LIPSTICK: Corruptibility-Aware and Explainable Graph Neural Network-based Oracle-Less Attack on Logic Locking

Yeganeh Aghamohammadi
University of California, Santa Barbara
Santa Barbara, CA, USA
yeganeh@ucsb.edu

Amin Rezaei
California State University, Long Beach
Long Beach, CA, USA
amin.rezaei@csulb.edu

2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC) | 979-8-3503-9354-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/ASP-DAC58780.2024.10473982

***Abstract*—In a zero-trust fabless paradigm, designers are increasingly concerned about hardware-based attacks on the semiconductor supply chain. Logic locking is a design-for-trust method that adds extra key-controlled gates in the circuits to prevent hardware intellectual property theft and overproduction. While attackers have traditionally relied on an oracle to attack logic-locked circuits, machine learning attacks have shown the ability to retrieve the secret key even without access to an oracle. In this paper, we first examine the limitations of state-of-the-art machine learning attacks and argue that the use of key hamming distance as the sole model-guiding structural metric is not always useful. Then, we develop, train, and test a corruptibility-aware graph neural network-based oracle-less attack on logic locking that takes into consideration both the structure and the behavior of the circuits. Our model is explainable in the sense that we analyze what the machine learning model has interpreted in the training process and how it can perform a successful attack. Chip designers may find this information beneficial in securing their designs while avoiding incremental fixes.**

***Index Terms*—Logic Locking, Logic Encryption, Machine Learning, Graph Neural Networks, Corruptibility, Explainability**

I. Introduction

While outsource manufacturing is becoming the norm in the semiconductor industry, it comes with increasing threats such as hardware Intellectual Property (IP) theft and overproduction. Logic locking [1]–[14] (a.k.a. logic encryption or logic obfuscation) is a technique to safeguard against these attacks by adding extra key-controlled gates to the circuits.

One of the well-studied threat models is the Oracle-Guided (OG) type of attacks [15]–[20], which assume having access to an activated Integrated Circuit (IC) purchased off the shelf in addition to a logic-locked netlist leaked from an untrusted foundry. With the aid of a Boolean satisfiability (SAT) solver, OG attacks try to prune out subsets of wrong keys by checking a relatively small number of input patterns. While the semiconductor industry must yet come up with protective mechanisms against OG attacks, Oracle-Less (OL) attacks can be detrimentally pervasive in the sense that they can leak confidential information to attackers with limited resources. Basically, we believe the weaker and yet more effective the attacker model is, the more it can be ubiquitous and harmful to hardware IP owners, and thus more critical to safeguard.

Recent advancements in Machine Learning (ML) have made it possible to propose OL attacks to predict the correct key of logic-locked circuits [21]–[26]. One of the suitable ML models for logic circuits is Graph Neural Network (GNN) which accepts inputs in the form of graphs and handles non-Euclidean data. GNNs are capable of learning the connections between diverse nodes and edges in a network [27] and thus recognizing patterns in graph-structured data, such as the schematic of a logic circuit. In state-of-the-art ML-based OL attacks, the success rate is reported in terms of ML model prediction accuracy, which is defined as minimizing the hamming distance between the correct key and the key reported by the attack.

A. Research Gap

The first observation is that ML-based OL attacks are inherently *approximate* attacks because of the fact that they try to find out an approximately correct key with respect to some parameters, for example, by reducing the hamming distance between the correct and reported keys. The second finding is that current OL attacks do not consider the behavior of the circuits under the reported key compared with the intended functionality, necessitating the use of more meaningful metrics, such as *key precision*, which takes into account output corruptibility in addition to circuit structure. The third observation is that a holistic security assessment of logic-locking techniques is overlooked, yet *explainable* ML models [28], which provide us with more reliable and trustworthy predictions, need to be taken into consideration.

B. Contributions

Motivated by the mentioned research gaps, in this paper, we are answering the following questions. First, *why does the accuracy of current GNN attacks differ drastically from the reported key's precision? Can integrating circuit functionality metrics into GNN models result in the discovery of a more relevant key?* We are particularly interested in understanding why hamming distance alone is insufficient to access the effectiveness of GNN attacks and which parameter or set of parameters are meaningful to use here. Second, *what features of the logic-locked circuits led the model to infer the reported key? What is the degree of each feature's influence?* Answering these questions will allow the designer to more securely and confidently pick from many proposed logic locking methods based on the design's trade-offs and circuit characteristics.

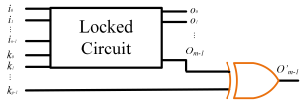

Fig. 1: Counterexample for hamming distance as a key precision metric

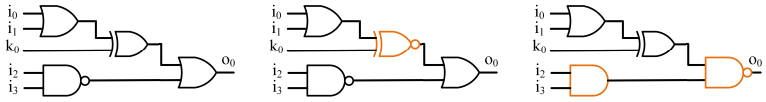

Fig. 2: Counterexample for the model prediction accuracy of state-of-the-art GNN-based attacks

Specifically, we introduce a corruptibility-aware and explainable GNN-based OL attack to predict the correct key of logic-locked ICs with a high key precision. To comprehend data behavior and identify relationships between nodes, edges, and node features, an explainable GNN would employ techniques including rule extraction and explainable reasoning [29] which, in the context of logic locking attacks, would benefit us in figuring out crucial patterns and topologies that are effective in choosing the key-bit values. The contributions of this paper are threefold:

- Proposing a novel and effective GNN-based OL attack on logic locking that takes the circuit's functionality into account in addition to its structure.
- Providing explainability of the inferred key by the proposed attack that functions as a rule-of-thumb for designers on how to safeguard their precious hardware designs.
- Showcasing the model's prediction accuracy and key precision on seen and unseen logic-locked benchmarks.

C. Background

In XOR-based logic locking [1], the key-bits can be matched with a combination of random inverters and buffers. Then, key-bit-controlled XOR gates are used to replace selected buffers and inverters. If an XOR gate is hiding a buffer, the correct key-bit is "0" while if it is hiding an inverter, the correct key-bit is "1". Additionally, MUX-based logic locking [2] chooses random signals and replaces them with 2-1 MUXs whose inputs are real signals and random dummy ones, and selectors are the key-bits. As a result, the correct key must select the real signal terminal and avoid the dummy one. Moreover, LUT-based logic locking [3] is being implemented to IC prefabrication to separate the inputs from the outputs so that a barrier stays between every path from inputs to outputs. In this locking method, the values stored in the Look Up Tables (LUTs) are the key inputs.

While traditional logic locking methods have been attacked by the OG SAT-based attack [15], post-SAT locking schemes such as SAR-Lock [4] and Anti-SAT [5] have been proposed to exponentially increase the number of input patterns that are required to prune wrong keys by the SAT-based attack. As an advanced post-SAT method, Bilateral Logic Encryption (BLE) [6] uses obfuscation and integrated locking on a sensitive component of a circuit. With this approach, the performance overhead is lower than locking the entire circuit, but the security impact, including structural complexity and logic complexity, is transferred to the whole circuit.

Recent studies have shown promising results in the advancement of ML-based OL attacks. SnapShot [21] employs neuro-evolutionary and deep learning methods and is the first of its kind to directly predict the key value from a locked synthesized gate-level netlist. In addition, SAIL [22] recovers the design of a locked circuit in gate-level netlist and extracts circuit features with the help of ML-based structural analysis. While SAIL works mostly on the XOR-based locked circuits, CutSAIL [23] derives missing k-cuts from the neighboring logic and predicts the functionality of the missing parts of the locked circuit. However, UNSAIL [7] inserts unsuited data during the training stage of an ML attack, causing the ML model to predict labels wrong.

More recently, OMLA [24] employs a GNN model to predict the keys of a locked circuit by deriving a small subgraph for each key gate. As a result, the key-bit value of a subgraph is also considered its label. GNNUnlock [25], utilizes GNN for node classification of circuits. The dataset for GNNUnlock is multiple logic-locked circuits of a single benchmark with different key sizes. The model uses adjacency matrices corresponding to the circuit, in which edges and nodes represent wires and gates, respectively. Finally, by mapping the key extraction process to a link prediction problem, UNTANGLE [26] gathers concealed links in the lock blocks and then learns the circuit structure, gate features, and link features.

The inability of ML models to be interpreted is one of their main drawbacks. This limitation can be overcome by creating post-hoc explanation procedures for predictions, giving rise to the explainability field [30]. For the purpose of producing accurate similarity estimations and discriminative feature representations, SGGNN [31] uses graph computation during both the training and testing phases of deep networks. In addition, PGExplainer [32] uses the trained GNN model as input and offers coherent justifications for the model's predictions. An interesting fact about PGExplainer is that it can be used in an inductive scenario to infer explanations of ambiguous nodes without having to retrain the explanation model.

II. Preliminary Study

We want to first make a distinction between two terms: One is ML model *prediction accuracy* which resembles how well a given prediction matches its actual value, and the other is *key precision* which shows how closely a logic-locked circuit under a given key operates to the original circuit.

Now, considering n, m, and p be the sizes of the input, output, and key respectively, we define the original circuit as $\mathbb{F} : \{0,1\}^n \rightarrow \{0,1\}^m$, and the locked circuit as $\mathbb{G} : \{0,1\}^p \times \{0,1\}^n \rightarrow \{0,1\}^m$ in which there is a p-bit correct key $K^* = (k_0^*, k_1^*, ..., k_{p-1}^*) : \{0,1\}^p$ such that $\mathbb{F}(X) = \mathbb{G}(X, K^*)$. We

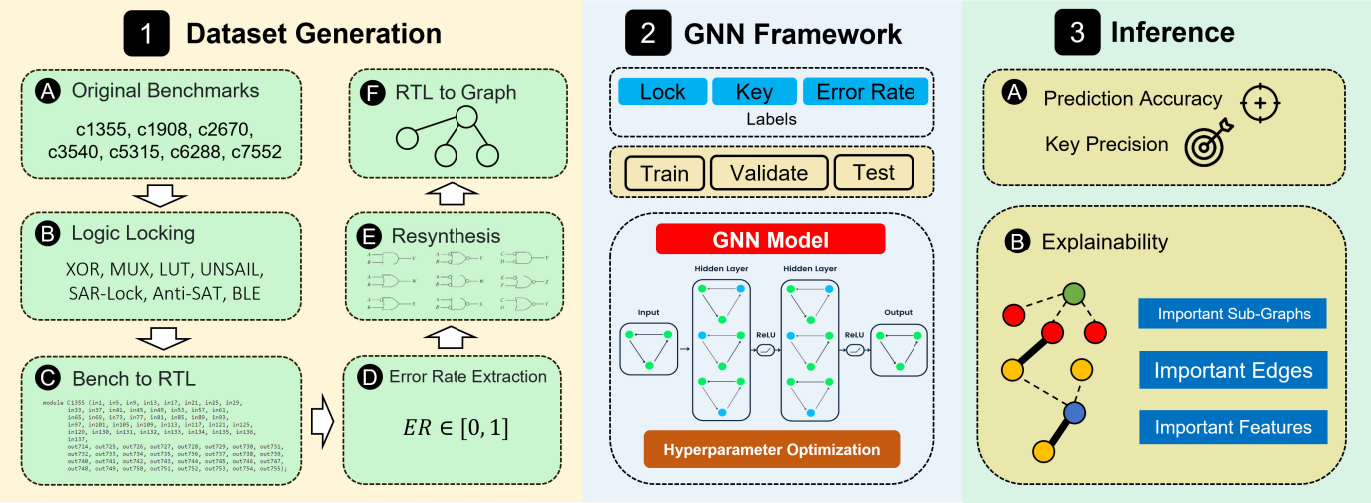

Fig. 3: LIPSTICK attack framework

also define the key error rate $ER(K)$ for a given key K of the locked circuit $\mathbb{G}$, as the number of input patterns in which $\mathbb{F}(X) \neq \mathbb{G}(X, K)$, divided by all the input patterns. It is clear that for the correct key, $ER(K^*) = 0$. Thus, the ER value is fundamentally dependent on the circuit's functionality rather than its topology.

Let $K^a = (k_0^a, k_1^a, ..., k_{p-1}^a)$ be a p-bit reported key by the attack, the Hamming Distance (HD) of K^a and K^* can be defined as the sum of the bitwise XOR of the two keys as follows:

$$HD(K^a, K^*) = \sum_{i=0}^{p-1} k_i^a \oplus k_i^* : \{0, 1, ..., p\} \quad (1)$$

Proposition 1: The smaller the $HD(K^a, K^*)$, the higher the key precision of the locked circuit $\mathbb{G}$ under K^a.

Counterexample 1: Consider the locked circuit in Fig. 1 with a key size of $p-1$. We increase the key size to p by XORing one of the outputs with additional key-bit k_{p-1}. In this case, there is a key K^a in which just the key-bit k_{p-1} is incorrect and all the other key-bit values are the same as K^*. In other words, while the HD of K^a is very low (i.e., $HD(K^a, K^*) = 1$), the locked circuit $\mathbb{G}$ under K^a outputs differently than the original circuit $\mathbb{F}$ in 100% of the input patterns (i.e., $ER(K^a) = 1$).

Hence, Proposition 1 is incorrect since it is possible to design a locking method in which incorrect keys with a small hamming distance from the correct key have high output corruptibility. We can conclude that a key of the locked circuit with a small hamming distance to the correct key does not necessarily outperform a random key with a large hamming distance. To address this issue, we believe that incorporating ER in the training dataset can be useful.

Although, graph representation preserves the topology of the circuit, using an undirected graph for netlist representation is one of the shortcomings of GNNs because the inputs/outputs neighborhood of the netlist will be indistinguishable. Traditional GNN-based attacks cannot effectively distinguish the difference between XOR and XNOR key gates due to not taking the circuit functionality into account.

Proposition 2: GNN-based attacks can report an approximate key K^a of the locked circuit $\mathbb{G}$ in which $HD(K^a, K^*)$ is very small.

Counterexample 2: We consider OMLA [24] as one of the GNN-based attacks, in which its prediction accuracy has been shown to be on average 80%. It means, for a reported key K^a, it is expected to predict almost 80% of the key-bits correctly (i.e., $HD(K^a, K^*) = 0.2p$). If we replace all the XOR gates with XNOR in the benchmarks with XOR-based locking [1] and push the inverters to the fanouts using bubble pushing, the new correct key will be the complement of the previous one. However, the attack prediction accuracy drops significantly to an average of 56% (i.e., $HD(K^a, K^*) = 0.44p$) which is not much better than reporting a random key. Fig. 2 depicts an example of such a transformation on one key-bit.

The above counterexample shows that Proposition 2 is incorrect and that GNN models are highly dependent on specific gates in the circuit but not on their functional dependencies with each other, and the model prediction accuracy can drop significantly by inverting the key-bits and bubble pushing.

III. LIPSTICK Attack

In this section, we propose **LIPSTICK**, a corruptibi**LI**ty-aware and ex**P**lainable GNN-based oracle-les**S** a**T**tack on log**I**c lo**CK**ing shown in Fig. 3.

1 We use seven of the ISCAS'85 [33] benchmarks shown in Table I and lock each of them with seven logic locking methods, including XOR-based locking [1], MUX-based locking [2], LUT-based locking [3], SAR-Lock [4], Anti-SAT [5], BLE [6], and UNSAIL [7] all with a 64-bit key size.

Then we convert .BENCH files of both the original and locked benchmarks into .V using ABC tool [34], and use ModelSim to simulate and extract the ER of 10 random wrong keys ranging from 0 to 1 in addition to the correct key. Finally,

TABLE I: ISCAS '85 benchmarks [33] information

Bench.	Gates	Functionality
c1355	1503	32-bit single-error corrector
c1908	1289	16-bit single-error corrector and double-error detector
c2670	1262	12-bit arithmetic logic unit and controller
c3540	1403	8-bit arithmetic logic unit
c5315	1350	9-bit arithmetic logic unit
c6288	4703	16x16 multiplier
c7552	1241	32-bit adder and comparator

we apply bubble-pushing to create 10 resynthesized versions of each benchmark.

Overall, our dataset consists of 5,390 elements of data with multiple labels, including a label for the locking method, a label for the designated key for each benchmark (be it correct or wrong), and a label for the ER of that key.

❷ We can define GNN as an undirected graph $G = (\mathcal{V}, \mathcal{E}, X, A)$ where $\mathcal{V}$ represents the vertex set, $\mathcal{E}$ is the edge set, X constitutes the node feature matrix, and A indicates the adjacency matrix of the graph. GNN may learn the embedding of a single node or the complete graph by using the graph's structure and node attributes. In order to compute the intended results, the GNN model adopts neighborhood aggregation by iteratively updating a node's embedding depending on the embeddings of its neighbors. The following equations show the GNN's i-th layer, where ${h_v}^{(i)}$ represents the embedding of node v at the i-th layer, and and $\mathcal{N}(v)$ shows a set of nodes adjacent to v. We use the same method as the Graph Isomorphism Network (GIN) architecture [29] to initialize the parameters and consider ${h_v}^{(0)} = X_v$.

$$ {a_v}^{(i)} = AGGREGATE^{(i)}({h_u}^{(i-1)} : u \in \mathcal{N}(v)) \quad (2) $$

$$ {h_v}^{(i)} = COMBINE^{(i)}({h_v}^{(i-1)}, {a_v}^{(i)}) \quad (3) $$

The choice of the GNN method defines which *aggregate* and *combine* functions to use. In this work, the target is graph classification, so for the aggregation function, we use the *readout* function described as follows:

$$ h_G = READOUT(\{{h_v}^{(I)} | v \in G\}) \quad (4) $$

Where h_G is the representation of the entire graph, and the *readout* function acquires the entire graph representation by aggregating node features from the final iteration.

We use the netlist-to-subgraph tool available in [24] to extract graphs and subgraphs from .V files of the dataset. The main focus of the training phase is to increase the model's prediction accuracy as well as key precision so that in the validation phase, it predicts a more accurate key with low ER. A well-trained model will also provide more meaningful information when fed into a graph explainer tool. A model learns graph features by incorporating a hyperparameter called learning rate, whose exact value is tricky to determine. One naive way is to set it to a constant value, which could either drastically increase the model's training time if the value is too small or stop the model from learning valuable features if it is too large. To provide a reasonable trade-off, we define the learning rate so that after 100 epochs, it gets its 0.01 value for the next 100 epochs. Moreover, we utilize Leaky ReLU as the GNN's activation function to keep the value of x using the maximum function $f(x) = max(0.01x, x)$. Finally, we employ an early stopping strategy to cease training the model if, after five consecutive iterations, the model did not achieve greater accuracy than prior iterations or if the loss value increased to 1 in order to prevent overtraining the model. At this point, if the model accuracy is still insufficient, we change the sliding window size, pooling window size, and number of layers in the model.

TABLE II: OMLA's [24] prediction accuracy and reported key precision under different feature maps

Prediction Accuracy	Key Precision	Epoch	Feature Map Description
80.78 %	59.75%	350	Default
80.63 %	61.33%	350	Random Assignment
77.63 %	62.29%	350	Highest Assign. to Lowest #Gates

❸ In the post-training phase, we validate the model's prediction accuracy and reported key precision using seen and unseen locked benchmarks. After making sure the model's prediction accuracy and reported key precision are acceptable, we feed the trained model to PGExplainer [32]. While feature explanation in GNNs is comparable to that in non-graph neural networks, PGExplainer concentrates on explaining graph structures. In order to offer explanations for numerous occurrences, PGExplainer makes use of a parametric explanation network built on a graph-generative model to provide topological explanations.

IV. Experimental Results

We implemented *LIPSTICK* on an Intel Core i7-10750H CPU, with a RAM size of 16 GB.

A. Attack Results

To compare model prediction accuracy and reported key precision (i.e., (1-ER)×100) in state-of-the-art works, we chose OMLA [24], and included different features in the circuit comprehension shown in Table II. It is evident that in OMLA, the model's prediction accuracy does not correlate with the reported key precision, and a model's accuracy of 80% does not assure high key precision. This is because state-of-the-art GNN models focus solely on the structures of the circuits, not their functionality. Another interesting observation here is that OMLA's model prediction accuracy stayed roughly the same when using random feature map assignment compared with the default case, which indicates that the model does not distinguish the gates in its inference.

Table III shows *LIPSTICK*'s prediction accuracy and reported key precision using various groups of locking schemes and benchmarks. Unlike column "5 Random" which only contains locking schemes that were used in the training set meaning XOR-based locking [1], MUX-based locking [2], LUT-based locking [3], and SAR-Lock [4], the "10 Random" and

TABLE III: LIPSTICK's prediction accuracy and reported key precision under random seen and unseen benchmarks. Abreviation guide: X=XOR-based locking [1], M=MUX-based locking [2], L=LUT-based locking, [3], S=SAR-Lock [4], B=BLE [6]. The word "Random" refers to random samples of the validation dataset which includes locking schemes from the training dataset as well as two unseen locking methods: Anti-SAT [5] and UNSAIL [7]. The values in "Random" columns are the average of key precision for the reported keys.

Locking Scheme	Prediction Accuracy	5 Random Key Prec.	10 Random Key Prec.	50 Random Key Prec.
X	92.64%	79.84%	75.57%	74.97%
M	93.11%	79.41%	75.44%	75.66%
L	92.75%	78.57%	75.68%	75.54%
S	93.43%	79.19%	76.21%	75.94%
X,M,L	85.50%	74.86%	70.63%	70.75%
X,L,S	84.16%	74.33%	70.58%	70.06%
X,M,S	82.22%	75.78%	69.16%	68.65%
M,L,S	84.87%	75.44%	70.33%	69.28%
X,M,L,S	76.95%	69.19%	65.39%	67.03%
X,M,L,S,B	51.23%	50.63%	49.97%	50.27%

"50 Random" columns contain unseen locking methods such as Anti-SAT [5] and UNSAIL [7] as well.

Each of the locking methods incorporates a unique algorithm to secure the circuit. Hence, because the structures of each of the circuits are different, each locking method offers different patterns to learn. For this reason, we train the GNN with single locking schemes as well as mixing the locking methods in the dataset. We did not include BLE [6] in the single-lock-scheme training because it incorporates the same ER for all the wrong keys, and hence does not let the GNN model learn meaningful information.

The first four rows are the results of single-lock-scheme training, whose prediction accuracy is above 92% and the average key precision is above 75%. Comparing Tables II and III, not only *LIPSTICK* outperforms OMLA in terms of prediction accuracy (i.e., finding a key with low hamming distance to the correct key) but also significantly improves the key precision (i.e., finding a key with low output corruptibility). By including unseen locking methods (i.e., "10 Random" and "50 Random" columns), key precision accuracy drops a little but is still reasonable due to the fact that these sets include data that GNN was not trained on and did not learn their features.

The results in the final two rows show that, despite the fact that our objective is to provide a universal model that can perform well on various locking methods and different circuit structures, a more diverse dataset (i.e., more than three logic locking methods) does not always result in high prediction accuracy or high key precision. Another reason is that including BLE [6] in the training dataset may mislead the model since in BLE any approximate key has high ER and thus low key precision.

B. Explainability Results

The prediction results of PGExplainer are shown in Fig. 4. In each sub-figure, a sample pattern of the explainable graph prediction is demonstrated. Colored nodes represent different features, and black edges with the corresponding nodes illustrate the patterns that PGExplainer was able to find. Fig. 4a and 4b are the explanation accuracies for the trained graphs with resynthesized benchmarks locked with XOR-based locking and MUX-based locking, respectively. By using the default setup of the PGExplainer we achieved 72% explanation accuracy. While our dataset includes resynthetized versions of the same function, which means that there are multiple structures whose ER are the same under a specific key, PGExplainer is based on one structure per label. In other words, by including ER in the training process, our model goes beyond the structure analysis of graphs, which puts emphasis on the need for the development of an appropriate GNN explanation tool that also goes beyond the structure. Besides that, by providing details on manifold connections and different patterns, PGExplainer gives us information on how to assign features to the input graphs before feeding them to the GNN model. This information is beneficial in allocating rational features to the elements of the input graphs.

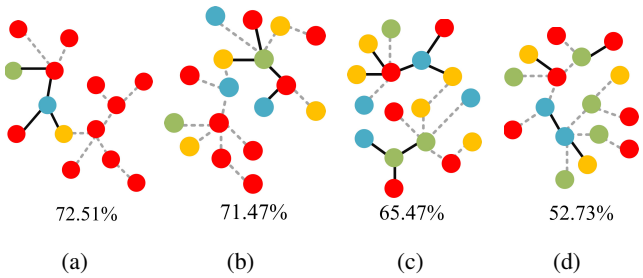

Fig. 4: Explanation accuracy of LIPSTICK on locking schemes. Bold edges demonstrate a sample pattern that PGExplainer [32] was able to find. (a) XOR-based locking (b) MUX-based locking (c) XOR-based and MUX-based methods (d) XOR-based, MUX-based, LUT-based, and SAR-Lock methods

Moreover, the explanation accuracy drops in 4c which uses a trained graph as input that was trained with two different logic locking methods. The reason for this accuracy drop is that the PGExplainer should focus on finding more explainable features that do not have common characteristics. Finally, Fig. 4d shows the explanation accuracy result for the trained graph with four locking methods in which the explainer is acting as a random predictor and cannot grasp enough information to provide plausible explanations. It is, however, justifiable by taking another look at the characteristics that each of the locking schemes provides to a circuit. Since the features are diverse, in the training phase, the GNN model did not have enough layers to learn all the various features offered by different locking methods.

V. Conclusion

In this work, we proposed *LIPSTICK*, a corruptibility-aware and explainable GNN-based OL attack on different logic locking methods. *LIPSTICK* incorporates circuit functionality labels in addition to structural parameters into the GNN model

with the goal of guiding the model into reporting a more relevant key. In addition, it includes different resynthesized versions of the same circuit, so the model can learn features from different structural views. Furthermore, it involves different logic-locked circuits with both correct and wrong key labels to let the model learn from wrong key insertion too. The experimental results depicted that *LIPSTICK* can achieve both higher model prediction accuracy and higher reported key precision compared to state-of-the-art GNN-based attacks. Moreover, by feeding the trained graphs to a graph explainer tool, we can receive information on how the GNN model is working on the dataset, what the important patterns are, and which components are more important when assigning features to the dataset.

ACKNOWLEDGMENT

This work is supported by the National Science Foundation under Award No. 2245247.

REFERENCES

[1] J. A. Roy, F. Koushanfar, and I. L. Markov, "Epic: Ending piracy of integrated circuits," In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1069-1074, 2008.

[2] J. Rajendran, H. Zhang, C. Zhang, G. S. Rose, Y. Pino, O. Sinanoglu, and R. Karri, "Fault analysis-based logic encryption," In *IEEE Transactions on computers*, pp. 410-424, 2013.

[3] A. Baumgarten, A. Tyagi, and J. Zambreno, "Preventing IC piracy using reconfigurable logic barriers," In *IEEE design & Test of computers*, pp. 66-75, 2010.

[4] M. Yasin, B. Mazumdar, J. Rajendran, and O. Sinanoglu, "SARLock: SAT attack resistant logic locking," In *International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 236-241, 2016.

[5] Y. Xie and A. Srivastava, "Anti-SAT: Mitigating SAT attack on logic locking," In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 2, pp. 199-207, 2019.

[6] A. Rezaei, Y. Shen, and H. Zhou, "Rescuing logic encryption in post-SAT era by locking & obfuscation," In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 13-18, 2020.

[7] L. Alrahis, S. Patnaik, J. Knechtel, H. Saleh, B. Mohammad, M. Al-Qutayri, and O. Sinanoglu, "UNSAIL: Thwarting oracle-less machine learning attacks on logic locking," In *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 2508-2523, 2021.

[8] A. Rezaei, Y. Shen, S. Kong, J. Gu, and H. Zhou, "Cyclic locking and memristor-based obfuscation against CycSAT and inside foundry attacks," In *Design, Automation & Test in Europe Conference & Exhibition (DATE),* pp. 85-90, 2018.

[9] R. Afsharmazayejani, H. Sayadi, and A. Rezaei, "Distributed logic encryption: Essential security requirements and low-overhead implementation," In *Proceedings of Great Lakes Symposium on VLSI (GLSVLSI)*, pp. 127-131, 2022.

[10] J. Maynard and A. Rezaei, "DK lock: Dual key logic locking against oracle-guided attacks," In *International Symposium on Quality Electronic Design (ISQED)*, pp. 1-7, 2023.

[11] A. Rezaei, A. Hedayatipour, H. Sayadi, M. Aliasgari, and H. Zhou, "Global attack and remedy on IC-specific logic encryption," In *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 145-148, 2022.

[12] A. Rezaei, J. Gu, and H. Zhou, "Hybrid memristor-CMOS obfuscation against untrusted foundries," In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 535-540, 2019.

[13] A. Rezaei and H. Zhou, "Sequential logic encryption against model checking attack," In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1178-1181, 2021.

[14] Y. Aghamohammadi and A. Rezaei, "CoLA: Convolutional neural network model for secure low overhead logic locking assignment," In *Great Lakes Symposium on VLSI 2023 (GLSVLSI)*, pp. 339–344, 2023.

[15] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic locking algorithms," In *International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 137-143, 2015.

[16] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "AppSAT: Approximately deobfuscating integrated circuits," In *International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 95-100, 2017.

[17] Y. Shen, Y. Li, S. Kong, A. Rezaei, and H. Zhou, "SigAttack: New high-level SAT-based attack on logic encryptions," In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 940-943, 2019.

[18] M. Zuzak, Y. Liu, I. McDaniel, and A. Srivastava, "A combined logical and physical attack on logic obfuscation," In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Article 68, pp. 1-9, 2022.

[19] A. Rezaei, R. Afsharmazayejani, and J. Maynard, "Evaluating the security of eFPGA-based redaction algorithms," In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, article 154, pp 1-7, 2022.

[20] P. -P. Chen, X. -M. Yang, Y. -T. Li, Y. -C. Chen, and C. -Y. Wang, "An approach to unlocking cyclic logic locking: LOOPLock 2.0," In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Article 155, 1–7, 2022.

[21] 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," In *ACM Journal on Emerging Technologies in Computing Systems* vol. 17, no. 3, article 30, 2021.

[22] P. Chakraborty, J. Cruz, and S. Bhunia, "SAIL: Machine learning guided structural analysis attack on hardware obfuscation," In *Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pp. 56-61, 2018.

[23] K. Shamsi and G. Zhao, "An oracle-less machine-learning attack against lookup-table-based logic locking," In *Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI)*, pp. 133-137, 2022.

[24] L. Alrahis, S. Patnaik, M. Shafique, and O. Sinanoglu, "OMLA: An oracle-less machine learning-based attack on logic locking," In *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 3, pp. 1602-1606, 2021.

[25] L. Alrahis, S. Patnaik, M. A. Hanif, H. Saleh, M. Shafique, and O. Sinanoglu, "GNNUnlock+: A systematic methodology for designing graph neural networks-based oracle-less unlocking schemes for provably secure logic locking," In *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 3, pp. 1575-1592, 2021.

[26] 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 *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1-9, 2021.

[27] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang and P. S. Yu, "A comprehensive survey on graph neural networks," In *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4-24, 2021.

[28] H. Yuan, H. Yu, S. Gui, and S. Ji, "Explainability in graph neural networks: A taxonomic survey," In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 5, pp. 5782-5799, 2023.

[29] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," In *International Conference on Learning Representations (ICLR)*, 2019.

[30] F. K. Dosilovic, M. Brcic, and N. Hlupic, "Explainable artificial intelligence: A survey," In *International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pp. 0210-0215, 2018.

[31] Y. Shen, H. Li, S. Yi, D. Chen, and X. Wang, "Person re-identification with deep similarity-guided graph neural network," In *European Conference on Computer Vision (ECCV)*, pp. 486-504, 2018.

[32] D. Luo, W. Cheng, D. Xu, W. Yu, B. Zong, H. Chen, and X. Zhang, "Parameterized explainer for graph neural network," In *International Conference on Neural Information Processing Systems (NIPS)*, pp. 19620-19631, 2020.

[33] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran," In *IEEE International Symposium on Circuits and Systems (ISCAS),* pp. 677-692, 1985.

[34] Berkeley Logic Synthesis and Verification Group, "ABC: A system for sequential synthesis and verification," https://people.eecs.berkeley.edu/∼alanmi/.