A Cognitive SAT to SAT-Hard Clause Translation-based Logic Obfuscation

Rakibul Hassan*, Gaurav Kohle†, Setareh Rafatirad†, Houman Homayoun†, and Sai Manoj Pudukotai Dinakarrao*
*Department of Electrical and Computer Engineering, George Mason University, Fairfax, VA, USA
†Department of Electrical and Computer Engineering, University of California Davis, Davis, CA, USA
*{rhassa2, spudukot}@gmu.edu; †{gkolhe, srafatir, hhomayoun}@ucdavis.edu

Abstract—**Logic obfuscation is introduced as a pivotal defense mechanism against emerging hardware threats on Integrated Circuits (ICs) such as reverse engineering (RE) and intellectual property (IP) theft. The effectiveness of logic obfuscation is challenged by recently introduced Boolean satisfiability (SAT) attack and it's variants. A plethora of counter measures have been proposed to thwart the SAT attacks. Irrespective of the implemented defenses, large power, performance and area (PPA) overheads are seen to be indispensable. In contrast, we propose a neural network-based cognitive SAT to SAT-hard clause translator under the constraints of minimal PPA overheads while preserving the original functionality with impenetrable security. Our proposed method is incubated with a SAT-hard clause generator that translates the existing conjunctive normal form (CNF) through minimal perturbations such as inclusion of pair of inverters or buffers or adding new lightweight SAT-hard block depending on the provided CNF. For efficient SAT-hard clause generation, the proposed method is equipped with a multi-layer neural network that first learns the dependencies of features (literals and clauses), followed by a long-short-term-memory (LSTM) network to validate and backpropagate the SAT-hardness for better learning and translation. For a fair comparison with the state-of-the-art, we evaluate our proposed technique on ISCAS'85 benchmarks. It is seen to successfully defend against multiple state-of-the-art SAT attacks devised for hardware RE. In addition, we also evaluate our proposed technique's empirical performance against MiniSAT, Lingeling and Glucose SAT solvers that form the base for numerous existing deobfuscation SAT attacks.**

I. Introduction

With the semiconductor industries inclining towards fabless business model i.e., outsourcing the fabrication to offshore foundries to cope-up with the operational and maintenance costs, the hardware security threats are exacerbating. These hardware threats could be in any form including intellectual property (IP) theft, hardware Trojan (HT) insertions, integrated circuit (IC) tampering, over production, and reverse engineering (RE). The threat could occur during any phase of the IC production cycle ranging from design phase, fabrication phase or even after releasing the design to the market (in the form of side-channel attacks).

Many hardware design-for-the-trust techniques have been proposed in the literature such as split manufacturing [1], IC camouflaging, and logic locking *a.k.a* logic obfuscation [2] to address the prevalent security treats. Among the aforementioned techniques, logic obfuscation technique can protect a design from majority of the attacks [3]. In logic obfuscation technique, the aim is to hide both the design functionality and the design structure from the attacker. To protect a design from being used as a black-box, the logic obfuscation technique inserts key-gates to hide the design functionality. The true functionality of the modified design will only be achieved when correct keys are provided.

Although the security of the IC/IP is enhanced via logic obfuscation schemes, the advent of Boolean satisfiability (SAT)-based attack [4], also known as "oracle-guided" threat model shows that by applying stimuli to the design and analyzing the output, the key and functionality of an IC/IP could be extracted in the order of a few minutes or less. To implement SAT attack, the attacker needs access to (a) an obfuscated netlist of IC (obtained after de-layering IC or constructed from layout), and (b) a functional/activated IC, to which the attacker applies the stimuli and monitor the output. The extracted netlist is converted into a conjunctive normal form (CNF[1]), fed to a SAT solver to determine the keys (assignment to each Boolean variable or literal in the CNF) to decrypt and reverse engineer (RE) the IC/IP.

To mitigate SAT attack-based threats, several logic obfuscation [3] techniques have been proposed such as SARLock [3], Anti-SAT [5], and SRCLock [6]. Recent literature also reports various post-SAT attacks such as removal attack [7], signal probability skew (SPS) attack [8], Bypass attack [9] against SAT attack defense methods. One of the major challenges in adopting the existing defenses including [3], [5] against SAT attacks is the imposed overheads in terms of power, performance and area (PPA) with no guarantee of security [10].

To address these concerns, we propose a PPA overhead-aware cognitive defense mechanism[2]. Our proposed solution encompasses a CNF generator that translates the provided SAT prone CNF into SAT-hard through minimal modification to the netlist such as flipping a literal (through addition of inverter gate or using XNOR instead of XOR are some of the naïve possibilities) in a clause of CNF. The conversion happens by learning the SAT and SAT-hard distributions along with preserving the functionality [11]. For this purpose, we employ a neural network with bipartite message passing mechanism to learn and determine the properties of a CNF and distinguish SAT and SAT-hard problems. The amount of clauses added or the perturbations introduced can be controlled to determine the trade-off between overheads and security.

[1]A CNF is a conjunction (i.e., AND) of one or more clauses, where a clause is a disjunction (i.e., OR) of literals.

[2]The source code is available at https://github.com/rakibhn/satconda

To the best of our knowledge, this is a novel IC/IP logic obfuscation solution that learns the SAT and SAT-hard distributions and encrypt the circuit with minimal PPA overheads. Two main contributions of this work are:

- The idea of exploring deep neural network for circuit obfuscation, especially for converting SAT problem to SAT-hard problem is unexplored and novel. Our defense mechanism against SAT attack utilized neural network model to learn the distinguishing features of SAT and SAT-hard problems.
- The proposed method seeds the learned parameter to the SAT-hard clause generator and then integrates that SAT-hard block to the original circuit in a way that the SAT attack fails to decrypt the keys used for the encryption.

We successfully defend the SAT attack [4] by introducing a SAT-hard block and integrating that block to the original circuit. In addition to the traditional SAT attack we evaluated our proposed scheme against some post-SAT attack deobfuscation mechanism such as AppSAT attack [12]. Using our method, we showcase that the existing obfuscation schemes can be made robust with minimal modifications. In addition to evaluating the SAT hardness, we have evaluated the PPA overheads incurred with additional security deployment through our model.

II. Background

Here, we present and introduce the concepts of logic obfuscation [3] and SAT attack [4] in brief.

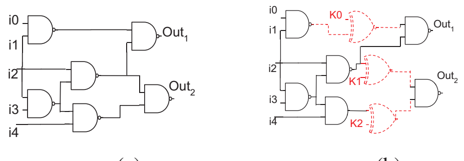
A. Logic Obfuscation

Logic obfuscation is implemented in a design by adding additional gates a.k.a "key-gates" to secure the circuit (IC/IP) by inducing randomness in the observable output(s). To achieve the desired output, all the key-gates must be correct.

Figure 1a depicts the original C17 benchmark circuit from ISCAS'85 benchmark [13] with corresponding logic obfuscated circuit shown in Figure 1b. The original circuit comprises of five inputs, with six NAND gates and two outputs. The obfuscated design has three additional gates, termed as key-gates. One of them is a XNOR gate and the other two are XOR gates. These additional gates are known as key-gates. For this obfuscated circuit in Figure 1b, if one assigns (k2, k1, K0) as (1, 0, 1), only then the circuit will function as desired i.e., as in Figure 1a. The complexity of determining the key inputs increase exponentially with the number of key-gates for brute-force approach.

B. SAT Attack

SAT attack generates carefully crafted input patterns and observes the corresponding output from the activated chip. The goal of SAT attack is to eliminate incorrect key-values in each iteration by observing the outputs for a given pair of inputs. To initiate the attack, the attacker first makes a copy (C_{obf}) of the obfuscated circuit then generates an input (i) that results in two different outputs for two different key values ($k1, k2$). This input is called Discriminating Input Pattern (DIP). A validation circuit confirms that for an input pattern, two different key values generate the same output (O_i). If the input-output



(a) (b)

Fig. 1: logic obfuscation on C17 benchmark circuit. (a) Original circuit design, (b) Encrypted circuit with additional key-gates (dotted lines/gates).

relation satisfies ($C_{obf}(i, k1) = O_i$ and $C_{obf}(i, k2) = O_i$) then the validation circuits are ANDed together to form a set of correct key validation circuit (CKVC). In each iteration, the SAT solver tries to find a new DIP and two key values that satisfy the above relation. This process continues until no more DIPs are found. The attacker eliminates numerous wrong keys from CKVC at each iteration and this step is iterated until it eliminates all the wrong keys and determines the correct key.

III. State-of-the-Art

Several techniques have been proposed to defend and secure the design from SAT attack. Here, we review some of the relevant defense techniques to thwart SAT attacks.

EPIC [14] is one of the preliminary works that inserts XOR/XNOR gates randomly as key-gates to the original netlist to achieve a obfuscated-netlist. However, one can decrypt the key-values by inspecting the XOR/XNOR gates and configuring them as buffers or inverters using the key-inputs [2].

Insertion of an additional circuit block, Anti-SAT block [5] along with the encrypted circuit is proposed to mitigate the SAT attack. This work shows that the time required to expose all the key-values is an exponential function of the key gates in the Anti-SAT block. By making the key-size large enough, the SAT attack becomes computationally complex and infeasible.

A similar work, SARLock has been reported in [3]. In this work the authors propose a SARLock block with the encrypted circuit that maximizes the number of DIPs, thus making the SAT attack runtime exponential with the number of secret-key bits. However, this technique was shown to be vulnerable to double-DIP attack reported in [15].

In [16] a SAT-resilient cyclic obfuscated circuit design was proposed by adding dummy paths to the encrypted circuit which make the combinational loop non-reducible. This defense mechanism was prone to another type of SAT-based attack named CycSAT [17] which can effectively decrypt the cyclic encryption.

Similar to the above discussed works, this work also presents a technique to insert and integrate an additional block into the design to enhance the security against SAT attacks. In contrast to the existing defenses discussed above, our proposed methodology utilizes a cognitive approach by utilizing neural network and extracts the feature variables automatically to learn the SAT and SAT-hard distributions. These distributions will be further utilized to translate a CNF from SAT to SAT-hard by adding additional circuitry with least overhead compared to existing defenses. Finally, that SAT-hard block will be tightly integrated with the previously SAT attack prone obfuscated circuit to further strengthen the obfuscation.

IV. Proposed Defense with SAT to SAT-Hard Translator

We present our proposed cognitive defense methodology against SAT attacks for RE in this section. The proposed technique generates a SAT-hard block and encrypts the original obfuscated circuit with that block leading to obfuscated circuit with an enhanced security. In order to cognitively generate the SAT-hard block, our proposed technique is equipped with a hybrid Message Passing Neural Network (MPNN) [18] framework that learns the SAT and SAT-hard distribution from a given number of SAT problems. For this purpose, we utilize a message passing neural network (MPNN) similar to [19] in this work. The MPNN is widely studied in recent times for generic SAT problems and is yet to be explored in the context of hardware security. The motivation for using a neural network is its ability to learn a SAT distribution and a SAT-hard distribution from the given samples. The best fit MPNN model shows the high accuracy for distinguishing a SAT problem from a SAT-hard one [19]. There are two advantages of deploying a neural network in logic locking regime. First, with the increased computation power and advancements in the SAT algorithms, the state-of-the-art SAT attacks are very advanced and powerful. Secondly, to protect the existing locking techniques from the powerful SAT-attacks, a cognitive counter measure is effective. Figure 2 depicts the operational workflow of the proposed framework and integration of the SAT-hard block with the obfuscated circuit, respectively. The details are presented below.

A. Learning the SAT and SAT-Hard Distributions

In this work, we trained and tested our neural network model offline and pass the best-fit model parameters i.e., $seed_1$ and $seed_2$ to the SAT-hard clause generator. The deployed MPNN encompasses of three layer fully-connected layers followed by a two-layer long short term memory (LSTM) network. The literal vector (L_{init}) and clause vector (C_{init}) are fed to a three-layer fully connected MPNN. The output from the MPNN, ($L_{msg}, C_{msg}, L_{sat}$), are fed to a two-layer long-short term memory (LSTM) (C_u, L_u) network. In order to learn the SAT and SAT-hard distributions, we encode the IC obfuscation problem as a SAT problem and then represent that problem as a directed graph, where each clause and each literal is represented as a node individually whose dependencies will be learned through message passing. The relationship among the clauses and literals are established using edges. Hidden states for literals and clauses are denoted by L_h and C_h, respectively. An adjacency Matrix (M) defines the relationship between literals and clauses. This relationship between literals and clauses are established by connecting edges among them.

Message is passed back and forth along the edges of the network [18]. The message passing starts by passing a message to a clause from its neighbouring literals. In the next step, a literal gets message from its neighbouring clause(s) and also from its complements. This message passing event occurs back and forth until the model refines a vector space for every node. The LSTM network updates the literals L^{t+1} and clauses C^{t+1} at each iteration, as follows:

$$C_u([M^T L_{msg}(L^t)]) \to C^{t+1} \tag{1}$$
$$C_u([C_h^t]) \to C_h^{t+1} \tag{2}$$
$$L_u([Flip(L^t), MC_{msg}(C^{t+1})]) \to L^{t+1} \tag{3}$$
$$L_u([L_h^t]) \to L_h^{t+1} \tag{4}$$

L_{sat} predicts SAT or SAT-hard for a particular literal and taking the average prediction of all the literals after T iteration, the proposed model predicts whether a problem is SAT or SAT-hard. This message-passing architecture lets the the proposed model to learn the features that can distinguish the SAT solvable CNFs from SAT-hard CNFs i.e., learn the distribution of SAT and SAT-hard CNFs.

B. SAT to SAT-Hard Translator

To achieve a nearly unbreakable[3] obfuscated circuit i.e., SAT-hard, the proposed technique generates a SAT-hard block and integrates or modifies the existing CNF to make the whole design SAT-hard. As random generation of SAT-hard clauses is not efficient, this work proposes a cognitive way of generating the SAT-hard clauses by learning the underlying CNF patterns. This is performed with the aid of a neural network from which the seeds for clause generation are extracted.

The SAT-hard block generation process begins with obtaining the seed values and passing them to $seed_1$ and $seed_2$ variables (Algorithm 1 line 1). The $seed_1$ and $seed_2$ values are determined as the values that best fit the SAT-hard distribution. A randomly generated decimal number between 0 and 1 is then compared with the $seed_1$ value. Another variable $Literal\_base$ is assigned as integer 1 if the decimal number is greater than the $seed_1$ otherwise integer 2. Then, the SAT-hard clause generator draws samples from a geometric distribution at a probability of $seed_2$ and assigns that value to a variable ($Rand\_geo$). Adding variables $Literal\_base$ and $Rand\_geo$ we have the length of a new clause (how many literals in the clause). Then, the clause generator samples a variable from the original CNF variable list without replacement (Algorithm 1 line 26). The clause generator samples the variable with a 50% probability of taking that variable or negating that variable. Appending the new clause with the previous clauses the generator again checks for the satisfiability (Algorithm 1 line 16) and keeps adding clauses until the new obfuscated CNF becomes SAT-hard (Algorithm 1 line 18). Algorithm 1 presents the aforementioned clause generation steps.

The significance of $seed_1$ and $seed_2$ is that they determine the achievable PPA overheads. The values of $seed_1$ and $seed_2$ are determined by fitting the learned MPNN data distribution to Bernoulli and Geometric distributions, respectively. If the learned distribution fits well to the Bernoulli and Geometric distribution then the value of $seed_1$ and $seed_2$ approaches to 1. Thus the PPA overhead is low. On the other hand if the learned distribution does not fit the Bernoulli and Geometric distribution well then the value of $seed_1$ and $seed_2$ is close to 0. This leads to a high PPA overhead.

[3]Here, the unbreakable refers to unbreakable with SAT solvers within a limited period of time i.e., period of 10 hours in this work.

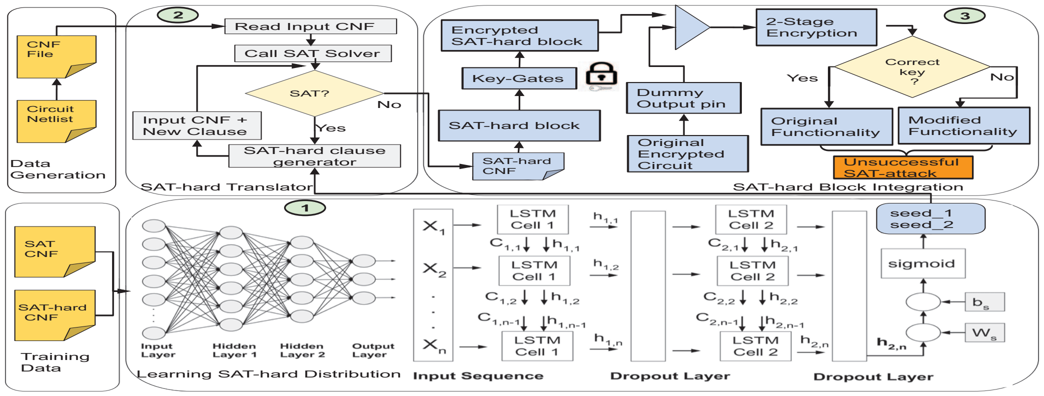


Fig. 2: Workflow of proposed cognitive SAT to SAT-hard clause translator. (1) Learning the SAT and SAT-hard distributions, (2) Translating a SAT CNF to a SAT-hard CNF, (3) Integrating the SAT-hard block with the original encrypted circuit

C. SAT-Hard Block Integration

Though the generated SAT-hard block can be effective against traditional SAT attacks, one can argue that it can be vulnerable to additional attacks such as partitioning-based removal attacks [7]. To address such vulnerabilities, we integrate the generated SAT-hard block effectively with the original circuit. Here, our main objective is to hide one of the original output pins from the attacker and integrate that hidden output pin with the SAT-hard block thereby thwarting such attacks. Finally, with proper key value, the original functionality of the hidden output pin is retrieved.

To achieve this, the proposed technique adds a dummy output pin by replacing one of the original output pin, as shown in Figure 2 (top-right block). The rationale behind introducing a dummy output pin is that the SAT attack algorithm matches the output with the activated chip and when it fails to get the value for one output pin then the algorithm fails to decode the keys. The proposed method generates a lightweight SAT-hard block and encrypts that block with a key-gate (AND gate). This key-gate and the dummy output (which has the original circuit's output) are XOR'ed together. The output of this XOR gate is the final output pin that was replaced with the dummy output pin. Now the added output pin is encrypted with an SAT-hard block and a key-gate. This is a two-step encryption. When the SAT attack algorithm tries to figure out the output of this encrypted pin it fails to do so as the SAT-solver could not solve that SAT-hard block. Thus, the SAT attack algorithm fails to decrypt the value of the key-gates. The functionality of the SAT-hard block is hidden and also encrypted with a key-gate, only correct key will expose this block. So attacks such as the partitioning-based removal attack [7] will not succeed to identify the SAT-hard block.

D. Training and Testing The MPNN Model

To train the neural network (NN) for learning SAT and SAT-hard distributions we provide the NN a pair of SAT problems where one problem is satisfiable (SAT) and the other one is SAT-hard. In the training pool, we have a total of C_m SAT problems and C_n SAT-hard problems where $C_m = C_1, C_2, \cdots, C_m$ and $C_n = C'_1, C'_2, \cdots, C_n$. Each of the SAT and SAT-hard problems has k number of clauses with each clause having n variables (ranges from 5 to 20). In some cases, the difference between SAT and SAT-hard samples could be a simple flipping of a literal in a clause.

Each of the training data is labeled either 0 or 1, where 0 means SAT-hard and 1 stands for satisfiable (SAT). For a $\{SAT, SAT-hard\}$ pair, the fully connected MLP tries to update the weights for each neuron and learn the relationship between the feature vectors, i.e., clauses and literals. The LSTM network is then updated with the predicted output from the MLP and continue to iterate the process. After a number of iterations the network predicts whether a problem is SAT or SAT-hard. We trained our MPNN model on ten thousand SAT problems (out of which five thousand are SAT-hard) for better learning and efficient modeling of SAT and SAT-hard distributions. The model that achieves the highest accuracy is chosen to generate the SAT-hard block in this work. For the best-fit model, the training-cost is 0.6930 and the validation cost is 0.6932, which is sufficiently good enough to ensure that the model generalizes well with no over or under-fitting.

V. Experimental Results

In this section, we describe the experimental setup and evaluate the impact of the proposed technique and compare with existing works in terms of SAT-hardness and the incurred overheads.

A. Experimental Setup

In this work, we used the MPNN model to obtain the seed required for our SAT-hard clause generator. We trained the model with 10,000 CNFs, out of which 5000 are SAT and 5000 are SAT-hard. We evaluate the performance on ISCAS'85 and ISCAS'89 benchmark circuits. We chose this benchmark for the purpose of comparison, as they are the ones widely used in the literature. We used two different obfuscation schemes to be integrated with our proposed model to make them SAT attack resilient. One algorithm was proposed by Rajendran et al. [20] that inserts XOR gates ("TOC'13") at different locations of the circuit to maximise the hamming distance between correct and incorrect outputs. The second algorithm was proposed by Dupuis et al. [21] that inserts AND/OR gates ("IOLTS'14")

Algorithm 1 Proposed SAT to SAT-hard Translator

```
**Input** : $solve\_clauses, seed\_1, seed\_2$
**Output** : $SAT - hard - cnf$
$is\_sat := solve(solve\_clauses)$
**if** $is\_sat == True$ **then**
    **while** true **do**
        $rand := gen\_decimal(0 - 1)$
        **if** $rand < seed_1$ **then**
            $lit\_base := 1$
        **else**
            $lit\_base := 2$
        **end if**
        $rand\_geo = rand.geometric(seed\_2)$
        $literal = lit\_base + rand\_geo$
        $new\_clause := generate\_clause(n\_var, literal)$
        $solve\_clauses+ = new\_clause$
        $is\_sat := solve(solve\_clauses)$
        **if** $is\_sat == True$ **then**
            $solve\_clauses+ = new\_clause$
        **else**
            $break$
        **end if**
    **end while**
    $solve\_clauses+ = new\_clause$
**end if**
**function** GENERATE_CLAUSE($n\_var$, $literal$)
    $array\_size := minimum(n\_var, literal)$
    $clause\_gen := gen.rand\_array(n\_var, array\_size)$
    $rand := gen\_decimal(0\ 1)$
    **if** $rand < 0.5$ **then**
        $new\_clause := clause\_gen + 1$
    **else**
        $new\_clause := -(clause\_gen + 1)$
    **end if**
    **return** $new\_clause$
**end function**
$SAT - hard - cnf := solve\_clauses$
```

at different carefully chosen locations of the circuit. While generating the key-bits for XOR-based logic locking [20] and AND/OR-based logic locking [21] we considered 5% area overhead.Thus, depending on the original circuit area, the number of key bits vary. The number of key bits ranges from 6 (for the smallest circuit) to 186 (for the largest circuit). On top of the obfuscated benchmark circuit the proposed method needs one more key bit to integrate the SAT-hard block to the original obfuscated circuit. For our best-trained model, the number of iterations for the algorithm to converge ranges from 60 iterations (for the smallest circuits) to 1005 iterations. We have also verified the satisfiability of CNF using three different SAT solvers, MiniSAT [22], Lingeling [23], and Glucose [24]. The rationale for choosing these solvers is that these solvers form basis for numerous SAT attacks crafted for deobfuscation in the past few years. The PPA overhead is calculated using Synopsys Design Compiler, Version: L-2016.03-SP3. SAED 90nm EDK Digital Standard Cell Library is used for logic synthesis. All the experiments were performed on a server with 8-core Intel Xeon E5410 CPU, running CentOS Linux 7 at 2.33 GHz, with 16 GB RAM.

B. Evaluation

We evaluate the SAT-hardness and the overhead analysis of our proposed method and compare them with other works.

1) SAT-hardness: Table I shows SAT attack's [4] performance on ISCAS'85 and ISCAS'89 benchmarks (we have only shown few circuits as a representative set, though all circuits have showcased similar behavior) with two different encryption algorithms [20], [21] before and after they are passed through the proposed model. It shows that the SAT attack in [4] successfully decrypts the keys of the obfuscated circuit (obfuscated by [20], [21]) in few seconds. The SAT attack fails to extract the keys when it is further encrypted using our model. For both the IOLTS'14 [21] and TOC'13 [20] encryption, the proposed technique successfully defends the SAT attack for all the circuits. To ensure the resiliency of the proposed method against post-SAT attack we also evaluated our method against recently introduced AppSAT attack [12]. As shown in Table I, our method remains resilient even against this attack within the timeout period which is set to 10 hours for this experiments.

Table II presents the satisfiability of the ISCAS'85 and ISCAS'89 benchmark circuits before and after the conversion through the proposed technique against traditional SAT solvers. It can be seen that all the encrypted benchmark circuits using IOLTS'14 [21] (ToC'13 [20] results not shown for conciseness) encryption were satisfiable (breakable) with all the three SAT-solvers [22]–[24]. This indicates that an attacker could perform a SAT attack with any of these SAT-solvers and reverse engineer the IP/IC. Table II also shows that once the IC/IP design is translated using our proposed model, it becomes SAT-hard, meaning none of the three experimented SAT-solvers, which were previously successful, could solve a satisfying assignment. This proves that the our model can effectively translate a SAT into a SAT-hard problem.

TABLE I: Evaluation of the proposed technique on various SAT attacks

Circuit Name	SAT attack [4]				AppSAT attack [12]			
	IOLTS'14 [21]		ToC'13 [20]		IOLTS'14 [21]		ToC'13 [20]	
	Time (s)		Time (s)		Time (s)		Time (s)	
	Before Conversion	After Conversion	Before Conversion	After Conversion	Before Conversion	After Conversion	Before Conversion	After Conversion
c880	0.061	timeup	0.080	timeup	0.1598	timeup	0.20	timeup
c1908	0.049	timeup	3.694	timeup	0.122	timeup	3.945	timeup
c3540	0.323	timeup	1.048	timeup	1.968	timeup	4.921	timeup
c5315	0.826	timeup	0.786	timeup	2.888	timeup	1.71	timeup
s526n	0.0087	timeup	0.008	timeup	0.089	timeup	0.044	timeup
s526	0.0084	timeup	0.008	timeup	0.090	timeup	0.045	timeup
s953	0.017	timeup	0.017	timeup	0.157	timeup	0.077	timeup
s5378	0.070	timeup	0.072	timeup	0.541	timeup	0.268	timeup
s35932	5.831	timeup	5.890	timeup	17.882	timeup	8.138	timeup

timeup = 10 hours

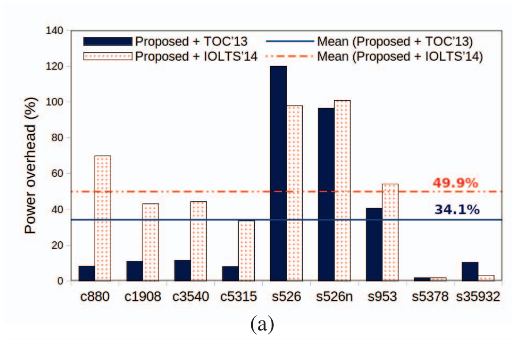
2) Overhead Analysis: In addition to SAT-hardness, we evaluate the imposed overheads through the conversion. Figure 3 reports the area and power overhead of the IOLTS'14 [21] and TOC'13 [20] encryption and compare them with the overhead of our proposed model (that successfully defends SAT attacks).

The IOLTS'14 [21] and TOC'13 [20] obfuscation schemes introduce an area overhead of 5% (on an average) when compared to the original circuit for the ISCAS'85 and ISCAS'89 benchmarks. However, as seen earlier despite incurring overheads, the SAT attacks [4] can reverse engineer the IP.

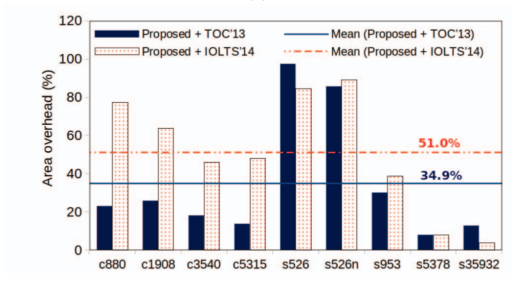
TABLE II: Evaluation of the proposed technique using IOLTS'14 encryption on different SAT-solvers

Circuit Name	miniSAT [22]		Lingeling [23]		Glucose [24]	
	Time(s)		Time(s)		Time(s)	
	Before Conversion	After Conversion	Before Conversion	After Conversion	Before Conversion	After Conversion
c880	0.0024	✗	0.1	✗	0.0014	✗
c1908	0.0041	✗	0.1	✗	0.0031	✗
c3540	0.0060	✗	0.1	✗	0.0061	✗
c5315	0.0086	✗	0.1	✗	0.0094	✗
s526n	0.0015	✗	0.2	✗	0.0045	✗
s526	0.0061	✗	0.2	✗	0.0035	✗
s953	0.0018	✗	0.2	✗	0.0012	✗
s5378	0.0034	✗	0.2	✗	0.0076	✗
s35932	0.0155	✗	0.2	✗	0.0202	✗

✗= Corresponding CNF was SAT-hard, not solved by the SAT solver before timeout



(a)



(b)

Fig. 3: Overhead analysis of the proposed technique with different encryption schemes: (a) Power overhead, (b) Area overhead.

Further encrypting with the proposed technique incurs additional area overhead of about 50% and about 34% on average compared to [21] and [20] respectively i.e., around 7.5% and 6.7% compared to original circuit, but guarantees security. In a similar manner, proposed technique incurs around 49% and 35% power overhead on an average compared to [21] and [20], respectively i.e., around 7.45% and 6.75% compared to original circuit. Thus, if a lightweight encryption which is moderate in terms of security exist, it can be made secure with the proposed technique with minimal overheads.

Based on the evaluations performed, we confirm that the proposed technique performs efficient translation of SAT to SAT-hard for ISCAS'85 and ISCAS'89 benchmarks with lower overhead, power consumption without deviating from the original functionality.

VI. Conclusion and Future Work

In this work we successfully defend the Boolean satisfiability (SAT) attacks against reverse engineering of hardware ICs/IPs by introducing a neural network based SAT-hard problem generator and achieve an enhanced obfuscation technique for hardware security regime. We observed that the integration of an SAT-hard block with a dummy output pin replacing an original output pin (keeping the same functionality) deceives the SAT attack. Our framework is evaluated on the state-of-the-art benchmarks such as ISCAS'85 and ISCAS'89 using two state-of-the-art encryption algorithms. We validate our model with the SAT attack and its variants and three other traditional SAT solvers.

References

[1] J. J. Rajendran *et al.*, "Is split manufacturing secure?" in *Conf. on Design, Automation and Test in Europe*, 2013.
[2] M. Yasin *et al.*, "On improving the security of logic locking," *IEEE Tran. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 9, pp. 1411–1424, 2015.
[3] M. Yasin *et al.*, "SARLock: SAT attack resistant logic locking," in *Int. Symp. on Hardware Oriented Security and Trust*, 2016.
[4] P. Subramanyan *et al.*, "Evaluating the security of logic encryption algorithms," in *Int. Symp. on Hardware Oriented Security and Trust*, 2015.
[5] Y. Xie and A. Srivastava, "Mitigating sat attack on logic locking," in *Int. Conf. on Cryptographic Hardware and Embedded Systems*, 2016.
[6] S. Roshanisefat *et al.*, "Srclock: Sat-resistant cyclic logic locking for protecting the hardware," in *Proceedings of the 2018 on Great Lakes Symp. on VLSI*.
[7] M. Yasin *et al.*, "Removal attacks on logic locking and camouflaging techniques," *IEEE Trans. on Emerging Topics in Computing*, 2017.
[8] M. Yasin *et al.*, "Security analysis of anti-sat," in *Asia and South Pacific Design Automation conf.*, 2017.
[9] X. Xu *et al.*, "Novel bypass attack and bdd-based tradeoff analysis against all known logic locking attacks," in *Int. conference on cryptographic hardware and embedded systems*, 2017.
[10] G. Kolhe *et al.*, "Security and complexity analysis of lut-based obfuscation: From blueprint to reality," in *Int. Conf. on Computer-Aided Design*. IEEE, 2019.
[11] R. Hassan *et al.*, "Satconda: Sat to sat-hard clause translator," in *21st Int. Symp. on Quality Electronic Design*, 2020.
[12] K. Shamsi *et al.*, "Appsat: Approximately deobfuscating integrated circuits," in *2017 IEEE Int. Symp. on Hardware Oriented Security and Trust*. IEEE, 2017.
[13] M. C. Hansen *et al.*, "Unveiling the iscas-85 benchmarks: A case study in reverse engineering," *IEEE Design & Test of Computers*, vol. 16, no. 3, pp. 72–80, 1999.
[14] J. A. Roy *et al.*, "Ending piracy of integrated circuits," *Computer*, vol. 43, no. 10, pp. 30–38, 2010.
[15] Y. Shen and H. Zhou, "Double dip: Re-evaluating security of logic encryption algorithms," in *Great Lakes Symp. on VLSI*, 2017.
[16] K. Shamsi *et al.*, "Cyclic obfuscation for creating sat-unresolvable circuits," in *Great Lakes Symp. on VLSI*, 2017.
[17] H. Zhou *et al.*, "Cycsat: Sat-based attack on cyclic logic encryptions," in *36th Int. Conf. on Computer-Aided Design*, 2017.
[18] J. Gilmer *et al.*, "Neural message passing for quantum chemistry," in *Int. conf. on Machine Learning*, 2017.
[19] D. Selsam *et al.*, "Learning a SAT solver from single-bit supervision," *arXiv preprint arXiv:1802.03685*, 2018.
[20] J. Rajendran *et al.*, "Fault analysis-based logic encryption," *IEEE Transactions on computers*, vol. 64, no. 2, pp. 410–424, 2013.
[21] S. Dupuis *et al.*, "A novel hardware logic encryption technique for thwarting illegal overproduction and hardware trojans," in *IEEE Int. On-Line Testing Symp.*, 2014.
[22] N. Sorensson and N. Een, "Minisat v1. 13-a sat solver with conflict-clause minimization," *SAT*, vol. 2005, no. 53, pp. 1–2, 2005.
[23] A. Biere, "Lingeling, plingeling and treengeling entering the sat competition," 2013.
[24] G. Audemard and L. Simon, "GLUCOSE: a solver that predicts learnt clauses quality," *SAT Competition*, 2009.