Designing with Deception: ML- and Covert Gate-Enhanced Camouflaging to Thwart IC Reverse Engineering

Junling Fan, David Koblah, Domenic Forte
Department of Electrical and Computer Engineering
University of Florida, Gainesville, USA
{fan.j, dkoblah}@ufl.edu, dforte@ece.ufl.edu

***Abstract*—Integrated circuits (ICs) are essential to electronic systems, yet they face significant risks from physical reverse engineering (RE) attacks that compromise intellectual property (IP) and overall system security. While IC camouflaging has emerged to mitigate these risks, existing approaches largely focus on localized gate modifications, neglecting comprehensive deception strategies. To address this gap, we present a machine learning (ML)-driven methodology – IP Camouflage – that integrates cryptic and mimetic deception principles to enhance IC security against RE. Our approach leverages a novel And-Inverter Graph Variational Autoencoder (AIG-VAE) to encode circuit representations, enabling dual-layered camouflage through functional preservation and appearance mimicry. By introducing new variants of covert gates – Fake Inverters, Fake Buffers, and Universal Transmitters – our methodology achieves robust protection by obscuring circuit functionality while presenting misleading appearances. Experimental results demonstrate the effectiveness of our strategy in maintaining circuit functionality while achieving strong resistance to SAT-based attacks with low structural overhead. Additionally, we validate the robustness of our method against advanced artificial intelligence (AI)-based RE attacks.**

***Index Terms*—Reverse Engineering, IC Camouflage, Cyber Deception, Machine Learning, Hardware Security**

I. Introduction

As integrated circuits (ICs) continue to serve as the backbone of modern electronic systems, the threat of reverse engineering (RE) has become increasingly consequential. Physical RE enables adversaries to extract sensitive information from hardware designs, including intellectual property (IP), underlying algorithms, and security features. By physically probing, delayering and imaging, and analyzing the results, attackers can reconstruct an IC's gate-level representation, enabling replication, cloning, vulnerability analysis, and/or tampering [1]. This poses substantial risks, particularly in critical domains such as defense, healthcare, and finance, where IP theft or hardware tampering could have catastrophic consequences to national security, public health and economic security, respectively. With the continuous advancement of RE tools and techniques, safeguarding ICs from these threats has become increasingly challenging [2], [3].

IC camouflaging has emerged as a technique to protect hardware IPs from RE attacks. This approach leverages fabrication-based technologies to intentionally obscure the true functionality of the circuit, deliberately confusing adversaries [4], [5]. Camouflaged sections are designed to mislead attackers during the IC RE process, resulting in incorrect or incomplete reconstructions. While this may be the intention, current methods primarily focus on *localized* gate layouts or interconnections, neglecting a holistic camouflage of the overall circuit's functionality. This narrow focus limits their ability to achieve broader, system-level deception, which could significantly hinder adversarial efforts during RE in real-world applications.

Cyber deception offers a proactive and versatile defense strategy for addressing these limitations. The taxonomy proposed by Pawlick et al. [6] categorizes cyber deception into six distinct strategies: perturbation, moving target defense (MTD), obfuscation, mixing, honey-x, and attacker engagement. These are classified broadly into two categories: *cryptic* deception, which obscures the existence or nature of "assets", and *mimetic* deception, which simulates realistic but misleading appearances [6]. According to this definition, existing camouflage strategies against IC RE focus on cryptic deception while mimetic deception remains unexplored.

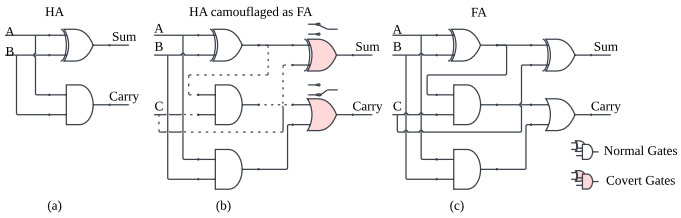

Figure 1: Illustration of mimetic deception for: (a) a Half Adder (HA), (b) an HA camouflaged deceptively to appear as a Full Adder (FA) using covert gates [7], and (c) a standard FA. The pink gates are covert gates that appear as the logic gate under SEM imaging but are actually buffers or inverters that only operate on the non-dashed input signal. Dashed lines are dummy inputs unbeknownst to attackers.

Our work addresses this through the following contributions:

- **Filling the Mimetic Deception Gap:** We introduce mimetic deception, complementing cryptic deception by simultaneously hiding the true functionality of circuits and presenting a misleading outward appearance. Figure 1 shows a toy example where a camouflaged Half Adder (HA) looks similar to a Full Adder (FA) in appearance. We also create new variants of covert gates [7] which would appear as inverters, buffers, etc. under SEM but may function differently.
- **Machine Learning-Based Implementation:** We employ an ML-based strategy using a variational autoencoder (VAE) model to achieve "dual-layered" camouflage with and-inverter graphs (AIGs); that is, it ensures efficient and systematic implementation of both cryptic and mimetic deception, overcoming the limitations of traditional approaches. Note that the proposed AIG-VAE is also a unique contribution which can be used in other EDA/CAD applications.
- **Demonstrated Robustness Against AI-Enhanced RE Tools:** Through experiments, we demonstrate how our strategy effectively thwarts advanced RE methods, including those leveraging AI, underscoring its practical security improvements.

The remainder of this paper is organized as follows: Section II offers background on threats against ICs and discusses how they may be impacted by the proposed camouflaging. In Section III, we provide background on the main sub-components of the proposed approach: covert gates, VAEs, and cyber deception. The section concludes with a few motivational examples, highlighting the vast potential of hardware security that arises from combining IC camouflaging with deception principles. The overall methodology (IP Camouflage) and main contributions are discussed in Section IV. Experimental results that demonstrate the effectiveness of the AIG-VAE and IC camouflage

are provided in Section V. Finally, Section VI concludes the paper and highlights directions for future research.

II. Attacks and Threat Model

The landscape of hardware security has rapidly evolved in the last few decades, driven by the changing nature of attacks and the corresponding development of countermeasures. The constant theme in this field is that there is no one-size-fits-all solution for all adversaries. However, our methodology improves IP security against a few distinct physical attacks.

A. Reverse Engineering

Reverse Engineering (RE) is a process that involves systematically analyzing electronic devices to extract design information at the chip, board, or system levels. It involves dissecting hardware designs to uncover vulnerabilities or proprietary information [8]. A typical process of IC RE includes:

1) Remove the chip packaging to expose the die for further analysis.
2) Sequentially remove individual chip layers for examination.
3) Capture high-resolution images of the exposed layers.
4) Annotate the captured images and extract the circuit's netlist using hardware assurance tools.

Adversaries can use RE for cloning, counterfeiting, and IP theft. Techniques like SEM imaging, dry or wet etching, and annotation tools are employed for accurate extraction and analysis of device designs [1], [2]. Given the complexity of modern hardware, the process requires significant resources. Although it has utility in security and assurance applications, such as hardware Trojan detection [9], [10], it is viewed as an attack in this paper.

B. SAT-based Attack

The SAT problem asks whether a given Boolean formula $F(x_1, x_2, x_3, \ldots, x_n)$ is satisfiable, i.e., whether there exists an assignment of variables $x_i \,\forall\, i$ that makes F evaluate to true. While SAT is NP-complete, modern SAT solvers are highly efficient and have been successfully applied in various domains such as equivalence checking, formal verification, and automatic test pattern generation (ATPG) [11], [12]. By leveraging the power of SAT solvers [13]–[15], circuit obfuscation, including logic locking and camouflaged circuits has been broken. Such attacks assume that a locked/camouflaged netlist is available along with a functional/unlocked chip (oracle). Then, the following iterative process occurs:

1) Identify a distinguishing input pattern (DIP) using the SAT solver. This is an input that differentiates between the outputs produced by two or more candidate keys[1].
2) Apply the DIP to oracle (unlocked chip) and observe the correct output response.
3) Rule out incorrect key assignments by adding constraints to the SAT solver that eliminate keys producing outputs inconsistent with the oracle's response.
4) Repeat the above steps until no further DIPs can be found. At this point, the correct key (de-camouflaged design) is uniquely identified.

The basis of the proposed approach is covert gates (see Section III-A) which SAT and VLSI-test based attacks do no scale well against [7]. Further, our ground-up approach that extends camouflaging to the IP level should also prevent the success of these attacks.

[1]For IC camouflaging, the netlist can be translated to one with keys where the correct key bits identify the correct function of the camouflaged cells.

C. Threat Model in This Work

This work addresses a threat model where an adversary, such as an IC reverse engineer, seeks to exploit ICs to compromise their security and functionality. The adversary's primary objectives are:

1) **Uncovering IC Functionality**: Using reverse engineering techniques to extract critical design information from security-focused IP, such as intrusion detection systems or encryption modules, to bypass or exploit their functionality. Further, the IP can also be stolen, replicated, and counterfeited.
2) **Tampering with Sensitive Signals**: Using the information from # 1 to target critical components or operations through physical attacks, such as laser fault injection (LFI) or focused ion beam (FIB), to disrupt normal functionality or manipulate sensitive signals.

We assume that the adversary has access to physical RE equipment, such as SEM imaging, but optical side channels [16] are counteracted by sensing techniques, e.g., [17]. Fault injection and non-invasive side channel analysis, which are underexplored in de-obfuscation, are considered out of scope and left for future work.

III. Related Works

A. IC Camouflaging and Covert Gates

IC camouflaging can generally be classified into two categories [7]: gate and interconnect camouflaging. Gate camouflaging focuses on replacing individual logic gates within the circuit with ones that could implement multiple functions depending on fabrication-related secrets (e.g., dummy contacts [5], threshold voltages [18], or semiconductor doping [19]). This approach effectively disrupts the ability of the attacker to identify the specific logic gates from SEM images. On the other hand, interconnect camouflaging [20] involves altering the connections between gates, making it difficult for an adversary to accurately map out the circuit's wiring or logic flow during IC RE.

The state-of-the-art (SotA) in IC camouflaging is covert gates[2], which effectively combine gate and interconnect variants through the use of always-on and always-off transistors [7]. The designer can create logic gates with different functions as well as circuits with dummy inputs (dummy connections) by altering transistor states from normal to always-on and always-off. Further, since the transistor states are not visible to attackers under SEM, the covert gates have the same appearance as standard CMOS gates, allowing them to achieve higher scalability against SAT and VLSI test-based attacks compared to other camouflaged gates. Given these advantages, we adopt the covert gate methodology as part of our camouflaging strategy.

B. Variational Autoencoder (VAE)

A variational autoencoder (VAE) [21] is a generative model that encodes data into a latent space and reconstructs it, introducing a probabilistic framework for generating new samples. In a VAE, the encoder maps the input x to a latent variable z by learning a probability distribution $z \sim q(z|x) = \mathcal{N}(\mu(x), \sigma(x)^2)$.

The decoder reconstructs x from z by generating $p(x|z)\hat{x} \sim p(x|z)$. To ensure meaningful interpolation, the latent space is regularized with a Gaussian prior $p(z) = \mathcal{N}(0, I)$, where I refers to the identity matrix. The VAE's objective combines two loss terms: *reconstruction loss*, which measures how well x is reconstructed from z:$\mathcal{L}_{\text{recon}} = -\mathbb{E}_{q(z|x)}[\log p(x|z)]$, and *KL divergence loss*, which aligns $q(z|x)$ with the prior $p(z)$: $\mathcal{L}_{\text{KL}} = D_{\text{KL}}(q(z|x) \parallel p(z))$,where

[2]To our knowledge, circuits implemented with covert gates have never been successfully broken in the literature. Thus we consider it to be the SotA.

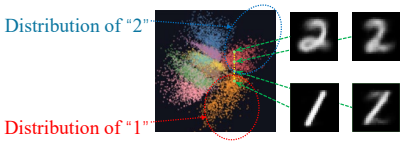

Figure 2: Latent space distribution of the MNIST dataset [22]. Using [23], a visualization shows how interpolation in the latent space can result in samples that mix features from different inputs.

$\|$ denotes the divergence between two probability distributions. The total loss function for the VAE is

$$\mathcal{L}_{\text{VAE}} = \mathcal{L}_{\text{recon}} + \mathcal{L}_{\text{KL}}. \tag{1}$$

Machine learning models based on VAE enable interpolation between two latent codes from 2 different samples, allowing the creation of a new sample with a balanced feature between two samples as Figure 2 shows. VAEs and such interpolation are the basis of our camouflaging approach where the two interpolated samples are the circuit's desired function and its desired appearance.

C. Cyber Deception: Cryptic and Mimetic Strategies

Cyber deception represents a proactive and innovative defense mechanism designed to counter cyber threats by misleading attackers, complicating their operations, and enhancing overall system security. Unlike traditional defenses that primarily focus on detection and prevention, cyber deception actively manipulates the attack surface to confuse, disorient, and frustrate adversaries. This strategic approach is rooted in the historical principle of deception used in warfare and espionage but has been adapted to address the complex challenges of cybersecurity in the digital era [6], [24].

Cryptic Deception. Cryptic deception aims to obscure the existence or true nature of assets, thereby *hindering attackers from identifying their targets*. By hiding critical information, cryptic methods effectively render the attack surface ambiguous, complicating an adversary's reconnaissance efforts and reducing the likelihood of a successful breach. A prominent example of cryptic deception is the use of moving target defenses (MTDs), which dynamically alter system configurations to create uncertainty and unpredictability. This approach forces attackers to expend significant time and resources without guaranteeing success. MTD strategies have been effectively applied to hardware security, as demonstrated in [17], [25], [26], where hardware configurations are reconfigured at run time to protect designs and/or on-chip assets from hardware Trojans, impedance-related side-channel attacks, and optical probing.

Mimetic Deception. Mimetic deception, on the other hand, focuses on imitation to present misleading but realistic appearances. This approach seeks to mislead attackers by simulating legitimate system behaviors or mimicking the characteristics of valuable targets. Mimetic techniques include the deployment of honeypots and decoys, which imitate operational systems to *lure attackers away from critical assets*. These methods also serve as intelligence-gathering tools, enabling defenders to study attackers' tactics, techniques, and procedures (TTPs) in real time [6], [24].

Relevance to Anti-RE and Hardware Security. Our proposed methodology aligns with the principles of both cryptic and mimetic deception. By leveraging techniques such as functional preservation and appearance mimicry, we achieve a dual-layered camouflaging strategy. The functional integrity of an IP is preserved, ensuring operational reliability, while the appearance is camouflaged to resemble alternate IPs. This hybrid approach effectively misleads RE adversaries, aligning with cryptic principles by concealing the true logic of circuits and mimetic principles by presenting a deceptive outward appearance. This strategy also highlights the evolving role of cyber deception in modern security paradigms. By combining cryptic and mimetic deception, our work contributes to the broader landscape, demonstrating applicability to IP anti-theft as well as hardware security. Examples of promising new capabilities enabled by this approach include:

1) **Misdirection to Hide Real On-chip Assets.** Consider an intrusion detection and prevention IP that monitors network traffic and detects potential security threats. Through IC RE, an attacker can gain an understanding of how this IP works, exploiting the information to evade detection or inject false alarms in subsequent attacks on products that use it. Our dual-layered approach could make this security-focused IP look like something innocuous such as a USB or memory controller, making it seem less valuable as an attack target.
2) **Creation of On-chip Decoys to Attract Physical Attacks.** Consider an IP that detects LFI. Using IC RE, an attacker could identify the locations of this IP within the chip and thus places to avoid aiming the laser. Our dual-layered approach could make this security-focused IP appear to be a target of interest such as a cryptographic core. When an attacker tries to attack the decoy core, it would instead trigger data zeroization or chip self-destruction.

The deception concept combined with anti-RE is unique to this paper and opens up many promising directions in hardware security.

IV. Proposed Methodology: IP Camouflage

A. Overview

Existing camouflaging approaches are localized, primarily residing at the gate-level where a gate's function can be one of several functions that the attacker needs to determine. Here, we propose *IP Camouflage* to create a camouflaged IP that preserves an IP's function while mimicking the appearance of a different IP, misleading RE adversaries as described above. We refer to the IP with the desired functionality as the "functional circuit". The IP whose appearance we want the product to resemble is called the "appearance circuit". The process begins by converting both circuits into And-Inverter Graphs (AIGs), which are then encoded into latent representations using the encoder of our Variational Autoencoder (VAE)-based model, the *AIG-VAE* (described in Section IV-B).

The encoder generates probabilistic distributions for each circuit, represented by a mean (μ) and standard deviation (σ) that follow a normal distribution. From these distributions, two latent space codes are sampled: $\mathbb{Z}_F$, representing the functional circuit, and $\mathbb{Z}_A$, representing the appearance circuit. These codes are interpolated using a proportion factor p, resulting in an intermediate latent code $\mathbb{Z}_{G'}$, which combines the characteristics of the functional circuit and the appearance circuit: $\mathbb{Z}_{G'} = (1-p)\mathbb{Z}_F + p\mathbb{Z}_A$.

The decoder then reconstructs the circuit G' from $\mathbb{Z}_{G'}$, followed by a threshold-based filter to refine the output, ensuring clear and interpretable connections:

$$filter(x, Th) = \begin{cases} 0, & x \leq Th \\ 1, & x > Th \end{cases} \tag{2}$$

The generated circuit $\hat{G}$ is further refined with a *Functional Preservation* to align its function with F, producing $\hat{G}_F$. After this, *Appearance Mimicking* is applied to ensure the produced circuit resembles A. The final result, denoted as $\hat{G}_{FA}$, achieves both functional alignment with the functional circuit and visual similarity to the appearance circuit. The process is shown in Figure 3.

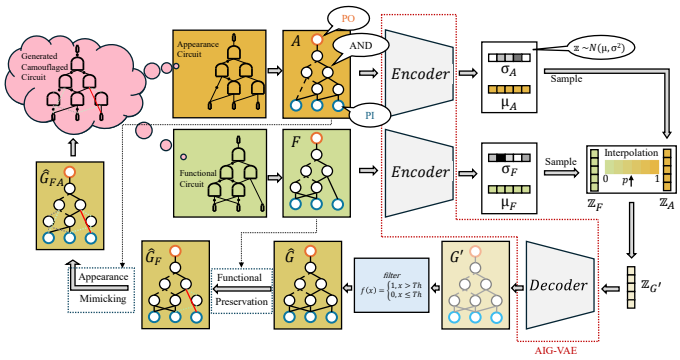

Figure 3: High-level Diagram of IP Camouflage. A functional circuit F and appearance circuit A are taken as inputs. After AIG-VAE and post-processing fixes, the final output is a camouflaged circuit $\hat{G}_{FA}$ in AIG format that matches F's function but appears as A.

Novelty. Unlike existing approaches that focus solely on localized camouflage, IP Camouflage leverages a machine learning model (AIG-VAE) to achieve holistic, system-level camouflage and deception. Note, however, that camouflaged gates are still utilized and present in our circuits. To be more specific, by integrating newly designed covert gates – Fake Inverters (FI), Fake Buffers (FB), and Universal Transmitters (UT), introduced in Section IV-D1 – during Functional Preservation and Appearance Mimicking, we enhance the robustness of the camouflage against adversaries. The subsequent sections provide detailed explanations of these processes.

B. Machine Learning Model: AIG-VAE

To effectively camouflage a circuit, our model must first understand the circuit's structure and function. Previous works, such as FGNN [27] and ABGNN [28], excel at learning functional representations of circuits by using asynchronous message-passing mechanisms, allowing the model to interpret the intricate relationships within circuit netlists. However, these models lack the generative capabilities needed to produce new circuits. This is where models like VGAE and D-VAE [29], [30] are important. These VAE-based models enable graph generation, allowing circuits to be generated from informative embeddings derived from circuit representation learning.

Building on this foundation, we propose our machine learning model, the And-Inverter Graph Variational Autoencoder (AIG-VAE), specifically designed for encoding and decoding circuits represented as AIGs. As shown in Figure 4, similar to Section III-B, the model consists of two core components: Encoder and Decoder.

1) Encoder

The encoder processes an input AIG X and maps it to a latent representation $\mathbb{Z}_X = E(X)$, capturing both structural and functional features of the circuit. Our encoder employs an asynchronous message-passing mechanism across the AIG, allowing messages to propagate throughout the graph while preserving circuit hierarchy and dependencies. This model is designed for AIGs with a single primary output (PO), multiple primary inputs (PIs), and various AND gates. To initialize the process, each PI is encoded manually using one-hot encoding. The message-passing follows a breadth-first search (BFS) order, meaning each node does not receive or update its message until all its precedent nodes have been traversed. The message-passing and update process for each node v can be formalized as follows:

- **Message Aggregation**: For each AND node, we aggregate messages from its two preceding nodes u_1 and u_2. If an edge between a node and a predecessor includes an inverter, the incoming embedding from that predecessor is negated. The aggregated message m_v for node v is given by: $m_v = \sum_{i=1}^{2} \text{sign}(e_i) \cdot h_{u_i}$, where h_{u_i} represents the embedding of the i-th preceding node, and $\text{sign}(e_i)$ is -1 if there is an inverter on the edge, $+1$ otherwise.

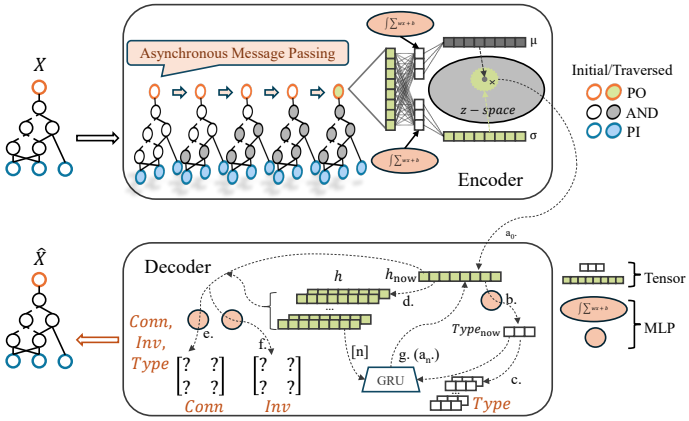

Figure 4: AIG-VAE Overview. The encoder uses asynchronous message passing to encode the input graph X into a latent representation in z-space. The decoder reconstructs the graph $\hat{X}$ as follows: (a_0) Sample a latent vector and initialize the first hidden state via an MLP; (b) Predict the node type (PI, PO, or AND) and assign PI to the first node; (c) Add the node type to the global tensor $Type$; (d) Add the hidden state to the global tensor h; (e) Predict connection edges with an MLP and store them in $Conn$; (f) Predict inverters with an MLP and store them in Inv (g, a_n); Update the hidden state for the next node using a GRU and repeat until all nodes are processed. The final AIG is reconstructed from $Conn$, Inv, and $Type$.

- **Node Update with GRU**: The aggregated message m_v, combined with the node type t_v (indicating whether it's a PI, PO, or AND), is fed into a Gated Recurrent Unit (GRU) [31] to update the node embedding. The GRU is a recurrent network cell designed to capture sequential and structural dependencies through gating mechanisms. Specifically, the GRU computes the next node embedding h_v in the following steps:

1) **Update Gate** (z_v): Determines how much of the previous embedding to retain in the updated embedding: $z_v = \text{sigmoid}(W_z m_v + U_z t_v + b_z)$.
2) **Reset Gate** (r_v): Controls how much of the previous embedding to forget: $r_v = \text{sigmoid}(W_r m_v + U_r t_v + b_r)$.
3) **Candidate Embedding** $(\tilde{h}_v)$: Computes the potential updated embedding: $\tilde{h}_v = \tanh(W_h(r_v \odot h_{v-1}) + U_h t_v + b_h)$.
4) **Final Node Embedding** (h_v): Combines the previous embedding h_{v-1} and the candidate embedding $\tilde{h}_v$, weighted by the update gate z_v: $h_v = (1 - z_v) \odot h_{v-1} + z_v \odot \tilde{h}_v$.

The notation used includes the sigmoid activation function, defined as $\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$, and $\tanh$, the hyperbolic tangent activation function defined as $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. The symbol $\odot$ represents element-wise multiplication. The weight matrices W_z, W_r, W_h are for the aggregated message m_v, while U_z, U_r, U_h represent the weight matrices for the node type t_v. b_z, b_r, b_h denote bias terms. This allows the model to differentiate between node types while capturing both structural and functional dependencies of the graph. The node embedding h_v is computed as: $h_v = \text{GRU}_e(m_v, t_v)$.

The embedding obtained from the final GRU update at the PO node is processed further to obtain the latent space representation. Specifically, this embedding, denoted as h_{PO}, is passed through two separate Multi-Layer Perceptrons (MLPs) to produce the mean μ and standard

deviation σ for the VAE's latent distribution: $\mu = \text{MLP}_{\mu}(h_{\text{PO}})$, $\sigma = \text{MLP}_{\sigma}(h_{\text{PO}})$. These parameters μ and σ define the Gaussian distribution $q(z|X) = \mathcal{N}(\mu, \sigma^2)$ from which the latent variable z is sampled. This approach enables the encoder to capture a probabilistic latent representation.

2) Decoder

The decoder reconstructs an AIG from the latent code $\mathbb{Z}_X$, generating an output $\hat{X} = D(\mathbb{Z}_X)$ that preserves the essential characteristics of the original circuit while allowing for controlled variation[3]. A sample is drawn from the latent space using the mean μ and standard deviation σ derived during encoding. During training, σ is used to augment data by adding stochasticity. However, during evaluation, only the mean μ is used for consistency: $z \sim \mathcal{N}(\mu, \sigma^2)$

To initialize the node embeddings, z is passed through a fully connected linear layer followed by a tanh activation to produce the first node embedding h_1: $h_1 = \tanh(\text{Linear}(\mathbb{Z}_X))$. The decoding process then unfolds as follows:

- **Node Type Prediction**: Using a MLP called `addNode`, the type of the node is determined from h. This MLP classifies the node as either a Primary Input (PI), Primary Output (PO), or an AND gate: $Type_i = \text{MLP}_{\text{addNode}}(h_i)$
- **Connection and Inverter Determination**: For each node, two additional MLPs predict the connections to all previous nodes and whether there are inverters on these connections: $Connection_{i,j} = \text{MLP}_{\text{connect}}(h_i, h_j)$ and $Inverter_{i,j} = \text{MLP}_{\text{inv}}(h_i, h_j)$, where h_i (h_j) represent the embeddings of current (previous) nodes.
- **Next Node Embedding**: To determine the embedding of the next node, the decoder uses a GRU specifically for decoding, denoted as GRU_d. The GRU takes the current embedding h and the node type as inputs, producing the embedding for the next node: $h_{\text{i+1}} = \text{GRU}_d(h_i, Type_i)$.

This process continues iteratively, building the entire AIG by adding nodes and determining connections, until the number of nodes matches that of the input circuit X from which the latent space code encoded from.

3) Loss Function

When calculating the loss function, the AIG structure is represented by three primary tensors that capture essential circuit information: $Type$ (which defines each node's type, such as PI, PO, or AND gate), $Connection$ (an adjacency matrix indicating the presence of connections between nodes), and $Inverter$ (indicating if an inverter is present on each edge). These tensors are calculated for the original circuit X and the reconstructed circuit $\hat{X}$, denoted as $Type_X$ and $Type_{\hat{X}}$, $Connection_X$ and $Connection_{\hat{X}}$, and $Inverter_X$ and $Inverter_{\hat{X}}$, respectively.

For each tensor, we use the mean squared error (MSE) to compute the discrepancy between the original and reconstructed values. Here, N denotes the total number of nodes in the graph: $\mathcal{L}_{\text{Type}} = \frac{1}{N \cdot 3}\Sigma_{i=1}^{N}\Sigma_{j=1}^{3}(Type_{X,i,j} - Type_{\hat{X},i,j})^2$, $\mathcal{L}_{\text{Connection}} = \frac{1}{N^2}\Sigma_{i=1}^{N}\Sigma_{j=1}^{N}(Connection_{X,i,j} - Connection_{\hat{X},i,j})^2$, $\mathcal{L}_{\text{Inverter}} = \frac{1}{N^2}\Sigma_{i=1}^{N}\Sigma_{j=1}^{N}(Inverter_{X,i,j} - Inverter_{\hat{X},i,j})^2$.

The KL divergence loss $\mathcal{L}_{KL}$ regularizes the latent space by measuring the divergence between the encoded distribution $q(z|X) = \mathcal{N}(\mu, \sigma^2)$ and a prior Gaussian distribution $p(z) = \mathcal{N}(0, I)$. It is computed as: $\mathcal{L}_{KL} = D_{KL}(q(z|X) \parallel p(z)) = \frac{1}{2}\sum_{i=1}^{d}\left(\sigma_i^2 + \mu_i^2 - \log(\sigma_i^2) - 1\right)$, where d is the dimensionality of the latent space, and μ and σ are the mean and standard deviation vectors from the encoder.

[3]Controlled variation refers to the ability to introduce slight variations during decoding by sampling nearby latent codes within the distribution in latent space, enabling exploration of similar but not identical representations.

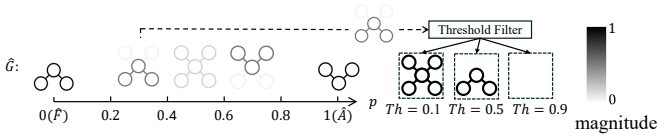

Figure 5: Interpolation between $\hat{F}$ and $\hat{A}$ and impact of threshold values on binary decision-making. The intensity of the color indicates the magnitude.

The overall loss function $\mathcal{L}$ is then defined as a weighted sum of these individual components, along with the KL divergence loss ($\mathcal{L}_{\text{KL}}$) to regularize the latent space: $\mathcal{L} = \alpha \cdot \mathcal{L}_{\text{Type}} + \beta \cdot \mathcal{L}_{\text{Connection}} + \gamma \cdot \mathcal{L}_{\text{Inverter}} + \delta \cdot \mathcal{L}_{\text{KL}}$. Here, α, β, γ, and δ are hyperparameters that control the relative contribution of each loss component to the overall objective. This loss function ensures that the reconstructed circuit retains the original structure and functionality while adhering to the latent space's constraints. For our implementation, we set $\alpha = 0.3$, $\beta = 0.3$, $\gamma = 0.3$, and $\delta = 0.1$ to balance the contributions of each component in the loss function.

C. Interpolation between Function and Appearance

In Section IV-A, we briefly introduced the concept of interpolating between the latent space representations of a functional circuit and the appearance circuit, enabling us to examine the intermediate states between the two. In this section, we will discuss this in greater detail.

The interpolation process blends the functional circuit F and appearance circuit A by leveraging their respective latent space representations, encoded as $\mathbb{Z}_F$ and $\mathbb{Z}_A$. These latent representations are interpolated based on a proportion factor p, defined : $\mathbb{Z}_G = (1-p)\mathbb{Z}_F + p\mathbb{Z}_A$, where $\mathbb{Z}_G$ balances between the latent representations of the functional and appearance circuits. The decoder then reconstructs a circuit $\hat{G}$ from $\mathbb{Z}_G$. Here, $p \in [0, 1]$ determines the interpolation factor. When $p = 0$, $\mathbb{Z}_G$ focuses entirely on the characteristics of the functional circuit, retaining the function and appearance of F. When $p = 1$, $\mathbb{Z}_G$ is dominated by the characteristics of the appearance circuit. For intermediate values of p, the generated circuit shows a blend of function and appearance of functional and appearance circuits, as illustrated in Figure 5.

Once the interpolated latent vector $\mathbb{Z}_G$ is obtained, it is decoded using the decoder to produce the new circuit $\hat{G} = D(Z_G)$. Due to the interpolation process, the rebuilt $\hat{G}$ varies as different proportion values are selected. In Figure 5, the lighter color in the figure indicates that the blended values decrease in magnitude. These decreased values do not directly indicate clear node connections or inverters. Therefore, a threshold value Th is applied to convert these values into binary decisions using a threshold filter, as shown in Equation 2. This filtering process ensures that the generated circuit has clear, interpretable connections and maintains structural integrity, even when blending between functional circuit and appearance circuit.

The final circuit, after applying thresholding, achieves the desired balance between functionality and appearance. By adjusting the interpolation factor p and the threshold Th, designers can fine-tune the generated circuit to better meet specific requirements.

D. Functional Preserve and Appearance Mimicking

The generated AIG $\hat{G}$, derived from the latent space distribution between the functional circuit F and the appearance circuit A, exhibits differences in both functionality and appearance compared to F and A. To reconcile these differences, we employ two fixes: *Functional Preserve*, which restores the functional behavior of F in

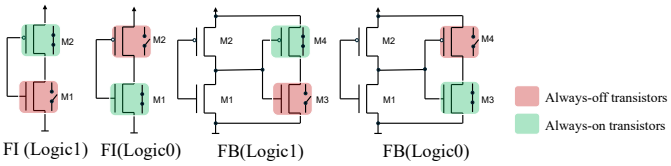

Figure 6: CMOS circuits of Fake Inverter (FI) and Fake Buffer (FB).

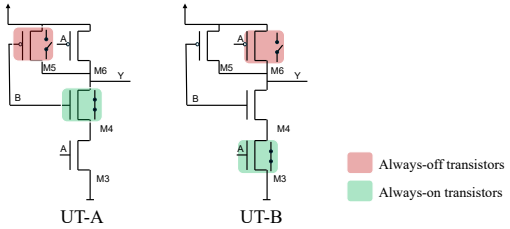

Figure 7: CMOS circuits of Universal Transmitter (UT). From left to right: UT-A, UT-B.

$\hat{G}$, and *Appearance Mimicking*, which further adjusts $\hat{G}$ to closely resemble A.

1) New Covert Gates

To implement these fixes, we introduce specialized components designed using the covert gate methodology. These camouflaged components leverage a unique manufacturing technology that enables transistors to remain in an always-on or always-off state [7]. This capability allows the creation of gates that deceive reverse engineers by emitting misleading SEM images while preserving the circuit's true functionality.

We have developed three types of components based on covert gates: the **Fake Inverter (FI)**, **Fake Buffer (FB)**, and **Universal Transmitter (UT)**. These components would be strategically incorporated during the Functional Preserve and Appearance Mimicking stages to obscure the circuit's true logic while presenting a deceptive external appearance. As shown in Figure 6, fake components, the FI and FB, are functionally distinct from the corresponding real component, but they maintain the same external appearance. The Universal Transmitter (UT), depicted in Figure 7, has two variations (UT-A and UT-B), having contrasting functions but the same appearance.

As shown in Table I, the various types of Fake Inverters (FI), Fake Buffers (FB), and Universal Transmitters (UT) provide distinct functionalities while maintaining consistent external appearances. These components are specifically designed to camouflage individual signals, effectively concealing their true functions from their outward appearance. For FI and FB, a signal can be made to appear inverted or non-inverted while functionally being logic1/logic0. For UT, the two inputs can be selected like in a MUX, while maintaining the appearance of a NAND gate. When used in the fix step, UT can choose between a signal and its negation, effectively functioning as a buffer or inverter – both sharing the same appearance. Like FI and FB, UT can also be configured to output constant logic1/logic0.

2) Functional Preservation and Appearance Mimicking

Building on the introduction of these covert gates – Fake Inverters (FI), Fake Buffers (FB), and Universal Transmitters (UT) – this section explains their integration into the processes of Functional Preservation and Appearance Mimicking.

The Functional Preservation process ensures that the generated circuit $\hat{G}$ functions identically to the original functional circuit F, while effectively camouflaging its true purpose. This is achieved by comparing the structure and behavior of $\hat{G}$ to F and making the necessary corrections. The process begins by equalizing the size of $\hat{G}$ and F by adding dummy nodes, ensuring that both circuits have

Table I: Function and Appearance of FI, FB, and UT

Component	Function	Appearance
FI	logic1 / logic0	1 Inverter
FB	logic1 / logic0	2 Inverters
UT-A	Buffer / logic 1 / logic 0	1 NAND
UT-B	Inverter / logic 1/ logic 0	1 NAND

Table II: Fix Steps for Functional Preservation and Appearance Mimicking

Functionality Preservation			Appearance Mimicking		
$\hat{G}$	Reference F	Fix Step	$\hat{G}_F$	Reference A	Fix Step
00/01	00/01	N/A	00/01	00/01	N/A
00/01	10	Connect	00/01	10	FB
00/01	11	Insert INV	00/01	11	FI
10	00/01	FB	10	00/01	N/A
10	10	N/A	10	10	N/A
10	11	UT-B	10	11	UT-A
11	00/01	FI	11	00/01	N/A
11	10	UT-A	11	10	UT-B
11	11	N/A	11	11	N/A

the same number of gates and nodes across all three types of nodes (PI, PO and AND). Subsequently, the connections and logic of each node in $\hat{G}$ are compared with those in F, and any discrepancies in logic or connections are identified and corrected. To address these discrepancies, Fake Inverter (FI), Fake Buffer (FB), and Universal Transmitter (UT) components are applied. These components enable the adjustments required to make $\hat{G}$ functionally equivalent to F while maintaining its camouflaged appearance.

With each corrective step applied to a signal in $\hat{G}$, Table II compares the generated circuit $\hat{G}$ with the functional circuit F, detailing the fix operations and their effects. The values $00/01$, 10, and 11 represent the states of connections and inversions: the first bit indicates the presence of a connection (0 for none, 1 for a connection), and the second bit indicates inversion (0 for no inversion, 1 for inversion). The "Fix Step" column outlines the actions taken, such as adding connections or applying FI, FB, or UT components. "Connect" and "Insert INV" means we have to restore the function with no covert gate applicable in this situation. "N/A" indicates that no fix is necessary.

Following the Functional Preservation is the Appearance Mimicking. $\hat{G}_F$, the output of Functional Preservation, and A are padded to the same size. Their edges are then compared, and the necessary fix steps are applied, treating A as the desired appearance and $\hat{G}_F$ as the desired function. The final circuit $\hat{G}_{F,A}$ is produced after Appearance Mimicking, which retains the functionality of the functional circuit while presenting an appearance similar to the appearance circuit.

V. Experimental Results

This section provides a comprehensive evaluation of IP Camouflage. Note that all the circuits produced by IP Camouflage successfully pass formal verification with the original circuit design. The experiments address three objectives:

1) **Validate Functional Understanding of the Model**: In Section V-A, we evaluate how well the AIG-VAE model captures the functional characteristics of circuits by correlating latent space distances with structural differences.

2) **SAT-Attack Resilience Analysis**: In Section V-B, we analyze the effectiveness of IP camouflage applying covert gate components against SAT-based attacks, which are widely acknowledged as a powerful and practical threat model in hardware security.

3) **Demonstrated Robustness Against AI-Enhanced Reverse Engineering Tools**: In Section V-C, we evaluate the robustness of our

Table III: Dataset Summary for Training and Testing

Dataset Subset	Number of Graphs	Total Number of Nodes
Training Dataset	1,482	63,091
Testing Dataset	371	14,515
Total	1,853	77,606

camouflaged circuits against advanced RE techniques by applying Graph Neural Networks (GNNs) [32] to classify circuit nodes.

A. AIG-VAE Model Training and Evaluation

Our dataset comprises combinational circuits from the ISCAS85 [33] and EPFL [34] benchmarks, converted into AIGs for uniform input representation suitable for the AIG-VAE model. Note that the former benchmarks were originally utilized for an RE case study. Output signals and their preceding signals are decomposed into tree graphs, which serve as model inputs. To manage complexity, trees exceeding a maximum node limit are excluded. The dataset is split into training and testing subsets (80-20 split). Table III summarizes the dataset with details on the number of graphs and nodes. Graph processing is facilitated by the Deep Graph Library (DGL) [35], which integrates seamlessly with PyTorch and handles graphs as directed acyclic graphs (DAGs).

The AIG-VAE model, implemented in PyTorch (version 2.3.1) [36], was trained on an NVIDIA A100 GPU with 80 GB memory. Using the Adam optimizer with a learning rate of 0.0001 and batch size of 1, training aimed to minimize a combined loss function (reconstruction loss, structural consistency, and latent space regularization; see Section IV-B3). The model was trained for 100 epochs with early stopping based on validation loss. Key training hyperparameters include a latent space dimension of 512, a KL divergence weight (δ) of 0.1, and reconstruction loss weights ($\alpha/\beta/\gamma$) of 0.3 each.

We evaluate the AIG-VAE model's ability to encode and capture the structural and functional characteristics of circuits by analyzing the relationship between *graph edit distance (GED)* and *latent space distance (LSD)* for all pairwise combinations of graphs in the testing dataset. GED, computed using the `Networkx` library [37], measures structural similarity as the minimum number of edit operations (node insertion, deletion, or substitution) required to transform one graph into another. LSD is the Euclidean distance between the latent representations of two graphs produced by the AIG-VAE encoder.

Given the computational cost of GED, a timeout was applied, and data points exceeding the limit were discarded. Following this, we gathered 136,445 valid data points. To better interpret the relationship between GED and LSD, we use a *binned analysis*, dividing LSDs into 20 bins and computing the mean GED and standard deviation for each bin. The results, shown in Figure 8, reveal a strong positive correlation: larger LSDs correspond to greater GED values, as indicated by the trend and a Pearson correlation coefficient of $r = 0.98$, with some errors. *This analysis demonstrates that the AIG-VAE effectively encodes features into the latent space, capturing both structural differences and functional characteristics.* These latent representations provide a meaningful and continuous basis for downstream tasks.

B. Evaluating Resistance to SAT-Guided Reverse Engineering

SAT-based attacks pose a major threat to logic locking (LL) and camouflaging by exploiting oracle access and iterative SAT solving to recover circuit functionality. To evaluate SAT resilience, we model covert gates – Fake Inverter (FI), Fake Buffer (FB), and Universal Transmitter (UT) – as key-programmable elements, each controlled by two keys to select logic-1, logic-0, or normal behavior. Gates

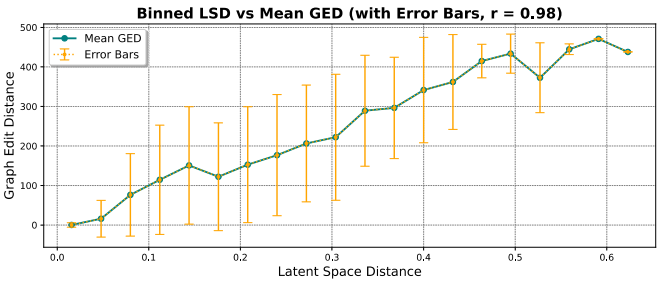

Figure 8: Binned plot of latent space distance (LSD) vs. mean graph edit distance (GED) for all pairwise combinations of graphs in the testing dataset.

Table IV: Circuit Pairs Used for SAT-Guided Reverse Engineering Evaluation

Pair	Circuit A	#$Nodes$	Circuit F	#$Nodes$
0	c5315_N7705	132	c432_N421	166
1	c5315_N7504	144	i2c_po061	53
2	banyan_8_out_5	125	c3540_N4589	122
3	c2670_N3809	154	memctrl_po0198	149

with the same appearance as covert gates are also treated as potential targets, greatly expanding the key space.

We tested four circuit pairs (Table IV), each combining a functional and an appearance circuit. For each, we generated multiple IP Camouflage configurations with varying thresholds and compared them to logic-locked versions with matched area overhead. As shown in Table V, IP Camouflage *consistently produces significantly more keys under similar area, power, and delay overheads* – enhancing resistance due to the exponential nature of SAT solving.

To assess solver run time, we ran SAT attacks on configurations with maximum and minimum key counts per pair. As reported in Table V, IP Camouflage circuits often caused timeouts or memory exhaustion on a 100 GB RAM, AMD EPYC 7702 64-core server using the Glucose SAT solver, while LL equivalents were typically solved within milliseconds. These results confirm the practical strength of IP Camouflage against SAT-guided RE.

C. GNN-RE Analysis for Post-Camouflage Node Classification

To evaluate the effectiveness of the proposed IP Camouflage methodology, we employed GNN-RE [32], which uses state-of-the-art node classification based on GraphSAINT [38]. The dataset includes eight benchmark circuits from ISCAS85 [33] and EPFL [34], with a fixed training set across all experiments as shown in Table VI. For each group, the test set includes camouflaged signals (functional) and their deceptive counterparts (appearance). The proportional parameter p was varied across $\{0.1, 0.3, 0.5, 0.7, 0.9\}$, and three F1-scores were computed:

- $F1_{\text{Valid}}$: Real-label classification accuracy on the validation set.
- $F1_{\text{Cryptic}}$: Test classification accuracy for functional circuit label.
- $F1_{\text{Mimetic}}$: Test classification accuracy for appearance circuit label.

Cryptic camouflage is achieved when $F1_{\text{Cryptic}} < F1_{\text{Valid}}$, indicating reduced recognition of functional nodes. Mimetic camouflage is considered successful when $F1_{\text{Mimetic}}$ significantly exceeds random guess and is high enough to be distinguishable. To further highlight the improvement, we compare our method to the random covert gate insertion strategy proposed in [7], using two baselines: (1) random insertion with area overhead matched to ours and (2) 5% gate-level random insertion originally used in [7].

The results in Table VII demonstrate that our proposed IP Camouflage *achieves both significantly lower* $F1_{Cryptic}$ *and higher* $F1_{Mimetic}$

Table V: IP Camouflage vs. Logic Locking for number of keys and overhead across Pairs 0–3. **Note:** Only edge-case configurations were considered. **P**: Pair index. **Th**: Threshold parameter for IP Camouflage. **A**, **Pwr**, **Dly**:Area, Power, and Delay Overhead from IP Camouflage. **#K**: Key count from IP Camouflage. **#K$_{eq}$ / A**: Number of keys required for logic locking to match the area overhead. **T$_{Ours}$**: SAT Attack Time of IP Camouflage. **T$_{LL}$**: SAT Attack Time of Logic Locking. **TO**: Time out for 24-hour limit. **OoM**: Out of memory for 100 GB limit.

P	Th	A	Pwr	Dly	#K	#K$_{eq}$ / AO	T$_{Ours}$	T$_{LL}$	P	Th	A	Pwr	Dly	#K	#K$_{eq}$ / AO	T$_{Ours}$	T$_{LL}$
0	0.01	1.40×	1.37×	1.00×	332	68 / 1.41×	TO	670 ms	2	0.01	1.34×	1.33×	1.00×	244	44 / 1.34×	TO	89 ms
0	0.02	1.38×	1.34×	1.00×	318	64 / 1.39×	-	-	2	0.02	1.32×	1.31×	1.00×	238	44 / 1.34×	-	-
0	0.03	1.37×	1.32×	1.00×	314	60 / 1.36×	-	-	2	0.03	1.29×	1.29×	1.00×	232	36 / 1.28×	-	-
0	0.04	1.34×	1.25×	1.00×	288	56 / 1.34×	-	-	2	0.04	1.24×	1.24×	1.00×	216	32 / 1.24×	-	-
0	0.05	1.32×	1.18×	1.05×	268	52 / 1.30×	-	-	2	0.05	1.20×	1.20×	1.00×	204	28 / 1.22×	-	-
0	0.06	1.26×	1.12×	1.21×	248	44 / 1.27×	-	-	2	0.06	1.17×	1.17×	1.00×	196	24 / 1.18×	-	-
0	0.07	1.26×	1.12×	1.21×	248	40 / 1.25×	-	-	2	0.07	1.17×	1.17×	1.00×	196	24 / 1.18×	-	-
0	0.08	1.12×	1.05×	1.21×	218	20 / 1.12×	-	-	2	0.08	1.16×	1.16×	1.00×	192	20 / 1.15×	-	-
0	0.09	1.08×	1.04×	1.21×	212	12 / 1.08×	3525 s	35 ms	2	0.09	1.16×	1.15×	1.00×	190	20 / 1.15×	10370 s	48 ms
1	0.01	1.18×	1.18×	1.00×	104	20 / 1.20×	156 s	14 ms	3	0.01	1.30×	1.30×	1.00×	262	44 / 1.30×	OoM	73 ms
1	0.02	1.15×	1.15×	1.00×	98	16 / 1.16×	-	-	3	0.02	1.28×	1.29×	1.00×	256	40 / 1.27×	-	-
1	0.03	1.14×	1.14×	1.00×	94	16 / 1.16×	-	-	3	0.03	1.26×	1.26×	1.00×	248	40 / 1.27×	-	-
1	0.04	1.14×	1.13×	1.00×	92	16 / 1.16×	-	-	3	0.04	1.24×	1.25×	1.00×	242	36 / 1.25×	-	-
1	0.05	1.12×	1.11×	1.00×	88	12 / 1.12×	-	-	3	0.05	1.21×	1.21×	1.00×	230	32 / 1.22×	-	-
1	0.06	1.12×	1.10×	1.00×	86	12 / 1.12×	-	-	3	0.06	1.21×	1.21×	1.00×	228	28 / 1.19×	-	-
1	0.07	1.12×	1.10×	1.00×	86	12 / 1.12×	-	-	3	0.07	1.21×	1.20×	1.00×	226	32 / 1.22×	-	-
1	0.08	1.11×	1.10×	1.00×	84	12 / 1.12×	-	-	3	0.08	1.16×	1.14×	1.00×	204	24 / 1.16×	-	-
1	0.09	1.10×	1.09×	1.00×	82	12 / 1.12×	247 s	14 ms	3	0.09	1.08×	1.08×	1.00×	182	12 / 1.08×	OoM	16 ms

Table VI: GNN-RE Dataset Configuration. A shared training set is used for all groups. Test pairs indicate functional (green) vs. appearance (red) circuits.

Split	Circuits (Signal)
Training	banyan (8_out_5), memctrl (po0501), c2670 (N3809), c3540 (N4589), i2c (po061), c5315 (N7504), c7552 (N10351), c432 (N421)
Group 1	Val: memctrl (po0198), c5315 (N7705) Test: memctrl/c5315 (po0198 / N7504), c5315/c432 (N7705 / N421)
Group 2	Val: i2c (po059), c432 (N430) Test: i2c/c2670 (po059 / N3809), c432/memctrl (N430 / po0472)
Group 3	Val: c7552 (N10110), c5315 (N7705) Test: c7552/i2c (N10110 / po061), c5315/memctrl (N7705 / po0501)
Group 4	Val: c7552 (N10110), c432 (N430) Test: c7552/memctrl (N10110 / po0501), c432/c5315 (N430 / N7504)

Table VII: GNN-RE F1-score comparison across four groups and five p values. $F1_{\textbf{Val}}$: Validation accuracy. $F1_{\text{Cryptic}}$ (lower = better), $F1_{\text{Mimetic}}$ (higher = better). Superscripts: **Ours** = IP Camouflage, **Rand5%** = 5% CG insertion from [7], **RandAM** = area-matched random insertion. The F1-score for random guess is 0.125.

Group	Metric	$p = .1$	$p = .3$	$p = .5$	$p = .7$	$p = .9$
1	$F1_{\text{Val}}$	0.59	0.58	0.59	0.58	0.58
$F1_{\text{Cryptic}}$	Ours	**0.50**	0.54	**0.52**	**0.46**	**0.48**
	Rand5%	0.58	0.58	0.58	0.58	0.58
	RandAM	0.52	**0.53**	0.53	0.52	0.53
$F1_{\text{Mimetic}}$	Ours	**0.28**	**0.26**	**0.28**	**0.42**	**0.39**
	Rand5%	0.12	0.12	0.12	0.12	0.12
	RandAM	0.18	0.14	0.18	0.11	0.18
2	$F1_{\text{Val}}$	0.57	0.55	0.57	0.55	0.57
$F1_{\text{Cryptic}}$	Ours	0.41	0.38	0.35	**0.33**	**0.34**
	Rand5%	0.56	0.56	0.56	0.56	0.56
	RandAM	**0.33**	**0.33**	**0.33**	**0.33**	**0.34**
$F1_{\text{Mimetic}}$	Ours	**0.27**	**0.25**	**0.24**	**0.23**	**0.20**
	Rand5%	0.12	0.12	0.12	0.12	0.12
	RandAM	0.23	0.16	0.16	0.16	0.16
3	$F1_{\text{Val}}$	0.52	0.51	0.51	0.53	0.51
$F1_{\text{Cryptic}}$	Ours	0.35	0.33	0.36	0.30	**0.27**
	Rand5%	0.50	0.50	0.50	0.50	0.50
	RandAM	**0.27**	**0.27**	**0.27**	**0.27**	**0.27**
$F1_{\text{Mimetic}}$	Ours	**0.26**	**0.23**	**0.38**	**0.33**	**0.31**
	Rand5%	0.12	0.12	0.12	0.12	0.12
	RandAM	0.21	0.21	0.21	0.21	0.21
4	$F1_{\text{Val}}$	0.51	0.51	0.51	0.54	0.52
$F1_{\text{Cryptic}}$	Ours	**0.30**	**0.38**	**0.37**	**0.41**	**0.34**
	Rand5%	0.51	0.51	0.51	0.51	0.51
	RandAM	0.41	0.41	0.41	**0.41**	0.41
$F1_{\text{Mimetic}}$	Ours	**0.30**	0.21	**0.33**	**0.33**	**0.32**
	Rand5%	0.12	0.12	0.12	0.12	0.12
	RandAM	0.27	**0.27**	0.27	0.27	0.27

scores compared to the baseline of random covert gate insertion. This dual effect indicates not only reduced classification accuracy of functional (true) circuits, but also increased misclassification as appearance (decoy) circuits. Such performance highlights the effectiveness of our structured, model-guided camouflage methodology in misleading GNN-based reverse engineering. Compared to the 5% random insertion scheme proposed in [7], our approach provides a stronger functional concealment and adversarial deception, under similar or lower area overhead.

VI. Conclusion and Future Work

This paper introduced a novel IC camouflaging methodology that integrates ML-driven approaches and covert gates to promote hardware security and anti-reverse engineering. By bridging cryptic and mimetic cyber deception strategies, our approach achieved dual-layered camouflage, not only concealing the IP's functionality but also mimicking the appearance of the camouflaged IP as another misleading design. Experimental validation demonstrated the effectiveness of the proposed method in achieving SAT resistance with low structural and performance overhead. Additionally, the methodology proved more resilient against AI-enhanced reverse engineering attacks than existing approaches. These findings represent a significant advancement, setting a new standard for IC camouflaging by incorporating cyber deception principles. Besides the exploration of promising new applications, our future work will focus on expanding the proposed methodology to generic process design kits or PDKs (as opposed to AIGs), investigating alternative AI techniques, and evaluating the effectiveness of fault and side-channel attacks.

Acknowledgments

The authors would like to thank Charles Kamhoua, Frederica Nelson, and Gregory Shearer of the Army Research Lab (ARL) for their encouragement and inspiration for this research. Also, this work has been supported in part by the US Army Research Office (ARO) under award # W911NF-19-1-0102 and in part by the Department of Defense through the Science, Mathematics, and Research for Transformation (SMART) Scholarship-for-Service Program.

REFERENCES

[1] S. E. Quadir, J. Chen, D. Forte, N. Asadizanjani, S. Shahbazmohamadi, L. Wang, J. Chandy, and M. Tehranipoor, "A survey on chip to system reverse engineering," *ACM journal on emerging technologies in computing systems (JETC)*, vol. 13, no. 1, pp. 1–34, 2016.

[2] R. Torrance and D. James, "The state-of-the-art in semiconductor reverse engineering," in *Proceedings of the 48th Design Automation Conference*, 2011, pp. 333–338.

[3] K. Shamsi, M. Li, K. Plaks, S. Fazzari, D. Z. Pan, and Y. Jin, "Ip protection and supply chain security through logic obfuscation: A systematic overview," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 24, no. 6, pp. 1–36, 2019.

[4] R. P. Cocchi, J. P. Baukus, L. W. Chow, and B. J. Wang, "Circuit camouflage integration for hardware ip protection," in *Proceedings of the 51st Annual Design Automation Conference*, 2014, pp. 1–5.

[5] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri, "Security analysis of integrated circuit camouflaging," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 709–720.

[6] J. Pawlick, E. Colbert, and Q. Zhu, "A game-theoretic taxonomy and survey of defensive deception for cybersecurity and privacy," *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 1–28, 2019.

[7] B. Shakya, H. Shen, M. Tehranipoor, and D. Forte, "Covert gates: Protecting integrated circuits with undetectable camouflaging," *IACR transactions on cryptographic hardware and embedded systems*, pp. 86–118, 2019.

[8] M. G. Rekoff, "On reverse engineering," *IEEE Transactions on systems, man, and cybernetics*, no. 2, pp. 244–252, 1985.

[9] C. Bao, D. Forte, and A. Srivastava, "On reverse engineering-based hardware trojan detection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 1, pp. 49–57, 2015.

[10] S. Rajendran and M. L. Regeena, "A novel algorithm for hardware trojan detection through reverse engineering," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 4, pp. 1154–1166, 2021.

[11] M. Yasin, J. Rajendran, O. Sinanoglu, M. Yasin, J. Rajendran, and O. Sinanoglu, "The sat attack," *Trustworthy Hardware Design: Combinational Logic Locking Techniques*, pp. 47–56, 2020.

[12] M. El Massad, S. Garg, and M. V. Tripunitara, "The sat attack on ic camouflaging: Impact and potential countermeasures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 8, pp. 1577–1590, 2019.

[13] J. Wang, H. Yang, S. Deng, and X. Li, "Cimsat: Exploiting sat analysis to attack compute-in-memory architecture defenses," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 3436–3450.

[14] H. Zhou, R. Jiang, and S. Kong, "Cycsat: Sat-based attack on cyclic logic encryptions," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 49–56.

[15] Y. Shen, Y. Li, S. Kong, A. Rezaei, and H. Zhou, "Sigattack: New high-level sat-based attack on logic encryptions," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 940–943.

[16] M. T. Rahman, S. Tajik, M. S. Rahman, M. Tehranipoor, and N. Asadizanjani, "The key is left under the mat: On the inappropriate security assumption of logic locking schemes," in *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2020, pp. 262–272.

[17] S. K. Monfared, K. Mitard, A. Cannon, D. Forte, and S. Tajik, "Laserescape: Detecting and mitigating optical probing attacks," *arXiv preprint arXiv:2405.03632*, 2024.

[18] B. Erbagci, C. Erbagci, N. E. C. Akkaya, and K. Mai, "A secure camouflaged threshold voltage defined logic family," in *2016 IEEE International symposium on hardware oriented security and trust (HOST)*. IEEE, 2016, pp. 229–235.

[19] S. Malik, G. T. Becker, C. Paar, and W. P. Burleson, "Development of a layout-level hardware obfuscation tool," in *2015 IEEE computer society annual symposium on VLSI*. IEEE, 2015, pp. 204–209.

[20] C. Yu, X. Zhang, D. Liu, M. Ciesielski, and D. Holcomb, "Incremental sat-based reverse engineering of camouflaged logic circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 10, pp. 1647–1659, 2017.

[21] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[22] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE signal processing magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[23] D. Bertucci and A. Endert, "Vae explainer: Supplement learning variational autoencoders with interactive visualization," *arXiv preprint arXiv:2409.09011*, 2024.

[24] P. B. López, M. G. Pérez, and P. Nespoli, "Cyber deception: State of the art, trends and open challenges," *arXiv preprint arXiv:2409.07194*, 2024.

[25] Z. Zhang, L. Njilla, C. A. Kamhoua, and Q. Yu, "Thwarting security threats from malicious fpga tools with novel fpga-oriented moving target defense," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 3, pp. 665–678, 2018.

[26] S. K. Monfared, D. Forte, and S. Tajik, "Randohm: Mitigating impedance side-channel attacks using randomized circuit configurations," *arXiv preprint arXiv:2401.08925*, 2024.

[27] Z. Wang, C. Bai, Z. He, G. Zhang, Q. Xu, T.-Y. Ho, B. Yu, and Y. Huang, "Functionality matters in netlist representation learning," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 61–66.

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

[29] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.

[30] M. Zhang, S. Jiang, Z. Cui, R. Garnett, and Y. Chen, "D-vae: A variational autoencoder for directed acyclic graphs," *arXiv preprint arXiv:1904.11088*, 2019.

[31] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014. [Online]. Available: https://arxiv.org/abs/1406.1078

[32] L. Alrahis, A. Sengupta, J. Knechtel, S. Patnaik, H. Saleh, B. Mohammad, M. Al-Qutayri, and O. Sinanoglu, "Gnn-re: Graph neural networks for reverse engineering of gate-level netlists," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 8, pp. 2435–2448, 2022.

[33] M. C. Hansen, H. Yalcin, and J. P. Hayes, "Unveiling the iscas-85 benchmarks: A case study in reverse engineering," *IEEE Design & Test of Computers*, vol. 16, no. 3, pp. 72–80, 1999.

[34] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The epfl combinational benchmark suite," in *Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS)*, 2015.

[35] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai *et al.*, "Deep graph library: A graph-centric, highly-performant package for graph neural networks," *arXiv preprint arXiv:1909.01315*, 2019.

[36] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[37] A. Hagberg, P. Swart, and D. Chult, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference*, 2008, pp. 11–15.

[38] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "Graphsaint: Graph sampling based inductive learning method," *arXiv preprint arXiv:1907.04931*, 2020.