# A Neural Network-based Cognitive Obfuscation Towards Enhanced Logic Locking

Rakibul Hassan, *Student Member, IEEE*, Gaurav Kolhe, *Student Member, IEEE*, Setareh Rafatirad, *Senior Member, IEEE*, Houman Homayoun, *Senior Member, IEEE* and Sai Manoj Pudukotai Dinakarrao, *Member, IEEE*

***Abstract*—Logic obfuscation is introduced as a pivotal defense against multiple hardware threats on Integrated Circuits (ICs) including 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 also been proposed to thwart the SAT attack. Irrespective of the implemented defense against SAT attacks, large power, performance and area overheads are seen to be indispensable. In contrast, we propose a cognitive solution which is a neural network based SAT-hard clause translator, SATConda, that incurs a minimal area and power overhead while preserving the original functionality with enhanced security. SATConda 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, SATConda 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. Our proposed SATConda is evaluated on ISCAS'85 and ISCAS'89 benchmarks and is seen to successfully defend against multiple state-of-the-art SAT attacks devised for hardware RE. In addition, we also evaluate our proposed SATConda's empirical performance against MiniSAT, Lingeling and Glucose SAT solvers that form the base for numerous existing deobfuscation SAT attacks.**

***Index Terms*—logic locking, message passing neural network, SAT, SAT-hard.**

## 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, hardware security threats are exacerbating. This hardware threat could be in any form including intellectual property (IP) theft, integrated circuit (IC) tampering, over production and cloning [1], [2]. What is worse, 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 form of side-channel attacks) .

To thwart the prevalent security threats, many hardware design-for-trust techniques have been introduced such as split manufacturing [3], IC camouflaging, and logic locking *a.k.a* logic obfuscation [4]. Among multiple aforementioned techniques, logic locking can thwart the majority of the attacks at various phases in the IC Production chain [5]. This is because logic locking requires the correct keys to unlock the true functionality of the design. Additionally, as a part of the post-manufacturing process, the activation of IC (i.e., providing correct keys) will be accomplished in a trusted regime to hide the functionality from the untrusted foundry and other attacks. Having key-programmable gates allows the designer or user to control the functionality using these key inputs.

Although logic locking schemes enhance the security of the IP, the advent of Boolean satisfiability (SAT) based attack [6], also known as "oracle-guided" threat model shows that by applying stimuli to the design and analyzing the output, the key value and functionality of an IC could be extracted in the order of a few minutes or less [7]. 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 can apply 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. It has been seen that modern SAT solvers can solve a SAT-problem with up to million variables [8].

To mitigate SAT attack several logic locking [5] techniques have been proposed. A recently proposed mechanism on logic locking was presented to mitigate SAT attack by introducing an additional logic block that makes SAT attack computationally infeasible [9]. Recent literature reported signal probability skew (SPS) attack [10] against Anti-SAT defense [9] which can break the Anti-SAT defense within few minutes.

One of the major challenges in adopting the existing defenses against SAT attacks or its variants is the imposed overheads in terms of area and power with no guarantee of security [11]. Previous works [9], [10] consider developing Anti-SAT solutions through embedding different metrics (properties of netlist that cannot be translated into CNF) or through heuristic intuitions. Such defenses involve challenges including complexity, incompleteness and high probability to

R. Hassan, and S. M. P. Dinakarrao are associated with Department of Electrical and Computer Engineering, George Mason University, Fairfax, VA 22030, USA. Email: {rhassa2,spudukot}@gmu.edu

G. Kolhe, S. Rafatirad, and H. Homayoun are associated with Department of Electrical and Computer Engineering, University of California, Davis, Davis, CA 92617, USA. Email: {gkolhe,srafatir,hhomayoun}@ucdavis.edu

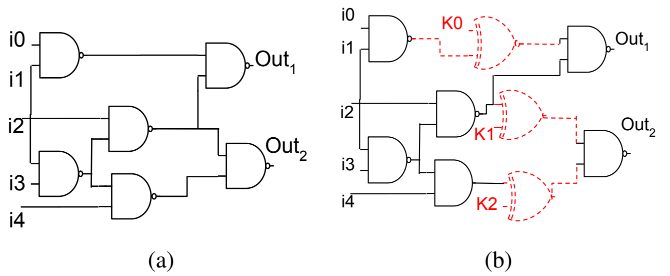
A preliminary version of this work is accepted in ISQED 2020 for publication.

[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.

exclude parameters that were not explored in literature. To address these concerns, we introduce SATConda[2], equipped with a CNF generator that can convert the provided SAT prone CNF into SAT-hard [3] through minimal modification to the netlist such as flipping a literal by adding one or few gates (through addition of inverter gate or using XNOR instead of XOR are some of the naïve possibilities) in a clause of CNF i. e., converts the SAT distribution to SAT-hard distribution by learning the distributions yet preserving the functionality. To perform such modifications, we deploy neural network with bipartite message passing mechanism to cognitively learn and determine the properties of a CNF and distinguish SAT and SAT-hard problems. Once learnt, the amount of clauses added or the perturbations are introduced cognitively, which can be controlled to determine the trade-off between overheads and security. From the ISQED accepted work, we have extended our work to a greater extent. We have added the following extension to this TCAD submission. We used the ISCAS'85 and ISCAS'89 benchmark circuits in the previous version without introducing any key-gates for logic locking. Here we introduce key-gates for logic-locking which is the key idea behind logic locking. The previous work only adds an SAT-hard block to a circuit (which does not have any key-gates) in order to prevent the SAT attack. In this work, we used two state-of-the-art encryption techniques to encrypt the original circuit. Though this encryption remains vulnerable to the recently proposed SAT attack, we successfully protect those encrypted circuits from the SAT attack. We performed area, power and delay overhead analysis for our present work while we only performed area and power overhead analysis for our previous work. We extensively evaluate our proposed model with the state-of-the-art SAT attacks. We have added case studies for our machine learning model and presented our best model's results. The two main contributions of this work are:

- SATConda induces additional clauses or flips the existing clauses to make the CNF (obfuscated IC) SAT hard. To perform such an operation cognitively, SATConda utilizes a neural network model to learn the distinguishing features of SAT and SAT-hard CNFs. SATConda seeds learned parameter to the 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 standard benchmarks against existing attacks such as SAT-attack[6] by introducing an SAT-hard block and encrypting that block the the original circuit. Using SATConda, we showcase the existing obfuscation schemes can be made robust with minimal modifications.

To the best of our knowledge the proposed technique is a novel defense mechanism against SAT-attack utilized neural network model to learn the SAT and SAT-hard clauses and encrypt the circuit under minimal overheads constraint. The idea of exploring deep neural network in circuit obfuscation, especially for converting SAT to SAT-hard is unexplored and is novel [14]. We also evaluate the translated obfuscated circuit with three different state-of-the-art SAT solvers. In addition to evaluating the SAT hardness, we have evaluated the area and power overheads incurred with additional security deployment through SATConda.



(a) (b)

Fig. 1: Logic locking on c17 benchmark circuit. (a) Original Circuit Design, (b) Encrypted circuit with additional key-gates (dotted lines/gates) [16]. Desired circuit behavior is achieved when (k2, k1, k0) = 101

## II. Background

Here, we discuss the basic information regarding the logic locking and the SAT attack.

### A. Logic Locking

Logic locking mechanism is implemented in a design by adding additional gates a.k.a "key-gates" to secure the circuit (IC/IP) by inducing the randomness in the observable output [4]. To achieve the desired output from the design, all the key-gates must be set to their proper input. Any incorrect insertion to any of the key-gates leads to the incorrect output. Thus, an attacker needs to know the correct assignment to those keys-gates to decode the actual functionality of the design.

Figure 1a depicts the original circuit and the Figure 1b shows a logic locked circuit of the same C17 circuit from ISCAS'85 benchmark [15]. The original circuit consists of five inputs with six NAND gates and two outputs. The encryption is done by adding three additional gates, termed as key-gates. For Figure 1b, if one assigns (k2, k1, K0) as (1, 0, 1), only then the circuit will function as it is intended to. Complexity of determining the key inputs will increase exponentially with the number of key-gates when attacker performs brute-force search.

### B. SAT Attack

Despite logic locking being successful in securing the IC from reverse engineering, the Boolean Satisfiability-based attack commonly known as SAT-attack [6] proposed in 2015 has successfully broke six state-of-the-art logic-locking defense mechanism proposed. The results have shown that the circuits can be successfully deobfuscated despite deploying logic locking solutions within few seconds. To mitigate this SAT-based attack(s), researchers have proposed several counter-measures

[2] SATConda is our proposed defense mechanism to fight against the SAT-attack. The source code is available at https://github.com/rakibhn/satconda

[3] SAT-hardness [12], [13] for a given CNF is the satisfiability measure whether a state-of-the-art SAT solver can achieve a solution within a given time limit. For this work, the time limit for the SAT solver is set to 10 hours.

from time to time and new attack model(s) has also been proposed to counterfeit that defense. We present a glimpse of SAT-attack methodology here:

*1) Attack Model:* The attack model was established under the assumption that the attacker has

- A gate-level netlist extracted from the obfuscated IC.
- An activated functional chip for observing the output pattern for a given input.

*2) Attack Methodology:* SAT attack generates a carefully crafted input patterns and observed the corresponding output from the activated functional chip. The goal of SAT attack is to eliminate incorrect key-values at each iteration by observing the outputs for a given pair of inputs. This input/output pairs are called Distinguishing Input Patterns (DIP). By observing this DIP, SAT-attack iteratively eliminates numerous wrong keys and this step is iterated until it eliminates all the wrong keys and determines the correct key [6].

## III. Related Work

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

EPIC [17] is one of the preliminary notable work that proposed inserting XOR/XNOR gates randomly as key-gates to the original netlist to achieve a logic locked netlist. One might be able to decrypt the key-values by inspecting the XOR/XNOR gates and configuring them as buffers or inverters using the key-inputs [4].

Insertion of an additional circuit block, Anti-SAT block [9] was proposed to add with the encrypted circuit to mitigate the SAT attack. The authors showed 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.

Similar work has been reported in SARLock [5]. In this work they proposed 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. This method was shown to be vulnerable to a SAT-based attack, double-DIP attack reported in [18].

In another work, advanced encryption standard (AES) circuit [4] was proposed into an encrypted circuit to prevent the SAT-attack. By adding this AES circuit, [4] makes the attack computationally intractable as the attacker cannot retrieve the input of the AES block by observing the output patterns. However, this techniques suffers from large area overhead since implementation of AES circuit requires for significant number of logic gates [9].

In [19] 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 [20] which can effectively decrypt the cyclic encryption.

The CycSAT attack [20] performs a pre-processing step so that it generates a cycle avoidance clause list. The more clauses that is responsible for the SAT solver to be trapped in an infinite loop the more pre-processing is done by the CycSAT. In [21] a number on methods are proposed to increase the cycles in a netlist exponentially to defense against the CycSAT attack. They proposed a technique to insert ‘Super Cycles’ block that causes an exponential increase in the cycles in a given netlist and leads to a exponential increase in the runtime of the CycSAT pre-processing step. As a result, the SAT solver fails to solve the netlist within a feasible amount of time.

Another defense mechanism called delay locking introduces functional key gates as well as delay key gates to a netlist. In this technique the original circuit functionality and the pre-defined timing constraints are locked using key-gates and delay-gates, respectievely [22]. The original functionality of the netlist is dependent on the correct key inputs and the correct timing constraints are satisfied when proper delay-keys are provided. An in-depth review of SAT attacks and defenses is presented in [23].

Unlike 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 yet providing security compared to existing defenses. Finally, the introduced SAT-hard block will be encrypted with the previously SAT-attack prone obfuscated circuit to further strengthen the obfuscation and enhance resilience against other kinds of attacks.

## IV. SATConda: SAT to SAT-hard Translator

Our proposed defense methodology against SAT attacks is presented here. SATConda generates an SAT-hard block and encrypts the original obfuscated circuit with that block and outputs a stronger obfuscated circuit. In order to cognitively generate the SAT-hard block, SATConda is equipped with a hybrid Message Passing Neural Network (MPNN) [24] framework that learns the SAT and SAT-hard distribution from a given number of SAT problems. Figure 2 depicts the operational flow of the proposed framework and integration of the SAT-hard block with the obfuscated circuit, respectively. We can divide the whole design flow into three different blocks. In Fig. 2, block 1 represents the training phase where we train our model with CNF files. So, block 1 takes inputs as SAT or SAT-Hard CNF as the training data. After the neural network being trained and evaluated, the model outputs the value of $seed_1$ and $seed_2$ for each benchmark circuit while trained on the SAT and SAT-hard problems for different encryption circuits as input. Here, the neural network output is the continuous value prediction from 0 to 1 for both the $seed_1$ and $seed_2$.The predicted value of $seed_1$ and $seed_2$ thus set the initialization for the SAT-hard block generator. For both the $seed_1$ and $seed_2$ values, our neural network predicts a value ranges from 0 to 1 for a given circuit. The significance of $seed_1$ and $seed_2$ will be discussed in detail in section IV-C. We represent the SAT-hard clause translator in block 2 that takes these learned parameters as input and outputs a set of

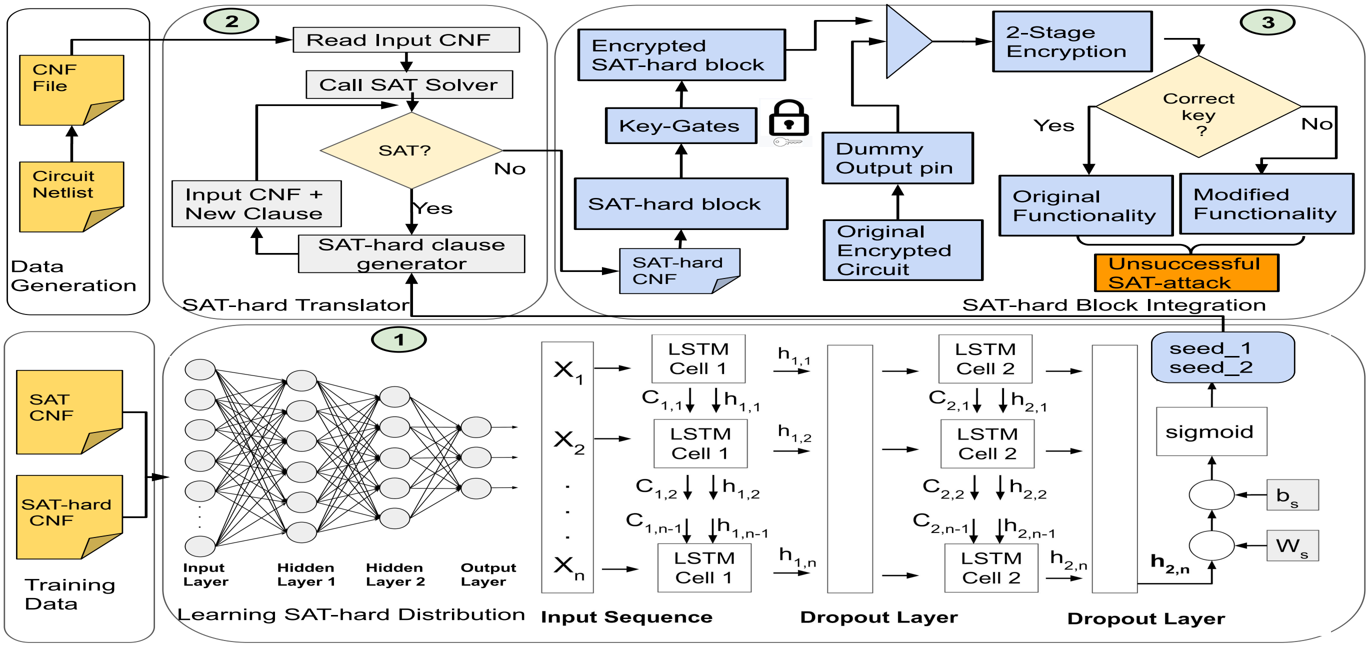


Fig. 2: Architecture of proposed SATConda.

clauses that is SAT-hard. Block 3 takes this SAT-hard CNF as input and integrates it with the original circuit thus enhancing the security.

### A. SAT to SAT-hard Translator

To achieve an unbreakable obfuscated circuit SATConda 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 pattern. This is performed cognitively with the aid of neural network from which the seeds for clause generation is extracted. The details of the neural network are discussed in the next subsections.

The core of our SAT-hard CNF translator, we require two decimal numbers for the initialization of literals for a given clause. In order to make use of the SAT-hard CNF block has lower overhead on the encrypted design we try to optimize the number of literals on each of the clauses. Thus, we initialize the lit_base value with either 1 or 2. We chose the value for lit_base either 1 or 2, and the choice of the selection was randomly picked and could be other values too. We picked this value to start with a lower number of literals during the initialization phase and increase the value only if it is required to make the CNF SAT-hard. We determine this value by comparing a randomly generated decimal number with seed1. In order to get the total number of literals on a given clause then we sample an integer with the probability of seed2 from a SAT-hard clause distribution. Finally, we get the number of literals for a clause. We continue this process until the CNF becomes SAT-hard. Our neural network model predicts the value of seed1 and seed2 for each of the benchmark circuits. The model that has higher prediction accuracy on the seed values causes lower overhead on the SAT-hard block.

The SAT-hard block generation process starts with getting the seed values and passing them to $seed_1$ and $seed_2$ variables (see Figure 2). The values of $seed_1$ and $seed_2$ are determined by fitting the learnt distribution to Bernoulli and Geometric distributions, respectively. A randomly generated decimal number between 0 and 1 is then compared with the $seed_1$ value (see Algorithm 1 line 5). Another variable $Literal\_base$ is assigned as integer 1 if the decimal number is greater than the $seed_1$ otherwise integer 2 (see Algorithm 1 line 6-8). Then the generator draws samples from a geometric distribution at a probability of $seed_2$ and assigns that value to a variable ($Rand\_geo$) (see Algorithm 1 line 10). Adding variables $Literal\_base$ and $Rand\_geo$ we have the length of a new clause (how many literals in the clause) (see Algorithm 1 line 12).The $n\_var$ variable is the number of literals present in a given CNF file. Then SATConda samples a variable from the original CNF variable list without replacement. SATConda samples the variable with a 50% probability of taking that variable or negating that variable. Appending the new clause with the previous clauses SATConda again checks for the satisfiability and keeps adding clauses until the new obfuscated CNF becomes SAT-hard.

A function named $GENERATE\_CLAUSE$ generates each clause using the learnt parameters ( discussed in Section IV-C ) and returns an SAT-hard block. Algorithm 1 line 23-33 shows this function. Then we integrate that block to the original circuit. We discuss this integration process in Section IV-B.

The range for the $seed_1$ and $seed_2$ is between 0 and 1. The greater the value of the seed the less number of clauses are required to make a circuit SAT-hard. Algorithm 1 presents the aforementioned clause generation steps.

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

### B. SAT-hard Block Integration

Though the generated SAT-hard block can be effective against traditional SAT attacks, it can be vulnerable to additional attacks such as partitioning-based removal attacks [25]. To address such vulnerabilities, we tightly integrate the generated SAT-hard clause (circuit) effectively with the original circuit. All the training and SAT-hard clause generation steps are offline and thus, the blocks 1 and 2 will not be included in the original circuit. Only the SAT-hard block will be integrated with the final IC to protect against SAT attacks.

Figure 3 illustrates the integration process of our proposed model. Here, our main objective is to hide the original output for a particular output pin from the attacker. To do so, SATConda adds a dummy output pin by replacing one of the original output pin, as shown in Fig. 3. 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 SATConda 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 this 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 because 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, the partitioning-based removal attack [25] will not succeed to identify the SAT-hard block. Finally, with proper key value, the original functionality of the hidden output pin is retrieved.

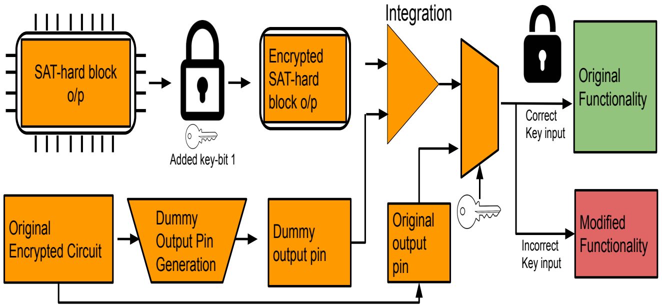


Fig. 3: Integration of proposed SATConda.

### C. Learning the SAT and SAT-hard Distributions

As aforementioned, the proposed technique introduces a SAT-hard block and integrates it with the input CNF. The generation of the SAT-hard block is performed in a cognitive manner to alleviate large overheads. For this purpose, we utilize a message passing neural network (MPNN) similar to [26] 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 [26]. 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.

The deployed MPNN encompasses of three layer fully-connected layers followed by a two-layer long short term memory (LSTM) 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.

Message is passed back and forth along the edges of the network [24]. 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 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. 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. The LSTM network [27] updates the literals $L^{t+1}$ and clauses $C^{t+1}$ at each iteration, as follows:

$$C_u([M^T L_{msg}(L^t)]) \rightarrow C^{t+1} \tag{1}$$

$$C_u([C_h^t]) \rightarrow C_h^{t+1} \tag{2}$$

$$L_u([Flip(L^t), MC_{msg}(C^{t+1})]) \rightarrow L^{t+1} \tag{3}$$

$$L_u([L_h^t]) \rightarrow L_h^{t+1} \tag{4}$$

Eq. 1 and 3 represents one iteration of the MPNN network where clause and literals updated their current embeddings. In Eq. 1, clause C updates its embedding based on messages passed from the neighboring literates whereas in Eq. 3 a literal updates its current embedding after receiving a message from the neighboring clause. Eq. 2 and 4 represent the hidden layer of the LSTM network for updating the clauses and literals. Thus, message is being passed across all the neighboring clauses and literals and the underlying circuit features are then learned by the model.

SATConda predicts whether a problem is SAT or SAT-hard. This message-passing architecture lets the SATConda 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. In order to generate the SAT-hard clause, we choose the seed values that best-fit SAT-hard distribution and utilize it for SAT-hard clause generator to generate an SAT-hard block with minimum number of clauses yet secure. In other words, the $seed_1$ and $seed_2$ values are determined as the values that best fit the SAT-hard distribution. This process leads to minimum area and power overhead which is one of the main challenges for logic locking and other obfuscation techniques.

### D. Training and Testing The MPNN Model

The training dataset is obtained by utilizing our SAT to SAT-hard converter for converting a SAT-CNF to a SAT-hard one which is explained in Section IV-A. To train the neural network (NN) for learning SAT and SAT-hard distributions we provide the NN a pair of SAT problems. In this pair, one problem is satisfiable (SAT) and the other one is SAT-hard. In the training pool, we have a total of $m$ SAT problems and $n$ SAT-hard problems where $\mathcal{C}_m = \{C_1, C_2, \cdots, C_m\}$ and $\mathcal{C}_n = \{C'_1, C'_2, \cdots, C_n\}$. 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.

The training dataset preparation starts with formulating a single clause with a number of variables and then calling a SAT-solver to test the satisfiability. This process continues iteratively until all the clauses become SAT-hard. By finishing this process, we achieve the SAT-hard part of the SAT problem pairs and yet to get the satisfiable part. For this, we perturb the SAT-hard CNF by means of flipping a literals so that the CNF is satisfiable by the SAT-solver.

We label each of the training data either 0 or 1, where 0 stands for SAT-hard and 1 means satisfiable (SAT). The fully connected MLP updates the weights for each neuron and learn the relationship between the feature vectors, i.e., clauses and literals for a $\{SAT, SAT-hard\}$ pair. Then the predicted output from the MLP is passed to the LSTM network and the LSTM network gets updated. After a number of iterations the network predicts whether a problem is SAT or SAT-hard. Our MPNN model is trained on ten thousand SAT problems (five thousand are SAT-hard) for better learning and efficient modeling of SAT and SAT-hard distributions. We choose the model that achieves the highest accuracy for generating the SAT-hard block. Our best-fit model has a training-cost of 0.6930 and a validation cost of 0.6932, which is sufficiently good enough to ensure that the model generalizes well with no over or under-fitting. All the training and SAT-hard clause generation steps are offline and thus, the blocks 1 and 2 will not be included in the original circuit. Only the SAT-hard block will be integrated with the final IC to protect against SAT attacks.

**Algorithm 1** SATConda Algorithm

```
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
$SAT - hard - cnf := solve\_clauses$
```

## V. Experimental Results

In this section we describe the experimental setup and evaluate the impact of SATConda in terms of SAT hardness and the incurred overheads.

### A. Experimental Setup

In this work, we used the MPNN model similar to [26] to obtain the seed required for our random 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 shown in Table I. We used two different obfuscation techniques to encrypt ISCAS'85 and ISCAS'89 benchmark circuits in this work. For ISCAS'85 benchmark, one algorithm was proposed by Rajendran et al. [28] that inserts XOR/XNOR gates (referred to as "DAC'12") at different locations of the circuit to prevent the fault-analysis attack. The second algorithm was proposed by Dupuis et al. [29] that inserts AND/OR gates (referred to as "IOLTS'14") at multiple chosen locations of the circuit.

For ISCAS'89 benchmark, we used the IOLTS'14 obfuscation [29] technique. It needs to be noted that the algorithm [6] was unable to encrypt the ISACAS'89 circuit using DAC'12 algorithm despite executing it for three days. Instead of DAC'12, we applied a similar technique that is also a XOR-based logic locking (referred to as "TOC'13") algorithm [30] to obfuscate ISCAS'89 benchmark circuit, which is also proposed by the same group. It needs to be noted that the proposed technique enhances the security of the CNF and is independent of the underlying obfuscation technique. We considered these obfuscation techniques as mere case studies to show the effectiveness.

While generating the obfuscated benchmark circuits for XOR/XNOR-based logic locking DAC'12 [28], AND/OR-based logic locking IOLTS'14 [29], and XOR-based logic locking ToC'13 [30], which adds 5% area overhead. As the circuit size increases, the number of key-bit as well increases. 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 through aforementioned techniques, the proposed technique needs one more key bit to integrate the SAT-hard block to the original obfuscated circuit. One additional key-bit is required to encrypt the dummy output pin with the original output pin. So, our proposed method requires two additional key-bits. The number of iterations required for the clause generation process depend on various factors. One key factor being the learnt parameters, i.e., seed values, from the neural network. We have discussed the significance of seed values in Section 4.2. In summary, if the learned distribution fits well to the Bernoulli and Geometric distributions, then the value of $seed_1$ and $seed_2$ approaches to 1. Thus, the algorithm requires less number of iterations for the clause generation process. For our best-trained model, the number of iterations ranges from 60 iterations (for the smallest circuits) to 1005 iterations.

In addition to verification against traditional SAT attacks used in hardware security domain, we have also verified the satisfiability of CNF using three different SAT solvers, MiniSAT [31], Lingeling [32], and Glucose [33]. 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 area and power overheads are calculated using Synopsys Design Compiler, Version: L-2016.03-SP3. SAED 90nm EDK Digital Standard Cell Library [34] 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

Here, we present the evaluation in terms of SAT-hardness and the overhead analysis in addition to our empirical findings of SATConda.

*1) SAT-hardness:* Table II shows SAT-attack [6] performance on two different encryption algorithms [29], [28] on ISCAS'85 benchmarks before and after they are passed through SATConda. It shows that the SAT-attack in [6] successfully decrypts the keys of the obfuscated circuit (obfuscated by [29],[28]) in less than a minute. However, the same SAT-attack fails to extract the keys when it is further encrypted using SATConda. For both the XOR/XNOR-based logic locking [28] and AND/OR-based logic locking [29], SATConda successfully defends the SAT-attack for all the circuits. The timeout for the SAT-attack is set to 10 hours in our experiments. In case of the c2670 circuit, encryption in [28] itself resists against SAT attack, thus, SATConda does not introduce additional circuit. Table III presents the satisfiability of the ISCAS'89 benchmark circuits on IOLTS'14 and TOC'13 encryption before and after the conversion through SATConda. From the results it could be observed that the proposed SATConda can enhance the security of the underlying obfuscation techniques.

In addition to SAT attack in [6], we have also evaluated the robustness with SATConda in terms of SAT-hardness against other SAT attacks such as AppSAT [35]. Table IV represents the SATConda evaluation on AppSAT-attack [35] for ISCAS'85 benchmark circuits. As can be observed that the proposed SATConda is able to resist the AppSAT attack, which earlier was able to decrypt the benchmark circuits. Table V represents the SATConda evaluation on AppSAT attack for ISCAS'89 benchmark circuits. Similar resiliency against AppSAT attack is observed with proposed SATConda when applied on both encryption schemes.

As it is nearly impossible to evaluate against all the existing and future SAT attacks, we consider the traditional SAT attacks which form basis for most of the present day SAT-based deobfuscation attacks for evaluation. The satisfiability of the encryption techniques were verified against three different SAT solvers, MiniSAT [31], Lingeling [32], and Glucose [33]. Table VI presents the satisfiability of the ISCAS'85 benchmark circuits on IOLTS'14 encryption before and after the conversion through SATConda. It can be seen that all the encrypted benchmark circuits using IOLTS'14 encryption were satisfiable (breakable) with all the three traditional SAT-solvers [31], [32], [33]. This indicates that an attacker could perform a SAT-attack with any of these SAT-solvers and reverse engineer the IP/IC. Table VI also shows that once the IC/IP design when translated using SATConda, it becomes SAT-hard, indicating that none of the three experimented SAT-solvers, which were previously successful, could solve a satisfying assignment.

Table VII shows the satisfiability of the same ISCAS'85 benchmark circuits on DAC'12 logic locking [28] before and after the conversion through SATConda. Similar to the IOLTS'14 logic locking [29], the DAC'12 logic locking [28] was vulnerable to these SAT solvers. When this encrypted circuit was further passed through SATConda it becomes SAT-hard. Meaning that with the specified timeout window the SAT solvers fails to satisfy for any assignment. Table VIII and IX present the satisfiability result for ISCAS'89 circuits using the IOLTS'14)logic locking [29] and TOC'13 logic locking [30] methods, respectively. Similar to ISCAS'85 benchmarks, ISCAS'89 benchmarks also showcase resiliency against the traditional SAT attacks when passed through the proposed SATConda.

*2) Overhead Analysis:* In addition to SAT-hardness, we evaluate the imposed overheads through the conversion. Table X reports the area and power overhead of the original encrypted circuit (XOR/XNOR-based logic locking [28] and AND/OR-based logic locking [29]) and compare them with the overhead of the proposed SATConda applied on top of existing encryption techniques to successfully defends SAT-

TABLE I: ISCAS'85 and ISCAS'89 benchmark circuits

| ISCAS'85 Circuit | | | | ISCAS'89 Circuit | | | |
|---|---|---|---|---|---|---|---|
| Circuit Name | #Inputs | #Outputs | #Gates | Circuit Name | #Inputs | #Outputs | #Gates |
| | | | | s382 | 24 | 27 | 392 |
| c432 | 36 | 7 | 160 | s400 | 24 | 27 | 414 |
| c499 | 41 | 32 | 202 | s641 | 54 | 42 | 459 |
| c880 | 60 | 26 | 383 | s526n | 24 | 27 | 494 |
| c1355 | 41 | 32 | 546 | s526 | 3 | 6 | 141 |
| c1908 | 33 | 25 | 880 | s953 | 45 | 29 | 950 |
| c2670 | 233 | 140 | 1193 | s1488 | 14 | 25 | 843 |
| c3540 | 50 | 22 | 1669 | s5378 | 214 | 228 | 5183 |
| c5315 | 178 | 123 | 2307 | s13207 | 700 | 790 | 11248 |
| c7552 | 207 | 108 | 3512 | s15850 | 611 | 684 | 13192 |
| | | | | s35932 | 1763 | 1728 | 31833 |

TABLE II: SATConda evaluation on SAT-attack [6] for ISCAS'85 circuit with different encryption techniques

| Circuit Name | IOLTS'14 [29] | | DAC'12 [28] | |
|---|---|---|---|---|
| | Time (s) | | Time (s) | |
| | Before Conversion | After Conversion | Before Conversion | After Conversion |
| c432 | 0.033 | timeout | 0.026 | timeout |
| c499 | 0.060 | timeout | 0.070 | timeout |
| c880 | 0.061 | timeout | 0.094 | timeout |
| c1355 | 0.042 | timeout | 0.312 | timeout |
| c1908 | 0.049 | timeout | 0.518 | timeout |
| c2670 | 0.544 | timeout | timeout | ** |
| c3540 | 0.323 | timeout | 3.264 | timeout |
| c5315 | 0.826 | timeout | 9.013 | timeout |
| c7552 | 0.467 | timeout | 26.75 | timeout |

timeout = 10 hours

** = Excluded from the experiment as the original encrypted circuit was unbreakable by SAT-attack

TABLE III: SATConda evaluation on SAT-attack [6] for ISCAS'89 circuits with different encryption techniques

| Circuit Name | IOLTS'14 [29] | | ToC'13 [30] | |
|---|---|---|---|---|
| | Time (s) | | Time (s) | |
| | Before Conversion | After Conversion | Before Conversion | After Conversion |
| s382 | 0.006 | timeout | 0.006 | timeout |
| s400 | 0.006 | timeout | 0.006 | timeout |
| s526n | 0.0087 | timeout | 0.008 | timeout |
| s526 | 0.0084 | timeout | 0.008 | timeout |
| s641 | 0.009 | timeout | 0.0092 | timeout |
| s953 | 0.017 | timeout | 0.017 | timeout |
| s1488 | 0.048 | timeout | 0.0344 | timeout |
| s5378 | 0.070 | timeout | 0.072 | timeout |
| s13207 | 0.252 | timeout | 0.232 | timeout |
| s15850 | 0.328 | timeout | 0.336 | timeout |
| s35932 | 5.831 | timeout | 5.890 | timeout |

timeout = 10 hours

TABLE IV: SATConda evaluation on AppSAT-attack [35] for ISCAS'85 circuits

| Circuit Name | IOLTS'14 [29] | | ToC'13 [30] | |
|---|---|---|---|---|
| | Time (s) | | Time (s) | |
| | Before Conversion | After Conversion | Before Conversion | After Conversion |
| c432 | 0.055 | timeout | 0.058 | timeout |
| c499 | 0.085 | timeout | 0.086 | timeout |
| c880 | 0.1598 | timeout | 0.20 | timeout |
| c1355 | 0.098 | timeout | 0.219 | timeout |
| c1908 | 0.122 | timeout | 3.945 | timeout |
| c2670 | 1.712 | timeout | 1.380 | timeout |
| c3540 | 1.968 | timeout | 4.921 | timeout |
| c5315 | 2.888 | timeout | 1.71 | timeout |
| c7552 | 4.506 | timeout | 2.94 | timeout |

timeout = 10 hours

TABLE V: SATConda evaluation on AppSAT-attack [35] for ISCAS'89 circuits with different encryption schemes

| Circuit Name | IOLTS'14 [29] | | ToC'13 [30] | |
|---|---|---|---|---|
| | Time (s) | | Time (s) | |
| | Before Conversion | After Conversion | Before Conversion | After Conversion |
| s382 | 0.066 | timeout | 0.032 | timeout |
| s400 | 0.067 | timeout | 0.033 | timeout |
| s526n | 0.089 | timeout | 0.044 | timeout |
| s526 | 0.090 | timeout | 0.045 | timeout |
| s641 | 0.112 | timeout | 0.053 | timeout |
| s953 | 0.157 | timeout | 0.077 | timeout |
| s1488 | 0.316 | timeout | 0.117 | timeout |
| s5378 | 0.541 | timeout | 0.268 | timeout |
| s13207 | 1.52 | timeout | 0.588 | timeout |
| s15850 | 2.265 | timeout | 0.887 | timeout |
| s35932 | 17.882 | timeout | 8.138 | timeout |

timeout = 10 hours

TABLE VI: SATConda evaluation using IOLTS'14 [29] encryption on different SAT-solvers for ISCAS'85 benchmarks

| Circuit Name | miniSAT [31] | | Lingeling [32] | | Glucose [33] | |
|---|---|---|---|---|---|---|
| | Time(s) | | Time(s) | | Time(s) | |
| | Before Conversion | After Conversion | Before Conversion | After Conversion | Before Conversion | After Conversion |
| c432 | 0.0039 | ✗ | 0.1 | ✗ | 0.0015 | ✗ |
| c499 | 0.0026 | ✗ | 0.1 | ✗ | 0.0011 | ✗ |
| c880 | 0.0024 | ✗ | 0.1 | ✗ | 0.0014 | ✗ |
| c1355 | 0.0018 | ✗ | 0.1 | ✗ | 0.0005 | ✗ |
| c1908 | 0.0041 | ✗ | 0.1 | ✗ | 0.0031 | ✗ |
| c2670 | 0.0049 | ✗ | 0.1 | ✗ | 0.0003 | ✗ |
| c3540 | 0.0060 | ✗ | 0.1 | ✗ | 0.0061 | ✗ |
| c5315 | 0.0086 | ✗ | 0.1 | ✗ | 0.0094 | ✗ |
| c7552 | 0.0095 | ✗ | 0.1 | ✗ | 0.0110 | ✗ |

✗= Corresponding CNF was SAT-hardisfiable by the SAT solver

attacks (The original XOR/XNOR-based logic locking [28] and AND/OR-based logic locking [29] fail to secure against SAT attacks).

Both the DAC'12 logic locking [28] and IOLTS'14 logic locking [29] introduce an average area overhead of 5% when compared to the original circuit for the ISCAS'85 benchmarks. However, as seen earlier despite incurring such overheads, the SAT attack [6] can reverse engineer the IP. Further encrypting with SATConda incurs additional overhead of about 57% and about 42% on average compared to [29] and [28], respectively

TABLE VII: SATConda evaluation using DAC'12 encryption [28] on different SAT-solvers for ISCAS'85 benchmarks

| Circuit Name | miniSAT [31] | | Lingeling [32] | | Glucose [33] | |
|---|---|---|---|---|---|---|
| | Time(s) | | Time(s) | | Time(s) | |
| | Before Conversion | After Conversion | Before Conversion | After Conversion | Before Conversion | After Conversion |
| c432 | 0.0049 | ✗ | 0.1 | ✗ | 0.004249 | ✗ |
| c499 | 0.005691 | ✗ | 0.1 | ✗ | 0.004277 | ✗ |
| c880 | 0.006145 | ✗ | 0.1 | ✗ | 0.005559 | ✗ |
| c1355 | 0.006179 | ✗ | 0.1 | ✗ | 0.006117 | ✗ |
| c1908 | 0.006226 | ✗ | 0.1 | ✗ | 0.007626 | ✗ |
| c2670 | 0.003507 | ✗ | 0.1 | ✗ | 0.008887 | ✗ |
| c3540 | 0.001801 | ✗ | 0.1 | ✗ | 0.009222 | ✗ |
| c5315 | 0.005401 | ✗ | 0.1 | ✗ | 0.009447 | ✗ |
| c7552 | 0.010083 | ✗ | 0.1 | ✗ | 0.010702 | ✗ |

✗= Corresponding CNF was SAT-hardisfiable by the SAT solver

TABLE VIII: SATConda with IOLTS'14 encryption [29] evaluated on different SAT-solvers for ISCAS'89 circuits

| Circuit Name | miniSAT [31] | | Lingeling [32] | | Glucose [33] | |
|---|---|---|---|---|---|---|
| | Time(s) | | Time(s) | | Time(s) | |
| | Before Conversion | After Conversion | Before Conversion | After Conversion | Before Conversion | After Conversion |
| s382 | 0.0016 | ✗ | 0.2 | ✗ | 0.00347 | ✗ |
| s400 | 0.0014 | ✗ | 0.2 | ✗ | 0.0031 | ✗ |
| s526n | 0.0015 | ✗ | 0.2 | ✗ | 0.0045 | ✗ |
| s526 | 0.0061 | ✗ | 0.2 | ✗ | 0.0035 | ✗ |
| s641 | 0.0014 | ✗ | 0.2 | ✗ | 0.0025 | ✗ |
| s953 | 0.0018 | ✗ | 0.2 | ✗ | 0.0012 | ✗ |
| s1488 | 0.0041 | ✗ | 0.2 | ✗ | 0.0070 | ✗ |
| s5378 | 0.0034 | ✗ | 0.2 | ✗ | 0.0076 | ✗ |
| s13207 | 0.0079 | ✗ | 0.2 | ✗ | 0.0065 | ✗ |
| s15850 | 0.0068 | ✗ | 0.2 | ✗ | 0.0121 | ✗ |
| s35932 | 0.0155 | ✗ | 0.2 | ✗ | 0.0202 | ✗ |

✗= Corresponding CNF was SAT-hardisfiable by the SAT solver

TABLE IX: SATConda evaluation using TOC'13 encryption [30] on different SAT-solvers for ISCAS'89 circuits

| Circuit Name | miniSAT [31] | | Lingeling [32] | | Glucose [33] | |
|---|---|---|---|---|---|---|
| | Time(s) | | Time(s) | | Time(s) | |
| | Before Conversion | After Conversion | Before Conversion | After Conversion | Before Conversion | After Conversion |
| s382 | 0.001 | ✗ | 0.2 | ✗ | 0.0007 | ✗ |
| s400 | 0.001 | ✗ | 0.2 | ✗ | 0.0006 | ✗ |
| s526n | 0.001 | ✗ | 0.2 | ✗ | 0.0007 | ✗ |
| s526 | 0.001 | ✗ | 0.2 | ✗ | 0.0009 | ✗ |
| s641 | 0.001 | ✗ | 0.2 | ✗ | 0.0007 | ✗ |
| s953 | 0.002 | ✗ | 0.2 | ✗ | 0.001 | ✗ |
| s1488 | 0.005 | ✗ | 0.2 | ✗ | 0.002 | ✗ |
| s5378 | 0.003 | ✗ | 0.2 | ✗ | 0.002 | ✗ |
| s13207 | 0.009 | ✗ | 0.2 | ✗ | 0.009 | ✗ |
| s15850 | 0.009 | ✗ | 0.2 | ✗ | 0.007 | ✗ |
| s35932 | 0.015 | ✗ | 0.2 | ✗ | 0.012 | ✗ |

✗= Corresponding CNF was SAT-hardisfiable by the SAT solver

as shown in Table X, but guarantees security. In a similar manner, proposed SATConda incurs around 54% and 36% power overhead on an average compared to [29] and [28], respectively. As observed that the majority of the overhead comes from the base encryption technique (Say DAC'12 or IOLTS'14) rather than the proposed SATConda. Thus, if a lightweight encryption which is moderate in terms of security exist, it can be made secure with SATConda with minimal overheads.

Table XII reports the area and power overhead of the ISCAS'89 encrypted circuit using (IOLTS'14 logic locking [29] and TOC'13 logic locking [30]) with our proposed method. Here, the TOC'13 [30] and IOLTS'14 [29] logic locking also introduce an average area overhead of 5% when compared to the original circuit. Further encrypting with SATConda incurs additional area overhead of about 59% and about 62% on average compared to [29] and [30], respectively. Similarly, proposed SATConda incurs around 68% and 74% power overhead on an average compared to [29] and [30], respectively. Table XIII reports the delay overhead of the ISCAS'89 encrypted circuit with our proposed method. We also evaluate the delay overhead for our proposed model. Table XI shows the delay overhead due to SATConda. With the increase in the original circuit size the delay overhead is also increased. However, our proposed model shows non-linear relation for delay overhead size as we keep increasing the circuit size. For example, the c3540 circuit has less delay overhead than the smaller c1908 circuit. We believe that the reason behind this lies in the input vs output ratio and the type of gate used for a given circuit. Though c3540 circuit has almost double gate count than c1908 circuit, the number of output pin for c1908 is more than c3540. In order to deal with more output pin to make that circuit SAT-hard our model needs to add more clauses which leads greater delay overhead.

From Table X, the area overhead follows another important trend. The relative area overhead due to our model follows a non-linear trend as the circuit size keeps increasing. For small circuits (i.e c432, c499 etc.) the area overhead is large compared to the area overhead for the large circuits such as c3540, c5315, and c7552. This depicts the scalability and better applicability for real-world larger circuits. For power overhead analysis, a very similar trend is observed.

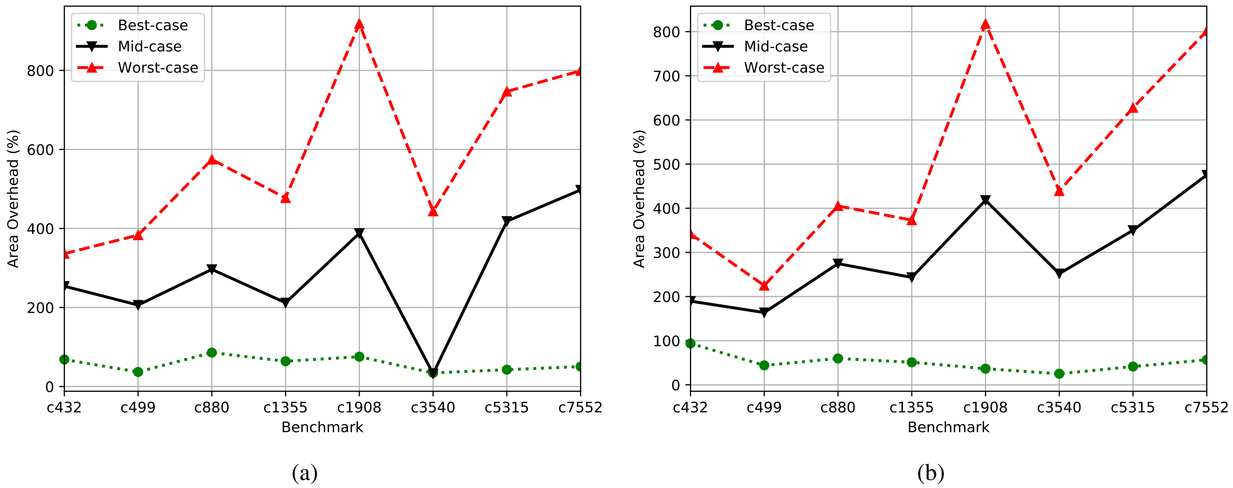
Based on the overall evaluations performed, we confirm that SATConda performs efficient translation of SAT to SAT-hard for both the ISCAS'85 and ISCAS'89 benchmarks with lower overhead, power consumption without deviating from the original functionality.

*3) Impact of MPNN:* As the clause generation process is based on the model learnt by the MPNN, different training data can lead to different learnt models and different seed values. We analyze the impact of the seed on the overheads here, as all of them lead to SAT-hard in terms of security. We compare three different models that we achieved from SATConda. We named them as best-fit model (for $seed_1 = 0.9$ and $seed_2 = 0.9$ - leading to lowest overhead), mid-fit model(for $seed_1 = 0.5$ and $seed_2 = 0.5$), and worst-fit model(for $seed_1 = 0.3$ and $seed_2 = 0.4$). Figure 4 depicts the area overhead (%), Figure 5 shows the power overhead (%), and Figure 6 illustrates the delay overhead for different models.

Figure 4 depicts the area overhead (%) for ISCAS'85 benchmark circuit for different models where Figure 4a shows area overhead on IOLTS'14 encryption and Figure 4b shows area overhead on DAC'12 encryption. As can be seen that the worst-fit model gives relatively large area overhead for all the benchmark circuits. The reason behind this is the worst-fit model adds a significant number of clauses to the original CNF file. Another important trend is observed from the Figure 4 that for best fit model the overhead is significantly lower and the overhead curve is almost horizontal than the mid-fit and the worst-fit model. On the other hand the worst-fit model shows significant fluctuations on different circuits as the model is not properly trained. On average, the area overhead for the best-fit, mid-fit, and the worst-fit model for the IOLTS'14 encryption is about 57%, 288%, and 584% respectively and for the DAC'12 encryption is about 51%, 295%, and 503% respectively; irrespective of the model fit used for generating or perturbting the clauses, each circuit shows security against SAT-attack.

Figure 5 depicts the power overhead (%) for ISCAS'85 benchmark circuit for different models where Figure 5a shows power overhead on IOLTS'14 encryption and Figure 5b shows

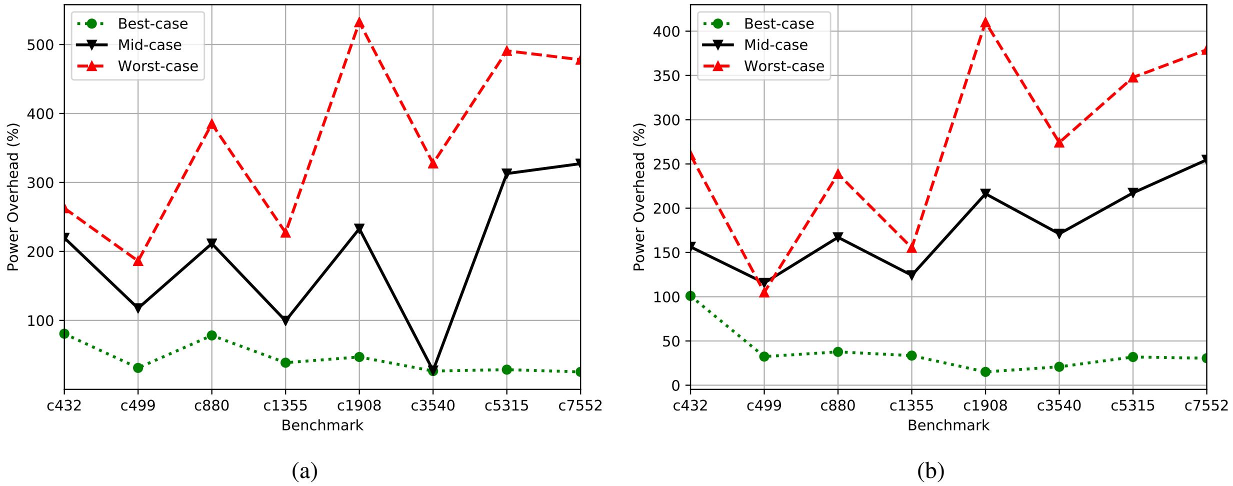
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3138686, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems



(a)

(b)

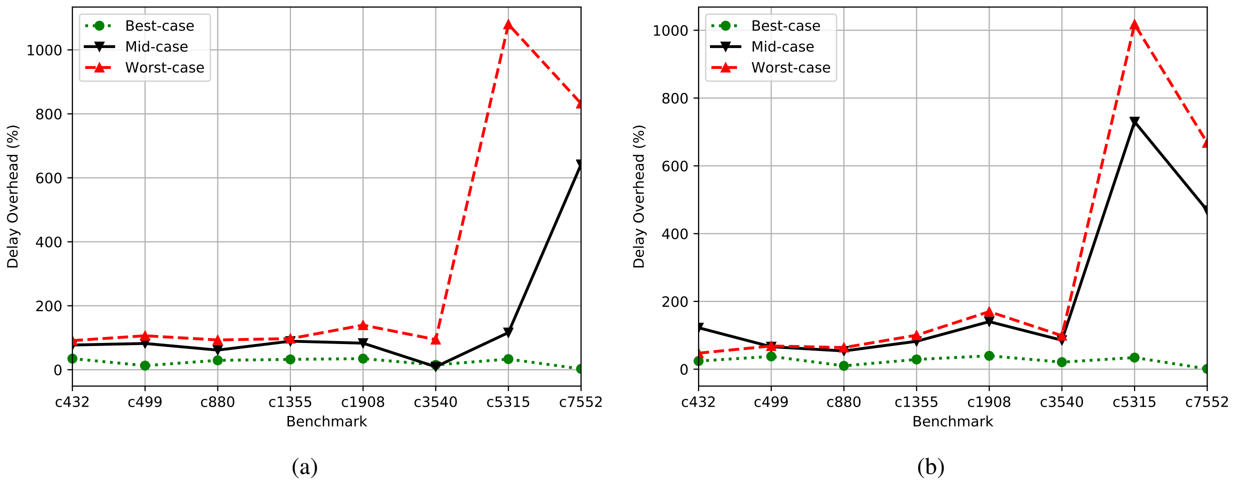
Fig. 4: Area overhead analysis for ISCAS'85 benchmark circuit for different cases: (a) Area overhead on IOLTS'14 encryption [29], (b) Area overhead on DAC'12 encryption [28]



(a)

(b)

Fig. 5: Power overhead analysis for ISCAS'85 benchmark circuit for different cases: (a) power overhead on IOLTS'14 encryption [29]; (b) Power overhead on DAC'12 encryption [28]



(a)

(b)

Fig. 6: Delay overhead analysis for ISCAS'85 benchmark circuit for different cases. (a) Delay overhead on IOLTS'14 encryption [29]; (b) Delay overhead on DAC'12 encryption [28]

TABLE X: Area and power overhead analysis of SATConda with different encryption schemes for ISCAS'85 circuits

| | Area Overhead | | | | | | Power Overhead | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Circuit | IOLTS-'14 [29] ($\mu m^2$) | SAT-Conda+[29] ($\mu m^2$) | Overhead (%) | DAC12 [28] ($\mu m^2$) | SAT-Conda+[28] ($\mu m^2$) | Overhead (%) | IOLTS-'14 [29] ($\mu W$) | SAT-Conda+[29] ($\mu W$) | Overhead (%) | DAC12 [28] ($\mu W$) | SAT-Conda+[28] ($\mu W$) | Overhead (%) |
| c432 | 1,058.79 | 2,079.20 | 96.38 | 1,113.92 | 2,158.15 | 93.74 | 18.21 | 37.40 | 105.34 | 19.62 | 39.42 | 100.91 |
| c499 | 1,918.88 | 3,044.98 | 58.69 | 1,978.63 | 2,867.02 | 44.90 | 51.82 | 75.17 | 45.06 | 54.60 | 72.61 | 32.98 |
| c880 | 1,613.51 | 2,861.77 | 77.36 | 1,754.96 | 2,756.61 | 57.08 | 31.19 | 53.01 | 69.97 | 38.65 | 52.62 | 36.13 |
| c1355 | 2,051.21 | 3,683.81 | 79.59 | 2,154.95 | 3,256.43 | 51.11 | 53.42 | 79.66 | 49.10 | 61.45 | 81.97 | 33.39 |
| c1908 | 2,170.10 | 3,553.32 | 63.74 | 2,628.10 | 3,587.28 | 36.50 | 49.19 | 70.46 | 43.25 | 68.53 | 78.84 | 15.05 |
| c3540 | 4,971.22 | 7,258.89 | 46.02 | 5,716.99 | 7,151.22 | 25.09 | 91.44 | 131.96 | 44.31 | 125.19 | 151.26 | 20.82 |
| c5315 | 7,279.01 | 10,770.00 | 47.96 | 8,660.90 | 12,231.05 | 41.22 | 148.50 | 198.50 | 33.67 | 205.55 | 271.21 | 31.94 |
| c7552 | 9,697.55 | 14,856.48 | 53.20 | 10,900.76 | 17,061.17 | 56.51 | 219.34 | 282.67 | 28.87 | 297.66 | 388.55 | 30.54 |
| **Average** | | | 57.7 | | | 42.8 | | | 54.44 | | | 36.54 |

TABLE XI: Delay overhead analysis of SATConda with different encryption schemes for ISCAS'85 circuits

| | Delay Overhead | | | | | |
|---|---|---|---|---|---|---|
| Circuit | IOLTS-'14 [29] ($ns$) | SAT-Conda+[29] ($ns$) | Overhead (%) | DAC12 [28] ($ns$) | SAT-Conda+[28] ($ns$) | Overhead (%) |
| c432 | 1.01 | 1.36 | 34.65 | 1.12 | 1.39 | 24.11 |
| c499 | 2.57 | 2.9 | 12.84 | 2.73 | 3.76 | 37.73 |
| c880 | 2.9 | 3.75 | 29.31 | 3.1 | 3.41 | 10 |
| c1355 | 2.6 | 3.44 | 32.31 | 2.71 | 3.49 | 28.78 |
| c1908 | 3.72 | 5.01 | 34.68 | 3.52 | 4.91 | 39.49 |
| c3540 | 4.85 | 5.58 | 15.05 | 4.84 | 5.85 | 20.87 |
| c5315 | 3.55 | 4.73 | 33.24 | 5.22 | 7.01 | 34.29 |
| c7552 | 4.86 | 4.99 | 2.67 | 7.05 | 7.12 | 0.99 |
| **Average** | | | 24.34 | | | 24.53 |

power overhead on DAC'12 encryption. On average, the power overhead for the best-fit, mid-fit, and the worst-fit model for the IOLTS'14 encryption is about 44%, 190%, and 360% respectively and for the DAC'12 encryption is about 37%, 177%, and 271% respectively.

Figure 6 depicts the delay overhead (%) for ISCAS'85 benchmark circuit for different models where Figure 6a shows delay overhead on IOLTS'14 encryption and Figure 6b shows delay overhead on DAC'12 encryption. On an average, the delay overhead for the best-fit, mid-fit, and the worst-fit model for the IOLTS'14 encryption is about 24%, 145%, and 316% respectively and for the DAC'12 encryption is about 24%, 218%, and 275% respectively. A similar trend was observed for ISCAS'89 benchmark circuits on area, power, and delay overhead using our different models, not presented for the purpose of conciseness.

In summary, Figures 4, 5, and 6 are presented to evaluate three different models with different accuracy. While integrating the SAT-hard block with the original IC, we only provide the block from the best-fit model with the lower area, power, and delay overhead.

From the above analysis, a straightforward observation can be made. The best-fit model shows lower area, power, and delay overhead compared to the other two cases. We have this best fit model when we train our MPNN with a large number of obfuscated circuit with large number of variables. In return, the best-fit model adds less number of clauses with minimum literals in each clause to meet the area, power, and delay overhead constraint.

## VI. Conclusion

In this work we successfully defend the popular SAT-attack by introducing a neural network based SAT-hard problem generator, SATConda, to 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 SAT-attack, AppSAT attack and three other traditional SAT solvers.

## References

[1] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor, "Trustworthy hardware: Identifying and classifying hardware trojans," *Computer*, vol. 43, no. 10, pp. 39–46, 2010.

[2] M. Rostami, F. Koushanfar, and R. Karri, "A primer on hardware security: Models, methods, and metrics," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1283–1295, 2014.

[3] J. J. Rajendran, O. Sinanoglu, and R. Karri, "Is split manufacturing secure?" in *Conf. on Design, Automation and Test in Europe*, 2013.

[4] M. Yasin, J. J. Rajendran, O. Sinanoglu, and R. Karri, "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.

[5] M. Yasin, B. Mazumdar, J. J. Rajendran, and O. Sinanoglu, "SARLock: SAT attack resistant logic locking," in *Int. Symp. on Hardware Oriented Security and Trust*, 2016.

[6] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *Int. Symp. on Hardware Oriented Security and Trust*, 2015.

[7] Z. Chen, G. Kolhe, S. Rafatirad, C.-T. Lu, S. M. PD, H. Homayoun, and L. Zhao, "Estimating the circuit de-obfuscation runtime based on graph deep learning," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 358–363.

[8] J. Franco and J. Martin, "Handbook of satisfiability frontiers in artificial intelligence and applications," 2009.

[9] Y. Xie and A. Srivastava, "Mitigating sat attack on logic locking," in *Int. Conf. on Cryptographic Hardware and Embedded Systems*, 2016.

[10] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Security analysis of anti-sat," in *Asia and South Pacific Design Automation conf.*, 2017.

[11] G. Kolhe, S. M. PD, S. Rafatirad, H. Mahmoodi, A. Sasan, and H. Homayoun, "On custom LUT-based obfuscation," in *Great Lakes Symp. on VLSI*, 2019.

[12] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "Full-lock: Hard distributions of sat instances for obfuscating circuits using fully configurable logic and routing blocks," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[13] R. Hassan, G. Kohle, S. Rafatirad, H. Homayoun, and S. M. P. Dinakarrao, "A cognitive sat to sat-hard clause translation-based logic obfuscation," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 1172–1177.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3138686, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

TABLE XII: Area and power overhead analysis of SATConda with different encryption schemes for ISCAS'89 circuits

| | Area Overhead | | | | | | Power Overhead | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Circuit | IOLTS-'14 [29] ($\mu m^2$) | SAT-Conda+[29] ($\mu m^2$) | Over-head (%) | TOC'13-XOR [30] ($\mu m^2$) | SAT-Conda+[30] ($\mu m^2$) | Over-head (%) | IOLTS-'14 [29] ($\mu W$) | SAT-Conda+[29] ($\mu W$) | Over-head (%) | TOC'13-XOR [30] ($\mu W$) | SAT-Conda+ [30] ($\mu W$) | Over-head (%) |
| s382 | 662.47 | 1249.53 | 88.62 | 662.47 | 1349.04 | 103.63 | 10.31 | 20.47 | 98.52 | 10.31 | 22.92 | 122.26 |
| s400 | 657.4 | 1328.3 | 102.05 | 657.40 | 1329.07 | 102.17 | 10.07 | 22.35 | 121.91 | 10.07 | 22.86 | 127.01 |
| s526n | 871.94 | 1649.53 | 89.1 | 871.94 | 1618.36 | 85.6 | 13.58 | 27.27 | 100.85 | 38.65 | 52.62 | 36.13 |
| s526 | 862.18 | 1589.89 | 84.4 | 862.18 | 1701.78 | 97.38 | 13.17 | 26.08 | 98.02 | 13.58 | 26.66 | 96.31 |
| s953 | 1842.92 | 2531.02 | 38.61 | 1824.92 | 2372.20 | 29.99 | 23.76 | 36.65 | 54.25 | 23.76 | 33.41 | 40.59 |
| s5378 | 676.40 | 729.87 | 7.9 | 676.40 | 729.87 | 7.9 | 10.17 | 10.36 | 1.82 | 10.17 | 10.36 | 1.82 |
| s35932 | 38188.04 | 39631.27 | 3.78 | 38188.04 | 43058.6 | 12.75 | 767.28 | 791.8094 | 3.2 | 767.31 | 847.08 | 10.4 |
| **Average** | | | 59.1 | | | 62.2 | | | 68.3 | | | 74.04 |

TABLE XIII: Delay overhead analysis of SATConda with different encryption schemes for ISCAS'89 circuits

| | Delay Overhead | | | | | |
|---|---|---|---|---|---|---|
| Circuit | IOLTS-'14 [29] ($ns$) | SAT-Conda+[29] ($ns$) | Over-head (%) | TOC'13-XOR [30] ($ns$) | SAT-Conda+[30] ($ns$) | Over-head (%) |
| s382 | 1.2 | 1.93 | 60.83 | 1.2 | 1.67 | 39.17 |
| s400 | 1.23 | 1.68 | 36.59 | 1.23 | 1.72 | 39.84 |
| s526 | 1.59 | 1.66 | 4.4 | 1.59 | 1.95 | 22.64 |
| s526n | 1.58 | 2.1 | 32.91 | 1.58 | 2.2 | 39.24 |
| s641 | 3.3 | 3.58 | 8.48 | 3.3 | 3.7 | 12.12 |
| s953 | 1.75 | 1.97 | 12.57 | 1.75 | 2.28 | 30.29 |
| s35932 | 7.7 | 11.29 | 46.62 | 7.7 | 12.9 | 67.53 |
| **Average** | | | 28.21 | | | 31.81 |

[14] R. Hassan, G. Kolhe, S. Rafatirad, H. Homayoun, and S. M. P. Dinakarrao, "Satconda: Sat to sat-hard clause translator," in *2020 21st Int. Symposium on Quality Electronic Design*. IEEE, 2020.

[15] M. C. Hansen, H. Yalcin, and J. P. Hayes, "Unveiling the iscas-85 benchmarks: A case study in reverse engineering," *IEEE Design & Test of Computers*, vol. 16, no. 3, pp. 72–80, 1999.

[16] S. Dupuis and M.-L. Flottes, "Logic locking: A survey of proposed methods and evaluation metrics," *Journal of Electronic Testing*, pp. 1–19, 2019.

[17] J. A. Roy, F. Koushanfar, and I. L. Markov, "Ending piracy of integrated circuits," *Computer*, vol. 43, no. 10, pp. 30–38, 2010.

[18] Y. Shen and H. Zhou, "Double dip: Re-evaluating security of logic encryption algorithms," in *Great Lakes Symp. on VLSI*, 2017.

[19] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "Cyclic obfuscation for creating sat-unresolvable circuits," in *Great Lakes Symp. on VLSI*, 2017.

[20] H. Zhou, R. Jiang, and S. Kong, "Cycsat: Sat-based attack on cyclic logic encryptions," in *36th Int. Conf. on Computer-Aided Design*, 2017.

[21] S. Roshanisefat, H. Mardani Kamali, and A. Sasan, "Srclock: Sat-resistant cyclic logic locking for protecting the hardware," in *Proceedings of the 2018 on Great Lakes Symposium on VLSI*. ACM, 2018, pp. 153–158.

[22] Y. Xie and A. Srivastava, "Delay locking: Security enhancement of logic locking against ic counterfeiting and overproduction," in *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM, 2017, p. 9.

[23] K. Zamiri Azar, H. Mardani Kamali, H. Homayoun, and A. Sasan, "Threats on logic locking: A decade later," in *Proceedings of the 2019 on Great Lakes Symp. on VLSI*, 2019.

[24] J. Gilmer *et al.*, "Neural message passing for quantum chemistry," in *Int. conf. on Machine Learning*, 2017.

[25] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Removal attacks on logic locking and camouflaging techniques," *IEEE Trans. on Emerging Topics in Computing*, 2017.

[26] D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura, and D. L. Dill, "Learning a SAT solver from single-bit supervision," *arXiv preprint arXiv:1802.03685*, 2018.

[27] J. Lei Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[28] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security analysis of logic obfuscation," in *Proceedings of the 49th Annual Design Automation Conf.*, 2012.

[29] 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 *2014 IEEE 20th Int. On-Line Testing Symp.*, 2014.

[30] 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, 2013.

[31] N. Sorensson and N. Een, "Minisat v1. 13-a sat solver with conflict-clause minimization," *SAT*, vol. 2005, no. 53, pp. 1–2, 2005.

[32] A. Biere, "Lingeling, plingeling and treengeling entering the sat competition 2013," 2013.

[33] G. Audemard and L. Simon, "GLUCOSE: a solver that predicts learnt clauses quality," *SAT Competition*, 2009.

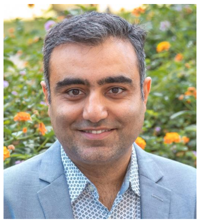
[34] R. Goldman, K. Bartleson, T. Wood, K. Kranen, C. Cao, V. Melikyan, and G. Markosyan, "Synopsys' open educational design kit: capabilities, deployment and future," in *Int. Conf. on Microelectronic Systems Education*, 2009.

[35] K. Shamsi *et al.*, "Appsat: Approximately deobfuscating integrated circuits," in *2017 IEEE Int. Symp. on Hardware Oriented Security and Trust*. IEEE, 2017.

**Rakibul Hassan** is a Ph.D student, currently conducting his research under the supervision of Dr. Sai Manoj P D, an Assistant professor at the Electrical and Computer Engineering Department, George Mason Universiy, Fairfax, VA, USA. Rakibul's present research interest includes computer architecture and IoT network security applying deep learning. He is also collaborating with another project, named, Adversarial attack on cloud computing. He has published his research at well-known conferences such as DATE, CASES, and ISQED. He received his B.Sc. degree in electrical and electronic engineering from Ahsanullah University of Science and Technology, Dhaka, Bangladesh in 2016. After completing his bachelor degree he worked for two years as a lecturer at Bangladesh University, Dhaka, Bangladesh. During that time, he published several peer-reviewed conference papers and one journal paper in IEEE Transactions on Computer-Aided Design.

**Gaurav Kolhe** is a Ph.D. student at Electrical and Computer Engineering Department of University of California, Davis,. His research interest are in the field of Heterogeneous Computing and Hardware Security and Trust, which spans the area of Computer Design and Embedded Systems. He is leading the research on ”Hybrid Spin Transfer Torque-CMOS Technology to Prevent Design Reverse Engineering”, a project funded by DARPA. He has previously worked for Information Sciences Institute, University of Southern California, where he had worked on DARPA’s ”Obfuscated Manufacturing for GPS (OMG)” program. He has received the ”Richard Newton Fellowship Stud dent award 2018” at Design Automation Conference. Gaurav received his Master’s degree in Computer Engineering in 2018 from George Mason University and BS degree in Electrical and Telecommunication Engineering in 2015 from Rajiv Gandhi College of Engineering, Nagpur, India.

**Setareh Rafatirad** is an Associate Professor in Department of Information Sciences and Technology at George Mason University. She obtained her M.Sc. and PhD in Computer Science from University of California, Irvine in 2009 and 2012. She received the ICDM 2019 Best Peper Award (9Paper Award. She received research funding from government agencies including NSF, DARPA, and AFRL for major projects. Her research interest covers several areas including Big Data Analytics, Data Mining, Knowledge Discovery and Knowledge Representation, IoT Security,and Applied Machine Learning. Currently, she is actively supervising multiple research projects focused on applying ML and Deep Learning techniques on different domains including House Price Prediction, Malware Detection, and Emerging big data application benchmarking and characterization on heterogeneous architectures.



**Houman Homayoun** is currently an Associate Professor in the Department of Electrical and Computer Engineering at University of California, Davis. Prior to that he was an Associate Professor in the Department of Electrical and Computer Engineering at George Mason University (GMU). From 2010 to 2012, he spent two years at the University of California, San Diego, as NSF Computing Innovation (CI) Fellow awarded by the CRA-CCC. Houman graduated in 2010 from University of California, Irvine with a Ph.D. in Computer Science. He was a recipient of the four-year University of California, Irvine Computer Science Department chair fellowship. Houman received the MS degree in computer engineering in 2005 from University of Victoria and BS degree in electrical engineering in 2003 from Sharif University of Technology. He is currently the director of UC Davis Accelerated, Secure, and Energy-Efficient Computing Laboratory (ASEEC). Houman conduct research in hardware security and trust, data-intensive computing and heterogeneous computing, where he has published more than 100 technical papers in the prestigious conferences and journals on the subject and directed over $8M in research funding from NSF, DARPA, AFRL, NIST and various industrial sponsors. He received several best paper awards and nominations in various conferences including GLSVLSI 2016, ICCAD 2019, and ICDM 2019. Houman served as Member of Advisory Committee, Cyber security Research and Technology Commercialization (R&TC) working group in the Commonwealth of Virginia in 2018. Since 2017 he has been serving as an Associate Editor of IEEE Transactions on VLSI. He was the technical program co-chair of GLSVLSI 2018 and the general chair of 2019 conference.

**Sai Manoj P D** (S’13-M’15) is an assistant professor at George Mason University. Prior joining to George Mason University (GMU) as an assistant professor, he served as research assistant professor and postdoctoral research fellow at GMU and was a postdoctoral research scientist at the System-on-Chip group, Institute of Computer Technology, Vienna University of Technology (TU Wien), Austria. He received his Ph.D. in Electrical and Electronics Engineering from Nanyang Technological University, Singapore in 2015. He received his Masters in Information Technology from International Institute of Information Technology Bangalore (IIITB), Bangalore, India in 2012. His research interests include on-chip hardware security, neuromorphic computing, adversarial machine learning, self-aware SoC design, image processing and time-series analysis, emerging memory devices and heterogeneous integration techniques. He won best paper award in Int. Conf. On Data Mining 2019, and his works were nominated for best paper award in prestigious conferences such as Design Automation & Test in Europe (DATE) 2018, International Conference on Consumer Electronics 2020, and won Xilinx open hardware contest in 2017 (student category). He is the recipient of the “A. Richard Newton Young Research Fellow” award in Design Automation Conference, 2013.