INSIGHT: Attacking Industry-Adopted Learning Resilient Logic Locking Techniques Using Explainable Graph Neural Network

Lakshmi Likhitha Mankali, *New York University;* Ozgur Sinanoglu, *New York University Abu Dhabi;* Satwik Patnaik, *University of Delaware*

https://www.usenix.org/conference/usenixsecurity24/presentation/mankali

This paper is included in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

INSIGHT: Attacking Industry-Adopted Learning Resilient Logic Locking Techniques Using Explainable Graph Neural Network

Lakshmi Likhitha Mankali
New York University
lm4344@nyu.edu

Ozgur Sinanoglu
New York University Abu Dhabi
os22@nyu.edu

Satwik Patnaik
University of Delaware
satwik@udel.edu

Abstract

Logic locking is a hardware-based solution that protects against hardware intellectual property (IP) piracy. With the advent of powerful machine learning (ML)-based attacks, in the last 5 years, researchers have developed several learning resilient locking techniques claiming superior security guarantees. However, these security guarantees are the result of evaluation against existing ML-based attacks having critical limitations, including (i) black-box operation, *i.e.,* does not provide any explanations, (ii) are not practical, *i.e.,* non-consideration of approaches followed by the semiconductor industry, and (iii) are not broadly applicable, *i.e.,* evaluate the security of a specific logic locking technique.

In this work, we question the security provided by learning resilient locking techniques by developing an attack (INSIGHT) using an explainable graph neural network (GNN). INSIGHT recovers the secret key without requiring scan-access, *i.e.,* in an oracle-less setting for 7 unbroken learning resilient locking techniques, including **2 industry-adopted logic locking techniques**. INSIGHT achieves an average key-prediction accuracy (KPA) of 2.87×, 1.75×, and 1.67× higher than existing ML-based attacks. We demonstrate the efficacy of INSIGHT by evaluating locked designs ranging from widely used academic suites (ISCAS-85, ITC-99) to larger designs, such as `MIPS`, Google `IBEX`, and `mor1kx` processors. We perform 2 practical case studies: (i) recovering secret keys of locking techniques used in a widely used commercial EDA tool (`Synopsys TestMAX`) and (ii) showcasing the ramifications of leaking the secret key for an image processing application. We will open-source our artifacts to foster research on developing learning resilient locking techniques.

1 Introduction

1.1 Security Threats in IC Supply Chain

Integrated circuits (ICs) serve as the workhorse of all electronic systems, from smartphones to artificial intelligence systems. The demand for high-performing systems has catalyzed technological scaling, making transistors miniaturized. As a result, manufacturing costs for ICs have increased substantially. For instance, the manufacturing costs for TSMC (a leading contract IC manufacturer) have surged 10×, jumping from four billion dollars (22nm node) [1] to 40 billion dollars (2nm node) [2]. Escalating manufacturing costs have forced leading chip (semiconductor) design companies (*e.g.,* Apple, NVIDIA) to transition to a globalized IC supply chain model. While chips are designed in-house, the fabrication, testing, and packaging of ICs are outsourced to off-shore and geographically disparate foundries, which could be (i) potentially untrustworthy and (ii) prone to geopolitical conflicts [3].

A U.S. government report revealed that the U.S. relies heavily on East Asia, which handles 75% of the world's IC production [4]. Meanwhile, the U.S. *only* manufactures 10% of the ICs, indicating a significant security threat. Several security concerns emanate due to the globalized landscape of manufacturing chips, such as hardware intellectual property (IP) piracy [5–10], reverse engineering [11, 12], IC counterfeiting [13, 14], and insertion of hardware Trojans [15–17]. In this work, we focus on the problem of hardware IP piracy.

1.2 Hardware IP Piracy

Hardware IP piracy poses a financial burden for IC design companies, causing losses of billions of dollars. A U.S. Department of Justice report indicates that one IC design company lost around \$8.75 billion due to IP theft [18]. Furthermore, IP theft by nation-state adversaries has significant ramifications on national security. For example, in 2009, attackers gained unauthorized access to sensitive information on the F-35 fighter jet, compromising its advanced capabilities and highlighting the risks of hardware IP piracy [19].

Researchers have proposed countermeasures against hardware IP piracy—three prominent examples are logic locking [20], IC camouflaging [11], and split manufacturing [21]. Logic locking protects the hardware IP from all the untrustworthy entities in the IC supply chain [22], *i.e.,* foundry, test-

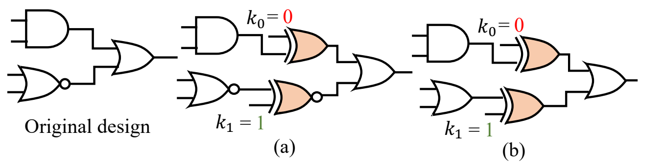


Figure 1: (a) Logic locking and (b) TRLL-locked design [24].

ing, and end-user, unlike IC camouflaging and split manufacturing. Logic locking has attracted attention from academia, government, and commercial entities. For instance, the Defense Advanced Research Projects Agency (DARPA) is investing in several initiatives [23] to thwart hardware IP piracy.

1.3 Logic Locking for Hardware IP Protection

Logic locking modifies the original design by inserting additional logic, *i.e., key-gates* controlled by a secret key (Figure 1(a)). The secret key is stored in a tamper-proof memory and programmed by a trusted facility (*e.g.,* an IC design house) *after* the fabrication and testing of ICs. Inserting key-gates results in additional inputs, *i.e., key-inputs*. The locked design functions correctly after applying the correct key, while an incorrect key produces corrupted outputs. *Logic locking is analogous to a password protection mechanism protecting the hardware IP using a password, i.e., a secret key.*

DARPA is proactively funding the research and development of secure logic locking techniques through programs such as the Automatic Implementation of Secure Silicon (AISS) [23], Structured Array Hardware for Automatically Realized Applications (SAHARA) [25], and Efficient Cross-Layered IP Protection Scheme (ECLIPSE) [26]. Besides, leading electronic design automation companies such as Synopsys and Mentor Graphics have integrated logic locking in their tools, *i.e.,* `TestMAX` [27] and `TrustChain` platform [28].

Although there have been 300+ papers in logic locking [22, 29], over the last 5 years, researchers have focused on developing locking techniques resilient to machine learning (ML)-based attacks, *which is also the scope of our work.*

1.4 ML-based Attacks on Logic Locking

ML-based attacks identify and exploit the structural and functional hints in the hardware implementation of locked designs to recover the secret key [30–34]. These attacks use different approaches, such as unsupervised learning [32], graph neural networks (GNNs) [31, 33], and random forest [30].
Limitations of Existing Attacks. Existing ML-based attacks: (i) operate as a black-box, *i.e.,* do not provide any explanations regarding the failure of attacks, (ii) do not consider industry-adopted locking techniques, and (iii) are not broadly applicable, *i.e.,* evaluate the security (predict the secret key) of a given locking technique and do not apply to different locking techniques. These limitations also compromise the evaluation methodology and heuristics to develop and benchmark learning resilient locking techniques. In our work, we question the security of learning resilient locking techniques by overcoming the aforementioned limitations.

1.5 Our Goals and Contributions

We assume the role of a practical attacker and aim to recover the secret key from several unbroken and industrially adopted locking techniques. These techniques involve various algorithms, such as (i) XOR/XNOR-based locking [24, 35], (ii) adversarial learning [36], (iii) multiplexer (MUX)-based locking [37], and (iv) adversarial sample generation [38]. Hence, there is a need to develop an attack that is broadly applicable and recovers the secret key by evaluating the hardware implementation of learning resilient locking techniques.

However, numerous obstacles exist to developing an attack that has broad applicability. **First**, as an attacker, we only have access to the underlying locked design with Boolean logic gates and wires; it is computationally challenging to examine each substructure in the locked design and ascertain the role played by the substructure in recovering the secret key. **Second**, the locking techniques considered in this work follow different algorithms to thwart learning-based attacks; developing a broadly applicable attack encompassing several techniques is challenging. **Finally**, it is challenging to obtain suitable/relevant explanations and use them to enhance the efficacy of the attack in recovering the secret key.

We address the abovementioned obstacles and develop a practical attack, INSIGHT, using explainable ML. Researchers have used explainable ML for malware detection [39–41], ML security [42], and network security [43], and explainability has provided insights into the decision-making process of ML models, making their outputs understandable to humans. Furthermore, GNNs are suitable for Boolean logic designs since circuits are best represented as graphs [44, 45]. Therefore, we utilize GNNs for our work and improve our attack significantly by using explainable GNNs (or explainers).

However, utilizing explainers to develop an attack is not straightforward, and several challenges exist in developing a practical attack: ① selecting a suitable explainer, *i.e.,* choosing an appropriate explainer for recovering the secret key of locked designs, ② identifying vulnerabilities, *i.e.,* finding exploitable hints (structural/functional), ③ attacking practical designs, *i.e.,* evaluating the security of logic-synthesized designs, ④ tackling unique graph representations, *i.e.,* recovering secret from non-XOR/XNOR/MUX key-gates, and ⑤ insufficient training datasets, *i.e.,* attacking locked designs having limited (or no) training samples.

We overcome challenge ① of selecting a suitable explainer by evaluating state-of-the-art explainers (§4.3). We overcome challenge ② of identifying structural vulnerabilities (hints) in locked designs by analyzing GNN explanations and modifying our problem modeling (§4.4). For challenge ③ of attacking practical locked designs, we use the explanations from the explainer and improve the architecture of the GNN by incorporating attention (§4.5). To enable the attack for ④ unique graph representations, *i.e.,* different key-gates, we perform local and global graph transformations and modify the attack pipeline to account for other key-gates (§4.5). Finally, we perform data augmentation and follow semi-supervised learning to attack locked designs with ⑤ insufficient training datasets (§4.6). Our primary **contributions** are as follows.

- We develop an explainability-guided GNN-based attack (INSIGHT) that recovers secret keys from the hardware implementation of 7 unbroken locking techniques. To the best of our knowledge, our work is the **first** to use explainable GNN to develop a successful attack in semiconductor supply chain security (§4).
- We showcase the efficacy and applicability of INSIGHT on 7 unbroken locking techniques. These techniques are published at premier EDA and security venues and are yet unbroken. In fact, **2 of the locking techniques are integrated into `Synopsys TestMAX`**, a widely used EDA tool in the semiconductor industry. INSIGHT achieves an improved accuracy of 1.75×, 2.87×, and 1.67× compared to existing open-source attacks OMLA, SCOPE, and MuxLink (§5).
- We experiment with designs from the widely used ISCAS-85, ITC-99, and EPFL academic suites to open-source designs such as the Stanford `MIPS` processor [46] (≈23k gates), Google `IBEX` processor [47] (≈18k gates), DARPA Common Evaluation Platform `GPS` [48] (≈193k gates), and `mor1kx` processor [49] (≈158k gates). We experiment with different key-sizes and synthesis settings, demonstrating scalability and broad applicability (§5).
- We perform 2 practical case studies. We showcase the efficacy of INSIGHT on designs locked using `Synopsys TestMAX` [50] (§5.5) and illustrate the ramifications of secret key leakage for an image processing application (§5.6).
- We mention the findings provided by INSIGHT (§5.4) and discuss potential countermeasures that IC designers can utilize to strengthen the locking techniques (§6.3).

2 Background and Preliminaries

We initially describe machine learning (ML)-attack resilient logic locking techniques (§2.1). Next, we discuss existing ML-based attacks on logic locking (§2.2) and their limitations (§2.3). Finally, we briefly discuss graph neural networks (GNNs), explainable GNNs, and key definitions (§2.4).

2.1 ML-attack Resilient Logic Locking

Researchers proposed various locking techniques in response to ML-based attacks (§1.4). These techniques: (i) eliminate the correlation between key-value and key-gate, (ii) do not depend on logic synthesis-based transformations, and (iii) replace inverters in the design with key-gates (XOR/XNOR/-MUX). For our work, we chose 7 unbroken locking techniques, including **2 industry-adopted techniques.**
UNSAIL [38] inserts key-gates to mitigate ML-based attacks that exploit hints from the design structure to resolve structural transformations caused by logic synthesis [30]. **TRLL** [24] inserts XOR/XNOR-based key-gates randomly, ensuring no correlation between the type of key-gate and the key-bit (Figure 1(b)). **TRLL+** [35] improves `TRLL` for designs with limited inverters and minimizes the number of correct keys. **TroMUX** [37] randomly inserts MUX key-gates to thwart ML-based attacks [32, 33]. **RGLock** [51] replaces XOR/XNOR gates with modified XOR/XNOR gates. **ALMOST** [36] uses adversarial learning, and **SimLL** [52] inserts MUX key-gates between logic gates, showcasing functional similarity to thwart ML-based attacks.

Apart from being unbroken, the above-mentioned locking techniques claim at least one of the following properties: (i) <=50% key-prediction accuracy (equivalent to random guess) against ML-based attacks, (ii) do not depend on logic synthesis-based transformations to improve security, and (iii) do not reveal the secret key through the presence of structural and functional hints.

Researchers evaluate the security of the aforementioned locking techniques using **key-prediction accuracy (KPA)**. KPA is defined as the percentage of correctly recovered key-bits, *i.e.,* $KPA = |Kc|/|K| * 100$, where $|Kc|$ is the number of correct key-bits and $|K|$ is the key-size. We use the **KPA metric** to evaluate the efficacy of INSIGHT (§5.2 and §5.3).

2.2 Existing ML Attacks on Logic Locking

SAIL [30] is an ML-based attack on X(N)OR-based locking. It converts the locked design to a vector format consisting of the structural information (connectivity between logic gates) and gate-types of the logic gates around the key-gate. SAIL uses a neural network to predict if the key-gates have undergone changes due to logic synthesis and restores any changes that might have occurred. Finally, SAIL predicts the secret key-value upon reverting the post-synthesis modification. **OMLA** [31] leverages GNN-based subgraph classification to recover the secret key-value in X(N)OR-based locking. It converts the locked design to a graph and extracts subgraphs around the key-gate. Next, OMLA uses GNN to exploit the structural and functional hints from the extracted subgraphs to predict the secret key. **SnapShot** [34] uses a convolutional neural network (CNN) to recover the secret key in X(N)OR-based locking. It converts the locked design to a vector format and extracts structural features (*e.g.,* path depth, fan-out) from the locality around key-gates. Based on the extracted features and locality, CNN predicts the secret key. **SCOPE** [32] uses unsupervised learning and exploits hints (*e.g.,* area) to recover the secret key in multiplexer (MUX)-based locking. It exploits the changes in the locked design when a single key-bit value is hard-coded and predicts each key-input (one at a time) based

Table 1: Distinctions and advantages of INSIGHT compared to existing ML/GNN-based attacks.

Attack	Task	Attacks industry-adopted locking techniques	Key-gate types	Explainer insights	Adaptability to new techniques	Applicable to logic-synthesized designs
SAIL [30] (2021)	Neural network-based classification	✗	X(N)OR/MUX	✗	✗	✓
SnapShot [34] (2021)	CNN-based classification	✗	X(N)OR	✗	✗	✓
OMLA [31] (2021)	GNN-based subgraph classification	✗	X(N)OR	✗	✗	✓
SCOPE [32] (2021)	Unsupervised learning	✗	MUX	✗	✗	✓
MuxLink [33] (2022)	GNN-based link prediction	✗	MUX	✗	✗	✗
INSIGHT (This Work)	Explainability-guided GNN-based node classification	✓	X(N)OR/MUX/ (N)AND/(N)OR	✓	✓	✓

on the difference of synthesis-based hints when the key is 0 and 1. **MuxLink** [33] leverages GNN-based link prediction to recover the secret key in MUX-based locking. It extracts the subgraph (neighborhood) around the input nets of MUX key-gates, considering the correct (incorrect) connection as a positive (negative) link. It extracts structural and functional hints from the subgraph and uses GNN to perform binary classification on the extracted subgraph to predict the correct connections (or secret key).

2.3 Limitations of Existing Attacks

Existing ML-based attacks lack in the following aspects. **Firstly,** the aforementioned attacks operate as a black-box, *i.e.,* provide only KPA as a result and do not provide explanations regarding the success/failure of the attack. As a result, these attacks cannot be strategically improved to evaluate the security of new/unbroken locking techniques. For instance, OMLA obtains a KPA of 49.97% for the unbroken locking techniques. In contrast, INSIGHT utilizes explainable GNN to improve the attack strategically achieving a KPA of 87.30% for the aforementioned locking techniques.

Secondly, existing ML-based attacks do not evaluate industry-adopted locking techniques (`TRLL` [24] and `TRLL+` [35]). These techniques are integrated into an industry-adopted and widely used semiconductor EDA tool, `Synopsys TestMAX`. In contrast, INSIGHT achieves a KPA of 88.76% on `TRLL` [24] and `TRLL+` [35] locking techniques.

Finally, existing ML-based attacks are not broadly applicable to different locking techniques. The attacks are proposed to (i) evaluate the security of either 1 or 2 locking techniques and (ii) apply to either X(N)OR-based locking or MUX-based locking techniques. Unlike existing attacks, INSIGHT recovers secret keys in 7 unbroken logic locking techniques consisting of 7 different types of key-gates, *i.e.,* X(N)OR/(N)AND/(N)OR and MUX (Table 1). Moreover, INSIGHT evaluates the locking techniques that utilize different key-gate insertion algorithms such as (i) random XOR/XNOR locking [24, 35], (ii) adversarial learning [36], (iii) random MUX-based locking [37], (iv) adversarial sample generation [38] and (v) functionality similar locking [52].

2.4 (Explainable) Graph Neural Networks

GNNs perform inference (predictions) on graph-structured data and demonstrate significant performance in applications such as learning patterns in protein interactions [53], recommender systems [54, 55], and computer security [56, 57]. GNNs can (i) handle irregular and structured data and (ii) incorporate topological information from the graph into the learning process. GNNs are used for (i) node classification, (ii) link prediction, and (iii) graph classification. **Explainable GNNs** make predictions of the model transparent and interpretable by providing human-understandable explanations. They explain predictions through the importance of features, nodes, and edges in the graph. Explanations can help to build trust in the model and facilitate the adoption of GNNs in real-world applications. Next, we provide some definitions.

Substructures (or subgraphs) are the local neighborhood around to-be-classified nodes. **Subgraph extraction** extracts the local neighborhood of unlabeled to-be-classified nodes. **Node classification** classifies the unknown nodes based on the features of the nodes in their local neighborhood. **Neighborhood-based node classification** uses subgraphs around to-be-classified nodes for classification instead of individual nodes. It captures structural information from the local neighborhood of the unlabeled node to predict node labels.

3 Threat Model

Our work considers an oracle-less threat model widely used in the logic locking literature [5–10, 22]. Next, we discuss the notations and outline the attacker's location, resources available to the attacker, and the attacker's goal.

Notations. A combinational design $\mathcal{D}_{orig}$ is represented as a directed acyclic graph with x primary inputs (PI) and y primary outputs (PO), where $PI = \{0,1\}^x$ and $PO = \{0,1\}^y$. $\mathcal{D}_{orig}$ implements the Boolean function $F_{orig} : PI \rightarrow PO$. $\mathcal{D}_{orig}$ is locked using a locking technique $\mathcal{L}$ with a secret key K, resulting in a locked design $\mathcal{D}_{lock}$ consisting of additional key-inputs $KI = \{KI_0, KI_1, ..., KI_{|K|-1}\}$, where $KI = \{0,1\}^{|K|}$ and $|K|$ is the cardinality of KI, *i.e.,* key-size. The locked design $\mathcal{D}_{lock}$ implements the Boolean function $F_{lock} : PI \times KI \rightarrow O$.

Note that $\mathcal{D}_{lock}$ is functionally equivalent to $\mathcal{D}_{orig}$ only upon loading the correct secret key ($\mathtt{K}$), *i.e.,* $F_{orig}(j) = F_{lock}(j, \mathtt{K})$, $\forall j \in PI$. $\mathcal{D}_{lock}$ is converted to a Graphics Database System II (GDSII)[*] represented as $\mathcal{D}^{GDS}_{lock}$ and sent to a potentially untrustworthy foundry for fabrication. The foundry converts $\mathcal{D}^{GDS}_{lock}$ to a chip $\mathbb{D}_{lock}$. $\mathbb{D}_{lock}$ is tested, packaged, and sent to a trustworthy facility (*e.g.,* design house), which loads the correct key in the tamper-proof memory and obtains an activated chip $\mathbb{D}_{orc}$. The activated chip is also known as an oracle in the logic-locking community. Additionally, $\mathcal{A}^{\mathbb{T}}$ represents an attacker $\mathcal{A}$ using technique $\mathbb{T}$ to recover the secret key ($\mathtt{K}$).

Attacker location. We assume that the attacker $\mathcal{A}$ is located in an untrustworthy foundry. A foundry could be potentially untrustworthy since (i) most chip design companies (*e.g.,* Apple, NVIDIA) outsource the fabrication of chips to off-shore foundries [58], and (ii) the U.S. relies heavily on overseas foundries for its manufacturing needs [4, 59] representing a pernicious security threat (§1.1,§1.2).

Attacker resources. The resources available are as follows.

- Attacker $\mathcal{A}$ has access to reverse-engineering tools [60] and can reverse-engineer GDSII of a locked design ($\mathcal{D}^{GDS}_{lock}$) to extract the reverse-engineered gate-level netlist ($\mathcal{D}_{lock}$).
- We consider an oracle-less threat model, *i.e.,* the attacker has no access to the functionally activated chip/oracle ($\mathbb{D}_{orc}$) for the following reasons.
  - Firstly, having simultaneous access to GDSII ($\mathcal{D}^{GDS}_{lock}$) and the activated chip/oracle ($\mathbb{D}_{orc}$) may not be practical. This limitation arises because an attacker is located in an untrustworthy foundry, but $\mathbb{D}_{orc}$ is available only after the fabrication, testing, and activation of the chip (performed by loading the correct key onto the memory).
  - Secondly, we assume scan-chain[†] protection techniques in $\mathcal{D}_{lock}$; these techniques (*e.g.,* DisORC [24], scan-chain locking [61]) either disable or protect the scan-chain access. For example, DisORC [24] corrupts the outputs of an oracle ($\mathbb{D}_{orc}$) whenever a potential attack utilizing a scan chain is detected. Since the outputs of $\mathbb{D}_{orc}$ are corrupted, the attacker can no longer rely on an oracle to obtain correct input/output pairs, thus preventing (restricting) the oracle-guided attacks. Scan-chain locking techniques [61] lock selected flip-flops to restrict the controllability and observability of the scan-chain. Restricting the controllability and observability of the scan-chain hinders an attacker's ability to access and control the internal states of the scan flip-flops. Consequently, scan-chain locking restricts oracle-guided attacks.
  - Thirdly, the oracle-guided threat model is less challenging than the oracle-less threat model, especially since there is no access to $\mathbb{D}_{orc}$. If chips are manufactured for military purposes [62], it is unlikely that an attacker in the foundry can access an activated chip/oracle ($\mathbb{D}_{orc}$).
  - Finally, we consider the same threat model (oracle-less) as the targeted locking techniques (`TRLL` [24], `TRLL+` [35], `UNSAIL` [38], `RGLock` [51], `TroMUX` [37], `ALMOST` [36] and `SimLL` [52]). These techniques mitigate attacks that do not have access to an oracle.
- Attacker $\mathcal{A}$ knows the locking technique implemented in the locked design. In addition, $\mathcal{A}$ can (i) identify key-inputs (KIs) and (ii) detect key-gates in the reverse-engineered locked design $\mathcal{D}_{lock}$. To identify KIs, $\mathcal{A}$ can trace the connections from the tamper-proof memory (TPM) to the chip, as the secret key is stored in the TPM. Furthermore, $\mathcal{A}$ can detect the key-gates by identifying the logic gates connected to key-inputs. These assumptions are consistent with *Kerckhoff's* principle, which asserts that the security of a system should rely exclusively on the secrecy of the secret key, $\mathtt{K}$, and not on the secrecy of other system aspects [63].

Attacker goal. The goal of attacker $\mathcal{A}$ is to recover the secret key $\mathtt{K}$ from the GDSII of a locked design, obtaining the recovered design $\mathcal{D}_{rec}$ and enabling IP piracy [5–10, 22].

4 INSIGHT

We explain why we use explainable machine learning (ML) to develop an attack (§4.1), followed by the problem modeling (§4.2). However, using explainers is not straightforward, and we face various challenges. We elaborate on these challenges and outline our solutions to overcome them (§4.3–§4.6). Finally, we discuss the overall architecture of INSIGHT (§4.7).

4.1 Why Explainable ML/GNN?

Recovering the secret key from unbroken locking techniques entails identifying structural and functional hints from the locked design. To develop a practical attack, an attacker requires feedback. Existing attacks utilize ML (GNN) models as a black-box and (i) perform feature engineering, (ii) tune the model parameters, and (iii) improve the model architecture based on human interpretation. However, a systematic approach requires understanding the inner workings of the trained ML model and making informed decisions. To that end, using explainable ML in the attack pipeline could aid an attacker in improving the attack systematically. INSIGHT uses explainable GNN to understand the reasons behind the incorrect predictions of the attack on targeted locking techniques. Explainable GNN provides explanations of the important nodes and features influencing the predictions of GNN-based node classification. Using the explanations obtained by explainable GNN, we determine the important nodes/features that help improve our attack. Without an explainable GNN, it would be difficult to ascertain the factors (important features and nodes in the graph) that lead to incorrect predictions of the attack (as it requires an exhaustive search among features and nodes around the key-gate).

[*]A database file used to represent IC layout information.

[†]Scan-chain access converts flip-flops into pseudo-primary inputs and outputs, enabling the attacker to access and control the internal states. Assuming scan chain access represents the best-case scenario for an attacker.

4.2 Problem Modeling

Problem Definition. Given a locked design D_{lock} having key-size $|K|$, INSIGHT aims to recover the secret key K of targeted locking techniques using explainable GNN.

Preliminary Problem Modeling. We map the problem of key prediction to GNN-based node classification. Node classification aims to classify the *key-gates* or *key-inputs* (KI_i) to either a key-value of $\{0\}$ or $\{1\}$. Recall that GNN-based node classification classifies unknown nodes based on the features of the nodes in their local neighborhood; therefore, we perform neighborhood-based node classification (§2.4). Next, we discuss the 4 steps in detail.

(1) Convert to Graph. The locked design D_{lock} is converted to a graph $G=(V,E)$ with nodes V and edges E, where nodes (edges) represent the Boolean logic gates (wires). $G=(V,E)$ is provided as an input to subgraph extraction.

(2) Subgraph Extraction. Subgraphs are extracted around all the key-gates with key-inputs $\{KI_0, KI_1, .., KI_{|K|-1}\}$ for r hop-size, where r is user-defined and design specific. Consider an example of extracting the subgraph around key-gate KI_i in graph G with nodes $\{u_0, u_1, ...u_N\}$, where N is the number of nodes in G. The subgraph extraction step returns a subgraph $G'(V',E')$ where any node $v \in V'$ is at a hop-size of r (or less). After extracting their features, the subgraphs around key-gates are provided to GNN for node classification.

(3) Feature Extraction. We extract the following features for the nodes in subgraphs: (i) gate type, (ii) input degree (inp), (iii) distance of the node from the key-gate (DE), (iv) a (+/-) sign indicating if a node is located in the fan-in/fan-out of the key-gate (sign), (v) subgraph size (SS), (vi) output degree (out), and (vii) truth table of logic gates (TT). Feature (i) and (vii) captures the functionality of the node, while (ii), (iii), (iv), (v), and (vi) capture the structure of the nodes in the subgraph. The feature for subgraph G' is given as X', where x'_v corresponds to the feature of node $v \in V'$.

(4) Node classification aims to predict the key-value for all the key-inputs (key-gates), similar to OMLA. GNN classifies the subgraphs extracted around key-gates. For predicting the key-value for G', *i.e.,* the subgraph extracted around KI_i, the GNN returns p_i, *i.e.,* its corresponding class as: $p_i = g(G', X,' \sigma)$ where $p_i \in \{0,1\}$, X' is the feature vector of G', g corresponds to GNN, and σ are trainable parameters of GNN (weights and bias). The key-value of KI_i is equal to p_i.

GNN. We leverage state-of-the-art GNN, *i.e.,* graph isomorphism network (GIN) [64], which is widely used and performs best for graph classification. Here, GNN performs neighborhood-based node classification for subgraphs extracted around the key-gates.

The preliminary problem modeling does not provide explanations that aid in (i) finding structural hints and (ii) improving the attack for targeted (unbroken) locking techniques.

Challenges. The challenges toward developing an explainability-guided GNN-based attack are (i) selecting a suitable explainer and (ii) inferring the explainer results to identify structural hints for targeted locking techniques. Next, we explain the challenges and our solutions in detail.

4.3 Selecting A Suitable Explainer

Since our attack aims to recover the secret key from the targeted locking techniques, it is necessary to understand the reasons behind the failure of existing ML-based attacks to identify the structural hints that aid in improving the attack.

Challenge ①. An explainable GNN must (i) be computationally efficient, *i.e.,* runtime should not be prohibitively large, and (ii) provide relevant explanations. Selecting a suitable explainer with these characteristics poses a key challenge.

Solution ①. To address challenge ①, we consider explainers that have demonstrated effectiveness in node classification tasks. We evaluate 4 state-of-the-art explainable GNNs—GNNExplainer [65], SubgraphX [66], ZORRO [67], and PGExplainer [68] renowned for their effectiveness in node classification tasks [69]. GNNExplainer and ZORRO provide explanations for **features and nodes** in the neighborhood of a given node, whereas PGExplainer provides explanations for **only nodes** in the neighborhood of a given node. SubgraphX provides information regarding the important subgraph, *i.e.,* the neighborhood of a given node. We provide the preliminary problem modeling (trained on `TRLL`-locked designs) as input to the explainers. We evaluate the relevancy of important features obtained from the explanations by temporarily removing the top feature (important nodes) from the dataset to observe the change in the model predictions. A significant change in predictions indicates the high importance of features (nodes).

Runtime. GNNExplainer and PGExplainer exhibit better computational efficiency than others. For example, the runtime of SubgraphX ($\approx$ 60 min) is 600$\times$ higher than GNNExplainer (6 s) and 1200$\times$ higher than PGExplainer when generating explanations for subgraphs with an average of 35 nodes. ZORRO has an acceptable runtime, *i.e.,* 58 s. Hence, we do not consider SubgraphX due to its large runtime.

Relevant feature explanations. Only GNNExplainer and ZORRO provide explanations for features. These explainers provide the importance scores for the features of preliminary problem modeling. The top feature returned by GNNExplainer is gate-type, and ZORRO is DE. Removing the gate-type feature reduces the key-prediction accuracy (KPA) of the preliminary model from 50.78% to 46.09%. Removing the DE feature reduces the KPA to 50.78%. These results indicate that removing the gate-type feature affects KPA more than removing DE, indicating higher importance for the gate-type feature. Hence, we do not consider ZORRO due to its non-relevant feature explanations.

Relevant node explanations. We evaluate the relevancy of nodes for GNNExplainer and PGExplainer. They both provide the importance scores of nodes in the subgraph around the key-gate. We consider the nodes with higher importance scores as

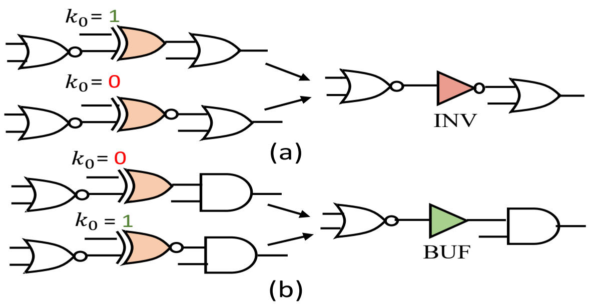


Figure 2: Mapping key-prediction problem to (a) inverter (INV) and (b) buffer (BUF) prediction problem.

important nodes. The KPA of preliminary problem modeling decreases from 54.68% to 46.87% for GNNExplainer and to 51.56% for PGExplainer by removing the important nodes returned by the corresponding explainers. Results indicate that removing important nodes returned by GNNExplainer affects KPA more than the important nodes returned by PG-Explainer. Hence, the important node explanations returned by GNNExplainer are more relevant.

In conclusion, GNNExplainer is a more suitable explainer for INSIGHT due to its (i) computational efficiency and (ii) ability to provide relevant explanations. Consequently, we utilize GNNExplainer in our work.[‡]

4.4 Identifying Structural Hints

Challenge ②. GNNExplainer provides explanations for extracted subgraphs of the targeted locking techniques (§4.3). The preliminary problem modeling obtains a KPA of 50.52%. The next challenge is identifying the reasons behind the failure of the preliminary problem modeling (§4.2). Understanding the reasons can help in improving the performance (KPA) of the attack. Next, we analyze the explainer results.

Failure of Preliminary Problem Modeling. We analyze the explainer results and observe that the preliminary problem modeling (§4.2) trains the same substructure (neighborhood around key-gate) for two key-values (classes) (Figure 2).

Example 1. Consider Figure 2(a); even though the logic gates in the neighborhood of the key-gates are identical, the key-input values are different. Such training samples create ambiguity for the ML model, resulting in lower KPA.

Identification of Common Structural Hints. Example 2. Consider Figure 2(a) for the `TRLL`-locked design; even though the key-input values for both substructures are different, both key-gates transform into an inverter upon applying the correct key, *i.e.,* XOR key-gate with key-input of "1" and XNOR key-gate with key-input of "0." Thus, if a GNN model were to train both designs with the same neighborhood, *i.e.,* substructure around the key-gate for inverter prediction instead of key-values, the ambiguity of the GNN model would be discarded.

[‡]Please refer to Appendix A.1 for further details on GNNExplainer.

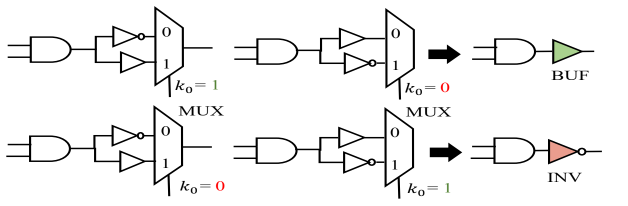


Figure 3: Example of transforming key-prediction for `TroMUX` [37] to inverter/buffer prediction problem.

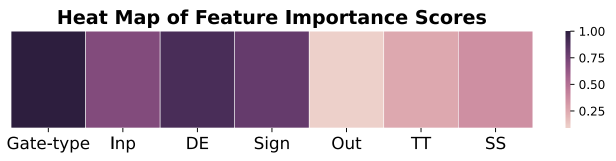


Figure 4: Heat map of importance scores of features of INSIGHT, *i.e.,* gate-type, inp, sign, out, TT, and SS.

The same is true for Figure 2(b), where both the substructures can be trained for buffer prediction.

In `TRLL` [24], `TRLL+` [35], `UNSAIL` [38], and `ALMOST` [36], the key-gates map to either a buffer (inverter) based on key-input and key-gates (XOR/XNOR). In `TroMUX` [37], an inverted and non-inverted logic of the locked wire is fed to the MUX-based key-gate. This construction can also be converted to an inverter/buffer prediction problem (Figure 3). The same is also true for `RGLock` [51]. Thus, recovering the key in the aforementioned locking techniques can be mapped to an inverter/buffer prediction problem.

Solution ②. To address challenge ②, we propose advanced problem modeling that classifies the *key-gates* in the locked design as an inverter (INV) or buffer (BUF). We map the inverter/buffer prediction to the GNN-based node classification problem as follows. The first two steps, *i.e.,* (1) conversion of locked design to graph and (2) extraction of the subgraph, remain the same (§4.2); however, the feature extraction and modified node classification are explained next.

Feature Extraction. In advanced problem modeling, we refine the feature selection by conducting an explainer analysis on 7 features (§4.2). We observe that the features (i) gate-type, (ii) inp, (iii) DE, and (iv) sign have higher importance compared to the remaining features (Figure 4). Hence, we utilize these 4 features in advanced problem modeling.

Modified Node Classification aims to predict whether the *key-gate* maps to INV/BUF. The GNN classifies the subgraphs extracted around key-gates as: $p_i = g(G', X', \sigma)$ where, $p_i \in \{0, 1\}$. The output class returned by GNN maps to an INV/BUF, *i.e.,* $p_i = 0$ for an INV and $p_i = 1$ for a BUF.

Post-processing. We recover the key-values by utilizing the (i) predictions of GNN-based node classification and (ii) type of the key-gate, *i.e.,* (X(N)OR/MUX/(N)AND/(N)OR) (Table 2). For example, if the key-gate is XOR, and GNN predicts INV (BUF), the key-value is inferred as 1 (0) (Figure 5 (a)).

Table 2: Mapping of key-gate to INV, BUF, constant–0 (const–0), and constant–1 (const–1) based on key-value.

Key-value	XOR	XNOR	AND	NAND	NOR	OR
0	BUF	INV	const–0	const–1	INV	BUF
1	INV	BUF	BUF	INV	const–0	const–1

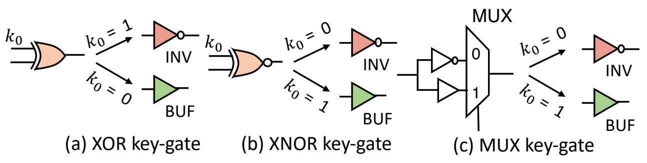


Figure 5: Mapping between key-gate, key-value, and GNN prediction (INV/BUF) for X(N)OR and MUX key-gates.

Similarly, if the key-gate is XNOR, and GNN predicts INV (BUF), the key-value is inferred as 0 (1) (Figure 5 (b)). If the key-gate is MUX, the key-value is inferred based on the connections of INV and BUF to the MUX. If GNN predicts INV, and INV is connected to the first (second) input of MUX, the key-value is inferred as 0 (1) (Figure 5 (c)).

In conclusion, the advanced problem modeling successfully predicts the secret key (KPA of **99.52**%) for designs locked with TRLL (Table 3).

4.5 Attacking Logic Synthesized Designs

Challenge ③. The semiconductor industry performs logic synthesis[§] to reduce the area, power, and delay of circuits. The advanced problem modeling (§4.4) reduces KPA (70%) for synthesized TRLL-locked designs. The KPA for non-synthesized designs from the same benchmark suite is 99.52% (Table 3). Modifying the attack to improve KPA for synthesized locked designs is challenging due to the different graph representations in logic-synthesized designs.

Solution ③. Using our advanced problem modeling, we input the trained model on synthesized TRLL-locked designs to GNNExplainer. We observe that the nodes in the subgraph, *i.e.,* the logic gates in the neighborhood of the key-gate contribute with different importance to the prediction. As manually inferring the explainer results and providing the important nodes to the attack is time-consuming, we improve the architecture of GNN by incorporating an attention layer. The attention layer assigns varying degrees of importance to the nodes in the subgraph using learnable weights [70]. The attention weight $\alpha_{vu}^{(t)}$ measures the strength of connectivity between the node $v \in V'$ and u in a graph G' for the t^{th} layer of GNN as follows.

$$\alpha_{vu}^{t} = \text{softmax}\left(f(a^\top[W^{(t)}h_v^{(t-1)}||W^{(t)}h_u^{(t-1)}])\right) \quad (1)$$

where $f(\cdot)$ is a LeakyReLU activation function, $h^{(t-1)}$ is output of $(t-1)^{th}$ GNN layer with $h_v^{(0)} = x_v'$, where $x_v' \in X'$, $W^{(t)}$ is a learnable weight matrix, and a is vector of learnable parameters. Thus, the modified GNN with an attention layer (i) captures relevant information (important nodes) from the local neighborhood of the key-gate without the need for GNNExplainer, and (ii) reduces the manual effort of identifying important nodes using GNNExplainer. Incorporating attention increases KPA by 10% for synthesized designs (Figure 6).

§A process transforming a behavioral chip description to an optimized hardware implementation, *i.e.,* a netlist consisting of Boolean logic gates.

Table 3: Key-prediction accuracy (KPA) for TRLL-locked designs ($|K| = 128$) using advanced problem modeling.

Design	b14_C	b15_C	b17_C	b20_C	b21_C	b22_C
Solution ②	99.78	99.68	99.06	99.53	99.53	99.53

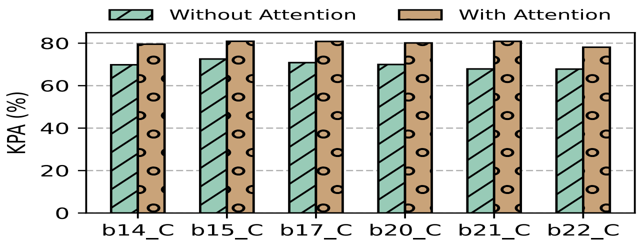


Figure 6: Improvement in key-prediction accuracy (KPA) for TRLL-locked designs for $|K| = 128$ with and without attention.

Challenge ④. Depending on optimization techniques (*e.g.,* minimizing area, power), logic synthesis tools may generate designs consisting of non-XOR/XNOR/MUX-based key-gates. We observe the KPA for our advanced problem modeling being limited for such designs. Thus, INSIGHT needs to be extended to tackle non-XOR/XNOR/MUX-based key-gates.

Solution ④. To address challenge ④, we first perform graph transformations to maximize the number of XOR/XNOR-based key-gates. We use a commercial logic synthesis tool (*e.g.,* Synopsys Design Compiler) with different optimization commands to generate structurally disparate (and functionally equivalent) locked designs. Doing so increases the number of XOR/XNOR-based key-gates in the locked design. However, even after these graph-based transformations, we observe non-XOR/XNOR-based key-gates. To that end, we extend the GNN prediction classes for other type of key-gates (Table 2). Considering key-values '1' and '0', the key-gates (AND/NAND/NOR/OR) either map to INV, BUF, constant–1, or constant–0. Thus, we introduce two additional classes to the prediction tasks, *i.e.,* constant–1 and constant–0. Incorporating the above-mentioned steps improves KPA by 1.53× (on average) for synthesized ALMOST designs (Table 4).

4.6 Lack (or Scarcity) of Training Data

Challenge ⑤. To facilitate (i) reduction in time-to-market, (ii) lower economic costs, and (iii) easier integration of chips, the semiconductor industry follows the concept of design-reuse [71]. Thus, an attacker is assumed to have access to

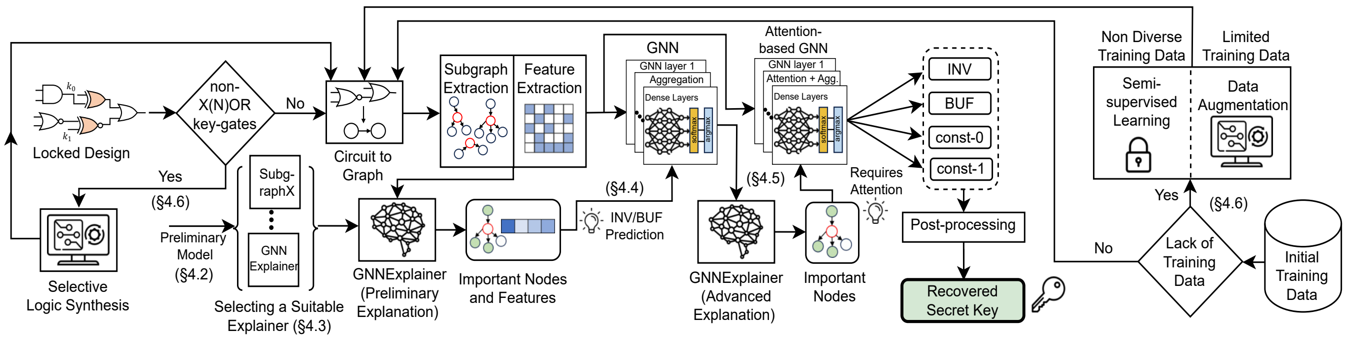


Figure 7: Architecture of INSIGHT.

Table 4: Improvement in key-prediction accuracy (KPA) for `ALMOST` ($|K|$ = 64) with non-XOR/XNOR-based key-gates.

Design	c1355	c1908	c2670	c6288	c7552
OMLA [31]	54.18	47.80	49.78	49.88	55.55
Solution 4	81.25	76.56	75	79.69	81.25
Improvement (×)	1.5×	1.6×	1.51×	1.6×	1.46×

Table 5: Improvement in key-prediction accuracy (KPA) for `RGLock` ($|K|$ = 128) with and without data augmentation.

Design	b14_C	b15_C	b17_C	b20_C	b21_C
No Data Augmentation (✗)	69.05	72.13	66.67	66.38	67.72
Solution 5	76.56	82.79	71.87	72.56	74.83
Improvement (×)	1.11×	1.15×	1.08×	1.09×	1.10×

datasets consisting of diverse examples to effectively capture the underlying patterns and relationships in the (graph) data. This is analogous to a library of publicly available designs having similar graph structures compared to the locked design under attack. However, an attacker faces considerable challenges when they have limited access to training data owing to a (i) closed-source (proprietary) locking technique and (ii) limited diversity of subgraphs in the training dataset. Thus, we utilize two techniques, *i.e.,* data augmentation and semi-supervised learning, to increase the training samples.

Solution 5. To address challenge 5, we propose data augmentation and semi-supervised learning. **Data Augmentation** augments training samples by graph transformations, *i.e.,* logic synthesis using logic optimization procedures (known as synthesis recipes). This step increases the overall training samples, which vary in structure, *i.e.,* graph-based representation. For instance, the GNN model trained using the circuit library approach showed lower KPA for synthesized `RGLock` designs. Data augmentation increases the KPA by 1.11× (Table 5). **Semi-supervised Learning** can be used by an attacker having access *only* to the locked design and *no/limited access to a circuit library*. This corresponds to cases when attacking a newly developed/unknown hardware IP. An attacker adopts semi-supervised learning and locks other (unprotected) logic gates in the design to generate training samples. Semi-supervised learning increases the KPA by 1.29× (from 67.14% to 86.94%) for the `TroMUX`-locked design, AES, compared to training on a circuit library, *i.e.,* designs from the ITC-99 benchmark suite.

Algorithm 1: Pseudocode of INSIGHT

```
Input: Locked design $\mathcal{D}_{lock}$, Circuit library $\mathbb{C}_{lib}$, Locking technique $\mathcal{L}$
Output: Secret key $\mathbb{K}_{pred}$
1  Function test_attention_GNN($Test_{subgraphs}$, $X_{test}$):
2      for $(G'(V', E'), X') \in (Test_{subgraphs}, X_{test})$ do
3          for $l = 1$ to $L$ do
4              for $v \in V'$ do
5                  $\mathbf{h}_v^{(l)} \leftarrow \sigma(\text{AGGREGATE}(\{\text{ATTENTION}(\mathbf{h}_v^{(l-1)}, \mathbf{h}_u^{(l-1)}) \mid u \in \mathcal{N}(v)\}))$
6              $y_{test}$.append(argmax($H^L$))
7      return $y_{test}$
8  if $\mathcal{D}_{lock}$ contains non-X(N)OR-based key-gates then
9      $\mathcal{D}_{lock} \leftarrow$ graph_transformation ($\mathcal{D}_{lock}$) /* Perform logic synthesis on $\mathcal{D}_{lock}$ to increase X(N)OR key-gates (§4.5) */
10 if circuit library is available then
11     $Train_{data} \leftarrow$ create_training_data($\mathbb{C}_{lib}$) /* Generate training dataset using circuit library (§4.6) */
12 else
13     $Train_{data} \leftarrow$ create_semi_supervised_training($\mathcal{D}_{lock}$, $\mathcal{L}$) /* Generate training dataset by locking unprotected logic (§4.6) */
14 if limited training dataset exists then
15     $Train_{data} \leftarrow$ data_augmentation($Train_{data}$) /* Increase training dataset by performing data augmentation (§4.6) */
16 $\mathbb{G}_{train}, G_{test} \leftarrow$ convert_to_graph($Train_{data}$, $\mathcal{D}_{lock}$)
17 $Train_{subgraphs}, Test_{subgraphs}, X_{train}, X_{test} \leftarrow$
       subgraph_feature_extraction($\mathbb{G}_{train}$, $G_{test}$) /* Extract subgraphs and features (§4.2) */
18 $T_{GNN} \leftarrow$ train_attention_GNN($Train_{subgraphs}$, $X_{train}$) /* Train modified GNN for advanced problem modeling (§4.5) */
19 $y_{test} \leftarrow$ test_attention_GNN($Test_{subgraphs}$, $X_{test}$) /* Obtain predictions for testing dataset ($\mathcal{D}_{lock}$) (§4.5) */
20 $Feat_{scores}, Nodes_{scores} \leftarrow$ GNNExplainer($y_{test}$, $Test_{subgraphs}$, $T_{GNN}$) /* Obtain explanations for predictions (§4.3) */
21 $\mathbb{K}_{pred} \leftarrow$ post_process($y_{test}$, $\mathcal{D}_{lock}$) /* Post-processing (§4.5) */
22 return $\mathbb{K}_{pred}, Feat_{scores}, Nodes_{scores}$
```

4.7 Final Architecture (Putting it all Together)

We illustrate and describe the final architecture of INSIGHT in Figure 7 and Algorithm 1. Given a locked design D_{lock} with $|K|$ *key-gates*, INSIGHT recovers the secret key using explainable GNN. We use GNNExplainer to improve our attack since it provides relevant explanations and is computationally efficient (§4.3). Next, as per the explanations from GNNExplainer, we map the key-prediction problem to INV/BUF prediction (§4.4). To attack logic-synthesized designs, we utilize another insight from GNNExplainer, highlighting the varied importance of nodes (in the neighborhood of *key-gates*) toward predictions. Consequently, we improve the attack by incorporating attention (§4.5). If the locked design consists of non-X(N)OR-based key-gates, we perform graph transformations, *i.e.,* logic synthesis, to increase the number of X(N)OR key-gates (lines 8–9 in Algorithm 1) and extend the GNN prediction classes for different types of key-gates (Table 2).

If an attacker has access to a circuit library ($\mathbb{C}_{lib}$), they generate the training dataset using a circuit library (lines 10–11).¶ If there is no access to a circuit library, we lock the unprotected logic gates in the locked design to generate training samples, *i.e.,* semi-supervised learning (lines 12–13) (§4.6). If a limited training dataset exists, we perform graph transformations to increase the number of training samples, *i.e.,* data augmentation (lines 14–15) (§4.6). Next, we convert the designs to graphs (line 16) and extract the (i) subgraph (neighborhood of *key-gates*) and (ii) features of the training and testing samples (line 17) (§4.2). We then train the modified GNN with the training subgraphs and obtain a trained GNN model (T_{GNN}) (line 18). Upon obtaining the trained GNN model, we test the GNN for testing subgraphs ($Test_{subgraphs}$) and obtain GNN predictions (y_{test}), *i.e.,* INV/BUF prediction (line 19). The inference algorithm of attention-based GNN is mentioned in lines 1–7. Upon obtaining the GNN predictions, (i) we pass the predictions (y_{test}), subgraphs of testing data ($Test_{subgraphs}$), and trained GNN model (T_{GNN}) through GNNExplainer to obtain corresponding explanations in terms of importance scores for features and nodes in the subgraph (line 20) and (ii) perform post-processing on the predictions (y_{test}) to obtain the secret key ($\mathtt{K}_{pred}$) using the mapping between INV/BUF prediction and key-gate types (line 21).‖ The computational complexity of INSIGHT depends on all the aforementioned steps. It is non-trivial to derive the time complexity of INSIGHT due to the closed-source commercial logic synthesis algorithms utilized for graph transformations.**

5 Results

We outline our experimental setup (§5.1) and evaluate INSIGHT on 7 unbroken locking techniques (§5.2, §5.3). We perform experiments on designs from the ISCAS-85 and ITC-99 academic suites to open-source designs such as the Stanford `MIPS` processor [46], Google `IBEX` processor [47], DARPA Common Evaluation Platform (CEP) `GPS` [48], and `mor1kx` processor [49]. Finally, we perform 2 practical case studies: (i) recovering the secret key from designs locked using `Synopsys TestMAX`, a commercial, widely used EDA tool (§5.5), and (ii) showcasing the ramifications of leaking the secret key using an image processing application (§5.6).

¶The to-be-attacked locked design is not a part of the circuit library.
‖Refer to Appendix A.2 for integrating INSIGHT with LL techniques.
**Refer to Appendix A.3 for details on computational complexity.

5.1 Experimental Setup

Implementation Setup. We implement INSIGHT using *Python 3.7* (converting locked designs to graphs and post-processing of graph neural network (GNN) predictions), *PyTorch 1.12* (GNN-based node classification), and deep graph library (GNNExplainer). We perform experiments on a single compute node of AMD *EPYC Rome* CPU with 128 cores operating at 2.25GHz with 512GB RAM each. We use a graph isomorphism network as the baseline GNN architecture. It consists of 2 multilayer perceptron layers and 4 to 6 GNN layers (based on the dataset). Moreover, we implement a multi-head attention layer in GNN where the number of heads varies between 1 and 4 (based on the dataset). We set the batch size and hidden dimension to 128 and use a LeakyRelu activation function. We set the number of epochs to 100 and use the Adam optimizer with a learning rate 0.001.
Locking Techniques. We implement `TroMUX` and `RGLock` in *Python 3.7*, as the source codes are not publicly available. We received the binaries of `TRLL` and `TRLL+` from the authors. We received locked benchmarks from the authors of `ALMOST` [36] and `SimLL` [52]. Also, we obtain open-source benchmarks for `TroMUX` [72] and `UNSAIL` [73]. For experiments with `Synopsys TestMAX`, we use version `T-2022.03-SP5-4`.
Designs. We demonstrate the efficacy of INSIGHT on designs from ISCAS-85 [74], ITC-99 [75], and EPFL [76] benchmark suites widely used in the hardware design/security community. We conduct experiments on open-source designs such as the Stanford `MIPS` processor [46] (≈23k gates), Google `IBEX` processor [47] (≈18k gates), DARPA CEP `GPS` [48] (≈193k gates), `mor1kx` processor [49] (≈158k gates), crypto cores [77], `openMSP430` microcontroller [78], arithmetic designs, and an image-processing application [79]. These designs encompass a diverse range of IPs comprising processors, arithmetic designs, controllers, microcontrollers, crypto cores, a GPS core, and an image processing application (Gaussian blurring), widely used in hardware designs.

5.2 Evaluation of Locking Techniques

We evaluate the efficacy of INSIGHT using key-prediction accuracy (KPA) (§2.1). We compare the KPA with existing and open-source ML-based attacks [31, 32] for non-synthesized (Table 6) and synthesized (Table 7) locked designs.
Non-Synthesized Designs. Table 6 depicts the performance of INSIGHT with existing ML-based attacks for `TRLL`, `TRLL+`, `TroMUX`, and `RGLock`. INSIGHT achieves a significantly higher KPA than existing ML-based attacks, *i.e.,* our attack improves KPA by **2.96**×, **1.64**×, and **1.86**× than SCOPE, MuxLink, and OMLA. Notably, INSIGHT achieves a KPA of ≈ 100

Table 6: Efficacy (measured in KPA) of INSIGHT compared to existing ML-based attacks for **non-synthesized** designs. "–" indicates designs do not support locking for key-size ($|K|$) = 256. All numbers are in percentage and averaged across ten trials.

Design	Key-size	SCOPE [32]				OMLA [31]				INSIGHT (This Work)			
		`TRLL` [24]	`TRLL+` [35]	`TroMUX` [37]	`RGLock` [51]	`TRLL`	`TRLL+`	`TroMUX`	`RGLock`	`TRLL`	`TRLL+`	`TroMUX`	`RGLock`
b14_C	128	38.59	21.12	41.64	25.54	50.54	50.78	51.09	51.87	99.78	93.05	89.84	76.02
	256	35.21	23.30	41.95	22.65	50.56	48.44	49.80	50.52	99.43	99.09	89.06	80.20
b15_C	128	27.85	10.51	44.14	13.20	46.24	46.87	50.47	50.58	99.68	92.71	87.5	78.91
	256	34.52	15.46	45.43	–	49.88	47.65	49.65	–	99.62	97.60	94.92	–
b17_C	128	30.31	10.58	43.20	29.29	51.01	52.56	49.65	49.61	99.06	95.70	92.18	77.11
	256	33.94	10.38	44.65	28.08	50.26	50.56	50.56	49.22	99.89	98.44	91.40	68.59
b20_C	128	31.87	19.37	44.69	30.00	48.98	51.56	51.79	50.47	99.53	98.91	83.59	78.20
	256	43.32	23.08	46.29	27.77	50.27	49.55	50.35	50.01	99.87	98.44	87.11	81.13
b21_C	128	32.15	22.26	45.44	30.00	50.00	50.45	48.43	49.29	99.53	94.14	87.50	85.16
	256	38.38	23.47	46.60	27.54	51.02	48.76	51.39	51.72	99.41	97.90	88.67	83.05
b22_C	128	33.20	20.59	48.20	30.55	51.82	49.65	49.33	51.78	99.53	93.12	84.37	82.97
	256	41.35	17.93	45.97	31.79	48.77	51.32	50.16	48.05	99.90	99.28	83.59	83.24
Average	128	32.33	17.46	44.55	26.56	49.76	50.31	50.13	50.6	**99.52**	**94.60**	**87.49**	**79.73**
	256	37.78	18.94	45.15	22.97	50.13	49.38	50.32	41.59	**99.69**	**98.46**	**89.12**	**79.24**

Table 7: Efficacy (measured in KPA) of INSIGHT compared to existing ML-based attacks for **synthesized** designs. "–" indicates designs do not support locking for key-size ($|K|$) = 256. All numbers are in percentage and averaged across ten trials.

Design	Key-size	SCOPE [32]				OMLA [31]				INSIGHT (This Work)			
		`TRLL` [24]	`TRLL+` [35]	`TroMUX` [37]	`RGLock` [51]	`TRLL`	`TRLL+`	`TroMUX`	`RGLock`	`TRLL`	`TRLL+`	`TroMUX`	`RGLock`
b14_C	128	37.65	20.97	38.12	14.68	50.28	52.35	50.47	47.14	79.93	77.78	82.20	76.56
	256	34.94	22.42	38.20	17.93	50.81	52.45	49.20	50.02	79.56	78.83	81.76	73.96
b15_C	128	33.71	10.52	37.26	10.00	48.42	49.15	48.89	51.21	78.90	70.36	84.25	82.79
	256	33.34	16.37	38.51	–	50.24	50.32	49.56	–	79.31	73.19	84.99	–
b17_C	128	35.58	15.23	38.91	15.00	51.79	51.95	47.72	50.87	81.98	70.91	84.53	71.87
	256	34.55	10.49	39.06	16.17	50.55	48.76	51.82	50.08	79.22	69.44	84.20	72.58
b20_C	128	39.17	18.83	40.62	16.09	48.55	49.20	50.40	50.26	80.11	77.78	82.67	72.56
	256	40.89	18.16	40.74	16.05	49.49	50.42	50.59	51.68	79.55	77.12	82.17	74.39
b21_C	128	39.33	22.77	39.76	17.26	50.77	52.34	50.14	51.42	81.46	78.56	82.67	74.83
	256	36.99	23.38	41.64	19.06	50.93	48.96	50.74	48.19	79.88	69.84	82.95	75.21
b22_C	128	40.82	22.91	44.61	17.35	50.86	51.20	50.69	48.14	77.51	78.03	82.65	73.53
	256	41.09	17.77	41.87	17.11	49.64	49.67	46.09	50.84	78.24	79.87	82.41	74.92
Average	128	37.71	18.54	39.88	15.06	50.11	51.03	49.71	49.84	**79.98**	**75.57**	**83.16**	**75.36**
	256	36.97	18.09	40.03	17.26	50.27	50.09	49.67	50.16	**79.29**	**74.71**	**83.08**	**74.21**

for `TRLL`. On average, INSIGHT achieves a KPA of 99.60%, 96.53%, 88.30%, and 79.48% for `TRLL`, `TRLL+`, `TroMUX`, and `RGLock`, respectively. In contrast, the KPA of OMLA is $\approx$ 50%, equivalent to a random guess. *Our results highlight that* INSIGHT *successfully recovers the secret keys for non-synthesized locked designs, indicating that the aforementioned techniques leak exploitable structural information.*

Synthesized Designs. Recall that the semiconductor industry performs logic synthesis to reduce the area/power/delay of circuits (§4.5). Hence, we perform security analysis on synthesized locked designs. Table 7 depicts the performance of INSIGHT on synthesized locked designs. INSIGHT improves KPA by **2.8**× and **1.56**× than SCOPE and OMLA. On average, INSIGHT achieves a KPA of 79.64%, 75.14%, 83.12%, and 74.78% for `TRLL`, `TRLL+`, `TroMUX`, and `RGLock`, respectively. Our results highlight that INSIGHT recovers the secret key for synthesized locked designs, whereas other ML-based attacks [31, 32] are severely limited. *In addition, our results indicate that the aforementioned techniques rely on synthesis-induced transformations to enhance security.*

Other Locking Techniques. We evaluate the security of `ALMOST` [36],`UNSAIL` [38], and `SimLL` [52]—these techniques claim security against ML-based attacks (§2.1). On average, INSIGHT achieves a KPA of 77.4% for `ALMOST`, 78.56% for `UNSAIL`, and 99.51% for `SimLL` (Table 8), indicating broad applicability across learning resilient locking techniques.

5.3 Analysis of INSIGHT

Effect of Graph Transformations. The underlying structure of the graph, *i.e.,* locked design changes with synthesis recipes.[††] Thus, we analyze the impact of synthesis recipes on the efficacy of INSIGHT. We fix the synthesis tool (widely used commercial EDA tool—`Synopsys Design Compiler`), locking technique as `TRLL`, and technology library as `Nangate` 45nm. Our results indicate that the differ-

[††]Sequence of optimization commands to minimize area/power/delay.

Table 8: Efficacy (KPA (%)) of INSIGHT on open-sourced ALMOST [36], UNSAIL [38], and SimLL [52] designs. "–" indicates designs are not open-sourced and publicly available.

Design	ALMOST [36]		UNSAIL [38]		
	$\|K\| = 64$	$\|K\| = 128$	CL_v2	CL_v4	XOR
c1355	81.25	79.69	–	–	–
c1908	76.56	75	–	–	–
c2670	75	73.44	89.84	82.03	81.54
c3540	79.69	75.78	84.09	84.09	84.49
c5315	81.25	77.34	77.37	75.78	80.91
c6288	79.69	70.31	71.61	–	–
c7552	81.25	77.34	79.68	73.48	–
c880	–	–	76.92	79.53	78.29
Average	**79.24**	**75.55**	**79.92**	**78.98**	**81.31**

Design	SimLL [52]
	$\|K\| = 512$
b14_C	99.78
b15_C	99.44
b17_C	99.22
b20_C	99.61
b21_C	99.78
b22_C	99.22
Average	**99.51**

ence between the average KPA for both synthesis recipes is negligible (refer to Table 13 in Appendix A.4 for results).

Scalability and Runtime. We analyze the impact of (i) key-size and (ii) design size on the attack runtime of INSIGHT (refer to Figure 9 in Appendix A.4 for results). The attack runtime increases with a negligible difference with an increase in key-size (64 to 256). In addition, INSIGHT is scalable to larger designs such as IBEX, mor1kx processor, and GPS; *i.e.,* the attack runtime on these designs (with an average of 123k gates) is 597.31 seconds. Note that INSIGHT performs inference on the subgraphs extracted around the key-gate. Thus, the runtime depends on the subgraph size rather than the design size. The training time for INSIGHT is: (i) 6 hours for circuit library-based training, (ii) 36 hours for samples generated using data augmentation, and (iii) 8 hours for semi-supervised learning. These numbers (training and attack runtime) show that an attacker in the foundry can perform this attack, given that the time allotted to foundries for fabrication is 3 months [16].

5.4 Findings of INSIGHT

- Recall that we embed structural and functional features to the datasets (§4.2) and convert the key-prediction problem to inverter/buffer prediction (§4.4). We note that the explainer attributes higher importance to structural features than functional features when the key-gate maps to a buffer. Conversely, higher importance is attributed to functionality, *i.e.,* gate-type, when the key-gate maps to an inverter.
- In the considered TRLL-locked designs, if the key-gate has inputs of primary input and *key-input*, the key-gate maps to an inverter irrespective of the logic gates in the neighborhood. Similarly, if the output of the key-gate is connected to a primary output, the key-gate maps to a buffer.
- Logic synthesis is a *double-edged sword* for the security of learning resilient locking techniques. While ALMOST uses adversarial learning to generate synthesis recipes for locked designs (§2.1), our experiments demonstrate that an attacker can use logic synthesis to reduce the security guarantees through tailored graph transformations (Table 4).

Table 9: Efficacy (KPA (%)) of INSIGHT on practical designs locked using Synopsys TestMAX ($|K| = 128$).

Design	#gates	TRLL [24]	TRLL+ [35]
Google IBEX	17,764	99.22	99.22
Stanford MIPS	23,511	97.65	99.22
DARPA CEP GPS	193,141	99.22	99.22
mor1kx	158,265	99.22	96.87
Average	–	**98.82**	**98.63**

Table 10: Efficacy (KPA (%)) of INSIGHT on practical designs locked using TroMUX [37].

Design	#gates	Key-size ($\|K\|$)	KPA (%)
AES	16,509	1,202	86.94
SEED	12,682	1,612	80.64
SPARX	8,146	2,582	99.07
Camellia	6,710	1,271	94.02
CAST	12,682	1,572	80.41
TDEA	2,269	214	99.53
PRESENT	868	241	87.55
openMSP430	5,921	543	84.89
Average	–	1,155	**89.13**

5.5 Security Evaluation on Practical Designs

We evaluate the security of practical designs locked using TroMUX [37] and Synopsys TestMAX. For TroMUX, we launch attacks on 7 locked crypto cores and microcontroller [72] for varying key-sizes. On average, we achieve a KPA of 89.13% (Table 10). TRLL and TRLL+ are integrated into an industry-adopted and widely used semiconductor EDA tool, Synopsys TestMAX. We evaluate the security of processors and GPS locked using TRLL and TRLL+ (Table 9). On average, INSIGHT achieves a KPA of **98.82**% and **98.63**%, highlighting that it (i) can successfully recover the secret keys from practical designs locked using industry-adopted locking techniques and (ii) is scalable w.r.t. design size and key-sizes.

5.6 Ramifications of Leaking Secret Key

We showcase the ramifications of leaking the secret key using an image processing application (hardware implementation of Gaussian blurring [79]) as follows. We lock the 16-bit adders in the hardware implementation with $|K| = 16$ using TRLL [24]. Subsequently, we launch INSIGHT and OMLA [31] to recover the secret key of the locked adders. INSIGHT recovers 93.75% of the secret key, while OMLA recovers only 50% of the secret key. Next, we evaluated the functionality of the recovered design by simulating the image processing module with adders recovered by INSIGHT and OMLA. For example, consider the image in Figure 8 (a) as input. The output of the design recovered by INSIGHT (Figure 8 (c)) is similar to the original output (Figure 8 (b)). However, the output of the design recovered by OMLA (Figure 8 (d)) is completely different from the original output (Figure 8 (b)). This case study shows that by using INSIGHT, an attacker can cause a practical attack by recovering the secret key.

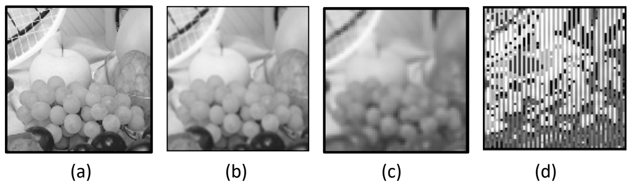


Figure 8: Gaussian blurring example. (a) input image, (b) output (golden) image, (c) output image recovered by INSIGHT, and (d) output image recovered by OMLA [31].

6 Related Work and Discussion

To the best of our knowledge, INSIGHT is the first to use explainable machine learning (ML) to develop a successful attack in semiconductor supply chain security. Here, we discuss (i) different hardware IP protections and attacks, (ii) the actionability of INSIGHT, and (iii) potential countermeasures.

6.1 Hardware IP Protections and Attacks

FuncTeller [80] proposed an oracle-guided attack on embedded field programmable gate array (eFPGA)-based hardware redaction and recovered the approximate functionality of eFPGA-redacted components using logic synthesis principles and querying input/output pairs from an oracle. **Chhotaray *et al.*** [10] introduced a formal syntax for design-hiding schemes to protect design IPs against reverse-engineering attacks. **Han *et al.*** [9] proposed an oracle-guided attack recovering the secret key in selected corrupt-and-correct-based locking techniques. It leverages logic synthesis principles utilized in IC design tools to recover the secret key. **Yasin *et al.*** [8] proposed the first-of-its-kind provably-secure logic locking technique, providing quantifiable resilience against oracle-guided attacks and removal attacks. The security guarantees of [8] were broken by Han *et al.* [9]. **Massad *et al.*** [12] proposed an oracle-guided attack on IC camouflaging. The attack employed an SAT solver to compute discriminating set of input patterns to recover the Boolean functionality of the camouflaged logic gates. **Rajendran *et al.*** [11] proposed an IC camouflaging-based defense technique considering an oracle-guided threat model. It strategically selects logic gates (to be camouflaged) by leveraging VLSI/IC testing principles to thwart reverse-engineering. This technique was broken by Massad *et al.* [12]. **Imeson *et al.*** [7] proposed a split manufacturing-based defense technique that thwarts oracle-less attacks from untrusted foundry and leveraged 3D IC technology to obfuscate security-critical wires in designs.

In contrast, INSIGHT: (i) is an explainability-guided attack that employs explainable GNN, (ii) attacks learning resilient logic locking techniques, (iii) provides explanations, and (iv) is an oracle-less attack.

6.2 Actionability from INSIGHT

INSIGHT provides explanations in terms of importance scores for the (i) nodes in the neighborhood of key-gate and (ii) features responsible for predictions (§4.4). Hardware designers can understand these explanations as they are just numbers (Appendix A.1). However, the actionability of explanations from INSIGHT varies with the level of expertise (beginner, intermediate, and advanced) in ML/GNNs. Next, we describe the actionability of the explanations in enhancing the attack and locking technique w.r.t. expertise in ML/GNNs.

Hardware designers with a **beginner's** knowledge of ML/GNN can interpret the important nodes or features in the subgraph extracted around key-gates based on importance scores. Through this information, a designer can protect the important nodes and features, as showcased in §6.3. Hardware designers with an **intermediate** knowledge of ML/GNN can modify the problem modeling of INSIGHT by analyzing the importance scores of nodes/features across different data samples, *i.e.,* subgraphs extracted around different key-gates. For instance, we modified the problem modeling by analyzing the explanations from INSIGHT (§4.4). Similarly, a designer can modify the locking technique by obtaining the global reason behind the success of the attack by analyzing explanations. Hardware designers with **advanced** knowledge of ML/GNN can map explanations provided by INSIGHT to the working of the GNN model. For instance, we modified the architecture of GNN by analyzing explanations (§4.5). Moreover, with advanced knowledge in ML/GNN, a designer can enhance a locking technique by performing and analyzing graph perturbations, *i.e.,* analyzing how changes such as locking important nodes or graph transformations impact the GNN predictions and performance of INSIGHT.

6.3 Limitations and Countermeasures

Limitations. INSIGHT could not recover (i) the entire secret key for many designs, *i.e.,* there remains scope for improvement, and (ii) the key-value in densely locked regions, *i.e.,* key-gates surrounded by a majority of locked gates.

Potential countermeasures. INSIGHT uses explainability-guided GNN-based node classification and recovers a significant portion (**87.30%**) of the secret key of 7 unbroken locking techniques. Existing ML-based attacks provide KPA as a result. Although designers can identify which key-gates are incorrectly predicted using KPA, the metric does not provide the reasons behind the predictions. Understanding the reasons behind the predictions will aid designers in strategically developing countermeasures against ML-based attacks. INSIGHT provides explanations regarding (i) important nodes in the neighborhood of key-gate and (ii) important features responsible for predictions (§4.4). Using these explanations, we propose 2 potential countermeasures against INSIGHT.

The **first** direction (**CM#1**) is a consequence of **important**

Table 11: Efficacy (KPA (%)) of INSIGHT compared to existing ML-based attacks (OMLA [31] and SCOPE [32]) for potential countermeasures (CM#1 and CM#2) for $|K| = 128$.

Design	OMLA [31]		SCOPE [32]		INSIGHT (This Work)	
	CM#1	**CM#2**	**CM#1**	**CM#2**	**CM#1**	**CM#2**
b14_C	51.51	42.31	32.26	26.98	70.31	68.75
b15_C	51.28	46.46	25.71	26.12	64.84	63.28
b17_C	47.50	49.55	34.09	35.18	67.96	64.84
b20_C	46.15	50.00	35.00	37.09	66.40	63.28
b21_C	55.17	49.32	20.69	28.69	65.62	70.31
b22_C	48.79	50.00	31.17	27.66	65.62	62.50
Average	**50.06**	**47.94**	**29.82**	**30.29**	**66.79**	**65.49**

Table 12: Area, power, and delay overheads (in %) for potential countermeasures (CM#1 and CM#2).

Design	CM#1			CM#2		
	Area	**Power**	**Delay**	**Area**	**Power**	**Delay**
b14_C	6.76	14.41	0	14.16	9.42	0.30
b15_C	2.59	13.69	0	10.04	4.32	0.10
b17_C	2.65	7.78	0	10.91	6.11	0
b20_C	2.47	5.31	0	11.55	6.63	0
b21_C	2.11	5.54	0.10	12.53	7.84	0.10
b22_C	0.66	2.39	0.10	10.77	5.96	0.10
Average	2.87	8.19	0.03	11.66	6.71	0.10

node explanations provided by explainable GNN. We observe that the nodes in the neighborhood of the key-gate contribute with different importance to the prediction. We lock the important nodes in the neighborhood of the key-gate to thwart INSIGHT from exploiting hints. To evaluate the efficacy of CM#1, we launch existing ML-based attacks (OMLA and SCOPE) and INSIGHT on selected ITC-99 designs (locked using CM#1) for $|K| = 128$. INSIGHT obtained a KPA of **66.79**%, whereas existing attacks (SCOPE and OMLA) obtained a KPA of 29.82% and 50.06% (Table 11).[‡‡]

Design tradeoffs. We observe that the number of key-gates in designs locked using CM#1 increases with the number of important nodes, leading to higher area and power overheads (Table 12). One potential way to reduce the overheads is to lock the important nodes in an iterative approach (in the descending order of importance scores). In this approach, important nodes are locked until the attack (INSIGHT) is thwarted.

The **second** direction (**CM#2**) is a consequence of **important feature** explanations returned by explainable GNN. Recall that the gate-type feature has higher importance compared to other features (§4.5). To thwart INSIGHT from exploiting the gate-type features from the neighborhood of the key-gate, we perform custom graph transformations in the neighborhood of the key-gates. For example, if the "XNOR" gate-type feature is of higher importance, we transform the neighborhood of the key-gate so that it does not utilize the"XNOR" gate. INSIGHT obtained a KPA of 63.83%, whereas existing attacks OMLA [31] and SCOPE [32] obtained a KPA of 47.94% and 30.29%, as shown in Table 11. **In conclusion**, CM#1 and CM#2 decrease the KPA of INSIGHT by 1.32× and 1.36× compared to targeted locking techniques.

Design tradeoffs. The custom graph transformations result in higher area and power overheads (Table 12). One potential way to reduce the overheads is to search for an important feature (gate-type) iteratively and eliminate the gate-type that results in higher overheads for custom graph transformations.

[‡‡]Providing theoretical analysis supporting the robustness (security) of proposed countermeasures involves delving into the mathematical underpinnings of ML/GNN models, an ongoing research in the ML community. Consequently, providing theoretical evidence against ML/GNN-based attacks is non-trivial and beyond the scope of the current work.

7 Conclusion and Ramifications

By harnessing the potential of an explainable graph neural network (GNN), we developed INSIGHT, which recovers the secret key (without requiring scan access) of **7** unbroken learning resilient locking techniques, including **2 industry-adopted techniques**. We overcame 5 critical challenges: (i) selecting a suitable explainer, (ii) identifying exploitable structural hints, (iii) attacking logic-synthesized designs, (iv) tackling unique graph representations, and (v) tackling insufficient training datasets. We showcased the efficacy of INSIGHT on designs ranging from widely-used academic benchmarks to processors. INSIGHT achieves an improved key-prediction accuracy of **2.87**×, **1.75**×, and **1.67**× compared to existing ML-based attacks. Finally, we perform 2 practical case studies showcasing the promise of INSIGHT on designs locked using `Synopsys TestMAX` and illustrating the ramifications of leaking the secret key. Our work highlights the potential of using explainable GNN, and our developed solutions can be used for other hardware security problems.

Ramifications. INSIGHT questions the security of learning resilient locking techniques (including industry-adopted techniques). INSIGHT: (i) acts as a litmus test showcasing the real efficacy of learning resilient locking techniques, (ii) cautions developers of locking techniques to remain cognizant of the fact that attackers can improve attacks by using explanations from ML/GNN explainers, and (iii) provides explanations, thereby aiding designers in developing countermeasures.

Ethical Considerations. We highlight that while INSIGHT can be potentially harmful at the hands of a malicious attacker whose goal is to break a logic locking defense, it is important to note that logic locking is not yet widely utilized by IC design companies for hardware IP protection. Our research, therefore, presents a timely opportunity to evaluate and enhance these techniques, paving the way for a solid and stable solution before widespread adoption. INSIGHT sheds light on vulnerabilities in learning resilient locking techniques, thus encouraging the research on effective countermeasures against ML/GNN-based attacks. Through this work, we invite logic-locking developers to work collaboratively with us on developing learning resilient locking techniques.

Acknowledgments

We thank the reviewers and Shepherd for providing valuable and constructive comments toward improving our work.

References

[1] M. LaPedus. What is the cost of fabs and R&D at 22-nm? https://rb.gy/zmk84x, October 2nd, 2009. [Online; last accessed 8-February-2024].

[2] E. Kinery. TSMC to up Arizona investment to $40 billion with second semiconductor chip plant. https://rb.gy/0z4zpd. [Online; last accessed 8-February-2024].

[3] J. Mark and D. T. Roberts. United States–China semiconductor standoff: A supply chain under stress . https://rb.gy/h76yt9, February, 2023. [Online; last accessed 8-February-2024].

[4] FACT SHEET: CHIPS and Science Act Will Lower Costs, Create Jobs, Strengthen Supply Chains, and Counter China. https://rb.gy/f35gpo, August 09, 2022. [Online; last accessed 8-February-2024].

[5] Y. Alkabani et al. Active hardware metering for intellectual property protection and security. In *Proc. USENIX Security Symposium.*, pp. 1–20, 2007.

[6] M. Rostami et al. A primer on hardware security: Models, methods, and metrics. *Proc. of IEEE*, pp. 1283–1295, 2014.

[7] F. Imeson et al. Securing computer hardware using 3d integrated circuit IC technology and split manufacturing for obfuscation. In *Proc. USENIX Security Symposium*, pp. 495–510, 2013.

[8] M. Yasin et al. Provably-secure logic locking: From theory to practice. In *Proc. of the ACM SIGSAC Conference on Computer & Communications Security*, pp. 1601–1618, 2017.

[9] Z. Han et al. Does logic locking work with EDA tools? In *Proc. USENIX Security Symposium*, pp. 1055–1072, 2021.

[10] A. Chhotaray and T. Shrimpton. Hardening circuit-design IP against reverse-engineering attacks. In *IEEE Symposium on Security and Privacy*, pp. 1672–1689, 2022.

[11] J. Rajendran et al. Security analysis of integrated circuit camouflaging. In *Proc. of the ACM SIGSAC Conference on Computer & Communications Security*, pp. 709–720, 2013.

[12] M. El Massad et al. Integrated circuit (IC) decamouflaging: Reverse engineering camouflaged ics within minutes. In *Proc. Network and Distributed System Security Symposium*, pp. 1–14, 2015.

[13] Annual Intellectual Property Report to Congress FY2021 . https://rb.gy/z70g1k, 2022. [Online; last accessed 8-February-2024].

[14] S. N. Dhanuskodi et al. COUNTERFOIL: Verifying provenance of integrated circuits using intrinsic package fingerprints and inexpensive cameras. In *Proc. USENIX Security Symposium*, pp. 1255–1272, 2020.

[15] K. Yang et al. A2: Analog malicious hardware. In *IEEE Symposium on Security and Privacy*, pp. 18–37, 2016.

[16] T. Trippel et al. Icas: An extensible framework for estimating the susceptibility of IC layouts to additive trojans. In *IEEE Symposium on Security and Privacy*, pp. 1742–1759, 2020.

[17] V. Gohil et al. Attrition: Attacking static hardware trojan detection techniques using reinforcement learning. In *Proc. of the ACM SIGSAC Conference on Computer & Communications Security*, pp. 1275–1289, 2022.

[18] C. Horton. US hits back at China over IP theft in chip case. http://rb.gy/qhkj3q, December 5th, 2018. [Online; last accessed 8-February-2024].

[19] D. Alexander. Theft of F-35 design data is helping U.S. adversaries -Pentagon. http://rb.gy/5uxh3r, 2013. [Online; last accessed 8-February-2024].

[20] J. A. Roy et al. EPIC: Ending piracy of integrated circuits. In *Proc. DATE*, pp. 1069–1074, 2008.

[21] R. Jarvis and M. McIntyre. Split Manufacturing Method for Advanced Semiconductor Circuits, 2007. US Patent 7,195,931.

[22] A. Chakraborty et al. Keynote: A disquisition on logic locking. *IEEE Trans. on Comput.-Aided Design of Integr. Circuits and Syst.*, pp. 1952–1972, 2019.

[23] S. Leef. AISS: Automatic Implementation of Secure Silicon. https://shorturl.at/QPlNl, 2019. [Online; last accessed 8-February-2024].

[24] N. Limaye et al. Thwarting all logic locking attacks: Dishonest oracle with truly random logic locking. *IEEE Trans. on Comput.-Aided Design of Integr. Circuits and Syst.*, pp. 1740–1753, 2021.

[25] Expanding Domestic Manufacturing of Secure, Custom Chips for Defense Needs . http://surl.li/tvcgf, 2021. [Online; last accessed 8-February-2024].

[26] A Cross-layer framework for cost-effective Intellectual Property (IP) Protection . http://surl.li/tvcfx, February, 2021. [Online; last accessed 8-February-2024].

[27] A. Cron. EDA Forms the Basis for Designing Secure Systems. http://surl.li/tvchc, 2016. [Online; last accessed 8-February-2024].

[28] Trustchain security platform. http://surl.li/tvcho. [Online; last accessed 2-December-2023].

[29] M. Yasin et al. *Trustworthy hardware design: Combinational logic locking techniques*. Springer, 2020.

[30] P. Chakraborty et al. SAIL: Analyzing structural artifacts of logic locking using machine learning. *IEEE Trans. Inf. Forensics Security*, pp. 3828–3842, 2021.

[31] L. Alrahis et al. OMLA: An oracle-less machine learning-based attack on logic locking. *IEEE Transactions on Circuits and Systems II: Express Briefs*, pp. 1602–1606, 2021.

[32] A. Alaql et al. SCOPE: Synthesis-based constant propagation attack on logic locking. *IEEE Transactions on Very Large Scale Integration Systems*, pp. 1529–1542, 2021.

[33] L. Alrahis et al. Muxlink: Circumventing learning-resilient mux-locking using graph neural network-based link prediction. In *Proc. DATE*, pp. 694–699, 2022.

[34] D. Sisejkovic et al. Challenging the security of logic locking schemes in the era of deep learning: A neuroevolutionary approach. *J. Emerg. Technol. Comput. Syst.*, pp. 1–26, 2021.

[35] N. Limaye et al. Rescue: Resilient, scalable, high-corruption, compact-key-set locking framework. *IEEE Trans. on Comput.-Aided Design of Integr. Circuits and Syst.*, pp. 2826–2838, 2023.

[36] A. B. Chowdhury et al. Almost: Adversarial learning to mitigate oracle-less ml attacks via synthesis tuning. *in ACM/IEEE Design Automation Conference*, 2023.

[37] F. Wang et al. Security closure of IC layouts against hardware trojans. In *Proceedings of the International Symposium on Physical Design*, pp. 229–237, 2023.

[38] L. Alrahis et al. UNSAIL: Thwarting oracle-less machine learning attacks on logic locking. *IEEE Trans. Inf. Forensics Security*, pp. 2508–2523, 2021.

[39] G. Severi et al. Explanation-Guided backdoor poisoning attacks against malware classifiers. In *Proc. USENIX Security Symposium*, pp. 1487–1504, 2021.

[40] A. Kuppa et al. Adversarial XAI methods in cybersecurity. *IEEE Trans. Inf. Forens. Sec.*, pp. 4924–4938, 2021.

[41] L. Yang et al. CADE: Detecting and explaining concept drift samples for security applications. In *Proc. USENIX Security Symposium*, pp. 2327–2344, 2021.

[42] X. Zhao et al. Exploiting explanations for model inversion attacks. In *IEEE/CVF International Conference on Computer Vision*, pp. 662–672, 2021.

[43] T. Zebin et al. An explainable AI-based intrusion detection system for dns over https (doh) attacks. *IEEE Trans. Inf. Forensics Security*, pp. 2339–2349, 2022.

[44] Z. Yang et al. Versatile multi-stage graph neural network for circuit representation. *Advances in Neural Information Processing Systems*, pp. 20313–20324, 2022.

[45] Y. Zhang et al. Grannite: Graph neural network inference for transferable power estimation. In *ACM/IEEE Design Automation Conference*, pp. 1–6, 2020.

[46] Opencores educational 16-bit MIPS processor. https://shorturl.at/DaJa5, 2013. [Online; last accessed 8-February-2024].

[47] Ibex RISC-V Core. https://encr.pw/FER0u, 2022. [Online; last accessed 8-February-2024].

[48] CEP v2.0 Security Evaluation Targets. https://github.com/mit-ll/CEP, 2019. [Online; accessed 8-February-2024].

[49] mor1kx - an OpenRISC processor IP core. https://shorturl.at/GnoNM, 2022. [Online; last accessed 8-February-2024].

[50] Synopsys TestMAX . https://l1nq.com/ZGOke. [Online; accessed 8-February-2024].

[51] N. Kavand et al. Securing hardware through reconfigurable nanostructures. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pp. 1–7, 2022.

[52] S. D. Chowdhury et al. SimLL: Similarity-based logic locking against machine learning attacks. In *ACM/IEEE Design Automation Conference*, pp. 1–6, 2023.

[53] M. Réau et al. Deeprank-gnn: a graph neural network framework to learn patterns in protein–protein interfaces. *Bioinformatics*, 2023.

[54] X. Xia et al. Self-supervised hypergraph convolutional networks for session-based recommendation. In *Proc. of the AAAI Conf. on Artificial Intelligence*, pp. 4503–4511, 2021.

[55] C. Huang et al. Knowledge-aware coupled graph neural network for social recommendation. *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 4115–4122, 2021.

[56] W. Song et al. Deepmem: Learning graph neural network models for fast and robust memory forensic analysis. In *Proc. of the ACM SIGSAC Conference on Computer & Communications Security*, pp. 606–618, 2018.

[57] M. Shen et al. Accurate decentralized application identification via encrypted traffic analysis using graph neural networks. *IEEE Trans. Inf. Forens. Sec.*, pp. 2367–2380, 2021.

[58] M. Hachman. Chip shortages will continue until 2023. https://rb.gy/24e9bl, 2022. [Online; last accessed 8-February-2024].

[59] Securing Defense-Critical Supply Chains. https://rb.gy/3te29x, February 2022. [Online; last accessed 8-February-2024].

[60] The most cost-effective way to get critical analysis. https://rb.gy/ef0whi.

[61] K. Z. Azar et al. From cryptography to logic locking: A survey on the architecture evolution of secure scan chains. *IEEE Access*, 9:73133–73151, 2021.

[62] M. E. Massad et al. Logic locking for secure outsourced chip fabrication: A new attack and provably secure defense mechanism. *arXiv preprint arXiv:1703.10187*, 2017.

[63] A. Kerckhoffs. La cryptographie militaire. *Journal des Sciences Militaires*, pp. 161–191, 1883.

[64] K. Xu et al. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[65] Z. Ying et al. Gnnexplainer: Generating explanations for graph neural networks. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[66] H. Yuan et al. On explainability of graph neural networks via subgraph explorations. In *Proceedings of the International Conference on Machine Learning*, pp. 12241–12252, 2021.

[67] T. Funke et al. Zorro: Valid, sparse, and stable explanations in graph neural networks. *IEEE Transactions on Knowledge &amp; Data Engineering*, pp. 8687–8698, aug 2023.

[68] D. Luo et al. Parameterized explainer for graph neural network. In *Proceedings of the International Conference on Neural Information Processing Systems*, 2020.

[69] H. Yuan et al. Explainability in graph neural networks: A taxonomic survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 5782–5799, 2023.

[70] P. Veličković et al. Graph Attention Networks. *International Conference on Learning Representations*, 2018.

[71] R. Saleh et al. System-on-chip: Reuse and integration. *Proceedings of the IEEE*, 94(6):1050–1069, 2006.

[72] F. Wang et al. Baseline and protected layouts. https://rb.gy/t5g40y, 2022. [Online; last accessed 8-February-2024].

[73] L. Alrahis et al. Reference benchmarks for *UNSAIL* locking. https://rb.gy/ghgph6, 2020. [Online; last accessed 8-February-2024].

[74] M. C. Hansen et al. Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering. *IEEE Design and Test of Computers*, 16(3):72–80, 1999.

[75] S. Davidson. Notes on ITC'99 Benchmarks. https://tinyurl.com/2fdhx4ns, 1999. [Online; last accessed 8-February-2024].

[76] L. Amarú et al. The EPFL combinational benchmark suite. In *Proc. of International Workshop on Logic & Synthesis*, 2015.

[77] S. Takeshi et al. Asic performance comparison for the iso standard block ciphers. In *Joint Workshop on Information Security*, pp. 485–498, 2007.

[78] OpenCores. Opencores. 16-bit openmsp430 microcontroller. https://encr.pw/3VeTy, 2021. [Online; last accessed 8-February-2024].

[79] D. Doy. Fpga image processing. https://l1nq.com/rohjG, 2019. [Online; last accessed 8-February-2024].

[80] Z. Han et al. Functeller: how well does efpga hide functionality? In *Proc. USENIX Security Symposium*, 2023.

[81] Z. Wu et al. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, January 2021.

[82] P. A. Ruetz. The architectures and design of a 20-mhz real-time dsp chip set. *IEEE Journal of Solid-State Circuits*, pp. 338–348, 1989.

A Appendix

A.1 GNNExplainer

High-level overview. GNNExplainer is a model-agnostic explainable GNN that provides explanations for predictions made by a GNN model on graph-based machine learning tasks (*e.g.,* node classification, link prediction). GNNExplainer takes the following inputs: (i) graph, (ii) features of the graph, (iii) prediction of the graph, and (iv) trained GNN model. In the context of INSIGHT, the (i) input graph refers to an extracted subgraph, *i.e.,* logic gates in the neighborhood of key-gates, (ii) features correspond to functionality/structure-based features extracted for the logic gates in the subgraph, (iii) prediction of the graph is an INV/BUF prediction, and (iv) trained GNN model refers to attention-based GNN trained on the training dataset (§4.4 and §4.5).

GNNExplainer provides explanations in terms of importance scores for nodes and features for a given graph. Consider an input (sub)graph G', its corresponding features X', a trained GNN model T_{GNN}, and the prediction $y = T_{GNN}(G')$. GNNExplainer generates an explanation $(G_S', X_S'^F)$, where $G_S' \subseteq G'$ is the (important) neighborhood in the input graph G'. X_S' represents the features of G_S', whereas $X_S'^F$ represents the set of features masked by the mask F, *i.e.,* $X_S'^F = \{x_i'^F \mid v_i \in G_S'\}$. The importance scores of nodes and features are real numbers in the range of $(0,1)$, with higher scores indicating greater importance of the respective node or feature.

Methodology of GNNExplainer. The importance scores are calculated using mutual information (MI), and GNNExplainer is formalized as an optimization framework, where H refers to entropy and Y refers to predicted label distribution:

$$\max_{G_S'} \mathrm{MI}(Y, (G_S', X_S')) = H(Y) - H(Y \mid G' = G_S', X' = X_S'). \quad (1)$$

For a given (sub)graph G', MI measures how much the probability of predicting y changes when the (sub)graph is restricted to G_S' and its features are restricted to X_S'. For example, if removing v_i from G' decreases the probability of prediction y with a large value, the node v_i is a good explanation, *i.e.,* an important node in (sub)graph G'. The entropy $H(Y)$ in Equation 1 is constant as T_{GNN} is fixed for a given trained GNN. Thus, maximizing mutual information between the predicted label distribution Y and explanation (G_S', X_S') is similar to minimizing conditional entropy $H(Y \mid G' = G_S', X' = X_S')$, which can be expressed as follows:

$$H(Y \mid G' = G_S', X' = X_S') = -\mathbb{E}_{Y \mid G_S', X_S'}[\log P_{T_{GNN}}(Y \mid G' = G_S', X' = X_S')]. \quad (2)$$

where $\mathbb{E}$ is the expected value and $P_{T_{GNN}}$ represents the probability of the trained GNN model T_{GNN} in predicting Y, given that G' is restricted to G_S' and X' is restricted to X_S'.

Thus, the explanation for prediction y is a subgraph G_S' that minimizes the uncertainty of T_{GNN} when the GNN inference/computation is limited to G_S'. In simpler terms, the subgraph G_S' is selected to optimize the understanding of how the GNN arrives at its prediction when focusing only on a subset of (sub)graph G', *i.e.,* G_S' and the subset of features (X_S').

Training process. GNNExplainer has no training process. It provides explanations for the predictions made by GNNs.

Nature of outputs. The explanations are real numbers in the range of $(0,1)$.

A.2 Integration of INSIGHT

INSIGHT can be integrated with existing X(N)OR/MUX-based logic locking techniques, including the 7 learning resilient locking techniques targeted in our work. Next, we discuss the feasibility of integrating INSIGHT with existing locking techniques, focusing on the (i) format of the locked designs compatible with INSIGHT, (ii) additional steps required to be performed on the locked designs, and (iii) software requirements for installing INSIGHT.

Compatible locking techniques. INSIGHT is compatible with X(N)OR/MUX-based locking techniques. For instance, in our work, we showcased the compatibility of INSIGHT with existing X(N)OR-based and MUX-based locking techniques.

Format of locked designs. INSIGHT is compatible with gate-level designs in the Verilog format, mapped to a technology library. Such a format (technology-mapped Verilog designs) is widely adopted in academic research and semiconductor industries for hardware design.

Additional steps. INSIGHT requires no additional steps for locked designs in Verilog format. INSIGHT converts the Verilog designs to graph format and performs explainability-guided GNN-based node classification and post-processing to recover the secret key (§4.7). However, users must manually analyze the explanations provided by INSIGHT to enhance the attack or improve the logic locking technique.

Software requirements. INSIGHT is tested on 2 Linux operating systems (Ubuntu and RedHat). The software requirements are minimal, and INSIGHT requires Python 3.7, Pytorch 1.7.0, and installation of standard Python packages (*e.g.,* networkx, gensim, scikit-learn, and numpy).

Example of integration of INSIGHT with existing locking technique. Consider an example of attacking designs locked using the SimLL technique [52]. SimLL-locked designs, open-sourced by the authors, are provided in BENCH format [52]. To ensure compatibility with INSIGHT, we convert the SimLL-locked designs from BENCH format to technology-mapped gate-level designs in Verilog format. Such a conversion is performed using a custom Python script. Next, we perform graph transformations, *i.e.,* logic synthesis on the locked designs to transform the MUX-based key-gates into Boolean logic gates. Finally, we provide the synthesized locked designs to INSIGHT to obtain predictions and explanations.

A.3 Computational complexity of INSIGHT

The computational complexity of the final architecture of INSIGHT (§4.7) depends on the time complexity of the subgraph extraction, feature extraction, commercial logic synthesis tools utilized for graph transformations, attention-based GNN inference, and GNNExplainer. The subgraph extraction step obtains the subgraph $G'(V',E')$ comprising the neighbors of the target node, *i.e.,* key-gate for a hop-size of r from graph $G(V,E)$. The time complexity of extracting neighbors is $O(|V|+|E|)$, where $|V|$ ($|E|$) is the number of nodes (edges) in the graph $G(V,E)$. The feature extraction step includes extraction of (i) gate-type, (ii) input degree of the nodes in the subgraph, (iii) distance of the nodes in the subgraph from

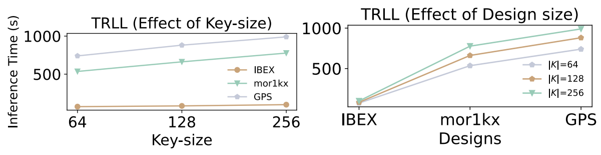
Table 13: Efficacy (KPA (%)) of INSIGHT on TRLL-locked designs synthesized with two different synthesis recipes.

Design	Setting_A	Setting_B
b14_C	79.64	79.54
b15_C	78.90	80.91
b17_C	81.98	80.83
b20_C	80.10	80.12
b21_C	81.45	80.88
b22_C	77.51	78.11
Average	**79.93**	**80.05**

the key-gate (DE), and (iv) a (+/-) sign indicating if the node is in fan-in/fan-out of the key-gate. The time complexity of extracting (i) gate-type is $O(1)$, (ii) input degree is $O(|E'|)$, (iii) DE is $O(1)$, and (iv) sign is $O(1)$. The time complexity of the attention-based GNN is $O(|E'|)$ [81] for a sparse subgraph and $O(|V'|^2)$ for a dense subgraph, where $|V'|$ ($|E'|$) denotes the number of nodes (edges) in the subgraph $G'(V',E')$. The time complexity of GNNExplainer is $O(|E'|)$ [68].

Deriving the time complexity of graph transformations is non-trivial due to the closed-source commercial logic synthesis algorithms, whose complexities are unknown. However, the practical runtime required for the graph transformations is an average of 30 minutes, making the graph transformation step extremely efficient.

A.4 INSIGHT: Additional Results



(a) (b)

Figure 9: Impact of (a) key-size and (b) design size on INSIGHT runtime for IBEX, mor1kx, and GPS designs.

Effect of architecture. The learning resilient locking techniques offer protection against hardware IP piracy for various design types, including arithmetic designs and controllers. Next, we analyze the security of two widely-used classes of designs, *i.e.,* arithmetic designs and controllers.

Arithmetic designs. We choose different architectures of widely-used and prevalent arithmetic designs (multipliers, log2, square-root, square, and adders) and lock them using TRLL. We adopt the standard *leave-one-out* training approach, *i.e.,* when attacking a locked multiplier (adder), we train the GNN using the remaining locked multipliers (adders).

Multipliers. We choose three multiplier architectures (Wallace-tree, Booth, and array) used in digital signal processing applications [82]. The architectures have average KPA of 82.42%, 96.48%, and 99.22%, respectively (Table 14).

Adders. We choose three adder architectures (ripple carry, carry-skip, and kogge-stone). The architectures have an average KPA of 97.26%, 99.22%, and 96.48% (Table 14).

Table 14: Efficacy (KPA (%)) of INSIGHT on architectures of adders and multipliers locked using TRLL [24] for $|K| = 128$.

Adder	Bit-width	KPA (%)
Kogge-stone	64	95.31
	128	99.22
Carry-skip	64	99.22
	128	99.22
Ripple carry	64	99.22
	128	93.75
Average	–	**97.92**
	–	**97.39**

Multiplier	Bit-width	KPA (%)
Booth	64	75.00
	128	89.84
Wallace-tree	64	99.22
	128	93.75
Array	64	99.22
	128	99.22
Average	–	**91.15**
	–	**94.27**

Log2. We choose two log2 architectures (binary search and Newton-Raphson). We observe both architectures having a comparable KPA, *i.e.,* 99.22% for binary search architecture and 98.44% for Newton-Raphson architecture (Table 15).

Square-root. We choose two square-root architectures (non-restoring and Newton-Raphson). We observe both architectures having a comparable KPA, *i.e.,* 96.09% for non-restoring architecture and 99.22% for Newton-Raphson architecture (Table 15).

Square. We choose three architectures (Wallace-tree, Booth, and array). We observe that Booth multiplier-based square design has a comparatively lower KPA (90.62%) than other designs, *i.e.,* 99.22% for array-based square design and 98.44% for Wallace-tree-based square design (Table 15).

Controllers. We choose four controllers from the EPFL benchmark suite [76] (Table 16). We observe an average KPA of 99.02%, indicating that INSIGHT exploits the structural hints in the hardware implementation of the aforementioned controllers when locked using TRLL.

Table 15: Efficacy (KPA (%)) of INSIGHT on log2, square-root, and square locked using TRLL [24] for $|K| = 128$.

Design	Architecture	KPA (%)
Log2	Binary search	99.22
	Newton-Raphson	98.44
Square-root	Non-restoring	96.09
	Newton-Raphson	99.22
Square	Array	99.22
	Booth	90.62
	Wallace-tree	98.44

Table 16: Efficacy (KPA (%)) of INSIGHT on controllers [76] locked using TRLL [24] for $|K| = 128$.

Design	KPA (%)
mem_ctrl	99.22
voter	99.22
i2c	98.44
arbiter	99.22
Average	**99.02**