Incremental SAT-Based Reverse Engineering of Camouflaged Logic Circuits

Cunxi Yu, *Student Member, IEEE*, Xiangyu Zhang, Duo Liu, Maciej Ciesielski, *Senior Member, IEEE*, and Daniel Holcomb, *Member, IEEE*

***Abstract*—Layout-level gate or routing camouflaging techniques have attracted interest as countermeasures against reverse engineering of combinational logic. In order to minimize area overhead, typically only a subset of gate or routing components are camouflaged, and each camouflaged component layout can implement one of a few different functions or connections. The security of camouflaging relies on the difficulty of learning the overall combinational logic function without knowing the functions implemented by the individual camouflaged components of the circuit. In this paper, we expand our previous work on using incremental SAT solving to reconstruct the logical function of a circuit with camouflaged components. Our algorithm uses the standard attacker model in which an adversary knows only the noncamouflaged component functions, and has the ability to query the circuit to learn the correct output vector for any input vector. Our results demonstrate a 10.5× speedup in average runtime over the best known existing deobfuscation algorithm prior to this technique. The results presented go beyond our previous work by showing that this technique, previously applied only to a particular style of gate camouflaging, is general and can be used to deobfuscate three different proposed styles of camouflaging. We give results to quantify the effectiveness of camouflaging techniques on a variety of ISCAS-85 benchmark circuits.**

***Index Terms*—Hardware security, incremental-SAT, logic de-camouflaging.**

I. Introduction

IC DESIGNERS have clear incentives against publicizing all implementation details of a design, as this may compromise their strategic advantage or leak sensitive information. However, once a circuit is fabricated and released to market, reverse engineering techniques can attempt to extract implementation details from the physical object without consent or knowledge of the designer. Circuit camouflaging is an attempt to obscure the true functionality of a circuit, and to limit the information that can be leaked through reverse engineering.

Gate-level camouflaging is a particular camouflaging technique in which the functions of certain combinational logic gates cannot be directly ascertained from imaging-based reverse engineering. In this case, the logic may be inferred using a combination of information obtained from reverse engineering and information obtained through observation of input–output vectors captured through scan chains or other mechanisms. In this paper, we present such an algorithm for extracting the functionality of reverse engineered netlists.

The specific contributions of this paper are as follows.

1) We present an incremental-SAT-based technique for reverse engineering camouflaged integrated circuits (ICs) that outperforms its nonincremental counterpart by 10.5× in terms of average runtime on ISCAS-85 benchmarks.
2) We show that selective gate camouflaging based on the objective of maximizing output corruption [2] offers no resistance to reverse engineering and can reduce the number of vectors required to deobfuscate a circuit.
3) We provide to the research community a new standard and widely applicable tool for logic deobfuscation that can be used to evaluate current and future approaches for selective camouflaging.[1]
4) We demonstrate that our technique is general and can efficiently resolve the obfuscated function of three proposed camouflaging techniques [2]–[4].

II. Related Work

Invasive techniques can be used to reverse engineer gate-level circuit functions. Invasive reverse engineering works by decapsulating the chip and imaging and removing each layer in succession to reveal the layers below for imaging. In recent years, reverse engineering of IC chips has become increasingly successful [5]. Among other applications, invasive reverse engineering is used for competitive analysis in the IC industry, and has famously been used by Nohl *et al.* [6] to identify cryptographic weaknesses in the Mifare Classic RFID tag. An overview of the state-of-the-art in invasive reverse engineering is given by Torrance and James [5].

Manuscript received April 15, 2016; revised August 1, 2016 and October 30, 2016; accepted January 3, 2017. Date of publication January 11, 2017; date of current version September 14, 2017. This work was supported in part by the National Science Foundation under Award CCF-1319496, Award CCF-1617708, and Award CNS-1563829, and in part by the University of Massachusetts, Amherst. This paper was recommended by Associate Editor Q. Xu.

C. Yu, X. Zhang, M. Ciesielski, and D. Holcomb are with the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003 USA (e-mail: ycunxi@umass.edu; xiangyuzhang@umass.edu; ciesiel@ecs.umass.edu; holcomb@engin.umass.edu).

D. Liu is with Intel Corporation, Hudson, MA 01749 USA.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2017.2652220

[1]The source code of our tool and benchmarks used in this journal paper are released publicly at our project website at https://ycunxi.github.io/Incremental-SAT-DeCam/.

Significant research effort is spent on extracting high-level meaning from the sea of gates obtained by invasive reverse engineering of fabricated circuits. Work by Li *et al.* [7] resolved subcircuit components by matching against a set of known components, and Subramanyan *et al.* [8] improved on this to operate on unstructured netlists, where subcircuits are not identified in advance. Further work by Li *et al.* [7] extracted word-level structures from unstructured netlists where all gates are known. Work by Gascón *et al.* [9] checked equivalence between a reference circuit and a circuit-under-investigation when the signal correspondence between the two is unknown. A primary difference between this paper and these previous reverse engineering works is that our attacker model assumes there is no complete gate-level model of the circuit-under-investigation.

A countermeasure against imaging-based reverse engineering is the use of camouflaged gates. Camouflaged gates are ones in which different logic functions can be implemented by same-looking cells to prevent the cell functions from being correctly inferred from their appearance. Camouflaged gate libraries [10] can use hard-to-observe structural techniques to create different gate functions [11], or functionality can be controlled without structural differences by changing doping of specific devices [3], [12]–[14]. A variant of dopant-programmable cells is to build components in a dual-Vt process technology such that inferring the correct component functions would require identification of which devices use high and low thresholds [15]. To minimize cost, it is often desirable to protect a circuit by camouflaging only a small subset of the gates [2], [16]. However, in emerging technologies, it can be more difficult to infer function from structure [17], and a reverse engineer may thus need to consider all gates as camouflaged. An overview of physical mechanisms for obfuscation is given by Vijayakumar *et al.* [18].

An attacker model for reverse engineering circuits with camouflaged gates is given by Rajendran *et al.* [19]. The logic function implemented by a camouflaged circuit should remain hard to discover when the attacker has knowledge of all noncamouflaged gates, and has the ability to apply inputs to the circuit and observe outputs. Techniques from oracle-guided synthesis [20] have recently been used in SAT-based attacks to reverse engineer gate camouflaging [21] and logic encryption [22]. The method in this paper uses the same approach. It is important to note the capabilities and limitations of oracle-guided synthesis in the circuit obfuscation setting. While the overall circuit produced by oracle-guided synthesis is guaranteed to be functionally equivalent to the obfuscated circuit, there is no guarantee it will match the obfuscated circuit on a gate-by-gate basis. For example, given that 2-input XOR and 2-input NAND gates produce different outputs only for the 00 input combination, the two gate functions are interchangeable if the 00 input combination cannot be justified or propagated to the outputs. This gate-level ambiguity is unavoidable if trying to synthesize a design-based only on inputs and outputs, but it is important to note that a design recovered through oracle-guided synthesis should not be used for certain classes of side channel attacks or fault injection attacks that require knowing the states of all combinational circuit nets.

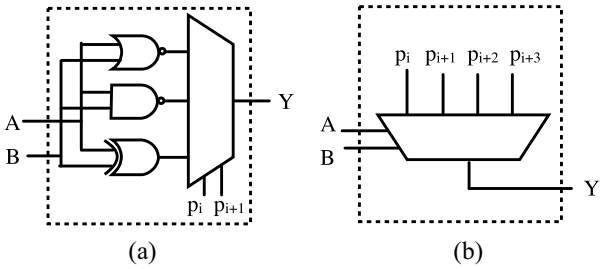


Fig. 1. Circuit constructs used to model camouflaged standard cells. Boolean variables p_i configure the logic function of the gate, and an attacker tries to learn the value of these variables. (a) NAND/NOR/XOR camouflage. (b) Fully camouflaged.

III. Circuit Camouflaging and Attacker Model

The attacker model we consider is that of an adversary possessing instances of a chip and wanting to reconstruct the logic function of some obfuscated functional block on the chip for IP theft or other purposes. Consistent with prior works in this area, including that of Rajendran *et al.* [2], the capabilities of the attacker are assumed to be as follows.

1) The attacker has knowledge of the logic functions and routing of all nonobfuscated components in the circuit. The information can be obtained from imaging the layers of the delayered chip.
2) The attacker has knowledge of which components are camouflaged and knows the possible functions they can each implement. This information can be learned from imaging because the camouflaged components do not generally look identical to their noncamouflaged counterparts. For example, a camouflaged gate style that can implement a number of different possible functions will be larger than the function-specific implementations of the same gates.
3) The attacker has the ability to apply input vectors to the camouflaged combinational circuit and observe its corresponding outputs. We denote the third capability as oracle access, and this can be achieved using scan-based test techniques [23], possibly combined with microprobing to gain access to deactivated scan chains [24].

As demonstrated by El Massad *et al.* [21], uncertainty about the logical functions of camouflaged components can be translated to uncertainty about the value of certain Boolean variables. In this paper, we model all the configurations of the camouflaged circuit using multiplexers with uncertain control bits. We denote these control bits as programming bits, and in Section IV will show how to solve for the value of these variables using SAT. The resolved values of programming bits indicate the true logic function of the camouflaged circuit. In this section, we show how this modeling mechanism can be applied to three different types of camouflaged components: 1) *camouflaged standard cells*; 2) *Obfusgates*; and 3) *transformable interconnects*.

A. Camouflaged Standard Cells Using Dummy Contacts

The first technique we consider is the use of camouflaged standard cells. The idea of standard cell camouflaging is to

implement multiple logic functions using a generic cell layout, such that the reverse engineer cannot infer the logical function of the gate from its structure. One way of camouflaging standard cells is to use *dummy contacts* [25]. Dummy contacts are structures that look like connections between metal layers, or connections between metal layers and polysilicon, but in reality they make no such connection. Depending on which contacts in a cell are true contacts or dummies, a same-looking cell can implement different logic functions. Therefore, under the assumption that the reverse engineer cannot distinguish true contacts from dummy contacts, the attacker will not know which logic function is implemented by the cell.

A specific variant of a camouflaged standard cell is a generic cell layout that can realize the functionality of 2-input XOR, NAND, or NOR gates depending on which contacts are true [2]. To model the camouflaged cell, we introduce a multiplexer-based model (Fig. 1). The functionality can be derived by resolving the values of program bits p_i, p_{i+1} [Fig. 1(a)]. Note that in this model p_i, p_{i+1} are forbidden to be "11" since there are only three possible functions. Additionally, we extend the model [Fig. 1(b)] to gain the capability of modeling other types of camouflaged standard cells. The function of Y can be represented as $Y = \bar{A}\bar{B}p_1 + \bar{A}Bp_2 + A\bar{B}p_3 + ABp_4$. Hence, this model is able to represent any of the 16 possible two-input functions.

B. Obfusgates: Dopant Programmable Logic Cells

Similar to the above-described camouflaged standard cells, Malik *et al.* [3] proposed implementing circuits using a library of indistinguishable components that they denote *Obfusgates*. An Obfusgate is created by combining a standard cell logic gate with a number of *Obfuscells* applied on its input and output ports. Malik *et al.* [3] demonstrated Obfusgates based on both NAND4 and AND2 standard cells, but here we consider only the NAND4 variant. Depending on the dopant polarity within the active area of the Obfuscells, each Obfuscell can have four different logic functions: 1) inverter; 2) buffer; 3) constant 1; or 4) constant 0. Because a reverse engineer typically does not consider the dopant polarity when reverse engineering a circuit, he will have difficulty learning the logical function of the Obfusgate.

Fig. 2 shows an Obfusgate comprising a NAND4 gate and five Obfuscells. We use a 4-to-1 multiplexer to model each Obfuscell (Fig. 2). The function of the camouflaged circuit can be resolved by finding appropriate values for the programming bits of the Obfuscells. For each NAND4-based Obfusgate, with five Obfuscells each having four possible functions, there are 4^5 total configurations possible. Many of these 4^5 configurations cause the Obfusgate to realize the same functions, and there are 162 unique 4-input logic functions that can be created by the NAND4 Obfusgate.

C. Transformable Interconnects

A third obfuscation technique that we consider is the so-called *transformable interconnects* techniques proposed by Chen *et al.* [4]. This technique uses two types of contacts in interconnects: 1) magnesium (Mg) contacts which are conductors and 2) magnesium oxide (MgO) contacts which are not conductors. The idea of this approach is that, when an attacker tries to reverse engineer the chip by delayering it, the Mg contacts will oxidize into MgO, and thus all Mg and MgO contacts will appear indistinguishable to the attacker. A reverse engineer inspecting the chip will be able to infer that some of the contacts must have initially been Mg and others must have been MgO, otherwise there would be a single net with multiple driving gates; however, the attacker would not be able to know which were originally Mg, and therefore he must figure this out using input–output examples.

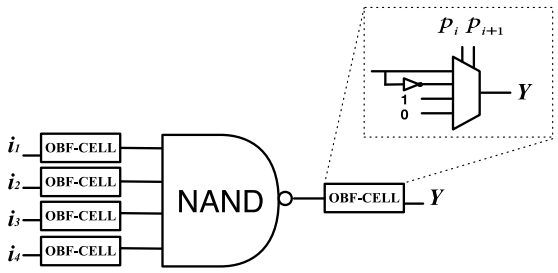


Fig. 2. Schematic of a single *Obfusgate* that consists of five Obfuscells together with a 4-input NAND gate. This gate can have 162 different logic functions depending on the logical functions realized by each Obfuscell.

An example of camouflaging using *transformable interconnect* is shown in Fig. 3. The reverse engineer's view of the circuit is as shown in Fig. 3(a), and from this he will infer that d1, d2, and d3 cannot all be true wires. The configurations in Fig. 3(b)–(d) represent his set of hypotheses for the true connectivity of the circuit. Each one of these would represent the circuit functionality under a single guess about which wire was a nonconducting dummy. The reverse engineer therefore models the transformable interconnect component as shown in Fig. 3(e), where the values of p_0 and p_1 select the true connectivity of the circuit. Now, just as in the previous components, the reverse engineer can use input–output examples from the circuit to infer the values of p_0 and p_1 and hence resolve the function of the circuit.

IV. Problem Formulation

Using the multiplexer-based component models from the previous section (see Figs. 1–3), we now present a deobfuscation algorithm. The algorithm discovers a programming vector assignment that configures the logic function of the model to always agree with the obfuscated circuit. Discovering such a programming vector is the goal of the attacker, as it deobfuscates the circuit function. He uses known input–output pairings of the circuit to provide information about which programming vector values are feasible. A feasible programming vector is one that induces a circuit function that does not contradict any known input–output pairings.

A. Defining Notation

1) Vector $I = \{i_0, \ldots, i_{m-1}\}$ represents an m-bit primary input vector to the circuit.

Algorithm 1 *Incremental SAT-Based Deobfuscation:* Incrementally Generate a Set of Constraints That Are Sufficient to Identify the Correct Model of the Circuit, and Then Solve the Constraints to Find the Model

1: $M(I,P,P') \leftarrow ckt(I,P,O) \wedge ckt(I,P',O') \wedge O \neq O'$ // see Fig. 4
2: $feas(P) \leftarrow \top$ // all programming vectors are feasible initially, unless the model itself imposes constraints
3: $feas(P') \leftarrow \top$
4: **for** $j = 1, 2, 3 \ldots$ **do**
5: // number of satisfying assignments to $feas(P)$ is the number of programming vectors that remain feasible
6: **if** $\exists I, P, P'.$ $\overbrace{M(I,P,P')}^{I \text{ distinguishes } P \text{ and } P'} \wedge \overbrace{feas(P) \wedge feas(P')}^{P \text{ and } P' \text{ both feasible}}$ **then**
7: $\widehat{I}_j \leftarrow I$
8: $\widehat{O}_j \leftarrow$ QUERYORACLE$(\widehat{I}_j)$ // $(\widehat{I}_j, \widehat{O}_j)$ is the j^{th} I/O pair discovered
9: $feas(P) \leftarrow feas(P) \wedge ckt(\widehat{I}_j, P, \widehat{O}_j)$ // strengthen feasibility constraint on P using new I/O pair
10: $feas(P') \leftarrow feas(P') \wedge ckt(\widehat{I}_j, P', \widehat{O}_j)$ // strengthen feasibility constraint on P' using new I/O pair
11: **else**
12: $\exists P.feas(P)$ // Find a single feasible programming assignment P
13: **return** P
14: **end if**
15: **end for**

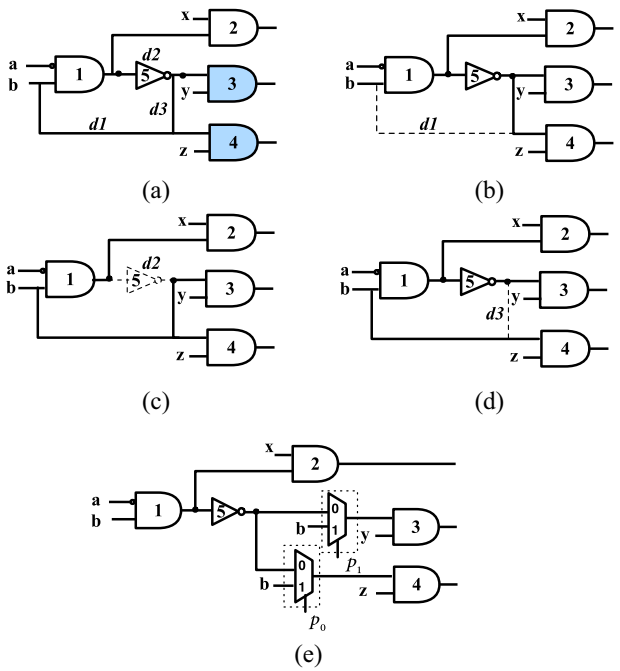


Fig. 3. (a) Design as viewed by reverse engineers. (b)–(d) Three valid circuit configurations when $d1$ or $d2$ or $d3$ is transformable interconnect. (e) Attacker model of *transformable interconnect*.

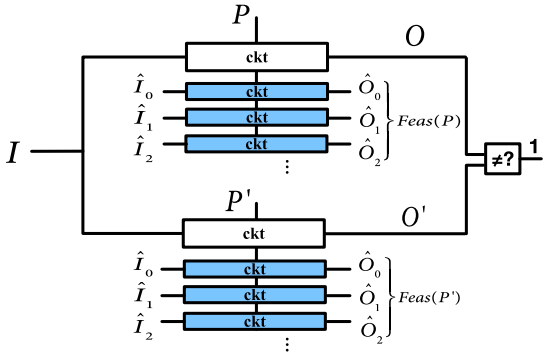


Fig. 4. Unshaded components comprise a miter used to find conditions, where two copies of the circuit produce different outputs due to different programming vectors. The shaded components enforce feasibility constraints that restrict programming vectors to be consistent with input–output examples.

2) Vector $P = \{p_0, p_1, \ldots\}$ represents a programming vector that specifies the logic function implemented by each camouflaged component in the circuit. The length of the programming vector depends on the number of camouflaged circuit elements and the number of possible realizations for each element. The value of P together with the noncamouflaged circuit components together fully specify the logical function of the overall circuit. We denote a programming vector as *feasible* if the logic function it induces does not contradict a set of known input–output examples. Learning new input–output examples incrementally constrains feasible values of the programming vector P.
3) Vector $O = \{o_0, \ldots, o_{n-1}\}$ represents an n-bit primary output vector.
4) The combinational circuit model, including all multiplexers and programming bits, is converted into a CNF formula *ckt* using Tseitin encoding. We use $ckt(I,P,O)$ to denote the CNF formula of the circuit when I, P, and O are the input variables, programming bits, and output variables, respectively. Wherever $ckt(I,P,O)$ appears in Algorithm 1 (at lines 1, 9, and 10), it always refers to a fresh copy of the circuit CNF with new variables for all internal circuit nodes. If two copies of the circuit CNF share a common input vector or programming vector, the respective inputs or programming vectors are equated

to each other outside of the CNF of the combinational circuit.

5) We use subformula $M(I, P, P')$ in Algorithm 1 (line 1) to represent the CNF-encoded miter of two copies of the circuit, as shown by the unshaded blocks in Fig. 4. This formula is satisfiable if and only if there exists an input I, and programming vectors P and P' that cause the circuit to map I to different output values. Restated, $M(I, P, P')$ is satisfiable if *some* P and P' cause the circuit to realize different logic functions. If $M(I, P, P')$ is unsatisfiable, then it means that *all* P and P' cause the circuit to realize the same logic function. On its own, this formula would typically be easily satisfiable and not meaningful, but it becomes useful when combined with additional feasibility constraints on P and P'. In that scenario, the formula $M(I, P, P')$ is used for checking whether the constraints can be satisfied by two different realizable logic functions, or whether the constraints are strong enough that only a single realizable logic function satisfies them.
6) We use subformulas $feas(P)$ and $feas(P')$ to denote the feasibility constraints applied to programming vectors P and P', respectively. These formulas are identical except for being applied to different copies of the programming vector. The formulas evaluate to true only for the subset of programming vectors that are consistent with a set of input–output pairings obtained from the oracle. These feasibility constraints are CNF-encodings of the shaded blocks in Fig. 4.

The goal of the attacker is to recover the function of the obfuscated circuit by finding a value of P that induces his model to realize the same function as the oracle. Two functions are equivalent if they produce the same outputs for all possible inputs, but equivalence is usually checked symbolically instead of by exhaustively applying inputs. However, symbolic equivalence checking cannot be applied between a model and a black box oracle, so SAT-based reverse engineering relies on an oracle-guided synthesis approach, as described in the remainder of this paragraph. Given that there are programming vectors to select all possible functions of all camouflaged components, there necessarily exists one or more values of P that will cause the model to realize the same function as the oracle. Because the attacker knows that there must exist such a value of P, he can find it by ruling out values of P using input–output examples from the oracle until only a single function remains. At this point, the one function that is not ruled out is known to be equivalent to the oracle by process of elimination. Any value of P that induces the model to have this function is a solution to the deobfuscation problem.

B. SAT Solving of Camouflaged Circuit

Boolean satisfiability solving is a common technique for reasoning about circuit logic that is used widely in automated test pattern generation (ATPG) [26]. A gate-by-gate translation using Tseitin encoding maps the circuit logic into a (CNF)-encoded SAT problem. The number of variables in the resulting CNF problem is equal to the number of nodes in the circuit, and the number of clauses in the CNF problem is linear in the number of circuit logic gates. The CNF clauses constrain the Boolean values of the circuit nodes to be consistent the logic gates that relate them. If a CNF problem is satisfiable, an SAT solver finds an assignment of 0 or 1 to each variable that is consistent with the circuit logic gates and any additional constraints added to the problem.

What makes the deobfuscation problem different from ATPG is that obfuscated components are represented in the CNF formula using the circuit constructs of Figs. 1–3, which get encoded into the CNF in the same manner as all other nodes and gates. The variables in the CNF problem then include circuit inputs, circuit outputs, and the programming vector. When a known input–output pair is applied to the circuit, unit clauses are added to the CNF to force the appropriate variables to take the desired values. To avoid confusion with arbitrary input and output vectors (I and O, respectively), we denote an input–output pairing that is known to be correct as $\widehat{I}$ and $\widehat{O}$ with various subscripts.

C. Incremental-SAT Algorithm

Our oracle-guided incremental-SAT-based algorithm is given in Algorithm 1. Following the notation described at the start of this section, the algorithm uses the subformula $M(I, P, P')$ to check whether two programming vectors induce different logic functions, and uses $feas(P)$ and $feas(P')$ to constrain programming vector assignments to be consistent with all previously observed input–output pairs. In Algorithm 1, both $feas(P)$ and $feas(P')$ are typically initialized to $\top$ meaning that the programming vectors are initially unconstrained; however, when using the modeling construct in Fig. 1(a), where only the 00, 01, and 10 values are used for each pair of programming bits, we initialize the constraints to rule out the 11 assignment.

The feasibility constraints are increasingly strengthened as the algorithm iterates through the loop. At the jth loop iteration in Algorithm 1, a satisfying assignment at line 6 produces an input vector I that can distinguish two feasible programming vectors P and P'. This vector I is assigned to $\widehat{I}_j$ and the oracle is queried to obtain the corresponding output $\widehat{O}_j$. The pair $(\widehat{I}_j, \widehat{O}_j)$ is known to be a correct input–output pairing according to the oracle, and it is used to strengthen the programming vector feasibility constraints at lines 9 and 10. To strengthen the constraints on P, a new copy of the circuit CNF formula is added with P as its programming vector and $\widehat{I}_j$ and $\widehat{O}_j$ applied to inputs and outputs as unit clauses (line 9). The feasibility constraint on P' is strengthened in the same way (line 10). The strengthening of the feasibility constraints corresponds to adding new shaded blocks in Fig. 4. Note that the strengthened feasibility constraints will necessarily have fewer solutions after being strengthened; specifically, among the values P and P' that satisfied the SAT formula at line 6, at least one will now be infeasible.[2]

[2]Because P and P' induce different outputs under the input vector $\widehat{I}_j$, no more than one of them can induce output vector $\widehat{O}_j$, which is now known to be correct.

TABLE I
VALUES PRODUCED AT EACH ITERATION OF ALGORITHM 1 DURING DEOBFUSCATION OF THE CAMOUFLAGED C17 CIRCUIT IN FIG. 6. THE NUMBER OF FEASIBLE CONFIGURATIONS IS THE NUMBER OF PROGRAMMING VECTORS THAT SATISFY THE CONSTRAINTS AT EACH ITERATION OF THE ALGORITHM

j	num. feasible	SAT solution (Alg. 1, line 6)			constraint learned	CNF statistics Incremental		CNF statistics Baseline	
	configurations	$\{I\}$	$\{P\}$	$\{P'\}$	$(\widehat{I}_j, \widehat{O}_j)$	*num. vars*	*num. clauses*	*num. vars*	*num. clauses*
1	27	01000	10,01,00	00,10,00	(01000,11)	114	364	114	364
2	14	10100	00,10,01	00,01,01	(00000,00)	170	640	239	739
3	8	10110	00,00,01	00,01,01	(10110,10)	236	916	364	1114
4	6	00100	10,01,01	00,01,01	(00100,00)	294	790	489	1489
5	4	00000	01,01,01	00,01,01	(00000,00)	344	1066	614	1864
6	2	11101	01,01,01	01,01,00	(11101,11)	333	1342	739	2239
7	1	UNSAT			-	95	1614	864	2614

Once the feasibility constraints are sufficiently strong, there will no longer exist two different programming vectors that induce distinct logic functions while also satisfying the feasibility constraints. At this point, the SAT call at line 6 becomes unsatisfiable, and a final SAT call is made (line 12) to find a single programming vector P that satisfies the feasibility constraints. Note that the value of P that is discovered may not be a unique solution, but it is known that no other feasible P' induces a different overall logic function,[3] as this is necessary for the SAT call at line 6 to be unsatisfiable.

Relative to SAT-based attacks of El Massad *et al.* [21] and Subramanyan *et al.* [22], a distinguishing feature of this paper is the use of incremental SAT. A typical SAT problem is encoded in CNF and solved by an SAT solver to output either a satisfying assignment or a result of UNSAT to indicate that no such assignment exists. In the process of solving the SAT problem instance, the solver spends considerable time learning from conflicts and making inferences to simplify the problem and guide its search toward a satisfying assignment. If solving a set of related SAT instances, it is desirable to reuse this reasoning to reduce the number of costly inferences made in each SAT call [27]; incremental SAT is the formulation that allows for efficient reuse of inference across related SAT instances. Our problem is amenable to solving by incremental SAT because each SAT query (line 6 of Algorithm 1) is solving an instance obtained by adding clauses (at lines 9 and 10) to the previously solved SAT problem instance. All inferences learned in one SAT problem are therefore still applicable in the subsequent one.

A number of engineering challenges are addressed in order to use an oracle-guided approach with incremental SAT solving. An overview is given here, with more information found in the user manual included with our program. The algorithm is implemented using a modified version of MiniSat [28] version 2.2.0. From within MiniSat, at each iteration of the algorithm, when a satisfying assignment to the CNF is produced (at line 6 of Algorithm 1), the primary input values $(\widehat{I}_j)$ are extracted and mapped into their corresponding signal names and printed to a file. The oracle, implemented as a standalone executable, is then queried (line 8) and the program waits for the oracle to map $\widehat{I}_j$ to $\widehat{O}_j$. Once the oracle has produced $\widehat{O}_j$, the program adds new clauses to the ongoing CNF problem in order to strengthen the constraints on P and P'.

[3]If two programming vectors do not produce different outputs for any input vectors, then they induce the same logic function.

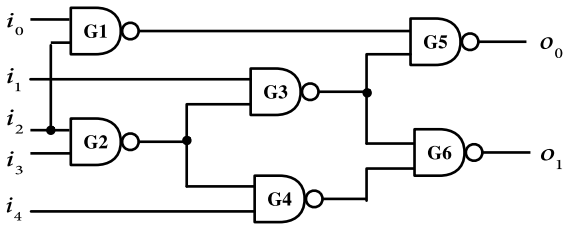


Fig. 5. Gate-level netlist of circuit c17.

We use the MiniSat 2.2.0 (simp) version because it implements variable elimination and simplification before solving. The overhead cost of performing the simplifications is justified because the simplified constraints are carried forward and used in all future iterations. Because this version of MiniSat can eliminate variables, care must be taken to "freeze" certain variables so that they will not be eliminated. In our case, the programming vectors P and P', and the input vector I, are frozen. Being frozen means that variables will always remain in the SAT problem, and this makes it possible to read out their values whenever a satisfying assignment is found.

D. Baseline SAT-Based Deobfuscation Algorithm

Our SAT formulation is similar to the approach that was first demonstrated by El Massad *et al.* [21], and we denote this earlier approach as the baseline algorithm that we compare against. Their algorithm is a significant advancement in reverse engineering, and was able to deobfuscate in minutes problems that would take years to deobfuscate using brute force [2]. Aside from attacking different circuits, the significant difference between our approach and the baseline approach is simply that we use incremental SAT to keep strengthening the feasibility constraints within a single persistent SAT problem. By contrast, the baseline approach treats each iteration of the algorithm as an unrelated SAT problem to solve, and does not carry forward anything learned at previous iterations except for the input–output pairings. Therefore, the baseline algorithm does not carry forward any learned information or simplifications from one iteration to the next. In Section V, we compare the runtime of our incremental approach to the baseline approach.

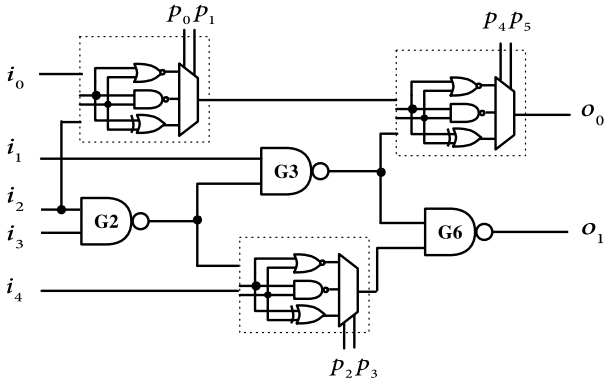


Fig. 6. Modeling of c17 benchmark with G1, G4, and G5 camouflaged using NAND/NOR/XOR camouflaging.

E. Illustrative Example

In this section, we demonstrate Algorithm 1 using ISCAS-85 benchmark circuit c17. We demonstrate the algorithm with two types of camouflaging techniques, NAND/NOR/XOR camouflaged standard cells and fully camouflaged logic gates. Additionally, we show an example in which the successfully resolved logic function of the circuit is correct but does not agree with the oracle circuit on a gate-by-gate basis. The gate-level netlist of c17 is shown in Fig. 5.

1) Example 1—NAND/NOR/XOR Camouflaging: The NAND/NOR/XOR camouflaged *c*17 circuit is shown in Fig. 6. The gates *G*1, *G*4, and *G*5 are camouflaged. We represent the camouflaged gates using the model given in Fig. 1(a). The value of the programming vector P is returned at the end of our algorithm, and this programming vector will select gate functions that make the model logically equivalent to the oracle. For the two bits of the programming vector corresponding to each gate, a value of 00 selects an XOR gate, a value of 01 selects a NAND gate, and a value of 10 selects a NOR gate. Because there are only three possible gate functions, a constraint is added to rule out the value 11 for each pair of programming bits.

The results of each iteration of Algorithm 1 when applied to this example are recorded in Table I. In the table, j is the iteration number; I, P, P' are the assigned values in the solution to the SAT call at line 6 of the algorithm; $(\hat{I}_j, \hat{O}_j)$ are the input–output pairs obtained and used to strengthen feasibility constraints at lines 9 and 10 of the algorithm. For example, the first iteration finds input value $I = 01000$ which is then applied to the oracle to obtain the input–output pairing (01000, 11). This one pairing reduces the number of feasible programming vectors from 27 to 14. After finding six input–output pairings, on the seventh iteration the SAT call at line 6 becomes unsatisfiable, indicating that the feasibility constraints imposed by these six pairs are sufficient to identify a unique logic function that matches the oracle and thus deobfuscates the circuit. In this example, there is only one feasible configuration remaining at the UNSAT iteration. Note that there are usually many feasible configurations remaining at the UNSAT iteration on large circuits. Although not depicted in Table I, a final SAT call (line 12 of Algorithm 1) produces the programming vector $P = 010101$ which selects the NAND functionality for gates *G*1, *G*4, and *G*5. This programming vector deobfuscates the circuit, and in this case it happens to do so in a way that is gate-by-gate logically equivalent to the camouflaged circuit. Note in Table I that the number of clauses and unresolved variables at each iteration of the algorithm are growing sublinearly and sometimes decreasing. This reflects the solving making simplifications to the problem as it runs. This becomes crucially important for large camouflaged designs.

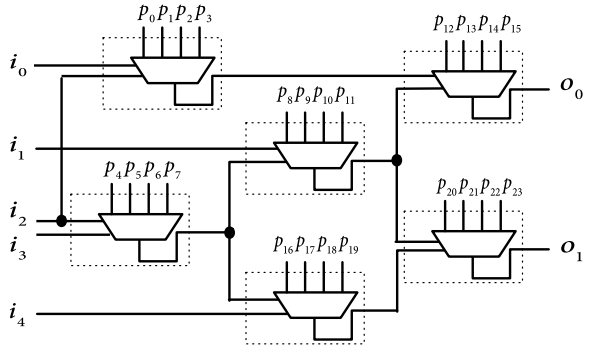


Fig. 7. Modeling of c17 with all gates fully camouflaged. In this scenario, the reverse engineer only knows the routing.

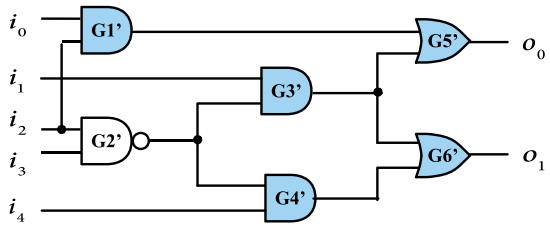


Fig. 8. Resolved function of c17 when all gates are fully camouflaged.

2) Example 2—All Gates Are Camouflaged: The most challenging case for our algorithm is deobfuscating a circuit in which all the logic gates are fully camouflaged. In this scenario, only the routing is known to the reverse engineer. We demonstrate this problem using c17, modeled as shown in Fig. 7. Our algorithm requires 24 iterations to deobfuscate this small circuit. It is notable that the resolved circuit (Fig. 8) is not gate-by-gate equivalent to the obfuscated circuit, but nonetheless has resolved the correct function for the circuit. Deobfuscation when only routing is known does not scale well because the reverse engineer has very little useful information. For example, we are not able to deobfuscate ISCAS benchmark circuit *c*432 after three days when all 160 gates are camouflaged in this manner.[4]

V. Evaluation of De-Obfuscation Algorithm

We evaluate our deobfuscation algorithm on a set of ISCAS-85 combinational benchmarks [29], including c432, c499, c880, c1355, c1908, c2670, c3540, c5315, and c7552.

[4]The ISCAS-85 benchmark c432 has 160 gates, some of which have more than two inputs. Because we use 2-input obfuscated gates, we map c432 into a circuit with 209 gates of 2 or fewer inputs, and then obfuscate all 209 cells of the remapped circuit.

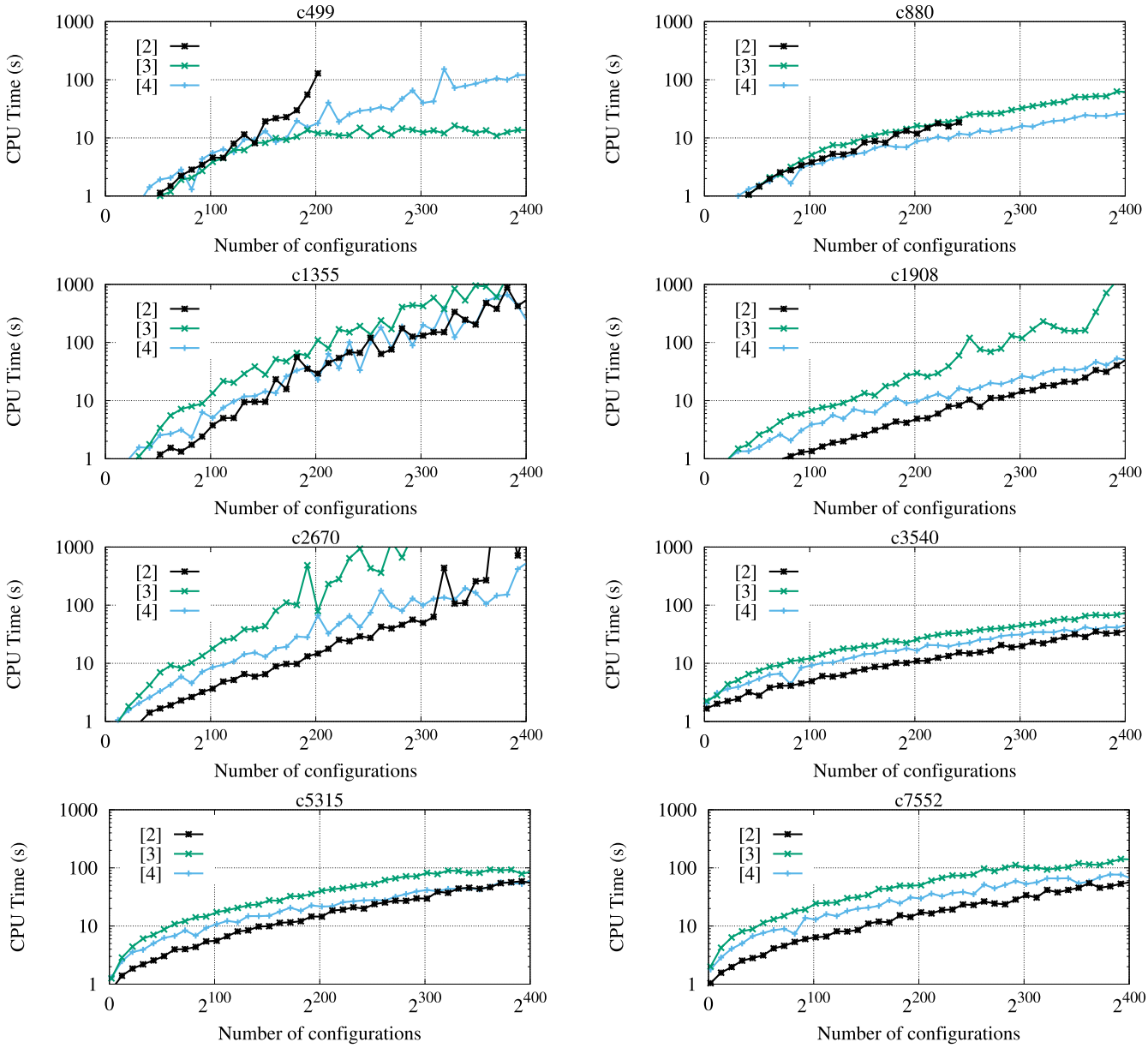


Fig. 9. Plots show average runtime to deobfuscate eight ISCAS-85 benchmarks with varied numbers of randomly obfuscated components using the camouflaging techniques presented in *camouflaged standard cells* [2], *Obfusgates* [3], and *transformable interconnects* [4]. Specifically, the runtimes shown are the average runtimes over ten random trials for each technique and number of obfuscated components.

The sizes of these circuits range from several hundred to several thousand gates. We use the attacker model introduced in Section III. Our previous work [1] demonstrated that our algorithm is able to efficiently deobfuscate the six largest ISCAS benchmarks with up to 200 gates implemented using NAND/NOR/XOR camouflaged standard cells.

A. Evaluation of Camouflaging Techniques

We apply our incremental deobfuscation algorithm to reverse engineer designs camouflaged using *camouflaged standard cells* [2], *Obfusgates* [3], and *transformable interconnects* [4]. We implement each of the three camouflaging techniques randomly as summarized in Table II and described here.

TABLE II
CAMOUFLAGABLE COMPONENTS IN THE ISCAS-85 BENCHMARKS WHEN APPLYING DIFFERENT CAMOUFLAGING TECHNIQUES. NOTE THAT, IN THE CASE OF TRANSFORMABLE INTERCONNECTS, ANY NET CAN BE CHOSEN, BUT THE CHOICE OF DUMMY CONNECTIONS IS RESTRICTED TO AVOID CREATING APPARENT COMBINATIONAL LOOPS

Camoflaging Technique	Camouflagable components in ISCAS-85 benchmarks
NAND/NOR/XOR camouflaged cells [2]	NAND2, NOR2, XOR2
NAND4 Obfusgates [3]	AND/NAND(2,3,4), INV, OR/NOR(2,3,4), BUFFER
Tranformable Interconnects [4]	any net

1) For *camouflaged standard cells*, we select gates to camouflage by randomly choosing among the NAND2, NOR2, and XOR2 gates of each benchmark circuit, as

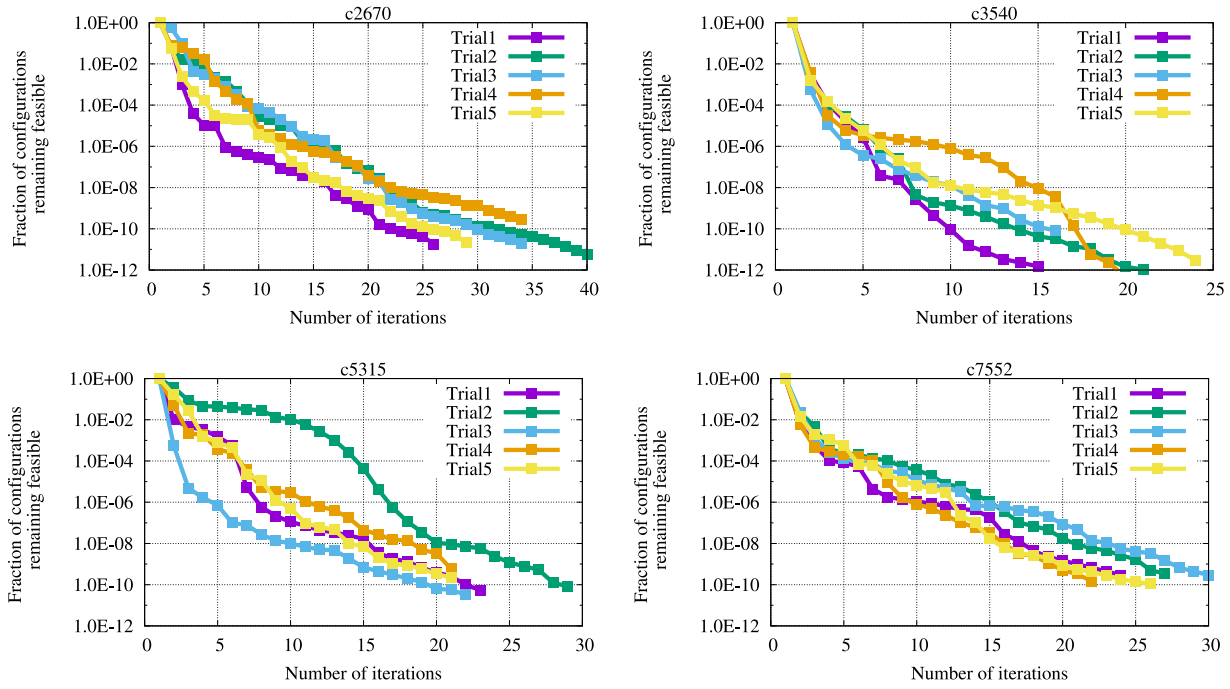


Fig. 10. Eliminating feasible configurations using input–output examples generated by the de-obfuscation algorithm for circuits c2670, c3540, c5315, and c7552 with 51 gates camouflaged using camouflaged standard cells that can implement NAND, NOR, or XOR gates. The initial model of each camouflaged circuit has 3^{51} configurations. The five trials in each plot denote five different random choices of which gates to camouflage.

these are the types of cells that can be realized by the camouflaged standard cells.

2) For *Obfusgates*, although each NAND4 obfusgate can implement 162 different logical function, most of the functions do not exist in the ISCAS-85 benchmarks. The gates from the ISCAS benchmarks that can realized by the NAND4 obfusgates are as shown in Table II.
3) For *transformable interconnects*, we select randomly a net to camouflage, and then add *dummy wires* from three other nets to the chosen net, giving a choice of four possible drivers for the net that the reverse engineer must resolve. If the dummy wires created cycles, then a reverse engineer could identify these wires as dummies from topological analysis, so when camouflaging a net, we avoid creating dummy wires from any signals in the transitive fanout of that net.

We apply all three techniques to the eight ISCAS benchmarks, as shown in Fig. 9. In each case, we vary the number of components that are obfuscated, and for each number of obfuscated components, we repeat the experiment ten times making different random choices of which components to obfuscate, and plot the average runtime. To provide a common framework for comparison, we plot the deobfuscation runtime against the number of possible configurations in Fig. 9. The number of possible configurations is 2^x, where x is the number of programming bits needed to select the functionality of the circuit. The value of x also indicates the number of camouflaged cells in the circuits. Specifically, in Fig. 9, the numbers of camouflaged cells for *camouflaged standard cells* [2], *Obfusgates* [3], and *transformable interconnects* [4] are $x/2$, $x/5$, and $x/2$, respectively. Note that we do not consider here that some programming bits select the same logic function.

B. Quantifying the Elimination of Feasible Solutions

As SAT-based deobfuscation (Algorithm 1) increasingly rules out infeasible programming vectors representing different possible circuit configurations, we present here a study of how many programming vectors are eliminated at each iteration of the algorithm. Fig. 10 shows, for circuits c2670, c3540, c5315, and c7552 with 51 camouflaged standard cells, the number of configurations that remain feasible at each iteration of the deobfuscation algorithm. At each iteration, the total number of feasible configurations are obtained by counting the number of satisfying assignments to the CNF-encoded feasibility constraint [$feas(P)$] applied to the programming vector (see line 5 of Algorithm 1). Note that the number of feasible programming vectors is not counted during normal operation of the deobfuscation algorithm. Counting the number of solutions to a CNF formula is known as the #SAT problem (or as *model counting* [30]). The data in Fig. 10 is obtained using the #SAT solver sharpSAT [31].

A diminishing number of configurations remain feasible as new input–output pairs are generated at each iteration of the algorithm. The number of feasible configurations is reduced very quickly. Even with only 51 gates camouflaged, fewer than one in a billion programming vectors induce the same circuit function as the oracle. Even with feasible programming vectors being so rare, there are a very large number of feasible configurations, given that the space of possible vectors is 3^{51}.

C. Effect of Corruptibility-Based Camouflaging

As an alternative to randomly selecting gate instances to camouflage, previous works have proposed selecting gates to obfuscate such that incorrect guesses of the camouflaged gates

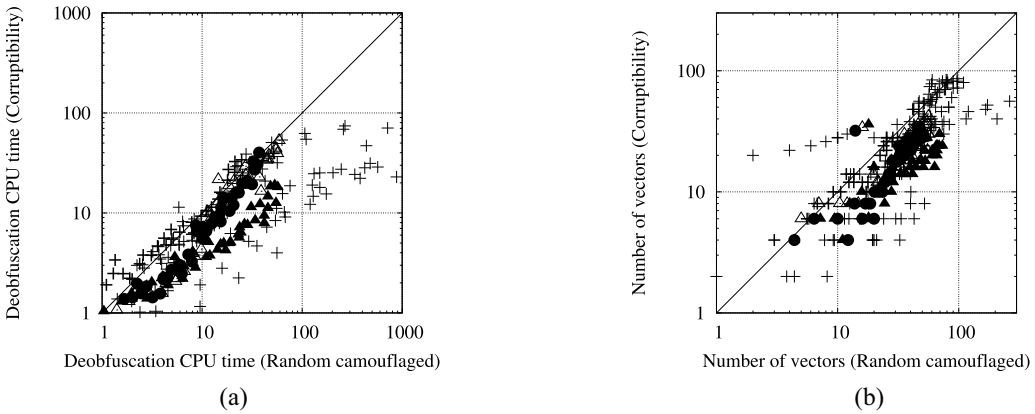


Fig. 11. Comparing the total deobfuscation CPU time and the number of vectors used for deobfuscation of *corruptibility*-guided and *random* camouflaged ISCAS-85 benchmarks. (a) Runtime of deobfuscation. (b) Number of vectors of deobfuscation.

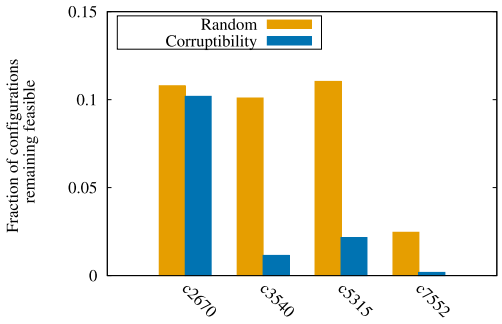


Fig. 12. Average fraction of configurations that remain feasible with respect to a single randomly chosen PI-PO pair.

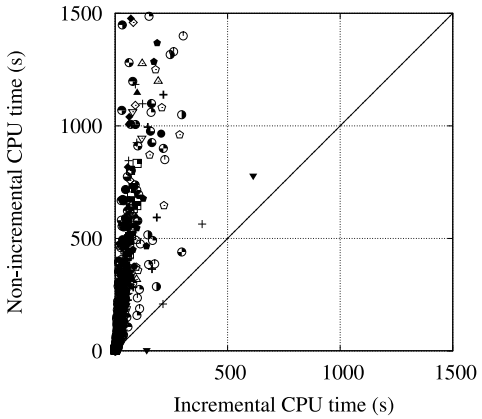


Fig. 13. Comparing the total deobfuscation runtime of baseline and incremental algorithms on 2400 randomly camouflaged circuits instances using different styles of camouflaged gates. The incremental solver gives an average speedup of 10.5×. Runtimes exceeding 1500 s are truncated from the plot.

will maximize corruption of the primary outputs [2], [16]. When using corruptibility as a metric, a gate is scored for camouflaging according the average Hamming distance between the output produced under the correct functionality of the gate and an incorrect guess of the gate functionality that could be made by a reverse engineer. We measure corruptibility using the fault simulation tool HOPE [32]. Since the corruptibility for each circuit is fixed, the camouflaging order is also fixed. The plots Fig. 11 compares randomly camouflaged circuits and circuits camouflaged according to corruptibility. When camouflaging is chosen to maximize output corruptibility, then any observation of outputs will be highly effective for ruling out large numbers of possible functions of camouflaged components. We notice in Fig. 11 that: 1) corruptibility-guided obfuscation generally requires less CPU time to deobfuscate [Fig. 11(a)] and 2) corruptibility-guided obfuscation generally requires fewer vectors to deobfuscate each circuit [Fig. 11(b)]. Note, however, that our algorithm does not necessarily minimize the number of vectors.

The vectors discovered using the SAT-based deobfuscation are not objectively chosen, as they are obtained from satisfying assignments produced by the SAT solver. The vectors may therefore be biased by the search heuristics of the solver or other factors. To provide a fairer comparison between random camouflaging and corruptibility-guided camouflaging, we perform experiments using randomly chosen input–output pairings. Instead of using the input–output pairs generated by the SAT solver, we choose input vectors randomly from the space of possible vectors, and obtain the corresponding output vectors from the oracle. Each input-output pair is translated to a feasibility constraint, and we again use sharpSAT to count the number of programming vectors that are consistent with the constraints. We repeat this for ten different randomly camouflaged instances of c2670, c3540, c5315, and c7552, and for the same circuits camouflaged according to corruptibility. Fig. 12 shows the average fraction of programming vectors that remain feasible after a single input–output pair. This result shows that, a smaller share of configurations remain feasible when using corruptibility-guided camouflaging instead of random camouflaging.

D. Limitation of SAT-Based Deobfuscation

It is well-known that certain circuits, such as multipliers, are particularly challenging for analyzing with SAT [33], and this limits the effectiveness of SAT-based deobfuscation on such circuits. To examine this, we tested our algorithm on two multipliers. The first is ISCAS-85 benchmark c6288, with ten

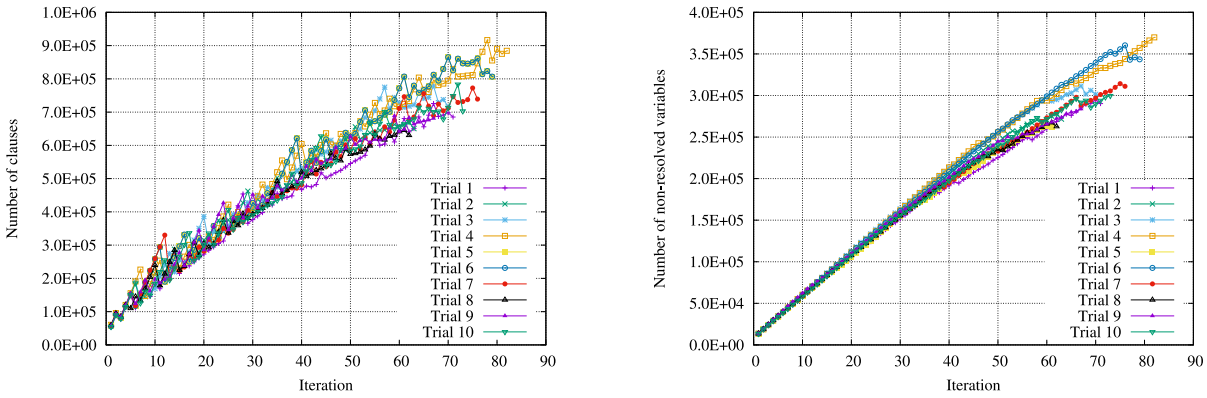


Fig. 14. Examining the variable and clauses elimination using incremental SAT solving on ten randomly camouflaged instances of ISCAS-85 benchmark c7552, each with 200 NAND/NOR/XOR camouflaged standard cells [2]. (a) Number of clauses remaining in each iteration. (b) Number of nonresolved variables in each iteration.

gates camouflaged using NAND/NOR/XOR camouflaged standard cells. Despite the small number of camouflaged cells, the design required 5.4 h to deobfuscate. All the iterations except the last one require no more than 2 s. The SAT call at the final iteration, because it is the one that returns an UNSAT result (at line 6 of Algorithm 1), takes many hours. Additionally, we repeat the same experiments on a 16-bit Montgomery multiplier [34] in field GF(2^{16}), and are not able to deobfuscate the multiplier in 6 h with one gate camouflaged. The reason is that solving GF multiplier problems using SAT is very difficult, which has been studied in [35]. In conclusion, any SAT-based algorithm such as ours will be limited when trying to reverse engineer designs such as multipliers, or cryptographic ciphers, that are notoriously hard for SAT. This fact may be exploited in future works where camouflaging can be deployed in a way that will resist SAT-based reverse engineering [36].

The problem of logic locking is closely related to the problem of camouflaging [37], and in logic locking, recent works have showed the importance of trying to thwart SAT attacks by ensuring that each input–output example provides limited information about the values of the key bits that unlock the circuit [38], [39]. In an extreme case, one can guarantee that an exponential number of input–output examples are needed to exactly learn the key bits, but a consequence of this is that output corruption under incorrect key guesses will be limited [38]. Similar approach has been used with camouflaging to quantify the security of the camouflaged circuits [40], [41]. Note that in logic locking, care must be taken to ensure that the key gates cannot be identified and removed, and thus to remain secure under reverse engineering logic locking can be combined with obfuscation [38], [42].

E. Incremental Algorithm Versus Baseline

We compare the runtimes of our incremental algorithm against a baseline result from our own reimplementation of the algorithm of El Massad *et al.* [21], which is the best result demonstrated before ours. Any style of camouflaging that can be attacked by the incremental SAT algorithm can also be attacked by the baseline algorithm, but our comparison of the two focuses on *camouflaged standard cells* because this is the technique deobfuscated in El Massad's work. Fig. 13 shows the runtime using baseline and incremental algorithms to deobfuscate ISCAS-85 benchmarks with randomly selected gates that use the NAND/NOR/XOR camouflaged standard cells and the cells from Fig. 1(b). The position of each datapoint indicates its runtime for the incremental solver and runtime for the baseline solver. The incremental solver gives a 10.5× reduction in average runtime across all examples compared to the baseline approach, and the average speedup is 6.5×; the difference between the two metrics is that average runtime is strongly influenced by the largest examples, where the incremental solver gives a larger improvement. While our result does not challenge the complexity results established by El Massad for the baseline algorithm, we do show that a significant performance improvement is possible using incremental SAT.

To understand the efficiency of the incremental algorithm, we study in more detail the incremental deobfuscation of circuit c7552 with 200 gates camouflaged using the NAND/NOR/XOR style [2]. Fig. 14 shows how the number of clauses and unresolved variables grow as the incremental algorithm proceeds. If the solver is not able to make any simplifications, then both clauses and unresolved variables will grow linearly with the number of iterations as new copies of the CNF-encoded circuit are added to the problem. Fig. 14(a) shows sublinear growth in the number of clauses in each iteration of the incremental algorithm, and Fig. 14(b) shows sublinear growth in the number of unresolved variables. This indicates that partial information from different copies of the circuit CNF are being combined together in useful ways.

VI. Conclusion

This paper proposes an incremental-SAT-based approach for deobfuscating camouflaged circuits. We have implemented the algorithm and tested its performance by using it to deobfuscate ISCAS-85 combinational benchmarks when camouflaged using three different styles of component camouflaging.

The results show that our algorithm is able to efficiently deobfuscate the ISCAS-85 benchmarks regardless of camouflaging style, and are able to do so 10.5× faster than the best existing approaches. Our tool will be released publicly to evaluate and support development of future selective component camouflaging approaches.

REFERENCES

[1] D. Liu, C. Yu, X. Zhang, and D. Holcomb, "Oracle-guided incremental SAT solving to reverse engineer camouflaged logic circuits," in *Proc. Design Autom. Test Europe (DATE)*, Dresden, Germany, 2016, pp. 433–438.

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

[3] S. Malik, G. T. Becker, C. Paar, and W. P. Burleson, "Development of a layout-level hardware obfuscation tool," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Montpellier, France, 2015, pp. 204–209.

[4] S. Chen *et al.*, "Chip-level anti-reverse engineering using transformable interconnects," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFTS)*, Amherst, MA, USA, 2015, pp. 109–114.

[5] R. Torrance and D. James, "The state-of-the-art in semiconductor reverse engineering," in *Proc. 48th Design Autom. Conf. (DAC)*, San Diego, CA, USA, 2011, pp. 333–338.

[6] K. Nohl, D. Evans, S. Starbug, and H. Plötz, "Reverse-engineering a cryptographic RFID tag," in *Proc. USENIX Security*, vol. 28. San Jose, CA, USA, 2008, pp. 185–193.

[7] W. Li, Z. Wasson, and S. A. Seshia, "Reverse engineering circuits using behavioral pattern mining," in *Proc. IEEE Int. Symp. Hardw. Orient. Security Trust (HOST)*, San Francisco, CA, USA, 2012, pp. 83–88.

[8] P. Subramanyan *et al.*, "Reverse engineering digital circuits using functional analysis," in *Proc. Conf. Design Autom. Test Europe*, Grenoble, France, 2013, pp. 1277–1280.

[9] A. Gascón *et al.*, "Template-based circuit understanding," in *Proc. IEEE Formal Methods Comput.-Aided Design (FMCAD)*, Lausanne, Switzerland, 2014, pp. 83–90.

[10] SypherMedia. (2012). *SypherMedia Library–Circuit Camouflage Technology*. Accessed on Apr. 21, 2015. [Online]. Available: http://www.smi.tv/SMI_SypherMedia_Library_Intro.pdf

[11] R. P. Cocchi, J. P. Baukus, L. W. Chow, and B. J. Wang, "Circuit camouflage integration for hardware IP protection," in *Proc. 51st ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, San Francisco, CA, USA, 2014, pp. 1–5.

[12] G. T. Becker, F. Regazzoni, C. Paar, and W. P. Burleson, "Stealthy dopant-level hardware trojans," in *Proc. 15th Int. Workshop Cryptograph. Hardw. Embedded Syst. (CHES)*, Santa Barbara, CA, USA, Aug. 2013, pp. 197–214.

[13] M. Shiozaki, R. Hori, and T. Fujino, "Diffusion programmable device: The device to prevent reverse engineering," *IACR Cryptology ePrint Archive*, vol. 2014, p. 109, 2014.

[14] A. Iyengar and S. Ghosh. (2015). *Threshold Voltage-Defined Switches for Programmable Gates*. [Online]. Available: http://arxiv.org/abs/1512.01581

[15] M. I. M. Collantes, M. E. Massad, and S. Garg. (2016). *Threshold-Dependent Camouflaged Cells to Secure Circuits Against Reverse Engineering Attacks*. [Online]. Available: http://arxiv.org/abs/1605.00684

[16] R. S. Chakraborty and S. Bhunia, "HARPOON: An obfuscation-based SoC design methodology for hardware protection," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 10, pp. 1493–1502, Oct. 2009.

[17] Y. Bi *et al.*, "Emerging technology-based design of primitives for hardware security," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 1, p. 3, 2016.

[18] A. Vijayakumar, V. C. Patil, D. E. Holcomb, C. Paar, and S. Kundu, "Physical design obfuscation of hardware: A comprehensive investigation of device and logic-level techniques," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 1, pp. 64–77, Jan. 2017.

[19] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Logic encryption: A fault analysis perspective," in *Proc. Conf. Design Autom. Test Europe (DATE)*, Dresden, Germany, 2012, pp. 953–958.

[20] S. Jha, S. Gulwani, S. A. Seshia, and A. Tiwari, "Oracle-guided component-based program synthesis," in *Proc. ACM/IEEE 32nd Int. Conf. Softw. Eng.*, vol. 1. Cape Town, South Africa, 2010, pp. 215–224.

[21] M. El Massad, S. Garg, and M. V. Tripunitara, "Integrated circuit (IC) decamouflaging: Reverse engineering camouflaged ICs within minutes," in *Proc. 22nd Annu. Netw. Distrib. Syst. Security Symp. (NDSS)*, San Diego, CA, USA, 2015, pp. 1–14.

[22] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *Proc. Hardw. Orient. Security Trust (HOST)*, Washington, DC, USA, 2015, pp. 137–143.

[23] B. Yang, K. Wu, and R. Karri, "Scan based side channel attack on dedicated hardware implementations of data encryption standard," in *Proc. Int. Test Conf. (ITC)*, Charlotte, NC, USA, 2004, pp. 339–344.

[24] O. Kömmerling and M. G. Kuhn, "Design principles for tamper-resistant smartcard processors," *Proc. Smartcard*, vol. 99. 1999, pp. 9–20.

[25] L.-W. Chow, J. P. Baukus, and W. M. Clark, Jr., "Integrated circuits protected against reverse engineering and method for fabricating the same using an apparent metal contact line terminating on field oxide," U.S. Patent 7 294 935, Nov. 13, 2007.

[26] T. Larrabee, "Test pattern generation using Boolean satisfiability," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 11, no. 1, pp. 4–15, Jan. 1992.

[27] N. Eén and N. Sörensson, "Temporal induction by incremental SAT solving," *Electron. Notes Theor. Comput. Sci.*, vol. 89, no. 4, pp. 543–560, 2003.

[28] N. Eén and N. Sörensson, "An extensible SAT-solver," in *Proc. 6th Int. Conf. Theory Appl. Satisfiabil. Test.*, Santa Margherita Ligure, Italy, 2003, pp. 502–518.

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

[30] E. Birnbaum and E. L. Lozinskii, "The good old Davis-Putnam procedure helps counting models," *J. Artif. Intell. Res.*, vol. 10, no. 1, pp. 457–477, 1999.

[31] M. Thurley, "sharpSAT–counting models with advanced component caching and implicit BCP," in *Proc. Int. Conf. Theory Appl. Satisfiability Test.*, Seattle, WA, USA, 2006, pp. 424–429.

[32] H. K. Lee and D. S. Ha, "HOPE: An efficient parallel fault simulator for synchronous sequential circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 15, no. 9, pp. 1048–1058, Sep. 1996.

[33] S. A. Cook and D. G. Mitchell, "Finding hard instances of the satisfiability problem," in *Satisfiability Problem: Theory and Applications: DIMACS Workshop, March 11-13, 1996*, vol. 35. Providence, RI, USA: Amer. Math. Soc., 1997, pp. 1–17.

[34] C. K. Koc and T. Acar, "Montgomery multiplication in GF (2k)," *Designs Codes Cryptography*, vol. 14, no. 1, pp. 57–69, 1998.

[35] J. Lv, P. Kalla, and F. Enescu, "Efficient gröbner basis reductions for formal verification of Galois field multipliers," in *Proc. Conf. Design Autom. Test Europe*, Dresden, Germany, 2012, pp. 899–904.

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

[37] M. Yasin and O. Sinanoglu, "Transforming between logic locking and IC camouflaging," in *Proc. 10th Int. Design Test Symp. (IDT)*, Amman, Jordan, Dec. 2015, pp. 1–4.

[38] M. Yasin, B. Mazumdar, J. J. Rajendran, and O. Sinanoglu, "SARLock: SAT attack resistant logic locking," in *Proc. IEEE Int. Symp. Hardw. Orient. Security Trust (HOST)*, McLean, VA, USA, 2016, pp. 236–241.

[39] Y. Xie and A. Srivastava, "Mitigating SAT attack on logic locking," in *Proc. 18th Int. Conf. Cryptograph. Hardw. Embedded Syst. (CHES)*, Santa Barbara, CA, USA, Aug. 2016.

[40] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "CamoPerturb: Secure IC camouflaging for minterm protection," in *Proc. 35th Int. Conf. Comput.-Aided Design ICCAD*, Austin, TX, USA, Nov. 2016, p. 29.

[41] M. Li *et al.*, "Provably secure camouflaging strategy for IC protection," in *Proc. 35th Int. Conf. Comput.-Aided Design (ICCAD)*, Austin, TX, USA, Nov. 2016, p. 28.

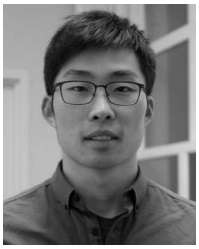
[42] S. Keshavarz, C. Paar, and D. Holcomb, "Design automation for obfuscated circuits with multiple viable functions," in *Proc. Design Autom. Test Europe (DATE)*, 2017.

Cunxi Yu (S'14) received the B.S. degree from the Department of Information and Electronic Engineering, Zhejiang University City College, Hangzhou, China, in 2013. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA, USA.

He was a Research Intern with IBM T. J. Watson Research Center, Yorktown Height, NY, USA, in 2015 and 2016. His current research interests include formal verification, hardware security, and logic synthesis.

Xiangyu Zhang received the B.S. degree from the Florida Institute of Technology, Melbourne, FL, USA, in 2014. He is currently pursuing the M.S. degree with the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA, USA.

He is currently a Intern with Mentor Graphics Emulation Division, Waltham, MA, USA, until 2017.



Duo Liu received the B.S. degree from Jiangnan University, Wuxi, China, in 2012, and the M.S. degree from the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA, USA, in 2015.

He was a Summer Intern with Synopsys Inc, Marlborough, MA, USA, in 2015. He is currently a Verification Engineer with Intel Corporation, Hudson, MA, USA.

Maciej Ciesielski (SM'95) received the M.S. degree in electrical engineering from Warsaw Technical University, Warsaw, Poland, in 1974, the Ph.D. degree in electrical engineering from the University of Rochester, Rochester, NY, USA, in 1983, and the Doctorate Honoris Causa degree from the Université de Bretagne Sud, Lorient, France, in 2008, for his contributions to the development of EDA tools for high level synthesis.

He is a Professor with the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA, USA. From 1983 to 1986, he was with GTE Laboratories, Waltham, MA, USA, on the SILC silicon compiler project. He joined the University of Massachusetts, in 1987. He has authored over 120 publications in international conferences, archived journals, and invited book chapters. His current research interests include electronic design automation, and specifically in synthesis, simulation, and formal verification of very large scale integration circuits and systems.

Daniel Holcomb (M'07) received the B.S. and M.S. degrees in electrical and computer engineering from the University of Massachusetts Amherst, Amherst, MA, USA, and the Ph.D. degree in electrical engineering and computer sciences from the University of California at Berkeley, Berkeley, CA, USA.

He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, University of Massachusetts Amherst. His current research interests include span formal verification, very large-scale integration, embedded systems, and hardware.