Discerning Limitations of GNN-based Attacks on Logic Locking

Armin Darjani, Nima Kavand, Shubham Rai, Akash Kumar
Chair of Processor Design, CFAED, Technische Universität Dresden, Dresden, Germany
{armin.darjani, nima.kavand, shubham.rai, akash.kumar@tu-dresden.de}

***Abstract*—Machine learning (ML)-based attacks have revealed the possibility of utilizing neural networks to break locked circuits without needing functional chips (Oracle). Among ML approaches, GNN (graph neural networks)-based attacks are the most potent tools that attackers can employ as they exploit graph structures inherent to a circuit's netlist. Although promising, in this paper, we reveal that GNNs have some impediments in attacking locked circuits. We investigate the limits of the state-of-the-art GNN-based attacks against logic locking and show that we can drastically decrease the accuracy of these attacks by utilizing these limitations in the locking process.**

***Index Terms*—Logic locking, Structural attacks, ML-based attacks, GNN**

I. Introduction

Logic locking is a design-for-trust technique that offers protection through various stages of the IC supply chain [1]–[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.

Logic-locking techniques can be categorized into gate and interconnect obfuscation. Interconnect obfuscation perturbs the netlist by injecting Multiplexers as new nodes into the design and connecting these nodes to the correct and arbitrary nodes from the circuit's netlist [4].

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

Although effective, various attacks have questioned the security of the logic locking technique. The SAT [5] attack was the first attack that challenged the security of logic locking. This powerful attack utilizes a functional duplicate of the target circuit (an oracle) to infer the valid key. This attack works based on the functionality of the circuit. By changing the primary inputs and key bits of the extracted target netlist and comparing the output with the Oracle SAT can infer the valid key set.

Although the SAT attack has been the standard attack for many years, its threat model and efficiency have been questioned. The main assumption in the threat model of the SAT attack is access to an oracle. This assumption may need to be revised for two reasons. Firstly, accessing a functional chip is not always possible; secondly, if such access is provided, it can lead to more powerful attacks such as probing [6].

Regarding efficiency, the SAT attack cannot break circuits with SAT-hard functions like multipliers. Moreover, breaking logic locking in large complex circuits is challenging for the SAT attack [7].

With regard to these shortcomings, many structural-based attacks have been proposed by the security community [8]–[13]. These attacks omit the need for an oracle from the threat model. They utilize the similarity in the netlist of the circuits as a basis to trace back the changes introduced into the circuit by logic locking.

Structural attacks have three main advantages over SAT attacks. Firstly they omit the need for an oracle leading to a more realistic threat model. Secondly, they can break logic locking in circuits with SAT-hard functions. And finally, they result better in attacking large and complex circuits where SAT fails to infer the valid key set.

Among the structural attacks, ML-based approaches have proved to be the most promising attacks [10]. The final netlist of a circuit is a product of security-agnostic synthesis tools. These tools only consider the circuit's area, delay, and performance when synthesizing an RTL code to a netlist. 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 [14].

The ML-based attacks revealed that although adding locking structures and re-synthesizing the design can lead to changes in a netlist, these changes are confined to the adjacency of the locked gates [10], [12]. As a result, by learning the small substructures of a circuit and by training the model based on the locking scheme, the ML model can infer the original substructure of the locked gates.

Among ML approaches, GNNs are the most promising one [12], [13] since the structure of a netlist is 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 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.

In this paper, we first investigate the challenges of protecting a circuit against GNN-based attacks. Then we show the limitations of GNN-based attacks and demonstrate how

by harnessing these limitations, the circuits can be protected against such attacks.

To the best of our knowledge, this is the first work that presents the limitations of neural network-based attacks on logic locking and provides the community with insights about possible measurements to thwart such attacks. The contributions of this paper 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 perturbations are possible in a circuit's graph, making circuit protection more difficult.

Limitations of GNNs in attacking interconnect obfuscation: We show the limits of GNNs in attacking interconnect-based logic locking techniques. We provide insights and approaches for customizing gate selection in the locking process that can decrease the accuracy of the GNNs by exploiting the graph structure of the circuits.

Limitations of GNNs in attacking gate obfuscation: We investigate the limits of GNNs in attacking gate obfuscation-based logic locking techniques. We provide insights and approaches for customizing gate selection in the locking process that can decrease the accuracy of the GNNs based on the self-referencing model of GNN-based attacks.

Analyzing the proposed approach: We analyze the proposed approaches using the ISCAS-85 benchmark and two of the state-of-the-art attacks and discuss the possible research directions to find countermeasures against GNN-based attacks.

II. GNN-based Attacks on Logic Locking

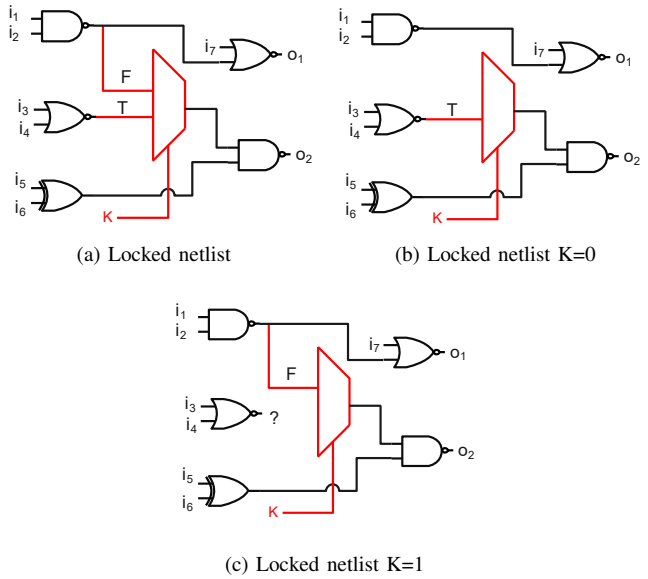
Any synthesized locked circuit is a netlist which is basically a graph, with edges being a network of wires between the gates. There is a good fit between the topology of an extracted netlist locality and the network of wires between gates. The attack's aim is to extract the value of the key bits from 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 [12]. In this section, we discuss two state-of-the-art attacks that harness the GNN to extract the keys from the structure of the netlists.

A. Threat Model

In GNN-based attacks, the adversary is located in an untrusted fab with access to the GDSII file. The attacker can obtain the netlist by reverse engineering this GDSII file. After that, distinguishing key inputs is easy as these particular inputs are connected to a tamper-proof on-chip memory. Moreover, the foundry has access to the standard cell library. Note that the attacker has no access to an oracle which makes GNN-based attack models more challenging than SAT attacks.

B. GNN-based attack against interconnect obfuscation

Fig. 1a shows an interconnect obfuscation scheme where one of the inputs of the target gate is obfuscated using a



(a) Locked netlist (b) Locked netlist K=0

(c) Locked netlist K=1

Fig. 1. Interconnect obfuscation. a) Locked netlist, b) Netlist after assigning the correct value to the key, c) Netlist after assigning an incorrect value to the key

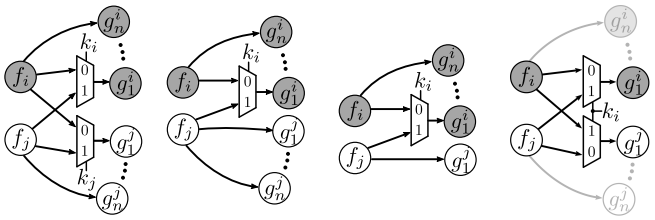


Fig. 2. Various D-MUX strategies [4]

Multiplexer connected to the correct cone and an arbitrary cone from the circuit. The valid key chooses the correct logic cone, whereas a wrong key can lead 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. 1c, setting the key bit to a wrong value leads to circuit reduction after re-synthesizing where choosing the correct path does not leave any dangling connections (Fig. 1b) Constant propagation attacks like SWEEP [8] and SCOPE [9] 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 extracted features to correct key values.

In [4] authors proposed D-MUX that changes the naive interconnect obfuscation to stop constant propagation attacks. This technique is shown in Fig. 2. This technique utilizes four strategies to select the target gates for locking. Then it brings Multiplexers into the circuit and obfuscates one of the outputs of the selected gates. This technique thwarts constant

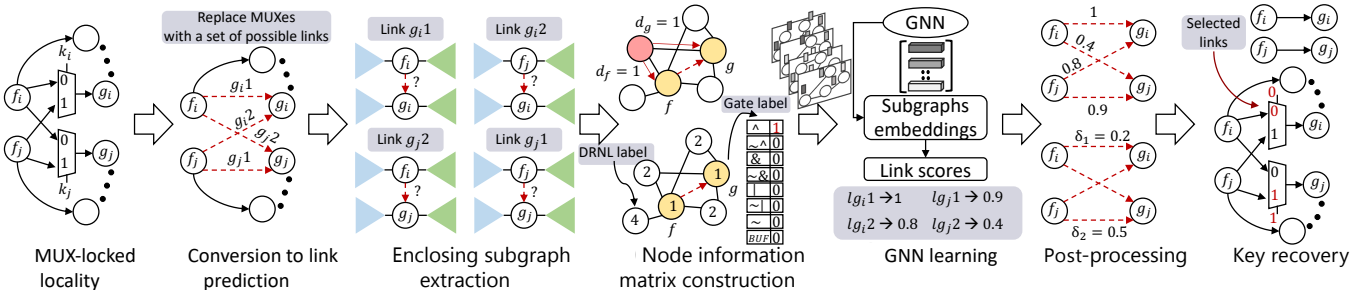


Fig. 3. MUXLink attack [13] converts the problem of guessing correct keys to a link prediction problem and predicts the correct connection utilizing GNN

propagation delay by stopping the synthesis tools from circuit reduction as for both values of each key in a D-MUX structure; both cones should be available in the circuit.

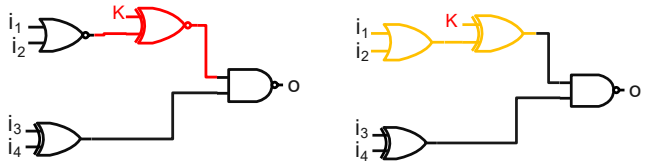
Although the D-MUX technique can thwart the constant propagation attacks, the authors in [13] proposed MUXLink, which showed that this technique is vulnerable when the attack is based on the link representation of the target circuit since D-MUX brings local modifications to the circuit that can be reversed based on the extracted features of the circuits' structure.

MUXLink attacks the interconnect obfuscation techniques by converting the logic locking scheme to a link prediction problem. Then it uses GNN to infer the correct links based on the features of the enclosing subgraph of the locked units. Fig. 3 shows the flow of the MUXLink attack. MUXLink extracts the h-hop subgraphs of the target nodes. Each node in this subgraph will be assigned an information matrix and tag corresponding to its relationship with the target nodes. This tag is calculated for each node in the subgraph based on the shortest path distance of each node to the target nodes (nodes f_i, f_j, g_i , and g_j). Finally, the GNN predicts the probability of each link, and post-processing uses these numbers to predict the correct links.

C. GNN-based attack against gate obfuscation

Fig. 4a shows a gate obfuscation scheme where an XOR key gate is placed in front of a logic gate from the design. 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 one here. However, in SAIL [10] the authors showed that re-synthesizing leads to local changes (Fig. 4b) that can be learned and used to trace back the changes and finally infer the correct key values. Although powerful, SAIL has shortcomings like complex learning models [12].

OMLA overcomes the shortcomings of the SAIL by utilizing the GNNs. Fig. 5 shows the general flow of the OMLA attack. This attack converts the circuit's netlist to an undirected graph and maps the key guessing problem to subgraph classification. Like MUXLink, Each node in this subgraph will be assigned an information matrix and a label corresponding to its relationship with the target key node. Each label corresponds to the distance of the gates from the key gate based on the number of hops.



(a) Before re-synthesis (b) After re-synthesis

Fig. 4. Gate obfuscation. a) Before re-synthesis. b) After re-synthesis

For the training phase, OMLA follows the self-referencing model. The locked target netlist is used to generate training/validation sets. This way, the model can learn the biases of the target circuit and infer the keys more accurately.

III. Limitations of 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 [14]. This means that 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 [10], [12]. 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.

To protect the design from GNN attacks, security designers should add perturbations to the netlist of the circuit in such a way that the GNN fails to infer the correct key values from the information learned during the training phase.

In this section, we first discuss the challenges for protecting circuits against GNN-based attacks; then, we show that customizing selection during the locking process can drastically decrease the accuracy of the GNN-based attacks for both interconnect and gate logic locking.

A. Challenges for protecting circuits against GNN

As mentioned, to protect the design from GNN-based attacks, the security designer should add some perturbations to the circuit's netlist. Fig. 6 shows various perturbation types that can be used to change the prediction of the GNNs. However, not all perturbations apply in a circuit's netlist. In

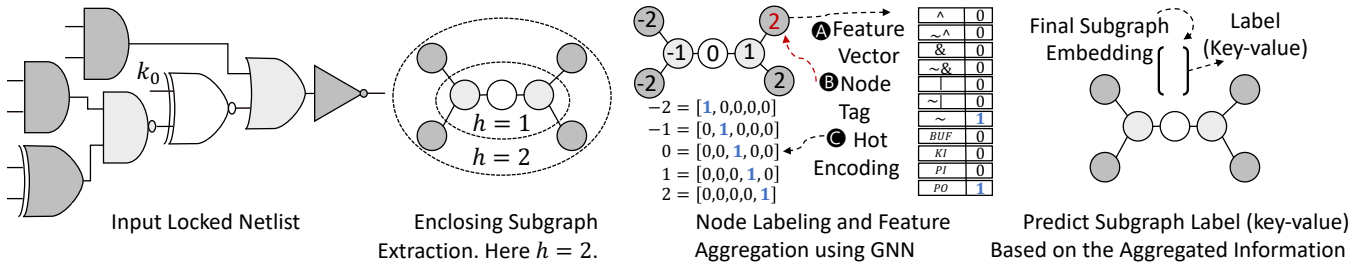
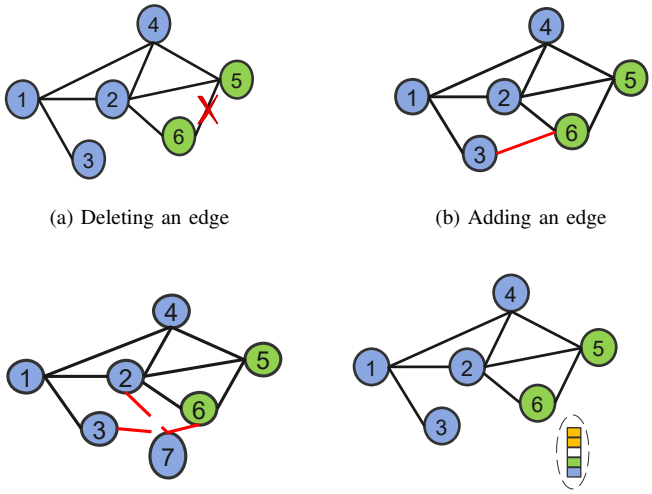


Fig. 5. OMLA attack [12] converts the netlist to a graph and guesses the correct key values based on the enclosing subgraph structures utilizing GNN



(a) Deleting an edge

(b) Adding an edge

(c) Node injection

(d) Feature modification

Fig. 6. Various graph perturbations

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 circuits' 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 [15] that can further be utilized to attack the circuit.
Adding an edge: This perturbation can be interpreted as connecting two gates in the circuit. Like deleting an edge, this type of perturbation can lead to unwanted output values in the circuit unless it is followed by a corrective add-on logic in the circuit that leaves traces [15].
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. Re-synthesizing a circuit cannot omit this perturbation. However, GNN-based attacks have shown that this perturbation only brings local changes to the netlist that can be learned and utilized for the attack.

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). Changing these features in a netlist may affect the attacks; however, the security designer cannot rely on this type of perturbation. The reason is that the attackers have access to the netlist based on the threat model. So, they can re-synthesize the design using the intended technology library. As a result, these features can be changed, and the attacker can continue the attack with the new netlist.

In the following, we discuss how the structural aware selection of a subset of nodes in a netlist for bringing node injection perturbation can stop the GNNs from learning meaningful information that leads to thwarting the GNN-based attacks.

B. Thwarting GNN-based attacks against interconnect obfuscation

Interconnect obfuscation perturbs the netlist by injecting Multiplexers as new nodes into the design and connecting these nodes to the correct and arbitrary nodes from the graph. As mentioned, GNN-based attacks predict the valid link based on the extracted locality of the locked nodes.

To evaluate the limitations of the GNN-based attacks against interconnect obfuscation, we used the MUXLink attack. We followed two approaches to decrease the accuracy of the MUXLink.

In the first approach, we locked some of the circuit's PIs and POs using two input switch boxes (SB). Fig. 7 shows this locking scheme. Here all the connections to PIs are now connected to the add-on Multiplexers. Moreover, a buffer is connected to each PI and PO to enable the MUXLink attack, as this attack does not consider PIs and POs as nodes. The intuition behind this approach is that PIs have the same feature vector and no predecessors in the graph. We are disconnecting them from their successors using Multiplexers in the link prediction phase. So, the GNN has no tool to distinguish them and predict the correct keys. The same goes with POs, as they have the same feature vector and no successor, if we disconnect them from their predecessors, GNN cannot classify them. In section IV, we show that obfuscating output interconnects of PIs and input interconnects of POs can ultimately thwart the MUXLink attack.

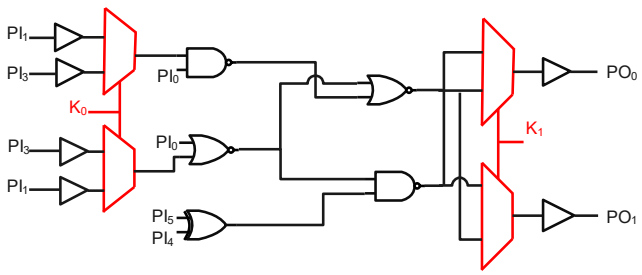


Fig. 7. Locking POs and PIs of a circuit

In the second approach, we perturb nodes from the same depth in the graph corresponding to gates from the same level in the circuit. Fig. 8 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 it chooses only one of the outputs of the two target nodes to obfuscate.

There are two intuitions behind this approach. Firstly, as the circuit structure is uniform, there is a good chance that nodes from the same level confirm well to the same predecessor and successor structures. Secondly, injecting multiple perturbations in the same vicinity of the graph can adversely affect the attack by affecting the labels of the nodes connected to the target nodes. Note that MUXLink uses double radius node labeling (DRNL) calculated based on the shortest path between nodes with a three-hop distance to the target nodes. So, the chance of manipulating these labels will be higher with multiple perturbations in the same vicinity. In section IV, we show that this approach can decrease the accuracy of the MUXLink comparing the D-MUX approach.

C. Thwarting GNN-based attacks against gate obfuscation

Like interconnect obfuscation, gate obfuscation 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. We use the OMLA attack to evaluate the limitations of the GNN-based attacks against gate obfuscation. OMLA follows a self-referencing model for attacking circuits as training based on the target circuit 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 (only 66% accuracy for the best case with a generic training model [12]). However, we show that this strength can be used against the attacker.

To show the limitations of the self-referencing model, we locked all and only the XOR and XNOR gates of the circuit. The intuition behind this technique is that the extracted local subgraphs of the XOR and XNOR gates will probably have similar structures different from other parts of the circuit. So if we lock all of the XOR and XNOR gates, there will be no XOR and XNOR gates left for training. As a result, the training phase cannot learn any useful information about the locked structures.

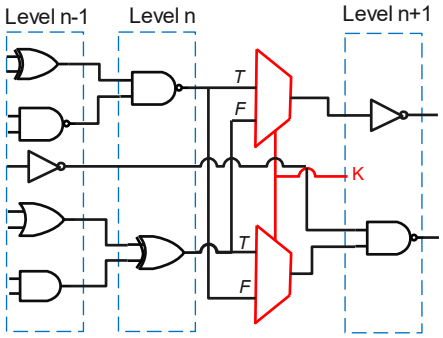


Fig. 8. Locking fanout cone of gates in same depth

In section IV we show how locking all XOR and XNOR gates can decrease the accuracy of the OMLA attack.

IV. Experimental Results

A. Experimental Setup

We used ISCAS-85 combinational benchmark sets to evaluate our approaches using MUXLink and OMLA attacks. The level of protection of each approach against the attack is evaluated using key-prediction accuracy (KPA) which shows the percentage of correctly guessed keys. Moreover, we report the key-guessing accuracy (KGA) for the MUXLink attack as this attack is capable of reporting X instead of Zero or One values when it fails to guess the correct key. KPA and KGA are calculated as follows:

$$KPA = \frac{KEY_{Correct}}{KEY_{Total}} * 100\% \quad (1)$$

$$KGA = \frac{KEY_{Correct} + KEY_X}{KEY_{Total}} * 100\% \quad (2)$$

All the attack codes have been compiled and run on a single node with an Intel processor running at 4 GHz and 24 GB of RAM with UBUNTU 20.04.3.

B. Analyzing limitations of GNN-based attacks against interconnect obfuscation

To assess our approaches against GNN-based attacks on interconnect obfuscation, we first locked the PIs and POs of the benchmark circuits using the obfuscation scheme shown in Fig. 7.

To evaluate the effects of obfuscating all the output logic cones of the gates inside the same depth, we separately obfuscated the circuit's first, second, and third levels, once with our approach and once with the D-MUX approach. Firstly, we only locked all the gates from the first level of the circuit. Then we attacked the circuit using MUXLink. We followed the same approach only for gates on the second level of the circuit, and finally, we locked and attacked the gates from the third level of the circuit.

Table I shows the results of the MUXLink attack for various levels and approaches. As shown in the table, obfuscating PIs and POs thwart the MUXLink attack, as this attack will not

TABLE I
ANALYZING THE LIMITATIONS OF GNN-BASED ATTACKS AGAINST INTERCONNECT OBFUSCATION

Benchmark	PI locking		Level 1 locking				Level 2 locking				Level 3 locking				PO locking	
	Our approach		D-MUX		Our approach		D-MUX		Our approach		D-MUX		Our approach		Our approach	
	KPA%	KGA%	KPA%	KGA%	KPA%	KGA%	KPA%	KGA%	KPA%	KGA%	KPA%	KGA%	KPA%	KGA%	KPA%	KGA%
c1355	0	100	100	100	80	80	9	100	0	100	100	100	0	100	0	100
c1908	0	100	67.3	82.6	19.14	77.3	84.37	93.7	16.66	82.5	88.8	91.66	3.2	96.2	0	100
c5315	0	100	93.75	96.8	8.3	86.9	86.2	100	79.93	83.2	91.11	100	61.53	84.61	0	100
c7552	0	100	35.93	65.62	19.82	76.72	90.62	93.75	54.10	71.01	79.11	100	53.11	75.2	0	100
c2670	0	100	70	86	10	92.5	97.05	97.05	67.7	85.9	32	74	19.23	80.7	0	100
c3540	0	100	66.66	74.35	44.93	69.23	61.36	82.9	49.01	68.6	52.27	75	49.3	67.28	0	100

TABLE II
ANALYZING THE LIMITATIONS OF GNN-BASED ATTACKS AGAINST GATE OBFUSCATION

Benchmark	Obs %	Key size	KPA %
c1355	58	113	51.2
c1908	34.5	68	49.11
c5315	8.4	101	58.4
c7552	15.6	204	46.44
c2670	9.4	47	52.4
c3540	5.6	42	51.33

extract meaningful structures from the PIs and POs subgraph. This table also shows that obfuscating all the output logic cones of the gates inside the same depth can decrease the accuracy of the MUXLink. Additionally, we can see that different security levels are achieved by locking different depths of circuits. This result can provide a basis for more research to find the best subset of the netlist's nodes for perturbation.

C. Analyzing limitations of GNN-based attacks against gate obfuscation

To show the limitations of the self-referencing model of the GNN-based attacks on gate obfuscation, we lock all the XORs and XNORs inside the circuits. This way, the attacker can only lock other types of gates for the training.

We used the YOSYS tool with a synthetic library containing BUFF, NOT, and all two-input logic gates for technology mapping. Note that based on [12], OMLA is agnostic to both synthesis tools and technology libraries. So, this setup is valid for the attack. For training, we locked the target netlist 500 times. We used XOR/XNOR gates as key gates for both test and training sets. In the test set, all the XOR/XNOR gates of the original circuits are locked by adding key XOR/XNOR gates. In the training set, we locked the remaining gates of the circuits randomly using XOR/XNOR key gates with a key size of 64.

The results of the attack on various circuits are shown in table II. For multiple circuits in the benchmark, the percentage of locked gates is also provided to demonstrate the overhead of this technique. These results show that the self-referencing model, which is the strength of the GNN-based attack, can be circumvented by locking all the structures with the same footprint in the circuit. Security designers can further utilize these results to develop locking algorithms that aim to find such structures in the circuit.

V. SUMMARY

In this paper, we investigated the limitations of the GNN-based attacks on the logic locking technique. For the first time, we discussed the challenges of protecting circuits against GNN-based attacks and proposed various approaches to overcome these challenges by limiting the learning capabilities of the GNNs. We showed that locking the first and last stages of the circuits can thwart GNN-based attacks against interconnect obfuscation. Moreover, we showed that obfuscating all the gates in the same depth of a circuit can decrease the accuracy of the GNN-based attacks. For gate obfuscation, we showed that limiting the self-referencing model of the GNN leads to preventing attacks.

ACKNOWLEDGMENT

This work was partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation)—Project Number 439891087—SecuReFET and the German Federal Ministry for Education and Research (BMBF) under the framework of VE-CirroStrato.

REFERENCES

[1] J. A. Roy *et al.*, "Markov. epic: Ending piracy of integrated circuits," in *DATE*, 2008.
[2] J. Rajendran *et al.*, "Security analysis of logic obfuscation," in *DAC*, 2012.
[3] M. Yasin *et al.*, "On improving the security of logic locking," *IEEE TCAD*, 2015.
[4] D. Sisejkovic *et al.*, "Deceptive logic locking for hardware integrity protection against machine learning attacks," *IEEE TCAD*, 2021.
[5] P. Subramanyan *et al.*, "Evaluating the security of logic encryption algorithms," in *HOST*, 2015.
[6] M. T. Rahman *et al.*, "Defense-in-depth: A recipe for logic locking to prevail," *Integration*, 2020.
[7] L. Mankali *et al.*, "Titan: Security analysis of large-scale hardware obfuscation using graph neural networks," *IEEE TIFS*, 2022.
[8] A. Alaql *et al.*, "Sweep to the secret: A constant propagation attack on logic locking," in *AsianHOST*, 2019.
[9] A. Alaql, *et al.*, "Scope: Synthesis-based constant propagation attack on logic locking," *IEEE TVLSI*, 2021.
[10] P. Chakraborty *et al.*, "Sail: Machine learning guided structural analysis attack on hardware obfuscation," in *AsianHOST*, 2018.
[11] L. Alrahis *et al.*, "Untangle: unlocking routing and logic obfuscation using graph neural networks-based link prediction," in *ICCAD*, 2021.
[12] L. Alrahis, *et al.*, "OMLA: An oracle-less machine learning-based attack on logic locking," *IEEE TCASII: Express Briefs*, 2021.
[13] L. Alrahis *et al.*, "MuxLink: circumventing learning-resilient mux-locking using graph neural network-based link prediction," in *DATE*, 2022.
[14] Y. Zhang *et al.*, "TGA: An oracle-less and topology-guided attack on logic locking," in *Proceedings of the 3rd ACM Workshop on Attacks and Solutions in Hardware Security Workshop*, 2019.
[15] L. Alrahis *et al.*, "GNNUnlock: Graph neural networks-based oracle-less unlocking scheme for provably secure logic locking," in *DATE*, 2021.