UNTANGLE: Unlocking Routing and Logic Obfuscation Using Graph Neural Networks-based Link Prediction

Lilas Alrahis[‡], Satwik Patnaik[†], Muhammad Abdullah Hanif[§], 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
[§]Institute of Computer Engineering, Technische Universität Wien, Vienna, Austria
{lma387, muhammad.shafique, ozgursin}@nyu.edu, satwik.patnaik@tamu.edu, muhammad.hanif@tuwien.ac.at

***Abstract*—Logic locking aims to prevent intellectual property (IP) piracy and unauthorized overproduction of integrated circuits (ICs). However, initial logic locking techniques were vulnerable to the Boolean satisfiability (SAT)-based attacks. In response, researchers proposed various SAT-resistant locking techniques such as point function-based locking and symmetric interconnection (SAT-hard) obfuscation. We focus on the latter since point function-based locking suffers from various structural vulnerabilities. The SAT-hard logic locking technique, InterLock [1], achieves a unified logic and routing obfuscation that thwarts state-of-the-art attacks on logic locking. In this work, we propose a novel link prediction-based attack, UNTANGLE, that successfully breaks InterLock in an oracle-less setting without having access to an activated IC (oracle). Since InterLock hides selected timing paths in key-controlled routing blocks, UNTANGLE reveals the gates and interconnections hidden in the routing blocks upon formulating this task as a link prediction problem. The intuition behind our approach is that ICs contain a large amount of repetition and reuse cores. Hence, UNTANGLE can infer the hidden timing paths by learning the composition of gates in the observed locked netlist or a circuit library leveraging graph neural networks. We show that circuits withstanding SAT-based and other attacks can be unlocked in seconds with 100% precision using UNTANGLE in an oracle-less setting. UNTANGLE is a generic attack platform (which we also open source [2]) that applies to multiplexer (MUX)-based obfuscation, as demonstrated through our experiments on ISCAS-85 and ITC-99 benchmarks locked using InterLock and random MUX-based locking.**

***Index Terms*—Logic locking, Routing obfuscation, Link prediction, Oracle-less attacks, Graph neural networks.**

I. Introduction

The globalization of the integrated circuit (IC) supply chain has led design companies to outsource the fabrication of chips to off-shore, untrustworthy foundries. Attackers present in these foundries can either steal the design intellectual property (IP) or engage in unauthorized overproduction of ICs [3]. The research community proposed various countermeasures such as logic locking, state-space obfuscation, and split manufacturing (amongst others) to ward off such threats. Logic locking is a holistic technique that can protect the design IP from untrusted entities (foundry, test facility, and end-user) in the IC supply chain. Logic locking accomplishes design IP protection by embedding key-controlled logic (key-gates) driven by an on-chip tamper-proof memory [4]. Applying the correct key (known to the designer) unlocks the chip resulting in the correct functionality, whereas the incorrect key results in an incorrect functionality. Researchers have developed a series of defenses [1,4]–[11] and attacks [12]–[25] over the last decade towards enhancing the security of logic locking. Most notably, Subramanyan *et al.* [13] proposed the *Boolean satisfiability* (SAT)-based attack, which broke all prior locking techniques. Researchers developed various SAT-resistant logic locking solutions (further details in Sec. II) to defend against the SAT-based attack. However, with each developed defense, new attack techniques exposed implementation vulnerabilities.

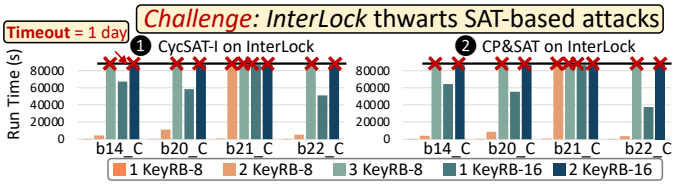

Fig. 1. The SAT-based CycSAT-I [14] and CP&SAT [1] attacks reach a timeout of one day on benchmarks locked using InterLock with different sizes of key-controlled routing blocks (KeyRBs) (based on the results in [1]).

In this work, we focus on one of the most prominent SAT-resistant techniques that thwarts the SAT-based attack [13] by constructing symmetric interconnection (routing obfuscation). Such an approach increases the depth of the SAT search tree, ensuring SAT-hard calls [10]. Although naïve routing obfuscation thwarts the SAT-based attack, it is vulnerable to re-modeling/encoding attacks [1,26]. Recently, Kamali *et al.* [1] proposed *InterLock* to mitigate the drawbacks of the prior locking techniques. In InterLock, key-controlled routing blocks (KeyRBs) perform routing and logic obfuscation, twisting logic with routing, thereby thwarting state-of-the-art attacks on logic locking. Next, we discuss the challenges as to why there has been no successful attack on InterLock.

A. Key Research Challenges Targeted in this Work

1) *SAT-hard calls:* InterLock ensures that any attack relying on SAT solvers (e.g., SAT-based attack, *AppSAT* [19]) encounters a complex SAT search tree. The authors in [1] launched the *cyclic-based SAT* (CycSAT-I [14]) (see ❶ in Fig. 1) and the *canonical prune and SAT* (CP&SAT [1]) (see ❷ in Fig. 1) attacks on locked benchmarks and demonstrated that both attacks fail to recover the secret key, running for a day without termination (timeout).
2) *Multiplexer (MUX)-based locking:* The construction of InterLock utilizes deep MUX trees for locking. In general, a MUX key-gate takes an original (true) wire and

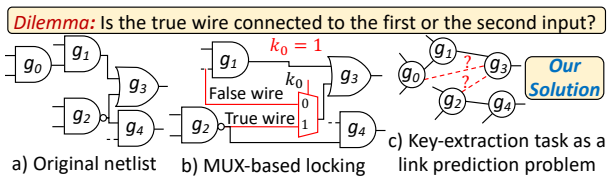

Fig. 2. We formulate the key-extraction task as a link prediction problem.

another (false) wire from the design. The select line of the MUX acts as the key-input. Applying the correct key-bit passes the true wire maintaining the original functionality. The correct key-bit can either be 0 or 1, depending on whether the true wire is connected to the first or the second input of the MUX. Hence, an attacker cannot infer the correct key-bit from the type of key-gate, unlike X(N)OR-based locking, which can be broken using machine learning (ML)-based structural attacks [22,27]. We illustrate an example of MUX-based locking and showcase the associated challenge in Fig. 2.

3) *Loop formation:* MUX-based locking may introduce combinational cycles in the locked design. Note that the SAT-based attack applies only to directed acyclic graphs (DAG) [13]. As a result, cycles trap the attack algorithm in infinite loops. Thus, authors in [1] use the CycSAT-I attack [14], which can decrypt cyclic logic encryption, to evaluate the security of InterLock. Although CycSAT-I can handle loops, it faces SAT-hard computations. Researchers formulated specific techniques, such as *SWEEP* and *SCOPE*, to tackle MUX-based locking [23,28]. However, both SWEEP and SCOPE cannot handle cycles in the design. To demonstrate this key limitation, we lock selected ITC-99 benchmarks using InterLock with 1 KeyRB-16.[1] We also lock selected ISCAS-85 and ITC-99 benchmarks using 2-input MUX-based locking with key-sizes (K) of $\{64, 128, 256, 512\}$, resulting in 24 locked designs. *We observe that both SWEEP and SCOPE attacks fail to decipher the keys due to the presence of loops in the locked designs.*[2]

B. Our Novel Concept and Contributions

In this work, we attack routing obfuscation, focusing on the rigorous InterLock technique. We showcase how an attacker can determine the hidden connections and gates using knowledge of the locked netlist structure (or utilizing a circuit library) *without relying on an oracle*. The intuition behind our work is that (i) modern ICs contain a large amount of repetition and reuse cores [30], and (ii) routing obfuscation introduces limited local structural changes in the locked design, which allows the attacker to learn the remaining (intact) structure of the locked design. To that end, we lock the ISCAS-85 benchmark c7552 with InterLock and visualize the locked design as a graph in Fig. 3. The KeyRB affects a restricted portion of the design (see ❶), leaving 99.45% of the original

[1]We provide further details about KeyRB construction in Sec. II.

[2]SWEEP and SCOPE rely on *ABC* [29] to convert a locked design into a DAG. When reading a design with combinational loops, the tool reports an error "Network contains a combinational loop" and cannot launch the attack.

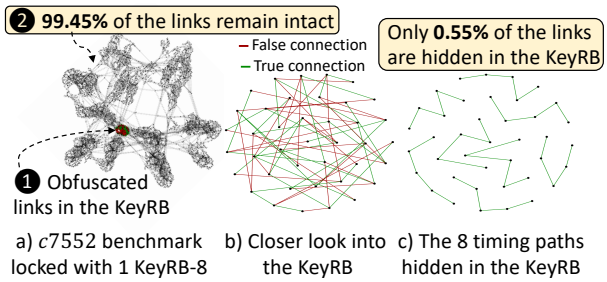

Fig. 3. ISCAS-85 benchmark c7552 locked using InterLock with one KeyRB-8 [1]. Only 0.55% of the links are obfuscated in the KeyRB.

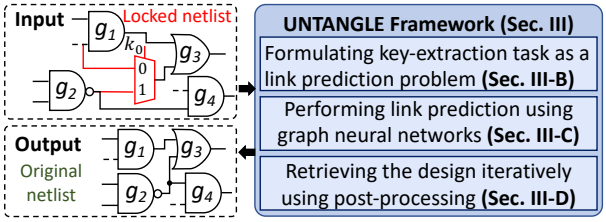

Fig. 4. An overview of our novel contributions.

connections untouched (see ❷).[3] Knowing which types of gates in a design are likely to be connected helps de-obfuscate the routing blocks. We propose UNTANGLE as a generic link prediction-based attack on MUX-based locking, using graph neural networks (GNNs), as shown in Fig. 2. The novel contributions of this work (see Fig. 4) are as follows.

1) **Formulating the key-extraction task as a link prediction problem:** We build a graph with edges based on the observable connections between gates outside the routing blocks. Then, using link prediction, we infer the links hidden in the blocks due to the routing obfuscation.
2) **Performing link prediction based on graph neural networks (GNNs):** Several heuristics exist for link prediction. In UNTANGLE, we are interested in learning the composition of gates in the network, i.e., the graph structure (connectivity) and node features (type of gates). Thus, we use a GNN model that exploits the structure of the design to learn link features. To that end, we extract local enclosing subgraphs around each considered link. The GNN takes in the enclosing subgraphs, utilizes the structure and gate features, and outputs vector embeddings that capture information about the target links and the composition of gates in the underlying design.
3) **Achieving certainty of the predicted key-bits:** We propose a novel post-processing algorithm to unlock the design. The algorithm examines the likelihood of each link (predicted by the GNN) and selects only a subset of the links, which are predicted to exist with extremely high confidence. The selected links are then added to the network (locked design), completing it iteratively. We recompute all the likelihoods for the remaining links and the procedure continues until the network is completed, retrieving the design with 100% precision.

[3]As the size and the number of KeyRBs increase, a larger portion of the design gets obfuscated. Yet, the majority of the connections remain accessible.

Key results: We perform an extensive experimental evaluation of UNTANGLE on selected ISCAS-85 and ITC-99 benchmarks locked using InterLock [1] and random MUX-based locking. UNTANGLE deciphers up to 100% and 99.61% of the key-bits, respectively, with a precision up to 100%. UNTANGLE can break the locked benchmarks, which the other state-of-the-art attacks fail to unlock. **We also open source UNTANGLE [2].**

II. Background and Related Work

A. SAT-based Attack [13] and Related Countermeasures

The SAT-based attack requires (i) a functional IC (with the correct key embedded) acting as an "oracle," and (ii) a locked reverse-engineered netlist. The attack starts by constructing a miter using two copies of the locked netlist. The miter is fed to a SAT solver to find a discriminating input pattern (DIP) for which at least two key assignments generate two different outputs. Subsequently, the DIP is fed to the oracle to prune out the invalid keys. This procedure repeats until the attack cannot determine more DIPs, resulting in the secret key. Researchers have developed a plethora of SAT-resistant techniques, which can be broadly categorized as follows.

1) **Point function-based obfuscation** [7,8,31] techniques force the SAT-based attack to rule out one incorrect key per iteration, thereby imposing an exponential number of DIPs (in terms of key-size) to unlock the design. However, these techniques are susceptible to various structural and functional attacks [18,21,32,33].
2) **Scan locking** [34,35] techniques obfuscate the scan data, limiting the controllability and observability of internal nets. Nevertheless, modeling attacks [16,36] have been successful in circumventing scan locking techniques.
3) **SAT-hard obfuscation** [1,10] techniques increase the execution time required for each SAT attack iteration by embedding key-controlled *SAT-hard* instances (KeyRBs) in the design. The KeyRBs perform routing obfuscation and are highly symmetric with different keys resulting in the same functionality (isomorphic solutions). These techniques are referred to as SAT-hard because symmetry is challenging for SAT solvers [10]. Nevertheless, routing obfuscation is not sufficient to ensure security. Modeling-based attacks can simplify the obfuscation using symmetry-breaking [26]. The state-of-the-art SAT-hard InterLock technique [1] performed both routing and logic obfuscation and was shown to be resistant to various state-of-the-art attacks, which we explain next.

B. InterLock–Intercorrelated Logic and Routing Locking [1]

InterLock is developed as an extension over the Full-Lock [10] technique. In both techniques, a KeyRB is constructed using MUX-based switch boxes (SwBs), as illustrated in Fig. 5(a). The KeyRB is a near non-blocking logarithmic network [37], which has N inputs, where N is a power of 2. The network is built using $2log_2(N) - 2$ stages, where each stage consists of $N/2$ SwBs. In Full-Lock, the SwBs are constructed using MUXes and inverters.[4] However, to avoid re-encoding attacks, InterLock embeds logic gates into the keyRB using the SwBs, as depicted in Fig. 5(b). Each SwB contains four MUXes and two logic gates $\{f_1, f_2\}$. The MUXes are controlled by a total of three key-inputs. The gates are constrained to be 2-input logic gates and are extracted from the original design. Each SwB has four inputs, $\{I_i, I_j, exI_i, exI_j\}$ and two outputs $\{O_i, O_j\}$. The exI inputs are connected to the circuitry outside the KeyRB. Depending on the key, outputs O_i and O_j could be $\{I_i, I_j, f_1(I_i, exI_i), f_1(I_j, exI_i)\}$ and $\{I_i, I_j, f_2(Ii, exI_j), f_2(I_j, exI_j)\}$, respectively.

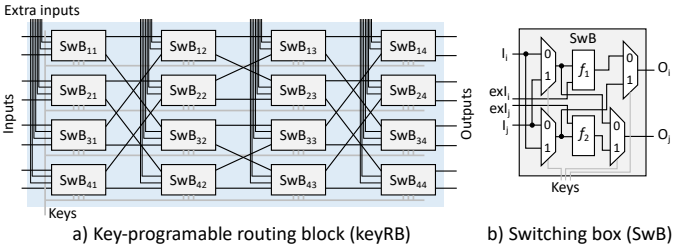

Fig. 5. KeyRB-8 in InterLock [1]. $\{f_1, f_2\}$ are 2-input gates from the circuit.

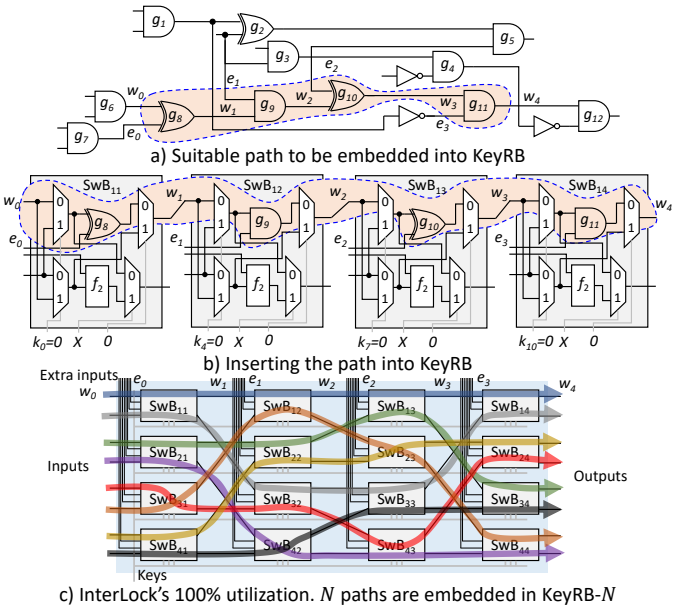

Fig. 6. Path embedding stage in InterLock (based on [1]).

InterLock searches for specific timing paths to incorporate into the KeyRB. The number of timing paths is the same as the number of inputs to the KeyRB (N), with a length equal to the number of stages in the block. We illustrate how a timing path is extracted from the original design and embedded into the network in Fig. 6. Upon applying the correct key, the outputs of each SwB resemble the fan-outs of the gates from the original design. The valid key maps the original fan-ins of the f_1 and f_2 gates to the inputs of the corresponding SwBs, i.e., the outputs from the previous SwB stage.

[4] UNTANGLE is also applicable to Full-Lock and other routing obfuscation methods. By (i) re-modeling the KeyRB in Full-Lock using the all-to-all edge-encoding in [26], then (ii) applying our link prediction model on the edges.

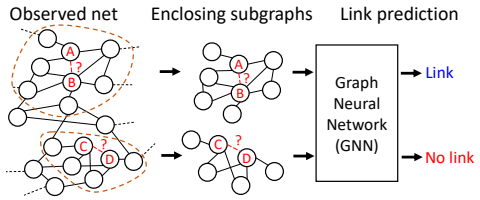

Fig. 7. Link prediction using graph neural networks (GNNs) (based on [38]).

C. Link Prediction Problem

We infer the timing paths in KeyRBs using link prediction. The underlying concept of link prediction is to estimate the likelihood of a link between two target nodes. This estimation is governed by the structure of the observed network and the attributes of the nodes [39]. Link prediction has a wide variety of applications, such as protein interaction prediction [40], friend recommendation in social networks [41], and drug response prediction [42]. Let $\mathcal{G} = (V, E, \boldsymbol{X})$ be an undirected graph, where $V = \{1, 2, \ldots, n\}$ is the set of n nodes, $E \subseteq V \times V$ is the set of observed edges, and $\boldsymbol{X} \in \mathbb{R}^{n \times k}$ is the matrix of node features. A row $\boldsymbol{X}_{i,:}$ denotes the feature vector of node i with length k. We denote the adjacency matrix of $\mathcal{G}$ as $\boldsymbol{A} \in \{0,1\}^{n \times n}$, where $\boldsymbol{A}_{i,j} = 1$ iff $(i,j) \in E$. Let U indicates the universal set of all possible connections between vertices in the network, then $|U| = \frac{|V|(|V|-1)}{2}$. We represent the missing links as $T = U - E$. A link prediction algorithm assigns a score to all links in T based on some computed heuristics. If the score for a link is greater than a specific threshold value, then the link is predicted to exist. Recently, GNNs have shown tremendous success in performing link prediction, exploiting both the structure of the graph and the associated node features to extract link features, surpassing the performance of traditional methods [38].

D. Graph Neural Networks (GNNs)

A GNN generates a vector representation (embedding) for each node in the graph such that similar nodes are placed together in the embedding space. The embedding of a target node v gets updated through message passing (neighborhood aggregation). The features of the neighboring nodes $\mathcal{N}(v)$ are accumulated to generate an aggregated representation. The aggregated information is then combined with the features of the target node to update its embedding. Consequently, after L rounds of message passing, each node is aware of its features, the features of the neighboring nodes, and the structure of the graph within the L-hop neighborhood. The message passing phase is abstracted as follows, where $\boldsymbol{z}_v^{(l)}$ indicates the embedding of node v at the l-th round.

$$\boldsymbol{a}_v^{(l)} = AGG^{(l)}\left(\left\{\boldsymbol{z}_u^{(l-1)} : u \in \mathcal{N}(v)\right\}\right) \quad (1)$$

$$\boldsymbol{z}_v^{(l)} = UPDATE^{(l)}\left(\boldsymbol{z}_v^{(l-1)}, \boldsymbol{a}_v^{(l)}\right) \quad (2)$$

GNNs mainly differ based on the choices of the $AGG(\cdot)$ and $UPDATE(\cdot)$ functions. In our work, we extract a subgraph around each target link. The extracted subgraphs hold information about the circuitry surrounding the link. Therefore, by performing graph classification, the label of the target link also becomes the label of its corresponding subgraph, as shown in Fig. 7. To obtain a graph-level representation, a global pooling is applied over the node embeddings.

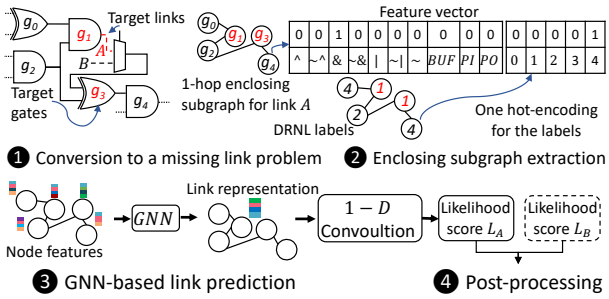

Fig. 8. The different steps of the proposed UNTANGLE framework.

III. Proposed UNTANGLE Attack

In this section, we provide an overview of the main steps of UNTANGLE attack (Fig. 8) and discuss the steps in detail.

A. Attacker Model

We assume an *oracle-less* setting where *only* the locked netlist is available. An attacker can obtain the locked netlist by reverse-engineering the GDSII in the untrusted foundry. The attacker can determine the location of the key-gates by tracing the key-inputs from the tamper-proof memory.

B. Formulating Key-extraction as a Link Prediction Problem

The resiliency of routing obfuscation comes from the complex connections introduced in the KeyRBs. We *untangle* the twisted network and consider the KeyRBs as gates with missing connections, as demonstrated in Fig. 9. InterLock [1] utilizes 100% of the KeyRB to enhance the resilience against re-encoding attacks and to minimize overheads. However, *we identify a vulnerability in this implementation*, which we describe next. The utilization of 100% indicates that each 2-input gate in a KeyRB is extracted from the original design, and therefore, cannot be skipped upon applying the correct key. As a result, outputs O_i and O_j are now restricted to $\{f_1(I_i, exI_i), f_1(I_j, exI_i)\}$, and $\{f_2(Ii, exI_j), f_2(I_j, exI_j)\}$, respectively, allowing us to infer the keys of the two independent MUXes, as illustrated in Fig. 9(b). Due to the removal of the last two MUXes (see Fig. 9(c)), we now consider a total of four possible links for each SwB, as shown in Fig. 9(d). Two of the links are correct (green) and two are incorrect (red), as shown in Fig. 10(a). The next step is to obtain the likelihood score for each link and identify the true links.

C. Link Prediction Based on Graph Neural Networks

1) Subgraph Extraction: We construct an undirected graph $\mathcal{G}$ based on the observable edges outside the routing block. Nodes in the graph map to the gates in the locked design. We assign a one-hot encoded feature vector to each node which captures the Boolean functionality. Additionally, the feature vector highlights if a gate has a link to a primary input (PI) or a primary output (PO). The length of the feature vector depends

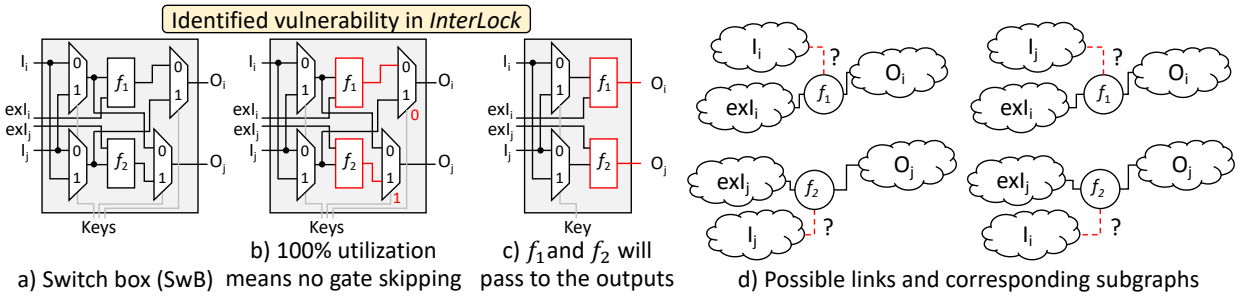

Fig. 9. Modeling routing de-obfuscation task as a link prediction problem.

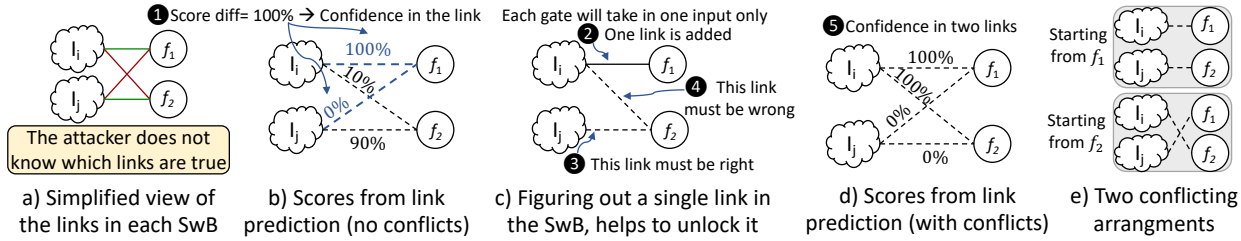

Fig. 10. Processing the outputs of link prediction in UNTANGLE (post-processing).

on the number of Boolean functions available in the target technology library. We use a GNN-based platform for the link prediction task [38]. The target nodes are grouped into set S. E.g., if we want to predict the likelihood of a link between nodes u, v, then $S = \{u, v\}$. Given $(S, \mathcal{G})$, an *h-hop enclosing subgraph* $\mathcal{G}_{(S,h)}$ is extracted around each pair of target nodes. Let $d(u, v)$ denote the shortest path distance between vertices u and v, then $\mathcal{G}_{(S,h)}$ is induced from $\mathcal{G}$ by $\cup_{v \in S}\{u \mid d(u, v) \leq h\}$. Please refer ❷ in Fig. 8 for an example of 1-hop subgraph extraction.

2) Node Labeling: We employ the double radius node labeling (DRNL) used in [38] to maximize the GNN's link representation power. Each node in the extracted subgraph is given a label to capture its relationship with the target link. These labels are used as additional node attributes, which are one-hot encoded and combined with the original features, as demonstrated by ❷ in Fig. 8. The target nodes are always given the unique label 1 so that the GNN distinguishes them from the rest of the subgraph. Let u and v be the target nodes, the DRNL label $f_l(i)$ of a node i is calculated as follows:

$$f_l(i) = 1 + \min(d_u, d_v) + (d/2)[(d/2) + (d\%2) - 1] \quad (3)$$

where $d_u := d(i, u)$, $d_v := d(i, v)$, and $d := d_u + d_v$. In the case when $d(i, u) = \infty$ or $d(i, v) = \infty$, then $f_l(i) = 0$. This happens if node i is only connected to one of the target nodes. The *labeling trick* is what makes graph classification suitable for link prediction. Instead of only learning the node features, the GNN is now aware of the target link and the relationship of the surrounding circuitry with it.

3) GNN Model: UNTANGLE is flexible with the type of GNN to use. We use the deep graph convolutional neural network (DGCNN) [43], which achieves superior results in graph classification. A graph convolutional layer is as follows:

$$\boldsymbol{Z}^{l+1} = f(\tilde{\boldsymbol{D}}^{-1}\tilde{\boldsymbol{A}}\boldsymbol{Z}^{l}\boldsymbol{W}^{l}) \quad (4)$$

where $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ adds self loops to allow self aggregation. $\tilde{\boldsymbol{D}}$ is the diagonal degree matrix, where $\tilde{\boldsymbol{D}}_{ii} = \sum_j \tilde{\boldsymbol{A}}_{i,j}$, and $\boldsymbol{W}^l \in \mathbb{R}^{k_l \times k_{l+1}}$ is a trainable weight matrix. $f(.)$ is an element-wise non-linear activation function. $\boldsymbol{Z}^l \in \mathbb{R}^{n \times k_l}$ is the output embedding of layer $l-1$. The initial embeddings are the node features $\boldsymbol{Z}^0 = \boldsymbol{X}$. The first step in the convolutional layer is $\boldsymbol{Z}^l\boldsymbol{W}^l$, which performs a linear feature transformation on node information, mapping the k_l feature channels to k_{l+1} channels. The second step aggregates the node information to neighboring vertices, including the node itself. Then $\tilde{\boldsymbol{D}}$ normalizes the aggregated information to ensure a fixed feature scale. Multiple convolutional layers can be employed to extract multi-scale sub-structure features from the network. After L layers, the output embeddings from each layer $l = 1, \ldots, L$ are concatenated horizontally, to capture the graph in a single output vector $\boldsymbol{Z}^{1:L} := [\boldsymbol{Z}^1, \ldots, \boldsymbol{Z}^L]$, where $\boldsymbol{Z}^{1:L} \in \mathbb{R}^{n \times \sum_{l=1}^{L} k_l}$. A *sort pool layer* takes in the $n \times \sum_{l=1}^{L} k_l$ tensor $\boldsymbol{Z}^{1:L}$ and sorts it row-wise according to $\boldsymbol{Z}^L$. The final tensor is reshaped to $c(\sum_{l=1}^{L} k_l) \times 1$, selecting c nodes to represent the graph. Then, the final embedding is fed to 1-D convolutional layers with filter and step size of $\sum_{l=1}^{L} k_l$ to classify the graph.

D. Post-processing

Random MUX-based locking considers the location of each MUX independently. Therefore, the corresponding missing links can be processed individually. We compare the likelihood scores of the links associated with a single MUX, and the link with the highest score gets predicted as the true wire. In the case of a tie, the corresponding key-bit will be left undeciphered. On the other hand, in InterLock, the obfuscated links are close together in the network. Through our experiments, we conclude that completing the network iteratively enhances the performance of the attack on InterLock. Each link prediction step adds more links to the network, aiding in constructing meaningful enclosing subgraphs to predict remaining links.

We describe the UNTANGLE post-processing approach for breaking InterLock in Algorithm 1. As discussed in Sec. III-B, four links are considered for each SwB, where two links $\{l_a, l_b\}$ are associated with each logic gate $\{f_1, f_2\}$. Let T denote the set of all the considered links. The link prediction platform assigns a probability score L_l for each link $l \in T$ (lines 58-64). In lines 22-24, the links are considered pairwise $\{l_a, l_b\}$, as shown in Fig. 10(b). The model looks for a pair $\{l_a, l_b\}$ having $L_a \geq up \;||\; L_b \geq up$ and $|L_a - L_b| \geq th$, where th and up are adjustable threshold and upper limit, respectively. E.g., in ❶ in Fig. 10, $up = 1$ and $th = 1$. Hence, the model selects one link with high confidence (see ❷). Figuring out one link in an SwB enables the model to obtain the remaining connections (see ❸ and ❹). Let C and R denote the sets of chosen and rejected links, respectively. C is added to the list of predicted links P (line 26) and R is removed from T (line 27). In case of a conflict (see ❺), the model gets two conflicting decisions for an SwB. In this scenario (line 29), the post-processing restarts with an adjusted th (lines 30-32). If the maximum $th = up = 1$ is reached and there is still a conflict, an average ensemble of the network at two h sizes $\{2, 3\}$ (lines 35-39) is used and the likelihoods are recomputed (lines 10-12). The GNN is not retrained for $h = 3$ as using the same model (trained for $h = 2$) is sufficient. Considering two-hop sizes together results in a robust model and prevents misclassifications. After selecting a set of links with high confidence, the links are removed from T, added to the network (lines 44-47), and link prediction is performed again. The th and up values (lines 49-53) are adjusted if no links are predicted. Finally, the original design is recovered once T is empty (lines 54-55).

E. Setup and Dataset Generation

1) Self-referencing Scenario: We train the GNN based on extracted data from the target locked design without relying on a circuit library. This setup does not require re-locking to be performed by the attacker. We use all non-obfuscated links in the locked design to create the "positive" training samples. Following the typical manner of learning-based link prediction, we randomly sample the same number of nonexistent links (unconnected node pairs) and use them as "negative" training data. We keep all the obfuscated links for testing.[5]

2) Circuit Library-based Scenario: The GNN is trained based on extracted data from a circuit library. The designs in the library are locked using the targeted locking technique. The library does not include the target design but includes circuits with a similar global design structure. Note that the Interlock technique hides specific parts of the design in the KeyRBs, leaving most of the design intact. We argue that a foundry with access to a library of various designs could readily identify/guess the high-level modules/functionality in the to-be-attacked design and construct a circuit library. The training samples in this scenario additionally include the obfuscated links in the library. We add the true obfuscated links to the

[5]By default, the target testing links do not appear in their corresponding enclosing subgraphs (because they are missing links). Hence, when extracting the samples for training, we remove the target training links from their enclosing subgraphs so that the GNN does not over-fit the training data, predicting testing links as negative because the target link does not exist [38].

Algorithm 1 Pseudo-code for the proposed post-processing

Input: Locked netlist graph ($\mathcal{G}$), List of target links (T), List of obfuscated gates (G), and GNN model
Output: Secret key (K)

1: $Done \leftarrow FALSE$
2: $th \leftarrow 0$ ▷ Initialize threshold
3: $up \leftarrow 1$ ▷ Initialize upper limit
4: $h \leftarrow 2$ ▷ Initialize enclosing subgraph h-hop distance
5: $Ensemble \leftarrow FALSE$
6: **Restart:**
7: **while** !$Done$ **do**
8: $L \leftarrow \{\emptyset\}$ ▷ Likelihood scores
9: $F \leftarrow \{\emptyset\}$ ▷ Scores after averaging ensemble
10: **if** $Ensemble$ **then**
11: **for** $h \in (2, 3)$ **do**
12: $L[:, h-2]$ =GET_PREDICTIONS(h, T)
13: **else**
14: $L[:, 0]$ =GET_PREDICTIONS(h, T)
15: **for** row_number in $L.length()$ **do**
16: $F.append(mean(L[row\_number,]))$
17: $L \leftarrow F$
18: **Restart-I:**
19: $P \leftarrow \{\emptyset\}$ ▷ Links to add in the network
20: $T_r \leftarrow \{\emptyset\}$ ▷ Links to remove from the network
21: $G_r \leftarrow \{\emptyset\}$ ▷ Unlocked gates
22: **for** $gate \in G$ **do**
23: **if** $L_a \geq up \;||\; L_b \geq up$ **then**
24: **if** $|L_a - L_b| \geq th$ **then**
25: **if** No conflict **then**
26: $P.append(C)$ ▷ Add chosen links
27: $T_r.append(R)$ ▷ rejected links
28: $G_r.append(gate)$
29: **else**
30: **if** $h = 2$ && $th \neq up$ **then**
31: $th \leftarrow th + 0.1$
32: go to Restart-I
33: **else if** $Ensemble$ **then**
34: $Done \leftarrow TRUE$
35: **else**
36: $Ensemble \leftarrow TRUE$ ▷ Activate ensemble
37: $th \leftarrow 1$
38: $up \leftarrow 1$
39: go to Restart ▷ Compute the likelihoods again
40: **if** !IsEmpty(P) **then**
41: **if** $h = 2$ **then**
42: $h \leftarrow 3$
43: $th \leftarrow 1$
44: $\mathcal{G}.add(P)$ ▷ Add predicted links to the network
45: $T.remove(P)$ ▷ Remove predicted links from the target links
46: $T.remove(T_r)$ ▷ Remove rejected links from the target links
47: $G.remove(G_r)$
48: **else**
49: **if** $th \geq \frac{up}{2}$ **then**
50: $th \leftarrow th - 0.1$
51: **else**
52: $up \leftarrow up - 0.1$
53: $th \leftarrow up$
54: **if** IsEmpty(T) **then**
55: $Done \leftarrow TRUE$
56: $K \leftarrow$ GET_KEY$(\mathcal{G})$ ▷ Infer K from the updated network
57: **return** K ▷ Key
58: **procedure** GET_PREDICTIONS(h, T)
59: $Temp \leftarrow \{\overline{\emptyset}\}$
60: **for** $link \in T$ **do**
61: $S \leftarrow (u, v)$ ▷ Target gates
62: $\mathcal{G}_{(S,h)} \leftarrow$SAMPLE$(\mathcal{G}, h, S)$ ▷ Get $\mathcal{G}_{(S,h)}$ with h-hop sampling
63: $Temp.append($GNN$(\mathcal{G}_{(S,h)}))$ ▷ Get the predictions
64: **return** $Temp$

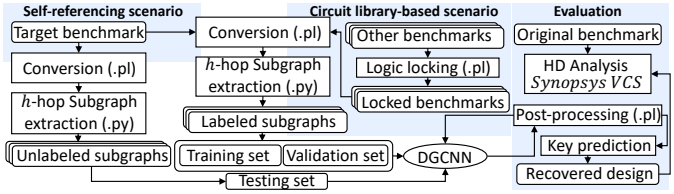

Fig. 11. Experimental setup and tool flow.

positive training samples, while the false obfuscated links are added to the negative samples.

IV. Experiments

A. Evaluation Setup, Tool Flow, and Evaluation Metrics

We summarize the experimental setup in Fig. 11. We evaluate UNTANGLE on selected ISCAS-85 and ITC-99 benchmarks locked using InterLock and random MUX-based locking. We implement the scripts for locking and circuit to graph conversion in *Perl*. We use the *PyTorch* implementation of SEAL/DGCNN [38] for link prediction, using four GNN layers with 32, 32, 32, and 1 output channels, respectively. For the sort pooling layer, we set c such that 60% of the subgraphs have vertices less than c. We use two 1-D convolution layers, with 16 and 32 output channels and a dense layer of 128 neurons for classification. Regarding the hop size h, it is stated that the performance saturates after $h \geq 3$ [38]. Thus, we train the GNN using 2-hop subgraphs for 50 epochs. We use the model with the best validation performance to predict the testing links. We perform the experiments on an Intel(R) Xeon(R) CPU $X5680$ with $64GB$ of RAM.

1) Dataset Generation for Random MUX-based Locking: We insert the MUXes randomly in the designs and also randomize the selection of false wires for locking. We lock each ISCAS-85 benchmark with $K : \{64, 128, 256\}$ and each ITC-99 benchmark with $K : \{256, 512\}$. We follow the self-referencing scenario (see Sec. III-E1) for random MUX-based locking. A feature vector of length 10 is associated with each node. We further extend the feature vector by the DRNL labels.

2) Dataset Generation for InterLock: Similar to [1], we only lock the ITC-99 benchmarks (in *BENCH* format). Note that as the KeyRB size increases, it embeds a larger portion of the design. This is why it is challenging to find suitable paths to embed in a KeyRB for small designs from the ISCAS-85 benchmark suite. We consider designs in *BENCH* format to satisfy the restrictions of the locking technique. The timing paths which are to be embedded in the KeyRBs must include 2-input gates only. We encountered challenges in meeting this requirement while handling Verilog netlists. Hence, to ensure a fair implementation, we follow the same setup as outlined in [1] and adhere to the *BENCH* format.

For InterLock, we observe better performance when using the circuit library-based scenario for dataset generation (see Sec. III-E2). The obfuscated links in InterLock are highly correlated. Therefore, when extracting random links from the remaining non-obfuscated network for training, such interference between missing links does not get captured in training. The designs in the library are locked using the same KeyRB instances n and size N as the target benchmark. We lock each ITC-99 benchmark with $\{1, 2, 3\}$ KeyRBs of sizes : $\{8, 16\}$.

TABLE I. UNTANGLE ON BENCHMARKS LOCKED USING INTERLOCK WITH 1 KEYRB-8

Benchmark	N	Attack Iteration	th	up	h	C	W	Prec.	Links Recovered	Links Left	Total Solved Key-bits
b22_C	8	1	0	1	2	24	0	100%	24	8	**46/48**
		2	0.9	1	3	4	0	100%	**28**	**4**	
b21_C		1	0	1	2	14	0	100%	14	18	**46/48**
		2	0.9	1	3	2	0	100%	16	16	
		3	1	1	3	2	0	100%	18	14	
		4	0.9	1	3	6	0	100%	24	8	
		5	0.9	1	3	2	0	100%	26	6	
		6	0.9	1	3	2	0	100%	**28**	**4**	
b20_C		1	0.1	1	2	22	0	100%	22	10	**48/48**
		2	1	1	3	8	0	100%	30	2	
		3	0	1	3	2	0	100%	**32**	**0**	
b14_C		1	0	1	2	18	0	100%	18	14	**45/48**
		2	1	1	3	8	0	100%	**26**	**6**	

3) Evaluation Methods and Metrics: In a circuit library-based scenario, UNTANGLE attacks each design independently by excluding its links from training/validation. We report the number of correct link decisions C, the wrong link decisions W, the number of deciphered keys, and the precision. We report the Hamming distance (HD) between the outputs of the original design and the outputs of the recovered design by UNTANGLE. For the key-bits that remain unresolved by UNTANGLE, we compute the HD as follows. For each design, we choose 100 random keys and compare the outputs of the recovered design with the golden outputs (original design) by applying $10,000$ random input patterns using Synopsys VCS.

B. Breaking InterLock [1] Using UNTANGLE

In all the cases, UNTANGLE achieves 100% precision, implying that it always makes correct decisions when adding links to the networks. We report the results of attacking the benchmarks locked using 1 KeyRB-8 in Table I. On average, UNTANGLE deciphers 95.83% of the key (46 out of 48 key-bits) in an average of 2 post-processing runs (see Algorithm 1), leaving only two key-bits unresolved per design.[6]

1) Effect of the Number of KeyRBs: Next, we study the effect of increasing the number of KeyRBs n, used for locking, on the performance of UNTANGLE. We report the results of the attack on benchmarks locked using $n : \{2, 3\}$ KeyRB-8 in Table II. *The results demonstrate that UNTANGLE maintains the same performance regardless of n.* UNTANGLE recovers up to 97.92% $(94/96)$ and 98.61% $(142/144)$ of the key-bits for $n = 2$ and 3, respectively. However, with the increase in the number of missing connections, the total number of post-processing runs increases. For example, the average number of post-processing runs required for $n = \{2, 3\}$, is $\{7, 10\}$.

2) Effect of KeyRB Size: Increasing the KeyRB size has a minor effect on the performance of UNTANGLE. The average percentage of deciphered keys drops from 96.35% to 89.24%, when 1 KeyRB-16 is used, compared to the case of KeyRB-8. The number of target links triples (from 32 to 96) and the missing links are all correlated. Nevertheless, UNTANGLE deciphers up to 94.44% $(136/144)$, 93.75% $(270/288)$, and 96.1% $(416/432)$ of the key, for KeyRB-16 with $n = 1$, 2, and 3, respectively, with 100% precision (see Table II).

[6]The unresolved key-bits are left for brute-force attack or SAT-based attack.

TABLE II. UNTANGLE ON BENCHMARKS LOCKED USING INTERLOCK WITH n KEYRBS

Benchmark	N	n	Attack Iterations	W	Prec.	Links Recovered	Links Left	Total Solved Key-bits
b22_C	8	2	5	0	100%	60	4	**94/96**
b21_C			6	0	100%	56	8	**92/96**
b20_C			8	0	100%	60	4	**94/96**
b14_C			7	0	100%	32	32	**80/96**
b22_C		3	6	0	100%	92	4	**142/144**
b21_C			7	0	100%	84	12	**138/144**
b20_C			18	0	100%	90	6	**141/144**
b14_C			8	0	100%	84	12	**138/144**
b22_C	16	1	11	0	100%	68	28	**130/144**
b21_C			16	0	100%	78	18	**135/144**
b20_C			15	0	100%	80	16	**136/144**
b14_C			13	0	100%	34	62	**113/144**
b22_C		2	12	0	100%	154	38	**269/288**
b21_C			8	0	100%	154	38	**269/288**
b20_C			9	0	100%	156	36	**270/288**
b14_C			9	0	100%	152	40	**268/288**
b22_C		3	11	0	100%	256	32	**416/432**
b21_C			10	0	100%	250	38	**413/432**
b20_C			9	0	100%	232	56	**404/432**
b14_C			14	0	100%	238	50	**407/432**

TABLE III. UNTANGLE ON RANDOM MUX-BASED LOCKING

Benchmark	K	Correct keys	Wrong keys	Undeciphered keys	Prec.	Acc.
c7552	64	57	1	6	98.44%	89.06%
	128	111	5	12	96.09%	86.72%
	256	224	8	24	96.88%	87.50%
c5315	64	61	0	3	100%	95.31%
	128	114	2	12	98.44%	89.06%
	256	228	8	20	96.88%	89.06%
c3540	64	59	2	3	96.88%	92.19%
	128	113	7	8	94.53%	88.28%
	256	225	14	17	94.53%	87.89%
c2670	64	50	3	11	95.31%	78.13%
	128	109	9	10	92.97%	85.16%
	256	212	16	28	93.75%	82.81%
b22_C	256	237	2	17	99.22%	92.58%
	512	477	2	33	99.61%	93.16%
b21_C	256	238	4	14	98.44%	92.97%
	512	466	6	40	98.83%	91.02%
b20_C	256	235	2	19	99.22%	91.80%
	512	471	11	30	97.85%	91.99%
b14_C	256	232	10	14	96.09%	90.63%
	512	459	15	38	97.07%	89.65%

C. Breaking Random MUX-based Locking Using UNTANGLE

For MUX-based locking, we run a single attack run because the MUXes are independent, and thus, predicting the link for a specific key-gate does not help in unlocking the rest of the key-gates. We report the accuracy and precision values in Table III. UNTANGLE deciphers up to 95.31% of the keys with a precision up to 100%, demonstrating the generic nature of our attack. *We also launch SWEEP [23] and SCOPE [28] on these locked designs—the attacks fail to recover any key-bit due to the existence of loops.* Increasing the key-size makes the observed design more incomplete due to the obfuscated (missing) connections, which impacts the performance of UNTANGLE. For example, the accuracy drops from 89.06% to 87.5% when attacking c7552 locked with a key-size of 64 and 256, respectively. Note that a larger key-size also leads to higher overheads (area, power, and timing).

D. HD of Designs Reconstructed by UNTANGLE

We report the HD values in Table IV. UNTANGLE achieves an average HD of 0.0015% and 0.013% when recovering designs locked with one KeyRB of size 8 and 16, respectively. The HD values indicate that UNTANGLE almost obtains the exact functionality of the design *without an oracle*. A subsequent oracle-guided attack can be carried out if an attacker desires an exact functionality (HD=0). Note that since UNTANGLE resolves the SAT-hard instances, the SAT-based attack will not encounter SAT-hard calls. To that end, we launch the SAT-based attack and decipher the remaining key-bits in 8 DIPs (on average).

TABLE IV. HAMMING DISTANCE (HD) OF DESIGNS RECONSTRUCTED BY UNTANGLE

Benchmark	HD for Recovered Benchmarks	
	1 KeyRB-8	1 KeyRB-16
b22_C	0.001%	0.009%
b21_C	0.004%	0.007%
b20_C	0%	0.01%
b14_C	0.001%	0.026%

E. UNTANGLE Run Time

The average run time for a single post-processing run of UNTANGLE on b14_C, b20_C, b21_C, and b22_C, locked using the most challenging case of 3 KeyRB-16 is 0.32, 0.34, 0.44, and 0.53 seconds, respectively. The SAT-based attack runs for a day without termination on the same locked benchmarks.

V. DISCUSSION

Comparison with Attacks: *NNgSAT* [17] leverages a neural network to solve SAT-hard instances in methods such as Full-Lock [10]. However, NNgSAT does not apply to InterLock and it requires an oracle. The *topology guided-attack* [44] is a dictionary-based rule learning attack that leverages the composition of gates in a circuit. In contrast to our approach, dictionary-based attacks cannot predict the key-bit value of a key-gate if its exact surrounding circuitry is not listed in the dictionary since it is based on exact matching. However, in UNTANGLE, we use a GNN to learn the composition of gates, which can handle variants naturally.

Possible Countermeasure: UNTANGLE is successful because (i) the effects of locking are limited and local, and (ii) the surrounding circuitry of a MUX key-gate is not obfuscated. A logic locking solution, which obfuscates the global structure of the design, is required. Large-scale MUX-based locking could be one way to get security at the expense of increasing overheads, which we plan to study in detail as part of future work. Furthermore, one way to improve the InterLock technique is to ensure that the KeyRBs inserted in a design are connected to each other, and the connections are obfuscated.

VI. CONCLUSION

In this work, we present UNTANGLE, a generic link prediction-based attack on MUX-based locking that can break the state-of-the-art SAT-hard locking technique, InterLock. We formulate the key-extraction task in MUX-based locking as a link prediction problem, leverage a graph neural network to learn the composition of gates in the locked netlist or a circuit library, and extract link features that assist in performing link prediction. We demonstrate that UNTANGLE can break SAT-resistant MUX-based locking (by resolving the SAT-hard instances) with precision up to 100% in an oracle-less setting, which is a first in the literature. We believe that UNTANGLE highlights the need for logic locking techniques that obfuscate the global structure of the design, as opposed to limited and local structural changes.

REFERENCES

[1] H. M. Kamali *et al.*, "InterLock: An intercorrelated logic and routing locking," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020, pp. 1–9.

[2] https://github.com/lilasrahis/untangle.

[3] M. Rostami *et al.*, "A Primer on Hardware Security: Models, Methods, and Metrics," *Proc. of the IEEE*, vol. 102, no. 8, pp. 1283–1295, 2014.

[4] J. Roy *et al.*, "Ending Piracy of Integrated Circuits," *IEEE Computer*, vol. 43, no. 10, pp. 30–38, 2010.

[5] J. Rajendran *et al.*, "Fault Analysis-Based Logic Encryption," *IEEE Computer*, vol. 64, no. 2, pp. 410–424, 2015.

[6] K. Shamsi *et al.*, "Cyclic obfuscation for creating SAT-unresolvable circuits," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*, ser. GLSVLSI '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 173–178. [Online]. Available: https://doi.org/10.1145/3060403.3060458

[7] M. Yasin *et al.*, "SARLock: SAT Attack Resistant Logic Locking," in *IEEE HOST*, 2016, pp. 236–241.

[8] ——, "Provably-Secure Logic Locking: From Theory To Practice," in *ACM/SIGSAC CCS*, 2017, pp. 1601–1618.

[9] Y. Xie *et al.*, "Delay locking: Security enhancement of logic locking against ic counterfeiting and overproduction," in *Proceedings of the 54th Annual Design Automation Conference 2017*, ser. DAC '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3061639.3062226

[10] H. M. Kamali *et al.*, "Full-lock: Hard distributions of sat instances for obfuscating circuits using fully configurable logic and routing blocks," in *Design Automation Conference (DAC)*, 2019, pp. 1–6.

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

[12] J. Rajendran *et al.*, "Security Analysis of Logic Obfuscation," in *IEEE/ACM Design Automation Conference*, 2012, pp. 83–89.

[13] P. Subramanyan *et al.*, "Evaluating the Security of Logic Encryption Algorithms," in *IEEE HOST*, 2015, pp. 137–143.

[14] H. Zhou *et al.*, "CycSAT: SAT-based attack on cyclic logic encryptions," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 49–56.

[15] L. Alrahis *et al.*, "ScanSAT: Unlocking obfuscated scan chains," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, ser. ASPDAC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 352–357. [Online]. Available: https://doi.org/10.1145/3287624.3287693

[16] ——, "ScanSAT: Unlocking static and dynamic scan obfuscation," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2019.

[17] K. Z. Azar *et al.*, "NNgSAT: Neural network guided SAT attack on logic locked complex structures," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020, pp. 1–9.

[18] L. Alrahis *et al.*, "GNNUnlock: Graph neural networks-based oracle-less unlocking scheme for provably secure logic locking," in *IEEE/ACM Design, Automation and Test in Europe Conference*, 2021, pp. 780–785.

[19] K. Shamsi *et al.*, "AppSAT: Approximately Deobfuscating Integrated Circuits," in *IEEE HOST*, 2017, pp. 95–100.

[20] K. Z. Azar *et al.*, "SMT attack: Next generation attack on obfuscated circuits with capabilities and performance beyond the SAT attacks," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2019, no. 1, pp. 97–122, Nov. 2018. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/7335

[21] M. Yasin *et al.*, "Removal attacks on logic locking and camouflaging techniques," *IEEE Transactions on Emerging Topics in Computing*, vol. 99, no. 0, p. PP, 2017.

[22] P. Chakraborty *et al.*, "SAIL: Machine learning guided structural analysis attack on hardware obfuscation," in *AsianHOST*, 2018, pp. 56–61.

[23] A. Alaql *et al.*, "Sweep to the secret: A constant propagation attack on logic locking," in *AsianHOST*, 2019, pp. 1–6.

[24] L. Li *et al.*, "Piercing logic locking keys through redundancy identification," in *DATE*, 2019, pp. 540–545.

[25] L. Alrahis *et al.*, "Functional Reverse Engineering on SAT-Attack Resilient Logic Locking," in *IEEE ISCAS*. IEEE, 2019, pp. 1–5.

[26] J. Sweeney *et al.*, "Modeling techniques for logic locking," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020, pp. 1–9.

[27] D. Sisejkovic *et al.*, "Challenging the security of logic locking schemes in the era of deep learning: A Neuroevolutionary approach," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 17, no. 3, pp. 1–26, 2021.

[28] A. Alaql *et al.*, "SCOPE: Synthesis-based constant propagation attack on logic locking," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 8, pp. 1529–1542, 2021.

[29] R. Brayton *et al.*, "ABC: An Academic Industrial-strength Verification Tool," in *International Conference on Computer Aided Verification*. Springer, 2010, pp. 24–40.

[30] D. Saha *et al.*, "SoC: a real platform for IP reuse, IP infringement, and IP protection," *VLSI Design*, vol. 2011, 2011.

[31] Y. Xie *et al.*, "Mitigating SAT attack on Logic Locking," in *CHES*. Springer, 2016, pp. 127–146.

[32] F. Yang *et al.*, "Stripped Functionality Logic Locking With Hamming Distance-Based Restore Unit (SFLL-hd)–Unlocked," *IEEE TIFS*, vol. 14, no. 10, pp. 2778–2786, 2019. [Online]. Available: https://github.com/Yangff/sfll_re

[33] D. Sirone *et al.*, "Functional Analysis Attacks on Logic Locking," *IEEE TIFS*, vol. 15, pp. 2514–2527, 2020. [Online]. Available: https://bitbucket.org/spramod/fall-attacks/src/master/

[34] R. Karmakar *et al.*, "Encrypt flip-flop: A novel logic encryption technique for sequential circuits," *arXiv preprint arXiv:1801.04961*, 2018.

[35] X. Wang *et al.*, "Secure scan and test using obfuscation throughout supply chain," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2017.

[36] N. Limaye *et al.*, "DynUnlock: Unlocking scan chains obfuscated using dynamic keys," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020, pp. 270–273.

[37] D.-J. Shyy *et al.*, "Log/sub 2/ (n, m, p) strictly nonblocking networks," *IEEE Transactions on Communications*, vol. 39, no. 10, pp. 1502–1510, 1991.

[38] M. Zhang *et al.*, "Link prediction based on graph neural networks," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18. Red Hook, NY, USA: Curran Associates Inc., 2018, p. 5171–5181.

[39] D. Liben-Nowell *et al.*, "The link-prediction problem for social networks," *Journal of the American society for information science and technology*, vol. 58, no. 7, pp. 1019–1031, 2007.

[40] Y. Qi *et al.*, "Evaluation of different biological data and computational classification methods for use in protein interaction prediction," *Proteins: Structure, Function, and Bioinformatics*, vol. 63, no. 3, pp. 490–500, 2006.

[41] L. A. Adamic *et al.*, "Friends and neighbors on the web," *Social networks*, vol. 25, no. 3, pp. 211–230, 2003.

[42] Z. Stanfield *et al.*, "Drug response prediction as a link prediction problem," *Scientific reports*, vol. 7, no. 1, pp. 1–13, 2017.

[43] M. Zhang *et al.*, "An end-to-end deep learning architecture for graph classification," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[44] Y. Zhang *et al.*, "TGA: An oracle-less and topology-guided attack on logic locking," in *Proceedings of the 3rd ACM Workshop on Attacks and Solutions in Hardware Security Workshop*, 2019, pp. 75–83.