Titan: Security Analysis of Large-Scale Hardware Obfuscation Using Graph Neural Networks

Likhitha Mankali, Lilas Alrahis, *Member, IEEE,* Satwik Patnaik, *Member, IEEE,* Johann Knechtel, *Member, IEEE,* and Ozgur Sinanoglu, *Senior Member, IEEE*

***Abstract*—Hardware obfuscation is a prominent design-for-trust solution that thwarts intellectual property (IP) piracy and reverse-engineering of integrated circuits (ICs). Researchers have proposed several large-scale obfuscation techniques that achieve high output corruption—thus offering resilience against seminal attacks along with acceptable power, performance, and area overheads. However, the research community has primarily evaluated hardware obfuscation on relatively small scales of obfuscation (*i.e.,* a fixed number of obfuscated components). Moreover, prior art caters toward specific schemes based either on gate obfuscation or interconnect obfuscation, *i.e.,* two prominent types of hardware obfuscation. The former shortcoming suggests focusing on large-scale obfuscation schemes, and the latter suggests the need for a holistic assessment framework.**

In this work, we propose Titan, a holistic framework considering large-scale gate and interconnect obfuscation schemes. More specifically, we propose a graph neural network (GNN)-based attack framework that is trained to exploit structural and functional properties of any secured circuit to recover its obfuscated components. We evaluate Titan on various obfuscation schemes, considering selected ITC-99 benchmarks with up to 50% obfuscation scale, *i.e.,* up to 21,326 obfuscated components. We observe a substantial information leakage through structural and functional properties of secured designs even for large-scale obfuscation. We quantify the information leakage in two ways: first, an average reduction of Hamming distance (HD, a well-established metric for attack evaluation) by 23.27 and 16.19 percentage points over the baseline of random guessing for gate and interconnect obfuscation, respectively; second, an average recovery of 63.40% and 77.94% of obfuscated components for gate and interconnect obfuscation, respectively. Importantly, these results are superior to six state-of-the-art attacks. We will open-source our framework and associated artifacts to enable reproducibility and foster future work.

***Index Terms*—Hardware Obfuscation, Graph Neural Networks**

Manuscript received April 20, 2022; revised September 18, 2022; accepted October 19, 2022. The work of Likhitha Mankali was supported by the Global Ph.D. Fellowship at New York University/New York University Abu Dhabi (NYUAD). This work is supported in part by the Center for Cyber Security at NYUAD. Besides, this work was carried out in part at the High Performance Computing facility at NYUAD. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Debdeep Mukhopadhyay. (*Corresponding author: Likhitha Mankali*)

Likhitha Mankali is with the Department of Electrical and Computer Engineering, Tandon School of Engineering, New York University, USA (email: lm4344@nyu.edu).

Lilas Alrahis, Johann Knechtel, and Ozgur Sinanoglu are with the Division of Engineering, New York University Abu Dhabi, UAE (email: lma387@nyu.edu; johann@nyu.edu; ozgursin@nyu.edu).

Satwik Patnaik is with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, USA (email: satwik.patnaik@tamu.edu).

Digital Object Identifier 10.1109/TIFS.2022.3218429

TABLE I
CAPABILITIES OF STATE-OF-THE-ART ATTACKS ON HARDWARE OBFUSCATION AND OUR PROPOSED ATTACK

	Unscalable	Obfuscated Type-Specific	Restricted Capabilities	Returns Invalid Netlist
SAT [5]	✓	✗	✗	✗
AppSAT [6]	✓	✗	✗	✗
Redundancy [7]	✓	✗	✗	✗
SWEEP [8]	✗	✓	✓	✗
SCOPE [9]	✗	✓	✓	✗
UNTANGLE [10]	✗	✓	✓	✗
OMLA [11]	✗	✓	✓	✓
Proposed Attack (Titan)	✗	✗	✗	✗

Note that ✗ indicates no related limitation for the attack, whereas ✓ indicates some limitation.

I. Introduction

A. Globalization of IC Supply Chain and Design-for-Trust

INTEGRATED circuit (IC) manufacturing costs have skyrocketed with aggressive miniaturization of advanced technology nodes. Recently TSMC announced an investment of $23 billion for commissioning a foundry for the state-of-the-art 3nm technology node [1]. The inability to commission and maintain foundries for advanced technology nodes have driven many semiconductor design companies (e.g., Apple, Qualcomm) into adopting a fabless business model [2]. In such a business model, the design of intellectual property (IP) is carried out by a *trustworthy* design house, while the fabrication, testing, and packaging are outsourced to third-party entities located in different geographical locations. Outsourcing of critical steps of the IC supply chain to potentially *untrustworthy* entities enables attackers to gain access to the underlying design, either in the form of a design netlist, physical layout, or a packaged IC. Consequently, a plethora of security issues emerged, ranging from design IP piracy, unauthorized overproduction of ICs, to implantation of malicious logic [3]. Over the past decade, researchers proposed several *design-for-trust* techniques to address the aforementioned security concerns [4]. Hardware obfuscation is one such technique that protects an IC against design IP piracy and reverse-engineering.

B. Hardware Obfuscation

Hardware obfuscation structurally modifies the underlying design to make it unintelligible (to an adversary) while maintaining the intended functionality [12]. The security goal of hardware obfuscation is to ensure that an adversary recovers an erroneous design that deviates from the original design in terms of output corruptibility.[1] Generally, hardware obfuscation is categorized into gate versus wire/interconnect obfuscation. In gate obfuscation, a logic gate is obfuscated using look-alike gates [13], [14] or by threshold voltage-dependent configuration of gates [15]. In interconnect obfuscation, the inputs of a logic gate are obfuscated using some dummy wires or vias [16], [17].

The research community has proposed various attacks questioning the efficacy of hardware obfuscation techniques. However, most of these attacks specifically target one of the obfuscation types (*i.e.,* gate or interconnect obfuscation). More importantly, although existing attacks deobfuscate specific instances of hardware obfuscation,[2] *they have been evaluated primarily on small-scale obfuscation, where only a limited set of gates/interconnects are obfuscated in a design.* Attacking large-scale obfuscation where most of the design-related information (*i.e.,* topology and functionality) poses significant challenges and remains an open problem for the hardware security community.

C. Limitations of State-of-the-Art Attacks

Now, we discuss the limitations of the state-of-the-art attacks on hardware obfuscation which serve as the motivation of our work. We review the attacks in Section II-B and highlight the limitations of state-of-the-art attacks in Table I.
Unscalable Attacks. Existing attacks such as the Boolean satisfiability (SAT)-based attack [5], approximate satisfiability-based attack (AppSAT) [6], and the redundancy attack [7] fail to deobfuscate designs within a limited time (e.g., 48 hours) due to scalability issues. This is especially true for large-scale obfuscation (e.g., 30–50% of gates or interconnects being obfuscated). Our experiments on obfuscated ITC-99 benchmarks for 30–50% obfuscation highlight that state-of-the-art attacks do not scale for large-scale obfuscation (Section V-D).
Obfuscation Type-Specific Attacks. Existing attacks like SWEEP [8], SCOPE [9] and UNTANGLE [10] specifically target interconnect obfuscation. On the other hand, an attack like OMLA [11] specifically targets gate obfuscation. These attacks have not been extended to other types of obfuscation. More importantly, *neither of these state-of-the-art attacks applies to unified obfuscated designs, i.e., designs where both gate and interconnect obfuscation are applied together.*
Restricted Capabilities. Existing attacks against interconnect obfuscation have restricted capabilities, limiting their applicability. For instance, SWEEP and SCOPE are designed to consider only two possible wires (*i.e.,* one dummy wire and one true wire) for every obfuscated interconnect. UNTANGLE [10] attacks obfuscation techniques where only a single input is obfuscated for a given logic gate. However, advanced interconnect obfuscation schemes obfuscate all the inputs of a given gate, *i.e.,* not only a single input [16].

[1]Output corruptibility, quantified by Hamming distance (HD), represents the average bit-level mismatch between the outputs of the original and erroneous design recovered by an adversary on the application of a large (e.g., 50,000) set of input patterns.

[2]*Deobfuscation* is defined as inferring the true functionality of the logic gate (for gate obfuscation) or the true connections between logic gates (for interconnect obfuscation).

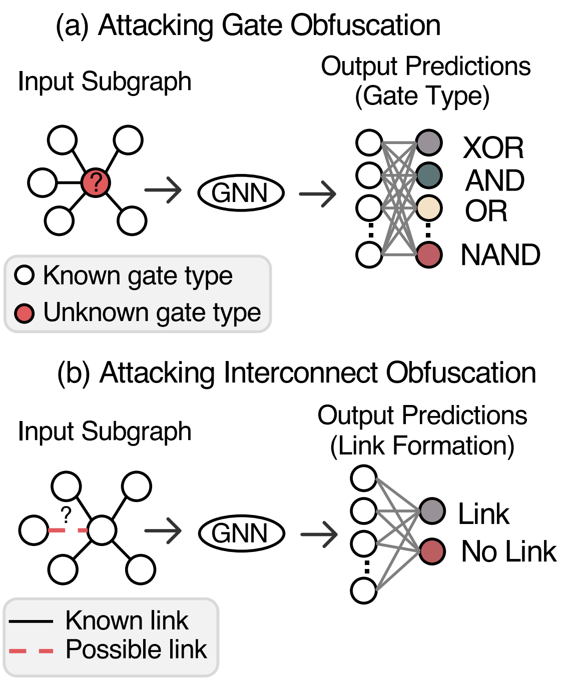
(a) Attacking Gate Obfuscation
Input Subgraph
Output Predictions (Gate Type)
?
GNN
XOR
AND
OR
NAND
Known gate type
Unknown gate type
(b) Attacking Interconnect Obfuscation
Input Subgraph
Output Predictions (Link Formation)
?
GNN
Link
No Link
Known link
Possible link

Fig. 1. High-level overview of our holistic attack framework (Titan) that is capable of tackling large-scale gate and interconnect obfuscation.

Returns Invalid Netlist. Existing attack on gate obfuscation, such as OMLA, considers either only two-input gates or only single-input gates to be possible options for an obfuscated gate. Consequently, an invalid netlist is returned when both two-input and single-input gates are used for obfuscating gates. For example, OMLA may predict a two-input gate, such as NAND, NOR, etc., in place of a one-input gate, such as INV. This misprediction results in floating nets in the design and results in an invalid netlist.

D. Research Challenges

The above discussion on limitations shows that there is a need for a generic attack framework that can evaluate the security of large-scale hardware obfuscation using any kind of obfuscation and any kind of practical gates and interconnections. Developing such an automated and practical attack poses the following research challenges (**RC**).

RC1 *Handling complex obfuscation.* As discussed earlier, existing attacks can tackle only specific instances of small-scale obfuscation. However, with the increased complexity of obfuscation, the attacks demonstrate several scalability issues. Hence, a scalable attack is required to evaluate the security of large-scale obfuscation.

RC2 *Exploiting a common vulnerability.* Gate and interconnect obfuscation schemes share a common goal but follow different implementation strategies. Therefore, developing a generalized attack methodology requires the identification (and exploitation) of common vulnerabilities to both types of obfuscation, especially for challenging attack scenarios where both types are applied together.

RC3 *Handling a variable number of plausible Boolean functions/interconnects.* An attack that can consider an arbitrary number of plausible nets/Boolean functions is required to handle different obfuscation implementations.

E. Our Research Contributions

We address the aforementioned research challenges by devising and implementing a unified attack framework, Titan, to tackle large-scale interconnect and gate obfuscation. We recover portions of the obfuscated design by either predicting the type of the logic gate (for gate obfuscation) or the true connections to a logic gate (for interconnect obfuscation). To that end, we use graph neural networks (GNNs) to capture structural and functional hints in a design. The GNN learns the topology and composition of gates and interconnects to predict the obfuscated nets/gates of some design under attack. Predicting the true connections for interconnect obfuscation can be mapped to link prediction whereas predicting the gate type can be mapped to node classification. The primary contributions of our work are as follows.

- We devise, implement, and evaluate a framework for GNN-based subgraph classification that helps in predicting the type of gate (for gate obfuscation) and the missing connections (for interconnect obfuscation), as shown in Fig. 1. We develop a unified subgraph-based learning method to solve both the tasks of link prediction and node classification within a single framework (Section IV-D). We incorporate tailored techniques to predict challenging obfuscation cases.
- We implement an oracle (working chip)-guided greedy algorithm as a post-processing step to enhance further the accuracy of GNN outputs (Section IV-F).
- We analyze and evaluate the state-of-the-art attacks on selected ITC-99 benchmarks for varying degrees of obfuscation (10%–50%) and provide a comprehensive comparison of their performance with our proposed attack (Section V-D). We obtained an average HD of 15.71% for the attack on interconnect obfuscation and 18.84% for the attack on gate obfuscation, which indicates that we recovered the partially functional netlist. We recovered an average of 77.94% and 63.40% of obfuscated components in interconnect and gate obfuscation, respectively. Our results are superior than the considered state-of-the-art attacks.
- We will open-source Titan and associated artifacts to enable reproducibility and foster future research.[3]

Paper Outline. The organization of the paper is as follows. We review prior obfuscation techniques and attacks and provide a background on GNNs in Section II. We present the threat model in Section III and expound on the proposed attack framework in Section IV. We discuss experimental results in Section V, provide some discussion in Section VI and conclude our work in Section VII.

II. Background and Preliminaries

Here, we discuss the background related to hardware obfuscation techniques, state-of-the-art attacks, and GNNs. Also, we list the commonly used notations and abbreviations in Table II.

[3]https://github.com/DfX-NYUAD/Titan

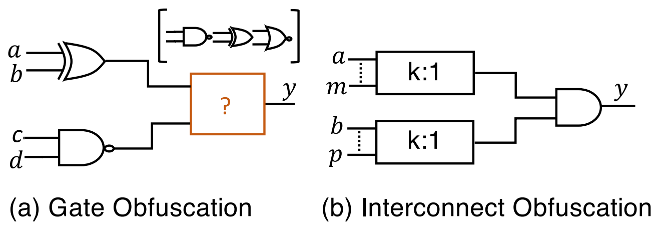

Fig. 2. Illustration of (a) gate and (b) interconnect obfuscation.

TABLE II
Commonly used abbreviations and notations

Term	Definition	Term	Definition
POs	Primary outputs	G	A graph
PIs	Primary inputs	$G^{'}$	A subgraph induced from G
IN	In-degree	$X^{'}$	Feature matrix of $G^{'}$
GNN	Graph neural network	V	Set of nodes in G
GIN	Graph isomorphic network	$V^{'}$	Set of nodes in $G^{'}$
MLP	Multi-layer perceptron	r	#Hops
X_f	Functionality feature matrix of G	$X^{'}_f$	Functionality feature matrix of $G^{'}$
E	Observed links in G	n	#Nodes in graph G
h^l_v	Embedding of v in l^{th} layer	$h^{[0:L]}_{G'}$	Final embedding of $G^{'}$
L	#GNN layers	K	Key
$\mathcal{N}(v)$	Neighborhood of node v	p	#Target nodes in a $G^{'}$
s	Query to GNN	$\mathcal{L}$	Likelihood
HD	Hamming distance	$X^{'}_d$	Distance labeling matrix of $G^{'}$
$E^{'}$	Observed links in subgraph $G^{'}$	A	Adjacency matrix of G
N	#Obfuscated gates/nets	S	Set of predict. for obfus. gates/nets
$\mathcal{O}$	Oracle	$\mathcal{R}$	Obfuscated design
$\mathcal{Y}$	Final recovered design	W.l.o.g	Without loss of generality

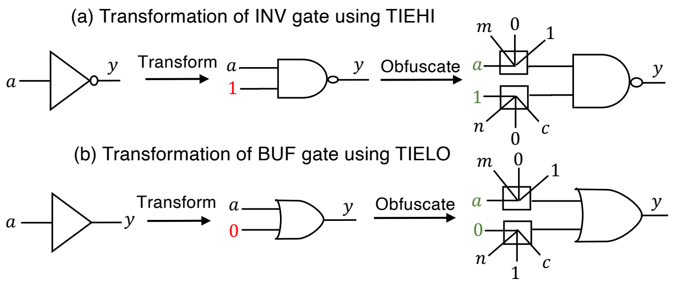
(a) Transformation of INV gate using TIEHI
a
y
Transform
a
1
y
Obfuscate
a
1
m
0
1
n
0
c
y
(b) Transformation of BUF gate using TIELO
a
y
Transform
a
0
y
Obfuscate
a
0
m
0
1
n
1
c
y

Fig. 3. Illustration of conversion of an inverter gate (INV) and a buffer gate (BUF) to other two-input gates using TIEHI and TIELO signals. The inputs highlighted in green represent the true connections.

A. Prior Hardware Obfuscation Techniques

Gate Obfuscation. Researchers have proposed various gate obfuscation a.k.a. gate camouflaging techniques [13]–[15]. For example, researchers utilized dummy contacts to construct look-alike gates [14]. These look-alike gates can implement three Boolean functions (NAND, NOR, or XOR) as shown in Fig. 2(a). Researchers utilized threshold voltage-based switches which can implement six Boolean functions (NAND, AND, NOR, OR, XOR, and XNOR) [15].

Interconnect Obfuscation. Researchers proposed a technique [16], where each to-be-obfuscated interconnect/wire is obfuscated by three additional dummy wires [16]. An example of interconnect obfuscation is illustrated in Fig. 2(b). The technique in [16] considers four nets for obfuscation, where two dummy nets are fixed to logic 0 and 1 using TIELO and TIEHI cells, respectively, to manage power, performance, and area overheads. Since designs inherently do not contain many TIEHI or TIELO cells, the authors convert some inverters (INV) and buffers (BUF) to two-input gates with one input fixed/tied to logic 0 or 1, as shown in Fig. 3. In [17], two types of vias/contacts are used: (i) conductive vias/contacts based on magnesium (Mg), and (ii) non-conductive vias/contacts based on magnesium oxide (MgO). The authors demonstrate that Mg oxidizes to MgO during the delayering of the IC (a common step in IC reverse-engineering), thereby transforming the Mg vias to MgO vias. Such a transformation makes it difficult for an adversary to distinguish between real and dummy vias.

Logic Locking. Initially proposed in [18], logic locking is a type of hardware obfuscation that inserts additional logic, *i.e.,* key-gates controlled by a secret key. The locked design functions correctly only upon applying the correct secret key. With an incorrect key, a locked design generates incorrect functionality and produces corrupted outputs. Initially, researchers proposed logic locking techniques that augmented key-gates to facilitate high output corruption [18], [19]. However, these techniques were circumvented by the SAT-based attack [5]. Subsequently, researchers proposed various techniques to thwart SAT-based attacks.

Such techniques, also known as SAT-resilient techniques, are categorized into: (i) point function-based locking or provably-secure logic locking (PSLL), (ii) SAT-hard locking, (iii) cyclic locking, and (iv) scan locking. In (i), researchers augmented the design with point functions that restrict the SAT-based attacks to prune incorrect keys in every attack iteration [20], [21]. In techniques under (ii), researchers locked the design using SAT-hard instances, such as switch-box-based routing networks [22], [23], that increase the time taken by each iteration of the SAT-based attack. The techniques under (iii) introduce combinational loops to hinder SAT-based attacks [24]. Finally, the techniques under (iv) obfuscate the scan data, limiting the controllability and observability of internal nets, thereby hindering oracle access [25].

However, all the aforementioned locking techniques are vulnerable to attacks. PSLL techniques are vulnerable to structural attacks [26]. SAT-hard techniques such as Full-Lock [22] and InterLock [27] have been thwarted by machine learning (ML)-based attacks such as neural network-guided SAT [28] or UNTANGLE [10]. Cyclic locking has been broken by advanced SAT-based attacks tailored for cyclic structures [29]. Finally, scan-locking techniques have also been broken by other SAT-based attacks, such as ScanSAT [30].

Apart from the aforementioned (and broken) techniques, we note that large-scale obfuscation is inherently resilient to SAT-based attacks, as shown in Sec. V-D. However, security analysis of large-scale obfuscation against structural or ML-based attacks remains an open problem for the hardware security community, and our work attempts to close this gap.

B. Prior Attacks on Hardware Obfuscation

Various attacks targeting hardware obfuscation schemes consider a logic locking scheme for modeling. For instance, state-of-the-art attacks [5]–[9] model hardware obfuscation as multiplexer (MUX)-based locking. We model obfuscation as MUX-based locking [31], where the possible nets/gates for each obfuscated component are connected to a MUX and the select lines of MUX are connected to so-called *key-bits*.

SAT-Based Attack [5] constructs a miter circuit using two copies of the obfuscated design with different key-bits. This miter circuit is fed to a SAT solver to produce a distinguishing input pattern (DIP), a special input pattern that generates differing outputs for at least two different keys. The DIP is provided as an input to a working chip (a.k.a. an *oracle*) and the corresponding outputs are recorded, which help in pruning the incorrect keys. **AppSAT** [6] is a modified version of the SAT-based attack that returns an approximately deobfuscated design. **SAIL** [32] is a ML-based attack that exploits the localized structural changes in the design (caused by deterministic procedures invoked by synthesis) to recover obfuscated components. **SWEEP** [8] and **SCOPE** [9] are constant-propagation attacks that exploit synthesis-based information (e.g., area, power) to recover the obfuscated design. **Redundancy attack** [7] is based on the premise that the to-be-protected design should be completely testable, *i.e.,* without *untestable faults*.[4] The attack retrieves obfuscated components by checking for untestable faults. **UNTANGLE** [10] is an ML-based attack targeting the Interlock obfuscation [27]. UNTANGLE uses GNN-based link prediction to predict the correct connections in obfuscated designs. **OMLA** [11] is another ML-based attack that uses GNN-based subgraph classification to predict the correct functionality of the obfuscated gates.

C. Graph Neural Networks (GNNs)

GNNs are the preferred choice for deep learning on graph-structured data. A GNN takes a graph as an input, performs *neighborhood aggregation*, and generates a vector representation (*embedding*) for each node in the graph [34]. The obtained embeddings capture the structure of the graph, the position of each node within the graph, and initial node features. Let $G(V, E)$ denote a graph with node set V and edge set E. Further, A denotes the $n \times n$ adjacency matrix of G where n represents the total number of nodes in G. Each node $v \in V$ is assigned an initial feature vector x_v that captures its properties. A round of neighborhood aggregation can be abstracted as follows, where $\mathcal{N}(v)$ denotes the direct neighborhood of a node v, and h_v^l represents the embedding of v at the l^{th} round. h_v^0 is initialized with x_v.

$$a_v^l = Aggregate^l(\{h_u^{l-1} : u \in \mathcal{N}(v)\}) \tag{1}$$

$$h_v^l = Update^l(h_v^{l-1}, a_v^l) \tag{2}$$

First, the information from $\mathcal{N}(v)$ is collected using the *Aggregate* function to generate an embedding a_v^l for the layer l. Next, the *Update* function updates the embedding of v by combining its previous state h_v^{l-1} with a_v^l. After L rounds of neighborhood aggregation, the obtained node embeddings can be used to perform multiple tasks such as node classification, (sub)graph classification, link prediction, etc. In our work, we

[4]Untestable faults correspond to the faults in a design that cannot be propagated or excited to an observable point in the design [33].

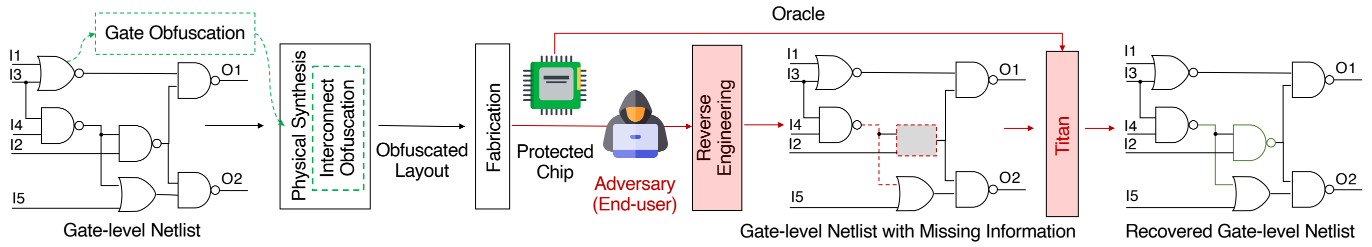

Fig. 4. A designer obfuscates the design either using gate obfuscation, interconnect obfuscation, or both. Gate obfuscation is implemented at the logic synthesis step, while interconnect obfuscation is implemented at the physical synthesis step. Finally, the obfuscated layout is sent to the trusted foundry for fabrication. The end-user procures the IC from the open market and reverse-engineers it to obtain a gate-level netlist with some missing obfuscated gates and/or wires. An adversary aims to recover the missing information using hints from the non-obfuscated parts in the design and the working chip (oracle).

employ GNNs to perform node classification as well as link prediction, as discussed in the subsequent subsections.

Subgraph-Based Learning for Node Classification. Researchers demonstrated that *subgraph-based learning* achieves better node classification performance than methods operating on single nodes [35]. In such a formalism, a subgraph $G^{'}$ is first extracted around v. Then, traditional neighborhood aggregation on $G^{'}$ generates an embedding for each node in the subgraph. Next, a *Read-out* function extracts a subgraph-level embedding which represents the subgraph as a whole and employs it as the target node embedding. In short, the problem of classifying a node v gets mapped to classifying the subgraph surrounding the node, *i.e.,* $G^{'}$.

Subgraph-Based Learning for Link Prediction. Given a subset of observed links $E \in D$, the goal of link prediction is to identify the unobserved true connections S referred to as *target links*, where $D = E+S$. Link prediction methods assign scores to all possible links to determine the possibility of those edges belonging to D. The GNN learns link formation by extracting subgraphs around the target links, which capture the structure and properties of the nodes surrounding the link and automatically generates a vector representation that embeds the properties of the link. The obtained representation can then be used for link prediction [36].

III. Adversarial Model and Goals

We illustrate the threat model of Titan in Fig. 4. We outline the resources and capabilities available to an adversary, which have been widely adopted by researchers in the hardware obfuscation community [14], [37]. We also state here that the adversary in this work is the *end-user*.

- An adversary obtains the obfuscated design by reverse-engineering the chip. They can either perform the reverse-engineering on their own or delegate the task to companies that perform on-demand reverse-engineering [38]. We consider that the end-user is equipped with state-of-the-art reverse-engineering tools and can procure multiple copies of the to-be-attacked chip from the open market.
- An adversary possesses a working chip (a.k.a. oracle) procured from open market. An adversary can apply any number of inputs to the chip and record the corresponding outputs. Additionally, an adversary is not restricted in the number of queries they can make to an oracle.
- An adversary has the knowledge of the targeted obfuscation technique which is consistent with Kerckhoff's principle and all prior work on hardware obfuscation [14], [37].
- An adversary can distinguish between regular and obfuscated gates. Obfuscated gates have distinguishable lithography patterns and dimensions which can be captured during the reverse-engineering process [14], [37]. Similarly, obfuscated interconnects exhibit distinct physical patterns [16].

To summarize, an adversary readily understands the type and number of obfuscated components in the design under attack, but not their true functionality or connectivity. Thus, the goal of an adversary is to infer the true functionalities of obfuscated gates/interconnects.

IV. Titan: A GNN-Based Attack Framework Against Large-Scale Hardware Obfuscation

A. Why Graph Neural Networks?

Due to the increasing complexity of circuit design, the industry follows design practices that facilitate the concept of reuse, commonly known as system-on-chip (SoC) design [39]. Different ICs utilize reusable components called IP cores to speed up the design and development stages. IP reuse can be implemented at the platform, chip, or block level. Therefore, one can observe repeated sub-circuits across and even within ICs. Based on this reality, we argue that hints about the structure and functionality of an obfuscated circuit can be inferred from a library of similar designs. We further argue that one can leverage ML algorithms to capture and learn the composition of gates and interconnects in a design (or design library) to decipher information regarding the obfuscated design.

A circuit must be represented either in a structured data form (*i.e.,* vector, matrix, etc.) for traditional ML techniques or as a graph for GNNs. For the former, feature engineering is needed to convert the circuit format into a fixed tensor form. However, for the latter, the circuit can be naturally represented as a graph and processed by a GNN. A GNN model takes the graph (*i.e.,* circuit), automatically generates an embedding that captures its properties (topology and functionality), and performs classification tasks in an end-to-end manner.

B. Modeling Netlists as Graphs

The first step of Titan is to represent the gate-level netlists of obfuscated designs as graphs to facilitate GNN-based learning.

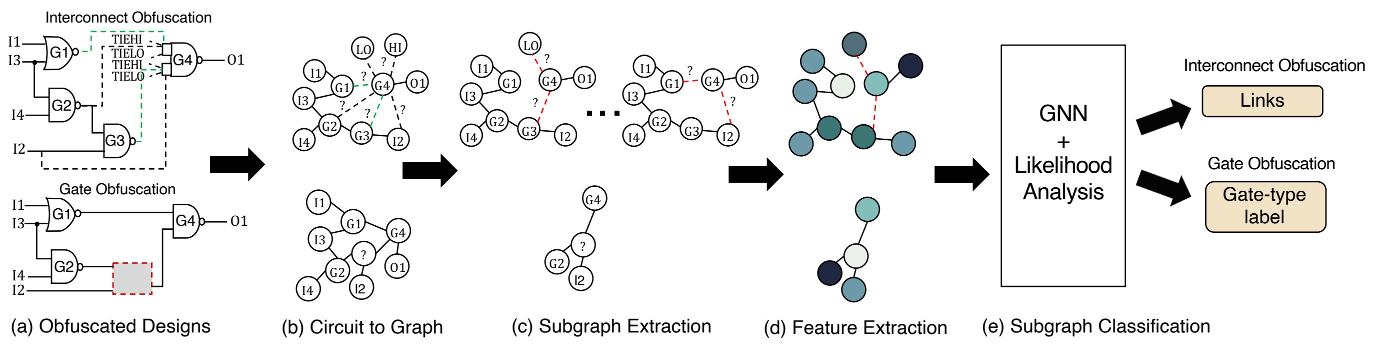

Fig. 5. Overall flow of the subgraph-classification-based attack on interconnect and gate obfuscation. The links in green and red color indicate true and dummy connections respectively.

We model a netlist as an undirected graph $G(V, E)$. Note that we use an undirected format for better representation capability [40]. Here, V denotes the set of vertices, *i.e.,* gates, primary inputs (PIs), primary outputs (POs), TIEHI cells (providing logic 1 signals), and TIELO cells (providing logic 0 signals) in the circuit, and E denotes the set of edges, *i.e.,* the wires. In the case of interconnect obfuscation, E does not include the obfuscated nets; the true nature or connectivity has to be predicted. Therefore, the obtained graph of an interconnect-obfuscated design is considered an incomplete graph with missing links.

C. Problem Formulation

Attack on Interconnect Obfuscation. An adversary aims to predict the correct connectivity in the obfuscated design (*i.e.,* the missing links in the graph) based on the available connections among the nodes. As discussed in Section II-C, the prediction of missing links in an incomplete graph (*i.e.,* link prediction) can be solved using subgraph classification. Thus, we formulate the problem of reverse-engineering the obfuscated interconnects as a subgraph classification task.

Attack on Gate Obfuscation. The attacker aims to predict the true type of obfuscated gates based on the available information of other gates and the structure of the design. As discussed in Section II-C, this problem is equivalent to node classification and can be tackled using subgraph classification.

D. GNN-Based Attack Framework: Subgraph Classification

We develop a framework for GNN-based subgraph classification applicable to interconnect and gate deobfuscation. Fig. 5 summarizes the steps of the attack. First, we extract subgraphs around the target nodes (for gate obfuscation) or target links (for interconnect obfuscation). Later, these subgraphs are given as input to the GNN for subgraph classification. For each query s of the obfuscated gates/nets, the subgraph-classification framework outputs the likelihood for each corresponding class $\ell_s \in \mathbb{R}^{1\times m}$. Here, m represents the number of classes, and each likelihood score ranges between 0 to 1. For interconnect obfuscation, two classes are considered, *i.e., link* or *no link*. In the case of gate obfuscation, without loss of generality (w.l.o.g.) but accounting for state-of-the-art obfuscation schemes [15], eight exemplary classes are considered, *i.e.,* INV, BUF, AND, NAND, NOR, XOR, OR, and XNOR.

Algorithm 1: Inference for Subgraph Classification

```
Input: $G(V,E)$;$X_f$; obfuscated components $\mathcal{S}$, #hops $r$;
Output: Likelihood $\mathcal{L}$ for the obfuscated components $\mathcal{S}$;
1  for $s \in \mathcal{S}$ do
       /* Subgraph extraction */
2      $G^{'}(V^{'}, E^{'}) \leftarrow$ EXTRACT$(G(V,E), s, r)$
       /* Functionality feature vector extraction
          for the nodes in subgraph $V^{'}$ */
3      $X_f^{'} \leftarrow X_f[V^{'}]$
       /* Distance labeling for nodes in subgraph
          for gate and interconnect obfuscation,
          respectively */
4      $X_d^{'} \leftarrow$ DISTANCE-LABEL$\left(G^{'}, s\right)$
       /* Final feature vector for the subgraph */
5      $X^{'} \leftarrow$ CONCATENATE$\left(X_d^{'}, X_f^{'}\right)$
       /* Initializing the nodes with feature
          vector */
6      for $v \in V^{'}$ do
7          $h_v^0 \leftarrow x_v^{'}$
       /* Passing the node information through $L$
          GNN layers */
8      for $l$ = 1 to $L$ do
9          for $v \in V^{'}$ do
10             $\mathcal{N} \leftarrow$ NEIGHBORS$(v)$
11             $h_v^l \leftarrow$ MLP$^l\left(h_v^{l-1} + \sum_{u\in\mathcal{N}} h_u^{l-1}\right)$
           /* Passing the node information at each
              layer through READOUT */
12         $h_{G^{'}}^l \leftarrow$ READOUT$\left(h_v^l | v \in G^{'}\right)$
       /* Concatenate all the outputs of GNN layers
          */
13     $h_{G^{'}}^{[0:L]} \leftarrow \left[h_{G^{'}}^0, h_{G^{'}}^1, \ldots\ldots, h_{G^{'}}^L\right]$
       /* Final GNN likelihood output */
14     $\ell_s \leftarrow$ DENSE_LAYER$\left(h_{G^{'}}^{[0:L]}\right)$
15     $\mathcal{L}$.APPEND$(\ell_s)$
16 return $\mathcal{L}$
```

Algorithm 1 describes the pseudo-code of the subgraph classification. As indicated, we propose a unified attack framework for interconnect and gate obfuscation, but it is important to note that devising such a framework entails additional challenges, especially when tackling state-of-the-art obfuscation schemes. Hence, we devise and implement dedicated techniques for the subgraph and feature extraction steps of the framework. These steps are discussed in detail next.

TABLE III
COMPARISON OF GNN-BASED LINK-PREDICTION PRECISION WITH AND WITHOUT CONSIDERING COMBINATIONS OF LINKS FOR 10% INTERCONNECT-OBFUSCATED DESIGNS

Technique	b14_C	b15_C	b20_C	b21_C	b22_C
Without combination of links	54.72	71.03	72.77	71.978	74.38
With combination of links	63.75	77.95	77.12	79.48	80.97

Attack on Interconnect Obfuscation. First, note that in traditional GNN-based link-prediction methods, subgraphs are extracted around the two ends of a link in question. During GNN testing, all test examples do not contain the target links as they are unknown beforehand. Thus, even during training, target links are removed from the graph to prevent the GNN from over-fitting the training data by only checking whether some target link exists or not. Second, for our objective, traditional link prediction is not sufficient, as explained next.
Challenges. There are dedicated challenges for tackling interconnect-obfuscation schemes [16], as follows.
Challenge 1. *Analyzing multiple obfuscated links for a single node.* In the state-of-the-art interconnect-obfuscation scheme of interest [16], all the inputs to a gate are obfuscated. Thus, when the GNN extracts a subgraph for a missing link that is an input to a gate, it is unaware of the other input connections to this gate as they are obfuscated as well. We solve this challenge by considering all the possible combinations of obfuscated input links to a given gate.

For example, consider the circuit in Fig. 6(a). Here, a two-input gate has its inputs obfuscated using a total of 4 nets each; hence, we have to consider 16 possibilities, *i.e.,* subgraphs. For 10% interconnect obfuscation for instance, by training the GNN on the combination of links, we observe an improvement of 6.88 percentage points (pp) for average precision over the results obtained by training the GNN only on a single link (Table III).
Challenge 2. *Analyzing connections from TIE cells.* As we indicate in Fig. 6(a), TIEHI and TIELO nodes, or TIEHI/LO nodes for short, are possible candidates for obfuscated links. To predict whether a TIEHI/LO node is a likely input for a given obfuscated node, a subgraph around the obfuscated node and the TIEHI/LO node must be extracted (see s_{16}). However, the TIEHI/LO nodes do not provide the GNN with meaningful structural information. Further, whenever a TIEHI/LO node is utilized as input to a gate, the type of the gate (*i.e.,* Boolean functionality) is possibly transformed due to such constant logic 0 or 1 connection. For example, when logic 1 is connected as an input to a NAND gate, the gate operates as an inverter. Therefore, to capture these two properties of the TIEHI/LO connections (*i.e.,* lack of structural information and possible transformation of gate-types), we transform the obfuscated node-type based on the type of TIE node, when applicable, as demonstrated in the figure.
Subgraph Extraction. Consider an input graph G with nodes $\{s_1, s_2, ..s_p\}$ as targets for the combination of missing links s for which the likelihood is to be predicted. $G^{'}$ denotes the enclosed subgraph (line 2 in Algorithm 1) within the r-hops neighborhood around the target nodes (*i.e.,* all the nodes that are r hops away from any of the targets). Let $V^{'}$ represents the set of vertices in $G^{'}$, where any node $v \in V^{'}$ follows the condition $d(v, s_t) <= r$ and where s_t is the target node and d is the shortest distance between nodes v and s_t. An example of one-hop subgraphs is illustrated in Fig. 6(a).
Feature Embedding. Fig. 6(b) illustrates the feature vector extraction for a given node $v \in V^{'}$ (*i.e.,* $x_v^{'}$). The features capture the function and structure of the nodes in subgraphs (lines 3–6 in Algorithm 1) as follows.

- *Functionality:* The feature vector for each node in the graph is one-hot encoded with the gate type (AND, NAND, NOR, etc) or PI/PO or TIEHI/LO pin. W.l.o.g., the length of the feature vector is 12 bits.
- *Distance Labeling:* This technique is used to distinguish the target nodes from their surrounding nodes in the subgraph. Consider p targets, $s_1, s_2, s_3, ...s_p$ in a subgraph, the distance-labeling vector for a node v in the subgraph is represented as $(d(v, s_1), d(v, s_2), ...d(v, s_p))$. The target nodes are encoded as $(0, 0, ...0)$. W.l.o.g., we set the maximum distance value to 10 (*i.e.,* any node at a distance > 10 is encoded as 10). Each distance value is one-hot encoded using 11 bits, where each bit represents a value from 0 to 10. The length of the feature vector depends on the maximum number of inputs to any gate in design, e.g., for a maximum of $p - 1$ inputs to some gate in design, the length is $11 \times p$. The final node feature matrix $X^{'}$ is obtained by concatenating the functionality feature vector matrix $X_f^{'}$ and the distance labeling vector matrix $X_d^{'}$, where $x_{d_v}^{'}$, $x_{f_v}^{'}$, and $x_v^{'}$ denote the corresponding feature vectors for node v.

Attack on Gate Obfuscation. In gate obfuscation, each subgraph captures the localities around the obfuscated node, *i.e.,* the node for which the type is to be predicted. The GNN learns the meaningful information about the node in question through its locality (*i.e.,* extracted subgraphs; line 2 in Algorithm 1). We follow the same subgraph-extraction process as in interconnect deobfuscation, except that we only consider a single target node s here, representing the obfuscated gate.
Challenge 3. *Distinguishing between single-input and multi-input gates.* For an obfuscated gate, the true function could be either a single-input gate (INV, BUF) or a multiple-input gate (AND, NAND, etc). Naturally, in case the GNN predicts a single-input gate for an obfuscated gate that is, in reality, a multiple-input gate, or vice versa, the recovered netlist would be invalid. Hence, in our proposed attack framework (Titan), for the GNN to distinguish between single-input and multiple-input gates, we embed the input degree (IN) of the obfuscated (target) node to all the nodes in its enclosing subgraph.
Incorporation of Features. Fig. 7 illustrates the feature-vector representation of a node in an extracted subgraph (lines 3–7 in Algorithm 1). The features are as follows.

- *Functionality:* The feature vector is one-hot encoded with the gate type. For an obfuscated node, the gate type is encoded as obfuscated, whereas for a non-obfuscated gate, the gate type could be (AND, NAND, etc) or PI/PO. W.l.o.g., the feature vector consists of 11 bits.
- *Distance Labeling:* We assign, to each node in the subgraph, its distance from the target node. This helps the GNN to

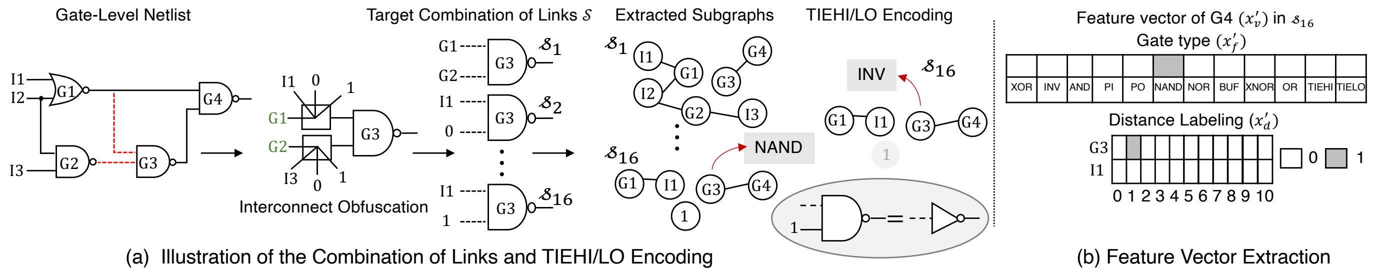
Gate-Level Netlist
Interconnect Obfuscation
Target Combination of Links $\mathcal{S}$
Extracted Subgraphs
TIEHI/LO Encoding
INV
NAND
(a) Illustration of the Combination of Links and TIEHI/LO Encoding
Feature vector of G4 (x'_v) in s_{16}
Gate type (x'_f)
XOR INV AND PI PO NAND NOR BUF XNOR OR TIEHI TIELO
Distance Labeling (x'_d)
0 1 2 3 4 5 6 7 8 9 10
(b) Feature Vector Extraction

Fig. 6. (a) Sampling of possible combination for missing connections as input for a gate in interconnect obfuscation scheme [16] and TIEHI/LO encoding. (b) Feature vector representation x'_v for a node v in the subgraph enclosing an obfuscated link in interconnect obfuscation.

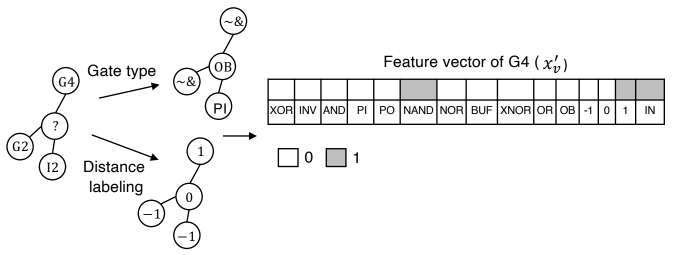
Gate type
Distance labeling
Feature vector of G4 (x'_v)
XOR INV AND PI PO NAND NOR BUF XNOR OR OB -1 0 1 IN

Fig. 7. Illustration of the feature extraction (x'_v) for a node v in the subgraph enclosing an obfuscated gate in gate obfuscation.

learn about the structural information of the subgraph. Note that, since the extracted subgraphs are undirected, the GNN cannot learn about input and output to a node. Hence, we utilize + and - notation (positive and negative ranges) for the distance values to differentiate between input and output.

- IN: This feature is a one-bit vector where 0 indicates a single-input gate, whereas 1 indicates a multiple-input gate, thus helping the GNN to distinguish between them.

E. GNN Topology

There are various GNN architectures available in the literature. We have tested our platform using different state-of-the-art models, such as line GNN [41], graph isomorphic network (GIN) [34], and deep graph convolutional neural network (DGCNN) [42]. We have chosen GIN since it is more expressive than other state-of-the-art models—line GNN and DGCNN (further details in Appendix). Furthermore, GIN has shown the best results for the required task, *i.e.,* interconnect and gate deobfuscation (lines 8–14 in Algorithm 1).

The GIN architecture performs neighborhood aggregation on G' as follows, where MLP refers to a multi-layer perception.

$$h_v^l = \mathrm{MLP}^l\left(h_v^{l-1} + \sum_{u\in\mathcal{N}(v)} h_u^{l-1}\right) \tag{3}$$

A READOUT function adds all the node embeddings from the same layer (*i.e.,* performs vector addition). The obtained layer-wise representations are then concatenated together, as below, to obtain the overall subgraph representation.

$$h_{G'}^{[0:L]} = \mathrm{CONCAT}\left(\mathrm{READOUT}\left(h_v^l \middle| v\in G'\right) \middle| l=0,...,L\right) \tag{4}$$

The subgraph embedding $h_{G'}^{[0:L]}$ gets passed through a dense layer to obtain the likelihood scores for the different classes.

F. Post-Processing

Likelihood Analysis. Recall that the GNN returns the likelihood of each possible combination of links (for interconnect obfuscation) or the likelihood of each possible gate/node-type (for gate obfuscation). Hence, we perform an initial likelihood analysis to obtain the final set of links or node types.

Analysis for Interconnect Obfuscation. We train the GNN on hop sizes of $r = \{2,3,4\}$. During the evaluation, the GNN reports the likelihood of each combination of links considering each hop size r. From the likelihoods obtained for each combination of links returned for different hop sizes, we do not directly choose the combination with the highest likelihood. Rather, we check for the likelihood of any given link in each combination that it occurs in and compute the average of all the related likelihoods and hops. This approach renders the selection more robust in the presence of variations in the predicted likelihoods. In other words, we determine the average likelihood for each link across all combinations and hop sizes and choose the final links based on the highest average likelihood.

Analysis for Gate Obfuscation. Here, we have a likelihood of each possible gate-type for each obfuscated gate for a hop size of 2. Hence, we choose the gate-type with the highest likelihood as the predicted class.

Post-Processing Using Oracle. We describe the post-processing algorithm in Algorithm 2. Here, we use the oracle to obtain true input-output mappings to refine the predicated deobfuscation. We utilize HD as a metric to guide the greedy search. The search can be terminated at any given point (based on the *timeout* value; lines 19–20) and returns the recovered design with the best HD at that point. We model the obfuscated gate/net components using MUXes, where all possible gates or nets of the obfuscated component is an input to the MUX. Furthermore, the select lines of the MUXes can be understood to be controlled by key-bits.

V. Experimental Investigation

In this section, we present the results of Titan considering selected benchmarks from the ITC-99 suite [43]. We also compare our results with state-of-the-art attacks in Section V-D.

Algorithm 2: Post-Processing

```
Input: Final set of links, node-types ($\mathcal{P}$); oracle $\mathcal{O}$; obfuscated
       design modelled as MUX-Lock $\mathcal{R}$; $timeout$;
Output: Final recovered design $\mathcal{Y}$
/* Extract keys mapped to $\mathcal{P}$ by converting
   obfuscated design into MUX-Lock-based design
   */
1  $K \leftarrow$ EXTRACT_KEYS($\mathcal{P}, \mathcal{R}$)
/* Calculate HD between oracle and the
   recovered design, using key extracted above
   */
2  $HD \leftarrow$ CALCULATE-HD($\mathcal{O}, \mathcal{R}, K$)
/* HD obtained using GNN predictions is set as
   starting value of HD for greedy algorithm */
3  HD_init $\leftarrow HD$
4  len_key $\leftarrow$ LENGTH($K$)
5  BestHD $\leftarrow HD$
/* best-$K$, i.e., the key providing low HD, is
   initialized with the key mappings of GNN
   predictions                                */
6  best-$K \leftarrow K$
7  start_time $\leftarrow$ CURRENT_TIME()
8  while $HD > 0$ do
       /* Select a random key-bit            */
9      $i \leftarrow$ RANDOM($0$, len_key $- 1$)
       /* Flip $i^{th}$ key-bit               */
10     $K' \leftarrow$ FLIP($K$,i)
       /* HD calculation between oracle and
          recovered design using new key (i.e.,
          with some flipped bit)              */
11     $HD \leftarrow$ CALCULATE-HD($\mathcal{O}, \mathcal{R}, K'$)
       /* Update best-HD and best-$\mathcal{K}$ if the
          obtained HD is lower than the current
          best-HD                             */
12     if $HD <$ best $-$ HD then
13         best $-$ HD $\leftarrow HD$
14         best-$K \leftarrow K'$
15     else
16         $K \leftarrow$ best-$K$
17     time $\leftarrow$ CURRENT_TIME()
18     time_elapsed $\leftarrow$ (time $-$ start_time)
       /* If time_elapsed is greater than or equal to
          the given timeout, the post-processing
          terminates with the current best-$K$ and
          best-HD                             */
19     if time_elapsed $\geq$ $timeout$ then
20         break
21 $\mathcal{Y} \leftarrow$ RECOVER(best-$K, \mathcal{R}$)
22 return $\mathcal{Y}$
```

A. *Experimental Setup*

Implementation. We have implemented selected obfuscation schemes (i.e., interconnect obfuscation [16] and gate obfuscation [15] considering, w.l.o.g., eight gate-types: INV, BUF, AND, NAND, NOR, XOR, OR, and XNOR) in *Python*. Further, we have developed a *Python* script to model obfuscated netlists as graphs. We use the *Pytorch*-based implementation of the GIN [34] model for subgraph-classification, where we have implemented tailored techniques for Titan as outlined in Section IV-D. W.l.o.g., we utilize a maximum hop size of r=4 and r=2 for tackling interconnect and gate obfuscation.

The experiments were performed on a single compute node with Intel(R) Xeon(R) CPU, comprising 28 cores operated at 2.4GHz, with 100GB memory guaranteed as exclusively available by the operating system.

Dataset Generation. We generate datasets by implementing the four input-based interconnect obfuscation scheme [16] (Section II-A) and an eight function-based gate obfuscation scheme for selected ITC-99 benchmarks [43] (in *BENCH* format), for both training and validation.

We generate separate training datasets considering the same obfuscation scale as the design under attack (w.l.o.g. 10%, 30%, 40%, or 50% in this work). We are following a standard leave-one-out scenario, *i.e.,* the training and validation datasets consists of different benchmarks other than the testing dataset. For example, to attack the ITC-99 benchmark b14_C, we use b15_C, b17_C, b20_C, b21_C, and b22_C for training and validation. We use 90% of the dataset generated for training and the remaining 10% for validation. We summarize the specifications of the datasets for each benchmark and obfuscation scale for both obfuscation techniques in Table IV.

For attacking interconnect obfuscation, we train the GNN on a balanced dataset of positive (correct) and negative (dummy) nets, where we randomly select 3,000 links each (correct and dummy) from the training dataset (*i.e.,* a total of 6,000 training links). For attacking gate obfuscation, we train the GNN on all the classes or gate types (w.l.o.g. 8 in this work).

GNN Topology. We use the default parameters of the GIN architecture [34], like using 2 MLP layers for each GNN layer. We set the batch size as 128, the hidden dimension as 128, and the number of GNN layers as 6; the latter has the best prediction performance. We train the GNN to minimize the cross-entropy cost function for 500 epochs using *Adam* optimizer with a learning rate of 0.01. The model with the best performance on the validation set (*i.e.,* lowest validation loss) is used for testing.

Evaluation Metrics. We use HD, precision, and key-prediction accuracy (KPA) to evaluate the performance of Titan. HD is a metric that evaluates the output corruption of the design. Precision is a metric that evaluates the performance of the ML model and is defined as the ratio of true positives to the sum of true and false positives. KPA is defined as the percentage of correctly predicted key-bits: $KPA = |K_c|/|K| * 100$ where $|K_c|$ is the number of correct key-bits and $|K|$ is the total number of key-bits [44]. Note that the KPA metric also indicates on the scale/percentage of deobfuscated components; recall that key-bits are used for configuration of obfuscated components in the MUX-based locking model (Section II-B). Note that we report KPA and key-size (*i.e.,* number of total key-bits) for context on the complexity of attacking large-scale obfuscation and related performance of state-of-the-art attacks.

Baseline HD of Obfuscated Designs. Here we calculate the HD considering a random guess/assignment for all obfuscated nets or gates. This baseline HD is calculated and averaged for all obfuscated benchmarks over 100 rounds of randomized obfuscation and, for each round, over 1,000 runs with a different random guess/assignment for each run, and considering 10,000 random input patterns for each run. For this thorough sampling, we observe an average HD of 23.44%, 35.5%, 39.12%, and 42.08% for 10%, 30%, 40%, and 50% of interconnect obfuscation, respectively. For gate obfuscation of 10%, 30%, 40%, and 50%, we observe an average HD of 25.62%, 41.69%, 44.18%, and 46.37%, respectively.

TABLE IV
TRAINING AND TESTING DATASETS FOR INTERCONNECT AND GATE OBFUSCATION

Benchmark	10% Interconnects Obfuscation		30% Interconnect Obfuscation		40% Interconnect Obfuscation		50% Interconnect Obfuscation		10% Gate Obfuscation		30% Gate Obfuscation		40% Gate Obfuscation		50% Gate Obfuscation	
	# Testing links	# Training links	# Testing links	# Training links	# Testing links	# Training links	# Testing links	# Training links	# Testing nodes	# Training nodes	# Testing nodes	# Training nodes	# Testing nodes	# Training nodes	# Testing nodes	# Training nodes
b14_C	4,776	78,395	15,518	254,417	22,065	349,523	28,464	444,216	389	6,318	1,168	18,959	1,558	25,280	1,947	31,600
b15_C	8,795	73,197	30,760	239,165	41,918	329,670	52,498	420,182	722	5,985	2,166	17,961	2,888	23,950	3,610	29,937
b17_C	28,559	53,185	97,039	172,896	134,254	237,334	172,005	300,685	2,347	4,360	7,034	13,084	9,390	17,448	11,738	21,809
b20_C	11,542	72,032	36,021	233,914	48,298	323,290	61,682	410,998	923	5,784	2,771	17,536	3,695	23,143	4,619	28,928
b21_C	11,038	71,953	34,848	235,087	48,580	323,008	62,987	409,693	917	5,790	2,751	17,376	3,669	23,169	4,586	28,961
b22_C	16,752	66,893	55,749	214,186	76,473	295,115	95,044	377,636	1,409	5,298	4,228	15,899	5,638	21,200	7,047	26,500

TABLE V
COMPARISON OF SUBGRAPH EXTRACTION TIME (IN SECONDS) DEPENDING ON THE NUMBER OF DUMMY NETS

Benchmark	**3 dummy nets**	**4 dummy nets**	**5 dummy nets**	**6 dummy nets**
b14_C	251	670	1,421	3,407
b15_C	558	5,330	6,538	11,259
b17_C	704	5,220	8,346	23,681
b20_C	503	1,300	3,041	6,982
b21_C	512	1,492	2,946	7,202
b22_C	763	2,423	4,560	10,032
Average	**548**	**2,739**	**4,475**	**10,427**

TABLE VI
COMPARISON IN THE PRECISION OF GNN PREDICTION (%) DEPENDING ON THE NUMBER OF DUMMY NETS FOR HOP SIZE OF 2

Benchmark	**3 dummy nets**	**4 dummy nets**	**5 dummy nets**	**6 dummy nets**
b14_C	67.43	40.12	47.61	39.05
b15_C	70.56	58.05	49.79	45.83
b17_C	74.98	58.74	52.35	48.61
b20_C	70.92	61.91	51.18	46.90
b21_C	72.86	55.71	53.79	46.13
b22_C	70.10	59.61	52.25	45.07
Average	**71.14**	**55.69**	**51.16**	**45.26**

TABLE VII
RESULTS OF TITAN (WITHOUT ORACLE POST-PROCESSING) ON INTERCONNECT OBFUSCATION

Benchmark	Obs %	N	Key-size	Precision (%)	HD (%)	KPA (%)
b14_C	10	709	1,418	71.51	14.85	81.38
	30	2,183	4,366	75.72	23.24	83.69
	40	2,958	5,916	70.25	22.29	79.93
	50	3,741	7,462	64.14	28.58	75.89
b15_C	10	1,229	2,458	85.76	3.65	90.6
	30	3,855	7,710	78.11	22.92	85.42
	40	5,184	10,368	78.66	26.15	85.68
	50	6,479	12,958	75.95	31.34	84.00
b17_C	10	4,025	8,050	91.87	5.47	94.47
	30	12,592	25,184	85.94	16.13	90.59
	40	16,946	33,892	78.60	23.36	85.83
	50	21,326	42,652	77.23	29.91	84.86
b20_C	10	1,661	3,322	85.49	8.73	90.25
	30	5,083	10,166	78.18	21.94	85.57
	40	6,789	13,578	75.68	26.01	83.55
	50	8,523	17,046	67.65	33.68	78.26
b21_C	10	1,613	3,226	86.61	9.92	90.79
	30	4,964	9,928	78.40	20.97	85.64
	40	6,679	13,358	75.73	25.29	83.71
	50	8,429	16,858	72.82	29.91	81.78
b22_C	10	2,534	5,068	88.44	8.29	92.47
	30	7,833	15,666	81.38	19.60	87.57
	40	10,506	21,012	77.19	21.57	84.98
	50	13,112	26,224	69.27	32.86	79.37

Note that "Obs %" is the obfuscation scale, "N" is the number of obfuscated components, "Key-size" is the equivalent number of total key-bits for the obfuscated design converted to a MUX-based locking problem.

B. Breaking Interconnect Obfuscation

GNN-Based Subgraph Classification. We showcase the results of Titan after GNN predictions (without oracle post-processing) for attacking interconnect obfuscation in Table VII. For HD, we observe an average reduction of 14.95, 14.70, 15.00, and 16.42 pp over the baseline HD (random guessing) for 10%, 30%, 40%, and 50% obfuscation scales, respectively. We observe that the subgraph-classification framework recovers more than 65% of the nets and achieves an average KPA of 85.26% for all obfuscation scales.

Effect of Number of Dummy Nets. We showcase the effect of increasing dummy nets on the subgraph extraction time for 30% obfuscation in Table V. We consider the number of dummy nets as 3, 4, 5, and 6, where two dummy nets are TIE HI and TIE LO, and the remaining connections are driven by internal nets of the design.

The average time taken for subgraph extraction in the case of 4, 5, and 6 dummy nets is 5×, 8×, and 19× compared to the 3 dummy-nets scenario that we target in this work. Furthermore, we can parallelize the subgraph extraction procedure to speed up execution time. The precision of the GNN decreases with the increase in dummy nets, as shown in Table VI.

Please note that increasing the number of dummy nets results in a considerable increase in power, performance, and area overheads [16]. Such overheads are typically not preferred by a security-enforcing designer; hence the number of dummy nets is limited in practice to less than or equal to four.

Post-Processing. We further perform a greedy, oracle-based post-processing on the GNN outputs. The results before and after post-processing are shown in Fig. 8. We observe an average reduction of 3 pp HD for all the obfuscation scales with the help of post-processing. Here we note that post-processing is performing better for smaller benchmarks: e.g., for the b14_C benchmark (smallest among all considered), HD decreases by 12.54 and 7.27 pp for 10% and 30% obfuscation scales, respectively, whereas for benchmark b17_C (largest among all considered), HD decreases by 0.2 and 0.96 pp for 10% and 30% obfuscation scales, respectively.

C. Breaking Gate Obfuscation

GNN-Based Subgraph Classification. The results of Titan after GNN predictions (without oracle post-processing) for tackling gate obfuscation are given in Table VIII. For HD, we observe an average reduction of 13.72, 21.46, 19.1, and 21.29

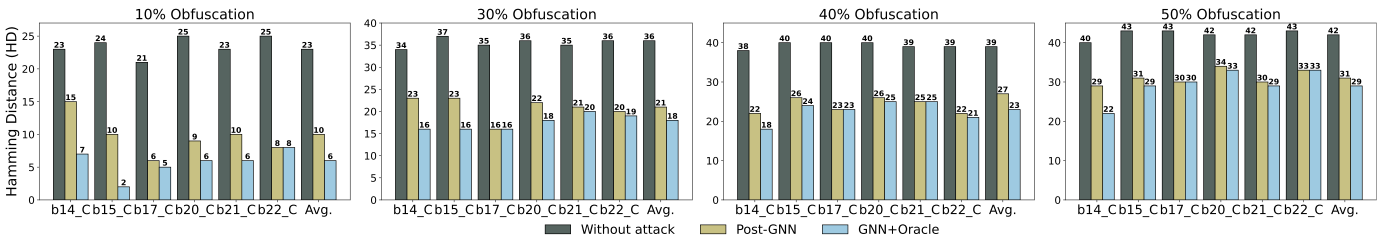

Fig. 8. Results of Titan on interconnect obfuscation. The bar plots represent the HD of the obfuscated designs without any attack (*i.e.,* baseline HD), Post-GNN attack, and the full GNN + Oracle attack.

TABLE VIII
RESULTS OF TITAN (WITHOUT ORACLE POST-PROCESSING) ON GATE OBFUSCATION

Benchmark	Obs %	N	Key-size	Precision (%)	HD (%)	KPA (%)
b14_C	10	389	1,109	73.39	14.65	87.37
	30	1,168	3,286	71.98	16.37	86.09
	40	1,558	4,356	59.75	23.67	85.42
	50	1,947	5,441	63.96	22.46	86.31
b15_C	10	722	2,042	55.87	10	88.25
	30	2,166	6,092	52.98	17.27	89.79
	40	2,888	8,154	62.38	25.34	90.95
	50	3,610	10,152	62.57	22.87	90.19
b17_C	10	2,347	6,653	59.23	11.18	88.06
	30	7,043	19,834	59.09	22.69	88.76
	40	9,390	26,506	57.42	26.3	88.21
	50	11,738	33,087	58.60	31.38	88.14
b20_C	10	923	2,525	54.42	8.39	85.50
	30	2,771	7,657	58.89	22.12	87.18
	40	3,695	10,231	70.52	23.67	86.72
	50	4,619	12,753	67.96	21.54	87.05
b21_C	10	917	2,561	75.27	12.83	85.90
	30	2,751	7,641	72.59	21.62	87.63
	40	3,669	10,177	63.25	25.95	86.30
	50	4,586	12,690	69.17	26.42	87.29
b22_C	10	1,409	3,885	59.57	14.39	86.23
	30	4,228	11,688	64.96	21.31	87.35
	40	5,638	15,547	62.39	25.5	87.25
	50	7,047	19,407	65.54	29.26	87.32

See Table VII for remarks on labels.

pp over the baseline HD (using random guess) for 10%, 30%, 40%, and 50% obfuscation scales, respectively. We observe that the subgraph-classification framework recovers more than 70% of the gates and achieves an average KPA of 87.47% for all the obfuscation scales.

Post-Processing. We perform the same, greedy and oracle-based post-processing on the GNN outputs. The results before and after post-processing are shown in Fig. 9. On average, we observe a 4.5 pp reduction in HD through post-processing. Similar to the case of post-processing on interconnect obfuscation, we observe that there is a more substantial reduction of HD for smaller designs.

D. *Comparison to State-of-the-art Attacks*

We compare Titan with state-of-the-art attacks, *i.e.,* SAT [5], AppSAT [6], redundancy attack [7]. We consider a timeout of 48 hours for all these attacks. Fig. 10 illustrates that the SAT and AppSAT attacks both fail to resolve all the obfuscated designs before timeout, specifically for large-scale obfuscation (30–50%, except for 30% obfuscation scale for b14_C). We observe that the redundancy attack has failed to recover the designs for all obfuscation scales (10–50%). This indicates that these attacks cannot handle such complex, large-scale obfuscation problem. Further, we launch OMLA [11] on gate obfuscation where it returns an invalid netlist because of its restriction to two-function-based gate obfuscation.[5]

Fig. 11, Fig. 12, Fig. 13, and Fig. 14 represent the comparison of Titan to ML-based attacks (UNTANGLE and SCOPE) and a greedy, oracle-based attack for interconnect and gate obfuscation. All results indicate that Titan outperforms the state-of-the-art attacks. Further analysis are provided below.

Greedy, Oracle-Based Approach on Random Key. Recall that, for Titan, we perform a greedy, oracle-based post-processing on top of the GNN outputs. To understand the effect of the GNN predictions on the overall attack framework, here we launch the post-processing approach alone on a randomly initialized key assignment, rather than on the key extracted using GNN predictions.

We observe that the decrease in HD is much less for this baseline approach when compared to Titan. More specifically, when attacking interconnect obfuscation, we observe that the average reduction in HD (over the baseline HD, Section V-A) for to the greedy, oracle-based approach alone is 5.43, 2.77, 2.44, and 2.13 pp for obfuscation scales of 10%, 30%, 40%, and 50%, respectively (Fig. 11). Similarly, when attacking gate obfuscation, we observe an average HD reduction of 4.62, 3.78, 4.48, and 5.54 pp for obfuscation scales of 10%, 30%, 40%, and 50%, respectively (Fig. 12). These reductions are on average 5× lesser than those of Titan for all scales of interconnect and gate obfuscation. Thus, the GNN predictions are essential for the success of Titan.

Comparison with SCOPE. We launch SCOPE with the default margin parameter of 0. Note that SCOPE returns a key where some key-bits are "don't care" bits. Thus, we cannot directly use the key returned by SCOPE for further post-processing and HD analysis. To reasonably resolve this, we randomly assign bits 0 or 1 to any "don't care" bits when calculating HD.

Second, we observe an average KPA of 58.21% and 51% on the original key returned by SCOPE for interconnect and gate obfuscation, respectively (Fig. 13 and Fig. 14). Accordingly, Titan performs better by 27.05 and 36.45 pp for interconnect and gate obfuscation, respectively. SCOPE deobfuscates designs such that they induce, on average, HD values that are increased/worsened by 12.55 and 19.04 pp for

[5]We could not perform a comparison with ML-based attacks such as SAIL [32] and SnapShot [44] as they are not publicly available.

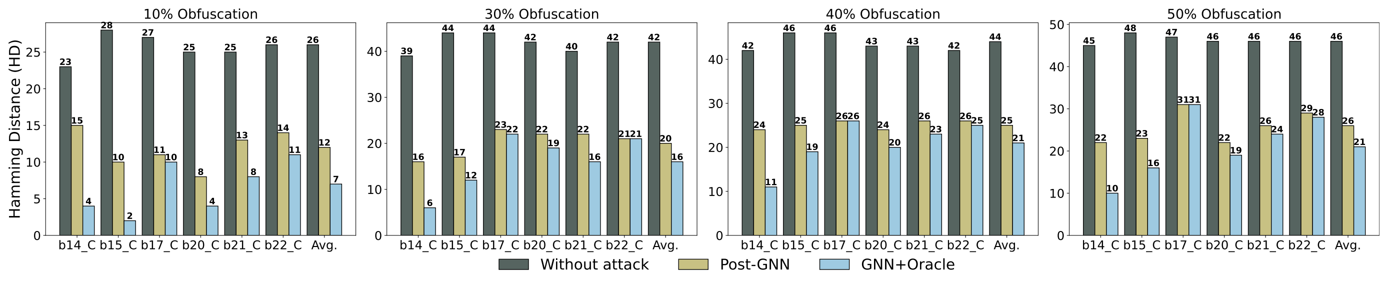

Fig. 9. Results of Titan on gate obfuscation with and without post-processing. The bar plots represent the HD of the obfuscated designs without any attack (*i.e.,* baseline HD), Post-GNN attack, and the full GNN + Oracle attack.

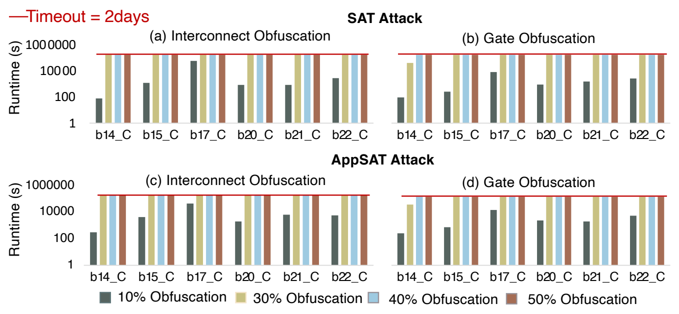

Fig. 10. SAT and AppSAT results for selected ITC-99 benchmarks for 10–50% of interconnect and gate obfuscation.

interconnect and gate obfuscation, respectively, compared to our Titan without post-processing.

Third, we further compare SCOPE with Titan, including post-processing. Note that SCOPE is an oracle-less attack; thus, for the fairness of this comparison, we employ the same greedy, oracle-based post-processing approach proposed in our work on top of the key returned by SCOPE. We observe an inferior performance of SCOPE compared to Titan framework for both interconnect and gate obfuscation (Fig. 11 and Fig. 12). In fact, for most of the benchmarks under gate and interconnect obfuscation, we observe that the results for the greedy baseline attack and the augmented SCOPE oracle differ only by a little.

Comparison with UNTANGLE. UNTANGLE is originally proposed to attack Interlock [27]. Thus, we cannot use its post-processing scheme while attacking the four-input-based interconnect-obfuscation scheme [16]. We devise a tailored post-processing for UNTANGLE, where we choose the highest-likelihood link from all the possible links per obfuscated net. As in our regular attack flow, we obtain the GNN likelihoods for hop sizes of 2, 3, and 4, and then compute the average of likelihoods for each link over different hops before choosing the highest-likelihood link. Note that UNTANGLE is only applicable for interconnect obfuscation. To that end, we observe that Titan performs better than UNTANGLE in terms of KPA and HD by an average of 9.42 pp and 5.30 pp over all scales of interconnects obfuscation (10–50%) (Fig. 13). We argue that UNTANGLE performs worse for two reasons: (i) UNTANGLE targets obfuscation techniques where only a single input of a given gate is obfuscated; (ii) UNTANGLE does not support TIEHI/LO connections that exist in the interconnect obfuscation scheme we consider [16].

TABLE IX
COMPARISON OF ACCURACY (%) TO STATE-OF-THE-ART GNN-BASED ATTACKS FOR 30% UNIFIED OBFUSCATION

Benchmark	#Obfuscated Components	OMLA + UNTANGLE	UNTANGLE + OMLA	Individual Attacks	Titan Post-GNN
b14_C	2,277	62.88	66.01	64.90	70.93
b15_C	4,223	66.99	66.54	67.39	76.65
b17_C	13,732	65.91	66.90	67.44	78.44
b20_C	5,392	65.37	68.26	66.80	73.99
b21_C	5,350	64.67	67.10	67.06	76.05
b22_C	8,231	64.57	66.22	65.60	69.69
Average	**6,534**	**65.06**	**66.84**	**66.53**	**74.29**

Note that UNTANGLE is an oracle-less attack; similar to the comparison with SCOPE, we employ the same oracle-based post-processing on top of the key provided by UNTANGLE. On average, Titan achieves 5.97, 8.03, 5.48, and 2.58 pp lower/better HD than UNTANGLE for obfuscation scales of 10%, 30%, 40%, and 50%, respectively (Fig. 11).

E. Attacking Unified Obfuscation

We consider a special case of obfuscation, *i.e.,* a unified obfuscation scheme to demonstrate the efficacy of Titan. A unified obfuscation scheme includes both interconnect and gate obfuscation. Titan is directly applicable to such a unified obfuscation scheme. In contrast, neither of the state-of-the-art GNN-based attacks apply to unified obfuscation since they target a specific type of obfuscation, *i.e.,* gate or interconnect obfuscation. To that end, we consider three cases to compare the efficacy of Titan with state-of-the-art GNN-based attacks.

1) We first launch OMLA to recover the gate-obfuscated components and subsequently launch UNTANGLE on the OMLA-recovered design to recover the interconnect-obfuscated components (OMLA + UNTANGLE).
2) We first launch UNTANGLE to recover the interconnect-obfuscated components and subsequently launch OMLA on the UNTANGLE-recovered design to recover the gate-obfuscated components (UNTANGLE + OMLA).
3) We launch both the attacks, OMLA and UNTANGLE, individually to recover the different types of obfuscated components separately.

Our GNN-based attack outperforms all the aforementioned scenarios of state-of-the-art attacks by 8–9 pp for 30% unified obfuscation, as shown in Table IX.

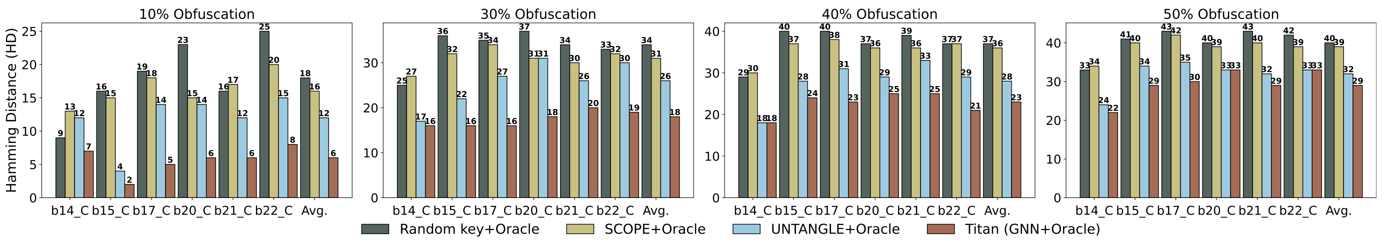

Fig. 11. Comparison of SCOPE + Oracle, UNTANGLE + Oracle, Random key + Oracle with Titan (GNN + Oracle) on interconnect obfuscation. The plots indicate that Titan performs superior for all benchmarks and scales of obfuscation (10%–50%).

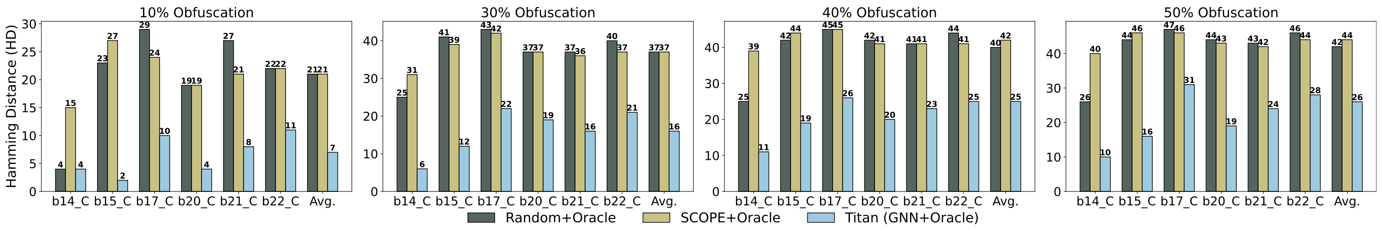

Fig. 12. Comparison of SCOPE + Oracle, Random key + Oracle with Titan (GNN + Oracle) on gate obfuscation. The plots indicate that Titan performs superior for all benchmarks and scales of obfuscation (10%–50%).

F. Runtime

Attack on Interconnects Obfuscation. The GNN model requires a maximum of 9, 5.6, 3.6, and 2.5 hours for training and subgraphs extraction on obfuscation scales of 10%, 30%, 40%, and 50%, respectively. The GNN takes less training time for a higher percentage of interconnect obfuscation, as explained next. For attacking interconnect obfuscation, the GNN trains on the subgraphs extracted around each obfuscated interconnect. These subgraphs consist of the neighboring nodes of the considered obfuscated interconnect. The true connections of obfuscated interconnects are unknown; thus, they are treated as missing links in the graph. With an increase in the obfuscation, the number of missing links (obfuscated interconnects) increases, as shown in Table IV. Thus, the graph becomes more sparse, and the extracted subgraphs are smaller (Table X). Furthermore, the GNN takes an average of 30 minutes for testing across all the designs.

Attack on Gate Obfuscation. The GNN model requires at most 33, 51, 55, and 60 minutes to train for obfuscation scales of 10%, 30%, 40%, and 50%, respectively. The training time increases slightly with the percentage of obfuscation. This is because obfuscated gates are not removed from the extracted subgraphs. Thus, there is no change in the size of subgraphs with the percentage of obfuscation. We observed an average subgraph size of 22 for a hop size of two for all the benchmarks across all percentages of obfuscation. However, the obfuscated components increase with the percentage of obfuscation, as shown in Table IV. Thus, we observe a slight increase in training time. Furthermore, the GNN takes an average of 3 seconds for testing across all the designs.

Post-Processing. The greedy, oracle-based approach for post-processing is an any-time algorithm; we can terminate the post-processing any time to obtain some refined result. W.l.o.g., we run post-processing for 48 hours in all experiments, including the application of post-processing for augmenting state-of-the-art results.

TABLE X
AVERAGE SUBGRAPH SIZE FOR A HOP SIZE OF 2 FOR DIFFERENT PERCENTAGES OF INTERCONNECT OBFUSCATION

Benchmark	10% Obfuscation	30% Obfuscation	40% Obfuscation	50% Obfuscation
b14_C	50	36	29	23
b15_C	54	41	33	25
b17_C	52	37	30	24
b20_C	49	34	26	22
b21_C	48	33	27	22
b22_C	47	34	27	21
Average	**50**	**36**	**29**	**23**

VI. POTENTIAL COUNTERMEASURES

Our work demonstrates that even large-scale obfuscation leaves structural traces and remains vulnerable to GNN-based attacks. In this work, we developed Titan that predicts the functionality of the obfuscated components (either gate or wire) by subgraph extraction, *i.e.,* Titan exploits the structural and functional hints from the locality (neighborhood) of the obfuscated components. Possible countermeasures against our proposed attack could include: (i) creating look-alike subgraphs around obfuscated components and (ii) hiding the locality of obfuscated components through design IP redaction.

For (i), if the subgraphs around obfuscated components can be engineered to look alike, the extracted subgraphs may not leak structural hints. For (ii), to hide the locality of obfuscated components, hardware redaction [45] can be applied, where a portion of the design is replaced using an embedded FPGA (eFPGA). Redacting the design using eFPGA ensures that the locality of each redacted component consists of eFPGA-based components that do not leak structural or functional hints about the original design. However, such advanced obfuscation using eFPGAs may incur design overheads.

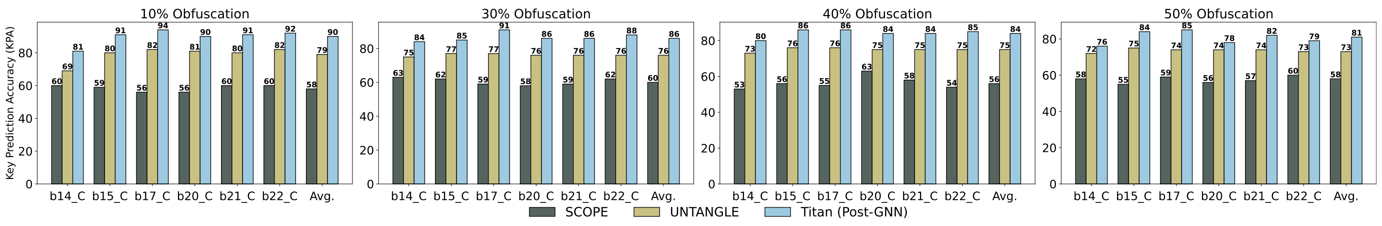

Fig. 13. Comparison of KPA for SCOPE and UNTANGLE with Titan on interconnect obfuscation for selected ITC-99 benchmarks without oracle-based post-processing. The plots indicates that Titan performs better than SCOPE for all benchmarks and scales of obfuscation (10%–50%).

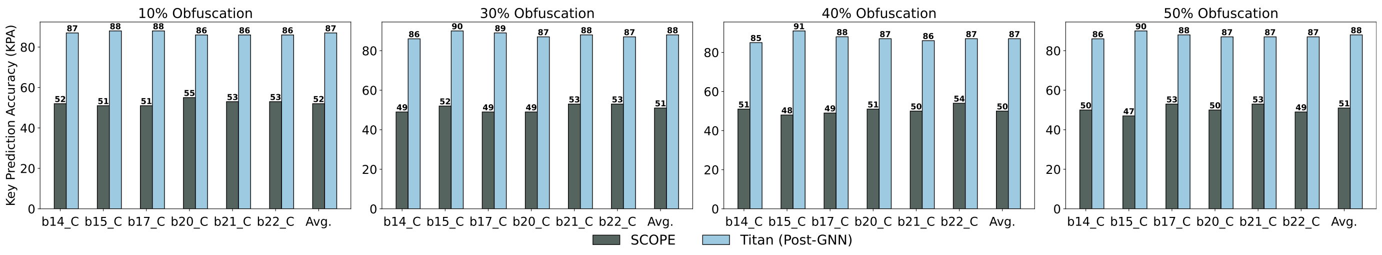

Fig. 14. Comparison of KPA for SCOPE with Titan on gate obfuscation for selected ITC-99 benchmarks without oracle-based post-processing. The plots indicates that Titan performs better than SCOPE for all benchmarks and scales of obfuscation (10%–50%).

VII. Conclusion

We propose a holistic attack framework, Titan, against large-scale hardware obfuscation, considering up to 50% of interconnects or gates being obfuscated. Our proposed framework is based on subgraph classification using a graph neural network (GNN). The GNN model exploits structural and functional hints that remain present even in large-scale obfuscation. Additionally, we implement an oracle-guided post-processing technique to refine the accuracy of the GNN outputs.

For the ITC-99 benchmarks, we recovered an average of 63.40% and 77.94% of the obfuscated components for 10–50% gate and interconnect obfuscation scales. More specifically, for 50% interconnect obfuscation on the smallest benchmark b14_C and the largest benchmark b17_C, our attack improved Hamming distance (HD) by 19.14 and 12.94 percentage points (pp) over the random-guessing baseline. Similarly, for 50% gate obfuscation, we improved HD by 35.5 and 16.31 pp for b14_C and b17_C, respectively. These considerable improvements indicate leakage of structural and functional traces even for large-scale obfuscated designs (30%–50% obfuscated components). Additionally, we compare our results against six state-of-the-art attacks. We note that state-of-the-art attacks either fail, are not applicable, or are limited in attack performance compared to Titan, especially for large-scale obfuscation (30%–50% obfuscated components) and also for unified obfuscation (gate and interconnect obfuscation). This is due to the significant complexity for such large-scale obfuscation; the equivalent key-size for a MUX-based locking model is up to 42,652 key-bits.

In short, while large-scale interconnect and gate obfuscation seem promising and represents a significant challenge for state-of-the-art attacks, Titan opens up new doors toward competitive, scalable, and generalized attacks. We will open-source Titan and corresponding artifacts to enable reproducibility and foster future work.

Appendix

For the subgraph classification task, the GNN should be able to distinguish different graph structures by mapping them to different representations in the embedding space, in order to classify the graphs correctly. The ability to map any two different graphs to different embeddings implies that isomorphic graphs must be mapped to the same representation and non-isomorphic ones to different representations. This property is a.k.a. expressive power of GNN. However, state-of-the-art GNNs such as DGCNN and line GNN are less expressive than GIN. Here, we explain the reason for the same.

The function in a GNN that is responsible for the generation of embeddings is neighborhood aggregation. In DGCNN architecture, the neighborhood aggregation on a graph $G^{'}$ for l^{th} layer is as follows.

$$Z^l = f(\tilde{D}^{-1} Z^{l-1} X W^{l-1}) \tag{5}$$

DGCNN is less expressive because the neighborhood aggregator function is a mean function. For instance, consider two different input graphs $G_1^{'}$ and $G_2^{'}$ as shown in Fig. 15 with all the nodes having same features. Thus, the resulting features of v_1 and v_2 upon neighborhood aggregation are also same. Fig. 15 demonstrates the neighborhood aggregation in DGCNN where $v1$ and $v2$ in $G_{1a}^{'}$ and $G_{2a}^{'}$ have the same embeddings. In contrast, $G_1^{'}$ and $G_2^{'}$ are not similar or isomorphic. For such cases, DGCNN misclassifies the graphs. Thus, DGCNN is less expressive. Line GNN, also has a mean neighborhood aggregation; thus, it is also less expressive.

However, GIN is an expressive network with injective aggregation function that helps it to map different graphs to different representations. Below, we showcase the injective property of the GIN aggregation. Consider an input subgraph $G^{'}$ to GIN. The GIN architecture performs neighborhood

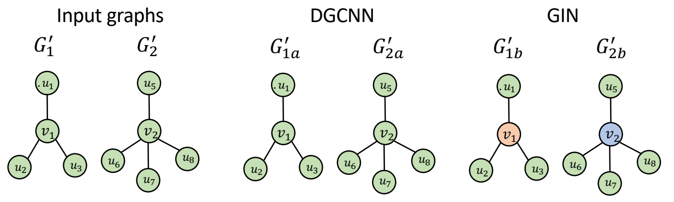

Fig. 15. Example of neighborhood aggregation in DGCNN [42] and GIN [34].

aggregation on $G^{'}$ at l^{th} layer is as follows, where MLP refers to a multi-layer perceptron.

$$h_v^l = \text{MLP}^l\left(h_v^{l-1} + \sum_{u\in\mathcal{N}(v)} h_u^{l-1}\right) \tag{6}$$

The MLP is modelled using a multi-set injective function $\phi(\sum_{x\in S} f(x))$, where f and ϕ are non-linear functions and S is a multi-set. Also, it is proven in [34] that the neighborhood aggregation of GIN is injective. Consider the same two different input graphs $G_1^{'}$ and $G_2^{'}$, as shown in Fig. 15. The resulting graphs upon neighborhood aggregation for v_1 and v_2 are $G_{1b}^{'}$ and $G_{2b}^{'}$ respectively. It can be observed that v_1 and v_2 have different embeddings, unlike DGCNN which has been obtained upon using an injective aggregation function. Thus, GIN is more expressive than other traditional GNNs and is best suitable for our attack.

References

[1] R. Zafar, "TSMC's Total 3nm Investment Will Equal At Least $ 23 Billion," https://wccftech.com/tsmc-3nm-investment-23-billion-project-end/, 2021, [Online; accessed 14-March-2022].

[2] D. S. Abhijit Mahindroo, Nick Santhanam, "The potential shake-up in semiconductor manufacturing business models," 2013. [Online]. Available: http://www.mckinsey.com/~/media/mckinsey/dotcom/client_service/semiconductors/issue%203%20autumn%202013/pdfs/3_futuremanufacturing.pdf

[3] M. Rostami, F. Koushanfar, and R. Karri, "A primer on hardware security: Models, methods, and metrics," *Proc. of the IEEE*, vol. 102, no. 8, pp. 1283–1295, 2014.

[4] W. Hu, C.-H. Chang, A. Sengupta, S. Bhunia, R. Kastner, and H. Li, "An overview of hardware security and trust: Threats, countermeasures, and design tools," *IEEE Trans. on Comput.-Aided Design of Integr. Circuits and Syst.*, vol. 40, no. 6, pp. 1010–1038, 2021.

[5] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust*, May 2015, pp. 137–143.

[6] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "AppSAT: Approximately deobfuscating integrated circuits," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust*, May 2017, pp. 95–100.

[7] L. Li and A. Orailoglu, "Piercing logic locking keys through redundancy identification," in *Proc. Design, Autom. Test Eur. Conf. Exhib.*, 2019, pp. 540–545.

[8] A. Alaql, D. Forte, and S. Bhunia, "Sweep to the secret: A constant propagation attack on logic locking," in *Asian Hardware Oriented Security and Trust Symposium*, 2019, pp. 1–6.

[9] A. Alaql, M. M. Rahman, and S. Bhunia, "SCOPE: Synthesis-based constant propagation attack on logic locking," *IEEE Trans. on VLSI Syst.*, vol. 29, no. 8, pp. 1529–1542, 2021.

[10] L. Alrahis, S. Patnaik, M. A. Hanif, M. Shafique, and O. Sinanoglu, "UNTANGLE: Unlocking routing and logic obfuscation using graph neural networks-based link prediction," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, 2021, pp. 1–9.

[11] L. Alrahis, S. Patnaik, M. Shafique, and O. Sinanoglu, "OMLA: An oracle-less machine learning-based attack on logic locking," *IEEE Trans. on Circuits and Syst. II: Express Briefs*, vol. 69, no. 3, pp. 1602–1606, 2022.

[12] M. Hoffmann and C. Paar, "Doppelganger obfuscation–exploring thedefensive and offensive aspects of hardware camouflaging," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, pp. 82–108, 2021.

[13] R. P. Cocchi, J. P. Baukus, L. W. Chow, and B. J. Wang, "Circuit camouflage integration for hardware IP protection," in *IEEE Des. Autom. Conf.*, 2014, pp. 1–5.

[14] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri, "Security analysis of integrated circuit camouflaging," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2013, pp. 709–720.

[15] I. R. Nirmala, D. Vontela, S. Ghosh, and A. Iyengar, "A novel threshold voltage defined switch for circuit camouflaging," in *IEEE European Test Symp.*, 2016, pp. 1–2.

[16] S. Patnaik, M. Ashraf, O. Sinanoglu, and J. Knechtel, "Obfuscating the interconnects: Low-cost and resilient full-chip layout camouflaging," *IEEE Trans. on Comput.-Aided Design of Integr. Circuits and Syst.*, vol. 39, no. 12, pp. 4466–4481, 2020.

[17] S. Chen, J. Chen, and L. Wang, "A chip-level anti-reverse engineering technique," *J. Emerg. Technol. Comput. Syst.*, vol. 14, no. 2, jul 2018.

[18] J. A. Roy, F. Koushanfar, and I. L. Markov, "Epic: Ending piracy of integrated circuits," in *Proc. Design, Autom. Test Eur. Conf. Exhib.*, 2008, p. 1069–1074.

[19] M. Yasin, J. J. Rajendran, O. Sinanoglu, and R. Karri, "On improving the security of logic locking," *IEEE Trans. on Comput.-Aided Design of Integr. Circuits and Syst.*, vol. 35, no. 9, pp. 1411–1424, 2016.

[20] M. Yasin, B. Mazumdar, J. Rajendran, and O. Sinanoglu, "SARLock: Sat attack resistant logic locking," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust*, 2016, pp. 236–241.

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

[22] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "Full-lock: Hard distributions of sat instances for obfuscating circuits using fully configurable logic and routing blocks," in *IEEE Design Automation Conference*, 2019, pp. 1–6.

[23] A. Saha, S. Saha, S. Chowdhury, D. Mukhopadhyay, and B. B. Bhattacharya, "Lopher: Sat-hardened logic embedding on block ciphers," in *IEEE Des. Autom. Conf.*, 2020, pp. 1–6.

[24] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "Cyclic obfuscation for creating sat-unresolvable circuits," in *Proceedings of the on Great Lakes Symposium on VLSI*, 2017, p. 173–178.

[25] R. Karmakar, S. Chatopadhyay, and R. Kapur, "Encrypt flip-flop: A novel logic encryption technique for sequential circuits," 2018.

[26] N. Limaye, S. Patnaik, and O. Sinanoglu, "Valkyrie: Vulnerability assessment tool and attack for provably-secure logic locking techniques," *IEEE Trans. Inf. Forensics Secur.*, vol. 17, pp. 744–759, 2022.

[27] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "Interlock: An intercorrelated logic and routing locking," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, 2020, pp. 1–9.

[28] K. Z. Azar, H. M. Kamali, H. Homayoun, and A. Sasan, "NNgSAT: Neural network guided sat attack on logic locked complex structures," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, 2020, pp. 1–9.

[29] H. Zhou, R. Jiang, and S. Kong, "Cycsat: Sat-based attack on cyclic logic encryptions," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, 2017, pp. 49–56.

[30] L. Alrahis, M. Yasin, N. Limaye, H. Saleh, B. Mohammad, M. Alqutayri *et al.*, "ScanSAT: Unlocking static and dynamic scan obfuscation," *IEEE Trans. Emerg. Topics Comput.*, pp. 1–1, 2019.

[31] M. Yasin and O. Sinanoglu, "Transforming between logic locking and ic camouflaging," in *Int. Design Test Symp.*, 2015, pp. 1–4.

[32] P. Chakraborty, J. Cruz, A. Alaql, and S. Bhunia, "SAIL: Analyzing structural artifacts of logic locking using machine learning," *IEEE Trans. Inf. Forensics Secur.*, vol. 16, pp. 3828–3842, 2021.

[33] M. Bushnell and V. Agrawal, *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Springer Publishing Company, Incorporated, 2013.

[34] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *CoRR*, vol. abs/1810.00826, 2018.

[35] H. Zeng, M. Zhang, Y. Xia, A. Srivastava, A. Malevich, R. Kannan *et al.*, "Decoupling the depth and scope of graph neural networks," *Proc. Adv. Neur. Inf. Proc. Sys.*, vol. 34, 2021.

[36] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," *Proc. Adv. Neur. Inf. Proc. Sys.*, vol. 31, 2018.

[37] M. El Massad, S. Garg, and M. V. Tripunitara, “Integrated Circuit (IC) Decamouflaging: Reverse Engineering Camouflaged ICs within Minutes.” in *Network and Distributed System Security*, 2015, pp. 1–14.

[38] The most cost-effective way to get critical analysis. [Online]. Available: https://www.techinsights.com/analysis-solutions/reverse-engineering/

[39] R. Saleh, S. Wilton, S. Mirabbasi, A. Hu, M. Greenstreet, G. Lemieux *et al.*, “System-on-chip: Reuse and integration,” *Proceedings of the IEEE*, vol. 94, no. 6, pp. 1050–1069, 2006.

[40] L. Alrahis, A. Sengupta, J. Knechtel, S. Patnaik, H. Saleh, B. Mohammad *et al.*, “GNN-RE: Graph neural networks for reverse engineering of gate-level netlists,” *IEEE Trans. on Comput.-Aided Design of Integr. Circuits and Syst.*, 2021.

[41] L. Cai, J. Li, J. Wang, and S. Ji, “Line graph neural networks for link prediction,” *IEEE Trans. on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021.

[42] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, “An end-to-end deep learning architecture for graph classification,” in *Thirty-second AAAI conference on artificial intelligence*, 2018.

[43] S. Davidson, “Notes on ITC’99 Benchmarks,” https://github.com/squillero/itc99-poli, 1999.

[44] D. Sisejkovic, F. Merchant, L. M. Reimann, H. Srivastava, A. Hallawa, and R. Leupers, “Challenging the security of logic locking schemes in the era of deep learning: A neuroevolutionary approach,” *ACM Journal on Emerg. Tech. in Comput. Syst.*, vol. 17, no. 3, 2021.

[45] J. Bhandari, A. K. Thalakkattu Moosa, B. Tan, C. Pilato, G. Gore, X. Tang *et al.*, “Exploring efpga-based redaction for ip protection,” in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, 2021, pp. 1–9.

Likhitha Mankali is a Ph.D. candidate at the Department of Electrical and Computer Engineering at Tandon School of Engineering, New York University, NY, USA. She is also a Global Ph.D. Fellow with New York University Abu Dhabi, UAE. Her research interests include Hardware Security, and using Machine learning for enhancing and quantifying the security of IP protection techniques.

Lilas Alrahis is a Postdoctoral Associate at New York University Abu Dhabi. She received the M.Sc. degree and the Ph.D. degree in electrical and computer engineering from Khalifa University, UAE, in 2016 and 2021, respectively. Her research interests include Hardware Security, Design for Trust, Logic Locking, and Applied Machine Learning. Dr. Alrahis won the MWSCAS Myril B. Reed Best Paper Award in 2016 and the Best Paper Award at the Applied Research Competition held in conjunction with Cyber Security Awareness Week, in 2019. Dr. Alrahis is currently serving as Associate Editor of the Integration, the VLSI Journal.

Satwik Patnaik is a Postdoctoral Researcher with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX, USA. He received his Ph.D. degree in Electrical engineering from Tandon School of Engineering, New York University, NY, USA, in September 2020. His research delves into semiconductor supply chain security with a prime focus on IP protection techniques and hardware Trojans. He has developed several computer-aided design frameworks for incorporating security, and his research leverages the 3D paradigm for security, exploits security properties of emerging devices, and utilizes machine learning, reinforcement learning, and game theory techniques for enhancing hardware security. Dr. Patnaik received the Bronze Medal in the Graduate Category at the ACM/SIGDA Student Research Competition held at ICCAD 2018, the Best Paper Award at the Applied Research Competition (ARC) held in conjunction with Cyber Security Awareness Week (CSAW) in 2017, and the third place at ARC in 2021. He has co-organized global hardware security competitions (HeLLO: CTF 2021, AI vs. Humans 2022) and is an active reviewer for premier journals and peer-reviewed conferences.

Johann Knechtel received the M.Sc. degree in Information Systems Engineering (Dipl.-Ing.) and the Ph.D. degree in Computer Engineering (Dr.-Ing., summa cum laude) from TU Dresden, Germany, in 2010 and 2014, respectively.

He is a Research Scientist with New York University Abu Dhabi, United Arab Emirates. From 2015 to 2016, he was a Postdoctoral Researcher with the Masdar Institute of Science and Technology, Abu Dhabi; from 2010 to 2014, he was a Ph.D. Scholar with the DFG Graduate School “Nano- and Biotechnologies for Packaging of Electronic Systems” hosted at TU Dresden; in 2012, he was a Research Assistant with the Chinese University of Hong Kong; and in 2010, he was a Visiting Research Student with the University of Michigan at Ann Arbor, MI, USA. His research interests cover VLSI physical design automation, with particular focus on emerging technologies and hardware security. He has (co-)authored around 50 publications.

Ozgur Sinanoglu is a professor of electrical and computer engineering at New York University Abu Dhabi. He obtained his Ph.D. in Computer Science and Engineering from University of California San Diego. He has industry experience at TI, IBM and Qualcomm. During his Ph.D. he won the IBM Ph.D. fellowship award twice. He is also the recipient of the best paper awards at IEEE VLSI Test Symposium 2011 and ACM Conference on Computer and Communication Security 2013.

Prof. Sinanoglu’s research interests include design-for-test, design-for-security and design-for-trust for VLSI circuits, where he has more than 200 conference and journal papers, and 20 issued and pending US Patents. Prof. Sinanoglu is the director of the Center for CyberSecurity at NYU Abu Dhabi. His recent research is being funded by US National Science Foundation, US Department of Defense, Semiconductor Research Corporation, Intel Corp, and Mubadala Technology.