A Neural Network-Based Cognitive Obfuscation 
Toward Enhanced Logic Locking

Rakibul Hassan, Graduate 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 its 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 (NN)-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 the 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 multilayer NN 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 obfuscation SAT attacks.

Index Terms—Logic locking, message-passing neural network (MPNN), satisfiability (SAT), SAT-hard.

I. INTRODUCTION

W ith the semiconductor industries inclining toward the 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, overproduction, 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 camouflage, and logic locking also known as 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 the Boolean satisfiability (SAT)-based attack [6], also known as the “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. To implement SAT attack, the attacker needs access to: 1) an obfuscated netlist of IC (obtained after delayering IC or constructed from layout) and 2) 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 (CNF1), 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 [7].

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 [8]. Recent literature reported a signal probability skew (SPS) attack [9] against Anti-SAT defense [8] which can break the Anti-SAT defense within few minutes.

1A CNF is a conjunction (i.e., AND) of one or more clauses, where a clause is a disjunction (i.e., OR) of literals.
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 [10]. Previous works [8], [9] 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 exclude parameters that were not explored in the literature. To address these concerns, we introduce SATConda, equipped with a CNF generator that can convert the provided SAT prone CNF into SAT-hard 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 a neural network (NN) with bipartite message-passing mechanism to cognitively learn and determine the properties of a CNF and distinguish SAT and SAT-hard problems. Once learned, the amount of clauses added or the perturbations are introduced cognitively, which can be controlled to determine the tradeoff 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 a 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 as follows.

1) 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 an NN 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.

2) We successfully defend the standard benchmarks against the existing attacks such as the SAT attack [6] by introducing a SAT-hard block and encrypting that block 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 the SAT attack utilized the NN model to learn the SAT and SAT-hard clauses and encrypt the circuit under the minimal overhead constraint. The idea of exploring deep NN in circuit obfuscation, especially for converting SAT to SAT-hard, is unexplored and is novel [13]. 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.

II. BACKGROUND

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

A. Logic Locking

The logic locking mechanism is implemented in a design by adding additional gates also known as “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 key gates to decode the actual functionality of the design.

Fig. 1(a) depicts the original circuit and Fig. 1(b) shows a logic locked circuit of the same C17 circuit from the ISCAS’85 benchmark [14]. 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 Fig. 1(b), if one assigns (k2, k1, K0) as (1, 0, 1), only then the circuit will function as it is intended to. The complexity of determining the key inputs will increase exponentially with the number of key gates when the attacker performs a brute-force search.

---

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 [11], [12] for a given CNF is the SAT 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 h.
B. SAT Attack

Despite logic locking being successful in securing the IC from RE, the Boolean SAT-based attack commonly known as the 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 countermeasures 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 the following.

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

2) Attack Methodology: The SAT attack generates a carefully crafted input patterns and observed the corresponding output from the activated functional chip. The goal of the 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 (DIPs). 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 [16] is one of the preliminary notable works 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 [8] 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 [17].

In another work, an 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 technique suffers from a large area overhead since the implementation of the AES circuit requires a significant number of logic gates [8].

In [18], a SAT-resilient cyclic obfuscated circuit design was proposed by adding dummy paths to the encrypted circuit which make the combinational loop nonreducible. This defense mechanism was prone to another type of SAT-based attack named CycSAT [19], which can effectively decrypt the cyclic encryption.

The CycSAT attack [19] performs a preprocessing step so that it generates a cycle avoidance clause list. The more clauses that are responsible for the SAT solver to be trapped in an infinite loop, the more preprocessing is done by the CycSAT. In [20], a number of methods are proposed to increase the cycles in a netlist exponentially to defense against the CycSAT attack. They proposed a technique to insert the “Super Cycles” block that causes an exponential increase in the cycles in a given netlist and leads to an exponential increase in the runtime of the CycSAT preprocessing 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 predefined timing constraints are locked using key gates and delay gates, respectively [21]. 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 [22].

Unlike the existing defenses discussed above, our proposed methodology utilizes a cognitive approach by utilizing NN 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 the 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 a 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 NN (MPNN) [23] framework that learns the SAT and SAT-hard distribution from a given number of SAT problems. Fig. 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 NN being trained and evaluated,
Fig. 2. Architecture of the proposed SATConda.

the model outputs the value of seed1 and seed2 for each benchmark circuit while trained on the SAT and SAT-hard problems for different encryption circuits as input. Here, the NN output is the continuous value prediction from 0 to 1 for both seed1 and seed2. The predicted value of seed1 and seed2, thus, set the initialization for the SAT-hard block generator. For both the seed1 and seed2 values, our NN predicts a value ranges from 0 to 1 for a given circuit. The significance of seed1 and seed2 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 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 NN from which the seeds for clause generation is extracted. The details of the NN are discussed in the next subsections.

For 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 NN 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 seed1 and seed2 variables (see Fig. 2). The values of seed1 and seed2 are determined by fitting the learned distribution to Bernoulli and Geometric distributions, respectively. A randomly generated decimal number between 0 and 1 is then compared with the seed1 value (see Algorithm 1, line 5). Another variable Literal_base is assigned as integer 1 if the decimal number is greater than the seed1 otherwise integer 2 (see Algorithm 1, lines 6–8). Then, the generator draws samples from a geometric distribution at a probability of seed2 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 SAT and keeps adding clauses until the new obfuscated CNF becomes SAT-hard.

A function named GENERATE_CLAUSE generates each clause using the learned parameters (discussed
Algorithm 1 SATConda Algorithm

Input : solve_clauses, seed_1, seed_2
Output : SAT – hard – cnf
1: is_sat := solve(solve_clauses)
2: if is_sat == True then
3:   while true do
4:     rand := gen_decimal(0 – 1)
5:     if rand < seed_1 then
6:       lit_base := 1
7:     else
8:       lit_base := 2
9:   end if
10:  rand_geo = rand.geometric(seed_2)
11:  literal = lit_base + rand_geo
12:  new_clause := generate_clause(n_var, literal)
13:  solve_clauses+ = new_clause
14:  is_sat := solve(solve_clauses)
15:  if is_sat == True then
16:    solve_clauses+ = new_clause
17:  else
18:    break
19:  end if
20: end while
21: solve_clauses+ = new_clause
22: end if
23: function GENERATE_CLAUSE(n_var, literal)
24:  array_size := minimum(n_var, literal)
25:  clause_gen := gen.rand_array(n_var, array_size)
26:  rand’ := gen_decimal(0 1)
27:  if rand < 0.5 then
28:    new_clause := clause_gen + 1
29:  else
30:    new_clause := –(clause_gen + 1)
31:  end if
32:  return new_clause
33: end function
34: SAT – hard – cnf := solve_clauses

in Section IV-C) and returns a SAT-hard block. Algorithm 1, lines 23–33 show this function. Then, we integrate that block to the original circuit. We discuss this integration process in Section IV-B.

The range for 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 [24]. 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, 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.

Fig. 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 pins, 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’d 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 a 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 [24] will not succeed to identify the SAT-hard block. Finally, with the proper key value, the original functionality of the hidden output pin is retrieved.

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 an MPNN similar to [25] 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 an NN 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 [25]. There are two advantages of deploying an NN in the 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. Second, to protect the existing locking techniques from the powerful SAT attacks, a cognitive countermeasure 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 are 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 [23]. The message passing starts by passing a message to a clause from its neighboring literals. In the next step, a literal gets the message from its neighboring 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 \( L_{\text{init}} \) and clause vector \( C_{\text{init}} \) are fed to a three-layer fully connected MPNN. The output from the MPNN, \((L_{\text{msg}}, C_{\text{msg}}, L_{\text{sat}})\), is fed to a two-layer 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 is established by connecting edges among them. The LSTM network [26] updates the literals \( L_t^{i+1} \) and clauses \( C_t^{i+1} \) at each iteration, as follows:

\[
C_u([M^T L_{\text{msg}}(L^i)]) \rightarrow C_t^{i+1} \tag{1}
\]

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

\[
L_u([\text{Flip}(L^i), MC_{\text{msg}}(C_t^{i+1})]) \rightarrow L_t^{i+1} \tag{3}
\]

\[
L_u([L_h^i]) \rightarrow L_h^{i+1}. \tag{4}
\]

Equations (1) and (3) represent one iteration of the MPNN network where clause and literals updated their current embeddings. In (1), clause \( C \) updates its embedding based on messages passed from the neighboring literals whereas in (3), a literal updates its current embedding after receiving a message from the neighboring clause. Equations (2) and (4) represent the hidden layer of the LSTM network for updating the clauses and literals. Thus, the 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 a 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 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 \( C_m = \{C_1, C_2, \ldots, C_m\} \) and \( C_n = \{C'_1, C'_2, \ldots, C'_n\} \). 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 SAT. 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 \((\text{SAT, SAT} – \text{hard})\) 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 10 000 SAT problems (5000 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 underfitting. All the training and SAT-hard clause generation steps are offline and, thus, 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.

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 [25] 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 the ISCAS'85 benchmark, one algorithm was proposed by Rajendran et al. [27] 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. [28] that inserts AND/OR gates (referred to as “IOLTS’14”) at multiple chosen locations of the circuit.

For the ISCAS'89 benchmark, we used the IOLTS’14 obfuscation [28] technique. It needs to be noted that the algorithm [6] was unable to encrypt the ISCAS'89 circuit using DAC’12 algorithm despite executing it for three days. Instead of DAC’12, we applied a similar technique that is also XOR-based logic locking (referred to as “TOC’13”) algorithm [29] to obfuscate the 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 [27], AND/OR-based logic locking IOLTS’14 [28], and XOR-based logic locking ToC’13 [29], 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 varies. 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. 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 the hardware security domain, we have also verified the SAT of CNF using three different SAT solvers, MiniSAT [30], Lingeling [31], and Glucose [32]. 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 the Synopsys Design Compiler, Version: L-2016.03-SP3. SAED 90-nm EDK Digital Standard Cell Library [33] 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.

### Table I

| ISCAS'85 Circuit | ISCAS'89 Circuit |
|------------------|------------------|
| Circuit Name     | #Inputs | #Outputs | # Gates | Circuit Name | #Inputs | #Outputs | # Gates |
| c432             | 56      | 7        | 160     | s382        | 24      | 27       | 392     |
| c499             | 41      | 32       | 202     | s400        | 24      | 27       | 414     |
| c860             | 60      | 26       | 383     | s526a       | 24      | 27       | 494     |
| c3155            | 41      | 32       | 546     | s526        | 3       | 6        | 141     |
| c1908            | 33      | 25       | 840     | s553        | 43      | 29       | 950     |
| c6270            | 233     | 140      | 1193    | s488        | 14      | 25       | 843     |
| c3540            | 50      | 22       | 1669    | s5378       | 214     | 228      | 3183    |
| c5315            | 178     | 123      | 2307    | s13207      | 700     | 790      | 11248   |
| c7952            | 207     | 108      | 3512    | s15850      | 611     | 684      | 13192   |
|                  |         |          |         | s35952      | 1763    | 1728     | 31833   |

### Table II

| Circuit Name | IOLTS’14 [28] | DAC’12 [27] |
|--------------|---------------|-------------|
| Before Conversion | After Conversion | Before Conversion | After Conversion |
| c432 | 0.033 | timeout | 0.026 | timeout |
| c499 | 0.000 | timeout | 0.070 | timeout |
| c860 | 0.061 | timeout | 0.094 | timeout |
| c3155 | 0.042 | timeout | 0.312 | timeout |
| c1908 | 0.049 | timeout | 0.518 | timeout |
| c6270 | 0.544 | timeout | timeout | ** |
| c3540 | 0.323 | timeout | 3.264 | timeout |
| c5315 | 0.826 | timeout | 9.013 | timeout |
| c7952 | 0.467 | timeout | 26.75 | timeout |

timeout = 10 hours

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

### 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 [27], [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 Rajendran et al. [27] and Dupuis et al. [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 [27] and AND/OR-based logic locking [28], SATConda successfully defends the SAT attack for all the circuits. The timeout for the SAT attack is set to 10 h in our experiments. In case of the c2670 circuit, encryption in [27] itself resists against the SAT attack; thus, SATConda does not introduce additional circuit. Table III presents the SAT 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 the SAT attack in [6], we have also evaluated the robustness with SATConda in terms of SAT-hardness against other SAT attacks such as AppSAT [34]. Table IV represents the SATConda evaluation on the AppSAT attack [34] 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 the AppSAT attack for ISCAS’89 benchmark circuits. Similar resiliency against the AppSAT attack is observed with the 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 the basis for most of the present-day SAT-based deobfuscation attacks for evaluation. The SAT of the encryption techniques was verified against three different SAT solvers, MiniSAT [30], Lingeling [31], and Glucose [32]. Table VI presents the SAT 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 [30]–[32]. Table VI also shows that once the IC/IP design was 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 SAT of the same ISCAS’85 benchmark circuits on DAC’12 logic locking [27] before and after the conversion through SATConda. Similar to the IOLTS’14 logic locking [28], the DAC’12 logic locking [27] 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. Tables VIII and IX present the SAT

### Table III

**SATConda Evaluation on SAT Attack [6] for ISCAS’89 Circuits With Different Encryption Techniques**

| Circuit Name | IOLTS’14 [28] | ToC’13 [29] |
|--------------|---------------|-------------|
|              | Before Conversion | After Conversion | Before Conversion | After Conversion |
| c382         | 0.006 time     | timeout       | 0.006 time       | timeout       |
| s400         | 0.006 time     | timeout       | 0.006 time       | timeout       |
| s516n        | 0.0087 time    | timeout       | 0.008 time       | timeout       |
| s526         | 0.0084 time    | timeout       | 0.008 time       | timeout       |
| s641         | 0.009 time     | timeout       | 0.0092 time      | timeout       |
| s953         | 0.017 time     | timeout       | 0.017 time       | timeout       |
| s1488        | 0.048 time     | timeout       | 0.0344 time      | timeout       |
| s5378        | 0.070 time     | timeout       | 0.072 time       | timeout       |
| s13207       | 0.252 time     | timeout       | 0.232 time       | timeout       |
| s13350       | 0.328 time     | timeout       | 0.336 time       | timeout       |
| s35932       | 5.831 time     | timeout       | 5.890 time       | timeout       |

timeout = 10 hours

### Table IV

**SATConda Evaluation on AppSAT Attack [34] for ISCAS’85 Circuits**

| Circuit Name | IOLTS’14 [28] | ToC’13 [29] |
|--------------|---------------|-------------|
|              | Before Conversion | After Conversion | Before Conversion | After Conversion |
| c432         | 0.055 time     | timeout       | 0.058 time       | timeout       |
| c499         | 0.085 time     | timeout       | 0.086 time       | timeout       |
| c880         | 0.1598 time    | timeout      | 0.20 time        | timeout       |
| c1335        | 0.098 time     | timeout       | 0.219 time       | timeout       |
| c1908        | 0.13 time      | timeout       | 3.945 time       | timeout       |
| c2670        | 1.712 time     | timeout       | 1.380 time       | timeout       |
| c3540        | 1.968 time     | timeout       | 4.921 time       | timeout       |
| c5313        | 2.885 time     | timeout       | 1.71 time        | timeout       |
| c7552        | 4.506 time     | timeout       | 2.94 time        | timeout       |

timeout = 10 hours

### Table V

**SATConda Evaluation on AppSAT Attack [34] for ISCAS’89 Circuits With Different Encryption Schemes**

| Circuit Name | IOLTS’14 [28] | ToC’13 [29] |
|--------------|---------------|-------------|
|              | Before Conversion | After Conversion | Before Conversion | After Conversion |
| c382         | 0.006 time     | timeout       | 0.032 time       | timeout       |
| s400         | 0.007 time     | timeout       | 0.033 time       | timeout       |
| s526         | 0.089 time     | timeout       | 0.044 time       | timeout       |
| s526         | 0.090 time     | timeout       | 0.045 time       | timeout       |
| s641         | 0.112 time     | timeout       | 0.053 time       | timeout       |
| s953         | 0.157 time     | timeout       | 0.077 time       | timeout       |
| s1488        | 0.316 time     | timeout       | 0.117 time       | timeout       |
| s5378        | 0.541 time     | timeout       | 0.268 time       | timeout       |
| s13207       | 1.52 time      | timeout       | 0.548 time       | timeout       |
| s13350       | 2.265 time     | timeout       | 0.887 time       | timeout       |
| s35932       | 17.882 time    | timeout       | 8.138 time       | timeout       |

### Table VI

**SATConda Evaluation Using IOLTS’14 [28] Encryption on Different SAT Solvers for ISCAS’85 Benchmarks**

| Circuit Name | miniSAT [30] | Lingeling [31] | Glucose [32] |
|--------------|--------------|----------------|--------------|
|              | Before Conversion | After Conversion | Before Conversion | After Conversion |
| c432         | 0.0039 time   | timeout         | 0.0015 time   | timeout         |
| c499         | 0.0026 time   | timeout         | 0.0012 time   | timeout         |
| c880         | 0.0024 time   | timeout         | 0.0014 time   | timeout         |
| c1335        | 0.0018 time   | timeout         | 0.0005 time   | timeout         |
| c1908        | 0.0041 time   | timeout         | 0.0031 time   | timeout         |
| c2670        | 0.0049 time   | timeout         | 0.0003 time   | timeout         |
| c3540        | 0.0060 time   | timeout         | 0.0061 time   | timeout         |
| c5313        | 0.0086 time   | timeout         | 0.0094 time   | timeout         |
| c7552        | 0.0095 time   | timeout         | 0.0110 time   | timeout         |

X= Corresponding CNF was SAT-hard/indistinguishable by the SAT solver

### Table VII

**SATConda Evaluation Using DAC’12 Encryption [27] on Different SAT Solvers for ISCAS’85 Benchmarks**

| Circuit Name | miniSAT [30] | Lingeling [31] | Glucose [32] |
|--------------|--------------|----------------|--------------|
|              | Before Conversion | After Conversion | Before Conversion | After Conversion |
| c432         | 0.0049 time   | timeout         | 0.004249 time | timeout         |
| c499         | 0.005691 time | timeout         | 0.004277 time | timeout         |
| c880         | 0.006145 time | timeout         | 0.005559 time | timeout         |
| c1335        | 0.006175 time | timeout         | 0.006171 time | timeout         |
| c1908        | 0.006221 time | timeout         | 0.007626 time | timeout         |
| c2670        | 0.005807 time | timeout         | 0.008887 time | timeout         |
| c3540        | 0.001801 time | timeout         | 0.009223 time | timeout         |
| c5313        | 0.005401 time | timeout         | 0.009447 time | timeout         |
| c7552        | 0.010083 time | timeout         | 0.010702 time | timeout         |

X= Corresponding CNF was SAT-hard/indistinguishable by the SAT solver
result for ISCAS'89 circuits using the IOLTS'14 logic locking [28] and TOC'13 logic locking [29] 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 [27] and AND/OR-based logic locking [28]) and compare them with the overhead of the proposed SATConda applied on top of existing encryption techniques to successfully defends SAT-attacks (the original XOR/XNOR-based logic locking [27] and AND/OR-based logic locking [28] fail to secure against SAT attacks).

Both the DAC'12 logic locking [27] and IOLTS'14 logic locking [28] introduce an average 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 RE the IP. Further encrypting with SATConda incurs additional overhead of about 57% and about 42% on average compared to [28] and [27], respectively, as shown in Table X, but guarantees security. In a similar manner, the proposed SATConda incurs around 54% and 36% power overhead on an average compared to [28] and [27], 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 [28] and TOC'13 logic locking [29]) with our proposed method. Here, the TOC'13 [29] and IOLTS'14 [28] 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 [28] and [29], respectively. Similarly, the proposed SATConda incurs around 68% and 74% power overhead on an average compared to [28] and [29], 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 nonlinear 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 versus 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 nonlinear 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 larger 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 learned by the MPNN, different training data can lead to different learned 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 the best fit model (for seed $1 = 0.9$ and seed $2 = 0.9$—leading to lowest overhead), the mid-fit model (for seed $1 = 0.5$ and seed $2 = 0.5$), and the worst fit model (for seed $1 = 0.3$ and seed $2 = 0.4$). Fig. 4 depicts the area overhead (%), Fig. 5 shows the power overhead (%), and Fig. 6 illustrates the delay overhead for different models.
TABLE X

| Circuit | IOLTS-14 [28] | SAT-Cond+28head | DAC12-27 | SAT-Cond+27head | DAC12-27 |
|---------|----------------|----------------|----------|----------------|----------|
| c432    | 1.05879        | 2.07920        | 96.38    | 1.11392        | 2.15815  |
| c499    | 1.91888        | 3.04498        | 82.69    | 1.19786        | 2.80702  |
| c880    | 1.81321        | 2.80117        | 77.36    | 1.75486        | 2.75661  |
| c1355   | 2.05121        | 3.68381        | 79.59    | 2.15495        | 3.25643  |
| c1908   | 2.17010        | 3.53322        | 63.74    | 2.62810        | 3.58738  |
| c3540   | 4.97122        | 7.25889        | 46.02    | 5.71699        | 7.15122  |
| c5315   | 7.27901        | 10.77000       | 47.96    | 8.66090        | 12.23105 |
| c7552   | 9.69755        | 14.46548       | 53.20    | 10.90076       | 17.06117 |
| Average | 57.7           | 42.8           | 34.65    | 37.3           | 24.11    |

Fig. 4. Area overhead analysis for ISCAS’85 benchmark circuit for different cases: (a) area overhead on IOLTS’14 encryption [28] and (b) area overhead on DAC’12 encryption [27].

TABLE XI

| Circuit | IOLTS-14 [28] | SAT-Cond+28head | DAC12-27 | SAT-Cond+27head | DAC12-27 |
|---------|----------------|----------------|----------|----------------|----------|
| c432    | 0.10           | 1.36           | 34.65    | 1.12           | 1.39     |
| c499    | 2.37           | 2.9            | 12.84    | 2.73           | 3.76     |
| c880    | 2.9            | 3.75           | 29.31    | 3.11           | 3.41     |
| c1355   | 2.6            | 3.44           | 32.31    | 2.71           | 3.49     |
| c1908   | 3.72           | 5.01           | 34.68    | 3.52           | 4.91     |
| c3540   | 4.83           | 5.58           | 15.05    | 4.84           | 5.85     |
| c5315   | 5.55           | 4.73           | 33.24    | 5.22           | 7.01     |
| c7552   | 4.86           | 4.99           | 2.67     | 7.05           | 7.12     |
| Average | 24.34          | 24.53          |          |                |          |

Fig. 5 depicts the power overhead (%) for the ISCAS’85 benchmark circuit for different models where Fig. 5(a) shows the power overhead on IOLTS’14 encryption and Fig. 5(b) shows the power overhead on DAC’12 encryption. On average, the power overhead for the best fit, mid-fit, and the worst fit models for the IOLTS’14 encryption is about 57%, 288%, and 584%, respectively, and that for the DAC’12 encryption is about 51%, 295%, and 503%, respectively, irrespective of the model fit used for generating or perturbing the clauses, each circuit shows security against SAT attack.

Fig. 6 depicts the delay overhead (%) for ISCAS’85 benchmark circuit for different models where Fig. 6(a) shows the delay overhead on IOLTS’14 encryption and Fig. 6(b) shows the delay overhead on DAC’12 encryption. On average, the delay overhead for the best fit, mid-fit, and the worst fit models for the IOLTS’14 encryption is about 44%, 190%, and 360%, respectively, and that for the DAC’12 encryption is about 37%, 177%, and 271%, respectively.

A similar trend was observed for the ISCAS’89 benchmark circuits on area, power, and delay overhead using our different models, not presented for the purpose of conciseness.
In summary, Figs. 4–6 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 a large number of variables. In return, the best fit model adds a 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 an NN-based SAT-hard problem generator,
SATConda, to achieve an enhanced obfuscation technique for the hardware security regime. We observed that the integration of a SAT-hard block with a dummy output pin replacing an original output pin (keeping the same functionality) deceives the SAT attack. Our framework was evaluated on the state-of-the-art benchmarks, such as ISCAS’85 and ISCAS’89 using two state-of-the-art encryption algorithms. We validated our model with the SAT attack, AppSAT attack, and three other traditional SAT solvers.

### REFERENCES

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

[2] M. Rostami, F. Koushanfar, and R. Karri, “A primer on hardware security: Models, methods, and metrics,” Proc. IEEE, vol. 102, no. 8, pp. 1283–1295, Aug. 2014.

[3] J. J. Rajendran, O. Sinanoglu, and R. Karri, “Is split manufacturing secure?” in Proc. Conf. Design Autom. Test Eur., 2013, pp. 1259–1264.

[4] M. Yasin, J. J. Rajendran, O. Sinanoglu, and R. Karri, “On improving the security of logic locking,” IEEE Trans. Comput-Aided Design Integr. Circuits Syst., vol. 35, no. 9, pp. 1411–1424, Sep. 2016.

[5] M. Yasin, B. Mazumdar, J. J. V. Rajendran, and O. Sinanoglu, “SARLock: SAT attack resistant logic locking,” in Proc. Int. Symp. Hardw. Orient. Security Trust, 2016, pp. 236–241.

[6] P. Subramanyan, S. Ray, and S. Malik, “Evaluating the security of logic encryption algorithms,” in Proc. Int. Symp. Hardw. Orient. Security Trust, 2015, pp. 137–143.

[7] J. Franco and J. Martin, Handbook of Satisfiability Frontiers in Artificial Intelligence and Applications. Amsterdam, The Netherlands: IOS Press, 2009.

[8] Y. Xie and A. Srivastava, “Mitigating SAT attack on logic locking,” in Proc. Int. Conf. Cryptograph. Hardw. Embedded Syst., 2016, p. 590.

[9] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, “Security analysis of anti-SAT,” in Proc. Asia South Pac. Design Autom. Conf., 2017, pp. 342–347.

[10] G. Kolhe, S. Manoj PD, S. Rafatirad, H. Mahmoodi, A. Sasan, and H. Homayoun, “On custom LUT-based obfuscation,” in Proc. Great Lakes Symp. VLSI, 2019, pp. 477–482.

[11] 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 Proc. 36th Annu. Design Autom. Conf., 2019, pp. 1–6.

[12] R. Hassan, G. Kohle, S. Rafatirad, H. Homayoun, and S. M. P. Dinakarao, “A cognitive SAT to SAT-hard clause translation-based logic obfuscation,” in Proc. Design Autom. Test Eur. Conf. Exhib. (DATE), Grenoble, France, 2021, pp. 1172–1177.

[13] R. Hassan, G. Kolhe, S. Rafatirad, H. Homayoun, and S. M. P. Dinakarao, “SATConda: SAT to SAT-hard clause translator,” in Proc. 21st Int. Symp. Qual. Electron. Design, IEEE, 2020, pp. 155–160.

[14] M. C. Hansen, H. Yalcin, and J. P. Hayes, “Unveiling the ISCAS-85 benchmarks: A case study in reverse engineering,” IEEE Design Test Comput., vol. 16, no. 3, pp. 72–80, 1999.

[15] S. Dupuis and M.-L. Flottes, “Logic locking: A survey of proposed methods and evaluation metrics,” J. Electron. Test., vol. 35, pp. 273–291, May 2019.

[16] J. A. Roy, F. Koushanfar, and I. L. Markov, “Ending piracy of integrated circuits,” in Proc. Int. Symp. VLSI, 2010, pp. 30–38, Oct. 2010.

[17] Y. Shen and H. Zhou, “Double DIP: Re-evaluating security of logic encryption algorithms,” in Proc. Great Lakes Symp. VLSI, 2017, pp. 173–178.

[18] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, “Cyclic obfuscation for creating SAT-unresolvable circuits,” in Proc. Great Lakes Symp. VLSI, 2017, pp. 49–56.

[19] S. Roshanisefat, H. M. Kamali, and A. Sasan, “SRCLock: SAT-resistant cyclic logic locking for protecting the hardware,” in Proc. Great Lakes Symp. VLSI, 2018, pp. 158–153.

[20] Y. Xie and A. Srivastava, “Delay locking: Security enhancement of logic locking against ic counterfeiting and overproduction,” in Proc. 54th Annu. Design Autom. Conf., 2017, p. 9.

[21] K. Z. Azar, H. M. Kamali, H. Homayoun, and A. Sasan, “Threats on logic locking: A decade later,” in Proc. Great Lakes Symp. VLSI, 2019, pp. 471–476.

[22] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, “Neural message passing for quantum chemistry,” in Proc. Int. Conf. Mach. Learn., 2017, pp. 1263–1272.

[23] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, “Removal attacks on logic locking-based obfuscation,” IEEE Trans. Emerg. Topics Comput., vol. 8, no. 2, pp. 517–532, Apr–Jun. 2021.

[24] D. Selsam, L. Mamm, B. Bünz, P. Liang, D. L. Dill, “Learning a SAT solver from single-bit supervision,” 2018, arXiv:1802.03685.

[25] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization,” 2016, arXiv:1607.06450.

[26] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, “Security analysis of logic obfuscation,” in Proc. 49th Annu. Design Autom. Conf., 2012, pp. 83–89.

[27] 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 Proc. IEEE 20th Int. On-Line Test. Symp., 2014, pp. 49–54.

[28] J. Rajendran et al., “Fault analysis-based logic encryption,” IEEE Trans. Comput., vol. 64, no. 2, pp. 410–424, Feb. 2015.

[29] N. Sorensson and N. Een, MiniSAT V1.6—A SAT Solver With Conflict-Clause Minimization, vol. 2005, SAT, New York, NY, USA, 2005, pp. 1–2.

[30] A. Biere, “Lingeling, plingeling and treengeling entering the SAT competition 2013,” in Proc. SAT Competition, 2013, pp. 51–52.

[31] G. Audemard and L. Simon, “GLUCOSE: A solver that predicts learnt clauses quality,” in Proc. SAT Competition, 2013, pp. 83–89.

[32] R. Goldman et al., “Synopsys open educational design kit: Capabilities, deployment and future,” in Proc. Int. Conf. Microelectron. Syst. Educ., 2009, pp. 20–24.

[33] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, “AppSAT: Approximately deobfuscating integrated circuits,” in Proc. IEEE Int. Symp. Hardw. Orient. Security Trust, McLean, VA, USA, 2017, pp. 95–100.
Gaurav Kolhe (Student Member, IEEE) received the B.S. degree in electrical and telecommunication engineering from Rajiv Gandhi College of Engineering, Nagpur, India, in 2015, and the master’s degree in computer engineering from George Mason University, Fairfax, VA, USA, in 2018. He is currently pursuing the Ph.D. degree with the Electrical and Computer Engineering Department, University of California at Davis, Davis, CA, USA.

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, Los Angeles, CA, USA, where he had worked on DARPA's “Obfuscated Manufacturing for GPS (OMG)” program. His research interests are in the field of heterogeneous computing and hardware security and trust, which spans the area of computer design and embedded systems.

Mr. Kolhe has received the “Richard Newton Fellowship Student Award 2018” at Design Automation Conference.

Setareh Rafatirad (Senior Member, IEEE) received the M.Sc. and Ph.D. degrees in computer science from the University of California, Irvine, CA, USA, in 2009 and 2012, respectively.

She is a Tenure Track Assistant Professor of Teaching with the Department of Computer Science, University of California at Davis, Davis, CA, USA. Before joining UC Davis, she was an Associate Professor with the Department of Information Sciences and Technology, George Mason University, Fairfax, VA, USA. She is currently 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. Her research interests cover several areas, including big data analytics, data mining, knowledge discovery and knowledge representation, IoT security, and applied machine learning.

Dr. Rafatirad received the ICDM 2019 Best Paper Award (9 Paper Award). She received research funding from government agencies, including NSF, DARPA, and AFRL for major projects.

Houman Homayoun (Senior Member, IEEE) received the B.S. degree in electrical engineering from Sharif University of Technology, Tehran, Iran, in 2003, the M.S. degree in computer engineering from the University of Victoria, Victoria, BC, Canada, in 2005, and the Ph.D. degree in computer science from the University of California, Irvine, CA, USA, in 2010.

He is currently an Associate Professor with the Department of Electrical and Computer Engineering, University of California at Davis, Davis, CA, USA. Prior to that, he was an Associate Professor with the Department of Electrical and Computer Engineering, George Mason University (GMU), Fairfax, VA, USA. From 2010 to 2012, he spent two years with the University of California, San Diego, CA, USA, as an NSF Computing Innovation (CI) Fellow awarded by the CRA-CCC. He is currently the Director of UC Davis Accelerated, Secure, and Energy-Efficient Computing Laboratory, GMU. He conducts 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 U.S. $8M in research funding from NSF, DARPA, AFRL, NIST, and various industrial sponsors.

Dr. Homayoun received several Best Paper Awards and nominations in various conferences, including GLSVLSI 2016, ICCAD 2019, and ICDM 2019. He was a recipient of the Four-Year University of California, Irvine Computer Science Department Chair Fellowship. He served as a 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 VERY LARGE SCALE INTEGRATION (VLSI) Systems. He was the Technical Program Co-Chair of GLSVLSI 2018 and the General Chair of 2019 conference.

Sai Manoj Pudukotai Dinakarrao (Member, IEEE) received the master’s degree in information technology from the International Institute of Information Technology Bangalore (IIITB), Bengaluru, India, in 2012, and the Ph.D. degree in electrical and electronics engineering from Nanyang Technological University, Singapore, in 2015.

He is an Assistant Professor with George Mason University (GMU), Fairfax, VA, USA. Prior joining to GMU as an Assistant Professor, he served as a Research Assistant Professor and a Postdoctoral Research Fellow with GMU and was a Postdoctoral Research Scientist with the System-on-Chip Group, Institute of Computer Technology, Vienna University of Technology, Vienna, Austria. 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.

Dr. Dinakarrao won the Best Paper Award in Int. Conf. On Data Mining 2019, and his works were nominated for the Best Paper Award in prestigious conferences, such as Design Automation and Test in Europe (DATE) 2018 and International Conference on Consumer Electronics 2020, and won the 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.