GNNUnlock: Graph Neural Networks-based Oracle-less Unlocking Scheme for Provably Secure Logic Locking

Lilas Alrahis[ϒ], Satwik Patnaik[†], Faiq Khalid[§], Muhammad Abdullah Hanif[§], Hani Saleh[ϒ], Muhammad Shafique[‡], and Ozgur Sinanoglu[‡]

[ϒ]Department of Electrical Engineering and Computer Science, Khalifa University, Abu Dhabi, UAE
[†]Electrical & Computer Engineering, Texas A&M University, College Station, Texas, USA
[§]Institute of Computer Engineering, Technische Universität Wien, Vienna, Austria
[‡]Division of Engineering, New York University Abu Dhabi, UAE
lilas.alrahis@ku.ac.ae, satwik.patnaik@tamu.edu, {muhammad.shafique, ozgursin}@nyu.edu

***Abstract*—Logic locking is a holistic design-for-trust technique that aims to protect the design intellectual property (IP) from untrustworthy entities throughout the supply chain. Functional and structural analysis-based attacks successfully circumvent state-of-the-art, provably secure logic locking (PSLL) techniques. However, such attacks are not holistic and target specific implementations of PSLL. Automating the detection and subsequent removal of protection logic added by PSLL while accounting for *all* possible variations is an open research problem.**

In this paper, we propose GNNUnlock, the first-of-its-kind oracle-less machine learning-based attack on PSLL *that can identify any desired protection logic without focusing on a specific syntactic topology*. The key is to leverage a well-trained graph neural network (GNN) to identify all the gates in a given locked netlist that belong to the targeted protection logic, *without requiring an oracle*. This approach fits perfectly with the targeted problem since a circuit is a graph with an inherent structure and the protection logic is a sub-graph of nodes (gates) with specific and common characteristics. GNNs are powerful in capturing the nodes' neighborhood properties, facilitating the detection of the protection logic. To rectify any misclassifications induced by the GNN, we additionally propose a connectivity analysis-based post-processing algorithm to successfully remove the predicted protection logic, thereby retrieving the original design.

Our extensive experimental evaluation demonstrates that GNNUnlock is $99.24\% - 100\%$ successful in breaking various benchmarks locked using stripped-functionality logic locking [1], tenacious and traceless logic locking [2], and Anti-SAT [3]. Our proposed post-processing enhances the detection accuracy, reaching 100% for all of our tested locked benchmarks. Analysis of the results corroborates that GNNUnlock is powerful enough to break the considered schemes under different parameters, synthesis settings, and technology nodes. The evaluation further shows that GNNUnlock successfully breaks corner cases where even the most advanced state-of-the-art attacks [4], [5] fail. We also open source our attack framework [6].

***Index Terms*—Logic locking, IP protection, Graph Neural Networks, Machine Learning, Oracle-less attack.**

I. Introduction

Logic locking (LL) is a design-for-trust technique that aims to protect the design intellectual property throughout the supply chain. The design's functionality is locked with a secret key (driven from an on-chip tamper-proof memory) and a set of added key-gates. The locked design functions correctly only when the correct secret key is in place. Earlier works in LL focused on the placement of key-gates and high output corruptibility [7]–[9]. However, these schemes were deemed vulnerable by the Boolean satisfiability (SAT)-based attack [10].

Provably secure logic locking (PSLL) ensures an exponential number of SAT calls for a successful key recovery [1]–[3], [11], [12]. In principle, a *perturb* unit injects errors into the design for some protected input patterns. These errors are canceled out by a *restore* unit when the correct key is applied. Recently, structural/functional *oracle-less* attacks have emerged to circumvent PSLL [4], [5], [13], [14].[1] **Each attack aims to break a specific locking scheme, under restricted parameter settings and circuit formats**, resulting in a large set of attack strategies for different PSLL implementations. Developing a holistic attack on PSLL is an open research problem. Table I outlines the drawbacks of oracle-less attacks on PSLL.

TABLE I. Capabilities offered by oracle-less attacks

Attacks	Different Circuit Formats	Different Locking Schemes	Different Parameter Settings
SPS [13]	✗	✗	✓
RE-based [14]	✗	✗	✓
FALL [5]	✗	✗	✗
SFLL-HD-Unlocked [4]	✗	✗	✗
GNNUnlock	✓	✓	✓

A. Motivation and Research Challenges

Restricted Parameter Settings: Stripped functionality logic locking (*SFLL-HD^h*) [1] showed resilience against all known attacks on LL. SFLL-HD^h protects all input patterns from a Hamming distance h from the secret key K. Sirone *et al.* [5] developed functional analysis attacks (FALL) on SFLL-HD^h which retrieved the secret key *without* requiring an oracle. The attack algorithms are based on a set of derived Lemmas defining functional properties of the protection logic used in SFLL-HD^h. The derived Lemmas hold for specific ranges of h values. Hence, by definition, the attack algorithms cannot break all cases of SFLL-HD^h. For example, their *Analyze-Unateness* algorithm is only applicable when $h = 0$, and their *Hamming2D* algorithm is only applicable when $h \leq K/4$. Their third algorithm *SlidingWindow* should apply to larger h values in principle, but it does not fare well since it requires computationally hard SAT calls. Yang *et al.* [4] proposed the *SFLL-HD-Unlocked* attack that performs connectivity analysis on the locked circuit followed by a Gaussian-elimination-based analysis to recover the secret key *without* requiring an oracle. Because of the use of Gaussian-elimination, the attack does not work when $h \leq 4$ due to the composition of singular matrices.

[1] *Oracle-guided* attacks are successful in approximately circumventing specific instances of PSLL [15]–[17]. However, these attacks require a working chip for functional verification. We focus on the *oracle-less* attacks as they pose a more significant and realistic threat, as highlighted by the LL community.

To demonstrate the shortcomings of existing attacks, we lock selected ISCAS-85 and ITC-99 benchmarks using SFLL-HD^h with $K/h = 2$, multiple times, generating 192 locked designs. *Such a locking ratio is critical as it achieves the highest resilience to removal attacks, as indicated in [1].* When the FALL attacks [5] were launched on these locked benchmarks; they reported 0 keys, failing to break any of the designs. Similarly, SFLL-HD-Unlocked attack [4] also was unable to recover the key for any design. *This analysis highlights the need for a holistic approach that can break SFLL-HD^h and other PSLL schemes under all the possible scenarios.*

Restricted Circuit Formats: The prior attacks on PSLL accept restricted and non-standard circuit formats. For example, the work in [14] showcased that functional reverse engineering (RE) aids in the detection and subsequent removal of the protection logic added by SFLL-HD^h. However, the approach is not generic and will only work on designs synthesized using only 2-input gates. This attack achieves low accuracy results when launched on benchmarks synthesized with a standard cell library. For example, the *restore* unit detection percentage in the ISCAS-85 benchmark c7552 locked with $K = 32$ and $h = 0$ drops from 100% to 0%. Moreover, both attacks [4], [5] accept circuits in bench format (the latter [5] requires topologically sorted AND-OR-INVERT gates), which is a non-industry format, as opposed to *Verilog/VHDL*. Thus, these techniques cannot be employed in the real-world design flow.

Locking Scheme-specific Attacks: All the aforementioned oracle-less attacks are locking scheme-specific. They target a distinct protection implementation (SFLL-HD^h in this case) and are not scalable to other variants (such as *Anti-SAT* [3]). The signal probability skew (SPS) attack [13] is another oracle-less attack targeting Anti-SAT. The SPS attack looks for two oppositely skewed nets feeding an AND gate, a property specific to the Anti-SAT protection and not to other PSLL schemes, and hence it is another scheme-specific attack.

Associated Research Challenges: Automating the detection of the protection logic added by PSLL while accounting for all possible variations among the implementations is an open research problem posing the following important challenges.

1) *Recognizing all implementation variants:* The protection logic structure depends on (i) the key-value, (ii) the Hamming distance value (for the case of SFLL-HD), (iii) the choice of the logic blocks (for the case of Anti-SAT), and (iv) the key-size. Hence, different settings will result in distinct topologies. Thus, utilizing an exact matching algorithm would be a naive approach.
2) *Handling different synthesis settings and circuit formats:* The structure of the protection logic and its integration with the original design depends on the synthesis procedures and the target technology library. The attack model should be able to understand and process different circuit formats without having to modify the attack tactic.

B. Our Novel Concept and Contributions

To address the above challenges, we propose the ***GNNUnlock*** **framework** that identifies all variants of the targeted protection logic. It is the first-ever concept to leverage graph neural networks (GNNs) along with node connectivity analysis to identify all the gates in a given locked netlist that form the protection logic, as depicted in Fig. 1. This work

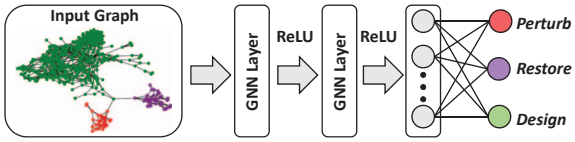

Fig. 1. High-level view of our work: Breaking PSLL using GNN.

is inspired by the ability of advanced machine learning (ML) models to adapt to new unseen data and the fact that a circuit is a graph with an inherent structure to it. Unlike other ML models, GNNs leverage the graph structure and node (gate) features to learn a representation for each node capturing its neighborhood characteristics.[2] GNNs are a fitting choice for identifying the protection logic, which can be conceived as a sub-graph with specific properties. The GNN learns the trend of the protection logic and not a syntactic implementation of it, and hence GNNUnlock deals with variations naturally. Our GNNUnlock framework employs the following techniques:

1) A framework for the **netlist-to-graph transformation (Section IV-B)** is developed which implements automatic & efficient feature extraction to capture each gate's functionality and connectivity in the gate-level netlist (in any format).
2) **GNN learning on locked circuits is achieved (Section IV-C)**, allowing the GNN to identify the structural features of all the nodes in the targeted protection logic.
3) **Post-processing rectification procedure (Section IV-D)** guided by the connectivity of the circuit and by the predictions of the GNN (on the under-attack circuit) is developed to remove the identified protection logic effectively.

The effectiveness of GNNUnlock is showcased by breaking 564 benchmarks locked using three different PSLL techniques – SFLL-HD [1], *TTLock* [2], and Anti-SAT [3]. The effect of different K, different h values (for the case of SFLL-HD, and other technology libraries are considered when evaluating GNNUnlock comprehensively. All the files locked at the register-transfer level (RTL) are synthesized following the commercial ASIC-design flow using Synopsys tools. **We also make our attack framework available online at [6].**

II. Background and Related Works

A. Provably Secure Logic Locking (PSLL)

1) Anti-SAT [3]: In the Anti-SAT block, the outputs of two complementary logic functions ($gl1$ and $\overline{gl2}$), locked with two sets of keys are fed into an AND gate, as shown in Fig. 2a. When the correct key is in place the output of the AND gate Y is 0. For a wrong key-input, the output signal Y could be 0 or 1 depending on the input X. Y is XORed with an internal net in the original netlist, locking it. Having the Y signal highly skewed to 0 ensures resilience against the SAT-based attack. Nevertheless, Anti-SAT is vulnerable to structural analysis-based removal attacks [13].

2) TTLock [2] and SFLL-HD [1]: modify the original design, as shown in Fig. 2b, to achieve resilience against removal attacks. TTLock protects an input pattern that is the same as the locking key. Tracing the key-inputs (KIs) helps identify the *restore* logic of TTLock, which is a basic comparator. However, the *perturb* unit structure depends on the selected secret key, and not controlled by the external

[2]We use the terms interchangeably gate or node, and circuit or graph.

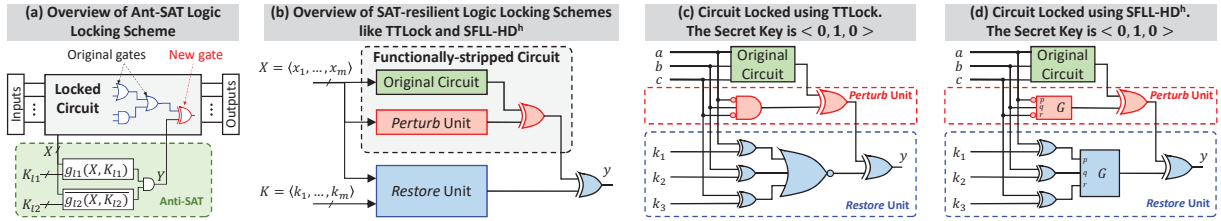

Fig. 2. Logic locking using Anti-SAT [3], TTLock [2], and SFLL-HD [1]. G block in (d) performs Hamming distance checking using adders and comparators.

KIs, as demonstrated in Fig. 2c. SFLL-HD^h protects all input patterns that are an h Hamming distance away from the secret key. TTLock is equivalent to SFLL-HD^0. For SFLL-HD$^{h>0}$, the *restore* and *perturb* units are Hamming distance checkers (G), as illustrated in Fig. 2d. The structure of the *perturb* unit depends on the hard-coded key, making it difficult to be traced.

B. Graph Neural Network (GNN)

In GNNs, nodes aggregate information from their neighbors using neural networks. Each node gathers information from its neighbors in a previous layer, where each one of these neighbors collects information from the previous layer. The aggregation functions in *GraphSAGE* [18] approach learn to aggregate feature information to obtain embeddings for the nodes, which are then used to perform the desired task such as node classification. The same aggregation parameters are shared for all nodes. Hence, once the model is trained, those parameters can generate embeddings on entirely unseen graphs. We follow such an inductive approach since we train on a set of locked benchmarks and *test the model on unseen benchmarks locked with unknown key-values*. We adopt the GraphSAGE architecture in which each node's previous state is concatenated with its neighbors' current state. The mean aggregation function is used to update the state of each node.

Due to neighborhood aggregation, the deeper the GNN model is, the more multi-hop neighbors get incorporated for the root node's computation. Hence, training time could grow exponentially with GNN depth [19]. Several methods, including GraphSAGE, perform layer sampling to ensure a limited set of neighbors is considered for a node in the next layer. Such sampling speeds up training but suffers from scalability issues. We leverage the graph sampling method *GraphSAINT* [19] to construct mini-batches for training. Rather than building a GNN on full training graph and then perform layer sampling, GraphSAINT samples the training graph and then builds a GNN on the sampled graph for each mini-batch,[3] ensuring scalability concerning graph size and GNN depth. Their random walk-based sampler is used in our experiments.

III. Attacker Model

We assume that the attacker will be present in the untrusted foundry with access to the GDSII mask information and reverse engineering tools that facilitate the gate-level netlist extraction. Besides, the attacker is aware of the locking algorithm and can distinguish between the regular primary inputs (PIs) and KIs. The attacker also knows the usage of the technology library and the usage of different synthesis settings. We do not assume the attacker to have access to an unlocked chip (i.e., an *oracle-less* setting). For SFLL-HD^h [1], the attacker knows the Hamming distance value, while the type of logic function (gl) is known for the case of Anti-SAT [3]. Note, all these assumptions are in line with the *Kerckhoffs's principle*, which states that everything about the system should be known to an attacker *except* for the value of the secret key.

[3]In each mini-batch, forward and backward propagation is performed iteratively on the sampled GNN to update weights via stochastic gradient descent.

IV. Proposed GNNUnlock Framework

Fig. 3 shows an overview of our methodology, with key steps discussed in the following subsections.

A. Dataset Generation

Using the underlying locking scheme, we generate a set of locked benchmarks for training and validation. Each benchmark is locked several times with randomly generated key-values and different key-sizes K, which results in an extensive training set. The training process is not aware of the correct key-values. However, different key-values cause the generation of varying *perturb* and *Anti-SAT* structures, and hence variants are accounted for during training. GNNUnlock transforms locked benchmarks into graphs (Section IV-B) and labels the nodes. For the case of SFLL-HD$^{h>0}$/TTLock, a node belongs to the *perturb*, the *restore*, or the *design* logic. For the case of Anti-SAT, a node belongs to the *Anti-SAT* block or the *design* logic.

GNNUnlock attacks each design independently by excluding its corresponding graphs from training/validation. For example, when attacking `b17_C` from ITC-99, only the graphs of `b14_C`, `b15_C`, `b20_C`, and `b21_C` are used in training, while the graphs of `b22_C` are used for validation to evaluate the model on unseen data, thereby avoiding biasing.

B. Netlist-to-Graph Transformation

We model connectivity between the gates in a design using an undirected graph $G(I, J)$. The set I of nodes represents all the gates, while the set J of edges represents the wires. An example of a logic locked netlist and the corresponding graph is shown in Fig. 3b. The set I does not contain PIs, KIs, or primary outputs (POs) of the design. Each node in the graph is associated with a feature vector $\hat{f}$ that contains information describing the node's important characteristics, such as its in-degree (IN) and out-degree (OUT). It also contains information on whether the node is connected to a PI, a PO, or a KI. Additionally, $\hat{f}$ captures the type and number of gates appearing in the neighborhood of the node (all nodes within two-hops away). For example, $\hat{f}_i$ shows that gate i is connected to a PO with $IN = 3$ and $OUT = 1$ (Fig. 3b). It also indicates that three XOR gates and one XNOR gate exist in i's neighborhood. The length of the feature vector $|\hat{f}|$ depends on the number of logic gates in the target library. Each locked design is represented by a graph instance with a corresponding adjacency matrix. To feed multiple graphs of different sizes to

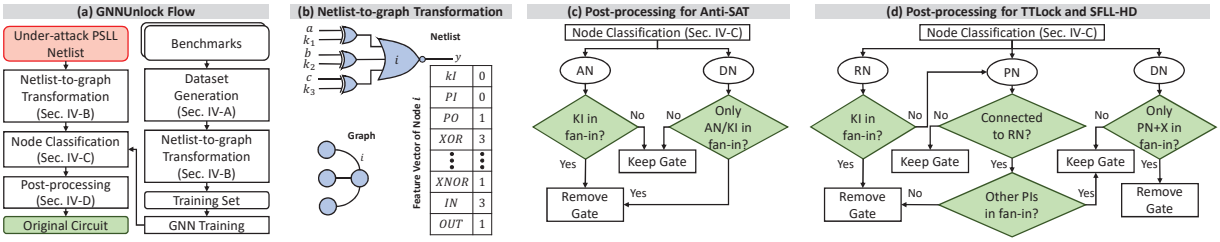

Fig. 3. Proposed GNNUnlock methodology. The *Anti-SAT*, *restore*, *perturb*, and *design* nodes are indicated by AN, RN, PN, and DN, respectively.

the GNN, a block-diagonal matrix is created for each dataset. Each block represents the adjacency of one locked design.

C. GNN Topology

In all experiments reported in Section V, graph sampling using the GraphSAINT method is performed. A two-layer GNN is constructed using GraphSAGE architecture, with mean and concatenation aggregation, and the *ReLU* activation function. Training stops after a fixed number of epochs based on convergence. The model with the best performance on the validation set is used to evaluate the test set accuracy. Details regarding the GNN model and sampling are shown in Table II.

D. Post-processing

Although our trained GNN achieves high node classification accuracy on its own (with an average of 99.99%, 99.95%, and 99.92%, for the case of Anti-SAT, TTLock, and SFLL-HD^2, respectively), we propose a post-processing algorithm to rectify any potential misclassifications to enhance the accuracy further. The algorithm considers predictions made by the GNN, connectivity of the circuit, and known properties of the protection logic, which is explained further in the next subsections.

1) Anti-SAT: Every node in the *Anti-SAT* block must have at least one KI in its fan-in cone. If a node without KIs in its fan-in cone is predicted as an *Anti-SAT* node, then the prediction will be dropped. For each *predicted design* node, we check its fan-in cone. If there are only other *predicted Anti-SAT* nodes in the fan-in cone, then the node initially classified as a *design* node is considered part of the Anti-SAT block. The flow of our post-processing algorithm is presented in Fig. 3c.

2) TTLock and SFLL-HD$^{h>0}$: Our post-processing algorithm for the case of SFLL-HD$^{h>0}$ and TTLock is presented in Fig. 3d. The *predicted restore* nodes are visited to identify possible protected inputs (set X). The following properties then guide the post-processing procedure. (i) All *restore* nodes have KIs in their fan-in cone. (ii) All *perturb* nodes have connections with the *restore* nodes in the netlist and are controlled solely by protected inputs X. If a *predicted restore* node has KIs in its fan-in cone, then the prediction is confirmed. Otherwise, the algorithm checks if the node is a *perturb* node. A *predicted perturb* node is verified if it has a connection to other *predicted restore* nodes in the netlist and if the node has X (and no other PIs) in its fan-in cone. In order not to misclassify *perturb* nodes as *design* nodes, the algorithm checks the fan-in cone of the predicted *design* nodes. If a predicted *design* node has X and other predicted *perturb* nodes in its fan-in, it is a *perturb* node.

V. Experimental Investigations

A. Evaluation Setup, Tool Flow, and Evaluation Metrics

GNNUnlock is evaluated on selected ISCAS-85 and ITC-99 benchmarks. Netlist-to-graph transformation is implemented in Perl/Python3. Tensorflow with Python3 implementation of GraphSAINT is used for GNN training [19]. Training is performed on a single node with 24 cores (2x Intel Xeon CPUs E5-2695 v2@2.4 GHz), 256GB RAM, and one NVIDIA Tesla K20m GPU (2,496 CUDA cores and 5GB of GDDR5 memory). Fig. 4 summarizes the experimental setup.

TABLE II. GNN configuration and sampling details. The #classes for SFLL-HD$^{h>0}$ and TTLock is 3, while for Anti-SAT it is 2

Architecture		Training and Sampling	
Input Layer	[$\|\hat{f}\|$,512]	**Optimizer**	Adam
Hidden Layer 1	[1024,512]	**Learning Rate**	0.01
Hidden Layer 2	[1024,512]	**Dropout**	0.1
Output Layer	[512,#classes]	**Sampler**	Random Walk
Aggregation	Mean with concatenation	**Walk Length**	2
Activation	ReLU	**Root Nodes**	3000
Classification	Softmax	**Max # Epochs**	2000

1) Dataset Generation for Anti-SAT: The benchmarks (in bench format) were locked using the Anti-SAT locking binary provided by the authors. Each ISCAS-85 benchmark is locked twice with $K : \{8, 16, 32, 64\}$, except for `c3540`, where $K = 64$ is not considered due to the limited number of PIs in the design. In total, `30` Anti-SAT locked ISCAS-85 benchmarks are obtained. $K = 8$ is used to evaluate the effectiveness of GNNUnlock in isolating the Anti-SAT block when its size is very small compared to that of the original design. Each ITC-99 benchmark is locked twice with $K : \{32, 64, 128\}$ resulting in a total of `36` Anti-SAT locked ITC-99 benchmarks. Two labeled datasets for Anti-SAT block identification have been created, one for locked ISCAS-85 benchmarks and one for locked ITC-99 benchmarks. A feature vector $|\hat{f}| = 13$ is associated with each node. *The Anti-SAT locking binary only accepts circuits in bench format, which includes a restricted set of logic gates (`8` gates)*. Hence, only `8` out of the `13` features are required to represent the neighborhood of each node. Rest of the features represent IN, OUT, and the connectivity to PIs, KIs, or POs.

2) Dataset Generation for TTLock and SFLL-HD$^{h>0}$: SFLL-HD^h is implemented in Perl as described in [1] to lock the benchmarks at RTL – TTLock is the case when $h = 0$. Each ISCAS-85 benchmark is locked thrice with $K : \{8, 16, 32, 64\}$ for $h : \{0, 2\}$, except for the `c3540` benchmark where $K = 64$ is not considered due to the limited number of PIs in the design. In total, `45` locked ISCAS-85 benchmarks for each h are obtained. Each ITC-99 benchmark is locked thrice with $K : \{32, 64, 128\}$ for $h : \{0, 2, 4\}$, resulting in a total of `54` locked ITC-99 benchmarks for each setting of h. Locked RTL files are synthesized using the standard ASIC design flow for the $65nm$ LPe technology. Synthesis is performed using Synopsys Design Compiler. $|\hat{f}| = 34$ is associated with each node. Since a full standard cell library is used for synthesis, a larger $|\hat{f}|$ (than

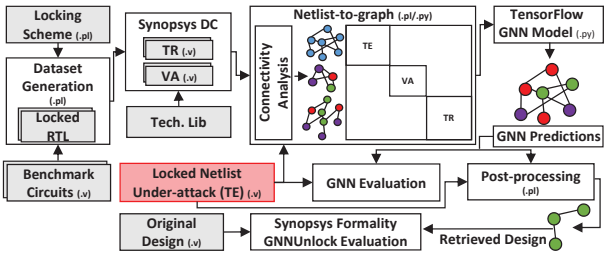

Fig. 4. Evaluation setup showing different components and platforms. TE, TR, and VA stand for testing, training, and validation, respectively.

the case of Anti-SAT) is required to capture all the possible combinational logic gates.

To verify that GNNUnlock handles different circuit formats and technologies, the Nangate $45nm$ open cell library is also used for synthesis when $h = 2$. Varying the circuits' format affects $|\hat{f}|$ only. For this case, $|\hat{f}| = 18$ is used. We also consider other corner cases for when $h : \{16, 32, 64\}$ with corresponding $K : \{32, 64, 128\}$, more details are in Section V-D. The characteristics of all the datasets are listed in Table III.

3) Evaluation Methods and Metrics: GNNUnlock attacks each design independently, by excluding its corresponding graphs from training and validation. For example, when attacking the TTLock-ed `b17_C` design from ITC, only the graphs of `b14_C`, `b15_C`, `b20_C`, and `b21_C` are included in the training set, resulting in `161,942` training nodes (`36` graphs), while the graphs of `b22_C` are included only in the validation set (to evaluate the model on unseen data), resulting in `74,753` validation nodes (`9` graphs). And only the graphs of `b17_C` are tested on, resulting in `110,685` testing nodes (`9` graphs).

The predictions of the GNN are compared with the true labels of the nodes to evaluate the performance of the attack. We report the accuracy and the non-averaged precision, recall, and F1-score for each classifier. We further evaluate GNNUnlock by removing the predicted protection logic to retrieve the unlocked design without accessing the true labels (Section IV-D). The recovered benchmark is then compared with the original benchmark via circuit-equivalence using Synopsys Formality.

B. Breaking Anti-SAT with GNNUnlock

For the ISCAS-85 dataset, the GNN gave 100% node classification accuracy for `25` of the tested graphs, and an average of 99.98% accuracy across all `30` graphs. For the remaining `5` graphs, only `6` nodes, were misclassified and then rectified during post-processing. For example, two *design* nodes feeding the XOR gate connecting the Anti-SAT block with the rest of the circuitry got misclassified as part of the Anti-SAT block. However, having no KIs in their fan-in, the prediction is ignored. For the ITC-99 dataset, the GNN gave 100% node classification accuracy for `29` out of `30` tested graphs. Only one node was misclassified in one of the `b21_C` graphs locked with $K = 128$, which was rectified during post-processing. Detailed results are reported in Table IV.

Having 100% node classification accuracy for a given benchmark locked with different key-sizes (such as the case for `c7552`, `b14_C`, etc.) shows that **the ratio of sizes between Anti-SAT block and design does not affect the performance of GNNUnlock and overall recovery of the original design**. Obtaining 100% node classification accuracy for Anti-SAT is validated since the Anti-SAT block has a specific structure controlled by external KIs, making it easy to be learned.

TABLE III. SUMMARY OF GENERATED DATASETS

Dataset	Benchmarks	Circuit Format	#Classes	$\lvert\hat{f}\rvert$	#Nodes	#Circuits
Anti-SAT	ISCAS-85	Bench	2	13	69,468	30
	ITC-99	Bench	2	13	450,359	36
TTLock	ISCAS-85	Verilog netlist $65nm$	3	34	29,745	45
	ITC-99	Verilog netlist $65nm$	3	34	347,380	54
SFLL-HD^2	ISCAS-85	Verilog netlist $65nm$	3	34	30,178	45
	ITC-99	Verilog netlist $65nm$	3	34	357,374	54
		Verilog netlist $45nm$	3	18	391,411	54
SFLL-HD^4	ITC-99	Verilog netlist $65nm$	3	34	356,420	54
SFLL-HD^{16}	ISCAS-85	Verilog netlist $65nm$	3	34	33,354	48
SFLL-HD^{32}	ITC-99	Verilog netlist $65nm$	3	34	465,032	72
SFLL-HD^{64}		Verilog netlist $65nm$	3	34	483,777	72

TABLE IV. RESULTS OF GNNUNLOCK ON ANTI-SAT. A *design* NODE AND AN *Anti-SAT* NODE ARE INDICATED BY DN AND AN, RESPECTIVELY. MN STANDS FOR MISCLASSIFIED NODES

Test	#Test Graphs	GNN Acc. (%)	Prec. (%)		Rec. (%)		F1-Score (%)		#MN	Removal Success (%)
			AN	DN	AN	DN	AN	DN		
c2670	8	99.98	99.79	100	100	99.98	99.90	99.99	2 DN	100
c3540	6	99.98	99.55	99.99	99.78	99.98	99.67	99.99	2 DN 1 AN	100
c5315	8	99.99	99.90	100	100	99.99	99.95	99.99	1 DN	100
c7552	8	100	100	100	100	100	100	100	–	100
b14_C	6	100	100	100	100	100	100	100	–	100
b15_C	6	100	100	100	100	100	100	100	–	100
b20_C	6	100	100	100	100	100	100	100	–	100
b21_C	6	99.99	100	99.99	99.92	100	99.96	99.99	1 AN	100
b17_C	6	100	100	100	100	100	100	100	–	100

C. Breaking SFLL-HD$^{h>0}$ and TTLock with GNNUnlock

For the SFLL-HD^2 ISCAS-85 dataset, the GNN gave 100% node classification accuracy for `20` tested graphs and an average of 99.83% accuracy across all `45` ISCAS-85 graphs. In total, `38` nodes were misclassified then rectified using post-processing. For example, `27` *design* nodes, coming from the different `25` graphs were misclassified as *perturb* nodes. *The misclassified nodes belonged to NOR-tree-like structures in the original designs.* This misclassification did not affect the removal of the protection logic because non-protected input patterns controlled these nodes and hence predictions were dropped. For SFLL-HD^2 ITC-99 dataset, the GNN gave 100% node classification accuracy for `29` of the tested graphs and an average of 99.97% accuracy across all `54` graphs (Table V).

For the TTLock ISCAS-85 dataset, the GNN gave 100% node classification accuracy for 29 of the tested graphs and an average of 99.94% accuracy across all 45 graphs. For the remaining 16 graphs, only 19 nodes were misclassified. The results show that **GNNUnlock can accurately detect all three types of nodes** with precision and recall values ranging from 93.17% to 100%. For the TTLock ITC-99 dataset, the GNN gave 100% node classification accuracy for 16 of the tested graphs and an average of 99.95% accuracy across all 54 graphs. Comparing accuracy results on ISCAS-85 Vs. ITC-99 (for both SFLL-HD$^{h>0}$ and TTLock) confirms that **GNNUnlock has a consistent performance for predicting the protection logic regardless of the benchmark size**. Table VI represents all the averaged results for TTLock datasets. It was observed that the *restore* predictor is 100% successful (with 100% precision and recall) in all ITC-99 test cases. This also confirms that the separation between the *perturb* and *design* nodes is challenging.

To study the effect of h on the performance of GNNUnlock, we launch the attack on the SFLL-HD^4 ITC-99 dataset. Note: due to lack of space, we only report the averaged metrics in

TABLE V. RESULTS OF GNNUNLOCK ON SFLL-HD^2. THE *design*, *perturb*, AND *restore* NODES ARE INDICATED BY DN, PN, AND RN, RESPECTIVELY

Test Set	#Test Graphs	GNN Acc. (%)	Prec. (%)			Rec. (%)			F1-Score (%)			#Misclassified Nodes	Removal Success (%)
			RN	PN	DN	RN	PN	DN	RN	PN	DN		
c2670	12	99.53	99.55	96.36	100	100	99.37	99.46	99.77	97.84	99.73	24 DN as PN 4 PN as RN	100
c3540	9	99.79	98.31	98.97	100	100	98.30	99.88	99.15	98.63	99.94	5 PN as RN 3 DN as PN 2 DN as RN	100
c5315	12	100	100	100	100	100	100	100	100	100	100	–	100
c7552	12	99.99	100	100	99.99	100	99.85	100	100	99.93	99.99	1 PN as DN	100
b14_C	9	99.97	99.88	100	99.98	100	99.43	100	99.94	99.72	99.99	2 PN as RN 5 PN as DN	100
b15_C	9	99.99	99.94	100	99.99	100	99.68	100	99.97	99.84	99.99	3 PN as DN 1 PN as RN	100
b20_C	9	99.98	99.46	100	99.99	100	99.04	100	99.73%	99.52%	99.99%	9 PN as RN 2 PN as DN	100
b21_C	9	100	100	100	100	100	100	100	100	100	100	–	100
b22_C	9	99.96	99.94	97.83	99.99	100	99.67	99.96	99.97	98.74	99.98	1 PN as RN 3 PN as DN 27 DN as PN	100
b17_C	9	99.94	99.46	95.60	100	100	99.52	99.95	99.73	97.52	99.98	3 DN as RN 57 DN as PN 6 PN as RN	100

TABLE VI. EXAMINING THE EFFECT OF h VALUE AND TECHNOLOGY NODE ON THE PERFORMANCE OF GNNUNLOCK. TR STANDS FOR TRAINING

Dataset	Benchmarks	Tech. Node (nm)	GNN Acc. (%)	Avg. Prec. (%)	Avg. Rec. (%)	Avg. F1-Score (%)	Removal Success (%)	Avg. TR Time (s)
TTLock	ISCAS-85	45	99.94	99.20	99.53	99.33	100	1,910
TTLock	ITC-99	45	99.95	97.48	99.36	98.33	100	22,777
SFLL-HD^2	ITC-99	45	99.94	98.73	99.77	99.21	100	24,123
SFLL-HD^2	ITC-99	65	99.97	99.56	99.85	99.70	100	22,052
SFLL-HD^4	ITC-99	65	99.90	98.40	99.60	98.95	100	233,017
SFLL-HD^{16}	ISCAS-85	65	99.24	97.68	97.30	97.40	100	1,907
SFLL-HD^{32}	ITC-99	65	99.83	97.56	98.45	97.96	100	28,049
SFLL-HD^{64}	ITC-99	65	99.69	97.60	97.96	97.74	100	32,494

Table VI. Here, we also report the results of GNNUnlock on the SFLL-HD^2 ITC-99 dataset with a technology node of $45nm$ to evaluate the effect of the target library on the attack's performance. The trend is the same in all cases, while there is a small difference in the values. GNNUnlock detects all three types of nodes accurately with average accuracy values of 99.94%, 99.97%, and 99.90%, for SFLL-HD^2 in $45nm$, SFLL-HD^2 in $65nm$, and SFLL-HD^4 in $65nm$ datasets, respectively. After post-processing, all nodes were classified correctly (100%), verifying that **GNNUnlock can handle different technology nodes and various parameter settings.**

D. Comparison with State-of-the-art Attacks

To demonstrate the superiority of GNNUnlock, a set of ITC-99 benchmarks are locked using SFLL-HD with $K/h = 2$, generating two datasets: one for $K = 128$, $h = 64$ and one for $K = 64$, $h = 32$. The ISCAS-85 benchmarks are locked using $K = 32$, $h = 16$. The characteristics are listed in Table III. When the FALL attacks [5] were launched on these benchmarks, they reported 0 keys. Next, the attack platform of SFLL-HD-Unlocked [4] was launched on the same benchmarks, which failed to identify the perturb signals and did not recover the secret key. However, our GNNUnlock was 100% successful in retrieving the unlocked designs in all of the cases. The results are documented in Table VI.

VI. CONCLUSION

In this paper, we proposed the first-of-its-kind scheme for unlocking provably secure logic locking (PSLL), leveraging a graph neural network (GNN) that learns the common structural features of the protection logic added by such techniques.

The GNN does not learn a syntactic implementation but absorbs the protection logic trend, allowing it to handle variants naturally. We also developed a post-processing algorithm to rectify any misclassifications by the GNN, depending solely on the nodes' connectivity, the GNN predictions, and the known properties of the protection logic. We demonstrated the superiority of GNNUnlock by comparing it to two state-of-the-art attacks in unlocking three PSLL techniques, Anti-SAT, TTLock, and SFLL-HD. Our GNNUnlock scheme was able to break all locked benchmarks while state-of-the-art attacks struggled with some corner cases. We believe GNNUnlock opens up new frontiers in advancing the state-of-the-art defenses and attacks in logic locking.

ACKNOWLEDGEMENTS

This work was carried out in part on the High-Performance Computing resources at Khalifa University. Besides, this work is supported in part by the Center for Cyber Security (CCS) at New York University Abu Dhabi (NYUAD).

REFERENCES

[1] M. Yasin *et al.*, "Provably-secure Logic Locking: From Theory to Practice," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 1601–1618.
[2] ——, "What to Lock?: Functional and Parametric Locking," in *Proc. Great Lakes Symp. VLSI*, 2017, pp. 351–356.
[3] Y. Xie *et al.*, "Mitigating SAT Attack on Logic Locking," in *Proc. Int. Conf. Cryptograph. Hardw. Embedded Syst.* Springer, 2016, pp. 127–146.
[4] F. Yang *et al.*, "Stripped Functionality Logic Locking with Hamming Distance-Based Restore Unit (SFLL-hd)–Unlocked," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 10, pp. 2778–2786, 2019.
[5] D. Sirone *et al.*, "Functional Analysis Attacks on Logic Locking," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 2514–2527, 2020.
[6] L. Alrahis. (2020) Netlist-to-graph Conversion Source Code. [Online]. Available: https://github.com/DfX-NYUAD/GNNUnlock
[7] J. Roy *et al.*, "Ending Piracy of Integrated Circuits," *IEEE Trans. Comput.*, vol. 43, no. 10, pp. 30–38, 2010.
[8] J. Rajendran *et al.*, "Fault Analysis-Based Logic Encryption," *IEEE Trans. Comput.*, vol. 64, no. 2, pp. 410–424, 2015.
[9] S. Dupuis *et al.*, "A Novel Hardware Logic Encryption Technique for Thwarting Illegal Overproduction and Hardware Trojans," in *Proc. IEEE Int. Onl. Test. Symp.*, 2014, pp. 49–54.
[10] P. Subramanyan *et al.*, "Evaluating the Security of Logic Encryption Algorithms," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust*, 2015, pp. 137–143.
[11] M. Li *et al.*, "Provably Secure Camouflaging Strategy for IC Protection," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, 2016, pp. 28:1–28:8.
[12] B. Shakya *et al.*, "CAS-Lock: A Security-Corruptibility Trade-off Resilient Logic Locking Scheme," *IACR Trans. on Cryptograph. Hardw. and Embedded Syst.*, pp. 175–202, 2020.
[13] M. Yasin *et al.*, "Security Analysis of Anti-SAT," *Proc. Asia and South Pacific Design Autom. Conf.*, pp. 342–347, 2016.
[14] L. Alrahis *et al.*, "Functional Reverse Engineering on SAT-Attack Resilient Logic Locking," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2019, pp. 1–5.
[15] Y. Shen *et al.*, "Double DIP: Re-evaluating Security of Logic Encryption Algorithms," in *Proc. Great Lakes Symp. VLSI*, 2017, pp. 179–184.
[16] K. Shamsi *et al.*, "AppSAT: Approximately Deobfuscating Integrated Circuits," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust*, 2017, pp. 95–100.
[17] X. Xu *et al.*, "Novel Bypass Attack and BDD-based Tradeoff Analysis Against All Known Logic Locking Attacks," in *Proc. Int. Conf. Cryptograph. Hardw. Embedded Syst.* Springer, 2017, pp. 189–210.
[18] W. Hamilton *et al.*, "Inductive Representation Learning on Large Graphs," in *Proc. Adv. Neur. Inf. Proc. Sys.*, 2017, pp. 1024–1034.
[19] H. Zeng *et al.*, "GraphSAINT: Graph Sampling Based Inductive Learning Method," in *Proc. of the Int. Conf. on Learn. Rep.*, 2020. [Online]. Available: https://github.com/GraphSAINT/GraphSAINT