Sādhanā (2024)49:169
https://doi.org/10.1007/s12046-024-02461-8

GAN4IP: A unified GAN and logic locking-based pipeline for hardware IP security

JUGAL GANDHI[1,2], DIKSHA SHEKHAWAT[1,2], M SANTOSH[1,2] and JAI GOPAL PANDEY[1,2,*]

[1]CSIR - Central Electronics Engineering Research Institute, Pilani 333031, India
[2]Academy of Scientific and Innovative Research (AcSIR), Ghaziabad 201002, India
e-mail: jugalacsir@ceeri.res.in; diksha@ceeri.res.in; msantosh@ceeri.res.in; jai@ceeri.res.in

MS received 27 July 2023; revised 25 November 2023; accepted 3 January 2024

Abstract. Intellectual property (IP) security has emerged as a critical concern in semiconductor industries. In the domain of hardware IP security, logic locking is a commonly used technique to prevent unauthorized access to IPs. This article proposes a conceptual pipeline to enhance the hardware IP security by leveraging generative models and logic locking concepts (GAN4IP) for hardware IP security. The proposed approach uses the concept of logic locking and generative adversarial networks (GANs) in a unified fashion to design secure hardware IPs. The GAN architecture uses deep learning techniques and graph-based representations of digital circuits to build obfuscated designs that can predict the behavior of locked netlists and generate secure designs. The proposed perspective method opens up new avenues for further investigation of highly secure electronic system design and has the potential to significantly impact the field of hardware IP security.

Keywords. Hardware security; IP security; logic locking; graph neural networks (GNNs); generative adversarial networks (GANs); GANs for hardware security.

1. Introduction

The increasing adoption of handheld edge devices connected to the Internet has led to the growth of the hardware security market, as it requires secure components to protect against cyber threats. There is a risk of breaching devices that do not have adequate hardware security, leading to severe consequences such as identity theft, financial fraud, privacy breaches, etc. Hardware security is crucial to protect sensitive information, prevent theft of intellectual property (IP), and ensure the integrity of the use of electronic devices, circuits, and systems. According to a research report by MarketsandMarkets [1], the global size of the hardware security market is expected to reach $15.38 billion by 2025, with a compound annual growth rate (CAGR) of 8.2% during the 2020-2025 period. Due to limited resources and fast time-to-market, system-on-chip (SoC) designers utilize third-party IPs (3PIPs) and outsource the design, manufacturing, and testing services to various locations throughout the globe. As shown in the figure 1, the presence of adversaries results in IP theft or the insertion of malicious components into the integrated circuit (IC) design chain that leads to security breaches. Design for trust (DfTr) is an approach to secure hardware systems design that focuses on incorporating security measures in the IC design process. Among the DfTr technique, logic locking [2] has the potential to provide resilience against vulnerabilities throughout the IC design chain.

Logic locking secures the hardware IP/ICs by addressing security vulnerabilities in the modern IC design chain. It introduces security measures to hardware IPs at an early stage in the IC design flow, providing proactive defense against potential threats. Over the years, several logic locking strategies have been proposed to protect against various attacks and minimize design overhead [3]. With the emergence of machine learning (ML) algorithms, researchers have initiated to use the potential of ML algorithms to improve system security [4]. Recent attacks such as SAIL [5], OMLA [6], MuxLink [7], UNTANGLE [8], Titan [9], etc. target ML-guided attacks in logic locking [3].

Manual design processes are time-consuming and prone to human error, making them less scalable and efficient. The transition from manual design to automated design of logic locked netlist using data and features from existing defense designs considerably improves the efficiency and effectiveness of logic locking. Using data and features from existing defense designs, a generative approach captures the knowledge and patterns observed in logic locking techniques. Through ML algorithms and techniques, the

*For correspondence

Published online: 05 May 2024

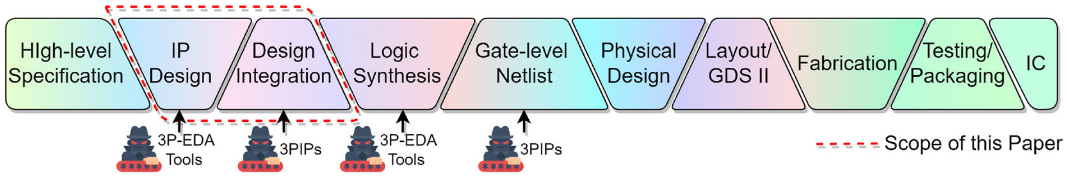

Figure 1. IC design flow and pre-silicon logic locking integration for defense against malicious agents.

automated design process analyzes the characteristics and properties of existing defense designs, extracts relevant features, and generates a logic locked netlist that exhibit similar security properties. This approach alleviates the workload for designers and improves consistency in the application of logic locking techniques. By automating the design process, designers can benefit from faster turnaround times, reduced design effort, and improved overall efficiency. In addition, it uncovers effective logic locking strategies that may not have been apparent through manual design.

Generative adversary networks (GANs) generate synthetic data using training experience with real data [10] to automate the design. Due to their ability to create high-fidelity data, GANs have attracted a lot of attention over the past decade. As a result, numerous potential threats are investigated using GAN [11]. Gandhi *et al* [3] reviewed ML algorithms in logic locking, Alrahis *et al* [12] presented a review on the usage of graph neural networks (GNNs) in logic locking, Dutta *et al* [11] have provided a review on GANs in security, and Tauhid *et al* [13] provided ML-guided hardware and software IP security analyzes. The research challenge in existing techniques and the contribution of this article are as follows.

1.1 *Research challenges*

Developing effective defense techniques against IP piracy poses significant research challenges. The current logic locking defense scheme presents the following research challenges.

(i) *ML-based Defense Circuit Generator*: The literature on generative models for IP security that can be used for logic locking is insufficient.
(ii) *Design Scalability*: Manual review of hardware design is not much in practice, and scaling logic locking for larger and more complex designs is quite challenging.
(iii) *Insufficient Training Data*: Creating a substantial volume of realistic synthetic training data for ML algorithms can be challenging, especially for specific circuits or architectures.
(iv) *Generalization Across Circuits and Architectures*: The generation of effective and optimized defense circuits that can balance security and resource constraints, such as area and power consumption, remains a challenge.
(v) *Automating Logic Locking*: Manual approaches are labor-intensive, time-consuming, and prone to human errors. Automating the logic locking process enables an efficient and consistent application of it to circuits and reduces the burden on designers.

GAN-based hardware IP security with logic locking can provide a unified potential solution to some of the challenges in existing work. This approach involves embedding the logic locking technique directly at the register transfer level (RTL) design. The GAN model is trained on existing obfuscated designs of various defenses and generates an obfuscated design that is functionally equivalent to the original design with protection against attacks.

1.2 *Proposed technical contributions*

The proposed research contributes to the field of hardware security by integrating ML techniques with traditional logic locking methodologies, providing a robust and efficient solution for IP protection. This research article presents a framework that combines GAN and logic locking (GAN4IP) to improve hardware IP security. These two techniques can be used to perform generative tasks on graph-structured data, allowing the generation of synthetic graph-structured data that are similar to the locked graph while preserving the structure and relationships inherent in the original graph netlist. The article outlines a comprehensive pipeline that covers data generation, model training, defense generation, security evaluation, optimization, graph-to-RTL conversion, and RTL verification for effectively addressing challenges in logic locking techniques. Using GANs, the proposed approach automates logic locking, improves security, and mitigates vulnerabilities in hardware designs. The article discusses the advantages of GAN-based logic locking and identifies potential areas for future research and development.

2. Preliminaries

Logic locking is a hardware IP security technique that introduces obfuscation and complexity. The integration of ML approaches enhances effectiveness and robustness of logic locking [2, 14]. An overview of logic locking and its integration with ML is discussed in the following section.

2.1 *Introduction to logic locking*

Logic locking, as discussed in [15], is a technique used to protect hardware IPs by inserting additional circuitry into the design, making it difficult for attackers to reverse engineer or clone the design. The fundamental concept behind logic locking involves the incorporation of additional gates controlled by a secret on-chip key into the original design. These additional gates create a functionally equivalent but obfuscated version of the original design. The on-chip hardware key controls the output of the netlist, enabling the design to be locked and triggered. When the on-chip hardware key is incorrect, the additional gates produce incorrect output, making it difficult for attackers to understand the functionality of the design [16]. When the key is correct, the additional gates produce the correct output, allowing users to operate the design correctly. Obfuscation techniques include adding additional gates to the design, including AND/OR/NOT/XOR gates, which are functionally unnecessary but serve to obscure the underlying logic in the design. Once the design is locked, it can only be unlocked by providing the correct key. This makes it very difficult for an adversary who does not have the key to reverse engineer or copy the design. The use of logic locking is becoming increasingly important as the value of IP in the semiconductor industry continues to grow.

Some existing logic locking technique(s) are challenging to employ as they require substantial modifications to the design, potentially leading to increased area and power consumption [17]. RTL locking is user-friendly as it does not require significant modifications to the design and can be implemented using existing design tools and flows. Although RTL locking shows a potential solution to some of the problems associated with existing logic locking techniques, it is not yet an optimum solution. Researchers continue to develop new logic locking techniques that are more secure, efficient, and user-friendly [18]. Logic locking has gained significance as a technique to protect IPs in applications such as security, military, finance, etc., where the consequences of IP theft can be substantial.

2.2 *Applications of machine learning in logic locking*

Machine learning has been widely explored to improve the security of hardware IPs [4, 13], and [19]. It has been used for various tasks, such as IP watermarking [20], hardware Trojan (HT) detection [21], logic locking [2], etc. In IP watermarking, ML algorithms are used to embed a unique signature or watermark into the design that can be used later to prove ownership or detect unauthorized use of IPs [20]. In HT detection [22, 23], and [24], ML algorithms are used to identify the presence of malicious circuits in the design that leak potentially sensitive information or cause a denial of service attack. ML techniques have found application in improving the effectiveness of logic locking attacks, as discussed in [6, 8], and [25]. In particular, ML algorithms are used to predict the correct key used to lock a design, effectively bypassing the logic locking mechanism.

3. Generative adversarial network (GAN)

GAN is a deep learning model that is used for unsupervised machine learning tasks. As depicted in figure 2, GANs rely on conditioning between two neural networks, a generator (G) and a discriminator (D), in an adversarial manner to generate new data samples that are indistinguishable from real data. The generator network receives random noise as an input vector and generates data samples designed to resemble real data. The discriminator network evaluates these generated samples and attempts to differentiate them from real data. The generator aims to produce samples that can deceive the discriminator, as they are real, while the discriminator strives to improve its ability to distinguish generated samples. This forms a min-max game between G and D, as given in (1), in which the generator aims to minimize the loss, while the discriminator seeks to maximize it. The GAN training process persists until the generator generates samples that are indistinguishable from the real data. This is often achieved when the generator and the discriminator reach a Nash equilibrium, as described in [10], which means that the generator cannot be improved by changing its strategy. The final result is a generator network that can produce new samples similar to the real data. Formally, G and D play two-player min-max game with value function $V(G,\ D)$ as:

$$\min_G \max_D \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[1 - \log D(G(z))] \tag{1}$$

3.1 *Applications of GAN in hardware security*

GANs have recently emerged as a promising tool in the domain of hardware security [11]. GANs are increasingly finding applications in hardware security research.

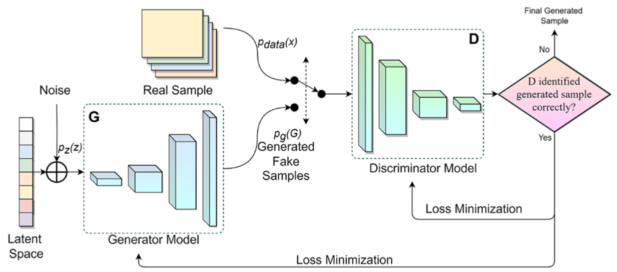

Figure 2. An abstract architecture of GAN [10].

Some examples of GANs in hardware security are as follows.

(i) *Side-channel Attacks*: GANs generate synthetic power traces or electromagnetic radiation signals that mimic the behavior of devices [26]. This facilitates side-channel attack analysis and countermeasure testing without risking the security of devices.
(ii) *Hardware Trojans*: A GAN-based anomaly detection method proposed in [27] is used to identify HTs in multicore systems, focusing on high-performance, low-power, and secure on-chip communications.
(iii) *Hardware Security Evaluation*: GANs are used to generate synthetic hardware designs that can be used for security evaluation [28]. When a large number of designs are generated with different parameters, it is possible to study the impact of different design choices on the system security.
(iv) *Obfuscation (Proposed Pipeline)*: GANs can be used to generate obfuscated circuits that are difficult to reverse engineer. By training GANs to generate circuits that have similar functionality but different structures, it is possible to hide the actual circuit design and make it difficult for attackers to extract sensitive information.

GANs have shown great potential in hardware security research and applications. They are used to generate synthetic data and designs that can be used for evaluation and testing and to generate obfuscated circuits and Trojans.

3.2 *Advantages of GAN in logic locking*

GANs are well-suited for logic locking due to their distinct features, which enhances the security and protection of hardware IP/ICs. The advantages of GAN are as follows.

(i) *Generative Capability*: GANs possess the ability to generate realistic and diverse samples, rendering them suitable for generating logic locked circuits with improved security features. GANs can learn the underlying patterns and structures of the original designs and generate locked circuits that preserve the desired functionality while incorporating obfuscation techniques.
(ii) *Adversarial Training*: GANs utilize an adversarial training framework consisting of a generator and a discriminator network. This adversarial approach enables the generator to continuously improve its ability to generate more secure and resilient locked designs. In contrast, the discriminator learns to distinguish between locked designs and original designs. This adversarial training process contributes to the enhancement of security properties in the generated logic locked circuits.
(iii) *Data-driven Approach*: GANs leverage large-scale datasets to learn the underlying data distribution and pattern. By training on a diverse set of RTL designs, GANs capture the characteristics and variations present in the dataset, enabling them to generate logic locked circuits that exhibit similar characteristics. This data-driven approach improves the fidelity and effectiveness of the generated locked designs.
(iv) *Flexibility and Customization*: GANs provide flexibility in terms of incorporating various obfuscation techniques and security measures into the generated logic locked circuits. Designers can customize the GAN architecture, loss functions, and training strategy to achieve specific security objectives, such as resistance against reverse engineering, side-channel attacks, etc.
(v) *Scalability*: GANs efficiently manage large-scale datasets and generate logic locked circuits. Once the GAN model is trained, it can generate locked designs, facilitating the application of logic locking to complex and extensive hardware IPs.

The distinct features and advantages outlined above establish GANs as a promising option for logic locking, providing designers with the means to reinforce the security of their hardware IPs.

4. Graph neural network (GNN)

A graph neural network (GNN) is a neural network designed to work with structured data represented as graphs [29]. Graphs serve as representations for objects and their relationships in various domains, such as social networks, biology, chemistry, transportation networks, circuits, etc. [30]. In a GNN, each graph node is associated with a feature vector, and each edge is linked to a weight. The GNN iterative updates the node representations by aggregating information from neighboring nodes, possibly incorporating edge features. This process iterates multiple times, enabling the network to progressively integrate information into the graph. GNNs have shown effectiveness in various tasks, including node classification, link prediction, and graph classification [31].

In the context of hardware IP security, GNNs can serve to identify unauthorized or malicious modifications to the design of a hardware IP block [32, 33]. GNNs can be applied to model the graph structure of a hardware IP block and learn representations of components and their connections. Subsequently, the GNN identifies modifications to the graph structure that indicate unauthorized or malicious changes, such as insertion of Trojan circuits or the elimination of security features. GNNs manage complex graph structures and identify modifications that may be difficult to detect using traditional methods. In addition, GNNs exhibit adaptability to emerging threats and design variations by learning from extensive labeled and unlabeled data.

4.1 *Need of graph representation and GNN*

Graphs serve as a natural way to represent complex hierarchical structures that are common in IC designs. By representing the netlist as a graph, designers gain improved insight into the interconnections among components, facilitating the identification of potential design problems. Graphs provide a convenient framework for the application of advanced graph-based algorithms and ML techniques. For example, graph-based algorithms, such as graph partitioning and clustering, can be beneficial in optimizing the placement and routing of design components. GNNs serve to learn patterns within the graph structure and derive features for tasks such as classification and regression. Transforming a netlist into a graph allows the application of automated graph-based techniques to verify the accuracy of a design and identify potential errors or vulnerabilities. The role of GNN is to extract essential features and relationships among the components of a netlist, generating a graph representation. In the context of logic locking, GNNs represent a locked design as a graph where the nodes correspond to gates and the edges correspond to their interconnections.

5. A proposed pipeline for GAN4IP

Utilizing a GAN-based defense in conjunction with logic locking can offer an efficient, scalable, and automated defense generation approach to enhance hardware IP security. A conceptual pipeline of GAN4IP is depicted in figure 3. The steps of the proposed pipeline are dataset preparation, model training, defense generation, functional verification and security evaluation, optimization, and RTL reconstruction.

The generative model-based IP security pipeline starts with the *dataset preparation phase*. It involves collecting a substantial dataset consisting of input-output pairs of locked and unlocked circuits. The quality and size of the dataset have a significant impact on the performance of the GAN model. A larger and more diverse dataset improves the ability of the model to generate realistic output vectors. The quality of the dataset is also critical, as it affects the ability of the model to generalize new inputs. Upon defining the dataset, the subsequent pivotal step involves feeding it to the discriminator and generator models. This initiates the training process, enabling the networks to learn the intricate parameters of logic locking from the dataset.

In the *model training phase*, the GAN model is trained on a dataset consisting of predefined defense designs. The primary objective of model training is to train the GAN to learn the underlying circuit design space distribution. The discriminator model is trained on locked graphs, extracting key parameters distinctive to logic locked circuits. In parallel, the generator model utilizes unlocked netlist graphs and introduces security logic that ensures functionality with the original circuits by minimizing the loss function. The ongoing Min-Max game between the networks involves the generator model striving to enhance its ability to generate functionally equivalent locked graphs that defy identification with any specific locking mechanism.

Once the GAN model is trained, the next is a *defense generation phase.* This utilizes the trained generator network to produce a locked version of the original graphs. This locked netlist graph includes additional gates or circuits designed to confuse attackers who attempt to reverse engineer the circuit. Subsequently, *verification and security checks* are performed on the generated locked graphs for functional verification and security. The functional verification ensures that the locked circuit behaves correctly and produces the expected outputs for the given inputs. Security analysis comprises of evaluating circuit resiliency against known attacks and effectiveness of defense mechanisms considering security parameters such as key effect, output corruptibility, etc. [34].

In the final stage of security and functional verification for the generated logic locked graph, the *optimization and RTL reconstruction* have been performed. Here, the final stages of the GAN-based logic locking defense process involve optimization and RTL reconstruction.

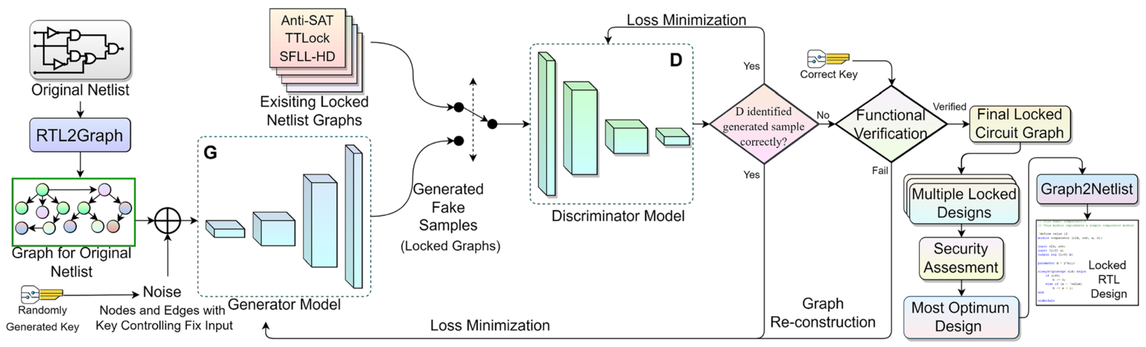

Figure 3. Proposed unified GAN and logic locking based pipeline for hardware IP security.

The optimized locked graph is extracted, considering the security, resources, and performance parameters. Subsequently, the optimized locked graph is transformed into an RTL design using the Graph2RTL technique. The details of the above steps have been explained from section 5.1 to section 5.8.

5.1 *Dataset generation/RTL2graph*

To create a dataset for the GAN-based logic locking, it is important to collect a diverse set of locked circuit samples. These encompass different key sizes, complexities, and functionalities relevant to the target application domain. The dataset should include various benchmark circuits such as ISCAS '85/'89, MCNC, etc., or custom circuits that capture the features and variations of real-world scenarios. This diverse collection serves as a foundation for the dataset generation process, enabling the generation of locked circuit samples and subsequent training of the GAN model. In GAN-based logic locking, the initial phase involves preparing the dataset used to train the ML model. As illustrated in figure 4, the RTL2Graph is utilized to generate graphs for original and logic locked circuits [24, 35]. This involves extracting the node features and matrices for each node, followed by generating nodes for further embedding into the original graph. The dataset generation process consists of four key steps, which are RTL pre-processing, HDL parsing, node, edge and graph generation, and graph optimization.

The pre-processing phase transforms multiple high-level circuit description modules into a single RTL netlist representing the circuit structure. This netlist is then subjected to parsing, creating an abstract syntax tree (AST) that captures the structural information of the RTL design. The subsequent step of generating nodes involves translating each basic block of RTL components into nodes within the graph, with each node representing a physical or logical circuit component. Edge generation establishes connections between nodes based on data flow, reflecting dependencies in the RTL code. The graph is generated by combining the nodes and edges, followed by optimization and merging to reduce complexity. Once optimized graphs are generated, the feature extraction phase meticulously gathers intricate details about components, connectivity, and functionality from both nodes and edges. This meticulous extraction process ensures that essential attributes are captured, laying the foundation for in-depth and insightful subsequent analyses. The specific design steps within each phase are described below.

5.1.1 *Pre-processing*

The pre-processing starts with the identification of the top-level module responsible for connecting and integrating all other modules in the design. This module serves as an entry point that facilitates the connection of various sub-modules. Then the RTL code from the original source files for each module is extracted. After determining its input, output, and internal signals, the RTL codes are merged from each module into a single RTL code file using the top-level module as the starting point. During this merging process, potential conflicts such as duplicate signal names or incompatible data types are identified and resolved through appropriate measures, such as signal renaming or code modification. Finally, the integrity of the merged RTL code is validated through comprehensive simulations, ensuring the seamless integration and functionality of the design.

5.1.2 *HDL parsing*

The HDL parsing phase is critical in the process of generating an AST from HDL code, which involves steps such as lexical and syntax analysis, AST generation, semantic analysis, and optimization. Initially, lexical analysis, or tokenization, divides the HDL code into tokens, including keywords, identifiers, operators, and literals. Subsequently, the syntax analysis checks for adherence of the code to the HDL syntax rules. Upon completion, AST generation ensues, creating a structural representation of the HDL code, inclusive of nodes for modules, ports, signals, and statements. After AST

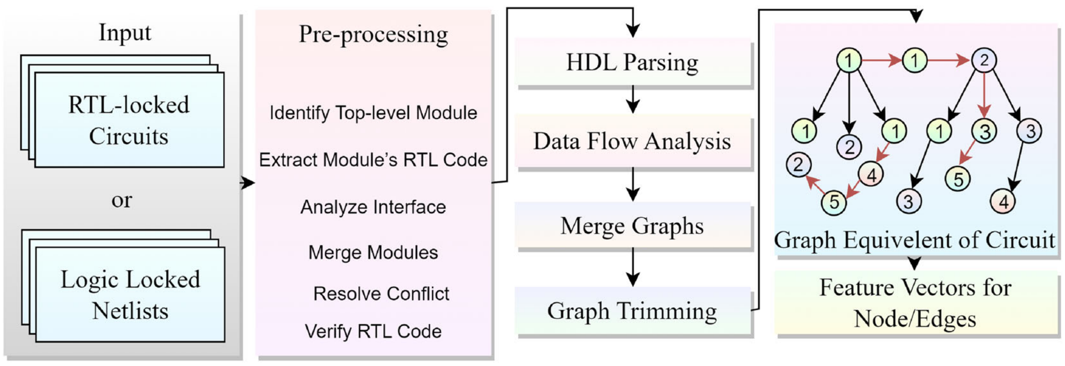

Figure 4. Flow for RTL2Graph generation and feature extraction for pre-processing of GAN4IP.

generation, the semantic analysis step includes type checking, name resolution, and scope analysis to ensure HDL code accuracy. Finally, AST optimization has been performed to improve performance or readability. This optimization process involves actions such as merging or eliminating redundant nodes or adding annotations to enhance visualization.

5.1.3 *Generation of nodes, edges, and graphs* This process involves the identification of nodes and edges followed by the construction of a graph. First, the nodes are identified, each representing components such as combinational logic blocks, and sequential logic blocks. These nodes are labeled with detailed information, including the component's name, input-output ports, type, etc. The graph edges are then identified, representing node connections and showing the data flow between the components. Edges are labeled with features such as source and destination nodes, connection type, and associated delays. Finally, the graph is constructed by organizing the nodes and edges to accurately reflect the structure and behavior of the hardware design .

5.1.4 *Graph optimization* The graph optimization phase is integral to the process of generating a graph from the HDL code. It aims to optimize the graph representation by minimizing its size and complexity while preserving the functionality of the design. In the initial stage, redundant nodes, and edges within the graph are identified using reachability analysis, fanout analysis, and common subexpression elimination techniques without affecting design functionality. Furthermore, redundant nodes and edges are merged without changing the design behavior. For example, multiple input ports can be merged into a single input port or multiple nodes performing the same operation can be merged into a single node. Once redundant nodes/edges are identified and merged, in the graph simplification stage, the graph can be further simplified by removing unnecessary components and connections from the graph to reduce size and complexity. Finally, the optimized graph must undergo verification to ensure that it retains the functionality of the original design using verification methods, such as simulation, formal verification, equivalence checks, etc., can be used for this purpose.

5.2 *Model training*

In GAN-based locking techniques, the model training phase, as illustrated in figure 5, involves training the neural network of the generator and discriminator. The role of the generator network is to generate new locked graphs that are functionally similar to the original netlist graph. The discriminator network is trained to differentiate between the existing defense-based netlist graph and the generated graph. The generator network receives feedback from the discriminator network to improve its ability to generate realistic samples. The steps involved in the model training stage are as follows.

(i) *Network Architecture Design*: It involves selecting the suitable architecture for the generator and the discriminator. The architecture should be optimized for the specific problem and capable of generating realistic and diverse circuits. Models such as DeepWalk [40], GraphGAN [50], LINE [51], Node2Vec [42], etc., can be used to perform GAN4IP. The existing graph embedding techniques outlines in table 1.

(ii) *Loss Function Design*: The loss function guides the training process and ensures that the generator generates realistic circuits and the discriminator correctly classifies them. A careful design of the loss function is necessary to strike a balance between the objectives of the generator and the discriminator. This balance enables the generator to produce realistic locked circuits, while the discriminator effectively distinguishes between real and generated logic locked graphs. Some commonly used loss functions in GAN include binary cross-entropy, mean squared error, and adversarial loss.

(iii) *Training Procedure*: Generator and discriminator networks are trained in an adversarial manner. In

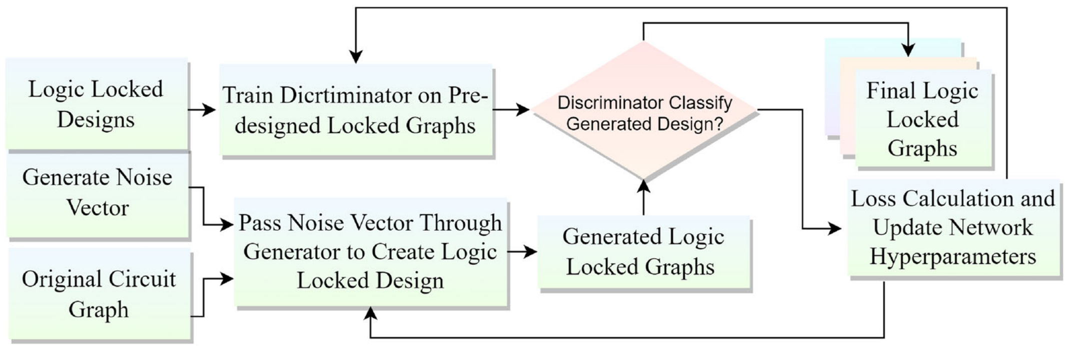
Logic Locked Designs
Train Dicrtiminator on Pre-designed Locked Graphs
Discriminator Classify Generated Design?
Final Logic Locked Graphs
Generate Noise Vector
Pass Noise Vector Through Generator to Create Logic Locked Design
Generated Logic Locked Graphs
Loss Calculation and Update Network Hyperparameters
Original Circuit Graph

Figure 5. Flowchart for the model training phase of GAN4IP.

Table 1. Existing graph embedding techniques [36].

Scheme		Method	Time Complexity	Description
Shellow graph embedding method	Factorization	LLE [37]	$O(\lvert E\rvert d^2)$	Global structure recovery
		Laplacian [38]	$O(\lvert E\rvert d^2)$	Factorization on laplacian
		HOPE [39]	$O(\lvert E\rvert d^2)$	Preserve asymmetric transitivity
	Random walk	DeepWalk [40]	$O(\lvert V\rvert d)$	First Extended Skip-gram model for graph
		Planetoid [41]	$O(\lvert V\rvert d)$	Integrated label information for embeddings
		Node2Vec [42]	$O(\lvert V\rvert d)$	Biased random walk procedure
		HARP [43]	$O(\lvert E\rvert d)$	General hierarchical meta-strategy
AutoEncoder-based graph embedding		SDNE [44]	$O(\lvert E\rvert\lvert V\rvert)$	First AutoEncoder based model
		GAE [45]	$O(\lvert V\rvert\lvert E\rvert)$	Reconstruct the adjacency matrix using GCN
		ARGAE [46]	$O(\lvert V\rvert\lvert E\rvert)$	Improve VGAE with GAN
Deep Graph Embedding method		GCN [47]	$O(\lvert E\rvert)$	Most influential GNN method
		GraphSAGE [48]	$O(\lvert V\rvert)$	Inductive framework for GNN
		GraphGNN [49]	$O(\lvert V\rvert\lvert E\rvert)$	GNN + RNN combination

each iteration, the generator generates a batch of synthetic locked circuits, which are then fed to the discriminator with a batch of logic locked circuits from the training dataset. The discriminator's objective is to classify circuits as either real or synthetic, while the generator's goal is to generate circuits capable of deceiving the discriminator.

(iv) *Hyperparameter Tuning*: To achieve optimal model performance, it is essential to fine-tune hyperparameters such as learning rate, batch size, number of layers, neurons per layer, etc. This involves a trial-and-error-based approach, in which various combinations of hyperparameters are evaluated on a validation set, and the combination yielding the best performance is selected for logic locked graph generation.

(v) *Evaluation*: After training the GAN, it is crucial to evaluate its performance using a separate testing dataset to assess its performance and subsequently, generate defense graphs for the original circuit.

5.3 *Generation of defense Netlist graphs*

In this phase, a set of defense netlist graphs is generated using the neural network trained generator. The transformation of the original circuit graph and the addition of nodes and edges to generate a locked graph are shown in figure 6. The steps involved in the defense generation phase are described below.

The defense generation phase starts with the selection of defense samples. This is used to select a set of circuits from the training dataset that serves as the foundation for defense generation. The trained graphs should be locked netlists which are targeted to mimic. This selection is aligned with the training data used for the discriminator, allowing it to capture the features of the locking mechanisms. The selected circuits are subsequently parsed into the trained generator network, which generates synthetic graphs integrated into the original graph circuit and functionally equivalent when applying correct key bits.

The synthetic circuits are then evaluated to ensure functional equivalence to the original circuits. This evaluation may involve running a testbench on the synthetic circuits to ensure that they behave correctly. After evaluation and verification of the synthetic circuits, they are integrated with the original design to create a logic locked circuit. This involves the replacement of specific components within the original design with synthetic circuits or the addition of synthetic circuits as additional layers of protection. Finally, generated designs are verified to ensure the features of the training logic locked samples. In back-propagation and minimizing the generator loss function, multiple locked designs are framed that are used to fool the discriminator network.

5.4 *Functional verification of generated graphs*

Functional verification of a graph-based defense circuit involves ensuring its correct behavior and producing the expected output by inserting golden input pairs. The functional verification pipeline of the generated graphs is shown in figure 7. A testbench is designed to accurately represent the functionalities of the original circuit. Subsequently, this testbench is utilized for simulations using EDA tools. The simulations cover a diverse range of input vectors, ensuring a comprehensive evaluation of the defense circuit's functionality under different conditions. The simulation results are then compared with the expected output to identify any discrepancies. In the cases where discrepancies are detected, iterative modifications are made to the defense circuit until it consistently produces the correct results. The performance of the defense circuit is verified to ensure that it is in line with the required hardware specifications. This comprehensive verification process ensures the reliability and effectiveness of the generated defense circuit in different operational scenarios.

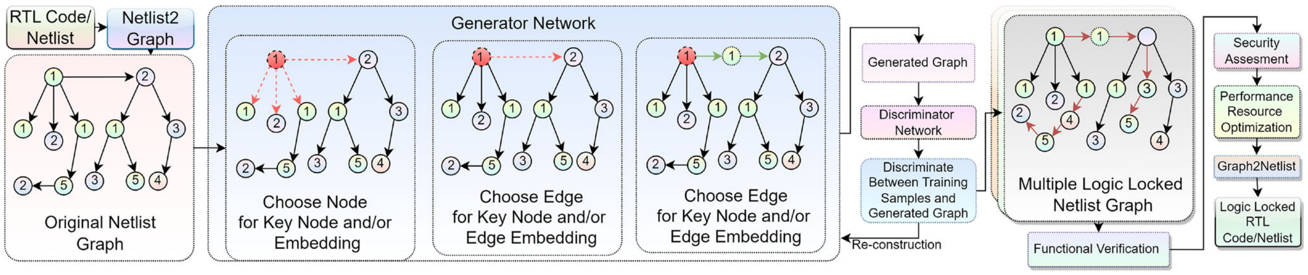
RTL Code/ Netlist
Netlist2 Graph
Original Netlist Graph
Generator Network
Choose Node for Key Node and/or Embedding
Choose Edge for Key Node and/or Edge Embedding
Choose Edge for Key Node and/or Edge Embedding
Generated Graph
Discriminator Network
Discriminate Between Training Samples and Generated Graph
Re-construction
Multiple Logic Locked Netlist Graph
Functional Verification
Security Assesment
Performance Resource Optimization
Graph2Netlist
Logic Locked RTL Code/Netlist

Figure 6. Transformation of a graph from original to logic locked graph in GAN4IP.

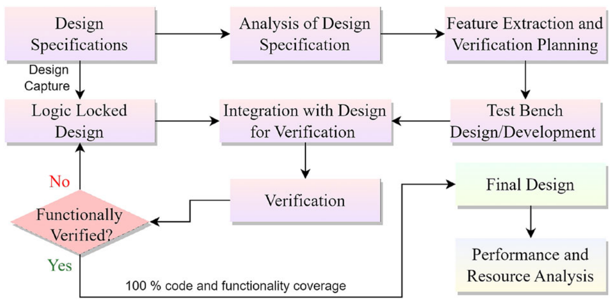
Design Specifications
Analysis of Design Specification
Feature Extraction and Verification Planning
Design Capture
Logic Locked Design
Integration with Design for Verification
Test Bench Design/Development
No
Functionally Verified?
Yes
Verification
Final Design
Performance and Resource Analysis
100 % code and functionality coverage

Figure 7. Functional verification phase of generated graphs.

5.5 *Security evaluation of designs*

Once the logic locked designs are functionally verified, the security of each circuit graph is evaluated based on security parameters [3], including average Hammering distance, key correctness, key effect, output error rate, area and power overhead, etc. Security evaluation of the design phase is a critical aspect of logic locked graph validation, where existing attacks or security metrics are employed to verify the security of the generated graphs. This phase aims to evaluate the resilience of the logic locked design against potential attacks and to ensure that the desired security properties are effectively enforced. During the security evaluation, various attack techniques, such as SAT-based attacks [52], structural analysis [53], SCOPE [54], etc., can be used to test the strength of the logic locked graph. These attacks attempt to break the locking mechanism and reveal the original design.

5.6 *Design selection for need of application*

The most optimal design for logic locking depends on various factors, such as security level, circuit size, power constraints, available resources, etc. Typically, a good design balances security with overhead in terms of area, power, and performance. It is important to note that an optimal design would also depend on the specific application and security required. For example, a high-security application may have high overhead in area and power, while a low-security application prioritizes performance and efficiency over security.

5.7 *Graph-to-RTL (Graph2RTL)/RTL reconstruction*

Graph2RTL reconstruction is the process of converting a graph representation of a digital circuit back into its corresponding RTL representation. This process aims to reconstruct a semantically equivalent RTL implementation of the original graph. The reconstruction flow, illustrated in figure 8, typically involves the following steps:

(i) *Node Mapping*: Initially, the nodes in the graph need to be assigned to their corresponding RTL components. Each node in the graph represents a specific function implemented by an RTL component. The mapping involves identifying the appropriate RTL component for each node.

(ii) *Edge Mapping*: The edges of the graph are assigned to the wires in the RTL design. This involves determining the routing and connectivity of the wires based on the graph topology.

(iii) *Connection Mapping*: After mapping the nodes and edges to the RTL components and wires, the connections between these components must be established. This requires identifying the correct input and output ports for each component and ensuring that the connections between them are consistent with the original RTL implementation.

(iv) *Signal Renaming*: In some cases, the signal names in the graph representation may vary from the original RTL code. This step updates the signal names to match the original RTL code to ensure correct functionality.

(v) *RTL Generation*: In this step, the RTL code is generated by instantiating the RTL modules mapped from the nodes in the graph and connecting them together with the interconnect wires mapped from the edges in the graph. The RTL code is optimized to minimize area and

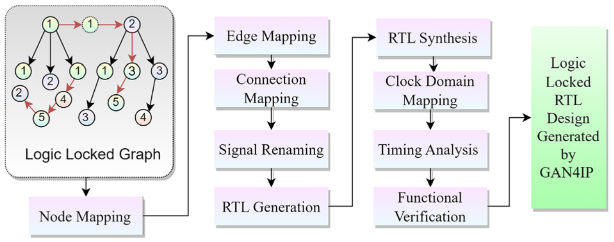

Figure 8. Transformation of graph-to-RTL in GAN4IP.

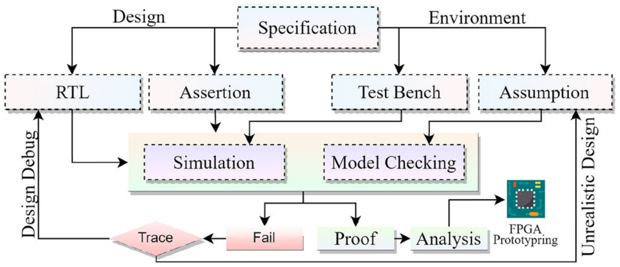

Figure 9. Steps involved in the RTL verification.

power overhead while maintaining the functionality of the original graph.

(vi) *RTL Synthesis*: Once the graph has been mapped to the RTL components and interconnects, the next step is to synthesize the RTL code. This involves generating a netlist from the RTL code, which is used for further design implementation.

(vii) *Clock Domain Mapping* (CDC): It is a critical issue in digital design and must be properly handled during Graph2RTL reconstruction. The clock domains of each component must be correctly identified, and the necessary clock domain crossing logic must be added to ensure that the design functions correctly.

(viii) *Timing Analysis*: It must be performed to ensure that the reconstructed RTL implementation meets the desired timing constraints. This involves identifying critical paths in the design and verifying that their timing is within acceptable limits.

(ix) *Verification*: The reconstructed RTL code is verified to ensure that it is semantically equivalent to the original graph representation. This is typically done using formal verification tools or simulation-based techniques.

5.8 *RTL verification of logic locked design*

As shown in figure 9, the verification phase includes several steps to ensure the functionality of the generated RTL netlist. The initial phase involves functional verification, ensuring the preservation of the original circuit behavior. This includes simulation with diverse input vectors and comparison of the output with the original circuit. Following this, the timing verification step meticulously scrutinizes the defense circuit's adherence to specified timing constraints, encompassing considerations such as clock cycle time, signal set-up, and hold-time requirements. Subsequently, power analysis is undertaken to mitigate power overhead, focusing on dynamic power consumption during the operational phases. The hardware testing stage involves a comprehensive evaluation of both the functionality and security aspects of tangible hardware such as a field-programmable gate array (FPGA) or an application-specific integrated circuit (ASIC).

Summary: Using ML techniques, the logic locking mechanism can create synthetic locked circuits. The flow of the GAN-based logic locking defense mechanism, GAN4IP, includes the conversion of RTL2graph, feature extraction and preprocessing stages, dataset generation, model training, defense generation, verification and security check, optimization, and RTL reconstruction. Each stage plays a critical role in ensuring the effectiveness of the defense mechanism.

6. Evaluation setup and analysis

This section discusses the tools, setup, and evaluation analysis for the proposed pipeline/framework.

6.1 *Tools and languages*

To work for the proposed scheme, a diverse dataset has been generated using multiple benchmark schemes such as ISCAS' 85/89, MCNC, and custom-designed netlist. They have been used to perform GAN-based logic locking. To implement locking, design verification, etc., tools such as Yosys, ABC compiler for hardware IP synthesis, and `.bench` file generation, `PyTorch` for the integration of DGCNN-based link prediction and GAN modeling, C and C++ to calculate security assessment parameters, Perl for `.bench` to `.v` (Verilog reconstruction) and Xilinx Vivado for FPGA emulations, Synopsys Design Compiler for post-synthesis area, power and delay estimation, and overhead analysis have been used. Intel Xeon CPU @3.1GHz with 64GB RAM and a 4GB NVIDIA K2200 GPU is used to perform the design.

6.2 *Components evaluation*

The proposed pipeline comprises several intricate components and substantial progress has been made in its implementation. This section details the evaluation of the components that have been partially implemented, recognizing that further refinements are yet to be performed.

6.2.1 *Dataset generation* By leveraging open-source repositories containing locking schemes, the generated logic locked synthesis-based `.bench` files span different complexity levels of ISCAS'85 benchmark netlists. Locking techniques, including RLL [2], SFLL-HD [55], Anti-SAT [56], D-MUX [57], etc., have been applied with key sizes of 32, 64, and 128 bits. A collection of 10 distinct netlists has been generated for each key size, resulting in a total of 4319 unique netlists, each associated with a unique key value, as shown in table 2, creating a more diverse and distinct dataset for training, testing, and validation of the GAN model. The graphs have been created from the original netlists to serve as the foundational basis for the generator model. The original verilog netlist files are initially converted to `.bench` format. After the locking mechanism is applied, the design is further transformed into its corresponding directed graph representation. The directed graph is used to train the discriminator model to recognize the characteristics of the generated graph. This allows the model to provide feedback based on whether the generated graph exhibits the features of the training dataset.

6.2.2 *Security evaluation parameters* In terms of security assessment, two approaches can be considered: based on security metrics/parameters and attack resiliency assessment. Evaluation involves five primary security parameters, namely Hamming distance, key correctness, key effect, output error rate (OER), and key prediction accuracy. The evaluation parameters have been implemented using C++ for security analysis of logic locked netlists. The functionality can be tested for various random input-output combinations to account for correct and incorrect key inputs. These parameters provide preliminary insights into security, serving as a useful indicator before conducting a more comprehensive security assessment through attack simulations.

6.2.3 *Attack resiliency evaluation* The most crucial step is to evaluate the security of the design in existing state-of-the-art attacks. For attack simulation, the proposed scheme aims to achieve resilience against different attacks, including ML-based attacks using graphs as input, OG-guided attacks, or OL attacks using `.bench`/Verilog files as input. SAT [52], SCOPE [54], and MuxLink [7] attacks have been recreated to verify the security of the generated logic locked netlist. For evaluation of SAT attack, the `.bench` file of the locked netlist and the oracle (original netlist) can be used. On the other hand, the `.bench` and graph format netlist can be used for SWEEP and MuxLink attacks, as they are OL- and ML-guided attacks.

Table 2. Dataset utilized for the proposed approach includes various benchmark netlists, covering a range of locking schemes, each with a unique key corresponding to different key lengths (16 to 256 bits).

Locking Technique	Benchmark Dataset/ Custom	#number of Unique Netlist	#number of Unique Keys	Unique Key-netlist Pairs	#total Number of Netlists
RLL	Mix	21	32/64/128	10	630
MUX	Mix	21	32/64/128	4	252
XOR	Mix	21	32/64/128	1	63
AND/OR insertion	Mix	21	32/64/128	4	252
Interference clique	Mix	21	32/64/128	4	252
Anti-SAT	ISCAS'85	10	32/64/128	10	300
SARLock	Mix	12	64/128	15	540
SFLL	MIx	9	64/128/256	10	270
D-MUX	ISCAS'85	10*	32/64/128/256	10	350
Crosslock	MIx	10	32/64/128	10	300
Cyclic	ISCAS'85/89	10	64/128	10	200
Hybrid	Mix	25^	64/128	10	460
DK-lock	ISCAS'89	15#	32/64/128	10	330
LightLock	GIFT Light-weight IP	1	16/32/64/128	10	40
	Posit Accelerator	2	16/32/64/128	10	80

* c432, c880 locked using 32-bit and 32, 64-bit D-MUX locking, respectively

# s27, s298 locked using 32-bit DK-Locking

^ c432, s27 locked using 32-bit hybrid locking

Mix: Randomly selected netlists from multiple benchmark datasets such as ISCAS'85 or '89/MCNC/ITC'99, etc

Table 3. A comparison of the proposed pipeline with some existing frameworks

Pipeline/ framework	GNN task	Network used	Loss function	Graph type	Circuit type	Features	Generative Network
GNN4TJ [24]	Graph classification	GCN	Cross-entropy	DFG	RTL/GLN	Type of Node Oper.	✗
GNN4IP [19]	Graph similarity	GCN	Cosine-similarity	DFG	RTL/GLN	Type of Node Oper.	✗
GNN-RE [58]	Node classification	GraphSAINT	Cross-entropy	Undirected	GLN	Input/Output Degree Connectivity to ports gate type neighborhood Info.	✗
ABGNN [59]	Node classification	ABGNN	Cross-entropy	Directed	GLN	-	✗
ICNet [60]	Graph regression	GAT	MSE	Undirected	GLN	Gate Type Key-gate Mask	✗
OMLA [6]	Subgraph classification	GIN	Cross-entropy	Undirected	GLN	Gate type Connectivity to ports Distance encoding	✗
MuxLink [7] UNTANGLE [8]	Link prediction	DGCNN	Cross-entropy	Undirected	GLN	Gate Type distance encoding	✗
Titan [9]	Subgraph classification	GIN	Cross-entropy	Undirected	GLN	Gate Type, #Inputs Distance Encoding	✗
Deep H-GCN [61]	Node regression	H-GCN	MSE	Heterogeneous Directed	Post-layout TNL	Design Parameters	✗
GNN4REL [62]	Subgraph regression	PNA	MSE	Undirected	GLN	Gate Type	✗
Proposed scheme	Graph generation	GraphGAN GNN GraphSAINT Node2Vec	Adversarial loss/ KL Divergence loss	Directed	RTL/GLN	Design parameters Type of node oper. Input/Output degree connectivity to ports Gate type Neighborhood Info.	✓

7. A qualitative analysis of the proposed pipeline

GNNs have become a versatile solution to address a variety of hardware security challenges [12]. Yasaei *et al* [24] proposed GNN4TJ, a GNN platform for the detection of HT without prior knowledge of the IP design or the HT structure. GNN4TJ converts RTL designs into DFGs, which are fed to a GNN for structure learning and graph classification to predict GNN4IP [19] extends this framework for IP piracy detection, generating embedding to assess the similarity between circuits. HW2VEC [35] combines GNN4TJ and GNN4IP. Alrahis *et al* [58] propose GNN-RE for functional reverse engineering in GLNs, while He *et al* [59] introduce ABGNN for subcircuit classification. OMLA [6] attacks X(N)OR locking, MuxLink and UNTANGLE [8] break MUX-based locking, and Titan [9] combines these attacks. SAT-based attack [52] is addressed by ICNet [60], which uses GNN for runtime prediction. Alrahis *et al* [62] used GNN to predict delay degradation in digital ICs, while Chen *et al* [61] tackle aging-induced degradation in analog ICs using graph convolution network (GCN).

Table 3 provides a comprehensive comparison between the proposed GAN4IP conceptual pipeline and existing frameworks in terms of key characteristics, loss functions, applicability, etc., highlighting the unique possible advantages of the proposed approach. In the qualitative analysis of the GAN-based pipeline for logic locking in hardware IP security, several key strengths and advantages have been identified. With its ability to integrate locked designs back into the original hardware IP, the proposed pipeline may show promise in protecting against malicious attacks.

8. Conclusion

The research contributes to the field of hardware security by offering an advanced approach that combines ML techniques with logic locking methodologies. In conclusion, the research article presents a comprehensive framework that takes advantage of GANs to enhance the security of hardware IPs. The proposed pipeline integrates various stages, including data generation/RTL2Graph, model training, defense generation, security evaluation, optimization, Graph2RTL conversion, and RTL verification. Using GANs, the pipeline enables the automatic generation of logic locked designs, improving security and mitigating vulnerabilities. GANs can be used in IP security, specifically for the protection of ICs. Here, the idea behind using GANs for IP security is to train a GAN to learn the behavior of multiple locking schemes that have been proposed on the original IC. The trained generator is then used to produce a new locking scheme that has the potential of multiple existing locking schemes and to implement it on the original ICs. These ICs can then replace the original IC in the supply chain, making it difficult for attackers to reverse engineer the locked IP. This concept opens new avenues for further investigation and has the potential to significantly impact the field of IP security.

Additionally, another direction in the application of GAN for logic locking is to train a GAN on the behavior of the original IC, including its power consumption, timing characteristics, and output responses. The generator network then produces logic locked ICs with behavior similar to the original. Another approach is to use a GAN to generate functionally equivalent circuits that are different from the original but produce the same output for a given input. This can be useful when IPs need to be protected, but the original circuits cannot be modified. In these cases, the use of GANs for IP security is an active area of research. The challenges of these design approaches include ensuring that the generator produces circuits that are functionally equivalent, secure, and reliable.

Appendix

The GitHub repository will be regularly updated, ensuring the inclusion of the recent code, datasets, and relevant research materials. The project is actively managed with periodic updates to reflect ongoing work. The latest version is available at https://github.com/Sky025/GAN4IP.git.

Acknowledgements

Jugal Gandhi acknowledges the fellowship support provided by CSIR-Human Resource Development Group (HRDG).

References

[1] Hardware security modules market https://www.marketsandmarkets.com/Market-Reports/hardware-security-modules-market-162277475.html

[2] Roy J A, Koushanfar F and Markov I L 2008 EPIC: Ending piracy of integrated circuits In: *2008 Design, Automation and Test in Europe*, pp. 1069–1074

[3] Gandhi J, Shekhawat D, Santosh M and Pandey J G 2023 Logic locking for IP security: a comprehensive analysis on challenges, techniques, and trends. *Comput. Secur.* 129: 103196

[4] Alrahis L and Sinanoglu O 2023 Graph neural networks for hardware vulnerability analysis–can you trust your GNN? arXiv preprint arXiv:2303.16690

[5] Chakraborty P, Cruz J and Bhunia S 2018 SAIL: machine learning guided structural analysis attack on hardware obfuscation In: *2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pp. 56–61

[6] Alrahis L, Patnaik S, Shafique M and Sinanoglu O 2022 OMLA: an oracle-less machine learning-based attack on logic locking IEEE trans. *Circuits Syst. II: Expr. Briefs* 69: 1602–1606

[7] Alrahis L, Patnaik S, Shafique M and Sinanoglu O 2022 MuxLink: Circumventing learning-resilient mux-locking using graph neural network-based link prediction. In: *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 694–699

[8] Alrahis L, Patnaik S, Hanif M. A, Shafique M and Sinanoglu O 2021 UNTANGLE: Unlocking routing and logic obfuscation using graph neural networks-based link prediction. In: *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–9

[9] Mankali L, Alrahis L, Patnaik S, Knechtel J and Sinanoglu O 2022 Titan: security analysis of large-scale hardware obfuscation using graph neural networks IEEE trans. *Inf. Forensics Secur.* 18: 304–318

[10] Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A and Bengio Y 2020 Generative adversarial networks. *Commun. ACM* 63: 139–144

[11] Dutta I K, Ghosh B, Carlson A, Totaro M and Bayoumi M 2020 Generative adversarial networks in security: a survey. In: *2020 11th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, pp. 0399–0405

[12] Alrahis L, Knechtel J and Sinanoglu O 2023 Graph neural networks: a powerful and versatile tool for advancing design, reliability, and security of ICs. In: *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, pp. 83–90

[13] Tauhid A, Xu L, Rahman M and Tomai E 2023 A survey on security analysis of machine learning-oriented hardware and software intellectual property. *High-Conf. Comput.* 3: 100114

[14] Sisejkovic D, Reimann L M, Moussavi E, Merchant F and Leupers R 2021 Logic locking at the frontiers of machine learning: a survey on developments and opportunities. In: *2021 IFIP/IEEE 29th International Conference on Very Large Scale Integration (VLSI-SoC)*, pp. 1–6

[15] Gandhi J, Shekhawat D, Santosh M and Pandey J G 2023 LOKI: a secure FPGA Prototyping of IoT IP with lightweight logic locking. In: *2023 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*

[16] Gandhi J, Shekhawat D, Santosh M and Pandey J. G 2023 LightLock: ensuring hardware IP security in IoT environment with lightweight logic locking. In: *International Symposium on VLSI Design and Test*

[17] Pilato C, Chowdhury A. B, Sciuto D, Garg S and Karri R 2020 ASSURE: RTL locking against an untrusted foundry. *CoRR.* abs/2010.05344

[18] Kamali H M, Azar K Z, Farahmandi F and Tehranipoor M 2022 Advances in logic locking: past, present, and prospects. *Cryptol. ePrint Arch.*

[19] Yasaei R, Yu S-Y, Naeini E K and Al Faruque M A 2021 GNN4IP: graph neural network for hardware intellectual property piracy detection. In: *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 217–222

[20] Kahng A, Lach J, Mangione-Smith W, Mantik S, Markov I, Potkonjak M, Tucker P, Wang H and Wolfe G 1998 Watermarking techniques for intellectual property protection. In: *Proceedings 1998 Design and Automation Conference. 35th DAC. (Cat. No.98CH36175)*, pp. 776–781

[21] Karri R, Rajendran J, Rosenfeld K and Tehranipoor M 2010 Trustworthy hardware: identifying and classifying hardware trojans. *Computer* 43: 39–46

[22] Huang Z, Wang Q, Chen Y and Jiang X 2020 A survey on machine learning against hardware trojan attacks: recent advances and challenges. *IEEE Access* 8: 10796–10826

[23] Yasaei R, Chen L, Yu S-Y and Al Faruque M A 2022 Hardware Trojan Detection using Graph Neural Networks. *IEEE Trans. Comput.-Aided Design Integr, Circuits Syst.* 1–14

[24] Yasaei R, Yu S-Y and Al Faruque M A 2021 GNN4TJ: graph neural networks for hardware Trojan detection at register transfer level. In: *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1504–1509

[25] Sisejkovic D, Collini L, Tan B, Pilato C, Karri R and Leupers R 2022 Designing ML-resilient locking at register-transfer level. arXiv preprint arXiv:2203.05399

[26] Wang Z, Wu Y, Park Y, Yoo S, Wang X, Eshraghian J K and Lu W D 2023 PowerGAN: a machine learning approach for power side-channel attack on compute-in-memory accelerators. arXiv preprint arXiv:2304.11056

[27] Wang K, Zheng H, Li Y, Li J and Louri A 2022 AGAPE: Anomaly detection with generative adversarial network for improved performance, energy, and security in manycore systems. In: *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 849–854

[28] Chhabria V. A, Kunal K, Zabihi M and Sapatnekar S S 2021 BeGAN: power grid benchmark generation using a process-portable GAN-based Methodology. In: *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–8

[29] Zhou J, Cui G, Hu S, Zhang Z, Yang C, Liu Z, Wang L, Li C and Sun M 2020 Graph neural networks: a review of methods and applications. *AI Open* 1: 57–81

[30] Gao C, Zheng Y, Li N, Li Y, Qin Y, Piao J, Quan Y, Chang J, Jin D and He X *et al.* 2023 A survey of graph neural networks for recommender systems: challenges, methods, and directions. *ACM Trans. Recomm. Syst.* 1: 1–51

[31] Alrahis L, Patnaik S, Shafique M and Sinanoglu O 2022 Embracing graph neural networks for hardware security. In: *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pp. 1–9

[32] Chakraborty P, Cruz J and Bhunia S 2019 SURF: joint structural functional attack on logic locking. In: *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 181–190

[33] Sisejkovic D, Merchant F, Reimann L M, Srivastava H, Hallawa A and Leupers R 2020 Challenging the security of logic locking schemes in the era of deep learning: a neuroevolutionary approach. *CoRR* abs/2011.10389

[34] Rajendran J, Zhang H, Zhang C, Rose G S, Pino Y, Sinanoglu O and Karri R 2015 Fault analysis-based logic encryption. *IEEE Trans. Comput.* 64: 410–424

[35] Yu S. Y, Yasaei R, Zhou Q, Nguyen T and Al Faruque M A 2021 HW2VEC: A graph learning tool for automating hardware security In: *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 13–23

[36] Song Z, Yang X, Xu Z and King I 2022 Graph-based Semi-supervised learning: a comprehensive review. *IEEE Trans. Neural Netw. Learn. Syst.* 34: 8174–8194

[37] Roweis S T and Saul L K 2000 Nonlinear dimensionality reduction by locally linear embedding. *Science* 290: 2323–2326

[38] Belkin M and Niyogi P 2001 Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering. Adv. Neural Inf. Process. Syst. p 14

[39] Ou M, Cui P, Pei J, Zhang Z and Zhu W 2016 Asymmetric transitivity preserving graph embedding. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1105–1114

[40] Perozzi B, Al-Rfou R, and Skiena S 2014 Deepwalk: online learning of social representations. In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 701–710

[41] Yang Z, Cohen W and Salakhudinov R 2016 Revisiting Semi-supervised Learning with Graph Embeddings In: *International Conference on Machine Learning*, pp. 40–48

[42] Grover A and Leskovec J 2016 NODE2VEC: Scalable feature learning for networks. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 855–864

[43] Chen H, Perozzi B, Hu Y and Skiena S 2018 HARP: Hierarchical representation learning for networks. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 2127–2134

[44] Wang D, Cui P and Zhu W 2016 Structural deep network embedding In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1225–1234

[45] Kipf T N and Welling M 2016 Variational graph auto-encoders. arXiv preprint arXiv:1611.07308

[46] Pan S, Hu R, Fung S-f, Long G, Jiang J and Zhang C 2019 Learning graph embedding with adversarial training methods. *IEEE Trans. Cybern.* 50: 2475–2487

[47] Kipf T N and Welling M 2016 Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907

[48] Hamilton W. L, Ying R and Leskovec J 2017 Inductive representation learning on large graphs. *CoRR* abs/1706.02216

[49] Fan W, Liu C, Liu Y, Li J, Li H, Liu H, Tang J and Li Q 2023 Generative diffusion models on graphs: methods and applications. arXiv preprint arXiv:2302.02591

[50] Wang H, Wang J, Wang J, Zhao M, Zhang W, Zhang F, Xie X, and Guo M 2018 GraphGAN: graph representation learning with generative adversarial nets. In: *Proceedings of the AAAI Conference on Artificial Intelligence*

[51] Tang J, Qu M, Wang M, Zhang M, Yan J and Mei Q 2015 Line: large-scale information network embedding. In: *Proceedings of the 24th International Conference on World Wide Web*, pp. 1067–1077

[52] Subramanyan P, Ray S and Malik S 2015 Evaluating the security of logic encryption algorithms. In: *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 137–143

[53] Azar K Z, Kamali H M, Homayoun H and Sasan A 2018 SMT Attack: next generation attack on obfuscated circuits with capabilities and performance beyond the SAT Attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019: 97–122

[54] Alaql A, Rahman M M and Bhunia S 2021 SCOPE: synthesis-based constant propagation attack on logic locking. *IEEE Trans. VLSI Syst.* 29: 1529–1542

[55] Yang F, Tang M and Sinanoglu O 2019 Stripped functionality logic locking with hamming distance-based restore unit (SFLL-HD)—unlocked. *IEEE Trans. Inf. Forensics Secur.* 14: 2778–2786

[56] Xie Y and Srivastava A 2019 Anti-SAT: mitigating SAT attack on logic locking. *IEEE Trans. Comput. Aided Des. Integr Circuits Syst.* 38: 199–207

[57] Sisejkovic D, Merchant F, Reimann L. M and Leupers R 2021 Deceptive logic locking for hardware integrity protection against machine learning attacks. *CoRR* abs/2107.08695

[58] Alrahis L, Sengupta A, Knechtel J, Patnaik S, Saleh H, Mohammad B, Al-Qutayri M and Sinanoglu O 2021 GNN-RE: graph neural networks for reverse engineering of gate-level netlists. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 41: 2435–2448

[59] He Z, Wang Z, Bail C, Yang H and Yu B 2021 Graph learning-based arithmetic block identification. In: *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–8

[60] Chen Z, Kolhe G, Rafatirad S, Lu C.-T, PD S. M, Homayoun H and Zhao L 2020 Estimating the circuit De-obfuscation runtime based on graph deep learning. In: *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 358–363

[61] Chen T, Sun Q, Zhan C, Liu C, Yu H and Yu B 2021 Deep H-GCN: fast analog IC aging-induced degradation estimation. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 41: 1990–2003

[62] Alrahis L, Knechtel J, Klemme F, Amrouch H and Sinanoglu O 2022 GNN4REL: graph neural networks for predicting circuit reliability degradation. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 41: 3826–3837