Effectiveness of Adversarial Component Recovery in Protected Netlist Circuit Designs

Jeffrey T. Mcdonald[1][a], Jennifer Parnell[1], Todd R. Andel[1] and Samuel H. Russ[2]

[1]*Department of Computer Science, University of South Alabama, Mobile, AL, U.S.A.*

[2]*Department of Electrical and Computer Engineering, University of South Alabama, Mobile, AL, U.S.A.*

Keywords: Component Identification, Obfuscation, Digital Logic Circuits, Intellectual Property Protection, Subcircuit Enumeration.

Abstract: Hardware security has become a concern as the risk of intellectual property (IP) theft, malicious alteration, and counterfeiting has increased. Malicious reverse engineering is a common tool used to achieve such goals; thus, the need arises to quantify effectiveness and limits of both circuit protection techniques and adversarial analysis tools. Aspects of physical reverse engineering are well studied and these techniques result in netlist extraction that details gate-level information from an integrated circuit (IC) artifact. Specification recovery from the netlist is a harder problem with more open research questions. In this paper, we focus on the more narrow question of how to recover design-level logic components that were used to build an IC. Such analysis assumes the library of known component building blocks can be identified and that an adversary has successfully accomplished netlist extraction. Likewise, techniques exist to harden IC's against reverse engineering through obfuscating transformations, particularly those that target component hiding. We report results of a case study analysis that compares effectiveness of component hiding algorithms against adversarial recovery approaches. As a contribution, we delineate six new approaches for subcircuit enumeration that extend a known algorithm for enumerating candidate components, seeking to improve number of potential candidates in obfuscated circuits. Our study examines algorithm performance in terms of ability to correctly identify original components and analysis time overhead. The study uses four different obfuscation approaches that target component hiding in a set of four benchmark circuits with well defined building blocks. Results indicate that all four hiding approaches are effective at increasing analysis run-time when algorithmic component identification is used, and two of the four were able to hide 95% of original components from our seven studied algorithms.

1 INTRODUCTION

Computers and electronics are complex systems made up of interconnected subsystems. Printed circuit boards (PCBs) contain a multitude of integrated circuit (IC) components with specific functionality that, in many cases, embody intellectual property (IP) from their designers. Studies in the last decade highlight that semiconductor equipment and materials companies have had adverse impacts due to IP challenges specifically (Design and Reuse, 2012). ICs are susceptible to reverse engineering by adversaries that wish to gain knowledge of functions, structure, and other embedded information in order to recover original design information. Designers typically layout circuits in a hierarchical and top-down approach, using smaller circuit building blocks (components) to build up larger functionality.

In this work we consider the power of adversarial analyzers that target component identification. We focus on algorithms that utilize a two-step process of component identification: 1) enumerating possible sub-circuit component candidates from a larger gate-level netlist and 2) semantically matching those sub-circuits against known components in a library. We reduce this adversarial question, and thus analysis of potential hiding algorithms, to a subgraph partition problem. Given a graph $G(V,E)$ with vertex set V representing digital logic gates and edge set E representing wiring between gates, we can represent the constituent components used to construct the circuit as a partition of the gate set V. The constituent component building blocks, typically smaller elements such as

[a] https://orcid.org/0000-0001-5266-7470

Mcdonald, J., Parnell, J., Andel, T. and Russ, S.
Effectiveness of Adversarial Component Recovery in Protected Netlist Circuit Designs.
DOI: 10.5220/0011275400003283
In *Proceedings of the 19th International Conference on Security and Cryptography (SECRYPT 2022)*, pages 181-192
ISBN: 978-989-758-590-6; ISSN: 2184-7711

adders, multipliers, multiplexers, decoders, etc., can be represented as a partition M of the of the gate set V. Given some semantically preserving transformation O on a circuit C such that $O(C) = C'$, we can pose the component recovery problem as whether an adversary can reproduce the original partition (M) given some semantically equivalent variant (C'). As figure 1 illustrates, a partition of a circuit $M = \{c_1, c_2, c_3\}$ represents the hierarchical composition of three building block components of the same type from a standard library or technology map in an overall circuit design. We ask whether an adversary can recover this partition (which represents the correct number, type, and connectivity of original components) given a variant C' which has the same functional semantics as C.

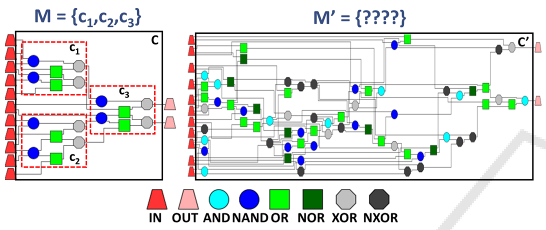


Figure 1: Adversarial Component Recovery.

From a protection viewpoint, obfuscation provides a technical means to transform circuits into forms that are harder to analyze and thus reverse engineer (Chakraborty and Bhunia, 2009; Vijayakumar et al., 2017; Zhang, 2016). An obfuscator O is a transformer that produces such semantically equivalent variants of an original circuit C such that $O(C) = C'$ and $\forall x : C(x) = C'(x)$. Component identification algorithms serve as adversarial attack vectors to such protections. In some cases, candidate enumeration algorithms which are part of the identification process assume orderly or normal connections of gates and wires that conform to standard CAD design components. Obfuscation algorithms can violate these assumptions and thus motivate the need for different approaches to delineate subcircuit candidates. We report in this paper the development of six derivative approaches to a polynomial-time algorithm for identifying subcircuit candidates first posed by Doom et al. (Doom et al., 1998) and White et al. (White et al., 2000). Our new approaches came out of the desire to increase the number of potentially viable subcircuits that might be present in obfuscated netlists, with the trade-off of increased runtime overhead. We perform a case study analysis that compares the effectiveness of these seven different subcircuit enumeration algorithms against circuit variants that are obfuscated by four different transformation algorithms. Our contributions include:

- We propose six extended approaches for subcircuit enumeration and report their effectiveness in increased candidate enumeration versus increased runtime overhead
- We evaluate effectiveness of seven total component identification approaches versus four different component hiding approaches used on gate-level netlists
- We further the body of knowledge in the area of algorithmic reverse engineering, a relatively new field of research

The rest of the paper is organized as follows: section 2 provides background and related work on hardware reverse engineering and obfuscation. Section 3 covers the transformation algorithms used in our case study that target hiding of component abstractions. Section 4 describes component identification algorithms, including six extended versions of the White algorithm (White et al., 2000). Section 5 details our experimental methodology including benchmark selection and testing environment. Section 6 provides results of the study with comparison of best component identification approaches and most effective hiding algorithms. Section 7 provides a summary and conclusions from the study as well as discussion of future work.

Benchmark circuits and a front-end interface to the software used in this case study can be found at http://soc.southalabama.edu/~mcdonald/research.htm.

2 BACKGROUND AND RELATED WORK

Reverse engineering (RE) is commonly practiced but there lacks research in quantification of its complexity: this stems from its nature as both an art and a skill with both human and automated techniques which provide context for its use. It has been used historically to gain advantage over competitors and as a means to recover lost designs for undocumented systems. The study and application of integrated circuit (IC) reverse engineering is thus a two-edged sword: it can be used for legitimate purposes such as identifying patent infringement or detecting insertion of malicious logic, and it can be used for illegal purposes such as probing a system to insert vulnerabilities or to steal IP technology (Fyrbiak et al., 2017). We take the position of the reverse engineer as an adversary with malicious purposes in this study to understand better how to protect IC against such analysis. In order to provide reasonable estimation of adversarial power, more research is needed to understand limits of automated approaches and their impact on proposed protection approaches (Azriel et al., 2021). This paper furthers that goal by posing new adversarial techniques and comparing them to known countermeasures in the form of obfuscation transformations. For our case study, we assume an adversary has physically reverse engineered an artifact (Torrance and James, 2011; Fyrbiak et al., 2017) and recovered a gate-level netlist of a target circuit under study.

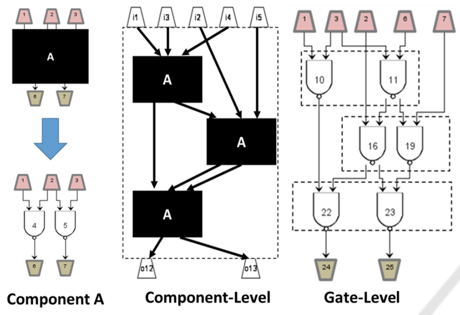


Figure 2: Design from Component Building Blocks.

Figure 2 illustrates a scenario where some component *A* from a circuit library constitutes its own input, output, and intermediate gate specification. At the component abstraction level, they represent blackboxes with specific semantics, but at the gate-level abstraction level, that information is lost once physical synthesis is conducted. Design level abstractions such as type and ordering of components are thus essential information for further recovery of abstract information for malicious reverse engineers.

Figure 3 illustrates an example of a successful adversarial reverse engineering case study by Nohl et al. (Nohl et al., 2008) on the Mifare Classic RFID tag by NPX. Their case study illustrated not only physical reverse engineering with a low budget, but also demonstrated the recovery of gate-level logic (netlist) from a silicon implementation and then recovery of component level information from the netlist. Nohl and his colleagues showed how recovery of design level components led to discovery of weaknesses in the cryptography embedded in the circuit, which led to further exploitation.

2.1 Specification Recovery

Various solutions and algorithms are posed by researchers with the goal of recovering data-paths and functionality of circuit modules in a data-path given a gate-level netlist, which builds the original specifications for a circuit design. Components are smaller

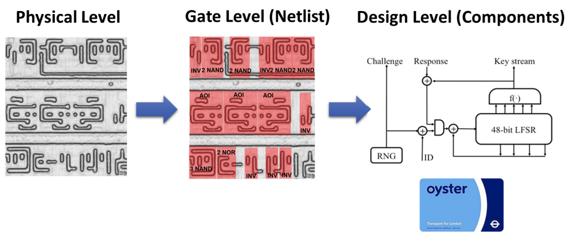


Figure 3: Design Recovery from Physical Artifact.

circuits used to build larger circuits and constitute one type of design abstraction found in gate-level netlists. We focus on these specifically in this paper and our case study. Recovery of high-level structures has been studied for some time, but has gained more attention recently because of the security threats facing the IC market worldwide. In their seminal work, Hansen et al. (Hansen et al., 1999) illustrated a human-oriented approach to identifying high-level structures in logic circuits based on looking for key patterns: 1) library modules, 2) repeated modules, 3) expected global structures, 4) computed functions, 5) control functions, 6) bus structures, and 7) common names. Gates with unknown function were classified as black-boxes. In our study, we focus only on the identification of known library modules using algorithmic methods.

Structural and Functional Analysis. (Subramanyan et al., 2013) developed a set of algorithms for analyzing unstructured netlists to recover components such as register files, counters, adders and subtracters. Their approach involved both structural and functional analysis and their benchmark set were Verilog netlists of eight large system-on-chip (SoC) designs including CPUs, routers, and micro-controllers. They also used their approach to show successful detection of trojan circuitry of no more than 50-60 gates in size. In our study, we evaluate small unobfuscated benchmark circuits and variants produced through obfuscation techniques using extended versions of the same core algorithm (Doom et al., 1998; White et al., 2000) for component identification.

A recent survey of algorithmic methods for IC reverse engineering by (Azriel et al., 2021) uses the term *specification discovery* to describe partial or full understanding of an IC's functionality. The survey summarizes four main approaches for specification discovery: 1) structural, 2) functional (behavioral), 3) data, and 4) control. In our study, we focus predominantly on behavioral identification based on semantics of known component building blocks from a circuit library or technology map. Azriel et al. also point out that circuit partitioning is typically a preprocessing steps for netlists, where circuits with millions of gates for example are separated into smaller subcircuits (modules) to allow for more efficient analysis. Such modules are then studied further with structural and functional techniques. The benchmark circuits and their obfuscated variants in our study are essentially at this level of analysis, where we focus primarily on functional analysis to recover components.

Behavioral Pattern Mining. (Li et al., 2012) proposed an approach to recover functional design blocks based on simulating traces of the gate-level netlist and then representing them as pattern graphs. These pattern graphs are then compared to pattern graphs from components in a known library. Input-output signal correspondence is cast as a subgraph isomorphism problem among the potential pattern graphs. Our algorithms of interest are different in that they require two steps: first, enumeration of subgraphs that may be viable components, and then semantically matching those subcircuits against components in a known component library. In Li et al.'s approach, they use semantic matching of I/O traces in graph form for their matching and their case study utilized publicly available, unobfuscated circuits.

Word-level Structure. (Li et al., 2013) also posed another approach that automatically derives word-level structures from gate-level netlists where an analyst can specify sequences of word-level operations for sub-circuit enumeration. Their framework generates collection of gates corresponding to world-level operations and was demonstrated on an SoC design over 400K cells and open-source components.

Control Logic Recovery. (Meade et al., 2016) proposed a methodology based on recovery of control logic represented in finite state-machines (FSM) similar to the work of Shi et al. (Shi et al., 2010). They point out that traditional structural and functional analysis may not fully reveal such control functions. Their approach treated reversing as a friendly analysis intended to discover added malicious logic; thus, there benchmark set was medium to large-scale microprocessors and cryptographic circuits with hardware trojans inserted. Our work does not address high-level control abstractions and also considers the reversing algorithm as an adversary. In separate works ((Alcaraz et al., 2013) and (Alcaraz and Wolthusen, 2014)), the general theory of graph-based network analysis for derivation of controllability and observability in the face of malicious attacks is considered. They study scale-free (random), power-law, and small-world network controllability and propose methods to preserve and restore control system functionality when adversaries target them.

Component Matching (Gascón et al., 2014) deal with the second step of our component identification algorithm of interest, which is specifically the Boolean matching problem (Cong and Minkovich, 2007). In the first step of component ID, some algorithm generates candidate structures (subcircuits) that could be matched to a known functionality. Gascón et al. give an automated approach to the second step which maps a potential candidate to a known component in a circuit library using templates. Their method overcomes many of the limitations with Permutation-Independent Equivalence Checking (PIEC) using word-level operations such as concatenation, extraction, shifting, and rotation.

2.2 Hardware Obfuscation

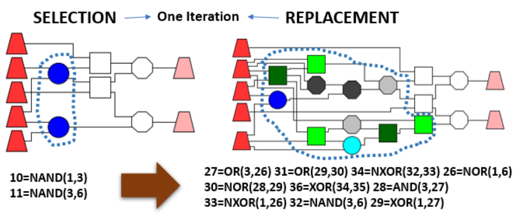
Chip to system reverse engineering is a well-studied area with a multitude of practical techniques published in the literature (Quadir et al., 2016). Popular techniques such as gate camouflaging (Cocchi et al., 2014) and physically unclonable functions (PUFs) (Bauer and Hamlet, 2014; Wendt and Potkonjak, 2014) are both methods that use alteration of CMOS or specific electronic properties to achieve anti-reverse engineering or circuit fingerprinting. In this paper, we do not address techniques designed to subvert analysis at the physical level such as those techniques surveyed by Vijayakumar et al. (Vijayakumar et al., 2017). In terms of techniques designed to thwart specification recovery, logic encryption (or locking) (Shamsi et al., 2019) has occupied the greatest amount of research focus with many proposed techniques (Aksoy et al., 2021) and attacks (Yasin et al., 2020). In a recent survey, (Shamsi et al., 2017a) summarized various adversarial models of attack which are categorized as 1) high-level recognition, 2) netlist recovery, 3) and oracle-guided attacks. The predominant research thrust has been in the use of Boolean satisfiability (SAT) and Satisfiability Modulo Theory (SMT) solvers in oracle-guided attacks to defeat both logic locking and gate camouflaging techniques, though these attacks are synonymous with learning a function with samples (Li et al., 2019; Shamsi et al., 2017b). Shamsi et al. also summarize the major categories of protection as 1) logic locking, 2) IC camouflaging, and 3) circuit diversification. Apart from early obfuscation approaches by (Chakraborty and Bhunia, 2009) that targeted transformation at the register transfer language (RTL) level and gate-level integration of finite state machine (FSM) in the netlist, little work has focused on transformation of the gate-level topology outside of the logic-locking context. We discuss next the obfuscation techniques used in our case study, which are forms of circuit diversification.

3 COMPONENT HIDING

The work presented in this paper centers on the component abstraction (subcircuits) within a gate-level netlist (a parent circuit). For our case study, we used implementations of algorithms that were developed by Norman, Parham, and Koranek (McDonald et al., 2009; McDonald et al., 2012). These algorithms vary from mostly random variation (with no hiding intent) to mostly deterministic (component information is targeted). Follow on work by McDonald et al. (McDonald et al., 2009; McDonald et al., 2011; McDonald et al., 2012) demonstrated that all four algorithms are effective against component identification (White et al., 2000). We provide explanation of each algorithm next.

3.1 Iterative Selection and Replacement

ISR was posed by (McDonald et al., 2009) as a random generator of semantically equivalent replacement logic for a given circuit. It operates by a sequence of iterations, where each iteration is composed of a selection and replacement. The selection algorithm picks (randomly or in guided fashion) a set of gates (a subcircuit) from the parent circuit. The replacement algorithm then generates a replacement set of gates (a subcircuit) with the same functional semantics (input size, output size, and truth table) as the selection subcircuit. Figure 4 illustrates the principle on a small sample circuit. In the example, a selection subcircuit consists of 2 gates which are replaced by a randomly generated, semantically equivalent version of 10 gates.


SELECTION
One Iteration
REPLACEMENT
10=NAND(1,3)
11=NAND(3,6)
27=OR(3,26) 31=OR(29,30) 34=NXOR(32,33) 26=NOR(1,6)
30=NOR(28,29) 36=XOR(34,35) 28=AND(3,27)
33=NXOR(1,26) 32=NAND(3,6) 29=XOR(1,27)
Figure 4: Example Iteration of ISR Algorithm.

The selection algorithm part of ISR can choose gates that have not been previously replaced, which attempts to guarantee that original gates are replaced at least once, or even more than once (which is referred to as a round). The number of iterations for ISR can be: 1) explicitly set (**iteration-based**), 2) limited based on a target gate size increase (**size-based**), or 3) based on a number of rounds (**round-based**). ISR can produce unlimited polymorphic variation, but does not specifically target any specific design-level abstraction for hiding–although such hiding may manifest due to the variation process itself (McDonald et al., 2009).

The replacement algorithm can maximize randomness and we use the more recent proposed method based on Random Boolean Logic Expansion (**RBLE**) (McDonald et al., 2020). With RBLE, the Boolean logic function of the selection subcircuit is expanded with random applications of Boolean logic laws performed in reverse (causing expansion versus reduction). RBLE has three policies that govern its expansion each time it is applied to a selected subcircuit. In fixed mode, only a fixed number of logic expansions n are applied. In target and strict size, the RBLE generator will apply expansions until either a strict gate size ($= n$) or target gate size ($>= n$) is reached.

3.2 Boundary Blurring

Boundary blurring was posed by (McDonald et al., 2012) and provides a deterministic means to hide component information, targeting the input/output semantics of a given subcircuit component. The principle idea is based on selecting gates strategically, at the input/output boundary of a component, and then mutating the function of the gate (randomly). After a gate is mutated, recovery logic is required to recapture the correct signal at some point lower hierarchically in the netlist (which is referred to as a recovery gate). Blur level determines where recovery terminates: one level down (**single-level blur**) or multiple levels down from the mutation (**multi-level blur**). Figure 5 shows an example of single-level blur and the selection of a specific replacement gate. In the example, the replacement gate type is mutated from NAND to NOR, then Boolean algebra is used to formulate the proper adjustment to the signal to bring it back to its expected values. This logic is then synthesized using heuristic options such as Espresso, standard canonical forms, and misII. The logic is inserted between the replacement gate and the targeted recovery gate, at which point the signal is no longer mutated.

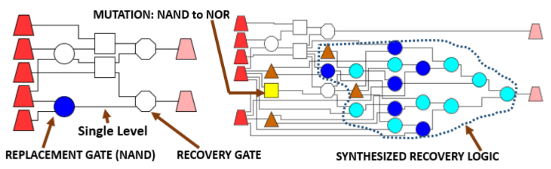


Figure 5: Example Single-Level Boundary Blur.

Additional signals can also be introduced into a component which change its overall black-box behavior, and this is referred to as a **Don't Care Blur**. While gates can be selected for mutation based on a number of different features, the most effective hiding occurs when gates are chosen based on specific locations at input and output boundaries of original components (McDonald et al., 2011).

3.3 Component Fusion

(McDonald et al., 2012) proposed an approach for hiding component information that relies on original component partitioning. As figure 6 illustrates, step 1 and 2 of the approach involves identification of gates that belong to specific subcircuit components, which come from some standard library. In step 3, the analyst must selectively pick a new gate partition so that gates on input and output boundaries of partitions are re-arranged, thus forming a new partition of the gate set. Random or predecessor-based partitions can also be used in the scheme. The algorithm then takes each component partition (subcircuit) and performs black-box synthesis using missII, single or randomly chosen canonical forms (SOPE,POSE,Reed Muller), and single or randomly chosen Espresso forms (SOP,POS). Synthesis and partition choices can introduce randomness to guarantee unique variants of an original circuit on every run of the algorithm. The synthesized versions of each component then replace existing gate logic and original connections are preserved (seen in step 4 of the example). This approach also target the input/output behavior of specific components at design time to evade behavioral analysis. Component fusion takes advantage of the fact the synthesis removes internal structural information of constituent components and also creates new I/O traces and signals within the circuit, different than those of its original components.

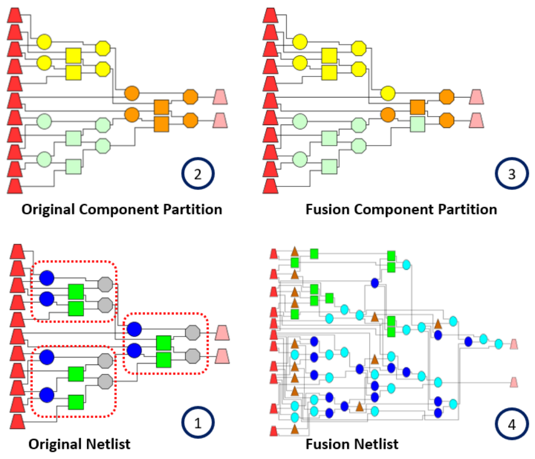


Figure 6: Example Component Fusion.

3.4 Component Encryption

Koranek (McDonald et al., 2012) also proposed a correlary approach known as component encryption. This algorithm is an adaptation of the classic white-box cryptography (WBC) approach proposed by Chow et al. (Chow et al., 2003). WBC was originally designed to allow key-embedded ciphers that were implemented as a network of encoded look-up-tables (LUTs). Component encryption also requires original component information and is optimal when it uses the component partition as input to the algorithm. Figure 7 illustrates a small example using the same starting circuit seen in Figures 1 and 6.

In the example (figure 7), step 1 and 2 represent the original gate-level topology and corresponding component partitioning of the circuit (there are 3 component subcircuits that are instances of ComponentX). The algorithm adds logic to encode and decode signals that are internal to the circuit for every potential component whose boundary is not at the parent circuit I/O boundary. These encodings use randomly generated Boolean permutation functions to change the I/O of each original component in unpredictable ways. Component outputs that are encoded must then be decoded by other components that use them for input. If $c(x)$ represents the function of some internal component, the output of the circuit would become $e(c(x))$ and a decoding function would provide $d(e(x)) = x, \forall x$. Subcircuit components that receive a signal from an encoded component must then decode such signals. The collection of circuitry for encoding outputs, decoding inputs, and the original component itself are synthesized into black-box LUTs: thus the overall circuit is transformed into a network of encoded LUTs (seen in step 3 and 4 of the example in figure 7). New components now have different semantic behavior (changed I/O sizes and changed function), thus targeting semantic-based identifiers.

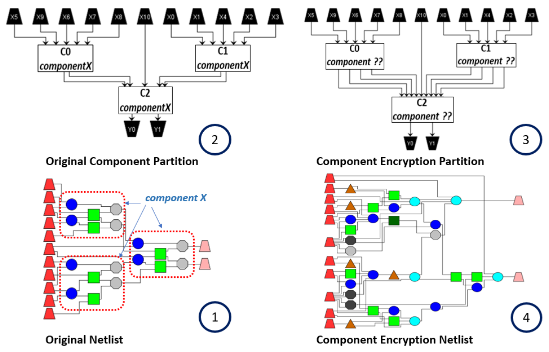


Figure 7: Example Component Encryption.

4 COMPONENT IDENTIFICATION

For our case study, we implemented a base algorithm posed by (White et al., 2000) and (Doom et al., 1998). This approach involves two steps:

- **Candidate Subcircuit Enumeration:** generates a list of all subgraphs from a target circuit that might be viable component subcircuits. These subgraphs once identified specify (arbitrary) inputs, outputs, and intermediate gates.
- **Semantic Matching:** Each candidate subcircuit is compared against known components in a provided circuit library. The matching we employ uses the ABC tool by Berkeley for semantic identification. Because inputs and outputs are specified arbitrarily, all possible enumerations of the I/O space must be considered to fully ensure no match is made. If done in a brute-force manner, an *n*-input, *m*-output circuit would have $n! * m!$ possible combinations to consider.

4.1 Basic Algorithm

(White et al., 2000) proposed an $O(n^3)$ complexity algorithm (which we call the **Basic Algorithm**) for candidate subgraph enumeration that guarantees each subgraph is only enumerated once. In the general case, subgraph enumeration is $O(2^n)$ based on connectivity, but the Basic Algorithm takes advantage of the fact that only some subgraphs are of interest as real-world building blocks. In particular, only subgraphs that exclusively contain vertices that symbolize fully specified gates are enumerated. These are the subcircuits known as feasible subgraphs. With edge set ($E(G)$) and vertex set ($V(G)$) of some graph G, the following definitions are utilized in the basic algorithm:

- **Fully Specified Vertex:** A gate that is joined within the subgraph by either all the vertices symbolizing its inputs or none of those vertices. In a subgraph H of a circuit graph G, a vertex v is a fully specified vertex if ($\forall$u — $uv \in E(G) \wedge u \in V(H)) \vee (\forall$u — $uv \in E(G) \wedge$ u $\notin V(H))$. See figure 8.
- **Subcircuit:** A subgraph H of a circuit graph G is a subcircuit of G if and only if it is connected and each vertex in H is fully specified. See figure 9.
- **Contained Vertex:** In a subgraph H of a circuit graph G, a vertex v is a contained vertex if (($\forall$u — $uv \in E(G) \wedge u \in V(H)) \vee (\forall$u — $uv \in E(G) \wedge$ u $\notin V(H)) \wedge (\forall$u — $uv \in E(G) \wedge$ u $\in V(H)) \vee (\forall$u — $uv \in E(G) \wedge u \notin V(H)$)). See figure 8.
- **Contained Subcircuit:** A subgraph H of a subcircuit graph G is a contained subcircuit of G if and only if each vertex in H is contained. See figure 9.
- **Frontier:** Frontier F of a subgraph H is all v such that $v \in N(H)$ and $v.index$ ¡ $H.index$.
- **Reachable Frontier:** The reachable frontier of a subgraph H is denoted by $F^R(H)$ and consists of all of the vertices v that may be added to H. For a subgraph $H_i = H_{(i-1)} + v_i, F^R(H_i)$ consists of all u such that: $u \in F(H_i)$ and either 1) $u \notin F(H_{(i-1)})$ or 2) $u \in F(H_{(i-1)})$ and $v \in F^R(H_{(i-1)})$ and $u.index < v_i.index$.

The first step in the Basic Algorithm assigns a unique integer index to the vertices in the graph, starting from the outputs proceeding to the inputs. Indexes are higher than the vertices that feed into it and the starting point for the enumeration can be an arbitrarily chosen vertex (typically at the output level). As the algorithm progresses, all vertices become a starting point for a sequence of unique subgraphs. In each round of the algorithm, a subgraph H begins with a single vertex and is then expanded. The neighboring vertices of subgraph H, $N(H)$, having indices less than H are considered to be within the frontier ($F(H)$) of subgraph H. The Basic Algorithm (enumerated as Algorithm 1) guarantees each subgraph is only enumerated once by assigning an index to each vertex and subgraph. Vertex indices provide a method for ordering the relation between any two vertices in a subgraph being emitted.

Algorithm 1: Basic Algorithm (White et al.).

```
1:  procedure ENUMERATE
2:      From output: assign unique integer $\forall v \in V(C)$
3:      for each $v \in V(C)$ do
4:          Create a subgraph $H$ containing $v$
5:          Determine $F(H)$ and $F^R(H)$
6:          for each vertex $u \in F^R(H)$ do
7:              $H' \leftarrow H + u$
8:              If !subcircuit($H'$): add vertices to $H'$ Else discard
9:              If !contained($H'$): add vertices to $H'$ Else discard
10:             Output $H'$
11:             Return $H'$ to Step 4
12:         end for
13:     end for
14: end procedure
```

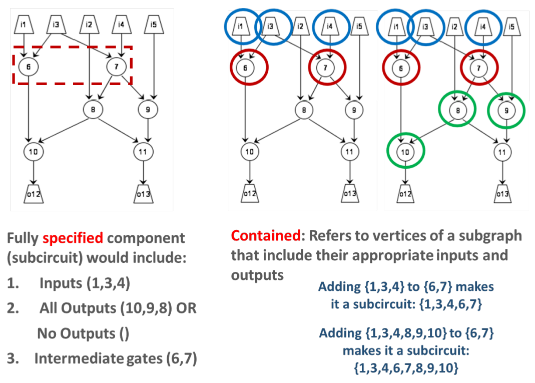

i1
i3
i2
i4
i5
6
7
8
9
10
11
o12
o13
Fully specified component (subcircuit) would include:
1. Inputs (1,3,4)
2. All Outputs (10,9,8) OR No Outputs ()
3. Intermediate gates (6,7)
Contained: Refers to vertices of a subgraph that include their appropriate inputs and outputs
Adding {1,3,4} to {6,7} makes it a subcircuit: {1,3,4,6,7}
Adding {1,3,4,8,9,10} to {6,7} makes it a subcircuit: {1,3,4,6,7,8,9,10}

Figure 8: Fully Specified and Contained Constraint.

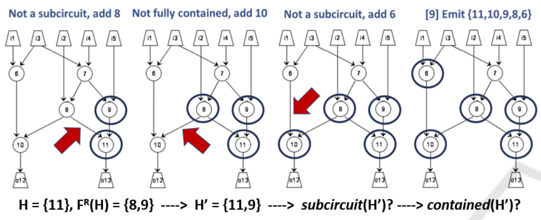

Not a subcircuit, add 8
Not fully contained, add 10
Not a subcircuit, add 6
[9] Emit {11,10,9,8,6}
H = {11}, F^R(H) = {8,9} ----> H' = {11,9} ----> subcircuit(H')? ----> contained(H')?

Figure 9: Subcircuit Constraint.

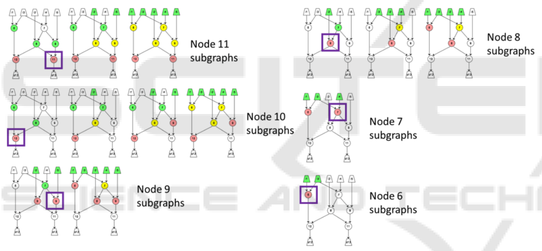


Figure 10: Example Subgraph Enumeration.

4.2 Basic Algorithm Variations

In our early experimentation, we observed that the Basic Algorithm is efficient in enumerating subcircuit candidates but limited when enumerating correct subcircuits that are created by polymorphic variation (obfuscation) algorithms. Essentially such obfuscation techniques might create circuit subgraphs which no longer have standard or expected boundaries in terms of the containment and frontier rules required (see Algorithm 1). Not all possible subgraph expansions are explored by the Basic Algorithm to start with, because it does disallow subgraphs without the right properties. This led us to explore extensions to the Basic Algorithm so that more subgraphs could be potentially enumerated, and thus potentially more candidate components identified. Our case study was designed to profile the adversarial gain in power (number of correctly identified components) versus the resultant overhead (which could potentially return to $O(2^n)$) if constraints of the algorithm were lifted. We describe six variations next that we developed next:

- **Adaptive Algorithm:** In Step 1 of the Basic Algorithm, this version starts with the inputs instead of the outputs. We wanted to see in this approach if input versus output oriented subcircuit enumeration would potentially open up different expansion opportunities.
- **Containment-Oriented Algorithm:** In step 8 of the Basic Algorithm, containment can be completely ignored by a provided parameter that is true or false. If true, this algorithm will immediately output H' (in step 9 of the Basic Algorithm). If false (containment not ignored), then this algorithm will check containment of H', and if not contained, will then allow the subgraph to be emitted if its intermediate gates are contained or if they can become contained by adding vertices. This approach was created to address non-standard connections among gates created through various obfuscation approaches.
- **Extended Algorithm:** This algorithm combines two sets of subgraphs: the first set comes from Basic Algorithm enumeration and then the second set comes from a modified form of the Basic Algorithm. In this case, the vertex chosen for inclusion to the subgraph based on the reachable frontier of H ($F^R(H)$) in Step 7 (Algorithm 1) is from a reverse ordered list of vertices that includes predecessors of vertices u in the reachable frontier.
- **Input-Bounded Algorithm:** This algorithm follows the same idea of the Extended Algorithm, except it computes both successors and predecessors of vertices u in ($F^R(H)$) when choosing the next vertex to add to the subgraph H' in Step 7 of the Basic Algorithm (Algorithm 1).
- **Combined Algorithm:** This algorithm follows the same steps of the Basic Algorithm 1-7, where expanded graphs are created by adding vertices from the reachable frontier of existing subgraphs that start with each individual vertex. The combined approach however uses four different methods to grow the subgraph: 1) it requires containment of H', but uses predecessor information similar to the Extended Algorithm; 2) it requires containment of $'H'$, but uses the Extended Algorithm to grow each subgraph; 3) it does not require containment of H', and 4) it does not require containment of H' but uses the Extended Algorithm to grow each subgraph.
- **Relaxed Algorithm, Algorithm 7:** Takes the Basic Algorithm but diverges after Step 4 and 5. The subgraph expansion instead uses a recursive

process that can be limited (called the **recursion depth**). Expansion proceeds by iterating through all elements of the reachable frontier of H, and outputs each such new subgraph (H') immediately with no constraints. The algorithm then adds vertices to the new graph (H') so that it is a subcircuit. It then computes the reachable frontier of this new subcircuit and then recursively calls expansion again on each subgraph.

5 CASE STUDY METHODOLOGY

We structure our case study around the selection of a set of transformation algorithms and a set of candidate subcircuit enumeration approaches which precede the component matching algorithm. We chose four custom benchmark circuits with small gate size that were built from a variety of components from a small circuit component library. An overview of the study is summarized in figure 11. All algorithms were implemented using Java and the use of open-source synthesis tools such as ABC and Espresso. All experiments were ran on a HP Omen laptop with 2.40 GHz i9 processor and 32 GB RAM. The case study involved taking the original four benchmark circuits and running all enumeration approaches/component identification algorithm on them. The same circuits were then obfuscated using the eight variations described below and then running all variants through the same enumeration/component ID algorithms again. The runtime overhead and identification accuracy was recorded and compared between the unobfuscated and obfuscated versions of each benchmark circuit.

Component Library: The component library which benchmark circuits were constructed from and all candidates were matched against include the following fourteen (with input/output size and acronym indicated). All circuits were specified in BENCH netlist format: half-subtractor (HS: 2/2), half-adder (HA: 2/2), 1-bit comparator (2-3COMP: 2/3), 2-bit decoder (2-4DEC: 2/4), 2-bit multiplexor (2-1MUX: 3/1), full-adder (FA: 3/2), full-subtractor (FS: 3/2), 2-bit demultiplexor (1-4DEMUX: 3/4), 3-bit decoder (3-8DEC: 3/8), priority encoder (4-2ENC: 4/2), 2-bit comparator (2-3COMP: 4/3), c17 - a conceptual component (c17: 5/2), 2-bit adder (ADDER2: 5/3), and 4-1 multiplexor/polygate (POLYGATE: 6/1).

Enumeration Algorithms: We studied all seven subcircuit candidate enumeration algorithms: Basic, Adaptive, Containment-Oriented, Extended, Input-Bounded, Combined, and Relaxed. For the Relaxed Algorithm, we used three different recursion depths (level 1, 2, and 3). All other algorithms had default options applied, thus totaling 10 enumeration types. Component identification used a brute-force approach of mutating all possible input/output combinations and using ABC to compare various versions of candidate components against the known components in the circuit library.

Transformation Algorithms: We studied the four transformation algorithms detailed in Section 3. Each algorithm could generate a unique variant, even given the same options, but we chose to only generate one variant for each approach. We also chose two different option sets for each of the four transformation algorithms. Details for each algorithm are as follows:

- **ISR:** We chose for standard options to use a iteration-based strategy with the algorithm running 50 iterations total. The selection size was either 10 gates or 15 gates (seen as RBLE10 and RBLE15 in figure 11).
- **Boundary Blurring:** We exercised this algorithm in Level One and Don't Care mode, choosing to pick randomly half the amount of total gates in the original circuit for blurring. We used canonical Quine-McCluskey reduction forms for synthesis.
- **Component Fusion:** We chose two different types of synthesis for component fusion: canonical and Espresso. The component configuration used for the algorithm was manually entered and involved taking the original configuration based on the design of each circuit and either eliding or extending the internal boundaries.
- **Component Encryption:** We chose two different types of synthesis for component encryption: canonical and Espresso. The component configuration used for the algorithm was manually entered and involved taking the original configuration based on the design of each circuit and either eliding or extending the internal boundaries.

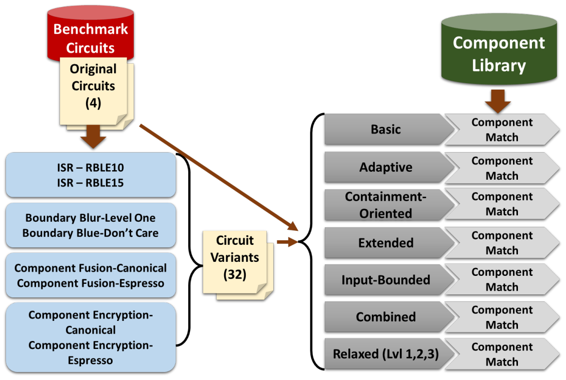


Figure 11: Case Study Overview.

Benchmark Circuits: The four benchmark circuits are summarized in table 1. In general, we chose circuits with less than 100 gates to make enumeration times reasonable since these circuits are obfuscated further and we wanted to characterize run-time overhead versus evaluate feasibility on large-scale circuits. We created the benchmarks based on various criteria. The **4-bit multiplier** uses NAND-only full-adder components and is a smaller design version of the large c6288 16-bit multiplier circuit: it represents a circuit with almost homogeneous design components. The multicomp circuit was created from six unique components in the library to represent potential diversity. The c17-polygate is the standard c17 circuit with 6 NAND gates, where every gate is replace with a polygate component, which are variants of basic multiplexors. In initial testing, different versions of the polygate component introduced unique gate and wiring configurations that were not enumerated by the Basic Algorithm.

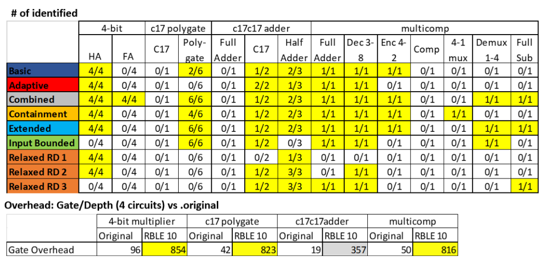
Table 1: Benchmark Summary.

	4-bit multiplier	multicomp	c17-polygate	c17c17adder
Size	96	50	42	19
Depth	2	3	3	2
HA	4	0	0	0
FA	8	1	0	1
2-1MUX	0	1	0	0
4-1MUX	0	0	0	0
c17	0	0	1	2
3-8DEC	0	1	0	0
4-2ENC	0	1	0	0
2-3COMP	0	1	0	0
1-4DEMUX	0	1	0	0
POLYGATE	0	0	6	0
Total	12	6	7	3

6 RESULTS

We performed our case study and recorded results for all benchmark circuits before and after the 8 variations of the 4 transformation algorithms were applied. We report first results of the best of each of the 4 transformation approaches in terms of hiding effectiveness and overhead.

ISR-RBLE10 results are summarized in figure 12 and show that it was able to hide 145/252 possible components across all enumeration algorithms. Our recorded runtime overhead of the identification algorithm variations compared from original vs. obfuscated variants was on average 84,463% higher. Overhead in terms of increased gate size was on average 15.9 times higher over the original. ISR-RBLE15 had similar effectiveness in hiding, but came at an average overhead of 22.4 times original gate size.



# of identified

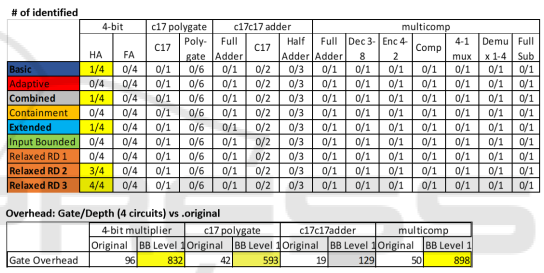
	4-bit		c17 polygate		c17c17 adder			multicomp						
	HA	FA	C17	Poly-gate	Full Adder	C17	Half Adder	Full Adder	Dec 3-8	Enc 4-2	Comp	4-1 mux	Demux 1-4	Full Sub
Basic	4/4	0/4	0/1	2/6	0/1	1/2	2/3	1/1	1/1	1/1	0/1	0/1	0/1	0/1
Adaptive	4/4	0/4	0/1	0/6	0/1	2/2	1/3	1/1	1/1	0/1	0/1	0/1	0/1	0/1
Combined	4/4	4/4	0/1	6/6	0/1	1/2	2/3	1/1	1/1	1/1	0/1	0/1	1/1	1/1
Containment	4/4	0/4	0/1	4/6	0/1	1/2	2/3	1/1	1/1	1/1	0/1	1/1	0/1	0/1
Extended	4/4	0/4	0/1	6/6	0/1	1/2	2/3	1/1	1/1	1/1	0/1	0/1	1/1	1/1
Input Bounded	0/4	0/4	0/1	6/6	0/1	1/2	0/3	1/1	1/1	0/1	0/1	0/1	1/1	0/1
Relaxed RD 1	4/4	0/4	0/1	0/6	0/1	0/2	1/3	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Relaxed RD 2	4/4	0/4	0/1	0/6	0/1	1/2	3/3	0/1	1/1	0/1	0/1	0/1	0/1	0/1
Relaxed RD 3	0/4	0/4	0/1	0/6	0/1	1/2	3/3	1/1	1/1	0/1	0/1	0/1	0/1	1/1

Overhead: Gate/Depth (4 circuits) vs .original

	4-bit multiplier		c17 polygate		c17c17adder		multicomp	
	Original	RBLE 10	Original	RBLE 10	Original	RBLE 10	Original	RBLE 10
Gate Overhead	96	854	42	823	19	357	50	816

Figure 12: Summary - ISR-RBLE10.

Boundary Blur-Single Level results are summarized in figure 13 and show that it was able to hide 242/252 possible components across all enumeration algorithms. Our recorded runtime overhead of the identification algorithm variations compared from original vs. obfuscated variants was on average 23,869% higher. Overhead in terms of increased gate size was on average 11.88 times higher over the original.



# of identified

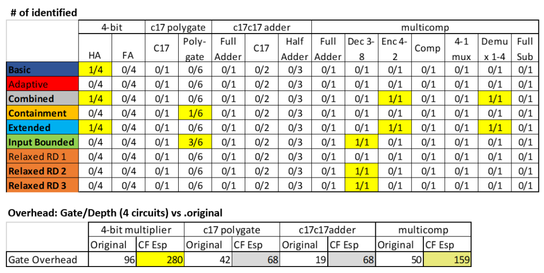
	4-bit		c17 polygate		c17c17 adder			multicomp						
	HA	FA	C17	Poly-gate	Full Adder	C17	Half Adder	Full Adder	Dec 3-8	Enc 4-2	Comp	4-1 mux	Demux 1-4	Full Sub
Basic	1/4	0/4	0/1	0/6	0/1	0/2	0/3	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Adaptive	0/4	0/4	0/1	0/6	0/1	0/2	0/3	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Combined	1/4	0/4	0/1	0/6	0/1	0/2	0/3	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Containment	0/4	0/4	0/1	0/6	0/1	0/2	0/3	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Extended	1/4	0/4	0/1	0/6	0/1	0/2	0/3	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Input Bounded	0/4	0/4	0/1	0/6	0/1	0/2	0/3	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Relaxed RD 1	0/4	0/4	0/1	0/6	0/1	0/2	0/3	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Relaxed RD 2	3/4	0/4	0/1	0/6	0/1	0/2	0/3	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Relaxed RD 3	4/4	0/4	0/1	0/6	0/1	0/2	0/3	0/1	0/1	0/1	0/1	0/1	0/1	0/1

Overhead: Gate/Depth (4 circuits) vs .original

	4-bit multiplier		c17 polygate		c17c17adder		multicomp	
	Original	BB Level 1	Original	BB Level 1	Original	BB Level 1	Original	BB Level 1
Gate Overhead	96	832	42	593	19	129	50	898

Figure 13: Summary - Boundary Blur-Level One.

Component Fusion-Espresso results are summarized in figure 14 and show that it was able to hide 238/252 possible components across all enumeration algorithms. Our recorded runtime overhead of the identification algorithm variations compared from original vs. obfuscated variants was on average 1,086% higher. Overhead in terms of increased gate size was on average 16.7 times higher over the original.



# of identified

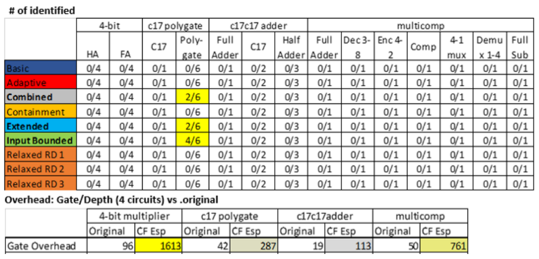
	4-bit		c17 polygate		c17c17 adder			multicomp						
	HA	FA	C17	Poly-gate	Full Adder	C17	Half Adder	Full Adder	Dec 3-8	Enc 4-2	Comp	4-1 mux	Demux 1-4	Full Sub
Basic	1/4	0/4	0/1	0/6	0/1	0/2	0/3	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Adaptive	0/4	0/4	0/1	0/6	0/1	0/2	0/3	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Combined	1/4	0/4	0/1	0/6	0/1	0/2	0/3	0/1	0/1	1/1	0/1	0/1	1/1	0/1
Containment	0/4	0/4	0/1	1/6	0/1	0/2	0/3	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Extended	1/4	0/4	0/1	0/6	0/1	0/2	0/3	0/1	0/1	1/1	0/1	0/1	1/1	0/1
Input Bounded	0/4	0/4	0/1	3/6	0/1	0/2	0/3	0/1	1/1	0/1	0/1	0/1	0/1	0/1
Relaxed RD 1	0/4	0/4	0/1	0/6	0/1	0/2	0/3	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Relaxed RD 2	0/4	0/4	0/1	0/6	0/1	0/2	0/3	0/1	1/1	0/1	0/1	0/1	0/1	0/1
Relaxed RD 3	0/4	0/4	0/1	0/6	0/1	0/2	0/3	0/1	1/1	0/1	0/1	0/1	0/1	0/1

Overhead: Gate/Depth (4 circuits) vs .original

	4-bit multiplier		c17 polygate		c17c17adder		multicomp	
	Original	CF Esp	Original	CF Esp	Original	CF Esp	Original	CF Esp
Gate Overhead	96	280	42	68	19	68	50	159

Figure 14: Summary - Component Fusion-Espresso.

Component Encryption-Espresso results are summarized in figure 15 and show that it was able to hide 244/252 possible components across all enumeration algorithms. Our recorded runtime overhead of the identification algorithm variations compared from original vs. obfuscated variants was on average 350% higher. Overhead in terms of increased gate size was on average 11.2 times higher over the original.



# of identified

	4-bit		c17 polygate		c17c17 adder		multicomp							
	HA	FA	C17	Poly-gate	Full Adder	C17	Half Adder	Full Adder	Dec 3-8	Enc 4-2	Comp	4-1 mux	Demu x 1-4	Full Sub
Basic	0/4	0/4	0/1	0/6	0/1	0/2	0/3	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Adaptive	0/4	0/4	0/1	0/6	0/1	0/2	0/3	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Combined	0/4	0/4	0/1	2/6	0/1	0/2	0/3	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Containment	0/4	0/4	0/1	0/6	0/1	0/2	0/3	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Extended	0/4	0/4	0/1	2/6	0/1	0/2	0/3	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Input Bounded	0/4	0/4	0/1	4/6	0/1	0/2	0/3	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Relaxed RD 1	0/4	0/4	0/1	0/6	0/1	0/2	0/3	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Relaxed RD 2	0/4	0/4	0/1	0/6	0/1	0/2	0/3	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Relaxed RD 3	0/4	0/4	0/1	0/6	0/1	0/2	0/3	0/1	0/1	0/1	0/1	0/1	0/1	0/1

Overhead: Gate/Depth (4 circuits) vs .original

	4-bit multiplier		c17 polygate		c17c17adder		multicomp	
	Original	CF Esp	Original	CF Esp	Original	CF Esp	Original	CF Esp
Gate Overhead	96	1613	42	287	19	113	50	761

Figure 15: Summary - Component Encryption-Espresso.

In terms of transformation algorithms, all algorithms hide some number of components and all algorithms induced, on average, increased analyzer runtimes for all enumeration types. These two factors typically are used to categorize a transformer as effective against a particular type of analysis in respect to traditional Man-at-the-End (MATE) attacks. Figure 16 summarizes effectiveness of all transformation algorithms and their hiding effectiveness against all identification algorithms.

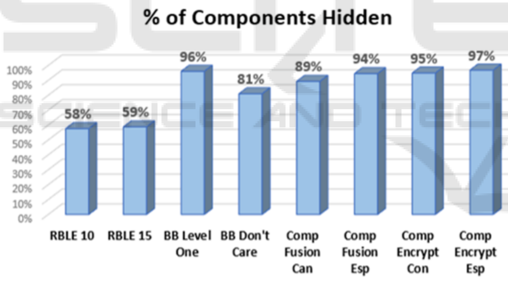


Figure 16: Transformation Algorithm Effectiveness.

In terms of whether the six new extensions to the Basic Algorithm for candidate subcircuit enumeration resulted in higher detection accuracy, figure 17 summarizes the overall identification accuracy by approach used. The Combined and Extended algorithm had the highest overall impact on identification in the end. Compared to the expected increase in number of subgraphs enumerated, figure 18 shows results of a study of 5 circuits that are part of the ISCAS-85 benchmark set. While the Relaxed algorithm has the greatest power to incorporate potential subcircuit components with recursion depth, some algorithms surprisingly show that they return less subgraphs than the Basic Algorithm to begin with. Algorithms such as Combined and Extended for example that return on average a greater # of subgraphs than the Basic Algorithm, also appear to better identify subcircuits even after the four transformation algorithms we studied were used. Based on the component study, some algorithms under-performed the Basic Algorithm: Adaptive and Relaxed with recursion depth 1 and 2. Again, this corresponds roughly to the fact that these algorithms enumerate less subgraphs than expected, as seen in figure 18.

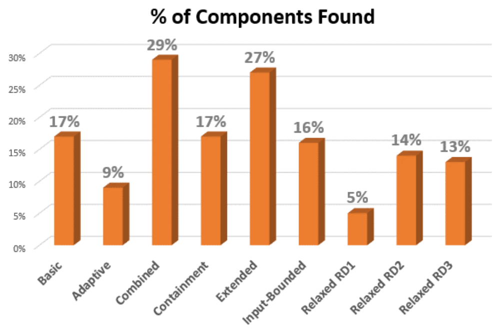
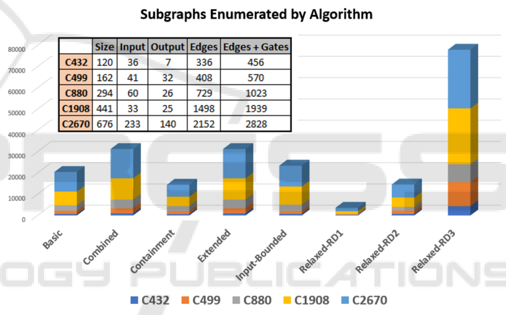


Figure 17: Enumeration Algorithm Effectiveness.



	Size	Input	Output	Edges	Edges + Gates
C432	120	36	7	336	456
C499	162	41	32	408	570
C880	294	60	26	729	1023
C1908	441	33	25	1498	1939
C2670	676	233	140	2152	2828

Figure 18: Subgraph Enumeration by Algorithm.

7 CONCLUSIONS

All component hiding algorithms we studied were considered effective: they concealed one or more components and on average increased time for identification. The most effective of our six new possible extensions to the Basic Algorithm at identifying components were the Combined and Extended approach. Our study showed that for the analyzers tested, the Component Encryption-Espresso and Boundary Blurring-Level One obfuscation algorithm concealed all components for 3 out 4 circuits and had the highest overall hiding rate. Our future work will consider large-scale studies of industry scale circuits and component libraries with knowledge gained from this study.

REFERENCES

Aksoy, L., Nguyen, Q., Almeida, F., Raik, J., Flottes, M., Dupuis, S., and Pagliarini, S. (2021). High-level Intellectual Property Obfuscation via Decoy Constants. In *IOLTS 2021*, pages 1–7, Torino, Italy. IEEE.

Alcaraz, C., Miciolino, E. E., and Wolthusen, S. (2013). Structural controllability of networks for non-interactive adversarial vertex removal. In *Critical Information Infrastructures Security*, pages 120–132. Springer.

Alcaraz, C. and Wolthusen, S. (2014). Recovery of structural controllability for control systems. In *Critical Infrastructure Protection VIII*, pages 47–63. Springer.

Azriel, L., Speith, J., Albartus, N., Ginosar, R., Mendelson, A., and Paar, C. (2021). A survey of algorithmic methods in ic reverse engineering. *Jour. of Crypto. Eng.*, 11(3):299–315.

Bauer, T. and Hamlet, J. (2014). Physical unclonable functions: A primer. *IEEE Security Privacy*, 12(6):97–101.

Chakraborty, R. and Bhunia, S. (2009). Harpoon: an obfuscation based soc design methodology for hardware protection. *IEEE Trans. CADIC Syst.*, 28(10):1493–1502.

Chow, S., Eisen, P. A., Johnson, H., and van Oorschot, P. C. (2003). White-box cryptography and an aes implementation. In *SAC '02*, pages 250–270. Springer.

Cocchi, R., Baukus, J., Chow, L., and Wang, B. (2014). Circuit camouflage integration for hardware ip protection. In *DAC'14*, pages 1–5.

Cong, J. and Minkovich, K. (2007). Improved sat-based boolean matching using implicants for lut-based fpgas. In *ISFPGA '07*, page 139–147.

Design and Reuse (2012). Innovation at risk: Ip infringement challenges the semiconductor equipment industry.

Doom, T. E., White, J. L., Wojcik, A. S., and Chisholm, G. (1998). Identifying high-level components in combinational circuits. In *8th GLS-VLSI*, pages 313–318.

Fyrbiak, M., Strauß, S., Kison, C., Wallat, S., Elson, M., Rummel, N., and Paar, C. (2017). Hardware reverse engineering: Overview and open challenges. In *IVSW'19*. IEEE Computer Society.

Gascón, A., Subramanyan, P., Dutertre, B., Tiwari, A., Jovanović, D., and Malik, S. (2014). Template-based circuit understanding. In *FMCAD'14*, page 83–90.

Hansen, M., Yalcin, H., and Hayes, J. (1999). Unveiling the iscas-85 benchmarks: a case study in reverse engineering. *Design & Test of Computers, IEEE*, 16(3):72–80.

Li, M., Shamsi, K., Meade, T., Zhao, Z., Yu, B., Jin, Y., and Pan, D. Z. (2019). Provably secure camouflaging strategy for ic protection. *IEEE Trans. on Comp.-Aided Des. of Integ. Circ. and Sys.*, 38(8):1399–1412.

Li, W., Gascon, A., Subramanyan, P., et al. (2013). Wordrev: Finding word-level structures in a sea of bit-level gates. In *HOST'13*, pages 67–74.

Li, W., Wasson, Z., and Seshia, S. A. (2012). Reverse engineering circuits using behavioral pattern mining. In *HOST'12*, pages 83–88. IEEE Computer Society.

McDonald, J., Kim, Y., and Grimaila, M. (2009). Protecting reprogrammable hardware with polymorphic circuit variation. In *CSRW '09*.

McDonald, J., Kim, Y., and Koranek, D. (2011). Deterministic circuit variation for anti-tamper applications. In *CSIIRW '11*.

McDonald, J., Kim, Y., Koranek, D., and Parham, J. (2012). Evaluating component hiding techniques in circuit topologies. In *ICC'12*, pages 1138–1143.

McDonald, J., Stroud, T., and Andel, T. (2020). Polymorphic circuit generation using random boolean logic expansion. In *SAC'20*.

Meade, T., Zhang, S., and Jin, Y. (2016). Netlist reverse engineering for high-level functionality reconstruction. In *ASP-DAC'16*, pages 655–660.

Nohl, K., Evans, D., Starbug, S., and Plötz, H. (2008). Reverse-engineering a cryptographic rfid tag. In *USENIX'08*, page 185–193, USA. USENIX Association.

Quadir, S. E., Chen, J., Forte, D., et al. (2016). A survey on chip to system reverse engineering. *J. Emerg. Technol. Comput. Syst.*, 13(1).

Shamsi, K., Li, M., Meade, T., et al. (2017a). Circuit obfuscation and oracle-guided attacks: Who can prevail? In *Proc. of GLS-VLSI 2017*, page 357–362.

Shamsi, K., Li, M., Meade, T., Zhao, Z., Pan, D. Z., and Jin, Y. (2017b). Appsat: Approximately deobfuscating integrated circuits. In *HOST'17*, pages 95–100.

Shamsi, K., Li, M., Plaks, K., et al. (2019). Ip protection and supply chain security through logic obfuscation: A systematic overview. *ACM Trans. Des. Autom. Electron. Syst.*, 24(6).

Shi, Y., Ting, C., Gwee, B., and Ren, Y. (2010). A highly efficient method for extracting fsms from flattened gate-level netlist. In *ISCAS'10*, pages 2610–2613.

Subramanyan, P., Tsiskaridze, N., Pasricha, K., Reisman, D., Susnea, A., and Malik, S. (2013). Reverse engineering digital circuits using functional analysis. In *DATE '13*, page 1277–1280.

Torrance, R. and James, D. (2011). The state-of-the-art in semiconductor reverse engineering. In *DAC'11*, pages 333–338.

Vijayakumar, A., Patil, V. C., Holcomb, D. E., Paar, C., and Kundu, S. (2017). Physical design obfuscation of hardware: A comprehensive investigation of device and logic-level techniques. *IEEE Trans. on Info. For. and Sec.*, 12(1):64–77.

Wendt, J. B. and Potkonjak, M. (2014). Hardware obfuscation using puf-based logic. In *ICCAD'14*, ICCAD '14, page 270–277. IEEE Press.

White, J. L., Wojcik, A. S., Chung, M., and Doom, T. E. (2000). Candidate subcircuits for functional module identification in logic circuits. In *Proc. of 10th GLS-VLSI*, page 34–38.

Yasin, M., Mazumdar, B., Sinanoglu, O., and Rajendran, J. (2020). Removal attacks on logic locking and camouflaging techniques. *IEEE Trans. Emerg. Topics Comp.*, 8(2):517–532.

Zhang, J. (2016). A practical logic obfuscation technique for hardware security. *IEEE Trans. Very Large Scale Integr. Syst.*, 24(3):1193–1197.