Oracle-Guided Incremental SAT Solving to Reverse Engineer Camouflaged Logic Circuits

Duo Liu, Cunxi Yu, Xiangyu Zhang, and Daniel Holcomb
ECE Department, University of Massachusetts Amherst
Amherst, MA, 01003, USA
{duo,ycunxi,xiangyuzhang}@umass.edu, holcomb@engin.umass.edu

***Abstract*—Layout-level gate camouflaging has attracted interest as a countermeasure against reverse engineering of combinational logic. In order to minimize area overhead, typically only a subset of gates in a circuit are camouflaged, and each camouflaged gate layout can implement a few different logic functions. The security of camouflaging relies on the difficulty of learning the overall combinational logic function without knowing which logic functions the camouflaged gates implement.**

In this paper, we present an incremental-SAT approach to reconstruct the logic function of a circuit with camouflaged gates. Our algorithm uses the standard attacker model in which an adversary knows only the non-camouflaged gate functions, and has the ability to query the circuit to learn the correct output vector for any input vector. Our results demonstrate a 5x speedup over the best existing deobfuscation algorithm.

Beyond demonstrating speedup, we use our powerful approach to produce new insights about the strength of obfuscation. First we show that deobfuscation is feasible even in the more challenging setting where layout reveals nothing about the possible logic function of camouflaged gates. Additionally, selectively camouflaging gates to maximize output corruption under incorrect deobfuscation hypotheses typically reduces the number of vectors needed to deobfuscate the circuit.

I. Introduction

An IC designer has 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 work are as follows:

- We present a new incremental-SAT-based algorithm for reverse engineering camouflaged integrated circuits, and demonstrate that it outperforms the best existing reverse engineering algorithms by 5x on ISCAS-85 benchmarks.
- We show that selective gate camouflaging based on the objective of maximizing output corruption [1] offers no resistance to reverse engineering and can reduce the number of vectors required to deobfuscate a circuit.
- We provide to the research community a new standard for logic deobfuscation. The new algorithm should be consulted when evaluating current and future approaches for selective camouflaging.

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. Tools such as Chipworks ICWorks [2] can reconstruct a design schematic from the images. Among other applications, invasive reverse engineering is used for competitive analysis in the IC industry, and has famously been used by Nohl et al. to identify cryptographic weaknesses in the Mifare Classic RFID tag [3]. An overview of the state of the art in invasive reverse engineering is given by Torrance and James [4].

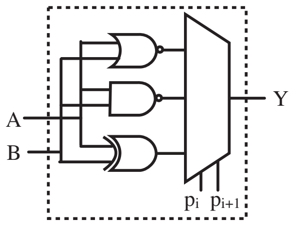
Significant research effort is spent on extracting high-level meaning from the sea of gates obtained by invasive reverse engineering of fabricated circuits. Work by Li et al. resolves subcircuit components by matching against a set of known components [5], and Subramanyan et al. improve on this to operate on unstructured netlists where subcircuits are not identified in advance [6]. Further work by Li et al. extracts word-level structures from unstructured netlists where all gates are known. Work by Gascón et al. checks equivalence between a reference circuit and a circuit-under-investigation when the signal correspondence between the two is unknown [7]. A primary difference between our work 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 the trend of reverse engineering is the use of camouflaged gates. Camouflaged gates are ones for which the logic function cannot be determined by traditional reverse engineering techniques. Circuits can be camouflaged to prevent reverse engineering using specialized gate libraries [8], other layout techniques [9], or by changing the doping of cells without altering their structure at all [10], [11], [12].

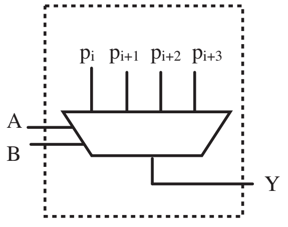
Rajendran et al. [13] present an attacker model for reverse engineering circuits with obfuscated gates. The logic function implemented by a camouflaged circuit should remain hard to discover when the attacker has knowledge of all non-camouflaged gates, and has the ability to apply inputs to the circuit and observe outputs. To minimize cost, it is desirable to protect a circuit by camouflaging only a small subset of the gates [14], [1]. Recent SAT-based approaches have shown that gate camouflaging can be reverse engineered [15], and a similar approach is used to reverse engineer logic encryption [16].

III. Problem Formulation

As demonstrated by El Massad et al [15], uncertainty about the logical functions of camouflaged gates can be translated to uncertainty about the value of certain Boolean variables. These Boolean variables can be considered as the programming vector of a circuit because they configure its functionality. For the specific case of camouflaged gates that can implement NAND, NOR, or XOR gates [1], this programming vector bits are as shown in Fig. 1a [15]. In the more challenging scenario where layout reveals nothing about the logic function of a gate, and the gate can implement any 2-input logic function, we use a look-up-table model as shown in Fig. 1b. The strategy of an attacker is to make observations of a combinational circuit's inputs and outputs and use those observations to eliminate programming vectors that are inconsistent with observations until only one possible circuit function remains consistent with the collected observations.



(a) NAND/NOR/XOR Camouflage



(b) Fully Camouflaged

Fig. 1: Circuit constructs used to model camouflaged gates. Boolean p_i variables configure the logic function of the gate, and an attacker tries to learn the value of these variables.

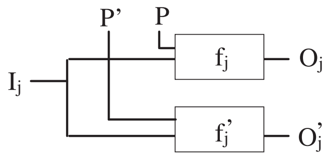


Fig. 2: Two copies of circuit share input vectors but have distinct programming vectors and outputs.

A. Defining Notation

Our formulation of the problem models obfuscated gates using the circuit construction of Fig. 1 where the p_i variables determine the logic function implemented by the gate. The problem of learning the logic function of each gate then becomes the problem of learning appropriate values of p_i variables for each obfuscated gate.

- Vector $I = \{i_0, \ldots, i_{m-1}\}$ represents an m-bit primary input vector to the circuit ($I \in 2^m$).
- Vector $P = \{p_0, p_1, \ldots\}$ represents a programming vector that has values which specify the logic function implemented by each of the x camouflaged gates in the circuit. The length of the programming vector is $2x$ for NAND/NOR/XOR camouflaging (see Fig. 1 1a) or $4x$ for the fully camouflaged style of gate (see Fig. 1 1b). Because P specifies a logical function of every camouflaged gate in the circuit, the combination of P and the non-camouflaged gates together fully specify the logical function of the overall circuit. We denote a programming vector as *feasible* if it is consistent with a set of known input-output examples.
- Vector $O = \{o_0, \ldots, o_{n-1}\}$ represents a n-bit primary output vector ($O \in 2^n$).
- The combinational circuit is converted into a CNF formula f using Tseitin encoding. In each iteration of our algorithm, we create two copies of the circuit CNF f_j and f_j', where subscript j indicates the j^{th} copy, or equivalently the circuit copies used to impose the j^{th} input-output constraint (at the j^{th} iteration of the algorithm). As Fig. 2 indicates, f_j and f_j' share a common input vector I_j, but produce independent outputs O_j, and O_j' respectively. The CNF formulas $f_0, f_1, \ldots$ share a common programming vector P, and likewise $f_0', f_1', \ldots$ all share common programming vector P'.
- The camouflaged circuit is used as an oracle. Its unknown logic function C maps inputs to outputs. An attacker cannot directly observe the logic functions of the camouflaged gates, but can apply any input vector I and observe the corresponding output vector O.
- Because there are programming vectors that describe all possible functions of all camouflaged gates, there necessarily exists some P that will cause the circuit to be functionally equivalent to the oracle. Yet, because the oracle is only known through input-output pairings, one can only demonstrate functional equivalence by matching all possible input-output pairings or by ruling out all values of P such that only one remains feasible.

B. SAT Solving of Camouflaged Circuit

The translation from combinational circuits to CNF clauses by way of Tseitin encoding is widely used in automated test pattern generation (ATPG) [17]. What makes the deobfuscation problem different from ATPG is that gates with unknown logic must be represented in the CNF formula, which is accomplished using the constructs of Fig. 1. Inputs, outputs, and programming vectors, as well as all combinational nodes in the circuit, correspond to specific variables in the CNF formula. Each gate in the circuit corresponds to CNF clauses that constrain the values of the variables in any satisfying assignment. When a known input-output pair is applied to the circuit, unit clauses are added to constrain the appropriate variables. To avoid confusion, we denote *known* input and output values as $\widehat{I}$ and $\widehat{O}$ with various subscripts.

Any input vector and corresponding output vector will only be consistent with a subset of the possible programming vectors, viz. a subset of the possible functions of camouflaged gates. Input-output vector pairs therefore serve to constrain the programming vector. Once the CNF is created and the unit clauses are added, a SAT solver is called to find a satisfying assignment. If the solver succeeds, it produces satisfying assignments to the

Algorithm 1 *Baseline SAT-based Deobfuscation:* 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: $j \leftarrow 0$
2: while $\exists I_j.\ \overbrace{f_j \wedge f'_j \wedge O_j \neq O'_j}^{I_j \text{ distinguishes } P,P'} \wedge \overbrace{\bigwedge_{k=0\ldots j-1} f_k \wedge f'_k \wedge I_k = \widehat{I}_k \wedge O_k = \widehat{O}_k \wedge O'_k = \widehat{O}_k}^{P \text{ and } P' \text{ are feasible}}$ do
3:     Apply $I_j$ to obfuscated circuit, learn true output $O_j$    {I/O pair $(\widehat{I}_j, \widehat{O}_j)$ is a new constraint for model}
4:     $j \leftarrow j + 1$
5: end while
6: $\exists P. \bigwedge_{k=0\ldots j-1} f_k \wedge I_k = \widehat{I}_k \wedge O_k = \widehat{O}_k$    {Find programming assignment $P$ that is feasible with respect to known I/O constraints}
```

Algorithm 2 *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: $F(P, P') \leftarrow \top$
2: while $j$ do
3:     $F(P, P') \leftarrow F(P, P') \wedge f_j \wedge f'_j$ {See Fig. 2}
4:     if $\exists I_j.\ \overbrace{F(P, P')}^{P \text{ and } P' \text{ are feas.}} \wedge \overbrace{O_j \neq O'_j}^{\text{as assumption}}$ then
5:         Apply $I_j$ to obfuscated circuit, learn true output $O_j$    {I/O pair $(\widehat{I}_j, \widehat{O}_j)$ is a new constraint for model}
6:         $I_j \leftarrow \widehat{I}_j \quad O_j \leftarrow \widehat{O}_j \quad O'_j \leftarrow \widehat{O}_j$ {Assumption of $O_j \neq O'_j$ released, add true outputs}
7:     else
8:         $\exists P. F(P, P')$    {Find programming assignment $P$ that is feasible with respect to accumulated I/O constraints}
9:         return $P$
10:    end if
11: end while
```

variables, and contained within this assignment are a specific input vector, and specific programming vectors.

C. Baseline SAT-based Deobfuscation Algorithm

The algorithm of Alg. 1, first formulated by El Massad, Garg, and Tripunitara [15] is the baseline algorithm against which we compare our approach. The 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 [1]. The algorithm executes a loop that continually finds new input and output vectors using SAT queries and an oracle circuit model. After some number of iterations, the constraints accumulated are sufficient to rule out all logical functions except for the one that is the true function of the obfuscated circuit. Each step of the baseline algorithm is explained briefly in the following paragraphs, using terminology that deviates slightly from El Massad's paper [15].

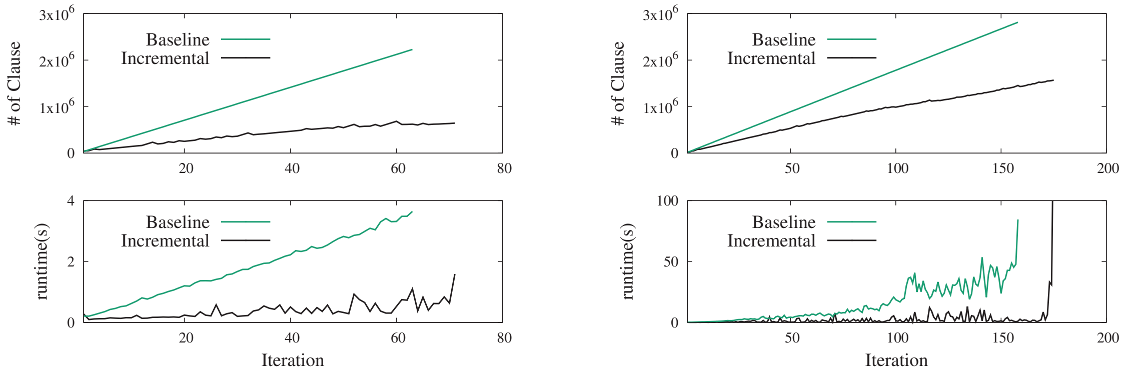
Find Discriminating Input Vector: (Alg. 1 line 2) This step of the algorithm checks whether there exists an input vector I_j that can distinguish two different programming vectors P and P' that are both feasible with respect to all previous input-output examples. Because programming vectors P and P' cause the circuit to map input vector I_j to different output vectors (O_j and O'_j), at most one of P and P' can be consistent with the true output of the obfuscated circuit for input I_j.

Query Oracle: (Alg. 1 line 3) At this step, an input vector I_j is applied to the oracle circuit and the true output $O_j = C(I_j)$ is obtained. This creates an input-output pair $(\widehat{I}_j, \widehat{O}_j)$ that will constrain future candidate programming vectors. Note that the input-output pair $\widehat{I}_j$ and $\widehat{O}_j$ is non-redundant with respect to all previous input-output pairs, and will rule out some programming vector that was previously feasible.

Resolve Circuit Function: (Alg. 1 line 6) The final step of the baseline algorithm occurs after the SAT query at line 2 is unsatisfiable. At this time, it is known that there are no two programming vectors that are consistent with $(\widehat{I}_0, \widehat{O}_0)$ through $(\widehat{I}_{j-1}, \widehat{O}_{j-1})$ while producing different outputs for any additional input vector. Restated, this means that there are no two programming vectors that realize different circuit functions[1] and are consistent with the accumulated input-output examples. However, there must always be at least *one* programming vector that is consistent with the oracle function C, so the algorithm terminates by solving a SAT query to produce a programming vector P that satisfies all accumulated input-output examples. Because all other programming vectors are ruled out, the programming vector produced by the solver is guaranteed to induce a circuit behavior that is functionally equivalent to the oracle, and therefore deobfuscation is successful. Note that there is no guarantee that P agrees with the oracle circuit on a gate-by-gate basis; only the overall logic function of the circuit is guaranteed to be the same.

Remarks on Baseline Algorithm: Foreshadowing our incremental-SAT solution, we make a few remarks on inefficiencies in the baseline SAT-based algorithm (Alg. 1). First, we note that total runtime is dominated by the SAT query at line 2 which is called repeatedly, and not the query at line 6 which is solved once. Second, we note that the size of the SAT problem at line 2 grows linearly with j, the number of iterations of

[1]Note that if P and P' do not produce different output for any input, then they induce the same circuit function even if $P \neq P'$.



(a) NAND/NOR/XOR Obfuscation (c7552) (b) Fully Obfuscated (c1908)

Fig. 3: Number of clauses in the SAT problem solved at each iteration and runtime used to solve it. Incremental solver makes simplifications to the SAT problem this results in a reduction of the number of clauses and SAT solving runtime at each iteration.

the while loop. Third, we note that the SAT problem solved at line 2 in the j^{th} iteration is identical to the SAT problem solved at the $j-1^{st}$ iteration but with two additional copies of the circuit CNF included.

IV. Incremental SAT Deobfuscation Algorithm

We achieve a significant speedup over the baseline algorithm by using incremental-SAT. At each iteration of the algorithm, new constraints are added incrementally to strengthen a persistent CNF formula F, and all learned clauses are carried forward throughout the algorithm as F is strengthened. To maximize the benefits of using incremental-SAT, we use a solver in a way that can eliminate variables and clauses to simplify the CNF formula F as the algorithm progresses.

Our algorithm is based on MiniSat, and the approach used by MiniSat for incremental SAT solving is referred to as solving with assumptions [18]. In this approach, the solver accepts assumption literals when solving a SAT problem, and restricts its search to assignments with those literals. Assumptions are not treated as permanent facts, and they are released once the SAT problem is solved. Furthermore, MiniSat will not learn any clauses that depend on these assumption literals, and all learned clauses derived while solving can safely be carried forward after the assumptions are released.

Our oracle-guided incremental-SAT based algorithm is given in Alg. 2. The algorithm is implemented using a modified version of MiniSat [19] version 2.2.0. Formula F will constrain the allowable values of the programming vectors P and P' and is initialized to $\top$; F will be continually strengthened throughout the algorithm. At the j^{th} iteration of the while loop in Alg. 2, two new copies, f_j and f'_j, of the circuit CNF formula are created and added to the overall formula F (see line 3); adding f_j and f'_j to the problem also adds variables for a new input vector I_j and new output vectors O_j and O'_j respectively (see Fig. 2). We solve for an input vector I_j under the *assumption* that $O_j \neq O'_j$ (see line 4). From within MiniSat, we call the oracle to determine the true output value for I_j, and denote this input-output pairing $\widehat{I}_j$ and $\widehat{O}_j$. Finally the known input-output pairing $\widehat{I}_j$ and $\widehat{O}_j$ is assigned as unit clauses to I_j, O_j and O'_j (line 6), and the next iteration of the loop begins.

We use the MiniSat 2.2.0 (simp) version because it implements variable elimination and simplification before solving. The cost of the simplifications is justified because the simplified formula is carried forward and used many times in the algorithm. 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 input vectors and programming vectors P and P' are frozen to remain in the SAT problem so that we can read their values out from satisfying assignments.

Both the baseline and incremental algorithms perform iterations of a loop and add two new copies of the circuit CNF at each iteration. Figure 3 compares the runtime and number of CNF clauses at each iteration (i.e. at line 2 of Alg. 1, and at line 4 of Alg. 2). Fig. 3a is deobfuscating ISCAS circuit *c7552* with 200 randomly selected camouflaged gates, and each gate can be either NAND, NOR or XOR (see Fig. 1a). Fig. 3b is deobfuscating ISCAS circuit *c1908* with 200 randomly selected camouflaged gates, and each gate can be any 2-input function (see Fig. 1b). In both examples, the incremental algorithm has at each iteration to solve a SAT problem with fewer clauses, because the solver performs simplifications and carries them forward. Additionally, the runtime to solve the SAT problem grows notably with each iteration in the baseline algorithm, but overall grows more slowly when using the incremental algorithm because of the learned clauses and simplifications that are carried forward.

V. Evaluation

We evaluate our deobfuscation algorithm on a set of ISCAS-85 combinational benchmarks [20] including *c432*, *c499*, *c880*, *c1355*, *c1908*, *c2670*, *c3540*, *c5315*, and *c7552*. The sizes of these circuits range from several hundred to several thousand gates. Using the attacker model of Rajendran et al. [1], we allow the attacker to have the following capabilities: (1) knowledge of the logic functions of all non-obfuscated gates, (2) knowledge of which gates are obfuscated and the possible functions they can implement; and (3) the ability to apply input vectors to the camouflaged circuit and observe its corresponding outputs.

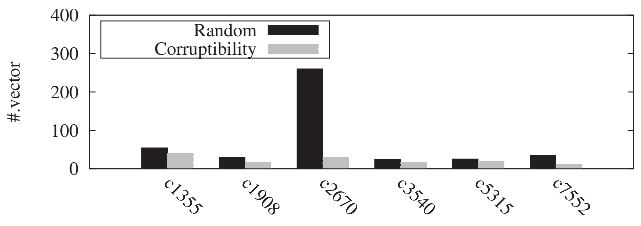


Fig. 4: Guiding camouflaging according to corruptibility reduces the average number of vectors needed for deobfuscation.

Fig. 6 shows the runtime for deobfuscating the six largest ISCAS benchmarks we used when up to 200 gates are camouflaged in a way that each gate could implement NAND, NOR, or XOR. For generality, 10 trials are used with different random choices of gates to camouflage; error bars show the ranges of the results across the 10 trials. A second method considered for selective camouflaging is to select gates to obfuscate such that incorrect guesses of the camouflaged gates will maximize corruption of the primary outputs [14], [1]. We measure corruptibility using the fault simulation tool HOPE [21]. Since the corruptibility for each circuit is fixed, the camouflaging order is also fixed. Each plot in Fig. 6 shows runtime both for randomly camouflaged circuits and circuits camouflaged according to corruptibility. All circuits camouflaged according to corruptibility are deobfuscated in less than 100 seconds. We notice that corruptibility-guided obfuscation generally requires fewer vectors to deobfuscate because maximizing output corruption under wrong guesses has the side effect of ruling out more wrong possibilities with every correct input-output vector pair. The average number of vectors required to deobfuscate each ISCAS-85 circuit are shown in Fig. 4; note that our algorithm does not necessarily minimize the number of vectors.

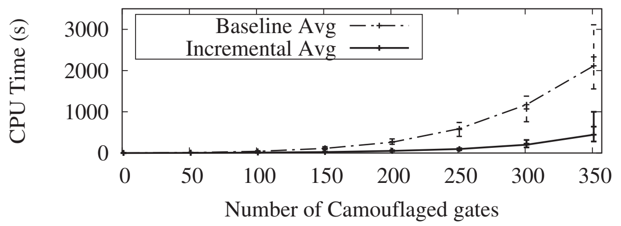


Fig. 5: Runtime of baseline and incremental algorithm on *c5315* with randomly selected camouflaged gates.

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. [15]. We focus first on *c5315* because El Massad gives detailed results of this circuit. Fig. 5 shows the runtime using baseline and incremental algorithm to deobfuscate randomly selected gates that use the NAND/NOR/XOR style of camouflaging. In the case of 350 camouflaged gates, the average runtime using the baseline is 2116 s, and the average runtime using our incremental algorithm is 446 s, for a speedup of almost 5x over baseline on the same computer, and over 20x relative to the absolute runtimes reported by El Massad. Additionally, we consider the case of *c5315* when 20 gates are fully obfuscated and can implement any 2-input function. In comparison with El Massad's observation of the problem taking more 3 hours to solve, across 10 trials using our incremental algorithm we find an average and max runtime of 5.5 s, and 10.8 s respectively. We find that even 200 such gates can be deobfuscated in 264.9 s on average.

For an additional comparison, we repeat all experiments from Fig. 6 using the baseline algorithm, and plot the baseline algorithm runtime against the incremental algorithm runtime for the same experiments. This comparison shows a typical speedup of around 5x (see Fig. 7).

VI. Conclusion

This paper proposes a novel incremental-SAT based algorithm for deobfuscating camouflaged circuits. We have implemented the algorithm and tested its performance by using it to deobfuscate ISCAS-85 combinational benchmarks camouflaged randomly and according to output corruptibility, and using two different styles of camouflaged gates. Our methods are shown to perform very well, providing a speedup of at least 5x over the work of El Massad [15], which is the best result demonstrated until now. Our approach sets a new mark for deobfuscation capability that must be considered in future works on selective camouflaging.

References

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

[2] Chipworks, "Reverse Engineering Software," http://www.chipworks.com/en/technical-competitive-analysis/resources/reerse-engineering-software, 2014, online; accessed 21-April-2015.

[3] K. Nohl, D. Evans, Starbug, and H. Plötz, "Reverse-Engineering a Cryptographic RFID Tag." in *USENIX security*, vol. 28, 2008.

[4] R. Torrance and D. James, "The State-of-the-art in Semiconductor Reverse Engineering," in *Proceedings of the 48th Design Automation Conference*, ser. DAC '11, 2011, pp. 333–338.

[5] W. Li, Z. Wasson, and S. A. Seshia, "Reverse engineering circuits using behavioral pattern mining," in *Hardware-Oriented Security and Trust (HOST), 2012 IEEE International Symposium on*, 2012, pp. 83–88.

[6] P. Subramanyan, N. Tsiskaridze, K. Pasricha, D. Reisman, A. Susnea, and S. Malik, "Reverse engineering digital circuits using functional analysis," in *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 2013, pp. 1277–1280.

[7] A. Gascón, P. Subramanyan, B. Dutertre, A. Tiwari, D. Jovanovic, and S. Malik, "Template-based circuit understanding," in *Formal Methods in Computer-Aided Design (FMCAD), 2014*. IEEE, 2014, pp. 83–90.

[8] SypherMedia, "SypherMedia Library – Circuit Camouflage Technology," http://www.smi.tv/SMI_SypherMedia_Library_Intro.pdf, 2012, online; accessed 21-April-2015.

[9] R. P. Cocchi, J. P. Baukus, L. W. Chow, and B. J. Wang, "Circuit camouflage integration for hardware IP protection," in *Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE*. IEEE, 2014, pp. 1–5.

[10] G. T. Becker, F. Regazzoni, C. Paar, and W. P. Burleson, "Stealthy dopant-level hardware trojans," in *Cryptographic Hardware and Embedded Systems-CHES 2013*. Springer, 2013, pp. 197–214.

[11] 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.

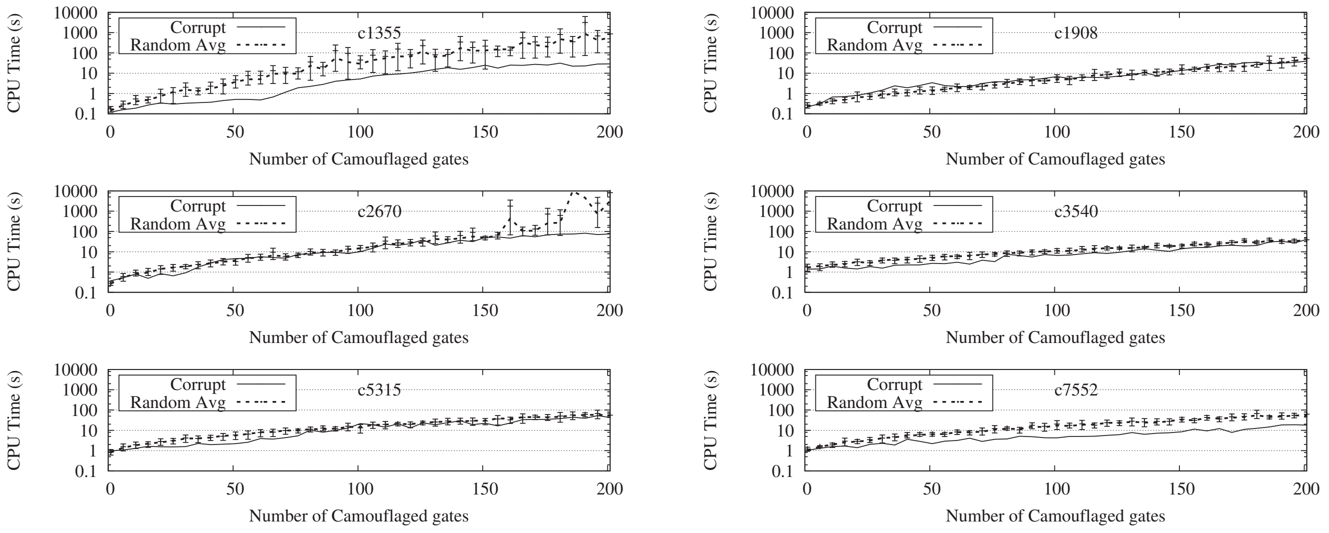
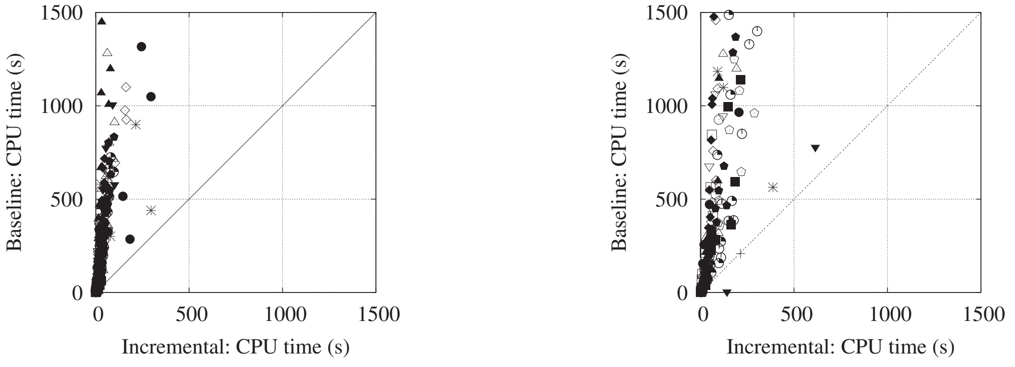


Fig. 6: Runtime to deobfuscate six ISCAS-85 benchmarks with randomly obfuscated gates, and gates chosen for obfuscation based on output corruptibility. Error bars mark the minimum and maximum observed over 10 random trials.



(a) NAND/NOR/XOR Camouflaging

(b) Fully Camouflaged

Fig. 7: Comparing the total deobfuscation runtime of baseline and incremental algorithms using two different styles of camouflaged gates. The position of each datapoint indicates its runtime for the incremental solver and runtime for the baseline solver. Incremental solver gives a speedup of around 5x. Runtimes that exceed 1500 seconds are truncated from the plot.

[12] S. Malik, G. Becker, C. Paar, and W. Burleson, "Development of a Layout-Level Hardware Obfuscation Tool," in *IEEE Computer Society Annual Symposium on VLSI, ISVLSI 2015*, 2015.

[13] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Logic Encryption: A Fault Analysis Perspective," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '12, 2012, pp. 953–958.

[14] R. S. Chakraborty and S. Bhunia, "HARPOON: an obfuscation-based SoC design methodology for hardware protection," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 28, no. 10, pp. 1493–1502, 2009.

[15] M. E. Massad, S. Garg, and M. V. Tripunitara, "Integrated Circuit (IC) Decamouflaging: Reverse Engineering Camouflaged ICs within Minutes," in *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2014*, 2015.

[16] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the Security of Logic Encryption Algorithms," in *Hardware-Oriented Security and Trust (HOST)*, 2015.

[17] T. Larrabee, "Test pattern generation using Boolean satisfiability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 1, pp. 4–15, 1992.

[18] N. Een and N. Sorensson, "Temporal Induction by Incremental SAT Solving," *Electronic Notes in Theoretical Computer Science*, vol. 89, no. 4, pp. 543–560, 2003.

[19] N. Een and N. Sörensson, "An extensible SAT-solver," *Theory and Applications of Satisfiability Testing*, 2004.

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

[21] H. K. Lee and D. S. Ha, "HOPE: An efficient parallel fault simulator for synchronous sequential circuits," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 15, no. 9, pp. 1048–1058, 1996.