MuxLink: Circumventing Learning-Resilient MUX-Locking Using Graph Neural Network-based Link Prediction

Lilas Alrahis[‡], Satwik Patnaik[†], Muhammad Shafique[‡], and Ozgur Sinanoglu[‡]

[‡]Division of Engineering, New York University Abu Dhabi, UAE
[†]Electrical & Computer Engineering, Texas A&M University, College Station, Texas, USA
{lma387, muhammad.shafique, ozgursin}@nyu.edu, satwik.patnaik@tamu.edu

***Abstract*—Logic locking has received considerable interest as a prominent technique for protecting the design intellectual property from untrusted entities, especially the foundry. Recently, machine learning (ML)-based attacks have questioned the security guarantees of logic locking, and have demonstrated considerable success in deciphering the secret key without relying on an oracle, hence, proving to be very useful for an adversary in the fab. Such ML-based attacks have triggered the development of learning-resilient locking techniques. The most advanced state-of-the-art deceptive MUX-based locking (D-MUX) and the symmetric MUX-based locking techniques have recently demonstrated resilience against existing ML-based attacks. Both defense techniques obfuscate the design by inserting key-controlled MUX logic, ensuring that all the secret inputs to the MUXes are equiprobable.**

In this work, we show that these techniques primarily introduce local and limited changes to the circuit without altering the global structure of the design. By leveraging this observation, we propose a novel graph neural network (GNN)-based link prediction attack, *MuxLink*, that successfully breaks both the D-MUX and symmetric MUX-locking techniques, relying only on the underlying structure of the locked design, i.e., in an oracle-less setting. Our trained GNN model learns the structure of the given circuit and the composition of gates around the non-obfuscated wires, thereby generating meaningful link embeddings that help decipher the secret inputs to the MUXes. The proposed MuxLink achieves key prediction accuracy and precision up to 100% on D-MUX and symmetric MUX-locked ISCAS-85 and ITC-99 benchmarks, fully unlocking the designs. We open-source MuxLink [1].

***Index Terms*—Deceptive Logic Locking, Graph Neural Networks, Machine Learning, Link Prediction, Oracle-less Attack**

I. Introduction

The globalized and, thus, distributed semiconductor supply chain creates an attack vector for the untrusted entities in stealing the intellectual property (IP) of a design. To ward off the threat of IP theft, researchers developed various countermeasures like state-space obfuscation, split manufacturing, and logic locking (LL). LL entails inserting additional key logic (XOR/XNOR gates, MUXes) in the original design. These added logic gates (referred to as key-gates) are driven by a secret key (known only to the designer) through an on-chip tamper-proof memory. We illustrate examples of LL in Fig. 1. The prime reason behind the widespread prevalence of LL is because it protects the design IP throughout the supply chain (foundry, test facility, end-users).

Researchers have developed various attacks on LL to recover the secret key, considering two threat models: the *oracle-guided* and the *oracle-less*. The oracle-guided attacks require a functional chip (with the secret key embedded) acting as an *oracle* [2], which may not be available in many practical settings. Towards a more realistic scenario, the oracle-less attacks rely only on the structure of the locked design, posing a significant threat to LL. Recently, machine learning (ML) algorithms have

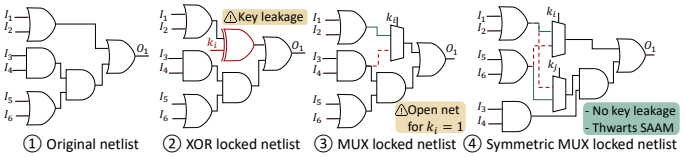

Fig. 1. Examples of XOR logic locking (LL) and MUX-based LL.

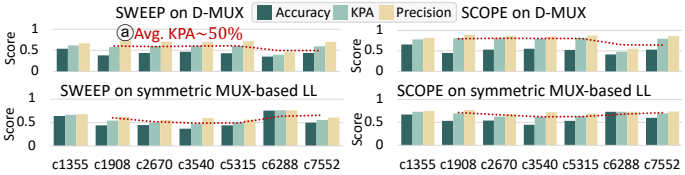

Fig. 2. D-MUX [10] and symmetric MUX-based LL [14] are resilient to the constant propagation attacks SWEEP [15] and SCOPE [14].

facilitated various oracle-less attacks on LL [3]–[8] leading to the development of learning-resilient LL to counteract these threats [4], [9], [10]. Other ML-based attacks [11]–[13] follow the oracle-guided model. We focus on the oracle-less threat model, which is a more challenging attack model.

In this work, we focus on the deceptive MUX-based (D-MUX) [10] and the symmetric MUX-based LL techniques [14]. In the following, first, we discuss the aspects of D-MUX and symmetric MUX-based locking which make them learning-resilient and outline the key research challenges for our work.

A. Key Research Challenges Targeted in this Work

1) **No key leakage:** Existing ML-based attacks [3], [5], [7] capture the local locking-induced structural modifications, which embed key information to decipher the secret key. An example of XOR LL is depicted in Fig. 1②, where there is a direct mapping between the type of the key-gate and the key-value. Although re-synthesis is performed to induce local transformations around the key-gate, thereby breaking the mapping between the type of the key-gate and key-value, potent ML-based attacks such as *SAIL* [3], *SnapShot* [5] and *OMLA* [7] expose the key information from the surrounding circuitry. Learning-resilient MUX-based locking eliminates the key-related leakage in the structural residue from locking, i.e., the location selection of the key-logic and the introduced changes, thwarting existing ML-based attacks. The authors in [10] launched SnapShot [5] on D-MUX locked benchmarks and demonstrated that the attack reports a consistent key prediction accuracy (KPA) around 50%, which means that D-MUX forces SnapShot to perform random guesses about the secret key.
2) **No circuit reduction:** A MUX key-gate takes in two wires from the design as inputs. A key-input acts as the select

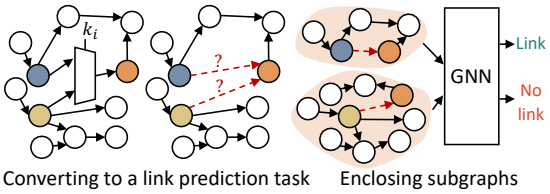

Fig. 3. Unlocking MUX-based obfuscation using GNN-based link prediction.

line, passing the true wire upon applying the correct key-value. One major vulnerability in naïve MUX-based locking is that a wrong key-bit could remove an entire logic cone (the true cone) from the circuit. An example is illustrated in Fig. 1③, where setting $k_i = 1$ disconnects the true wire (green). The structural analysis attack *SAAM* [10] exposed this vulnerability. The learning-resilient MUX-based locking techniques ensure no reduction in the design for the wrong key-values and are completely resilient to SAAM.

3) **Symmetric paths:** Naïve MUX-based locking is vulnerable to the constant propagation attacks *SWEEP* [15] and *SCOPE* [14]. The attacks hard-code the value of one key-bit at a time and perform re-synthesis. A difference in the design features, including power consumption, total area, etc., is observed in the re-synthesized circuits, which correspond to the two possible values for a single key-input. Thus, the attacks learn the correlation between the extracted features and the correct key. The D-MUX and the symmetric MUX-based LL add pairs of MUX key-gate, where the true logic cones of the MUXes are symmetrically interconnected, eliminating this kind of attack. In addition, ensuring no circuit reduction for wrong key-values (see the point above) enhances the resilience against constant propagation attacks. An example of symmetric MUX-based LL is shown in Fig. 1④. We locked seven ISCAS-85 benchmarks using D-MUX and the symmetric MUX-based LL. We performed the same evaluation as in [10], where each circuit is copied 100 times and locked with a key-size of $K = 64$; resulting in 700 locked benchmarks. The locked benchmarks are directly attacked using SCOPE as it does not require training. For SWEEP, we generate one dataset for each target benchmark, where the 100 locked versions are kept for testing, while the 600 other benchmarks and used for training. We repeat this analysis for both techniques, and the average accuracy, precision, and KPA for each benchmark are shown in Fig. 2. The attacks report an average $KPA \approx 50\%$ (see Fig. 2ⓐ)[1] confirming the resilience of the techniques against existing learning-based and constant propagation attacks.

B. Our Novel Research Contributions

In this work, we propose the *MuxLink* platform, that leverages efficient graph neural network (GNN)-based link prediction to devise the first-ever attack on the MUX-based learning-resilient LL.[2] We decipher the secret MUX inputs in an *oracle-less* setting by exploiting the underlying structure of the target design *without requiring any circuit library or re-locking for training*. We show that D-MUX and symmetric-MUX-based locking introduce only limited local modifications to the designs, and provide resilience against locality-based learning attacks specifically, but not *necessarily* against *any* learning-based approach. Our novel contributions are as follows.

1) A **link prediction-based platform** is developed, in which the task of deciphering the secret MUX inputs is converted to a link prediction problem. We employ a GNN to learn meaningful link representations and decipher the true connections, recovering the original design.
2) A **key recovery post-processing** guided by the likelihood scores of the GNN is developed to recover the secret key.

Major Results: We evaluate the effectiveness of MuxLink through an extensive experimental analysis on selected ISCAS-85 and ITC-99 benchmarks locked using D-MUX [10] and symmetric MUX-based locking [14]. MuxLink deciphers up to 100% of the key-bits with a precision up to 100% in seconds, unlocking benchmarks which the other state-of-the-art oracle-less attacks fail to unlock. **We open-source MuxLink [1].**

[1]More details on the evaluation metrics are given in Sec. IV.

[2]*GNNUnlock* [6], [16] is a GNN-based oracle-less attack targeting SAT-resilient LL, in which the protection logic has a structure that can be learned by a GNN, isolated, and detached. D-MUX and the symmetric MUX-based LL do not have isolable logic, and thus, are not applicable to GNNUnlock.

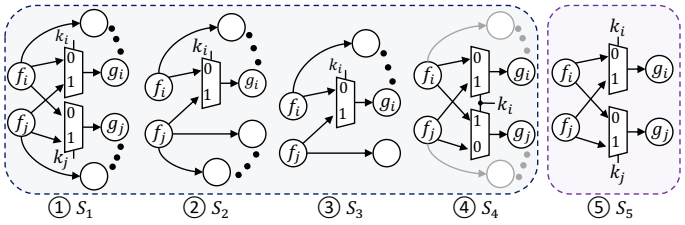

Fig. 4. $S_1 \rightarrow S_4$ are the D-MUX locking strategies. S_5 is the symmetric MUX-based locking strategy.

II. Background and Related Works

A. Tests for Learning-Resilient LL

The authors in [10] consider two types of designs to evaluate LL for learning resilience. The first category includes designs that are synthesized with only a single type of gate (e.g., AND gate). The second category includes synthesized designs constituting randomly selected and well-distributed logic gates. In addition, the authors propose two learning-resilience tests, AND netlist test (ANT) and random netlist test (RNT). Suppose a locking technique fails either of the two tests. In that case, the locking technique in question is regarded as conclusively vulnerable as the resilience of the technique is governed by the secret key and the structure of the design.

B. Initial Learning-Resilient LL Techniques

UNSAIL [4] injects identical key-gate structures with differing key-bit values in the locked design, which leads to flawed inferences from the ML models used in the SAIL attack [3]. The key-gate insertion phase in UNSAIL is guided by a targeted re-synthesis procedure. A learning-resilient LL should ideally deliver security without having any dependence on the synthesis tool to obfuscate the locking-induced transformations [10].

In [9], the authors introduced truly random LL (TRLL), in which random decisions are made regarding the insertion of the key-gates. TRLL involves (i) replacement of inverters with an XOR key-gate, (ii) insertion of XOR key-gate at any location in the design, and (iii) insertion of XOR key-gate followed by an inverter. Although TRLL does not rely on synthesis tools and passes RNT, it fails ANT since there are no inverters to be replaced or coupled with an XOR gate, and this technique reduces to a conventional XOR-based LL technique.

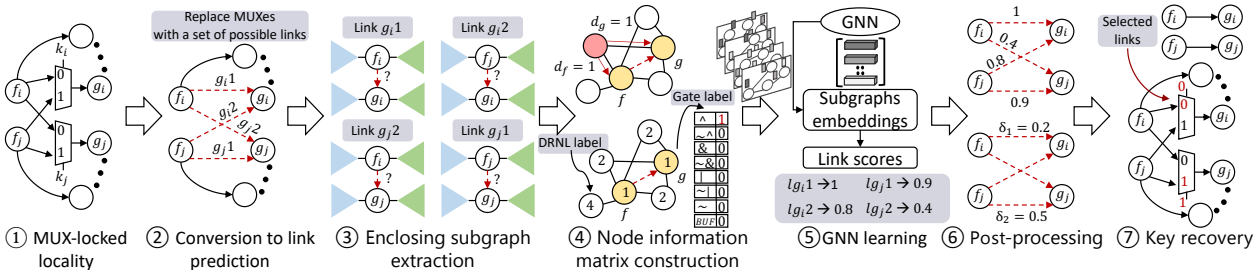

Fig. 5. The methodology work flow. The blue and green triangles represent fan-in cones and fan-out cones, respectively.

1) Deceptive MUX-based LL (D-MUX) [10]: Supporting both the ANT and RNT concepts remains a challenge for X(N)OR-based LL. The authors in [10] state that vulnerabilities manifest due to the insertion of additional logic without leaving key-related, structural traces. Furthermore, the authors conjecture that MUX-based LL has an important advantage, as it inserts the same structure (i.e., a MUX), and it reconfigures the existing logic. A new learning-resilient LL, D-MUX, is proposed [10], which ensures that each wire feeding to the MUX has the same probability of being true/false. Multiple locking strategies (S_1–S_4 in Fig. 4) are followed by D-MUX.

In the S_1 strategy, two multi-output nodes $\{f_i, f_j\}$ are selected as inputs to two locking MUXes. The MUXes obfuscate one randomly selected output node for each input node, i.e., $\{g_i, g_j\}$. Two individual key-inputs $\{k_i, k_j\}$ are used, where each key-input acts as a select line for one MUX. The S_2 strategy selects two multi-output nodes $\{f_i, f_j\}$, but performs locking using a single key-input k_i controlling a single MUX. One randomly selected output node for a randomly selected input node is locked. E.g., in Fig. 4②, S_2 selects f_i and one of its output nodes g_i. The S_3 strategy selects and locks one multi-output node f_i using a single key-input k_i controlling one MUX. f_j in the case of S_3 is a single-output node. Finally, the S_4 strategy sets no restrictions on $\{f_i, f_j\}$. A single key-input k_i drives two MUXes and locks one output node for each input node. In all the strategies, the MUXes are configured to cause no circuit reduction and no combinational loops.

The cost of the S_4 strategy, in terms of the number of gates added, is larger compared to the rest of the strategies. However, S_4 is always applicable as there are no restrictions on $\{f_i, f_j\}$. To reduce costs, the enhanced D-MUX (eD-MUX) only uses S_4 when none of the other strategies is viable.

2) Symmetric MUX-based LL [14]: Concurrent to D-MUX, Alaql *et al.* [14] propose another technique (Fig. 4⑤), which can be considered as a special case of D-MUX. We denote this locking strategy as S_5. Note that S_5 is equivalent to S_4, but two individual key-inputs are driving the individual MUXes. Here, $\{f_i, f_j\}$ are one-output nodes.

C. Link Prediction Problem

Link prediction refers to the problem of inferring missing links from an observed graph. Let $\mathcal{G}(\mathcal{D}, \mathcal{V})$ denotes a graph with a set of edges $\mathcal{D}$ and a set of nodes $\mathcal{V}$. Given $\mathcal{V}$ and a subset of true links $\mathcal{E} \in \mathcal{D}$, the objective is to identify the unobserved true links $\mathcal{S}$ referred to as *target links*, where $\mathcal{D} = \mathcal{E} + \mathcal{S}$. Link prediction has varied usages in recommender systems, drug discovery, and knowledge graph completion, etc. Traditional link prediction heuristics rely on handcrafted features, which might fail to express the complex patterns in the graph that actually determine the link formations. Recently, the authors in [17] demonstrated how GNNs can directly learn suitable "heuristics" from local enclosing subgraphs around links. *SEAL* [17] extracts an enclosing subgraph around each target link, computes a subgraph embedding using a GNN, and uses it for link prediction. Since the subgraph embeds information regarding the target link, the label of the subgraph can be considered the label of the target link.

D. Graph Neural Networks (GNNs)

GNNs generate an embedding for each node $v \in \mathcal{V}$ in a graph $\mathcal{G}$ through iterations of *message passing* [18] as follows, where h_v^l denotes the embedding of v at the l^{th} iteration.

$$a_v^l = AGG^l(\{h_u^{l-1} : u \in N(v)\}) \quad (1)$$

$$h_v^l = COMBINE^l(h_v^{l-1}, a_v^l) \quad (2)$$

The AGG function collects information from the neighbors of v, $N(v)$, and extracts an embedding a_v^l for the layer l. The COMBINE function updates the features of v by combining h_v^{l-1} with a_v^l. The updated embedding, h_v^l, captures information regarding v and its neighborhood. After L iterations of message passing, a *read-out* is performed to generate a graph-level embedding, $h_\mathcal{G}$, which can be used for graph classification.

III. Proposed MuxLink Attack Model

Attack Model: We assume an adversary located in the fab with access only to the GDSII representation of a locked design. The attacker performs reverse engineering to obtain the locked netlist and determines the location of the key-gates by tracing the key-inputs from the tamper-proof memory. Fig. 5 shows an overview of the main steps of MuxLink.

A. Enclosing Subgraph Extraction

The first step is identifying the key-controlled MUXes by tracing the key-inputs (see Fig. 5①) and removing them from the netlist. The netlist is then converted to an undirected graph $\mathcal{G} = (\mathcal{E}, \mathcal{V})$, where $\mathcal{V}$ represents the set of nodes (gates), and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ represents the set of observed links (wires). $\boldsymbol{A}$ is the symmetric adjacency matrix of $\mathcal{G}$. The graph representation of the netlist does not include primary inputs and primary outputs, as we are interested in capturing the composition of gates and their connectivity. All the inputs to the MUXes are marked as target links, added to set $\mathcal{S}$ and excluded from $\mathcal{E}$ (see Fig. 5②). Next, MuxLink extracts an h-hop enclosing subgraph for each pair of target nodes f and g (see Fig. 5③). The h-hop enclosing subgraph for (f, g) is induced from $\mathcal{G}$ containing the nodes $\{\ j \mid d(j,f) \leq h \text{ or } d(j,g) \leq h\ \}$, where $d(y, x)$ is the shortest path distance between x and y. As discussed in Sec. II, the techniques only check if the inputs to the MUXes $\{f_i, f_j\}$ are driving a single gate or multiple gates.

However, the locking techniques do not consider the structure of the fan-out and fan-in cones of the selected gates, and hence, $\{f_i, f_j, g_i, g_j\}$ all have unique surroundings. Consequently, the extracted subgraphs in Fig. 5③ are different and will have distinct link representations, allowing MuxLink to decipher the correct connections.

B. Node Information Matrix Construction

A node information matrix $\boldsymbol{X}$ is constructed for each extracted subgraph, where each node is associated with an 8-bit one-hot encoded vector that encodes its Boolean functionality. E.g., the feature vector of node g in Fig. 5④ indicates that it is an XOR gate. For link prediction, the GNN must distinguish the target link and capture the relationship between the target nodes (colored in yellow in Fig. 5④) and the surrounding circuitry. To achieve this, we employ the double radius node labeling (DRNL) [17]. Each node in the subgraph is assigned a tag that captures its relationship with the target link. Let f and g be the target nodes, the DRNL label $f_l(j)$ of a node j is:

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

where $d_f = d(j, f)$, $d_g = d(j, g)$, and $d = d_f + d_g$. $(d/2)$ is the integer quotient and $(d\%2)$ is the remainder of d divided by 2. See Fig. 5④ for an example of DRNL labeling. If j has a path to only one of the target nodes, then $f_l(j) = 0$. The target nodes are tagged with 1 allowing the GNN to differentiate them from the rest of the gates. Each node's label is one hot-encoded and concatenated to its corresponding row in $\boldsymbol{X}$. The dimension of $\boldsymbol{X}$ depends on the largest assigned label in a given dataset, which depends on the target circuit and the subgraph size.

C. Dataset Generation

MuxLink takes the graph representation of the target netlist $\mathcal{G}$ and extracts enclosing subgraphs for a set of sampled positive links (observed wires) and a set of sampled negative links (unobserved wires) for training. We generate a balanced dataset and use a maximum of $100,000$ training links. 10% of the sampled links are kept for validation. The links between the target nodes are always removed from the subgraphs. During the attack phase, the enclosing subgraphs around the links in $\mathcal{S}$ are fed to the trained GNN, as shown in Fig. 5⑤. The GNN reports the likelihood score for each link.

D. GNN Learning

We employ the deep graph convolutional neural network (DGCNN) [18] for graph classification. A DGCNN layer performs the following operation, where $\boldsymbol{H}^{l+1} \in \mathbb{R}^{n \times c_{l+1}}$ is the output embedding matrix of layer l, c is the number of output channels, and n is the number of nodes in the subgraph.

$$\boldsymbol{H}^{l+1} = \sigma(\tilde{\boldsymbol{D}}^{-1}(\boldsymbol{A} + \boldsymbol{I})\boldsymbol{H}^l\boldsymbol{B}^l) \quad (4)$$

$\tilde{\boldsymbol{D}}$ is the diagonal degree matrix, $\boldsymbol{B}^l$ is a trainable weight matrix, and $\sigma(.)$ is a non-linear activation function. The initial embedding matrix is the same as the node information matrix $\boldsymbol{H}^0 = \boldsymbol{X}$. After L aggregation layers, the following concatenation is performed by DGCNN $\boldsymbol{H}^{1:L} := [\boldsymbol{H}^1, ..., \boldsymbol{H}^L]$ to represent a subgraph by a single vector. The tensor is then sorted row-wise according to $\boldsymbol{H}^L$ and reshaped to $k(\sum_{l=1}^{L} c_l) \times 1$, selecting k nodes to represent the subgraph. In MuxLink, we set k such that 60% of subgraphs have nodes less than or equal to k. Then, the final obtained embedding is fed to 1-D convolutional layers for classification.

Algorithm 1: MUXLINK POST-PROCESSING FOR $\{S_1, S_4, S_5\}$

```
1: Input: Threshold th, Likelihoods L
2: Output: Deciphered keys {k_i, k_j}
3: δ_1 = |l_{g_i1} − l_{g_i2}| //Diff in likelihood scores for g_i
4: δ_2 = |l_{g_j1} − l_{g_j2}| //Diff in likelihood scores for g_j
5: if δ_1 >= th||δ_2 >= th then
6:     if δ_1 > δ_2 then
7:         if l_{g_i1} > l_{g_i2} then
8:             k_i = 0, k_j = 1
9:         else
10:            k_i = 1, k_j = 0
11:    else if δ_2 > δ_1 then
12:        if l_{g_j1} > l_{g_j2} then
13:            k_i = 0, k_j = 1
14:        else
15:            k_i = 1, k_j = 0
16:    else
17:        k_i = X, k_j = X
18: else
19:    k_i = X, k_j = X
20: return {k_i, k_j}
```

E. Post-processing

The likelihood scores are processed to recover the secret key. Key prediction depends on the structure of the obfuscated locality and a controlled threshold parameter th. We describe the MuxLink post-processing approach for the different localities in the following subsections.

1) S_1 and S_5: Two key-inputs $\{k_i, k_j\}$ control two MUXes with the same inputs $\{f_i, f_j\}$. These strategies obfuscate two output nodes $\{g_i, g_j\}$. Hence, four links are considered during post-processing $\{(f_i, g_i), (f_j, g_i), (f_j, g_j), (f_i, g_j)\}$ denoted as $\{g_i1, g_i2, g_j1, g_j2\}$, respectively. The post-processing of such locked localities is outlined in Algorithm 1. $l_x \in [0, 1]$ is the likelihood score for link x. First, the absolute difference δ between the likelihood scores of the possible links for each gate $\{g_i, g_j\}$ is computed (lines 3-4). If none of the $\{\delta_1, \delta_2\}$ values is greater than th, MuxLink does not make a decision and reports X for both key-bits (lines 18-19). Else, MuxLink checks which difference is larger (δ_1 or δ_2), then selects the link that has the highest likelihood score as the true link and predicts the key-bit value that passes that link. In the example shown in Fig. 5⑥, $\delta_1 = |1 - 0.8| = 0.2$ and $\delta_2 = |0.9 - 0.4| = 0.5$. If $th = 0.01$, MuxLink will execute lines 11-17 as $\delta_2 > \delta_1$. Since $l_{g_j1} > l_{g_j2}$, $k_j = 1$ and $k_i = 0$ (lines 12-13).

2) S_2 and S_3: Here, a single key-input k_i controls one MUX, which obfuscates a single output node g_i. Hence, two links are considered during post-processing $\{g_i1, g_i2\}$. MuxLink computes the difference in likelihood scores between the two possible links, as follows: $\delta = |l_{g_i1} - l_{g_i2}|$. If $\delta < th$, MuxLink assigns $k_i = X$. However, if $\delta >= th$ and $l_{g_i1} > l_{g_i2}$, then $k_i = 0$. If $\delta >= th$ and $l_{g_i2} > l_{g_i1}$, then $k_i = 1$.

3) S_4: One key-input k_i controls two MUXes, with the same inputs $\{f_i, f_j\}$ but in a different order. This strategy obfuscates two output nodes $\{g_i, g_j\}$. The same analysis outlined in Algorithm 1 is followed, but only k_i is returned, ignoring k_j.

IV. EVALUATION OF OUR MUXLINK ATTACK MODEL

We summarize the experimental setup and the process of dataset generation in Fig. 6. We evaluate MuxLink on selected designs from the ISCAS-85 and ITC-99 combinational

benchmarks locked using D-MUX [10] and symmetric MUX-based locking [14]. We implement both techniques in Python as described in [10], [14]. We use the eD-MUX implementation of D-MUX.[3] We implement the post-processing in Perl.

We lock the ISCAS-85 benchmarks with $K : \{64, 128, 256\}$, except for the `c1355` benchmark where $K = 256$ was not achievable due to the small size of the design. Additionally, we lock the larger ITC-99 benchmarks with $K : \{256, 512\}$, resulting in a total of 64 locked designs. The benchmarks are locked and attacked in *BENCH* format, following the methodology widely used by the LL community, and as also adopted by SWEEP and SCOPE attacks [14], [15].

GNN Topology: We start with the default DGCNN architecture of [18], which has four graph convolution layers with $\{32, 32, 32, 1\}$ output channels, two 1-D convolutional layers with $\{16, 32\}$ output channels, a fully-connected layer of 128 neurons, a dropout layer with a dropout rate of 0.5, and a *softmax* layer of 2 output units for classification. We use the *tanh* activation function in the graph convolution layers and the *ReLU* function in the rest of the layers. We use stochastic gradient descent with the Adam updating rule and train DGCNN for 100 epochs with an initial learning rate of 0.0001, and save the model with the best performance on the 10% validation set to predict the testing links. MuxLink runs on a single machine utilizing 10 cores (2x Intel(R) Xeon(R) CPU E5-2680 v4@2.4GHz) and a single NVIDIA-V100 GPU.

Evaluation Metrics: We use four metrics for attack evaluation: *accuracy* (AC), *precision* (PC), KPA, and Hamming distance (HD). AC measures the ratio of correctly deciphered key-bits out of the entire key, i.e., $(K_{correct}/K_{total}) \cdot 100\%$. PC measures the ratio of correctly deciphered keys, counting every X value as a correct guess, i.e., $((K_{correct} + K_X)/K_{total}) \cdot 100\%$. Finally, KPA measures the percentage of correctly deciphered key-bits out of the entire predictions, i.e., $(K_{correct}/(K_{total} - K_X)) \cdot 100\%$.

MuxLink Performance: The AC, PC, and KPA of MuxLink on D-MUX and symmetric MUX-based locked benchmarks are presented in Fig 7, having $h = 3$ and $th = 0.01$.[4] MuxLink achieves an average AC, PC, and KPA of 94.61%, 95.41% and 95.37%, respectively, on the ISCAS-85 benchmarks locked using D-MUX, and an average AC, PC, and KPA of 98.49%, 99.43% and 99.43%, respectively, on the ITC-99 benchmarks locked using D-MUX. Additionally, MuxLink achieves an average AC, PC, and KPA of 96.95%, 97.31% and 97.30%, respectively, on the ISCAS-85 benchmarks locked using symmetric MUX-based LL, and an average AC, PC, and KPA of 98.90%, 99.38% and 99.38%, respectively, on the ITC-99 benchmarks locked using symmetric MUX-based LL. Overall, MuxLink achieves an AC, PC, and KPA up to 100%. These results show that MuxLink is capable of breaking the previously thought of as "learning-resilient" schemes.

Effect of the LL Scheme: The resilience of symmetric MUX-based LL against MuxLink is lower compared to D-MUX. Under the same K, symmetric MUX-based LL locks a fewer number of localities. This is because each obfuscated locality is controlled with two key-inputs $\{k_i, k_j\}$, with only two possible combinations $\{0, 1\}$ and $\{1, 0\}$. However, the equivalent implementation of S_4 in D-MUX (in addition to S_2 and S_3) is controlled via a single key-input. Therefore, D-MUX achieves a larger obfuscation under the same K.

[3]To verify our implementation of the locking techniques, we launched SWEEP [15] and SCOPE [14] attacks on the locked benchmarks and observed the same attack resilience promised by the original work [10], [14] (see Fig. 2).

[4]Later we study the effect of h and th on the performance of the attack.

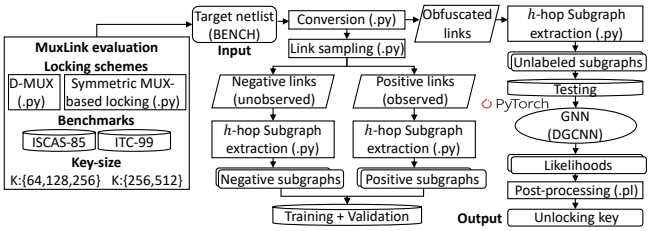

Fig. 6. Experimental setup and tool-flow.

Effect of the Benchmark Size: The broken red lines in Fig. 7 show the moving average of the score (AC, PC, and KPA) of MuxLink for the ISCAS-85 benchmarks locked with $K = 256$ and for the ITC-99 benchmarks locked with $K = 512$, versus the benchmarks (ordered from smallest to largest). The trend lines show that the performance of MuxLink enhances with the increase in the benchmark size. The larger the design is, the lower the impact of the obfuscation is. For instance, the KPA of MuxLink for the D-MUX locked `c1908` benchmark with $K = 256$ is 92.27%, while the KPA for the D-MUX locked `c7552` benchmark with the same K is 98.44%. The plots also illustrate that, in general, the performance of MuxLink is better on the ITC-99 benchmarks compared to the performance on the smaller ISCAS-85 benchmarks.

Effect of the Key-size (K): The increase of K faintly affects the performance of MuxLink on D-MUX. For instance, MuxLink KPA drops from 93.75% to 92.94% on the D-MUX locked `c2670`, when K is increased from $K = 64$ to $K = 256$, respectively. Nevertheless, the average AC, PC, and KPA across the different key-sizes for the ISCAS-85 benchmarks are consistent around 95% for the D-MUX scheme. However, the performance of MuxLink is slightly affected by the increase of K when attacking symmetric MUX-based locking. For example, MuxLink KPA drops from 96.88% to 92.13% on the symmetric MUX-based locked `c2670`, when K is increased from $K = 64$ to $K = 256$, respectively. On average, the KPA of MuxLink drops from 99.11% to 94.65%, when moving from $K = 64$ to $K = 256$ on the symmetric MUX-based locked ISCAS-85 benchmarks.

Hamming Distance: We compute the HD between the outputs of the recovered (D-MUX locked) design by MuxLink and those of the original design. The goal of a defender is to enforce an HD of 50% (maximum corruption), while the objective of an attacker is to recover the original design, i.e., obtain an HD of 0%. For each benchmark, we set the recovered key pattern by MuxLink and compute the HD by simulating 100,000 random input patterns using *Synopsys VCS*. For the cases where some key-bit values are undeciphered (X values), we measure the HD for all the possible remaining key-bit assignments and compute the average. We report the results in Fig. 8. The average HD value for the ISCAS-85 reconstructed by MuxLink is a mere 3.39%. Hence, using MuxLink, we (almost) determine the correct functionality.

Post-processing Threshold (th): We repeated the post-processing stage for a range of $th \in [0, 1]$ with a step of size 0.05. The GNN does not require any re-training as the th value

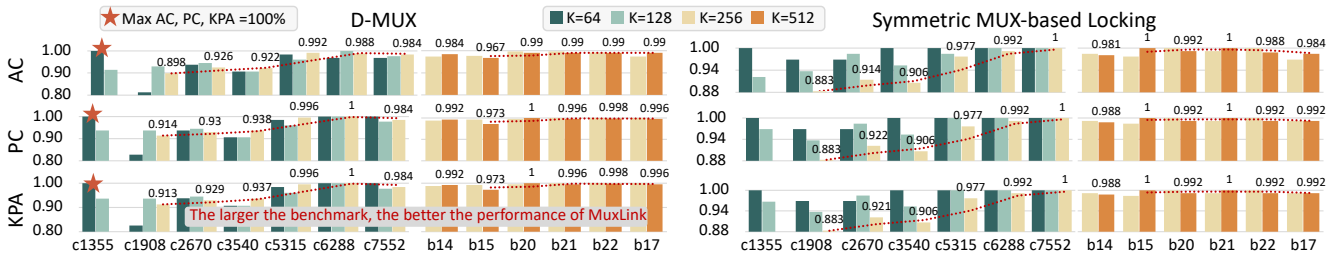

Fig. 7. Accuracy (AC), precision (PC), and KPA for MuxLink on the learning-resilient MUX-based locking techniques. The data labels are added for the ISCAS-85 benchmarks with $K = 256$ and the locked ITC-99 benchmarks with $K = 512$. We work on combinational counterparts of ITC-99 benchmarks.

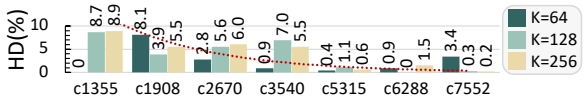

Fig. 8. Hamming distance (HD) between the outputs of original designs and the D-MUX locked designs recovered by MuxLink.

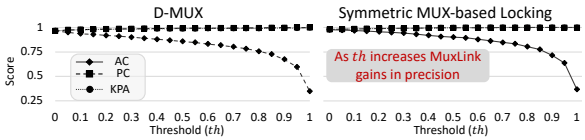

Fig. 9. MuxLink performance under different post-processing th settings.

only affects the post-processing. This analysis is performed for both locking schemes. The average AC, PC, and KPA on the ISCAS-85 and ITC-99 benchmarks under the different th settings are shown in Fig. 9. Setting a strict threshold of $th = 1$ enforces a PC of 100% for all the evaluated benchmarks. The ratio of the predicted key-bit values gets smaller with the increase in th (reaches around 30% for $th = 1$). However, the small set of predicted keys is guaranteed to be correct. Even with a $th = 0$, MuxLink achieves an average PC of 96.54% on the D-MUX locked benchmarks and 98.8% on the symmetric MUX-based locked benchmarks.

Subgraph Size and Runtime: We study the effect of h-hop sampling on the performance and runtime of MuxLink. We repeat the experiments with $th = 0.01$ and vary $h \in [1, 4]$ with a step size of 1 (see Fig. 10). The reported runtime includes subgraph sampling, training, testing, and post-processing. The performance of MuxLink in terms of AC, PC, and KPA improves with the increase in h and saturates after $h \geq 3$. We primarily notice a jump in performance moving from $h = 1$ to $h = 2$. Nevertheless, the 1-hop analysis sheds light on a fundamental vulnerability of the D-MUX and the symmetric MUX-based locking. Although the schemes claim protection at the locality level, MuxLink can decipher the obfuscated connections with high AC even when only considering the 1-hop neighborhood of the obfuscated gates. With the increase in h, the number of neighbors and the runtime of MuxLink increase exponentially. Thus, we limit the hop size to $h = 3$.

Summary: MuxLink was evaluated on two "learning-resilient" techniques: D-MUX [10] and the symmetric MUX-based locking [14]. We consider different key-sizes, h-hop sizes, and threshold th values. On average, MuxLink deciphers 96.87% of the key-bits with a PC of 97.50%. The existing ML-based attacks [3], [5], [7] fail to break these techniques because they try to extract non-existent key leakage. However, MuxLink learns link formation and deciphers the key.

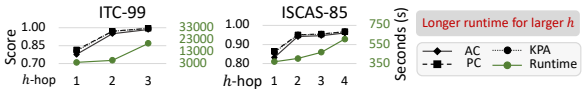

Fig. 10. MuxLink performance and runtime for different h-hop numbers.

V. Conclusion

We propose *MuxLink* as a graph neural network (GNN)-based link prediction attack that successfully breaks the state-of-the-art learning-resilient D-MUX and symmetric MUX-locking, relying only on the structure of the locked design. The GNN learns the structure and connectivity of the target circuit around non-obfuscated wires, thereby generating meaningful link heuristics that help decipher the secret inputs to the locking MUXes. MuxLink achieves accuracy and precision up to 100% on D-MUX and symmetric MUX-locked ISCAS-85 and ITC-99 benchmarks. To the best of our knowledge, MuxLink is the first attack aimed at breaking learning-resilient logic locking. This work exposes a new source of exploitable leakage, i.e., link formation, and demonstrates that there is still a gap in designing learning-resilient logic locking.

References

[1] https://github.com/lilasrahis/MuxLink.
[2] P. Subramanyan *et al.*, "Evaluating the Security of Logic Encryption Algorithms," in *IEEE HOST*, 2015, pp. 137–143.
[3] P. Chakraborty *et al.*, "SAIL: Analyzing structural artifacts of logic locking using machine learning," *IEEE TIFS*, vol. 16, pp. 3828–3842, 2021.
[4] L. Alrahis *et al.*, "UNSAIL: Thwarting oracle-less machine learning attacks on logic locking," *IEEE TIFS*, vol. 16, pp. 2508–2523, 2021.
[5] D. Sisejkovic *et al.*, "Challenging the security of logic locking schemes in the era of deep learning: A neuroevolutionary approach," *JETC*, vol. 17, no. 3, May 2021.
[6] L. Alrahis *et al.*, "GNNUnlock+: A systematic methodology for designing graph neural networks-based oracle-less unlocking schemes for provably secure logic locking," *IEEE TETC*, pp. 1–1, 2021.
[7] ——, "OMLA: An oracle-less machine learning-based attack on logic locking," *IEEE TCAS-II*, pp. 1–1, 2021.
[8] ——, "UNTANGLE: unlocking routing and logic obfuscation using graph neural networks-based link prediction," in *ICCAD*, 2021.
[9] N. Limaye *et al.*, "Thwarting all logic locking attacks: Dishonest oracle with truly random logic locking," *IEEE TCAD*, 2020.
[10] D. Sisejkovic *et al.*, "Deceptive logic locking for hardware integrity protection against machine learning attacks," *IEEE TCAD*, 2021.
[11] H. Chen *et al.*, "GenUnlock: An automated genetic algorithm framework for unlocking logic encryption," in *ICCAD*, 2019, pp. 1–8.
[12] F. Tehranipoor *et al.*, "Deep RNN-oriented paradigm shift through BOCANet: Broken obfuscated circuit attack," in *GLVLSI*, 2019, p. 335–338.
[13] K. Z. Azar *et al.*, "NNgSAT: Neural network guided sat attack on logic locked complex structures," in *ICCAD*, 2020, pp. 1–9.
[14] A. Alaql *et al.*, "SCOPE: Synthesis-based constant propagation attack on logic locking," *IEEE TVLSI*, vol. 29, no. 8, pp. 1529–1542, 2021.
[15] ——, "Sweep to the secret: A constant propagation attack on logic locking," in *AsianHOST*, 2019, pp. 1–6.
[16] L. Alrahis *et al.*, "GNNUnlock: Graph neural networks-based oracle-less unlocking scheme for provably secure logic locking," in *DATE*, 2021, pp. 780–785.
[17] M. Zhang *et al.*, "Link prediction based on graph neural networks," in *NIPS*, ser. NIPS'18. Red Hook, NY, USA: Curran Associates Inc., 2018, p. 5171–5181.
[18] ——, "An end-to-end deep learning architecture for graph classification," in *AAAI*, vol. 32, 2018.