ReGDS: A Reverse Engineering Framework from GDSII to Gate-level Netlist

Rachel Selina Rajarathnam[†], Yibo Lin[‡], Yier Jin[*] and David Z. Pan[†]
[†]Department of Electrical & Computer Engineering, The University of Texas at Austin, TX, USA
[‡]School of Electronics Engineering & Computer Science, Peking University, Beijing, China
[*]Department of Electrical & Computer Engineering, University of Florida, Gainesville, FL, USA
rachelselina.r@utexas.edu, yibolin@pku.edu.cn, yier.jin@ece.ufl.edu, dpan@ece.utexas.edu

***Abstract*—With many fabless companies outsourcing integrated circuit (IC) fabrication, the extent of design information recoverable by any third-party foundry remains clouded. While traditional reverse engineering schemes from the layout employ expensive high-resolution imaging techniques to recover design information, the extent of design information that can be recovered by the foundry remains ambiguous. To address this ambiguity, we propose ReGDS, a layout reverse engineering (RE) framework, posing as an inside-foundry attack to acquire original design intent. Our framework uses the layout, in GDSII format, and the technology library to extract the transistor-level connectivity information, and exploits unique relationship-based matching to identify logic gates and thereby, recover the original gate-level netlist. Employing circuits ranging from few hundreds to millions of transistors, we validate the scalability of our framework and demonstrate 100% recovery of the original design from the layout.**

To further validate the effectiveness of the framework in the presence of obfuscation schemes, we apply ReGDS to layouts of conventional XOR/MUX locked circuits and successfully recover the obfuscated netlist. By applying the Boolean SATisfiability (SAT) attack on the recovered obfuscated netlist, one can recover the entire key and, thereby, retrieve the original design intent. Thus ReGDS results in accelerated acquisition of the gate-level netlist by the attacker, in comparison to imaging-based RE schemes. Our experiments unearth the potential threat of possible intellectual property (IP) piracy at any third-party foundry.

***Index Terms*—Reverse Engineering, Supply chain security, Subgraph Isomorphism**

I. Introduction

Advances in integrated circuit (IC) manufacturability and the ability to outsource IC fabrication have accelerated technology research and development, resulting in reduced time-to-market, and affordable products and services. However, outsourcing fabrication has resulted in a significant increase in intellectual property (IP) piracy concerns [1]. The eminent IP piracy schemes could be broadly classified as chip overbuilding [2], counterfeiting [3] and Reverse Engineering (RE) [4]. Traditional imaging-based RE techniques involve IC decapsulation and delayering, to obtain the transistor connectivity information using high-resolution imaging techniques [4]–[6]. Tools such as 'Degate' [7], have been developed to assist image RE, that match the standard cell library against the design layout to identify logic gates. From the transistor connectivity information, one can retrieve the gate-level netlist [8], to disclose design intent. Several approaches have been explored for transistor-level to gate-level netlist conversion,

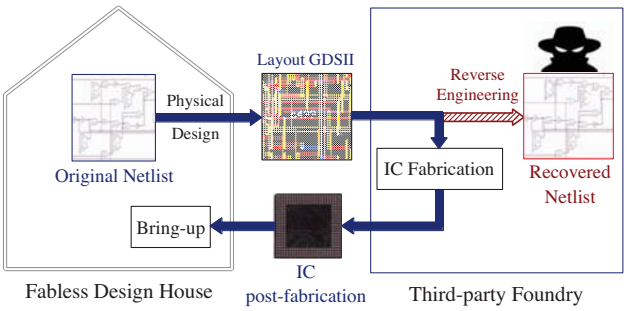

Fig. 1: Potential threat of IP piracy at a third-party foundry.

such as Boolean logic analysis [9], graph-based pattern matching [10], [11], hybrid approaches [12] and Genetic algorithm [13]. In addition, there are also electronic design automation (EDA) tools available for the same [14], [15]. With the gate-level netlist, module/unit level information can be obtained to recover original design intent [16], [17]. The RE schemes, although primarily intended for legal verification of unlicensed IP use and product failure analysis, have also been employed for illicit purposes. With the escalation of IP piracy, developing defending strategies at various levels has become fundamental.

Albeit several IP protection schemes have been devised to thwart IP piracy, a foundry-level attacker is well-equipped to be able to overpower the defense schemes [18]. The prominent IP piracy prevention methods include IC Camouflaging, Logic Locking and Split Manufacturing. While IC camouflaging disables imaging-based RE schemes using look-alike standard cells in layout [19], [20], the complete control on implementation by the foundry renders it ineffective to hide design intent. On the other hand, Logic locking involves additional functionality and key inputs to lock the original circuit, rendering it non-functional in the absence of the original key [21]. However, the improvement of attack schemes continue to challenge the effectiveness of logic locking [18], [22]–[25]. Furthermore, Split manufacturing aims to avoid unintended modifications to the original layout by an untrusted foundry, by dividing the design layout into Front End of Line (FEOL) and Back End of Line (BEOL) layers and manufacturing them in different foundries [26].

As many companies choose to remain fabless due to the

978-1-7281-7405-1/20/$31.00 ©2020 IEEE

expensive manufacturing costs, the security risk of loss of control over the fabrication process is inevitable. While an untrusted foundry may have hideous intent [18], the extent of original design information available to a trusted foundry remains unexplored. We explore the possibility of reverse engineering within the foundry by an untrusted entity to understand the extent of design information that is recoverable. The fabless design house generates the layout GDSII after backend design and sends it to a third-party foundry for fabrication. After fabrication, the design house receives the IC for bring-up and production, as shown in Fig. 1, where the solid blue arrows show the intended design flow. With the technology library available, it is plausible for any third-party foundry to recover the original design intent by reverse engineering the design layout, even without expensive imaging-based RE techniques.

In this paper, we examine the potential attack within a third-party foundry, by reverse engineering of the original design intent from the layout, depicted as the dashed red arrow in Fig. 1. To investigate the possible attack at the foundry, we develop the first layout reverse engineering framework `ReGDS`, from layout GDSII to the gate-level netlist. The framework employs the Layout vs Schematic comparison (LVS) tool used for physical verification [27], and a graph matching algorithm. Transistor-level connectivity information is extracted from the layout, using the LVS tool. The existing transistor-level to logic gate-level conversion schemes require prior knowledge of original design information, which is unavailable during layout RE [8]–[14], [28]. For digital circuits, the extracted netlist, from the layout GDSII, does not contain explicit power network information, resulting in inconsistent transistor terminal connections. To compensate for this inconsistency, we develop a novel connectivity representation for transistors in digital circuits, to facilitate logic gate identification using graph matching algorithm. As the proposed framework utilizes the technology library as one of its input, it can be extended to all digital transistor-based logic styles, such as CMOS, pass transistor, etc.

`ReGDS` provides the capability of reverse engineering from the layout to Register Transfer Level (RTL) by integrating techniques that recover higher-level macro units from the gate-level netlist [16]. This work opens the door for a new direction in layout reverse engineering, paving the way for gate-level security analysis using layout information. Our main contributions are listed as follows:

- To the best of our knowledge, we are the first to develop an end-to-end layout reverse engineering framework, `ReGDS`, from GDSII to the gate-level netlist, to expose the potential threat of IP piracy at any third-party foundry.
- For a unique representation of transistor connections, we propose a novel digital connectivity index (DCI) coding scheme to handle inconsistencies in transistor terminal connections, to accelerate connectivity-based pattern matching.
- In comparison to traditional imaging-based RE schemes, the proposed `ReGDS` framework significantly accelerates the process of layout reverse engineering to retrieve the original gate-level netlist at the foundry.
- Experimental results demonstrate successful and complete recovery of the original design from both unobfuscated and obfuscated layouts, affirming the inherent threat of possible IP piracy at the foundry.

The rest of the paper is organized as follows: Section II provides a background on the Layout vs Schematic comparison check and graph matching algorithm, employed in the proposed framework. Section III presents the `ReGDS` framework and the experiments are reported in Section IV. Section V demonstrates the effectiveness of the proposed `ReGDS` framework even in the presence of obfuscation schemes, and finally, Section VI concludes the paper.

II. Preliminaries

This section provides a brief overview to the Layout vs Schematic comparison check, used in physical design verification, and graph matching problem, which form an integral part of the proposed `ReGDS` framework.

A. Layout vs Schematic Comparison Check

Circuit connectivity at different stages of design - schematic and layout, is widely represented in SPICE format [29], comprising of subcircuit definition(s) referred to as '*subckt*(s)'. We refer to leaf circuit elements such as transistors, resistors, capacitors, and diodes, as '*devices*'. While Design Rule Checks ensure manufacturability of the design layout, the Layout vs Schematic comparison (LVS) check ensures the functional correspondence of the design layout to the original design intent. The LVS checker reads in a source database (circuit schematic or gate-level netlist) and the corresponding layout database (in GDSII format) along with the technology library rule information, consisting of device recognition and layout metal layer definitions, as depicted in Fig. 2.

Intermediate SPICE netlist are generated from both the source and layout databases, and a one-to-one correspondence matching is performed based on '*device*' type and connectivity. The generated intermediate SPICE netlists differ in the hierarchical structure - the source extracted netlist is logic-driven, while that of the layout is based on physical location and connectivity. While the LVS tool allows both *flat* and *hierarchical* modes of operation, the latter is preferred for scalability. In *hierarchical* mode, the LVS tool runtime is in the order of $O(M)$, where M refers to the total hierarchical units in the layout database. As the intermediate netlists are

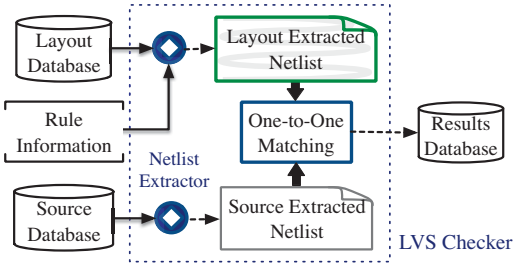

Fig. 2: Layout vs Schematic check.

generated irrespective of whether the LVS check passes or fails, we obtain the layout extracted SPICE netlist even in the absence of the original source database.

The layout extracted netlist consists of the below three types of *subckts*:

- *Leaf subckts* consists of *devices*,
- *Mixed subckts* comprising of both *devices* and other *subckt* instances, and
- *Hierarchical subckts* contains instances of other *subckts*.

The top-level modules are *hierarchical subckt*s, while the leaf modules are *leaf subckt*s. *Mixed subckt*s are generally found distributed at other levels.

B. Graph Isomorphism

Exact graph matching or graph isomorphism identifies the unique one-to-one mapping between the sets of vertices of two graphs. Graphs $G(V_G, E_G)$ and $H(V_H, E_H)$ are isomorphic if there exists a bijective mapping, between the sets of vertices, $f : V_G \rightarrow V_H$ such that an edge $(u, v) \in E_G$ if and only if $(f(u), f(v)) \in E_H$, and is represented as $G \simeq H$. Subgraph isomorphism, a generalization of the graph isomorphism problem, determines the existence of a subgraph, within the larger graph G, that is isomorphic to H. However, as graph isomorphism is a NP-problem and the subgraph isomorphism problem is NP-complete, no known polynomial algorithms exist [30].

In this work, we employ the *vf2* subgraph isomorphism algorithm [30], which is based on Depth First Search and backtracking, for logic gate identification. This algorithm has a spatial complexity of $O(|V_G|)$ with a best and worst case time complexity of $O(|V_G|^2)$ and $O(|V_G|! \cdot |V_G|)$, respectively, where $|V_G|$ represents the number of vertices in the larger graph G. To further improve the runtime of the subgraph isomorphism algorithm, we incorporate partial matching and a staggered approach, as discussed in Section III-C.

III. ReGDS Framework

With the availability of the design layout and the technology library, we explore the extent of design information available to a foundry, posing as an insider attack. The proposed layout reverse engineering framework, ReGDS, is illustrated in Fig. 3. Leveraging the LVS tool, we extract the transistor connectivity information from the layout ❶, as described in Section III-A. Due to the absence of power network information and thereby the current flow direction, there are inconsistencies in the transistor terminal connections. To overcome this, we develop a novel connectivity-based representation for digital circuits ❷, as explained in Section III-B. Connectivity graphs are constructed from the transistor network and pattern matching is employed to identify logic gates and thereby recover the original gate-level netlist ❸, as detailed in Section III-C.

As the proposed ReGDS framework considers the technology library as an input for reverse engineering, it is applicable to all digital transistor-based logic styles. In this work, we consider CMOS technology implementation as a proof of concept.

A. Transistor Connectivity Extraction

The LVS check is employed to extract transistor connectivity from the design layout, as discussed in Section II-A. A synthetic source database is constructed that adheres to the same technology as the design layout. For instance, a simple inverter netlist is sufficient to constitute the source database. With the constructed source database and the available design layout and technology library, the LVS check is run. Though the check fails due to mismatches between the layout and the source databases, the transistor-level connectivity information is completely extracted from the layout database, in SPICE format. A post-processing step is applied on the layout extracted netlist, consisting of the following operations:

- To reduce the problem size, multiple devices connected in parallel, with similar physical features, are reduced to a single device, and
- *Mixed subckt*s, containing both hierarchical instances and *device*s, are identified and flattened, to enable logic gate identification.

B. Digital Connection Graph

The repetitive standard cell blocks in digital circuit facilitate the use of pattern matching for logic gate identification. To represent connectivity between various circuit elements in a graph $G(V, E)$, P. Wu et al. introduced the Terminal Connectivity Index (TCI) coding scheme [31], which represents transistors as a set of vertices V and connections as a set of edges E, with edge weights representing transistor terminal connectivity. The lack of explicit power connectivity information in digital circuits, results in the layout extracted netlist to have transistor *Drain*/*Source* terminal connections that do not comply with the current flow direction. For instance in CMOS technology, the N-type transistor has the *Source* terminal at a lower potential with respect to the *Drain* terminal, while the P-type transistor has *Source* terminal at a higher potential in comparison to the *Drain*. However, the inconsistency in transistor *Drain*/*Source* terminal connectivity in the layout extracted SPICE netlist, results in one-to-many mapping for the same affiliation, with the TCI coding scheme [31].

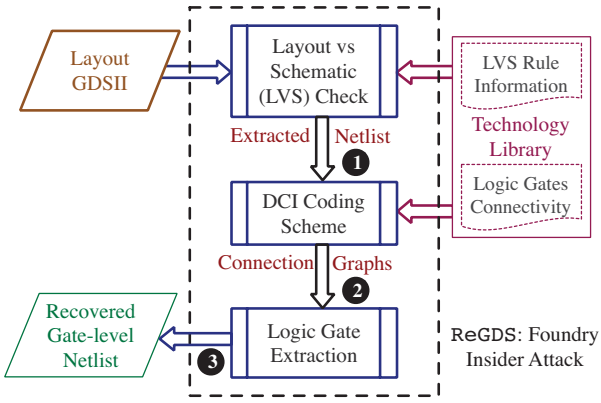

Fig. 3: The proposed ReGDS framework.

TABLE I: Digital Connectivity Index for transistors M_i and M_j, where D, G, S and B represent $Drain$, $Gate$, $Source$ and $Bulk$ terminals, respectively.

$DCI_{i,j}$		Terminals of M_j		
		D/S	G	D/S-B
Terminals of M_i	D/S	8	2	-
	G	2	32	-
	D/S-B	-	-	1

Inspired by the TCI coding scheme, we develop a Digital Connectivity Index (DCI) coding scheme, to ensure a bijective mapping between the transistor connectivity and its corresponding graph representation. The edge weights for the connection between any two transistors M_i and M_j, based on the proposed DCI coding scheme, are enumerated in Table I. We construct a labeled undirected weighted Digital Connection Graph (DCG), $G_D(V_D,\ E_D)$, using the proposed DCI coding scheme, where V_D represents the set of transistors and the set of weighted edges E_D represent transistor connectivity. The edge weight $w_{i,j}$ is the sum of all existing digital connectivity indices between the transistors M_i and M_j,

$$w_{i,j} = \alpha_{i,j}\left(\sum_{M_i}\sum_{M_j} DCI_{i,j} + \beta_i\beta_j\right) \tag{1}$$

where $\alpha_{i,j} \in \{0,1\}$ represents the existence of connection between transistors M_i and M_j, and $\beta_i, \beta_j \in \{0,1\}$ represent the existence of $Drain/Source$ to $Bulk$ terminal tie-off in transistors M_i and M_j, respectively. For example, if transistors M_i and M_j are connected, $\alpha_{i,j} = 1$, and when transistor M_i has $Drain/Source$ terminals connected to $Bulk$, $\beta_i = 1$.

To ensure unique representation of transistor connectivity, we assign same weights for $Drain/Source$ terminal connections between transistors M_i and M_j, allowing the terminals to be used interchangeably. As the connection between two logic gates can be represented as the connection from $Drain/Source$ terminal of M_i to $Gate$ terminal to M_j or vice versa, we assign a minimum weight of 2. Next, we consider connections between $Drain/Source$ terminals of transistors and assign a weight of 8; a weight of 4 is not assigned as it is possible to have two connections between M_i and M_j each with weight 2, resulting in a total of 4. Similarly, connections between the $Gate$ terminals of transistors are assigned the largest weight of 32, as they generally represent connectivity within a logic gate. The additional weight of 1 is added if $Drain/Source$ to $Bulk$ terminal tie-off exists in both the transistors, to signify the end of the pullup/pulldown network in a digital circuit.

To understand the bijective mapping of the proposed DCI coding scheme, we consider two different SPICE implementations of a 2-input NAND gate by interchanging their $Drain/Source$ terminal connections, as shown in Fig. 4(a). Using the TCI coding scheme [31], we obtain two different connection graphs as depicted in Fig. 4(b), while the proposed DCI coding scheme results in a unique DCG, as shown in Fig. 4(c). Although the $Drain/Source$ terminal connections are interchanged, the actual connectivity remains the same, which is apprehended by our DCI coding scheme. For instance, consider transistors M_2 and M_3, which have their $Drain/Source$ terminals connected to the same net and have $Bulk$ terminal tied off to $Drain/Source$ terminals. This results in an edge weight of $w_{2,3} = 1 \times (8 + 8 + 1) = 17$ between M_2 and M_3, as shown in Fig. 4(c).

Using the proposed DCI coding scheme, we generate DCGs corresponding to the flattened *Mixed* and *leaf subckt*s within the layout extracted netlist. *Hierarchical subckt*s, containing only hierarchical components, are not considered for logic gate identification. We annotate Γ to represent the collection of DCGs that represents the logic gates in the technology library.

C. Logic Gate Identification

Logic gates are identified by connectivity-based matching of each *subckt* digital connectivity graph (DCG) $G_D(V_D,E_D)$, against the set of technology library logic gate DCGs Γ. Net renaming is employed for tie offs in the layout netlist to avoid mismatches during pattern matching. As the worst case time

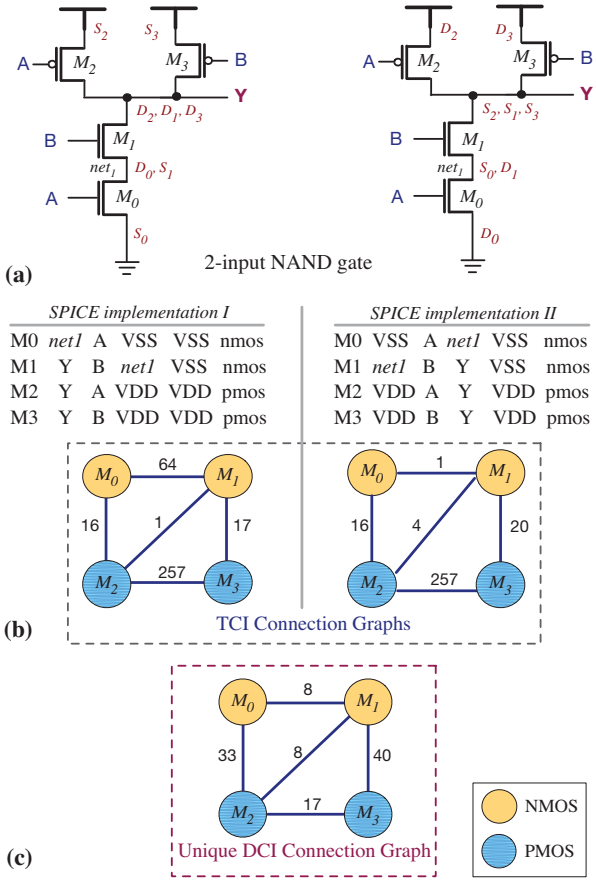

Fig. 4: (a) Two implementations of 2-input NAND gate by interchanging transistor $Drain/Source$ terminal connections. (b) Terminal connectivity graphs using TCI [31] coding scheme. (c) Unique Digital connectivity graph using the proposed DCI coding scheme.

complexity for subgraph isomorphism is $O(|V_D|! \cdot |V_D|)$, as mentioned in Section II-B, we employ *Partial match and logic stitch*, and a *Staggered* approach to further speed up the logic gate identification process.

1) Partial Match & Logic Stitch: To accelerate the pattern matching process, we break down large complex gates and identify them as a combination of multiple simpler gates. For example, a 2:1 multiplexer (MUX) can be built using one inverter and two transmission gates, which can be individually identified and grouped together, as shown in Fig. 5. We extend *Partial match and Logic stitch* to implement frequently occurring complex gates such as MUX, XOR/XNOR, majority gates, and adders.

2) Staggered Pattern Matching: A straightforward way for logic gate identification is to arrange the set of library logic gate DCGs in decreasing order of size, as Γ', and compare with the *subckt* DCG G_D. We refer to this approach as the *Baseline*. Although the technology library consists of several logic gates, only a portion is generally utilized for physical design implementation. For instance, complex logic gates with six or more inputs are less probable, while basic logic gates such as NOT, NAND, NOR, XOR, XNOR, and flip flops are highly likely to be used in the design implementation. We implement a *Staggered* approach which exploits the frequency of logic gate occurrence and the dependency of pattern matching algorithm time complexity on the vertices of the larger graph. The set of library DCGs Γ, is divided into smaller k non-overlapping sets of DCGs Γ_x based on the graph size and logic gate frequency of occurrence, such that $\Gamma_x \subset \Gamma$, where x=1, 2, ..., k. After each iteration of pattern matching, the identified vertices in *subckt* DCG, G_D, are removed and a new DCG G'_D, is generated such that $|V(G'_D)| \leq |V(G_D)|$. This reduction in problem size helps to further reduce the logic gate extraction runtime. As only the identified vertices in the graph at any stage are removed before the next stage, there is no loss of information, and the solution obtained remains optimal.

After logic gates are identified in all the *subckt*s (flattened *mixed* and *leaf*) within the layout extracted netlist, a gate-level HDL netlist is written out. The pseudo-code of the logic gate extraction (LGE) algorithm is summarized in Algorithm 1. Only *Mixed* and *leaf subckt*s are considered for LGE (lines 1-2) and the *Mixed subckt*s are flattened (line 4). Digital Connectivity Graphs are constructed for flattened *Mixed* and *leaf subckt*s (line 6). Logic gates are identified using a subroutine (line 7),

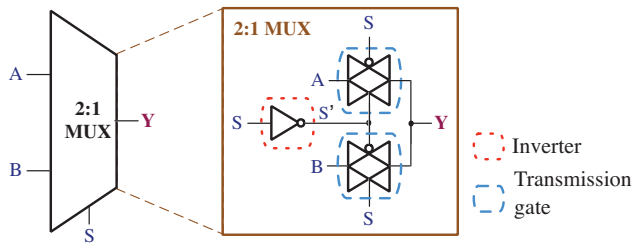

Fig. 5: Partial match and logic stitch of 2:1 MUX.

Algorithm 1 Logic Gate Extraction

Input: Layout extracted SPICE netlist with N subckts $S_{D_1}, S_{D_2}, \ldots, S_{D_N}$; Technology library-based DCGs Γ
Output: Gate-level Netlist

```
1: for i ← 1 to N do
2:     if S_Di contains device(s) then
3:         if S_Di is a Mixed subckt then
4:             Flatten S_Di;
5:         end if
6:         Construct DCG G_Di from S_Di;        ▷ Section III-B
7:         LogicGateIdentification (G_Di, Γ);
8:     end if
9: end for
10: return Gate-level Netlist;

11: function LogicGateIdentification(G_Di, Γ):
12:     for x ← 1 to k do                        ▷ Staggered matching
13:         subGraph Isomorphism (G_Di, Γ_x);
14:         Remove matched vertices;
15:         Construct G'_Di ∋ |V(G'_Di)| ≤ |V(G_Di)|;
16:     end for
17: end function
```

which incorporates *vf2* subgraph isomorphism algorithm [30] (line 13). The matched vertices in the *subckt* DCG, G_D, are then removed to reconstruct smaller DCG, G'_D (lines 14-15). The steps are repeated for all k groups of library DCGs Γ for all *subckt*s, to obtain the gate-level netlist (line 10).

The time complexity of Algorithm 1, dominated by the graph matching algorithm, is $O(|V_S|! \cdot |V_S|)$, where V_S denotes the set of vertices in the largest *subckt* among the flattened *Mixed* and *leaf subckt*s. However, as $|V_S| << T_{CNT}$, where T_{CNT} is the total transistor count in the design, the overall LGE runtime is empirically in the order of $O(T_{CNT})$, based on experiments in Section IV. The space complexity of the LGE algorithm is $O(|V_S|)$.

To evaluate the effectiveness of the logic gate extraction, we employ *Recovery ratio* $\gamma \in [0, 1]$, defined as,

$$\gamma = \left(\frac{\eta_G}{\eta_T}\right) \tag{2}$$

where η_G is the number of transistors identified as logic gates and η_T is the total transistor count in the layout extracted SPICE netlist. The unique mapping of the proposed DCI coding scheme ensures accurate identification for pattern matching, resulting in better runtime compared to the TCI coding scheme with multiple representations for the same connectivity [31].

D. Summary

ReGDS provides an efficient layout reverse engineering framework to recover the gate-level netlist from the design layout, in GDSII format, with the availability of the technology library. The proposed framework leverages the Layout vs Schematic comparison tool, used for physical verification, to extract transistor-level connections from the design layout. A

unique representation for transistor-level affiliation, the DCI coding scheme, is developed to overcome the transistor terminal connectivity inconsistencies due to the absence of explicit power network information. Logic gates are identified using pattern matching to reconstruct the original gate-level netlist.

For an inside foundry attack, ReGDS framework significantly accelerates the process of layout RE to obtain gate-level netlist, in comparison to imaging-based RE schemes. Due to the utilization of the technology library in the proposed layout RE framework, it is applicable to all digital transistor-based logic styles and technologies. The effectiveness of our ReGDS framework is demonstrated and verified in Section IV.

The proposed layout RE framework, ReGDS, is developed to analyze the extent of design information retrievable by any third-party foundry, and thus assumes the availability of the standard cell library. However this might not always be the case. In the absence of the library, layout RE becomes very challenging without the layer, device and connectivity information, to retrieve transistor connections.

IV. ReGDS Methodology & Evaluation

Though the proposed ReGDS framework is applicable to all digital library-based logic styles, herein, we consider CMOS technology for evaluation. ISCAS benchmarks [32], [33] and the modules of Common Evaluation Platform (CEP) SoC platform [34], were employed to validate the proposed layout RE framework. Experiments were performed on a 4-core Linux machine powered by Intel i7-3770 CPU, running at 3.40GHz. All the circuits were synthesized using *Synopsys Design Compiler* [35] and physical implementation was facilitated through *Cadence Innovus* [28] tool to generate the layouts in GDSII format. Commercial *TSMC* 40*nm* library [36] was employed to emulate an insider foundry attack.

The extraction of transistor-level connectivity information from the design layout, using the LVS tool, is described in Section IV-A. Section IV-B reports the identification of logic gates from transistors to reconstruct the original gate-level netlist, using the custom C++ tool. A library independent version of the C++ routine is open sourced[1]. Comparisons, in terms of runtime and recovery, have been made between the different connectivity coding schemes - TCI [31] and the proposed DCI schemes. Finally, Section IV-C concludes with empirical time and space complexity analysis for the proposed layout RE framework.

A. SPICE Netlist Extraction

With the design layout, in GDSII format, and the technology library available, an inverter SPICE netlist was constructed to serve as the synthetic source database for all the experiments. *Mentor Graphics Calibre* LVS tool [27] was employed, in *hierarchical* mode, to extract the transistor-level connectivity information from the layout, in SPICE format. We refer to the time taken by the LVS tool as the *SPICE Netlist Extraction (SNE)* runtime and is in the order of $O(M)$, where M is the total hierarchical units in the layout. A post-processing step was performed on the layout extracted SPICE netlist, as mentioned in Section III-A, to facilitate logic gate extraction.

To validate the precision of the layout extracted netlist, LVS check was run to compare the former with the original gate-level netlist. LVS check passed for all the circuits [32]–[34], affirming the complete extraction of transistor-level connectivity information from the layout.

B. Connectivity-based Logic Gate Extraction

A custom tool was implemented in C++, for logic gate identification based on Algorithm 1. Using the proposed DCI coding scheme, we construct DCGs corresponding to the flattened *Mixed* and *leaf subckt*s in the layout extracted netlist. As the technology library contains several gates with both inverting and non-inverting variants, for example, NAND/AND, we construct DCGs only for the inverting variants and identify the non-inverting variant with an inverter, as AND+NOT for NAND. The DCGs for NAND/NOR/AOI logic gates were constructed instead of AND/OR/AO, and DCGs were constructed for other logic gates such as Flip Flops, adders and majority gates, that only have non-inverting variant. Thus we obtain a set of DCGs Γ, based on the logic gates in the technology library. With two sets of DCGs from the layout extracted netlist and the technology library, graph matching was done to recover the gate-level netlist as per Algorithm 1. We refer to the time taken by our custom tool as the *Logic Gate Extraction (LGE)* runtime.

While a *Baseline* approach arranges the logic gates in decreasing order of size to construct the set of technology library DCGs Γ', the proposed *Staggered* approach organizes the library gates into smaller k non-overlapping sets of DCGs Γ_x, $x = 1, 2, \ldots k$, based on size and frequency of occurrence, such that $\Gamma_x \subset \Gamma$. We compare the impact of *Staggered* approach on the *LGE* runtime on selected benchmark circuits [32]–[34], as tabulated in Table. II. It is important to note that both the approaches employ *Partial match & Logic stitch*, without which the runtime of *Baseline* approach would further increase. It is evident that the proposed *Staggered* approach outperforms the *Baseline* resulting in an average of 1.5× runtime reduction across all the benchmarks.

The logic gate composition of sample ISCAS circuits [32], [33], for the original (Ω) and the corresponding recovered (Υ) gate-level netlists are compared in Table III. It can be noticed that the number of complex gates, such as flip flops, XOR/XNOR, adders, multiplexers and majority gates, remain the same for both the original and recovered netlists. The difference in total logic gate count arises due to the difference in logic gates such as AND/OR, AND-OR-INVERT, inverters and buffers, caused by our initial consideration of inverting logic gate variants over their non-inverting counterparts.

We compare the performance of the proposed DCI coding scheme with the TCI coding scheme [31] in terms of *LGE* runtime, and the *Recovery ratio* γ as per Eqn. (2). The results of the experiments on all the considered circuits [32]–[34] were terminated after a timeout (*TO*) of 1 week and are

[1]https://github.com/rachelselinar/ReGDS-Logic-Gate-Extraction.

TABLE II: Impact of Staggered approach on LGE runtime using the proposed DCI coding scheme.

Benchmark Suite		ISCAS'85 [32]			ISCAS'89 [33]				CEP [34]				
Circuit		c3540	c5315	c6288	s953	s1423	s15850	s35932	SHA256	FIR	IDFT	RSA	RISCV
#Transistors		2.28k	4.54k	8.94k	1.82k	3.28k	19.5k	69.1k	87.5k	118.6k	1.66M	2.48M	7.37M
LGE Runtime (s)	Baseline	0.3	0.22	0.5	0.13	0.3	2.5	0.7	3.5	4.5	485.0	265.0	1028.0
	Staggered	0.13	0.17	0.36	0.07	0.12	2.0	0.55	2.85	3.8	470.0	194.0	934.0
Runtime Improvement		2.3×	1.3×	1.4×	1.9×	2.5×	1.3×	1.3×	1.2×	1.2×	1.1×	1.4×	1.1×

tabulated in Table IV. Due to one-to-many mapping by the TCI coding scheme [31], only circuits with less than $3k$ transistors converge within the *TO* and the recovery ratio γ_{TCI} is less than 0.5 for all experiments. The bijective mapping of the proposed DCI coding scheme clearly outperforms the TCI coding scheme in terms of both recovery ratio γ_{DCI} and runtime. The empirical space complexity of the LGE algorithm, employing the proposed DCI coding scheme, for CEP modules [34] is recorded in Table V. The size of the design layout, in GDSII format, and the memory usage of LGE algorithm are listed with the total transistor count in the considered benchmark.

In summary, based on the results, it is evident that:

- As TCI coding scheme [31] results in one-to-many mappings, only an average of $\gamma_{TCI} = 0.4$, i.e., 40% of transistors are identified as simple logic gates, such as INV/NAND/NOR. However, the unique representation using the proposed DCI coding scheme ensures $\gamma_{DCI} = 1.0$, i.e., 100% recovery of logic gates for all the circuits, facilitating the RE process.
- The *Recovery ratio* γ, directly affects the *LGE* runtime: the proposed DCI coding scheme always ensures better runtime than the TCI coding scheme, resulting in more than three orders of magnitude runtime difference.
- The proposed *Staggered* approach results in 1.5× average runtime reduction in comparison to the *Baseline*, with *Partial match & Logic stitch* approach.
- Our ReGDS framework scales well for large designs, with *LGE* runtime of ∼16 minutes for the CEP RISCV [34] module with 7.37M transistors.

TABLE III: Comparison of logic gate composition.

Circuit		#Gts	#FFs	#XRs	#AOs	#NDRs	#IVBs	#Ots
c432	Ω	84	-	-	44	19	21	-
	ϒ	91	-	-	43	21	27	-
c499	Ω	135	-	24	79	23	8	1
	ϒ	150	-	24	79	35	11	1
c3540	Ω	333	-	11	165	102	54	1
	ϒ	379	-	11	165	107	95	1
c6288	Ω	593	-	2	28	268	75	220
	ϒ	602	-	2	28	269	83	220
s349	Ω	71	15	2	18	14	16	6
	ϒ	81	15	2	18	14	26	6
s838	Ω	171	32	1	59	45	34	-
	ϒ	193	32	1	59	45	56	-
s1196	Ω	271	18	-	110	99	44	-
	ϒ	290	18	-	110	101	61	-
s1423	Ω	326	74	1	111	83	53	4
	ϒ	372	74	1	110	93	90	4

Ω and ϒ represent the original and corresponding recovered netlists.
#Gts: Total logic gates; **#FFs**: Flip Flop count; **#XRs**: XOR/XNOR gates; **#AOs**: AND-OR/OR-AND gates; **#NDRs**: AND/NAND/OR/NOR gates; **#IVBs** represents total inverter and buffer count in circuit; **#Ots** represents complex gates such as adders, MUX and majority gates.

Validation of Recovered Gate-level Netlist: We verify the equivalence of the recovered gate-level netlist with the original netlist using *Cadence Conformal Logic Equivalence Checker* (LEC) [28]. For all the considered circuits [32]–[34], LEC check passed to ensure the functional equivalence between the original and recovered netlists. Equivalence checking for some of the CEP modules such as *IDFT, DFT, RSA* and *RISCV* [34], containing more than 10k flip flops, was not feasible at the top level due to the large number of unmapped flip flop points (> 10k), and is beyond the scope of this work. Instead subunit implementation and verification was done to ensure equivalence between original and recovered netlists.

C. ReGDS Overall Assessment

The overall runtime of the ReGDS framework, comprising of the *SPICE Netlist Extraction* (SNE) and *Logic Gate Extraction* (LGE) runtimes, for selected benchmark circuits [32]–[34] is plotted in Fig. 6. SNE runtime refers to the time taken by the LVS tool in *hierarchical* mode, to obtain the layout extracted netlist. LGE runtime is the time required by our custom tool, employing the proposed DCI coding scheme, to identify logic gates and recover the original netlist.

The SNE runtime depends on the total number of *subckt*s or hierarchical levels in the layout extracted netlist. For example, consider the ISCAS and CEP benchmark circuits [33], [34] containing 19k to 69k transistors: *s15850* contains 189 *subckt*s and has larger SNE runtime, while *s38417* and *s35932*, containing 86 and 46 *subckt*s, respectively, have lower SNE runtime. Similary *AXI2WB* with higher *subckt* count of 183, has larger SNE runtime in comparison to *DES3* and *MD5* modules with 125 and 96 $subckt$s, respectively.

On the other hand, due to the hierarchical nature of the layout extracted netlist, the LGE runtime depends on the size of the largest *subckt*, among flattened *Mixed* and *leaf subckt*s. For instance, though CEP modules *IDFT* and *DFT* have similar transistor count of 1.6 million transistors, IDFT module has significantly larger LGE runtime compared to the latter. This is due to the presence of 8 large flattened *subckt*s in *IDFT*, each containing about 100k instances, while *DFT* only contains 3 large flattened *subckt*s of similar size.

Thus the proposed layout RE framework unveils the possibility of any third-party foundry to completely reverse engineer the design layout and recover the original gate-level netlist. In addition, ReGDS framework accelerates the reverse engineering process at the foundry, in comparison to traditional imaging-based RE schemes. On a further note, advanced process technologies should not affect the LGE runtime as it depends on the size of the largest *subckt*, among *Mixed* and *leaf subckt*s.

TABLE IV: Recovery and runtime evaluation of TCI [31] and the proposed DCI coding schemes.

Benchmark Suite	Circuit	#Transistors	#FFs	TCI Coding Scheme [31]		DCI Coding Scheme		Runtime Impr.† DCI vs TCI
				Runtime (s)	γ_{TCI}	Runtime (s)	γ_{DCI}	
ISCAS'85 [32]	c432	478	-	32	0.26	0.02	1.0	3.2×
	c1908	982	-	2,391	0.42	0.12	1.0	4.3×
	c880	1,076	-	2,472	0.38	0.05	1.0	4.7×
	c1355	1,394	-	15,031	0.42	0.03	1.0	5.7×
	c499	1,410	-	173,612	0.34	0.03	1.0	6.7×
	c2670	1,962	-	4,708	0.41	0.09	1.0	4.7×
	c3540	2,280	-	91,454	0.31	0.13	1.0	5.8×
	c5315	4,541	-	*TO*	-	0.17	1.0	-
	c6288	8,942	-	*TO*	-	0.36	1.0	-
ISCAS'89 [33]	s386	486	6	79	0.41	0.02	1.0	3.6×
	s349	734	15	1,879	0.44	0.03	1.0	4.8×
	s420	756	16	63	0.44	0.03	1.0	3.3×
	s510	820	6	1,905	0.32	0.04	1.0	4.7×
	s400	872	21	146	0.48	0.03	1.0	3.7×
	s526	922	21	3,344	0.45	0.03	1.0	5.0×
	s838	1,530	32	14,157	0.44	0.06	1.0	5.4×
	s953	1,824	29	33,034	0.46	0.07	1.0	5.7×
	s1488	1,860	6	89,394	0.32	0.04	1.0	6.3×
	s1196	1,908	18	20,785	0.38	0.07	1.0	5.5×
	s1423	3,282	74	*TO*	-	0.12	1.0	-
	s13207	9,725	270	*TO*	-	0.55	1.0	-
	s15850	19,480	451	*TO*	-	2.0	1.0	-
	s38417	63,340	1,564	*TO*	-	1.43	1.0	-
	s35932	69,156	1,728	*TO*	-	0.55	1.0	-
CEP [34]	WB2AXI	9,344	188	*TO*	-	0.6	1.0	-
	DES3	25,240	394	*TO*	-	3.5	1.0	-
	AXI2WB	30,496	760	*TO*	-	4.8	1.0	-
	MD5	69,400	784	*TO*	-	2.05	1.0	-
	SHA256	87,456	1,556	*TO*	-	2.85	1.0	-
	FIR	118,614	2,608	*TO*	-	3.8	1.0	-
	IIR	132,242	2,831	*TO*	-	4.65	1.0	-
	AES128	1,224,034	6,726	*TO*	-	31.0	1.0	-
	GPS	1,463,746	9,213	*TO*	-	21.0	1.0	-
	AES192	1,480,104	9,382	*TO*	-	12.0	1.0	-
	IDFT◇	1,661,567	42,819	*TO*	-	470.0	1.0	-
	DFT◇	1,663,389	42,815	*TO*	-	154.0	1.0	-
	RSA◇	2,483,773	58,034	*TO*	-	194.0	1.0	-
	RISCV◇	7,374,469	185,849	*TO*	-	934.0	1.0	-
Avg★	-	-	-	-	0.39	-	1.0	5.1×

†The runtime improvement for the proposed DCI coding scheme is represented as orders of magnitude difference.
◇These benchmark circuits have more than 10^4 unmapped flip flops and equivalence verification was done at subunit level.
★The average values are computed for the available entries. **#FFs** is the total flip flop count in circuit. Timeout(*TO*) > 1 week.

TABLE V: Memory usage of LGE algorithm using the proposed DCI coding scheme on CEP benchmarks [34].

Circuit	DES3	SHA256	FIR	IIR	AES128	AES192	IDFT	DFT	RSA	RISCV
#Transistors	25,240	87,456	118,614	132,242	1,224,034	1,480,104	1,661,567	1,663,389	2,483,773	7,374,469
GDS† (MB)	3.4	8.9	8.1	9.4	252.8	301.1	116.4	116.8	217.5	738.2
MU★ (MB)	37.1	41.5	37.6	43.3	435.5	501.8	437.4	293.7	454.5	1291.5

GDS† refers to the file size (in Mega Bytes) of the generated design layout in GDSII format.
MU★ refers to the total memory (in Mega Bytes) used by the LGE algorithm.

The impact of increased design rules in advanced process technologies on SNE runtime requires further investigation.

V. ReGDS on Obfuscated Designs

The intention of this section is to evaluate the influence of layout reverse engineering on obfuscated design layout and to examine if the hierarchical differences between the original netlist (logical) and layout RE'd netlist (physical), challenge the retrieval process. ReGDS can also be employed on obfuscated design layout to retrieve the obfuscated netlist, on which attacks like SAT [22] could be applied to recover the original netlist. The influence of IC camouflaging and Logic locking schemes on layout RE are discussed in Section V-A and Section V-B, respectively. While Split Manufacturing [26] is generally employed when the foundry is considered to be vicious, our work extends to any third-party foundry irrespective of its trustworthiness.

A. IC Camouflaging

We consider camouflaging of the original design by employing look-alike standard cell layout implementations [19],

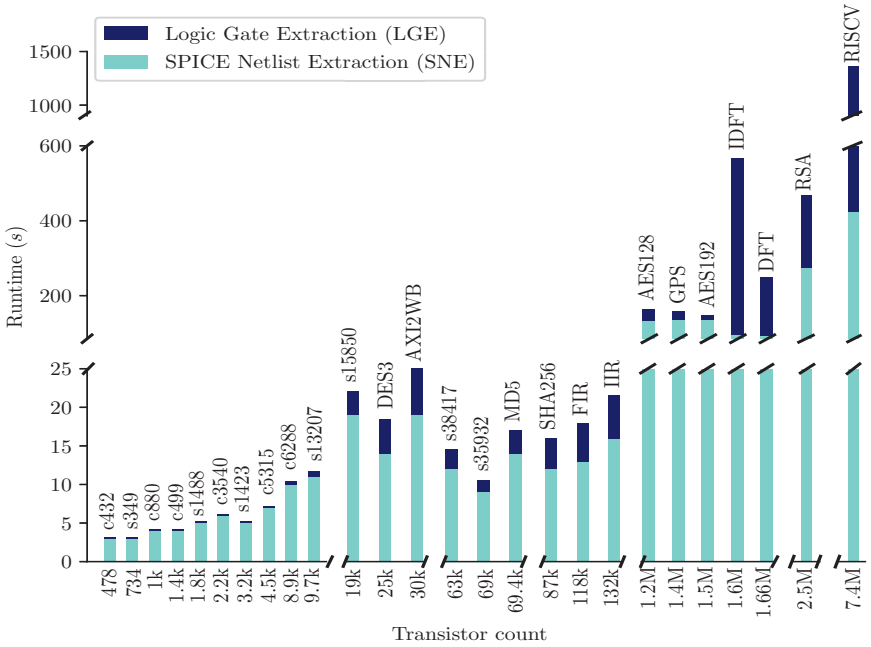

Fig. 6: ReGDS framework overall runtime.

[20]. The look-alike standard cells are appended to the technology library during physical implementation. Though imaging-based RE schemes, post-fabrication, are hampered due to the look-alike cells in the layout, the foundry has complete control on the implementation. As ReGDS considers the technology library as an input, to which the layout look-alike standard cells have been appended, the layout RE is not affected due to stand-alone camouflaging schemes.

The ReGDS graph formulation represents transistors as graph nodes with labels to differentiate their types. The formulation could be further extended to incorporate additional elements. Other camouflaging schemes that utilize dummy vias [37], varying functionalities based on device threshold voltage [38], etc., require further investigation and are interesting future directions to explore. Nevertheless, we speculate that dummy contacts or vias could be represented as resistors or capacitors.

B. Logic Locking

Logic locking aspires to lock the original circuit, with additional functionality and key inputs, rendering it non-functional without the original key [21]. To evaluate the effectiveness of the proposed ReGDS framework on locked design layouts, we consider ISCAS combinational benchmark *c*499 [32] and apply random MUX and random XOR/XNOR locking schemes [21], for different key sizes: 12, 24, 61 and 122 bits. Synthesis and physical implementation was facilitated through *Synopsys Design Compiler* [35] and *Cadence Innovus* [28] tools, respectively, to generate layouts in GDSII format. Commercial *TSMC* 40*nm* library [36] was employed for these experiments. The proposed ReGDS framework was applied to the layouts of locked designs to recover the obfuscated gate-level netlists, on which SAT attack [22] was applied to completely recover the key and thereby the original design.

The experimental results are tabulated in Table VI, where *%Tcnt* refers to the total transistor count ratio with respect to the original unlocked *c499* benchmark circuit with 1410 transistors, and *Key Recovery* refers to the extent of original key recovered. We observe an increase in deobfuscation runtime with increasing key size, especially for random XOR/XNOR locking scheme. However, the increasing key size and the number of transistors in the obfuscated design layout, only marginally increase the RE runtime. Thus gate-level obfuscation schemes do not increase the complexity of reverse engineering for the ReGDS framework.

Similarly, a third-party foundry could also reverse engineer obfuscated design layouts to obtain the obfuscated gate-level netlist, using ReGDS, on which SAT [22], approximate [23], [24] or genetic algorithm based [25] attacks could be applied to recover the original design intent.

TABLE VI: ReGDS on locked ISCAS benchmark *c*499.

Locking scheme	%Tcnt	Key size	RE Runtime (s)			Key size
			ReGDS	SAT	Total	
None	1.00	-	4.03	-	4.03	-
Random MUX	1.32	12	4.04	0.05	4.09	100%
	1.40	24	4.04	0.07	4.11	100%
	1.81	61	4.06	0.08	4.14	100%
	2.30	122	4.08	0.15	4.23	100%
Random XOR	1.32	12	4.04	0.05	4.09	100%
	1.40	24	4.04	0.07	4.11	100%
	1.73	61	4.05	6.30	10.35	100%
	2.35	122	5.08	9.00	14.08	100%

VI. Conclusion

In this paper, we present ReGDS, a layout reverse engineering framework for digital circuits, to expose the potential threat of possible IP piracy by untrusted entities within any third-party foundry. Leveraging the Layout vs Schematic comparison tool, we successfully extract transistor connections from the design layout. We develop a novel Digital Connectivity Index coding scheme to ensure unique transistor connectivity representation. Graph pattern matching is then employed to extract logic gates and completely reconstruct the original gate-level netlist. In comparison to imaging-based RE schemes, the proposed framework significantly accelerates the recovery of original netlist at the foundry. On the other hand, design houses can also evaluate the risks at the foundry. ReGDS forms a platform for further development and evaluation of stronger defense mechanisms against layout RE. In the future, we plan to explore the extension of ReGDS to mixed circuits, and investigate layout RE resilient obfuscation schemes based on reconfigurable logic.

Acknowledgments

This work was partly supported by DARPA OMG program and NSF (NSF-1812071). We thank all the anonymous reviewers for their invaluable comments that helped to further improve this work. We also acknowledge Wuxi Li, Keren Zhu and Zheng Zhao from the University of Texas at Austin for all the helpful discussions.

References

[1] T. Hoeren, "The Protection of Pioneer Innovations-Lessons Learnt from the Semiconductor Chip Industry and Its IP Law Framework," *J. Marshall J. Info. Tech. & Privacy L.*, vol. 32, p. 151, 2015.

[2] F. Koushanfar, "Provably secure active IC metering techniques for piracy avoidance and digital rights management," *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 1, pp. 51–63, 2011.

[3] N. Akkaya, B. Erbagci, and K. Mai, "Combatting IC counterfeiting using secure chip odometers," *IEEE International Electron Devices Meeting (IEDM)*, pp. 39–5, 2017.

[4] R. Torrance and D. James, "The state-of-the-art in semiconductor reverse engineering," *48th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 333–338, 2011.

[5] 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, p. 6, 2016.

[6] J. Kumagai, "Chip detectives [reverse engineering]," *IEEE Spectrum*, vol. 37, no. 11, pp. 43–48, 2000.

[7] M. Shobert, "Degate." [Online]. Available: https://degate.org

[8] A. Rankin, "Spice to verilog netlist translator and design methods using spice to verilog and verilog to spice translation," Sep. 14 2004, US Patent 6,792,579.

[9] R. E. Bryant, "Extraction of gate-level models from transistor circuits by four-valued symbolic analysis," *The Best of ICCAD*, pp. 337–346, 2003.

[10] M. Ohlrich, C. Ebeling, E. Ginting, and L. Sather, "SubGemini: identifying subcircuits using a fast subgraph isomorphism algorithm," *30th ACM/IEEE Design Automation Conference*, pp. 31–37, 1993.

[11] L. Yang and C.-J. R. Shi, "FROSTY: A fast hierarchy extractor for industrial CMOS circuits," *Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*, p. 741, 2003.

[12] G. Pelz and U. Roettcher, "Pattern matching and refinement hybrid approach to circuit comparison," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 2, pp. 264–276, 1994.

[13] N. Vijaykrishnan and N. Ranganathan, "SUBGEN: a genetic approach for subcircuit extraction," *Proceedings of 9th International Conference on VLSI Design*, pp. 343–345, 1996.

[14] "Silvaco: Spice netlist to verilog gates converter." [Online]. Available: https://www.silvaco.com/products/digital_cad/spice_netlist_verilog_gates_conversion/catalyst_ad.html

[15] "Cadence: Verilog model from SPICE netlist." [Online]. Available: https://www.eetimes.com/document.asp?doc_id=1229215

[16] T. Meade, S. Zhang, Z. Zhao, D. Pan, and Y. Jin, "Gate-level netlist reverse engineering tool set for functionality recovery and malicious logic detection," *International Symposium for Testing & Failure Analysis (ISTFA)*, 2016.

[17] J. L. White, A. S. Wojcik, M.-J. Chung, and T. E. Doom, "Candidate subcircuits for functional module identification in logic circuits," *Proceedings of the 10th Great Lakes symposium on VLSI*, pp. 34–38, 2000.

[18] S. Engels, M. Hoffmann, and C. Paar, "The End of Logic Locking? A Critical View on the Security of Logic Locking," *Cryptology ePrint Archive, Report 2019/796*, 2019, https://eprint.iacr.org/2019/796.

[19] L. W. Chow, J. P. Baukus, B. J. Wang, and R. P. Cocchi, "Camouflaging a standard cell based integrated circuit," Apr. 3 2012, uS Patent 8,151,235.

[20] X.-Y. Wang, Q. Zhou, Y.-C. Cai, and G. Qu, "Spear and Shield: Evolution of Integrated Circuit Camouflaging," *Journal of Computer Science and Technology*, vol. 33, no. 1, pp. 42–57, 2018.

[21] J. A. Roy, F. Koushanfar, and I. L. Markov, "EPIC: Ending Piracy of Integrated Circuits," *Proceedings of the conference on Design, Automation and Test in Europe (DATE)*, pp. 1069–1074, 2008.

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

[23] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "AppSAT: Approximately deobfuscating integrated circuits," *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 95–100, 2017.

[24] Y. Shen and H. Zhou, "Double DIP: Re-Evaluating Security of Logic Encryption Algorithms," *Proceedings of the on Great Lakes Symposium on VLSI*, pp. 179–184, 2017.

[25] H. Chen, C. Fu, J. Zhao, and F. Koushanfar, "GenUnlock: An Automated Genetic Algorithm Framework for Unlocking Logic Encryption," *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, 2019.

[26] R. W. Jarvis and M. G. Mcintyre, "Split manufacturing method for advanced semiconductor circuits," Mar. 27 2007, US Patent 7,195,931.

[27] "Mentor Graphics Calibre." [Online]. Available: https://www.mentor.com/products/ic_nanometer_design/verification-signoff/physical-verification

[28] "Cadence," https://www.cadence.com.

[29] L. W. Nagel, "SPICE-Simulation Program with Integrated Circuit Emphasis," *Memo No.. ERL-M382, Electronics Research Laboratory, Univ. of California, Berkeley*, 1973.

[30] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub) graph isomorphism algorithm for matching large graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 10, pp. 1367–1372, 2004.

[31] P.-H. Wu, M. P.-H. Lin, T.-C. Chen, C.-F. Yeh, X. Li, and T.-Y. Ho, "A novel analog physical synthesis methodology integrating existent design expertise," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 2, pp. 199–212, 2014.

[32] 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.

[33] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," *IEEE International Symposium on Circuits and Systems,*, vol. 3, pp. 1929–1934, May 1989.

[34] "Common Evaluation Platform." [Online]. Available: https://github.com/mit-ll/CEP

[35] "Synopsys," https://www.synopsys.com.

[36] "TSMC: 40nm Standard Cell Library." [Online]. Available: https://www.tsmc.com/english/dedicatedFoundry/technology/40nm.htm

[37] S. Patnaik, M. Ashraf, J. Knechtel, and O. Sinanoglu, "Obfuscating the interconnects: Low-cost and resilient full-chip layout camouflaging," *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 41–48, 2017.

[38] I. R. Nirmala, D. Vontela, S. Ghosh, and A. Iyengar, "A novel threshold voltage defined switch for circuit camouflaging," *21th IEEE European Test Symposium (ETS)*, pp. 1–2, 2016.