Thwarting GNN-Based Attacks Against Logic Locking

Armin Darjani, Nima Kavand, Shubham Rai, *Member, IEEE*, and Akash Kumar, *Senior Member, IEEE*

***Abstract*— The globalization of the IC manufacturing flow has exposed intellectual property (IP) to many untrustworthy entities. As a result, security should be considered a new paradigm in designing circuits to protect the integrity and confidentiality of the IP. Logic locking is a holistic design-for-trust (DFT) technique that can protect circuits against IP piracy and reverse engineering. However, a large body of recent research has demonstrated successful methods of recovering the secret key and restoring the original functionality of existing locking systems. Although SAT attack has been a de facto technique to break the logic locking, the threat model and efficiency of this attack have been questioned recently. To overcome these shortcomings, researchers have proposed powerful structural attacks that break the locked circuits without the need for functionally unlocked circuits (Oracle). Among structural attacks, machine learning (ML)-based attacks are the most potent attacks as they harness the power of neural networks to learn traces of the locking structures and use this knowledge to reverse back and neutralize the locking scheme. Among ML approaches, GNN (graph neural networks)-based attacks are shown to be the most capable tools that attackers can employ as they exploit graph structures inherent to a circuit's netlist. In this paper, (1) We discuss the inherent structural weaknesses of the logic locking techniques. (2) Knowing these weaknesses, we investigate the challenges of protecting circuits against GNN-based attacks. (3) We propose GNN-resilient Interconnect-based obfuscation (GRIN) and GNN-resilient Gate-based Obfuscation (GREGO) logic locking schemes with learning resilient structures. We evaluate our secure schemes using ISCAS-85 and ITC-99 benchmarks and provide comprehensive security and overhead analysis of our proposed schemes.**

***Index Terms*— Logic locking, structural attacks, GNN-based attacks, learning-resilient.**

I. Introduction

THE ever-growing complexity of the IC has steeply increased the cost of IC manufacturing which propelled companies to go fabless over the years. As a result, various entities from different regions of the globe may carry out the different stages of the IC manufacturing flow. This outsourcing and globalization have brought many perils regarding the integrity and confidentiality of IPs that lead to the loss of several billions of dollars each year.

Manuscript received 20 October 2023; revised 22 February 2024 and 25 June 2024; accepted 4 July 2024. Date of publication 22 July 2024; date of current version 30 July 2024. This work was supported by the German Research Foundation (DFG) through the Projects SecuReFET and SecurReFET II under Project 439891087. The associate editor coordinating the review of this article and approving it for publication was Prof. Stefano Tomasin. *(Corresponding author: Akash Kumar.)*

Armin Darjani and Nima Kavand are with the Department of Computer Science, Technische Universität Dresden, 01062 Dresden, Germany (e-mail: armin.darjani@tu-dresden.de; nima.kavand@tu-dresden.de).

Shubham Rai is with Robert Bosch GmbH, Corporate Research, 71272 Renningen, Germany (e-mail: shubham.rai@de.bosch.com).

Akash Kumar is with the Faculty of Electrical Engineering and Information Technology, Ruhr University Bochum, 44801 Bochum, Germany (e-mail: akash.kumar@rub.de).

Digital Object Identifier 10.1109/TIFS.2024.3431991

To confront this dilemma, researchers have proposed various design-for-trust (DFT) solutions. Logic locking is the most prominent and holistic DFT approach that offers protection through multiple stages of the IC supply chain [1], [2], [3]. By locking the design and adding new logic elements, this technique hides the true functionality of the circuit from an adversary. The design can only function correctly by feeding a valid key set to the circuit stored in a tamper-proof memory.

As shown in Fig. 1, logic locking techniques can be categorized into gate and interconnect/routing obfuscation. Interconnect obfuscation techniques lock the netlist by intertwining multiple paths coming from various logic cones in a locking structure. These techniques inject Multiplexers as new nodes into the design and connect these nodes to the correct and arbitrary nodes from the circuit's netlist [4], [5].

In gate obfuscation techniques [1], the designer perturbs the netlist by injecting a Multiplexer and a new logic gate into the netlist. This new logic gate is connected to the same input cone of the locked gate. For example, XOR/XNOR-based logic locking can be interpreted as a Multiplexer connected to the output of a gate and an inverter.

Although deemed as a safe technique, proposing Seminal Boolean satisfiability (SAT)-based attack [6] challenged the security of logic locking. This powerful attack utilizes a working duplicate of the target circuit (an Oracle) to infer the valid key. This attack works based on the functionality of the circuit. Using a SAT solver, it looks for distinguishing input patterns (DIPs) that prune the key search space based on the extracted outputs from the Oracle. By eliminating the wrong keys from the search space, this attack returns the correct key when no more DIPs can be generated.

Despite of its initial success, the threat model and efficiency of the SAT attack have been questioned. The main assumption in the threat model of the SAT attack is access to an Oracle with an open scan chain in test mode. This assumption may need to be revised since accessing a functional chip is not always possible, and all the commercial vendors prevent access to the scan chain; secondly, if such access is provided, it can lead to more powerful attacks such as probing [7] and smart brute force simulation attacks [8]. Regarding efficiency, the SAT attack cannot break circuits protected by SAT-hard structures. These structures employ routing obfuscation techniques [9], [10], [11], [12] or AND-tree structures such as [13] and [14] to exponentially increase the run-time of the SAT attack. Moreover, breaking logic locking in large complex circuits is challenging for the SAT attack [15].

Knowing these shortcomings and in search for more realistic assumptions and accessible attack environments, researchers have started to propose structural-based attacks [16], [17],

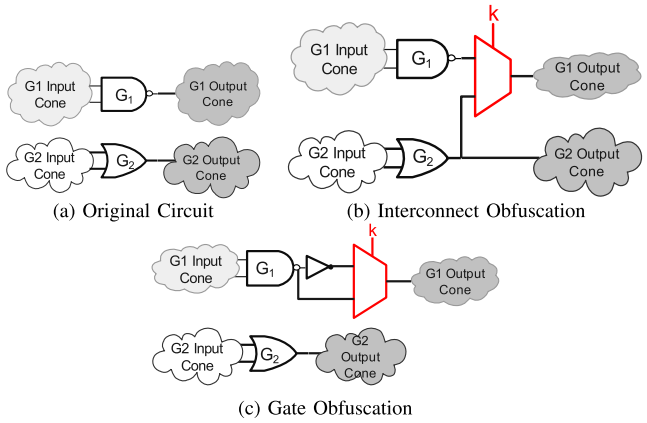


Fig. 1. Logic locking techniques. a) Original circuit. b) Interconnect obfuscation. c) Gate obfuscation.

[18], [19], [20], [21], [22], [23]. These attacks omit the need for an Oracle from the threat model; hence, they are called Oracle-less attacks. They utilize the structural features of the design to trace back the changes introduced into the circuit by logic locking. There are three main advantages of structural attacks over SAT attacks. Firstly, they omit the need for an Oracle, leading to a more realistic threat model [17]. Secondly, they can break logic locking in circuits with SAT-hard functions [24]. Finally, they are a better choice for attacking circuits with complex arithmetic units where SAT fails to infer the valid key set [25]. The most promising structural attacks are those based on machine learning (ML) [18], [19], [20], [21], [24], [26]. The final netlist of a circuit is a product of security-agnostic synthesis tools. When synthesizing an RTL code to a netlist, these tools only consider the circuit's area, delay, and power. As the boolean functions are repeated multiple times inside a circuit, the final netlist of a circuit will contain repeated structures and duplicated basic functions [27].

The ML-based attacks revealed that although adding locking structures and re-synthesizing the design can lead to changes in a netlist, the scope of these changes is small and confined to the adjacency of the locked structures [18], [20], [21]. Moreover, these changes follow only a few rules used by logic synthesis tools. Utilizing these structural properties by training a neural network based on the locking scheme, the ML can infer the original design from the locked circuit. Among ML approaches, graph neural networks (GNNs) are the most promising neural networks to break logic locking [20], [21] since the structure of a netlist can be easily mapped to a graph with logic gates as nodes and the wires between them as edges. This graph contains multiple subgraph localities with the same network of wires and gates. GNN-based attacks showed that the structure of the extracted subgraph local to the add-on security gates could leak information about the value of the key bit inside that subgraph.

This work[1] investigates the challenges of protecting circuits against GNN-based attacks. Although these attacks are powerful, we show that they have limitations, and we present locking schemes that exploit those limitations to protect circuits.

[1]This article is an extended version of the prior work [31], with the title "Discerning Limitations of GNN-based Attacks on Logic Locking", which is accepted to be published in Design Automation Conference (DAC), 2023.

A. Motivation and Related Works

Researchers have recently proposed learning-resilient logic locking schemes that bring various levels of security against ML-based structural attacks. UNSAIL [28] is a synthesis-dependant protection scheme for XOR/XNOR-based obfuscation proposed to thwart the SAIL [18] attack. However, this technique cannot protect the circuit against GNN-based attacks, as re-synthesizing the design locked by UNSAIL can leak structural information. TRLL [32] is an XOR/XNOR-based logic locking technique that is proposed to thwart ML-based attacks against gate-based logic locking techniques. The purpose of this technique is to circumvent the need for re-synthesizing the locked circuit after adding XOR/XNOR logic cells for security. In their work, the authors argue that by harnessing the inverter cells inside a circuit, one can confuse ML-based attacks. In this technique, an inverter can be replaced with an XOR logic cell connected to a key bit value $key = 1$, or the inverter can be connected to an XOR cell with a key bit value $k = 0$. Other logic gates of the design can be locked by an XOR cell connected to a key bit $k = 0$ or an XOR cell connected to an inverter with a key value $k = 1$. One of the weaknesses of the TRLL is the dependence of this technique on the number of available inverter cells in the circuit. TRLL+ [29] compensates for this weakness by adding random inversions to the design.

Both TRLL and TRLL+ fail the AND-netlist test (ANT) introduced in [4] that explores the learning resiliency of the locking scheme. Although some may argue that it is impractical to have only a single type of gate in a commercial design, the ANT technique actually provides insights into the dependence of the logic locking scheme to design properties. For example, if TRLL+ is used in a design with multiple similar structures, it might not provide the intended levels of security. We learned that this flaw is always relevant to gate-based obfuscation techniques unless all the structures with the same footprint are locked in the design. Moreover, in this work, we show that by proper training in GNN-based attacks, TRLL can be broken.

IsoLock [30] is a protection scheme based on isomorphic structures of the netlist graph that prevents ML models from inferring the correct key from the structure of interconnect-based obfuscation. Although effective and not vulnerable to the ANT test, this technique confines the security designer's options regarding both the number of keys and the placement of the secure logic structures. Moreover, this technique could be potentially vulnerable as re-synthesizing the design with various technology libraries might change the isomorphic feature of the locked structures.

Table I summarizes the shortcomings of learning-resilient techniques. This table shows that, although the TRLL+ and IsoLock can bring some levels of security against GNN-based attacks, the lack of generality and potential vulnerabilities point out the need for a new locking scheme based on deep analyses of GNN-based attacks.

TABLE I
SUMMARIZED ANALYSIS OF THE LEARNING-RESILIENT LOGIC LOCKING TECHNIQUES

Technique	Security Objective	Method	Weakness
UNSAIL [28]	Gate obfuscation attacks	Confusing the attack based on post-synthesis circuit	1-Potential leakage after re-synthesizing. 2-Depend on the synthesis tool.
TRLL, TRLL+ [29]	Gate obfuscation attacks	Bringing random inversion to the design	1-Fails ANT. 2-Can be broken by proper training of the GNNs.
IsoLock [30]	Link prediction attacks	Intertwining Isomorphic graphs	1-Limiting security designer. 2-Potential leakage after re-synthesizing.

B. Research Contributions

This paper first investigates the challenges of protecting a circuit against GNN-based attacks. Knowing these challenges, we propose GNN-resilient interconnect-based obfuscation (GRIN) and GNN-resilient gate-based obfuscation (GREGO) locking schemes that compensate for the shortcomings of the previously proposed schemes. Our proposed techniques do not induce structural hints or information leaks, and not only do they protect the circuit from GNN-based attacks, but also from non-ML structural attacks, such as SCOPE [17]. Our contributions to this work are as follows:

- ***Challenges of protecting circuits against GNNs:*** We investigate the challenges of protecting circuits against GNN-based attacks based on graph perturbations. We demonstrate that not all graph perturbations are possible in a circuit's graph, making circuit protection more challenging.
- ***GRIN:*** We analyse the structural weaknesses of interconnect-based obfuscation. Based on this analysis, we propose two provably GRIN logic locking techniques that protect the circuits against both GNN-based and constant propagation attacks. We also demonstrate the limitations of GNN-based attacks by using a level-based locking approach.
- ***GREGO:*** We show the inherent weaknesses of gate-based logic locking that results in successful ML-based attacks even against state-of-the-art protective schemes like TRLL. Based on our analysis, we present two GREGO schemes that thwart the GNN-based attacks against gate-based logic locking by weakening the self-referencing model of such attacks.
- ***Analysing the security of the proposed methods:*** We analyse the proposed approaches using the ISCAS-85 and ITC-99 benchmarks. Here we show the resilience of our GRIN approaches against MUXLink [21] and SCOPE [17] attacks. We show the security strength of our proposed GREGO schemes using OMLA [20] attack.
- ***Analysing the area and power overheads:*** We evaluate the area and power overhead of the GRIN techniques using an open-source CMOS library and Cadence Genus synthesis tool. Moreover, we provide insights into the security and overheads tradeoffs for our proposed GREGO techniques.

II. PRELIMINARIES

Any synthesized locked circuit is a netlist which is basically a graph, with edges being a network of wires between the gates. There is a good fit between the topology of an extracted netlist locality and the network of wires between gates. An ML-based attack aims to extract the value of the key bits from the features of the key gates' locality in the graph. The most promising ML approach that exploits the graph structure to infer such information is GNN [20].

In this section, we first introduce the GNNs. We then provide the threat model that illustrates the environment in which the structural attacks are carried on. Later, we discuss the inherent weaknesses of the interconnect and gate-based logic locking and introduce the GNN-based MUXLinks [21] and OMLA [20] attacks, which are respectively state-of-the-art interconnect and gate-based attacks.

A. GNNs

GNNs generate a vector representation for each vertex in the graph. These vectors are called embeddings that are utilized for various classifications. These embeddings are the output of multiple iterations of message passing [33]. In each iteration, each vertex $v \in \mathcal{V}$ updates its embedding using the information it gathers from the vertices in its neighborhood. The mathematical abstraction of each round can be formulated as equations (1) and (2):

$$a_v^{(l)} = AGG^l(\{c_u^{l-1} : u \in \mathcal{N}(v)\}) \tag{1}$$

$$c_v^l = UPDATE^l(c_v^{l-1}, a_v^l) \tag{2}$$

In (1), the $a_v^{(l)}$ is the output of $l'th$ iteration of aggregation function for node v that aggregates the embeddings of all the neighbors of node v. For each neighbor u the c_u^{l-1} is the result of update function of node u for iteration $(l-1)'th$. In (2), the c_v^l is the output of the update function for $l'th$ iteration of the GNN. The update function takes the latest embeddings (the embeddings from $(l-1)'th$ iteration) of node v and the result of the aggregation function of node v for $l'th$ iteration. So, in each iteration, The target vertex's features are combined ($UPDATE$ function) with the new information collected during each iteration. The final embedding, which can be used for graph classification, is generated after L iterations of message passing. Various implementations can be used for both AGG and $UPDATE$ functions. A readout of the subgraphs is carried out for link prediction to generate link embeddings used for classification.

B. Threat Model

As mentioned, GNN-based attacks follow the Oracle-less model. In this model, the attacker does not have access to an Oracle. Moreover, the attacker does not have any prior knowledge with regard to the functionality of the design. The adversary locates in an untrusted fab or test facility with access to the GDSII file of the design. The attacker

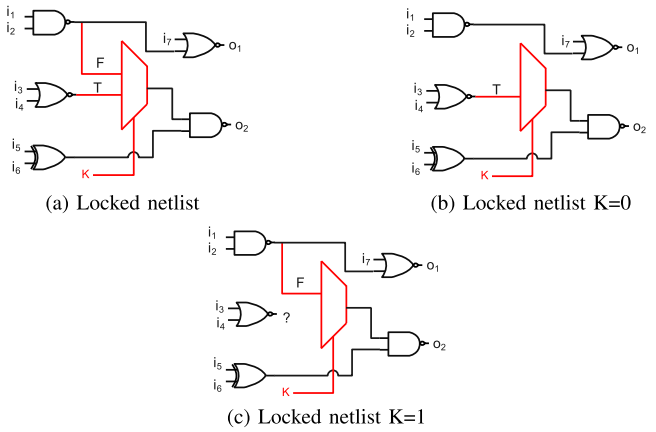


Fig. 2. Interconnect based obfuscation. a) Locked netlist, b) netlist after assigning the correct value to the key, c) netlist after assigning an incorrect value to the key.

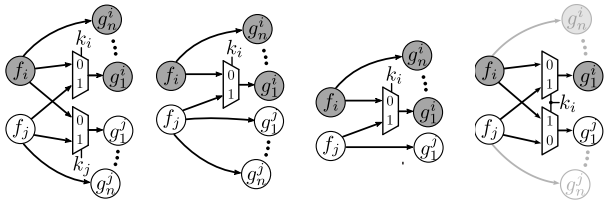


Fig. 3. D-MUX locking strategies [4]. In a, b, and c at least one of the chosen signals should have a $fanOut > 1$ where in d designer can choose any signal regardless of the fanOut.

can obtain the netlist by reverse engineering this GDSII file. After that, distinguishing key inputs is easy as these particular inputs should be connected to a tamper-proof on-chip memory. Moreover, the attacker has access to the standard cell library and locking algorithm used by the designer. Note that the attacker has no access to an Oracle which makes GNN-based attack models more challenging than SAT attacks.

C. Inherent Structural Weakness of Interconnect-Based Logic Locking

Fig. 2a shows an interconnect obfuscation scheme where one of the inputs of the target gate is obfuscated using a multiplexer connected to the correct cone and an arbitrary cone from the circuit. The valid key selects the correct logic cone, whereas a wrong key leads to a wrong connection. Although this scheme brings a good level of complexity to the netlist by adding new connections to the circuit's graph, it suffers from a fundamental weakness. As shown in Fig. 2c, setting the key bit to a wrong value leads to circuit reduction after re-synthesizing where selecting the correct path does not leave any dangling connections (Fig. 2b)

Constant propagation attacks like SWEEP [16] and SCOPE [17] exploit this weakness and attack the circuit by hard-coding the value of one key bit at a time and performing re-synthesis. These attacks gather design features, including power, area, number of AND gates in AIG representation, etc., during the re-synthesizing phase by initializing each key bit separately to constant values one and zero. Finlay, the attack correlates extoweracted features to correct key values.

In [4] authors proposed D-MUX that changes the naive interconnect obfuscation that eliminates the circuit reduction weakness and thwarts the constant propagation attacks. Fig. 3 shows D-MUX locking strategies. This technique utilizes four strategies to select the target gates for locking. a and b strategies search for gates with multiple outputs, whereas, in the c strategy, one of the target gates can have a fanout equal to one. The d strategy does not incur any restrictions in selecting the target gates. The strategies can be selected during the locking phase with regard to trade-offs between security needs and design overheads. The only limitation is that the locked design should avoid combinational loops. Note that the d strategy can be applied to any key size since it is always applicable. D-MUX guarantees the presence of both logic cones connected to the locked structure after re-synthesizing the design using constant values for the key inputs.

Although the D-MUX technique can thwart the constant propagation attacks, the authors in [21] proposed MUXLink, which showed that this technique is vulnerable when the attack is based on the link representation of the target circuit. This attack revealed that D-MUX brings local modifications to the circuit that can be reversed based on the extracted features of the circuit's structure.

MUXLink works based on the assumption that modern IC designs largely consist of repetitive structures. The intuition behind this attack is that the circuit modules are reused multiple times as most circuits contain regular structures used for functionalities like adders, etc. Harnessing the GNN, MUXLink learns the structures of unobfuscated parts of the locked design. The learned model will be utilized to deobfuscate the locked structures of the design using link prediction methods.

Fig. 4 shows the flow of the MUXLink attack. This attack first converts the netlist into a graph. Then it omits the connections between target nodes for each D-MUX structure from the graph representation. Then MUXLink extracts the h-hop enclosing subgraphs around the target nodes and assigns an 8-bit one-hot encoded vector and a tag to each node in the subgraphs. Here the one-hot encoder contains the information about the type of gate, and the tag contains relationship information between the node in the subgraph and the target nodes. This tag is calculated by the double radius node labeling (DRNL) equation that is shown in (3) where $df = d(j, f)$, $dg = d(j, g)$, and $d = df + dg$. $(d/2)$ is the integer quotient, and $(d\%2)$ is the remainder of d divided by 2. In this equation, $d(y, x)$ is the shortest path distance between x and y. If j has a path to no or only one of the target nodes, then $f_l(j) = 0$.

$$f_l(j) = 1 + min(df, dg) + (d/2)[(d/2) + (d\%2) - 1] \quad (3)$$

The target links are (f_i, g_i), (f_i, g_j), (f_j, g_i), (f_j, g_j). The MuxLink trains a GNN on the unobfuscated interconnects, and by learning the link connections in circuit subgraphs, it makes predictions on the target links. The absolute difference δ between the likelihood scores for all its possible links is computed for each locked structure. As long as δ exceeds the classification threshold, the link with the highest likelihood score is predicted to exist. Otherwise, MuxLink cannot make any predictions. To conclude, ***MUXLink is a powerful attack***

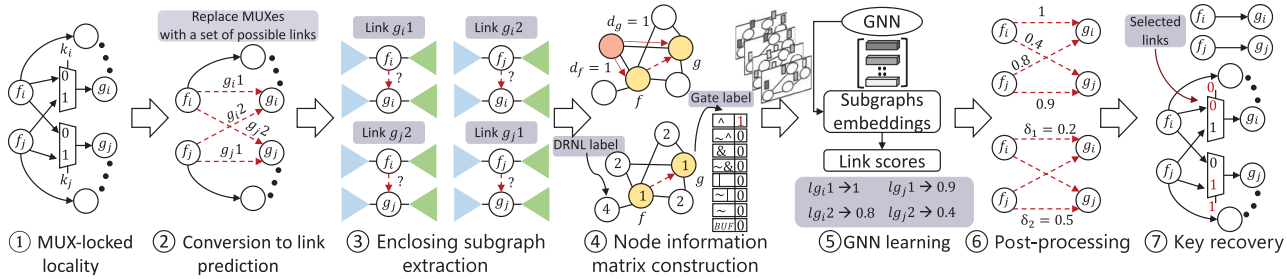


Fig. 4. MUXLink attack flow [21]. This attack converts the problem of guessing correct keys to a link prediction problem and predicts the correct connection utilizing GNN.

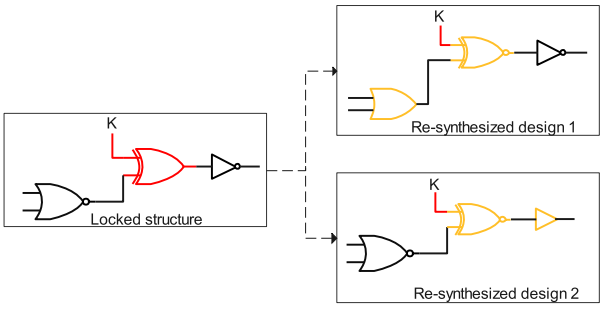


Fig. 5. XOR-based obfuscation technique. Here the value of the K is zero. For obfuscation, this structure should be re-synthesized which can result in various structures based on the rules that synthesis tools utilize.

when the extracted subgraphs of target nodes lead to different δ values with an adequate difference.

D. Inherent Structural Weakness of Gate-Based Logic Locking

Fig. 5 shows a basic gate obfuscation scheme. The protection of this scheme is based upon re-synthesizing as without re-synthesizing, it is easy to infer the value of the correct key, which is $zero$. Here, re-synthesizing might result in multiple local changes based on various rules that synthesis tools use mostly for gaining the best possible design regarding power, area, and performance.

In SAIL [18], the authors showed that re-synthesizing leads to local changes that can be learned and used to trace back the changes and finally infer the correct key values. SAIL showed that ML models could learn key bit values by learning the induced changes around the key gates in local enclosing subgraphs. Although SAIL is a powerful attack, it suffers from drawbacks like the need for pre-synthesis localities and complex learning models. Moreover, UNSAIL [28] and TRLL [32] schemes have been proposed that thwart the SAIL attack. UNSAIL uses synthesis transformations to inject bad data during the learning phase to confuse the attack. However, this technique is vulnerable to advanced learning attacks like OMLA, which extracts a subgraph adjacent to the target gates and decides the key value based on the structure of this subgraph. TRLL protects the circuit against SAIL by bringing random inversions to the design in order to decorrelate the relationship between the type of the key gates and the key values. As mentioned, this technique fails the ANT test. Moreover, TRLL and TRLL+ are susceptible to GNN-based attacks. In this section, we show that by proper training, the GNN-based attacks can break the TRLL technique.

In OMLA [20], authors showed that utilizing GNNs outperforms the attacks based on the traditional models. OMLA maps the problem of resolving the key-bit value to subgraph classification. Fig. 6 shows the flow of this technique. In this flow, each node is assigned feature vectors of length-d that capture the corresponding boolean function along with the connection of the gate to the key input, primary inputs, and primary outputs. Using distance encoding, each node in the extracted subgraph is tagged to enhance the structural power of the GNN. This tag measures the shortest path distance between a node and its target key gate. Moreover, the IN/OUT notion is hot encoded along with the distance tags.

OMLA follows a self-referencing model. Here the locked circuit is used for both training and validation. Using this model, the GNN learns the biases and behavior of the target netlist, resulting in significantly more accurate results when compared with a generic training model (89.55% against 66.68%). To conclude, ***OMLA is a powerful attack when there exist multiple subgraphs in the circuit with the same structural footprint as the parts chosen by the locking algorithm.***

To show the inherent weakness of gate-based obfuscation, we show that by changing the training approach, the OMLA attack can break TRLL and TRLL+. Fig 7 shows locking structures that TRLL uses to obfuscate a circuit. Without prior knowledge about the key values, one cannot guess the value of the key bits based on these locking structures. TRLL+ locks circuits using a simple rule. In this technique, a random number of inverter cells inside a circuit replace with XOR gates connected to a $key = 1$ or an XNOR gate connected to a $key = 0$. Other logic cells can be locked based on the traditional random XOR/XNOR logic locking technique. Like TRLL, no one can guess the correct key value as it is unclear from the design.

Although these techniques can thwart ML-based attacks when trained by random logic locking, we show that by changing the training approach, the OMLA attack can break them. We evaluate three implementations as follows:

- **Basic TRLL:** We implement the TRLL technique on each circuit after technology mapping using. We first lock the circuits using the structures shown in Fig. 7 with a 64-bit length key. For training, we re-lock these circuits using the same structures with 64 new key bits for 300 times and then use the OMLA attack.

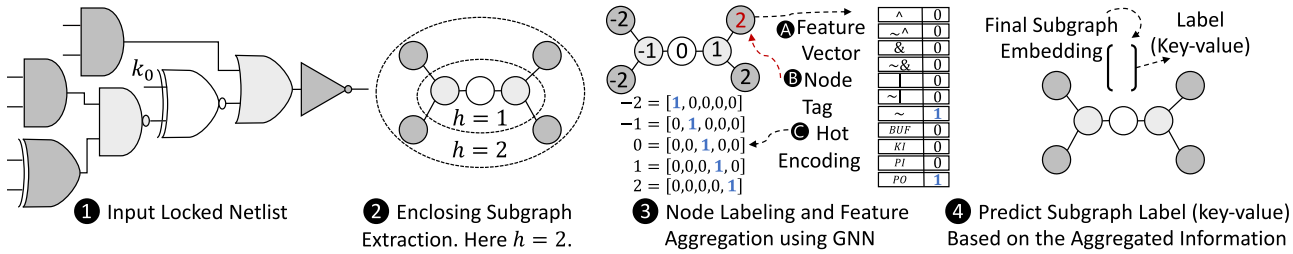


Fig. 6. OMLA attack flow [20]. This attack converts the netlist to a graph and guesses the correct key values based on the enclosing subgraph structures utilizing GNN.

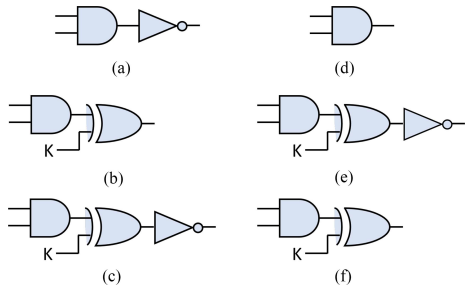


Fig. 7. TRLL locking structures [32]. a and d are two gate structures in the original circuits. b, and c are locked structures of a. e and f are locked structures of d.

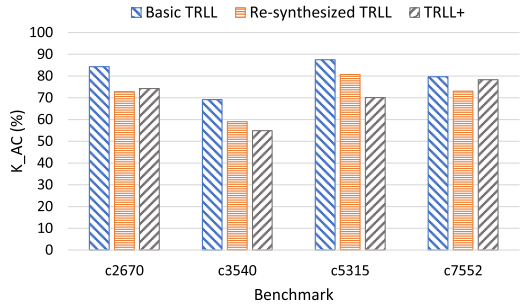


Fig. 8. Analysis of the OMLA attack on various implementations of TRLL and TRLL+.

- **Re-synthesized basic TRLL:** Like the basic TRLL approach, we implement the TRLL technique on each circuit using the locking structures of Fig. 7. Then we re-synthesize these locked circuits. For the training set, we first re-lock the re-synthesized locked circuits with the structures of Fig. 7 for 300 times. We re-synthesize all the training circuits and then use the OMLA attack.
- **TRLL+:** Here, we randomly select 32 inverter cells and 32 two input logic cells for each circuit after technology mapping. Then we replace the inverters with XOR/XNOR gates based on the value of the key bits. For other 32 bit key bits, we add XOR/XNOR gates to the circuit and re-synthesize the circuits. For the training set, we re-lock the re-synthesized circuits using the same approach for 300 times and re-synthesize all of them. Finally, we use the OMLA attack to break the test circuits.

Fig. 8 shows the results of the OMLA attack for various implementations. Here K_AC is the accuracy of the OMLA attack. These results show that training GNN-based attacks using the same approach for locking a circuit results in successful attacks. Although human eyes cannot guess the key values of the add-on key gates, the substructures surrounding the inverter cells can be learned by the GNN as the extracted subgraphs adjacent to the inverters are alike. So, if the training set contains enough locked inverter structures, the GNN can easily classify locked inverter cells based on the key values. This is why even the basic TRLL can be broken by adequate training of the GNN. Re-synthesizing the locked circuit cannot help hide these structures, as training the GNN based on the same scheme reveals the optimized locked inverter substructures. This analysis confirms the previous work [34] that has shown that even replacing a logic gate or group of logic gates with u-input LUTs cannot provide intended security for the circuit. A u-input LUT can build all 2^{2^u} possible functions, so it brings more complexity to the circuit comparing XOR/XNOR locking gates, however in [34], the authors proposed a GNN-based attack that can guess the true structure of the cut replaced by an LUT.

III. Challenges of Protecting Circuits Against GNN-Based Attacks

The final netlist of a circuit is a product of security-agnostic synthesis tools. This netlist contains repeated structures and repeated basic functions [27]. So, the graph of the netlist has multiple subgraph localities with the same network of wires and gates. Although adding key structures (gate or interconnect) and re-synthesizing the design can lead to subgraph changes, these changes will be confined to only a small number of hops and are still detectable [18], [20]. As a result, the structure of the extracted subgraph local to the add-on security gates can leak information about the value of the key bit inside that subgraph. So, the naive gate or interconnect obfuscations cannot protect the circuits.

Security designers can follow two approaches to protect the design from GNN attacks. Firstly, the designer can bring perturbations to the netlist graph of the circuit to confuse the GNN from inferring the correct key values from the information learned during the training phase. As a result of the changes in the graph nodes' characteristics (tags and labels), GNN-based attacks cannot map the locked region to the correct key value. Fig. 9 shows possible perturbations on a graph. In the following, we discuss the meanings and possibility of these perturbations with regard to a circuit's netlist.

- *Deleting an edge:* Removing an edge in a netlist can be interpreted as removing the connection between two logic gates. As this type of perturbation can change the final outputs of the circuit to wrong values, it is not a possible solution for thwarting GNN-based attacks. If the designers want to use this sort of perturbation

in the circuit, they should neutralize the effects of such perturbation by adding new corrective structures to the circuit. However, this additional circuitry may leave its traces [24] that can further be utilized to attack the circuit.

- *Adding an edge:* This perturbation can be interpreted as connecting two gates in the circuit by adding a wire. Like deleting an edge, this type of perturbation can lead to output corruptions in the circuit unless it is followed by a corrective add-on logic in the circuit that leaves traces [24].
- *Node injection:* This perturbation in a circuit can be interpreted as adding new gates and connections. Both gate and interconnect obfuscations follow this type of perturbation. In XOR/XNOR-based locking, an XOR/XNOR gate is added in front of a target signal, and a wire is added to the circuit to connect the target gate and the new XOR/XNOR gate. In interconnect-based logic locking each Multiplexer is a new injected node in the graph connecting to two or more wires coming from the target gates. In this manner, we can speculate that interconnect-based locking brings more complex perturbation to the graph as it changes the topology of the graph by bringing new paths between nodes.
- *Modifying features of the nodes:* The feature vector of a node in GNN-based attacks contains the type of the corresponding gate, connections to primary inputs (PIs), connections to primary outputs (POs), and connections to key inputs (KIs). To change the feature of a node, the security designers should re-synthesize the circuit using various synthesis options and technology libraries. This leads to changes both in the topology of the netlist's graph and the gate types; however, the security designer cannot rely on this type of perturbation. Based on the threat model, the attackers have access to the netlist. So they can re-synthesize the design using the intended technology library and synthesis options. As a result, these features can be changed, and the attacker can continue the attack with the new netlist.

The second approach for protecting designs from GNN-based attacks is to limit the attacker in the training phase. As the GNN-based attacks provide the best results when they utilize the self-referencing model [20], the security designer can utilize key gate placement algorithms that limit the training phase of the attacks. In this approach, the attacker should perturb all structures with the same footprint using the node injection perturbation technique to deprive the self-referencing model from learning useful information in the training phase.

IV. GNN-Resilient Interconnect-Based Obfuscation

This section proposes two provably secure GRIN schemes that can thwart GNN-based attacks against interconnect obfuscation. In addition, we discuss a depth-based interconnect logic locking scheme that locks a subset of gates based on their depth within the netlist graph. Here we argue the resilience of our schemes based on the features and techniques discussed in the MUXLink, while our claims remain true if an attacker uses another link prediction technique.

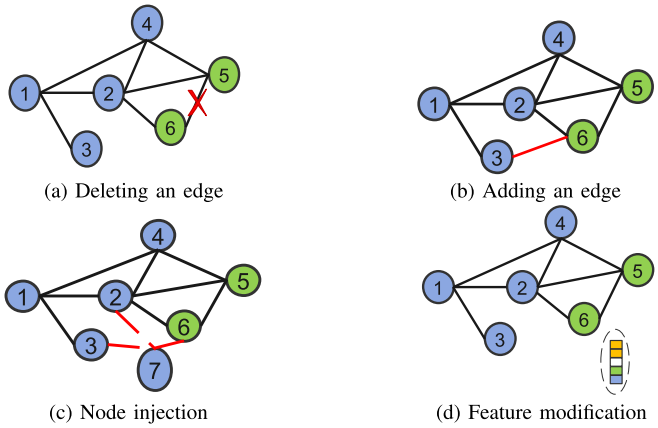


Fig. 9. Various graph perturbation techniques. a) Edge deletion. b) Edge addition. c) Node injection. d) Feature modification.

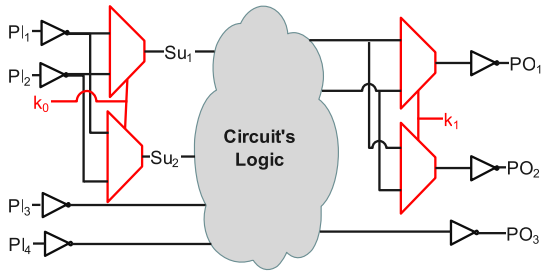


Fig. 10. PI/PO Obfuscation. Here a PI pair containing $\{PI_1, PI_2\}$ and a PO pair containing $\{PO_1, PO_2\}$ are locked.

A. PI/PO Obfuscation

In the original D-MUX scheme, the algorithm searches for an arbitrary pair of nodes (f_i, f_j) for implementing one of the strategies shown in Fig. 3. The only rule for selecting pairs is avoiding cyclic loops in the circuit. This naive pair selection approach leads to distinguishable substructures where the predicted δ values for true and false links are adequately different. In the PI/PO obfuscation approach, we change the naive pair selection approach of the D-MUX and select pairs of structurally similar nodes to lock.

Here, we lock the intended number of PIs/POs using strategy d of the D-MUX shown in Fig. 3. By locking pairs of primary inputs or pairs of primary outputs of a circuit, PI/PO obfuscation can thwart the MUXLink attack, as locking substructures lead to similar embeddings for locked substructures. In the following, we provide a formal explanation that demonstrates how locking the primary inputs and primary outputs of a circuit can guard the circuit from the MUXLink attack.

For the PI obfuscation, given a set of all primary inputs of the circuit that we call $PI_{Circuit} = \{PI_1, PI_2, \ldots, PI_n\}$, we select a subset PI_{locked} with m members, where $m \leq n$ and $m \mod 2 = 0$. Now, let Su_i be the set of successors nodes of PI_i where $i \in \{1, 2, \ldots, n\}$. Now, for each $\{PI_i, PI_{i+1}\} \in PI_{locked}$, we omit the connections between PI_i and PI_{i+1} to Su_i and Su_{i+1} respectively. Then, we connect each pair (PI_i, PI_{i+1}) to the pair of multiplexers, as shown in Fig. 10.

The selectors of the pair of multiplexers is a set of key inputs $KI_{PI} = \{KI_1, KI_2, \ldots, KI_k\}$ where $\frac{m}{2} \geq k \geq 1$.

For describing the PO obfuscation, we can use the same approach with small changes. For PO obfuscation, given a set of all primary outputs of the circuit that we call $PO_{Circuit} = \{PO_1, PO_2, \ldots, PO_n\}$, we select a subset PO_{locked} with m members, where $m \leq n$ and $m \mod 2 = 0$. Now, let $Pred_i$ be the predecessor node of PO_i where $i \in \{1, 2, \ldots, n\}$. Now, for each $\{PO_i, PO_{i+1}\} \in PO_{locked}$, we omit the connections between the $Pred_i$ and $Pred_{i+1}$ to PO_i and PO_{i+1} respectively, and we connect $Pred_i$ and $Pred_{i+1}$ to a pair of multiplexer. The selectors of the pairs of multiplexers is a set of key inputs $KI_{PO} = \{KI_1, KI_2, \ldots, KI_k\}$ where $\frac{m}{2} \geq k \geq 1$.

Fig. 10 shows PI/PO obfuscation where we lock set of $PI_{locked} = \{PI_1, PI_2\}$ with K_0 as the selector and we lock set of $PO_{locked} = \{PO_1, PO_2\}$ with K_1 as the selector. Note that the difference between this scheme and naive D-MUX implementation is that firstly, we use a node selection algorithm (PI/PO obfuscation) instead of random pair selection, and secondly, for the PI obfuscation, we have locked all the successors of selected PIs.

In the following, we provide a formal explanation for the resiliency of the PI obfuscation scheme, shown in Fig. 10. The same approach can be used for the resiliency of PO obfuscation.

In this circuit, the set of successors of the PIs is $Su_{PIs} = \{Su_1, Su_2, Su_3, Su_4\}$ where each member of the Su_{PIs} is a set of nodes. In our example, $PI_{circuits}$, PI_{locked}, Su_1, and Su_2 are as follows:

$$\begin{aligned} PI_{circuit} &= \{PI_1, PI_2, PI_3, PI_4\} \\ PI_{locked} &= \{PI_1, PI_2\} \\ Su_1 &= \{Su_1^1, Su_1^2, \ldots, Su_1^v\} \\ Su_2 &= \{Su_2^1, Su_2^2, \ldots, Su_2^y\} \end{aligned}$$

where the v is the number of successors of the PI_1 node and, y is the number of successors of the PI_2 node.

For Su_1, the Muxlink attack calculates a set of likelihood scores for the PI_1 and PI_2 as $LScore_{Su_1}^{PI_1}$ and $LScore_{Su_1}^{PI_2}$. Finally, the MUXLink compares each member of $LScore_{Su_1}^{PI_1}$ with the corresponding member inside $LScore_{Su_1}^{PI_2}$ for each Su_1^i where $i \leq v$.

$$\begin{aligned} LScore_{Su_1}^{PI_1} &= \{(PI_1, Su_1^1), (PI_1, Su_1^2), \ldots, (PI_1, Su_1^v)\} \\ LScore_{Su_1}^{PI_2} &= \{(PI_2, Su_1^1), (PI_2, Su_1^2), \ldots, (PI_2, Su_1^v)\} \end{aligned}$$

For each Su_1^i, if $(PI_1, Su_1^i) = (PI_2, Su_1^i)$, the MUXLink attack returns X, meaning it cannot provide any predictions. So, if we can show that for each possible links (PI_1, Su_1^i) and (PI_2, Su_1^i) where $i \leq v$, we have the equation $(PI_1, Su_1^i) = (PI_2, Su_1^i)$ and for each (PI_1, Su_2^i) and (PI_2, Su_2^i) where $i \leq y$, $(PI_1, Su_2^i) = (PI_2, Su_2^i)$ we can conclude that PI obfuscation thwarts the MUXLink attack.

In the link prediction stage, for each node inside Su_1 and Su_2, the MUXLink attack extracts the 3-hop subgraph around PI_1, PI_2, and that specific node. Note that there are no links between PI_1 or PI_2 to other nodes inside the circuit as all the unknown links are omitted by the MUXLink. Now, let's consider node Su_1^1 as an example. We name the set of nodes in the 3-hop subgraph of Su_1^1, $3hop_{Su_1^1}$, which is a set of nodes as follows:

$$3hop_{Su_1^1} = \{N_1, N_2, \ldots, N_l\}$$

The MUXLink attack should calculate the likelihood score for (PI_1, Su_1^1) and (PI_2, Su_1^1). The MUXLink attack starts with calculating the likelihood score for pair (PI_1, Su_1^1). For this, each node inside $3hop_{Su_1^1}$ set is assigned a DRNL label, as we showed in equation 3, along with an 8-bit encoder that encodes the gate type. As there is no path from the nodes inside the $3hop_{Su_1^1}$ to PI_1 DRNL value will be set to 0 for all the nodes inside $3hop_{Su_1^1}$. Moreover, the MUXLink attack should assign these labels to a 3-hop subgraph of PI_1, however as the PI_1 is the primary input, there is no predecessor for this node, and we have already disconnected it from its successors using PI obfuscation. So, the 3-hop subgraph of PI_1 is an empty set. After assigning DRNL and gate type labels to nodes inside $3hop_{Su_1^1}$ the MUXLink starts AGG, and UPDATE showed in equations 1 and 2 for these nodes. After multiple iterations of AGG and UPDATE, each node inside the $3hop_{Su_1^1}$ will be assigned an embedding. We show the embeddings of nodes inside $3hop_{Su_1^1}$ as:

$$Embeddings_{Su_1^1}^{PI_1} = \{Emb_{N_1}, Emb_{N_2}, \ldots, Emb_{N_l}\}$$

Finally, MUXLink calculates the likelihood score for (PI_1, Su_1^1) based on these embeddings and the trained model.

After this, the MUXLink starts with calculating the likelihood score for pair (PI_2, Su_1^1). As PI_2 is also a primary input buffer and is disconnected from other nodes, there is no 3-hop subgraph for PI_2. The MUXLink attack extracts the $3hop_{Su_1^1}$, which is the same as it did for pair (PI_1, Su_1^1). The DRNL will be set to 0, for all nodes inside $3hop_{Su_1^1}$, and the gate types are the same as before. MUXLink goes through the same process for calculating the embeddings of nodes inside $3hop_{Su_1^1}$. So the embedding set $Embeddings_{Su_1^1}^{PI_2}$ will be equal to the previous calculation.

Based on these embeddings and the unchanged trained model, MUXLink calculates the likelihood score for (PI_2, Su_1^1). Since $Embeddings_{Su_1^1}^{PI_1} = Embeddings_{Su_1^1}^{PI_2}$ and the model remains unchanged, the likelihood score of (PI_1, Su_1^1) equals (PI_2, Su_1^1). Consequently, MUXLink returns X for this prediction.

This reasoning extends to all pairs (PI_1, Su_1^i) and (PI_2, Su_1^i), as well as (PI_1, Su_2^j) and (PI_2, Su_2^j), where i and j represent the successors of PI_1 and PI_2 respectively. Therefore, MUXLink returns X for all predictions regarding potential output links from PI_1 and PI_2.

To demonstrate the security of the PO (Primary Output) obfuscation, we can employ a similar approach, given that locked POs are entirely detached from the circuit and lack successors. Consequently, the node embeddings within the 3-hop subgraph of their predecessors will remain the same, effectively thwarting the MUXLink attack.

This formal explanation demonstrates that we can safeguard our circuits against MUXLink attack by locking pairs of primary inputs (PIs) or primary outputs (POs). However, PI/PO obfuscation might not offer the same level of security if we lock the PIs or POs of a circuit module that will be instantiated within a larger circuit. Consequently, the primary limitation of PI/PO obfuscation is that it restricts the security designer to working with only the PIs and POs of the entire circuit.

B. CUT-Based Obfuscation

Even though the PI/PO obfuscation technique can thwart the MUXLink attack, it limits the number of keys and confines the designer to only locking the inputs and outputs of the entire circuit. Here, we propose CUT-based obfuscation that provides the security designers with locking structures that can lock any arbitrary signal inside the circuit in order to safeguard the circuit against MUXLink attack.

Fig. 11c shows the cut-based obfuscation technique. In this technique, given a pair of nodes $\{f_i, f_j\}$ like shown in Fig. 11a, we first omit the connections of f_i and f_j to all of their successors that we call $Succ_{f_i}$ and $Succ_{f_j}$ respectively. We connect f_i and f_j to D-MUX structure as shown in Fig. 11b where all the nodes in $Succ_{f_i}$ and $Succ_{f_j}$ are connected to the output of the two multiplexers with a key bit as their selector. We call this scheme improved D-MUX. The difference between the D-MUX technique shown in Fig. 3d with the improved D-MUX is that we connect all fanout gates of f_i and f_j to the locking multiplexers.

Following the disconnection of f_i and f_j from their successors, we omit the links originated from the predecessors of f_i and f_j to these two nodes. These sets of predecessor nodes are denoted as $Pred_{f_i} = \{Pred^1_{fi}, Pred^2_{fi}, \ldots, Pred^n_{fi},\}$ and $Pred_{f_j} = \{Pred^1_{fj}, Pred^2_{fj}, \ldots, Pred^m_{fj},\}$ respectively. Finally, we add a $n+m \rightarrow n+m$ SB, and we connect the nodes from $Pred_{f_i}$ and $Pred_{f_j}$ to f_i and fj through this SB locked with a set of key $KI = \{K_1, K_2, \ldots, K_{n+m}\}$. Fig. 11c shows the CUT-based obfuscation scheme for a $\{f_i, f_j\}$ pair where both n and m are equal to 2 for $Pred_{f_i}$ and $Pred_{f_j}$.

The CUT-based technique completely isolates the $\{f_i, f_j\}$ pair from the circuit as both the inputs and outputs of this pair are connected to locking multiplexers. This translates to a pair of dangling nodes during a link prediction attack. So, during the attack, the DRNL value of all the nodes in the 3-hop subgraph of the SB's input nodes will be set to 0. The same is true for the 3-hop subgraph of the output nodes of the final multiplexers. This leads to the same embeddings for all the nodes after multiple iterations of the GNN, effectively thwarting the link prediction attack.

Algorithm 1 defines the various steps of this scheme. The inputs of this algorithm are the key, the netlist, and pairs of $\{f_i, f_j\}$ that are extracted from the pair selection function that is explained in Algorithm 2. Here all the pairs $\{f_i, f_j\}$ are stored in list N. In each iteration of the algorithm, one key bit is selected for locking the fanout of the selected pair (as selectors of D-MUX structure), and two or more keys are selected for locking the fanin cone of each gate in the selected pair based on the number of fanin signals of these gates (as selectors of SB). This process is shown in lines 6 to 8 of Algorithm 1. After that, all the connections of the logic cells in the selected pair are removed, and a locked structure containing the SB and D-MUX is added to the circuit. Here, line 10 adds improved D-MUX structure as shown in Fig. 11b, and line 12 adds the SB as shown in Fig. 11c. Finally, line 14 updates all the connections of the circuit where the output of the multiplexers are connected to the gates that were connected to f_i and f_j, and the inputs of f_i and f_j are connected to the SB.

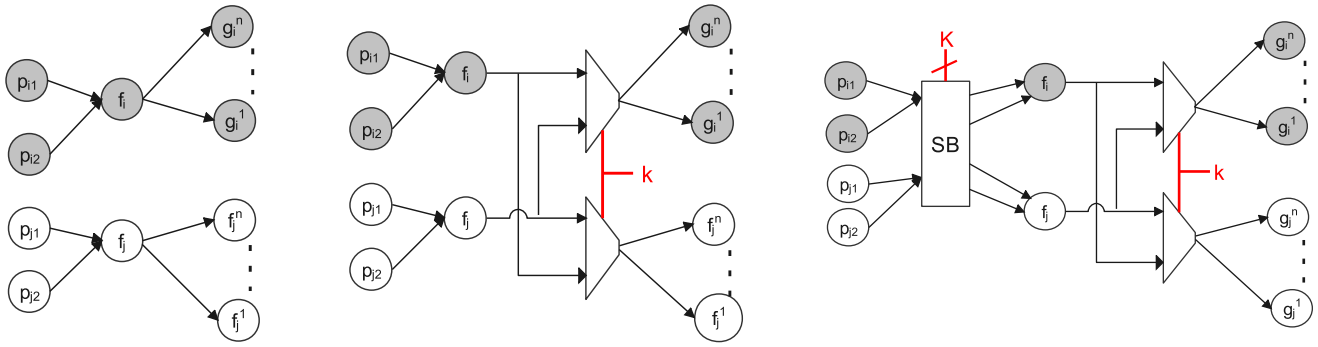
Algorithm 2 shows the pair selection approach. The inputs of this algorithm are Netlist, strategy, number of required pairs, and maximum number of iterations. The output is a list of pairs like $\{f_i, f_j\}$. The designer can select the strategy. For

Algorithm 1 CUT-Based Locking

```
1: Input: Pair selection S, Key K, Netlist Net
2: Output: Locked netlist
3: N ← S(Net)
4: K_list ← ToList(K)
5: While |K_list| ≥ 0 do
6:     select randomly n ∈ N
7:     N ← RemoveFrom(n, N)
8:     K_fanout, K_fanin ← GetAndRemove(K_list)
9:     RemoveConnections(n)
10:    Conn_out ← AddMUXs(n[0], n[1], fanout_n[0],
11:            fanout_n[1], K_fanout)
12:    Conn_in ← AddSB(n[0], n[1], fanin_n[0], fanin_n[1],
13:            K_fanin)
14:    UpdateNetlist(Net, Conn_out, Conn_in)
15: end While
16: return Net
```

Algorithm 2 Pair Selection

```
1: Input: Netlist Net, Selection Strategy St, Num of pairs
   Num, Max iteration Itr_max
2: Output: List of pairs
3: ListOfPairs ← {}
4: S_list ← St(Net)
5: For s in S_list do
6:     PairSelected ← False
7:     Itr ← Itr_max
8:     While Itr ≥ 0 do
9:         C ← RandSel(Net)
10:        Cond_1 ← CheckForLoop(fanout_s, fanout_C)
11:        Cond_2 ← CheckForLoop(fanin_s, fanin_C)
12:        If Cond_1 and Cond_2
13:            ListOfPairs.append(s, C)
14:            DeletFromNet(s, C)
15:            DeletFromS_list(s, C)
16:            break
17:        End If
18:        Itr ← Itr − 1
19:    end While
20: end For
21: return ListOfPairs
```



(a) Target gates (b) Improved D-MUX structure (c) CUT-based locked structure

Fig. 11. CUT-based interconnect-obfuscation technique. a) Target logic gates with their fanin and fanout cones. b) Improved D-MUX structure that obfuscates all the fanout signals of the target gates. c) CUT-based structure disconnects target gates completely from the circuit.

example, a designer can select nodes with high activity or the nodes with the highest effect on the output of the circuit. After selecting a list of nodes (line 4), the algorithm starts a loop (lines 5 to 20)to find a suitable pair from netlist Net for each node inside (S_{list}). Here, for each node s inside S_{list}, the algorithm tries to extract node C from the netlist where locking them using CUT-based obfuscation does not lead to combinational loops inside the circuit. Lines 10 and 11 search for a combinational loop for fanouts and fanins of pair $\{s, C\}$. If such a pair is found (lines 12 to 16), the while loop will be ended, the pair is added to the output list $pairs$, and the process will be carried out for the next node in S_{list}. In this algorithm, we use the maximum iteration numbers Itr_{max} to decrease the time overhead of the algorithm. If no pair $\{s, C\}$ is found after Itr_{max} iterations, it indicates that the algorithm has been unable to find a suitable pair C for node s within the allotted time. Consequently, the algorithm should initiate the process for the next node within S_{list}. Finally, the algorithm outputs the list of pairs that will be used in Algorithm 1.

We can explain the security of the CUT-based technique by using a similar approach to PI/PO obfuscation. Here, we break down the link prediction problem into two separate predictions. Firstly, the task is to predict the output connections of both f_i and f_j. This situation closely resembles the PI obfuscation problem since neither f_i nor f_j have predecessors. The difference is that f_i and f_j are not input buffers. Thus, the gate type feature of f_i and f_j can play a role in link prediction. As a result, this scheme does not provide 100% protection as PI obfuscation.

Secondly, the attack should predict the correct connections among the inputs and outputs of the SB. For example in Fig. 11c the following pairs should be $\{p_{i1}, f_i\}$, $\{p_{i2}, f_i\}$, $\{p_{j1}, f_j\}$, $\{p_{j1}, f_j\}$ predicted as correct links. This problem can resemble the problem of predicting PO obfuscation. The difference is that f_i and f_j are not output buffers, and they are two input gates from the design. So, although the DRNL value for the 3-hop subgraph of all the input nodes of the SB is set to zero, f_i and f_j can play a role in link prediction. As a result, this scheme does not provide 100% protection as PO obfuscation.

Indeed, by omitting the distance feature (DRNL), the CUT-based scheme achieves a notable level of protection. Our experiments outlined in Section VI our findings illustrate that the accuracy of the link prediction attack consistently stays below 24% across all benchmarks when utilizing CUT-based obfuscation, completely neutralizing the MUXLink attack.

C. Depth-Based Obfuscation

In this technique, we perturb nodes from the same depth in the graph corresponding to gates from the same level in the circuit. Fig. 12 shows this approach. In this approach, we disconnect a gate from its successors. Then we connect these links to the output of the multiplexer of the SB. Note that this is different from the D-MUX approach, as D-MUX chooses only *one* of the outputs of the two target nodes to obfuscate. To safeguard the circuit against GNN-based attacks, we have proposed PI/PO and CUT-based techniques that successfully thwart the MUXLink attack. As we explained in the paper, PI/PO obfuscation confines the security designer to the inputs and outputs of the IP. The CUT-based technique does not confine the designer but adds the penalty of SB to the naive D-MUX technique. We proposed depth-based obfuscation to answer the following question:

Can we find a set of structurally similar intertwined nodes and lock them with a naive D-MUX technique without the need for SB and stop the MUXLink attack?

We propose depth-based obfuscation to show that the accuracy of the GNN-based attacks can be affected by selecting and locking various sets of intertwined nodes (logic gates at the same level). We aimed to show that locking a group of intertwined nodes (nodes from the same level) using a naive D-MUX technique can affect the accuracy of the MUXLink attack. There are two intuitions behind this approach. Firstly, as the circuit structure is uniform, nodes from the same level confirm well to the same predecessor and successor structures. This implies that there is a high chance that nodes that are at the same level have similar substructures regarding their successors and predecessors. Secondly, injecting multiple perturbations in the same vicinity of the graph can adversely affect the attack by affecting the $DRNL$ labels of the nodes connected to the target nodes. So, the chance of manipulating these labels will be higher with multiple perturbations in the same vicinity. For example, locking *all* the gates from $level\ n$ of the circuit can cut off the path from the gates in $levels \leq n-1$ to $levels \geq n+1$.

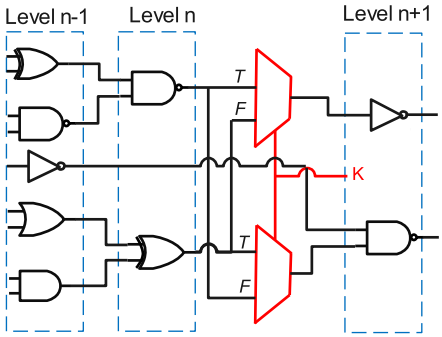


Fig. 12. Depth-based interconnect obfuscation. All logic cells from level n are locked.

In Section VI, we show that locking various levels of various circuits using this technique can affect the accuracy of the MUXLink attack. These results provide a basis for the community in search of ML-resilient placement algorithms.

V. GNN-Resilient Gate-Based Obfuscation

Gate-based obfuscation techniques are inherently weaker than interconnect obfuscation regarding GNN-based attacks. The reason is that this scheme does not generate new paths in the netlist graph. Gate-based obfuscation techniques only change a small subgraph adjacent to the add-on key gate(s) in the graph netlist. As explained, these changes are confined to finite numbers of rules the synthesis tools use for synthesizing the circuits [18]. GNN-based attacks break the gate-based logic locking based on the notion of repeated locked structures in a circuit. Following the self-referencing model is crucial for GNN-based attacks 6. OMLA shows that a self-referencing model for attacking circuits provides the best accuracy. Training on the same circuit that the attacker is trying to break leads to better results as the model is trained based on the biases of the target netlist. Using a generic model, this attack can only gain 66% accuracy in the best case where self-referencing results in up to 98% key-prediction accuracy [20]. These results show that we can thwart these attacks by depriving the GNN-based attacks of their strength, which is learning based on the similarity of the extracted subgraphs of the locked structures. In this section, we present two GREGO schemes that harness this approach to stop GNN-based attacks. Our discussion is based on the features and techniques discussed in the OMLA [20] while our claims remain true if an attacker uses another GNN-based technique.

A. Gate Type Locking

In our first scheme, we stop the GNN from learning useful information from the self-referencing model by locking all the XOR/XNOR gates of the circuit. The intuition behind this technique is that locking all the gates from the same type limits the training phase of the GNN attack to learn from substructures of the target circuit that are not similar to substructures selected for locking the circuit. In this scheme, we lock each XOR and XNOR logic gate using XOR/XNOR-based logic locking. Each XOR logic gate inside a design is locked using an XOR with $k = 0$ or an XNOR with $k = 1$ randomly. We do the same for all the XNOR logic gates inside the design. The reasons we select XOR/XNOR logic functions for this technique are as follows:

- *Complexity:* XOR/XNOR logic gates are complex cells that break into multiple logic gates if the attacker tries to re-synthesize the design with a technology library that does not contain XOR/XNOR logic cells. As a result, new locked structures will still be unique, and because all of these structures are locked inside the circuit, GNN cannot learn useful information during the training phase.
- *High activity:* XOR/XNOR logic cells are high-activity logic functions. As a result, a wrong guess for the key value for these logic cells injects high corruption into their fanout cone.
- *Low overhead:* As locking an XOR or XNOR logic cell can be implemented by optimal 3-input XOR logic gates [35], [36], the overhead of locking such cells is lower as compared to locking other gates types with XOR/XNOR gates.

In Section VI, we show that locking all the XOR/XNOR gates of a circuit can drastically decrease the accuracy of the OMLA attack.

B. Unit Equivalent Structure Locking

While effective, locking all the XOR/XNOR gates may not be a viable option in all the circuits. This technique can pose high overhead for the circuit structure if the design contains thousands of XOR/XNOR gates. On the other side, this technique cannot protect the circuit where there are only a few XOR/XNOR gates inside the circuit that lead to a very small number of corrupted output bits.

The strength of the GNN-based attacks is to harness the structure of the target circuit to break the locking scheme. However, the self-referencing model of these attacks can be exploited to prevent them. Here, we propose an approach that aims to lock the similar substructures of a circuit to prevent the self-referencing attack models. In this approach, we use a notion that we name unit equivalent structures (UES). Each UES is a set of substructures inside the circuits with the same structural footprint. For example, 2-input Multiplexers and the logic gates connected to the Multiplexers' inputs, output, and selector signals can be counted as a UES. Locking all the members of a UES prevents the self-referencing model of the GNN attacks from learning useful information during the training phase. Each UES contains members with a unique structural footprint that leads to unique locked structures in the circuit. Training the GNN attacks by locking random substructures of the circuit will lead the attack to random guesses, as no structural traces regarding targets' locked substructures can be learned. A circuit may contain multiple UES, and the security designer can select and lock these UES based on the security and overhead tradeoffs. Finding and grouping the substructures in various UESs is outside the scope of this paper. However, we show the effectiveness of this technique using UES with multiplexer substructures as its members.

In this scheme, we first synthesize the design using a generic library containing MUX cells. Then we lock the multiplexers' selector signal using random XOR/XNOR-based logic locking.

TABLE II
STATISTICS OF THE BENCHMARKS

Benchmark	# Primary inputs	# Primary outputs	# Logic gates
c2670	233	140	1193
c3540	50	22	1669
c5315	178	123	2406
c7552	207	108	3512
b14	277	299	10343
b20	522	512	20716
b22	767	757	31095
b17	1452	1512	33741

Note that this is different from the gate-type scheme, as the selector signal of a multiplexer can be a fanout of any gate type. We demonstrate in Section VI that this approach can thwart the OMLA attack for circuits with multiplexers.

VI. Experimental Results

A. Experimental Setup

We evaluate our protection techniques on four ITC-99 benchmarks and four ISCAS-85 benchmarks. Table II shows the statistics of these benchmarks. We use MUXLink and OMLA attacks for security evaluation for interconnect-based obfuscation and gate-based obfuscation techniques, respectively. We run the attacks on a server with 48 cores (Intel(R) Xeon(R) Platinum 8160 CPU @ 2.10GHz) and 300 GB of RAM. For synthesizing designs and training set generation for security and overhead analysis, we used the cadence GENUS synthesis tool using a 45-nm OSU free technology library.

B. GRIN Analysis

1) Implementation: We use the available online MUXLink source code for transforming circuits into graphs. For D-MUX implementation, we use the code provided in the MUXLink repository. We set the hop size for the MUXLink attack to 3 as it results in the best guesses by the attack [30]. For implementing PI/PO obfuscation, as MUXLink does not include the PIs and POs in the graph, we add a buffer to each PI and PO chosen randomly for locking and lock these buffers using the scheme shown in Fig. 10. We also implement improved D-MUX and CUT-based obfuscation techniques using Algorithms 1 and 2 that result in obfuscation schemes shown in Fig. 11. For implementing D-MUX, improved D-MUX, and CUT-based technique, we chose 16 pairs of signals from the circuit that satisfy the conditions of Algorithm 2 and lock them with D-MUX, improved D-MUX, and CUT-based techniques. For D-MUX, we follow the implementation of the strategy (d) from Fig. 3. as proposed in the original paper [4]. Here, only the connections between (f_i, g_i^1) and (f_j, g_j^1) are obscured, while other output links from f_i and g_i remain unchanged. We secure 16 pairs of (f_i, f_j) using strategy (d) from Fig. 3. This results in a total of 32 locked links, with each pair contributing 2 locked links for each (f_i, f_j). For Improved D-MUX, we use the same (f_i, f_j) pairs; however, we lock all the output connections of the pairs. So, as shown in Fig. 11, all the connections from f_i to g_i^1 to g_i^n will be locked. The same goes for f_j. So, for each pair (f_i, f_j), the number of locked links is the sum of the output links of f_i and f_j. We do the same for the CUT-based technique. Comparing D-MUX and Improved D-MUX, we have shown the effects of forward connections in link prediction attacks while comparing Improved D-MUX and CUT-based techniques can highlight the importance of obliterating the DRNL (path) feature for the under-attacked nodes as we proposed in CUT-based technique. To gain a better insight into the security and overheads trade-off of these techniques compared to D-MUX we also provide the number of locked links in the security analysis section.

Moreover, we use the available online SCOPE [17] attack's source code for analyzing the security of the proposed CUT-based technique against non-ML-based structural attacks.

2) Security Analysis: For evaluating the security of the PI/PO, CUT-based, and depth-based obfuscation techniques, we use link accuracy (L_AC) and link precision (L_Prec) metrics. L_AC is the ratio of the correctly predicted links out of all the locked links, and L_Prec is calculated as the ratio of the correctly guessed links plus the number of un-guessed (X) link connections out of all locked links. These metrics are shown in Equations (4) and (5). ***It is important to note that in the case of small values of*** L_AC***, having high*** L_Prec ***values shows great resilience of the protection technique as in such a scenario, the attack cannot make any prediction about the connections of the locked links; As a result, most of the locked links remain un-guessed by the attack as an X value***.

For evaluating the security of the SCOPE attack, we use K_AC and K_Prec equations as SCOPE reports the guessed value of the key bits. K_AC shows the percentage of correctly guessed key bits, and K_Prec shows the sum of the correctly guessed key bits and un-guessed key bits out of all key bits. Equations (6) and (7) shows K_AC and K_Prec

$$L\_AC = \frac{Links_{(Correct)}}{Locked\_Links_{(Total)}} * 100\% \tag{4}$$

$$L\_Prec = \frac{Links_{(Correct)} + Links_{(X)}}{Locked\_Links_{(Total)}} * 100\% \tag{5}$$

$$K\_AC = \frac{KEYs_{(Correct)}}{KEYs_{(Total)}} * 100\% \tag{6}$$

$$K\_Prec = \frac{KEYs_{(Correct)} + KEYs_{(X)}}{KEYs_{(Total)}} * 100\% \tag{7}$$

Table III shows the L_AC and L_Prec of the MUXLink attack against D-MUX, Improved D-MUX, and CUT-based techniques for various circuits. This table does not show the L_AC and L_Prec values of the MUXLink attack against PI/PO obfuscation since, as we proved in Section IV, this technique thwarts the MUXLink completely. The L_AC and L_Prec values of MUXLink attack against this technique are 0% and 100% respectively for all the circuits. This shows that for the PI/PO technique, MUXLink cannot make any prediction, so it returns value X for all of the locked links.

As shown in Table III, the CUT-based technique drastically decreases the ability of the MUXLink attack to predict the correct link connections. The average L_AC of 13.14% and L_Prec of 90.23% indicate that the attack cannot provide any prediction based on the features of the locked structures. This is the result of cutting the locked nodes entirely from

TABLE III
SECURITY ANALYSIS OF THE D-MUX, IMPROVED D-MUX, AND CUT-BASED OBFUSCATION TECHNIQUES

benchmark	D-MUX			Improved D-MUX			CUT-based		
	L_AC (%)	L_Prec (%)	#Locked Links	L_AC (%)	L_Prec (%)	#Locked Links	L_AC (%)	L_Prec (%)	#Locked Links
c2670	94.11	94.11	32	64.5	65.5	78	10.25	92.3	78
c3540	100	100	32	60.9	72.38	75	18.24	86.48	75
c5315	82.3	85.2	32	63.7	64.7	148	24.8	85.27	148
c7552	100	100	32	72.3	83.5	89	24.71	87.64	89
b14	100	100	32	69.7	77.6	168	10.11	92.26	168
b20	96.8	99.96	32	71	76.92	169	6.5	93.49	169
b22	93.75	96.87	32	72.03	80.05	236	5.22	03.34	236
b17	96.87	100	32	74.1	75.89	112	5.35	91.07	112
Avg	**95.47**	**97.01**	**32**	**68.25**	**74.56**	**134**	**13.14**	**90.23**	**134**

TABLE IV
NUMBER OF LOCKED LINKS IN VARIOUS LEVELS

Benchmark	Level				
	1	2	3	M	M+
c2670	241	192	123	30	23
c3540	247	205	217	25	29
c5315	653	573	192	11	14
c7552	235	277	280	26	24
b14	283	139	94	331	365
b22	1223	596	584	751	720

the graph, leading to a value of zero for the DRNL equation. Moreover, this table shows that improved D-MUX can also decrease the prediction ability of the MUXLink attack as it affects the DRNL value by bringing more changes to the structure of the circuit comparing the D-MUX scheme.

Fig. 13 illustrates the results for depth-based obfuscation. Regarding the locking parameters, we lock all the gates from the same level, considering a circuit as a directed graph. Thus, the "Level 1" bar shows accuracy and precision when we lock only all the gates from Level 1 of the circuit (depth 1) if we consider the netlist's graph as a directed graph. Note that primary inputs are at level 0 of this graph. The same goes for the other levels. In Fig. 13, the M corresponds to the middle level of a circuit. In this context, the term "middle level" specifically refers to the central level of each circuit if we consider the circuit as a directed graph. For instance, for the circuit c2670, the middle level corresponds to a level (depth) of 15, while for the b22 circuit, it aligns with a level (depth) of 31. In this figure, $M+$ corresponds to the scenario where we only lock all the nodes inside level $M+1$. In Table IV, we've provided the count of locked links for each level. Our analysis shows that locking each level provides various security and overhead trade-offs for various circuits. This figure shows that obfuscating all the output logic cones of the gates inside the same depth can decrease the accuracy of the MUXLink. This result can provide a basis for more research to find the best subset of the netlist's nodes for perturbation.

We use the SCOPE constant propagation attack to show the protection capability of the proposed techniques against non-ML-based structural attacks. Table V shows the K_AC and K_Prec of the SCOPE attack against D-MUX, improved D-MUX, and CUT-based protection schemes. Again we do not show the PI/PO results as for all the circuits, the values of K_AC and K_Prec are 0% and 100%, respectively. This table shows that the CUT-based technique can also thwart

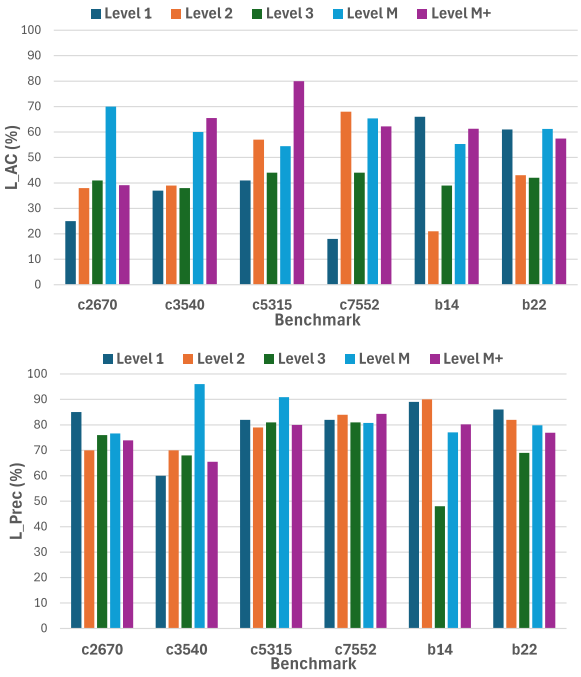
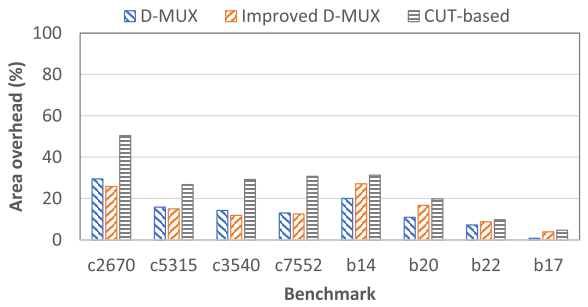


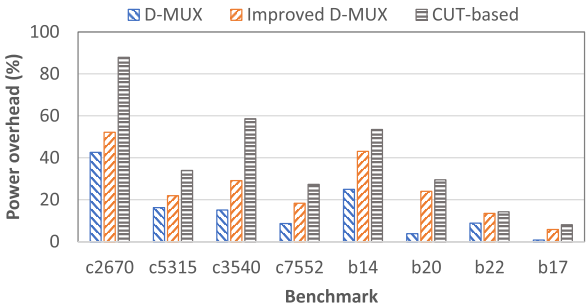
Fig. 13. Accuracy analysis of MUXLink attack against the depth-based interconnect-based obfuscation technique. For each level we lock only all the gates inside that level. Here Level M corrosponds to the middle level of each circuit and Level M+ is the level after Level M.

constant propagation attacks. Note that the SCOPE attack is more successful against improved DMUX comparing the default D-MUX. The reason is that improved D-MUX brings more changes to the CNF of the circuit that is utilized by the SCOPE attack. Although a CUT-based attack brings the same changes to the circuit, inserting an SB hinders the synthesis tool from optimizing the locked structure. As a result, re-synthesizing the design by setting the key values to *zero* and *one* does not provide useful information for SCOPE. Note that, for the CUT-based obfuscation technique, the average of K_AC is 30% and the average K_Prec is 70%. This shows that the SCOPE attack reports wrong values for 30% of the key bits. As the K_AC is also 30%, we can conclude that this attack provides only random guesses for the key bits it predicts (non-X value predicted key bits).

In the context of SAT attacks, many techniques have been proposed that are all susceptible to various oracle-less attacks [24]. It is imperative to integrate novel structural resilience mechanisms into circuits, along with anti-SAT



(a) Area overhead



(b) Power overhead

Fig. 14. a) Area overheads of the D-MUX, Improved D-MUX, and CUT-based obfuscation techniques for ISCAS-85 and ITC-99 benchmarks. b)Power overheads of the D-MUX, Improved D-MUX, and CUT-based obfuscation techniques for ISCAS-85 and ITC-99 benchmarks.

techniques. Our proposed techniques should complement anti-SAT schemes to stop both SAT and structural attacks.

3) Overhead Analysis: Fig. 14 shows the different techniques' overhead compared to the original circuits. To have a fair comparison for all the techniques, we lock 16 pairs of signals. We do not show the PI/PO technique's overhead separately since, as we mentioned, PI/PO locking is a variant of the improved D-MUX structure where all the locked pairs are selected from the PIs or POs of the circuit. This figure shows that the CUT-based technique poses a small overhead for bigger circuits. This shows that the CUT-based technique can be utilized to lock complex commercial circuits with low overheads. Note that although we select 16 pairs of signals for the improved D-MUX and CUT-based obfuscations, they lock more than 32 links since they lock all the fanout of the locked logic cells where D-MUX only locks 32 signals. We show the number of locked links (sum of fanout of the locked pairs) in Table III.

C. GREGO Analysis

1) Implementation: We use the available online OMLA repository to attack our proposed schemes [20]. We use the default values for subgraph training and classification for running the OMLA attack ($hidden\_dim = 64$, $num\_layers = 6$). For implementing the gate-type technique, we first synthesize the designs using GENUS tools and lock all the XOR/XNOR gates inside the circuits; Then, we re-synthesize this locked circuit, resulting in the final locked test circuit. For the training set, we randomly select 64 gates (except for locked gates) from the circuit and lock them by bringing XOR/XNOR gates connected to KIs. We do this 500 times, and finally, we synthesize all 500 training circuits using the GENUS tool.

We follow the same approach for the UES technique, but instead of locking XOR/XNOR gates for the test circuit, we lock all the selectors of the Multiplexers by bringing XOR/XNOR gates connected to the key bits.

2) Security Analysis: The OMLA attack reports the accuracy of the key prediction as Equation 6. This attack does not report un-guessed key bits (value X), so we only use the K_AC equation for evaluating the security of gate-type and UES techniques.

Table VI shows the outcome of the OMLA attack against the gate-type technique. As shown in the table, the accuracy of the OMLA attack is less than 60% for all the circuits that indicate this attack has randomly guessed the values of the key bits. The reason is that the subgraph structures adjacent to XOR/XNOR gates are not learned during the training as no XOR/XNOR gates exist to lock at the training phase.

Table VII shows the same results for the UES technique. The results of this table indicate that, like the gate-type technique, if the GNN is not trained based on the locked structures in the test circuit, it cannot infer the correct value for the key bits, and it only can provide random guesses.

3) Overhead Analysis: The size of keys is different for various benchmarks based on the number of XOR/XNOR for the gate type and MUX cells for UES techniques, respectively. So, we report the area overhead of these techniques along with the key size for each circuit. The key size shows the number of locked cells for each test circuit in Tables VI and VII. As shown in the tables, the gate-based obfuscation technique's overhead depends on the circuit structure and the technology library the designer uses. Note that the gate-type technique can lead to high overhead, particularly when a circuit contains many numbers of intended gate types (e.g., XOR/XNORs) inside a circuit, especially for very large circuits like a processor. So, security designers may consider targeting multiple UES sets, ensuring that the final locked circuit maintains a manageable level of overhead.

D. Output Corruption

A low output corruptibility in a logic locking scheme can lead to approximate attacks. Regarding approximate attacks in an Oracle-less environment, the attackers can only estimate the amount of output corruption as, based on the threat model, they have knowledge about the locking technique. For example, if a locking technique implemented on the circuit generally results in low output corruption, the attacker can conclude that using random keys results in low corruption for the circuit under attack. However, the attackers cannot use this circuit as it is not possible to distinguish the corrupted outputs. Knowing the exact corrupted outputs is important as, for most circuits, the significance of the output bits is different. For example, the corrupted bit can be a control signal or the most significant bit for multiplying two numbers.

Aside from the above discussion, in this paper, for interconnect obfuscation, we have proposed GRINN techniques, which are basically locked substructures that a security designer can utilize to lock any part of a circuit (any PI/PO for PI/PO obfuscation or any internal signals with the CUT-based obfuscation). These techniques can bring various output corruptions

TABLE V
SECURITY ANALYSIS OF THE D-MUX, IMPROVED D-MUX AND CUT-BASED OBFUSCATION TECHNIQUES AGAINST **SCOPE** ATTACK

Benchmark	D-MUX		Improved D-MUX		CUT-based	
	K_AC (%)	K_Prec (%)	K_AC (%)	K_Prec (%)	K_AC (%)	K_Prec (%)
c2670	43.7	62.5	32.3	76.4	33.8	74.6
c5315	40.6	62.5	44.1	64.7	36.2	73.7
c3540	37.5	68.7	29.4	82.5	26.3	73.61
c7552	40.6	78.1	52.9	79.4	34.5	66.6
b14	53.1	62.5	50	58.8	32.2	70
b20	53.1	68.7	52.9	61.7	26	72.3
b22	43.7	56.2	58.8	73.5	33.8	74.6
b17	40.6	68.7	70.5	79.4	26.5	73.4
Average	**44.1**	**65.9**	**48.8**	**72.0**	**31.1**	**72.3**

TABLE VI
SECURITY AND OVERHEAD ANALYSIS OF THE GATE-TYPE TECHNIQUE

Benchmark	Key size	Area overhead (%)	K_AC (%)
c2670	39	17.2	54.5
c3540	45	7.9	52.3
c5315	105	14.5	53.2
c7552	210	30.1	45
b14	120	14.2	48.3
b20	200	5.7	47.4
b22	430	8.0	51.57
b17	420	2.4	50.8

TABLE VII
SECURITY AND OVERHEAD ANALYSIS OF THE UES TECHNIQUE

Benchmark	Key size	Area overhead (%)	K_AC (%)
c2670	23	12.4	51
c3540	26	6.5	47.4
c5315	118	14.6	56.1
c7552	78	14.1	48.2
b14	252	21.2	52.7
b20	553	12.4	54.18
b22	681	10.7	47.4
b17	286	2.7	58

to the circuit based on key size and the signals that the security designer selects. Therefore, reporting the output corruption for an arbitrary scenario like random selection with a random number of keys would not provide useful information as various key sizes and signal selection approaches can lead to different output corruption. For the GRECO techniques, using various technology libraries, various gate types, and various UESs can lead to different output corruptions. Thus, like GRINN techniques, providing output corruption results for a special case cannot reflect the security of such schemes. The security designer can employ our proposed scheme in conjunction with node selection algorithms to achieve the desired level of output corruptibility within the circuit.

VII. CONCLUSION

Logic locking is a holistic design for trust approach that can protect the circuit through multiple stages of the IC manufacturing flow. However, recent ML-based structural attacks show the inherent structural vulnerability of this protection scheme. GNN-based attacks have the ability to utilize the structural similarity of the substructures inside the target circuits to infer the key values as each key structure confirms well to a class of substructures.

In this work, we first propose GRIN schemes that target GNN-based attacks against interconnect-based obfuscation. These techniques thwart the GNN-based attacks by omitting the path feature of the nodes inside the extracted subgraphs adjacent to the locking structures. Moreover, we show the depth-based obfuscation technique revealing that locking various circuit depths leads to different security levels. This information can be utilized for an algorithmic approach that places the locked structures into the circuit.

For gate-based obfuscation, we propose GREGO schemes. Gate-type and UES techniques stop the GNN-based attacks by locking all the similar substructures of the circuits. These techniques aim at the self-referencing model of the GNN-based attacks and prevent these attacks from learning useful information in the training phase. Moreover, we show that bringing artificial randomness using inversions cannot provide the intended security if the attack is trained adequately. Therefore, random inversion-based locking techniques do not compensate for the inherent vulnerability of gate-based logic locking. We analyze the security and overheads of our proposed techniques using ISCAS-85 and ITC-99 benchmarks. Experimental results show that our proposed techniques can thwart state-of-the-art GNN-based attacks.

REFERENCES

[1] J. A. Roy, F. Koushanfar, and I. L. Markov, "EPIC: Ending piracy of integrated circuits," in *Proc. Conf. Des. Automat. Test Eur.*, 2008, pp. 1069–1074.

[2] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security analysis of logic obfuscation," in *Proc. 49th Annu. Design Automat. Conf.*, 2012, pp. 83–89.

[3] 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.

[4] D. Sisejkovic, F. Merchant, L. M. Reimann, and R. Leupers, "Deceptive logic locking for hardware integrity protection against machine learning attacks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 6, pp. 1716–1729, Jun. 2022.

[5] Y.-W. Lee and N. A. Touba, "Improving logic obfuscation via logic cone analysis," in *Proc. 16th Latin-Amer. Test Symp. (LATS)*, Mar. 2015, pp. 1–6.

[6] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, May 2015, pp. 137–143.

[7] M. T. Rahman et al., "Defense-in-depth: A recipe for logic locking to prevail," *Integration*, vol. 72, pp. 39–57, May 2020.

[8] R. Purdy and R. D. S. Blanton, "Large-scale logic-locking attacks via simulation," in *Proc. 23rd Int. Symp. Quality Electron. Design (ISQED)*, Apr. 2022, pp. 1–6.

[9] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "InterLock: An intercorrelated logic and routing locking," in *Proc. 39th Int. Conf. Comput.-Aided Design*, 2020, pp. 1–9.

[10] S. D. Chowdhury, G. Zhang, Y. Hu, and P. Nuzzo, "Enhancing SAT-attack resiliency and cost-effectiveness of reconfigurable-logic-based circuit obfuscation," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2021, pp. 1–5.

[11] A. Saha, S. Saha, S. Chowdhury, D. Mukhopadhyay, and B. B. Bhattacharya, "LoPher: SAT-hardened logic embedding on block ciphers," in *Proc. 57th ACM/IEEE Design Automat. Conf. (DAC)*, Jul. 2020, pp. 1–6.

[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 *Proc. 56th Annu. Design Automat. Conf.*, 2019, pp. 1–6.

[13] A. Darjani, N. Kavand, S. Rai, M. Wijtvliet, and A. Kumar, "ENTANGLE: An enhanced logic-locking technique for thwarting SAT and structural attacks," in *Proc. Great Lakes Symp. VLSI*, 2022, pp. 147–151.

[14] A. Alaql and S. Bhunia, "SARO: Scalable attack-resistant logic locking," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 3724–3739, 2021.

[15] L. Mankali, L. Alrahis, S. Patnaik, J. Knechtel, and O. Sinanoglu, "Titan: Security analysis of large-scale hardware obfuscation using graph neural networks," *IEEE Trans. Inf. Forensics Security*, vol. 18, pp. 304–318, 2023.

[16] A. Alaql, D. Forte, and S. Bhunia, "Sweep to the secret: A constant propagation attack on logic locking," in *Proc. Asian Hardw. Oriented Secur. Trust Symp. (AsianHOST)*, Dec. 2019, pp. 1–6.

[17] A. Alaql, M. M. Rahman, and S. Bhunia, "SCOPE: Synthesis-based constant propagation attack on logic locking," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 8, pp. 1529–1542, Aug. 2021.

[18] P. Chakraborty, J. Cruz, and S. Bhunia, "SAIL: Machine learning guided structural analysis attack on hardware obfuscation," in *Proc. Asian Hardw. Oriented Secur. Trust Symp. (AsianHOST)*, Dec. 2018, pp. 56–61.

[19] L. Alrahis, S. Patnaik, M. A. Hanif, M. Shafique, and O. Sinanoglu, "UNTANGLE: Unlocking routing and logic obfuscation using graph neural networks-based link prediction," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Nov. 2021, pp. 1–9.

[20] L. Alrahis, S. Patnaik, M. Shafique, and O. Sinanoglu, "OMLA: An oracle-less machine learning-based attack on logic locking," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 3, pp. 1602–1606, Mar. 2022.

[21] L. Alrahis, S. Patnaik, M. Shafique, and O. Sinanoglu, "MuxLink: Circumventing learning-resilient MUX-locking using graph neural network-based link prediction," in *Proc. Design, Automat. Test in Eur. Conf. Exhib. (DATE)*, Mar. 2022, pp. 694–699.

[22] L. Li and A. Orailoglu, "Piercing logic locking keys through redundancy identification," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2019, pp. 540–545.

[23] P. Chakraborty, J. Cruz, and S. Bhunia, "SURF: Joint structural functional attack on logic locking," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, May 2019, pp. 181–190.

[24] L. Alrahis et al., "GNNUnlock: Graph neural networks-based oracle-less unlocking scheme for provably secure logic locking," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, Feb. 2021, pp. 780–785.

[25] H. M. Kamali, K. Z. Azar, F. Farahmandi, and M. Tehranipoor, "Advances in logic locking: Past, present, and prospects," Cryptol. ePrint Arch., Paper 2022/260, 2022. [Online]. Available: https://eprint.iacr.org/2022/260

[26] D. Sisejkovic, F. Merchant, L. M. Reimann, H. Srivastava, A. Hallawa, and R. Leupers, "Challenging the security of logic locking schemes in the era of deep learning: A Neuroevolutionary approach," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 17, no. 3, pp. 1–26, 2021.

[27] Y. Zhang et al., "TGA: An oracle-less and topology-guided attack on logic locking," in *Proc. 3rd ACM Workshop Attacks Solutions Hardw. Secur. Workshop*, 2019, pp. 75–83.

[28] L. Alrahis et al., "UNSAIL: Thwarting oracle-less machine learning attacks on logic locking," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 2508–2523, 2021.

[29] N. Limaye and O. Sinanoglu, "RESCUE: Resilient, scalable, high-corruption, compact-key-set locking framework," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 9, pp. 2826–2838, Dec. 2022.

[30] S. Elsharief, L. Alrahis, J. Knechtel, and O. Sinanoglu, "IsoLock: Thwarting link-prediction attacks on routing obfuscation by graph isomorphism," Cryptol. ePrint Arch., Paper 2022/1752, 2022. [Online]. Available: https://eprint.iacr.org/2022/1752

[31] A. Darjani, N. Kavand, S. Rai, and A. Kumar, "Discerning limitations of GNN-based attacks on logic locking," in *Proc. 60th ACM/IEEE Design Automat. Conf. (DAC)*, Jul. 2023, pp. 1–6.

[32] N. Limaye, E. Kalligeros, N. Karousos, I. G. Karybali, and O. Sinanoglu, "Thwarting all logic locking attacks: Dishonest Oracle with truly random logic locking," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 9, pp. 1740–1753, Sep. 2021.

[33] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Proc. AAAI Conf. Artif. Intell.*, 2018, vol. 32, no. 1, pp. 1–8.

[34] K. Shamsi and G. Zhao, "An oracle-less machine-learning attack against lookup-table-based logic locking," in *Proc. Great Lakes Symp. VLSI*, 2022, pp. 133–137.

[35] M. H. Moaiyeri, R. Faghih Mirzaee, K. Navi, T. Nikoubin, and O. Kavehei, "Novel direct designs for 3-input XOR function for low-power and high-speed applications," *Int. J. Electron.*, vol. 97, no. 6, pp. 647–662, Jun. 2010.

[36] P. Vijaya Lakshmi, S. Musala, A. Srinivasulu, and D. Pal, "Three novel single-stage full swing 3-input XOR," *Int. J. Electron.*, vol. 105, no. 8, pp. 1416–1432, Aug. 2018.

Armin Darjani received the M.Sc. degree in computer hardware architecture from the Sharif University of Technology, Tehran, Iran, in 2017. He is currently pursuing the Ph.D. degree with the Center for Advancing Electronics Dresden (CFAED) at Technische Universität Dresden. In March 2021, he joined the Chair of Processor Design with CFAED, as a Research Associate. His research interests include hardware security, reconfigurable architectures, and hardware/software co-design for secure systems.

Nima Kavand received the B.Sc. degree in computer engineering from Shahid Beheshti University in 2015 and the M.Sc. degree in computer system architecture from the Sharif University of Technology, Tehran, Iran, in 2017. He is currently pursuing the Ph.D. degree with Technische Universität Dresden, Dresden, Germany. His research interests include hardware security, circuit design, and VLSI design automation for emerging technologies.

Shubham Rai (Member, IEEE) received the Ph.D. (Dr.Ing.) degree (summa cum laude) from TU Dresden, Germany, in 2022. He is currently a Research Scientist with Robert Bosch GmbH, Renningen, Germany, in the field of AI deployment to enable HW-SW codesign for AI accelerators. His research interests include HW-SW codesign, emerging technologies, logic synthesis, and hardware security.

Akash Kumar (Senior Member, IEEE) received the joint Ph.D. degree in electrical engineering and embedded systems from Eindhoven University of Technology, Eindhoven, The Netherlands, and the National University of Singapore (NUS), Singapore, in 2009. From 2009 to 2015, he was with NUS. From October 2015 to March 2024, he was a Professor with TU Dresden, Germany, where he directed the Chair for Processor Design. Since April 2024, he has been directing the Chair of Embedded Systems with Ruhr University Bochum, Germany. His research interests include the design and analysis of low-power embedded multiprocessor systems, and designing secure systems with emerging nano-technologies.