (c_e(k_1, \hat{x}, r^i) = \hat{y}) \wedge (c_e(k_2, \hat{x}, r^i) = \hat{y})$ 
8      $\psi(k, r) \leftarrow \psi(k, r) \wedge \psi_i(k, r)$ 
9      $i \leftarrow i + 1$ 
10  while  $i < q$  do
11     $\hat{x} \leftarrow \text{IndQuery}(c_e(k, x, r))$ 
12     $\hat{y} \leftarrow \text{maj}_{j \in [1, q_r]}(c_e(k_*, \hat{x}, r_j))$ 
13     $\psi_i(k, r) \leftarrow (c_e(k, \hat{x}, r) = \hat{y})$ 
14     $\psi(k, r) \leftarrow \psi(k, r) \wedge \psi_i(k, r)$ 
15     $i \leftarrow i + 1$ 
16  while  $\mathcal{K}.size() < t$  and  $\psi$  is SSAT-solvable do
17     $k_i \leftarrow SSAT(\psi)$ 
18     $\mathcal{K}.addkey(k_i)$ 
19     $\psi(k, r).bankey(k_i)$ 
20   $\mathcal{K}.sortwrtkeyerror()$ 
21  return  $k_{\dagger} \leftarrow \mathcal{K}[0]$  being the current best key hypothesis.

```

Random Input Patterns. Another straightforward approach is to give up on extracting discriminating queries (uncertainty samples) intelligently and simply query the random circuit on n random input/difference patterns. Interestingly, this can certainly save computational resources in mining the queries and outperform other query techniques as we only have to use the SSAT solver once to solve the final key condition $\psi(k, r)$.

SSAT Solution Enumeration. Once a set of queries is collected and their IO constraints assembled into $\psi(k, r)$, we can, instead of just finding one key that maximizes the probability of $\psi(k, r)$ via SSAT, extract a collection of keys. This can be done with a black-box SSAT solver by making a call to find one solution, banning this via a simple clause, then solving the formula again until some limit is reached or the formula becomes UNSAT altogether. We can then take these extracted keys and test them via oracle queries to get the best. More efficient enumeration may be possible by modifying the SSAT solver, but we leave this to future work.

The final flow for our SSAT-based OGRCL can be seen in Algorithm 2. This algorithm can be made fully interactive or not by controlling the query limits passed to it.

IV. EXPERIMENTS

Setup. We implemented our algorithms in C++ using the `neos` framework [39]. We use the SharpSSAT solver [33] as it was faster than the other solvers (ClauSSAT [40], ssatABC [31]) we tried in preliminary runs (ErSSAT [31] produced

incorrect results in some cases). We utilize ABC [41] to simplify circuits/conditions. All tests were run on an AMD EPYC 7551 32-core/128-thread server with 128GB of RAM. We used a set of ISCAS combinational [42] and sequential (next-state) [43] circuits to generate the random circuits (see Table I). For generating random circuits, we randomly insert l XORs with keys first, then s XORs with random inputs. All random inputs have the same probability of p_r that we vary across our tests. Functional queries were all simulated via $c_e(k_*, x, r)$ circuits.

bench	#ins	#outs	#gates	bench	#ins	#outs	#gates	bench	#ins	#outs	#gates
randnet4	4	1	12	s349	24	26	161	s1238	32	32	509
randnet5	5	2	22	s400	25	27	164	c1355	41	32	546
randnet6	6	2	64	s499	23	44	174	c1908	33	25	880
randnet7	7	2	130	c499	41	32	202	c2670	157	64	1193
randnet8	8	2	223	s510	25	13	211	c3540	50	22	1669
s298	17	20	119	s832	23	24	287	c5315	178	123	2307
s386	13	13	159	s641	54	42	379	c7552	206	107	3512
c432	36	7	160	c880	60	26	388	-	-	-	-

TABLE I: Combinational and sequential benchmark set used. “randnet” are small circuits synthesized via ABC from random truth tables.

bench	p_r			0.05			0.1			0.15			0.2		
	#k	#r	#g	time	#q	kerr	time	#q	kerr	time	#q	kerr	time	#q	kerr
randnet4	2	2	16	0.9	6	E	0.8	5	E	0.9	6	E	0.7	4	E
randnet4	3	3	18	0.8	5	E	0.7	4	E	0.6	3	E	0.7	4	E
randnet5	3	3	28	0.7	4	E	0.6	3	E	0.6	3	E	1	7	E
randnet5	5	5	32	1.2	9	E	3.5	22	E	0.8	5	E	2.7	19	E
randnet6	7	7	78	2.2	7	E	210	66	E	1.3	8	E	1.8	10	E
randnet6	13	13	90	770	21	E	to	1	U	to	1	U	to	1	U
randnet7	13	13	156	29	18	E	460	75	E	to	1	U	370	67	E
randnet7	26	26	182	to	1	U	to	1	U	to	1	U	to	1	U
randnet8	23	23	269	210	136	E	to	1	U	to	1	U	to	2	U
randnet8	45	45	313	to	1	U	to	1	U	to	1	U	to	1	U

TABLE II: Baseline fully-interactive SSAT-OGRL on small circuits with #k/#r/#g many keys/randoms/gates. “to” is timeout of 30mins and times are in seconds. “E” under “kerr” denotes formally correct keys.

Baseline SAT-OGRL against OGRCL. We first ran the baseline SAT-based OGRL on random circuits with the random inputs shared in the miter, and random inputs in the IO conditions set to constants sampled from a binary Bernoulli distribution with probability p_r . As expected, the baseline deterministic SAT-based OGRL could not perform well here, even with a few small probability random inputs. On average, it achieved a 56.20% key bit accuracy, not much better than random guessing and iterating an average of 1.63 DIPs before hitting early termination due to UNSAT IO constraints or miters. Most of the time, it terminated after the first DIP.

Baseline SSAT-OGRL. Our baseline fully interactive SSAT-OGRL could iterate and find provably correct keys for tiny circuits as shown in Table II. However, computational complexity (both in terms of time and memory) increases drastically even with a small size increase of the circuit (see Table II) beyond 7 inputs reaching the 30 minute timeout. The joint MPDIP miter ANDed with the IO constraints seemed to produce hard SSAT instances.

Less-Interactive SSAT-OGRL. As seen in Table III and Figures 4, our SSAT procedure with less-interactive MPDIP miter optimized querying (denoted as “miter + SSAT”) shows superior performance compared to the black-box optimization (denoted as “BBO”) baseline (an OGRCL procedure that tries

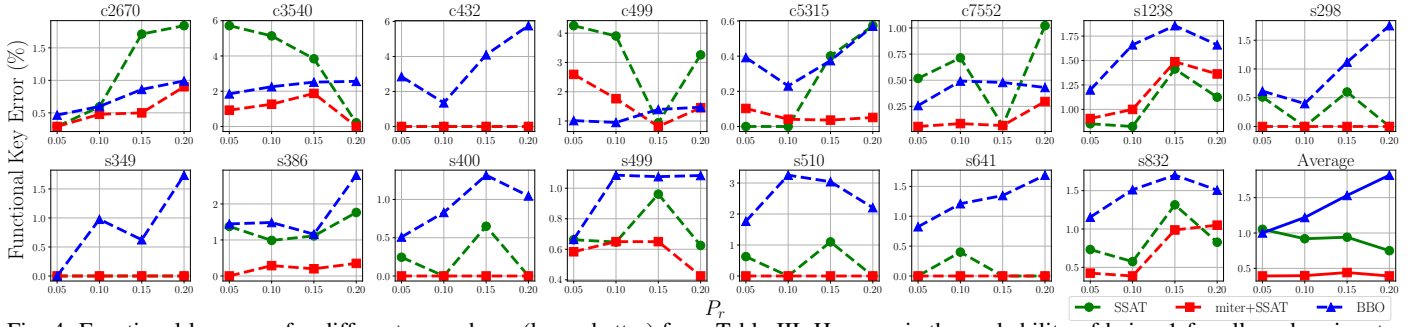


Fig. 4: Functional key error for different procedures (lower better) from Table III. Here, p_r is the probability of being 1 for all random inputs.

p_r	0.05									0.1									0.15									0.2								
	method			SSAT			miter + SSAT			BBO			SSAT			miter + SSAT			BBO			SSAT			miter + SSAT			BBO								
bench	#k	#r	#q	time	kerr	time	kerr	kerr	#q	time	kerr	time	kerr	kerr	#q	time	kerr	time	kerr	kerr	#q	time	kerr	time	kerr	kerr										
s298	6	6	7	5.2	0.01	24.2	E	0.01	7	2.6	E	20.0	E	0.01	8	12.5	0.01	135	E	0.01	8	5.3	E	38.0	E	E										
s298	12	12	13	45.0	E	258	E	0.01	14	22.8	E	145	E	E	15	172	E	212	E	0.01	15	51.0	E	285	E	0.04										
s386	8	8	9	5.8	0.01	155	E	0.02	9	18.5	0.01	87.5	E	0.01	10	50.0	0.01	175	E	0.01	10	25.2	0.01	185	E	0.03										
s386	16	16	17	355	0.01	210	E	0.01	18	186	0.01	272	0.01	0.02	19	187	0.01	278	0.00	0.02	20	350	0.02	335	0.01	0.03										
c432	8	8	9	14.6	E	145	E	0.02	9	30.4	E	140	E	E	10	30.8	E	242	E	0.05	10	29.4	E	235	E	0.04										
c432	16	16	17	270	E	322	E	0.04	18	360	E	352	E	0.03	19	370	E	362	E	0.03	20	382	E	342	E	0.07										
s349	9	9	10	26.0	E	180	E	E	10	51.0	E	168	E	0.00	11	6.7	E	73.8	E	E	12	73.8	E	250	E	0.01										
s349	17	17	18	370	E	355	E	E	19	330	E	345	E	0.02	20	365	E	340	E	0.01	22	372	E	332	E	0.03										
s400	9	9	10	6.2	E	26.2	E	E	10	26.5	E	172	E	0.01	11	10.6	E	54.0	E	0.01	12	33.8	E	172	E	0.01										
s400	17	17	18	34.2	0.00	270	E	0.01	19	380	E	290	E	0.01	20	275	0.01	272	E	0.02	22	312	E	310	E	0.02										
s499	9	9	10	14.0	0.00	145	0.00	0.00	10	7.2	0.01	60.2	E	0.01	11	7.2	0.01	14.2	E	0.01	12	14.0	0.00	200	0.01	0.01										
s499	18	18	19	362	0.01	295	0.01	0.01	20	355	0.01	298	0.01	0.01	22	362	0.01	318	0.01	0.01	23	130	0.01	325	E	0.02										
c499	11	11	24	72.8	0.00	215	0.00	0.01	25	47.2	E	198	E	0.01	26	67.0	0.00	220	E	0.01	28	66.8	E	225	E	0.02										
c499	21	21	45	to	U	1000	0.05	0.01	47	to	U	1000	0.04	0.01	50	1105	0.02	965	0.02	0.02	53	to	U	1100	0.03	0.01										
s510	11	11	12	53.5	0.01	180	E	0.01	13	68.0	E	125	E	0.03	13	205	0.02	205	E	0.02	14	51.0	E	164	E	0.01										
s510	22	22	24	495	E	390	E	0.02	25	510	E	418	E	0.04	26	490	E	398	E	0.04	28	512	E	458	E	0.03										
s832	15	15	16	425	0.01	148	E	0.01	17	131	0.01	160	E	0.01	18	99.0	0.01	308	E	0.01	19	216	0.01	230	E	0.01										
s832	29	29	31	688	0.01	470	0.01	0.01	33	810	E	550	0.01	0.02	35	695	0.02	548	0.02	0.02	37	715	0.01	568	0.02	0.02										
s641	19	19	20	585	E	380	E	0.00	22	722	E	430	E	0.01	23	710	E	422	E	0.01	24	285	E	392	E	0.01										
s641	38	38	40	122	E	695	E	0.01	43	468	0.01	975	E	0.02	45	200	E	882	E	0.02	48	88.8	E	755	E	0.02										
c880	20	20	43	122	0.00	425	E	0.01	45	118	E	398	0.00	0.01	48	125	E	438	E	0.01	50	125	E	462	E	0.03										
c880	39	39	83	120	0.08	to	0.07	0.01	87	590	0.02	905	0.02	0.03	92	665	0.04	930	0.02	0.03	98	135	0.06	to	0.05	0.03										
s1238	26	26	55	162	0.01	465	0.01	0.01	58	162	0.01	485	0.01	0.01	62	155	0.01	502	0.01	0.02	65	155	0.01	458	0.01	0.01										
s1238	51	51	108	182	0.01	730	0.01	0.01	114	380	0.01	862	0.01	0.02	120	208	0.01	800	0.02	0.02	128	265	0.01	878	0.02	0.02										
c1355	16	16	17	478	E	510	E	0.01	18	488	E	512	E	0.01	19	312	E	518	E	0.01	20	to	0.01	1150	0.01	0.01										
c1355	32	32	34	to	U	to	0.04	0.01	36	to	U	to	0.04	0.02	38	to	U	to	0.05	0.02	40	to	U	to	0.05	0.01										
c1908	16	16	17	385	E	708	E	0.01	18	605	E	790	E	0.02	19	578	E	710	E	0.01	20	492	0.02	700	0.02	0.02										
c1908	32	32	34	440	0.01	to	0.03	0.01	36	to	U	to	0.04	0.02	38	440	0.04	to	0.04	0.02	40	to	U	to	0.04	0.03										
c2670	16	16	17	740	E	930	E	0.00	18	to	0.00	980	0.00	0.00	19	to	0.01	945	0.01	0.01	20	to	0.01	842	0.01	0.01										
c2670	32	32	34	1065	0.01	to	0.01	0.01	36	to	0.01	to	0.01	0.01	38	580	0.03	to	0.00	0.01	40	440	0.03	to	0.01	0.01										
c3540	16	16	17	to	U	to	0.01	0.02	18	940	0.02	742	0.02	0.02	19	838	0.01	1148	0.01	0.02	20	301	0.00	630	E	0.01										
c3540	32	32	34	390	0.04	to	0.00	0.02	36	540	0.08	1180	0.01	0.03	38	415	0.07	to	0.03	0.03	40	542	E	to	E	0.04										
c5315	16	16	17	to	E	to	0.00	0.00	18	900	E	685	0.00	0.00	19	742	E	912	0.00	0.00	20	678	E	1080	0.00	0.00										
c5315	32	32	34	540	E	to	E	0.00	36	to	E	to	E	0.00	38	490	0.01	to	E	0.00	40	120	0.01	1105	E	0.01										
c7552	16	16	17	540	E	to	E	E	18	1125	0.00	to	0.00	0.00	19	332	E	to	E	0.00	20	1055	E	1055	0.00	0.00										
c7552	32	32	34	120	0.01	to	0.00	0.01	36	1155	0.01	1030	E	0.01	38	942	0.00	to	0.00	0.01	40	515	0.02	to	0.00	0.01										

TABLE III: SSATOGRL algorithm with ranked keys results. BBO runs for 10 min with no iteration limit (or 20 min for circuits with > 500 gates). Each experiment is run 4 times, and all the values are the mean value of these 4 experiments. “#k” and “#r” denote the number of keys and random inputs, respectively, in the circuit after gate-overhead-based k/r insertion. “#q” is the number of queries made. “kerr” is the error rate (0-1) of the best key hypothesis at the end of the attack (with E denoting formal equivalence). “time” are in seconds.

to find a key that minimizes disagreement with the oracle via a simulated annealing algorithm given the same query budget as the SSAT procedure), with $10.25\times$ more success rate in key recovery. We can observe an average of $3.21\times$ final functional key error improvement over the BBO procedure for SSAT paired with the key ranking strategy (see Figure 4) and $8\times$ more success rate in key recovery for SSAT paired with random querying r_a (denoted as “SSAT”). However, this N-random query 1-SSAT call (“SSAT”) is the least time-consuming, with an average speed of $11.04\times$ over BBO (see Table III).

V. CONCLUSION

In this paper, we proposed and evaluated a set of novel SSAT-based oracle-guided random circuit learning procedures. Further improving the reliability of these procedures, exploring SSAT solver modifications that can improve them, and exploring beyond-SSAT formulations will be important future research goals here.

ACKNOWLEDGMENT

This work was supported by a grant from the National Science Foundation (NSF-2155189).

REFERENCES

- [1] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri, "Security analysis of integrated circuit camouflaging," in *Proc. ACM Conf. on Computer & Communications Security*, 2013.
- [2] J. A. Roy, F. Koushanfar, and I. L. Markov, "Epic: Ending piracy of integrated circuits," in *Proc. Design, Automation and Test in Europe*, ser. DATE '08, 2008, pp. 1069–1074.
- [3] K. Vaidyanathan, B. P. Das, E. Sumbul, R. Liu, and L. Pileggi, "Building trusted ICs using split fabrication," in *Proc. IEEE Int. Symp. on Hardware Oriented Security and Trust*, 2014.
- [4] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "AppSAT: Approximately deobfuscating integrated circuits," in *Proc. IEEE Int. Symp. on Hardware Oriented Security and Trust*, 2017, pp. 46–51.
- [5] K. Shamsi, D. Z. Pan, and Y. Jin, "On the impossibility of approximation-resilient circuit locking," in *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2019, pp. 161–170.
- [6] D. Liu, C. Yu, X. Zhang, and D. Holcomb, "Oracle-guided incremental sat solving to reverse engineer camouflaged logic circuits," 2016.
- [7] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "Circuit obfuscation and oracle-guided attacks: Who can prevail?" in *Proc. IEEE Great Lakes Symp. on VLSI*, 2017, pp. 357–362.
- [8] Y. Shen and H. Zhou, "Double dip: Re-evaluating security of logic encryption algorithms," in *Proc. IEEE Great Lakes Symp. on VLSI*. ACM, 2017, pp. 179–184.
- [9] K. Shamsi, M. Li, D. Z. Pan, and Y. Jin, "Kc2: Key-condition crunching for fast sequential circuit deobfuscation," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 534–539.
- [10] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security analysis of logic obfuscation," in *Proc. IEEE/ACM Design Automation Conf.*, 2012, pp. 83–89.
- [11] S. Dupuis, P.-S. Ba, G. Di Natale, M.-L. Flottes, and B. Rouzeyre, "A novel hardware logic encryption technique for thwarting illegal overproduction and hardware trojans," in *IEEE Int. Online Testing Symp.* IEEE, 2014, pp. 49–54.
- [12] J. Rajendran, H. Zhang, C. Zhang, G. S. Rose, Y. Pino, O. Sinanoglu, and R. Karri, "Fault analysis-based logic encryption," *IEEE Transactions on Computers*, vol. 64, no. 2, pp. 410–424, 2015.
- [13] M. Yasin, J. J. Rajendran, O. Sinanoglu, and R. Karri, "On improving the security of logic locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 9, pp. 1411–1424, 2016.
- [14] M. Yasin, B. Mazumdar, J. Rajendran, and O. Sinanoglu, "Sarlock: Sat attack resistant logic locking," in *Proc. IEEE Int. Symp. on Hardware Oriented Security and Trust*, 2016, pp. 236–241.
- [15] Y. Xie and A. Srivastava, "Mitigating sat attack on logic locking," in *Proc. Int. Conf. on Cryptographic Hardw. and Embed. Systems*. Springer, 2016, pp. 127–146.
- [16] M. Yasin, A. Sengupta, B. C. Schafer, Y. Makris, O. Sinanoglu, and J. J. Rajendran, "What to lock?: Functional and parametric locking," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*. ACM, 2017, pp. 351–356.
- [17] M. Yasin, A. Sengupta, M. Ashraf, M. Nabeel, J. Rajendran, and O. Sinanoglu, "Provably-secure logic locking: From theory to practice," in *Proc. ACM Conf. on Computer & Communications Security*, 2017, pp. 1–1.
- [18] K. Shamsi, M. Li, D. Z. Pan, and Y. Jin, "Cross-lock: Dense layout-level interconnect locking using cross-bar architectures," in *Proceedings of the 2018 on Great Lakes Symposium on VLSI*. ACM, 2018, pp. 147–152.
- [19] B. Liu and B. Wang, "Embedded reconfigurable logic for asic design obfuscation against supply chain attacks," in *Proc. Design, Automation and Test in Europe*, 2014, pp. 1–6.
- [20] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "Cyclic obfuscation for creating sat-unresolvable circuits," in *Proc. IEEE Great Lakes Symp. on VLSI*, 2017, pp. 173–178.
- [21] I. Tashdid, D. Saiham, N. Anjum, T. Farheen, and S. Rahman, "Ecologic: Enabling circular, obfuscated, and adaptive logic via efp-ga-augmented socs," *arXiv preprint arXiv:2508.04516*, 2025.
- [22] K. Shamsi and Y. Jin, "Circuit deobfuscation from power side-channels using pseudo-boolean sat," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9.
- [23] R. K. Datta, G. Zhao, D. Jain, and K. Shamsi, "On hardware trojan detection using oracle-guided circuit learning," in *Proceedings of the Great Lakes Symposium on VLSI 2024*, 2024, pp. 198–203.
- [24] D. Jain, S. Ahmed, G. Zhao, R. Datta, and K. Shamsi, "Trojan localization in generic ams circuits from combined power and functional queries," in *2025 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE Computer Society, 2025, pp. 239–249.
- [25] P. Subramanian, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *Proc. IEEE Int. Symp. on Hardware Oriented Security and Trust*. IEEE, 2015, pp. 137–143.
- [26] M. El Massad, S. Garg, and M. V. Tripunitara, "Integrated circuit (IC) decamouflaging: Reverse engineering camouflaged ICs within minutes," in *Network and Distributed System Security Symposium (NDSS)*, 2015.
- [27] B. Ghosh, D. Basu, and K. S. Meel, "Justicia: A stochastic sat approach to formally verify fairness," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 9, 2021, pp. 7554–7563.
- [28] N.-Z. Lee and J.-H. R. Jiang, "Towards formal evaluation and verification of probabilistic design," *IEEE Transactions on Computers*, vol. 67, no. 8, pp. 1202–1216, 2018.
- [29] M. L. Littman, S. M. Majercik, and T. Pitassi, "Stochastic boolean satisfiability," *Journal of Automated Reasoning*, vol. 27, no. 3, pp. 251–296, 2001.
- [30] R. Salmon and P. Poupart, "On the relationship between stochastic satisfiability and partially observable markov decision processes," in *Proceedings of the 35th Conference on Uncertainty in Artificial Intelligence (UAI)*, vol. 407, 2019, pp. 1–407.
- [31] N.-Z. Lee, Y.-S. Wang, and J.-H. R. Jiang, "Solving exist-random quantified stochastic boolean satisfiability via clause selection," in *IJCAI*, 2018, pp. 1339–1345.
- [32] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *Computers, IEEE Transactions on*, vol. 100, no. 8, pp. 677–691, 1986.
- [33] Y.-W. Fan and J.-H. R. Jiang, "Sharpssat: A witness-generating stochastic boolean satisfiability solver," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 4, 2023, pp. 3949–3958.
- [34] A. Mondal, M. Zuzak, and A. Srivastava, "Statsat: A boolean satisfiability based attack on logic-locked probabilistic circuits," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.
- [35] N. Limaye, E. Kalligeros, N. Karousos, I. G. Karybali, and O. Sinanoglu, "Thwarting all logic locking attacks: Dishonest oracle with truly random logic locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 9, pp. 1740–1753, 2020.
- [36] S. Ahmed, D. Jain, and K. Shamsi, "Improving error tolerance and scalability in pseudo-boolean sat-based generic side-channel analysis," in *2025 IEEE International Test Conference (ITC)*. IEEE, 2025, pp. 434–437.
- [37] S. Dasgupta and J. Langford, "A tutorial on active learning," in *Proc. Int. Conf. on Machine Learning*, 2009.
- [38] N. Hansen, A. Auger, R. Ros, S. Finck, and P. Pošík, "Comparing results of 31 algorithms from the black-box optimization benchmarking bbob-2009," in *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation*, 2010, pp. 1689–1696.
- [39] "Netlist encryption and obfuscation suite," [Online]. <http://www.bitbucket.com/kavehshm/neos>.
- [40] P.-W. Chen, Y.-C. Huang, and J.-H. R. Jiang, "A sharp leap from quantified boolean formula to stochastic boolean satisfiability solving," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 5, 2021, pp. 3697–3706.
- [41] B. L. Synthesis and V. Group, "ABC a system for sequential synthesis and verification," 2016, [Online] <http://people.eecs.berkeley.edu/~alanmi/abc/>.
- [42] F. Brglez, "A neutral netlist of 10 combinational benchmark circuits and a target simulator in fortran," in *International Symposium on Circuits and Systems*, 1985, 1985, pp. 695–698.
- [43] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *1989 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1989, pp. 1929–1934.