Examining Tradeoffs for Hardware-Based Intellectual Property Protection

J. Todd McDonald[1] and Yong C. Kim[2]
[1]University of South Alabama, Mobile, AL, USA
[2]Air Force Institute of Technology, Wright Patterson AFB, USA
jtmcdonald@usouthal.edu
ykim@afit.edu

Abstract: The ability to protect critical cyber infrastructure remains a multi-faceted problem facing both the commercial sector and the federal government. Hardware intellectual property (IP) embedded within application-specific integrated circuits and programmable logic devices are subject to adversarial analysis in the form of subversion, piracy, and reverse engineering. We consider the effect of transforming the programmatic logic or netlist definitions for such environments so that malicious adversaries are hindered or prevented from recovering original, higher level abstractions of combinational logic design. In this paper, we provide observations on obfuscating algorithms that use random and deterministic techniques to transform logic-level definitions into alternative, functionally equivalent forms. We define the tradeoff space for both types of techniques and show how limitations have driven research methods.

Keywords/Key Phrases: Circuit protection, malicious reverse engineering, obfuscation, security research methodologies

1. Introduction

Semiconductor integrated circuits are integral in commercial and military systems. Intellectual property (IP) protection for hardware-based circuit programs are often supported by government policy and legislation, but the cost of piracy and compromise is borne primarily by creators of IP themselves. At one time, vendors concluded (wrongly) that programmatic logic, particularly encryption algorithms, released solely in hardware implementations were secure because reverse engineering from the silicon level and above was prohibitively expensive for typical adversaries. Recent history demonstrates that topological recovery of gate-level logic descriptions from physical hardware samples does not require a large capital investment (Nohl *et al*. 2008).

System-on-chip (SoC) design methodologies have become prominent over the last decade, allowing designers to build electronic systems from virtual components that are connected in specific, higher-level design topologies (Robert *et al.* 2002). This methodology has promoted extensive design reuse of intellectual property cores produced from a market of vendors and is driven by reduced time to market and higher productivity. As a result, synthesized hardware designs most often follow highly structured and predictable patterns in terms of their layout and component configuration. With no obscuring or tamper-proofing in place, reverse engineering has become economical for a larger number of less-sophisticated adversaries who are driven by monetary gain or nefarious purposes. To add to this problem, reprogrammable hardware devices such as Field Programmable Gate Arrays (FPGAs) are becoming the predominant mechanism for embedded system architectures. Hardware level design specifications are now roughly equivalent to our traditional notion of “software” because FPGAs can be reprogrammed easily (Kim and McDonald 2009).

As Charles Reich once said (Reich, 1995), “What we do not understand, we cannot control.” This applies equally to intellectual property protection and reverse engineering. Given easier access to programmatic logic in hardware systems (through increasing prevalence of FPGA realizations) and the relative ease for recovering gate-level logic relationships (a by-product of the SoC paradigm), adversaries have great power to reverse engineering higher level component relationships and understand how circuit-based programs work. Once understanding is gained, control follows in the form of illegal copying, tampering, and compromise.

In order to deter and frustrate malicious adversaries in this environment, a handful of methods can be used for protection purposes. Three traditional categories have arisen: watermarking (prevention of piracy), tamper-proofing (prevention of subversion), and obfuscation (prevention of reverse engineering). *Obfuscation* has been a standing theoretic question of interest to the cryptography community for some time. Researchers have formalized the highest level of protection as a simulation-based virtual black box and have produced counterexamples that demonstrate no general *perfect* obfuscator exists (Barak *et al.* 2001). In practice, however, the question of protection reduces to whether something is better than nothing at all. We can answer this question by considering specific attack vectors (analysis methods used by an adversary) and defining techniques which prevent or hinder such analysis. Protection then becomes a tradeoff between performance and the efficacy of the methods used (whether an adversary can recover original design information from an obfuscated variant).

In this paper, we consider obfuscation techniques for combinational logic synthesis solutions and define their trade-off space. We first define methods and metrics associated with protection and delineate limitations along deterministic and pseudo-random lines. We provide observations from several years of research in building practical obfuscation algorithms and the problems of defining what, if any, security is gained from their use.

2. Obfuscation Metrics

Many techniques exist which prevent adversaries from obtaining a circuit-level gate netlist from a physical hardware, silicon-based program. We assume that an adversary has already accomplished these necessary physical analysis steps of reverse engineering and has successfully recovered a combinational logic description of the code. Given this starting point, adversaries gain further program understanding by analyzing structure, dynamic signal propagation, and functional characteristics of the circuit.

We represent such circuits as directed acyclic graphs (DAGs) that consist of Boolean logic gates as vertices in a connected graph. A basis set $\Omega \in \{AND, OR, XOR, NAND, NOR, XNOR, NOT\}$ defines the typical collection of logic gates at this level and allows us to define a circuit C as a DAG having nodes (*intermediate gates*) mapping to functions in Ω or having *input* nodes with in-degree 0. We define *outputs* of the circuit by distinguishing one or more intermediate gates. The domain set δ_Ω is a set of all possible circuits derivable from the basis Ω.

By convention, an obfuscating algorithm, $O: \delta_\Omega \times \mathbb{N} \rightarrow \delta_\Omega$, is a publically known, polynomial time algorithm that takes as an argument circuit, $C \in \delta_\Omega$, and a key, $k \in \mathbb{N}$, representing the specific sequence of transformations accomplished. O returns a functionally equivalent version $C' \in \delta_\Omega$. We can partition the domain set δ_Ω into subsets based on circuits with n inputs and m outputs. Circuit-related features (gate size, number of levels, controlling pathways, etc) provide further subset delineation. A circuit family $\delta_{n\text{-}m\text{-}S\text{-}\Omega}$ is the set of all circuits with input size n, output size m, maximum intermediate gate size S, and basis Ω. We let δ_C represent a subset of circuits that compute the same function, $f_C: \{0,1\}^n \rightarrow \{0,1\}^m$, as circuit C. Figure 1 depicts our domain of interest $\delta_C \subseteq \delta_{n\text{-}m\text{-}S\text{-}\Omega} \subseteq \delta_\Omega$. We visualize the essential operation of algorithm O as a polynomial time application which chooses variants from the family δ_C when given a circuit C and creates some finite distribution of equivalent circuits on repeated operations, $\{C'_1, C'_2, C'_3, \ldots\}$.

In order to defeat a malicious adversary, we look for opportunities to defeat any and all phases of the reverse engineering process from the logic level and above. The goal of reverse engineering is to recover higher levels of abstraction from lower level representations. We define four categories of abstraction that represent primary attack vectors for achieving program understanding of combinational logic designs. We let $D: \delta_\Omega \times O_P \rightarrow \delta_\Omega$ represent a polynomial time adversary algorithm that takes a circuit (C') and knowledge of the obfuscating algorithm O_P to

produce a variant in some reduced, more abstract form. The powers of any given *D* define the metrics for feature hiding that are associated with the reverse engineering process.

2.1 Topology Recovery

Given the DAG for a circuit *C*, an adversary accomplishes topology recovering by reproducing the original node and vertices configuration of $C \in \delta_C$ when given some variant $C' \in \delta_C$. In most cases, topology recovery of the original circuit is not a primary adversarial goal. However, topology recovery (and therefore topology hiding) in smaller, sub-circuit contexts make recovery of other abstract relationships possible. Figure 2 illustrates adversary *D* which, when given a variant *C'* and the obfuscating algorithm O_P, produces a variant $C'' \in \delta_C$ with higher abstracts relationships (in this case, topology) relative to *C'*. We measure the effectiveness of *D* in terms of topology by comparing the logic gate configuration reflected in the DAGs for *C* and *C"*.

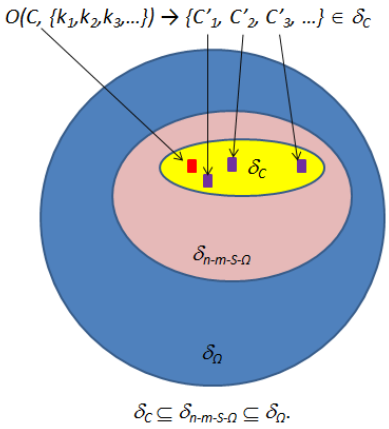

Figure 1: Given circuit *C*, *O* produces a distribution of variants (C'_1, C'_2, C'_3, …) chosen from a functionally equivalent circuit family δ_C, keeping secret the specific obfuscating transformations used to produce each variant (k_1, k_2, k_3, …)

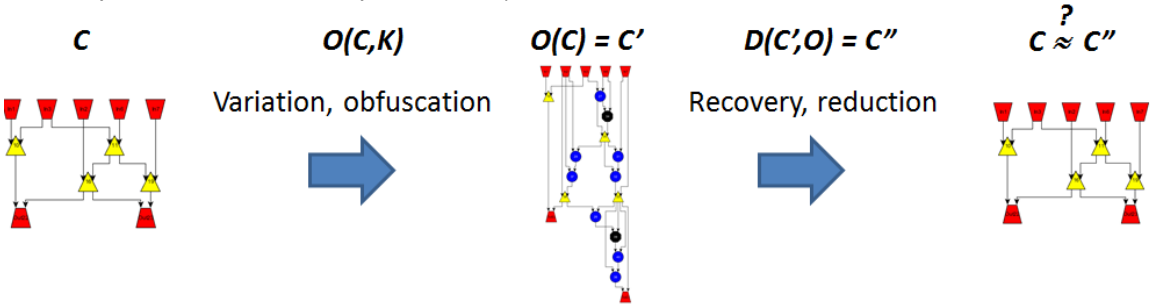

Figure 2: Topology recovery for a trivially obfuscated variant

2.2 Signal Recovery

We define a signal as the truth table value (0,1) for an intermediate gate within the graph of circuit, given a certain input set. The truth table for a circuit is the collection of all possible input bit strings of length *n* ($\{0,1\}^n$), all possible intermediate gate bit patterns (the 0 or 1 outputs of each gate given a specific input string), and the associated output bits produced by the circuit. Figure 3 illustrates a combinational level circuit description and its corresponding full truth table and highlights the concept of an execution trace where each gate produces a signal given a specific input signal combination. Output signals are distinguished intermediate gates and, together with the full set of corresponding input signals, constitute the full semantic definition of the overall circuit. Enumerating the full truth table is intractable because of the exponential (2^n) blow-up in

truth table size, but choosing a smart, tractable, set of input vectors provides a reasonable alternative. An adversary may look for specific signals when performing reverse engineering by using a combination of static and dynamic analysis, with the hope that known functional signals may reveal higher level of abstractions such as control flow or groupings of gates. Signal recovery may be measured by whether or not an original signal from the collective execution traces of an original circuit *C* are also present in the obfuscated variant *C'*. Figure 4 illustrates signal recovery from a variant of the circuit seen in Figure 3.

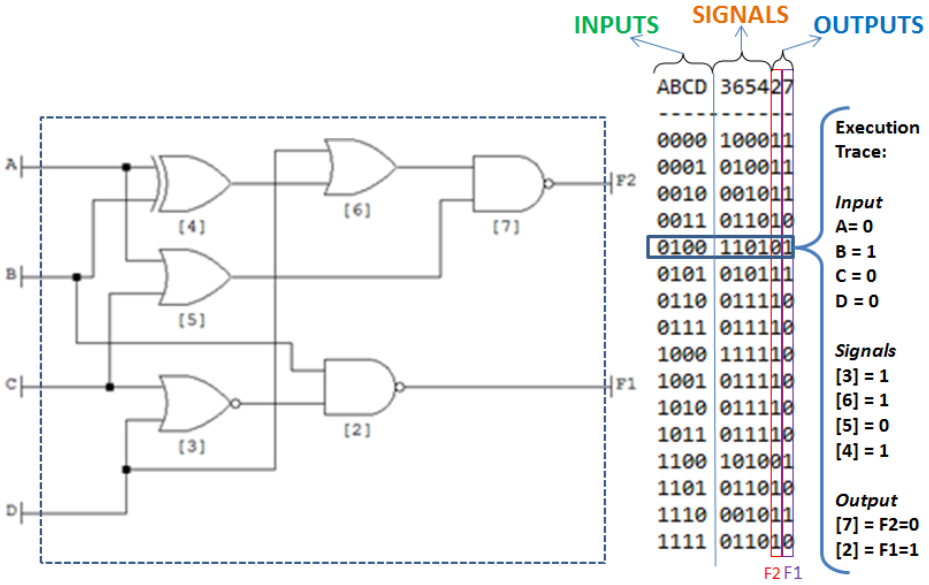

Figure 3: Full truth table for combinational circuit illustrating inputs, outputs, and signals

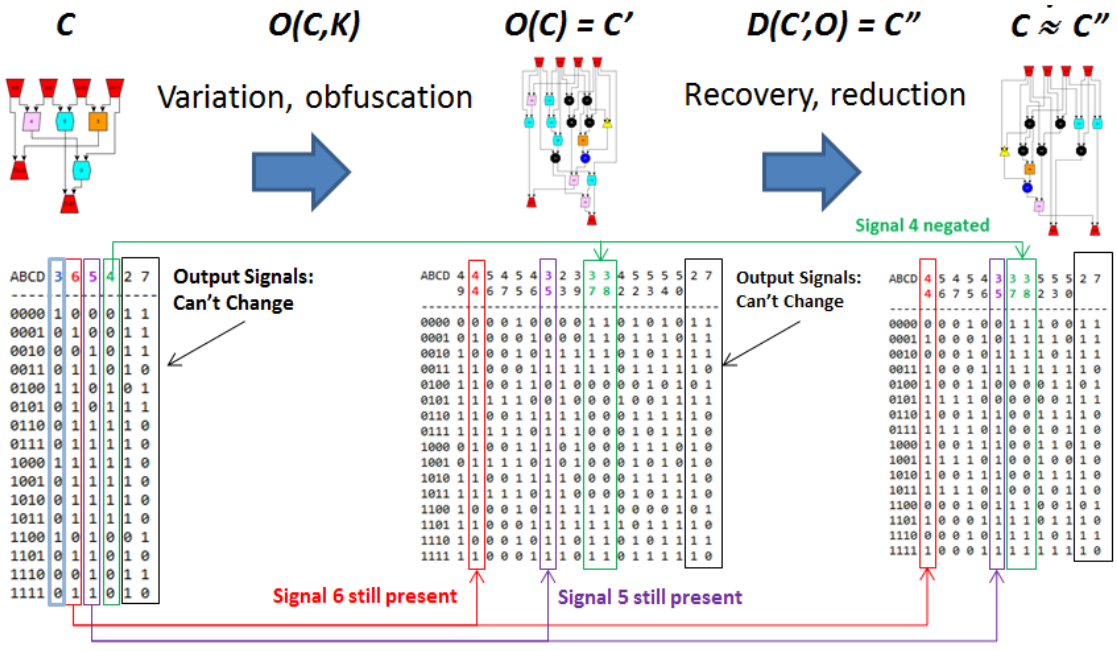

Figure 4: Signal recovery from an obfuscated variant of a circuit *C* (from Figure 3)

Figure 4 illustrates that the variant *(C')*, after adversarial logic reduction (*C"*), still contains two original (5,6) and one negated (4) intermediate gate signal found in the original *C*. Signal hiding has been accomplished only for gate 3 since there are no traces which match its pattern in the truth table of the variant. Figure 4 also illustrates that adversarial algorithm *D* does not recover the original topology of *C* from *C"*.

2.3 Component Recovery

Building a circuit from known components is a typical hardware synthesis exercise. Typical components include multiplexors, adders, decoders, and a variety of vendor-specific technologies and components represent distinct functionalities that build upon other components. We define a component as a subgraph with an input/output pattern that is well known. A circuit *C* can be decomposed into a smaller set of combinational subcircuits $s = \{s_1, s_2, \ldots, s_n\}$ that are used to individually implement sub-functions within a circuit. Adversarial analysis at this level of abstraction involves two distinct problems: 1) enumerating the possible candidate subgraphs that might represent components from the DAG of a given circuit, and 2) matching candidate subcircuits to known component functionalities.

Adversarial reverse engineers gain program understanding by using topology and signal relationships combined with identified components. Component identification represents an extension of signal recovery because components have distinct output patterns; likewise, it represents an extension of topology recovery because components known configurations with clear input and output boundaries. Component identification and netlist partitioning are well studied problems that form the basis of most adversarial recovery techniques. An adversary is successful in component recovery based on the number of original components they can identify in an obfuscated variant. Figure 5 illustrates a hard-case component-configuration circuit (*C*) whose obfuscated variant (*C'*) requires further component identification but whose reduced version (*C"*) clearly reveals original component configuration.

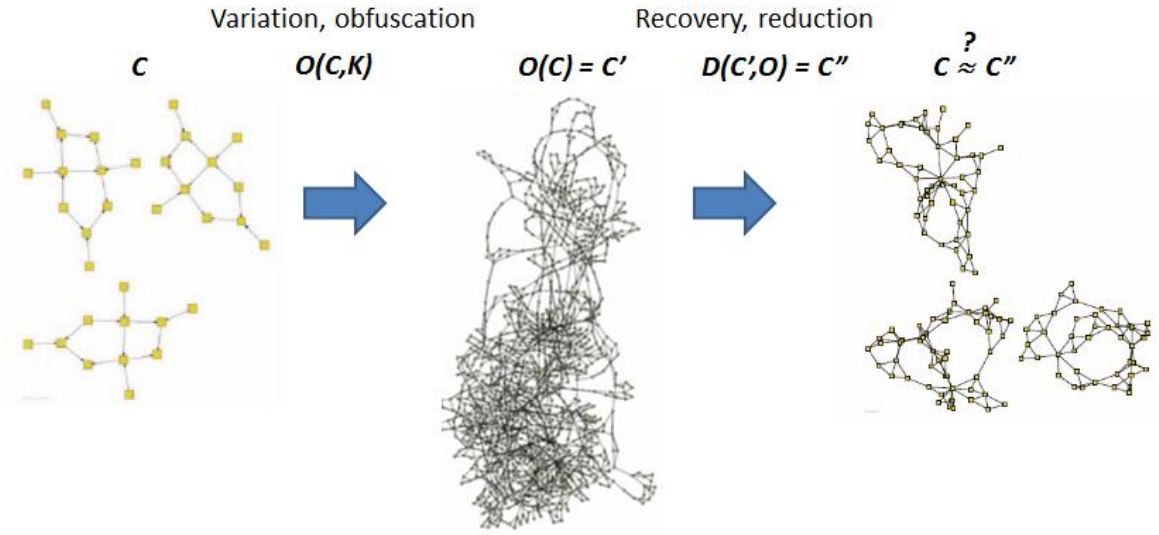

Figure 5: Component recovery from a variant of circuit *C* with 3 independent components

3. Obfuscation Techniques

Topology, signal, and component recovery uses heuristic methods that normally sacrifice optimal solutions for efficiency. Likewise, adversaries use Boolean logic reduction techniques to normalize obfuscating transformations and enable other recovery heuristics even though precise reduction is not tractable in circuits with larger input/output spaces. We define now two major categories of obfuscation techniques which produce circuit variation: random and deterministic.

3.1 Random Selection and Replacement Obfuscation

Obfuscating algorithms that use selection and replacement follow a simple algorithm. As Figure 6 illustrates, our methodology involves selecting a sub-circuit based on the DAG of an original circuit (C_{sub}) and replacing it with another sub-circuit that is semantically equivalent (C_{rep}), creating a distribution of intermediate forms until a final variant *C'* is produced. The algorithm terminates after some number of iterations of selection/replacement or when some particular goal is met.

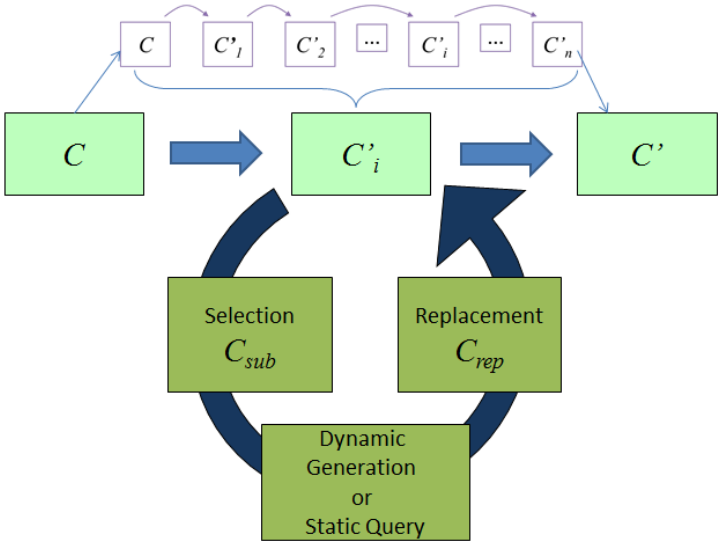

Figure 6: General selection and replacement methodology

Our algorithm incorporates random variability in how sub-circuits are chosen. The number of gates can be chosen randomly from 1 to 3. Likewise, the size of the replacement subcircuit can vary from 1 to 3 gates larger than the selection size. Five different strategies are used to choose gates from different levels of the circuit: 1) output level; 2) fixed, predetermined level; 3) current largest level in the circuit; 4) random level; and 5) completely random choice for all gates. The algorithm can choose a single strategy or pick one randomly from the five possibilities. The algorithm can also incorporate original component information and choose gates from different components. Instead of choosing a fixed number of total iterations, we can also configure the obfuscator to continue until a certain goal is met such as requiring all original gates to be replaced at least once. The method for generating a replacement follows two lines, which we discuss next.

3.1.1 Replacement from Fully Uniform, Equivalent Choices

Our traditional method for choosing a replacement involves generation of circuit libraries based on specific input, output, and gate size configurations. For each sub-circuit C_{sub} chosen during an iteration, the associated circuit family for the sub-circuit is defined based on the number of inputs *x*, number of outputs *y*, number of gates *s*, and gate basis Ω ($C_{sub} \in \delta_{x\text{-}y\text{-}s\text{-}\Omega}$). Given a gate-size increment of *k*, the replacement circuit is chosen from the family of circuits $C_{rep} \in \delta_{x\text{-}y\text{-}(s+k)\text{-}\Omega}$, with the stipulation that C_{rep} also belongs to the functional family of the selected sub-circuit C_{sub}. The circuit library $\delta_{x\text{-}y\text{-}(s+k)\text{-}\Omega}$ may be either generated out of band (before the obfuscator runs) or dynamically at runtime. For small circuit families of 1 to 3 gates, in-memory, dynamic access is comparatively efficient to disk access. By creating all possible circuit configurations for a given sub-circuit family, the algorithm can make a uniform, random choice from the set of all possible circuits that define a family.

Several creation options allow variability in the properties of constituent circuits. Six options

govern whether libraries will have circuits with degenerate gate conditions (gates that are non-standard in optimal digital logic design). Creation of replacement of circuits with gates from a limited basis Ω set (for example, NAND-only circuits) is also possible. The six different options for a circuit library include: 1) *SymmetricGates:* Should we consider a gate with inputs (*X1,X2*) as equivalent to a gate with inputs (*X2,X1*)? 2) *RedundantGates*: Should we allow gates that are identical to other gates based on their function? 3) *AllowConstants*: Should we allow the circuit immediate access to the constants True and False? 4) *DoubleInputs*: Should we allow both inputs to a gate to originate from the same place? 5) *ExactCount*: Does the set contain all circuits within a certain size bound or only all circuits of an exact size? and 6) *SimpleOutputs*: Defines which gates may be outputs. This option requires outputs of the circuit to be lowest level sinks in the graph of a circuit.

3.1.2 Replacement Based on Boolean Logic Expansion

As an alternative to using gate structure as the basis to find functional replacements, we can also use a replacement system based upon Boolean algebra laws. Standard characteristics of logic systems include *interpretation* (the assignment of meanings to formulas and their elements), *soundness* (the notion of validity) and *completeness* (the notion of semantic consequence). Sound algebras guarantee that identities which equate an expression with any other expression preserve the semantics of the expression. Complete algebras guarantee that any two semantically equivalent expressions are actually equivalent using identities (or rather, all valid equalities can be derived using the axioms of the algebra). Since Boolean algebra identities are both sound and complete: 1) applying Boolean algebra identities will always yield other semantically equivalent expressions, and 2) there is some sequence of Boolean algebra identities which can be used to transform one Boolean algebra expression into any other equivalent expression. A complete algebra includes the following axioms based on the operations AND, OR, and NOT:

1) *Idempotence*: ($A * A = A + A = A$)
2) *Commutativity* ($A * B = B * A$; $A + B = B + A$)
3) *Associativity* ($A * (B * C) = (A * B) * C = A*B*C$; $A + (B + C) = (A + B) + C = A+B+C$)
4) *Absorption*: ($A * (A + B) = A + (A * B) = A$)
5) *Distributivity:* ($A * (B + C) = (A * B) + (A * C)$; $A + (B * C) = (A + B) * (A + C)$)
6) *0 and 1:* Universal bounds (0 or the empty set) and *I* (1 or the universal set) must exist, such that $0 * A = 0$; $0 + A = A$; $1 * A = A$; $1 + A = 1$
7) *Complementation*: A unary operation (NOT, ‘) must exist, such that: $A * A' = 0$; $A + A' = 1$
8) *Involution:* $(A')' = A$
9) *DeMorgan’s Theorem:* based on other identities, $(A+B)' = A' * B'$; $(AB)' = A' + B'$

Functions expressed in the form of Boolean algebra can be transformed into other equivalent expressions through the repeated application of Boolean algebra identities. Replacement strategies that use Boolean algebra laws *explicitly* accomplish what strategies using actual gate configurations *implicitly* accomplish. The replacement algorithm creates a logical expression from a given sub-circuit selection and then applies one or more Boolean equivalence laws to them (based on a random choice), returning a circuit structure derived from the logical expression. Figure 7 illustrates a gate structure and the application of Boolean laws to create a replacement gate structure.

3.2 Deterministic Obfuscation Approaches

Deterministic approaches to obfuscation are characterized by algorithms that target specific hiding properties. These algorithms create circuit characteristics which provide an adversary greater chance to identify variant features and therefore reverse or reduce them. On the other hand, combinations of overlapping random selection and replacement make rule-based identification and reversal prohibitive because the rule space grows too large. Examples of deterministic algorithms include introduction of opaque predicates (if-then-else logic that is always true or false), targeting removal of specific signals, diffusion of control flow across multiple parts

of the circuit, and targeting removal of component boundaries. Random approaches may produce the same results, but provide no guarantee that topology, signal, or component recovery are hindered or prevented.

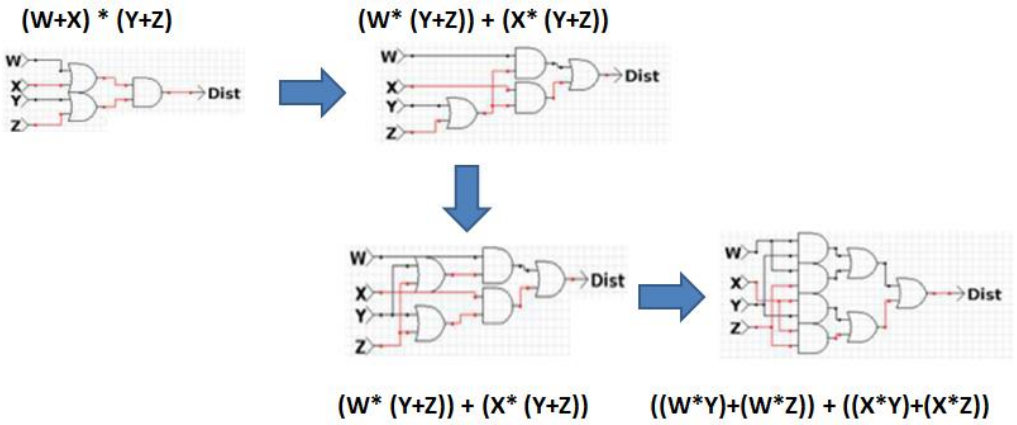

Figure 7: Replacement circuit created by repeatedly applying the Distributive law

Because of the critical role that components play in standard System-on-chip design methodologies, we target hiding of component boundaries using three different deterministic approaches. Each approach incorporates some degree of randomness, but follows a specific method for changing the component relationships between gates within the circuit. In each case, the overall semantics of the circuit are preserved.

3.2.1 Boundary Blurring

Boundary blurring is based on targeting output gates of known components that are part of an original circuit *C* in an attempt to change the input/output patterns. The algorithm uses a partitioned set of gates that represents original component configurations. For each iteration *i*, the algorithm chooses a component s_i and an output gate within the component, $g_{orig.}$ The algorithm changes the gate type of g_{orig} randomly, and, at some *w* number of levels closer to the output of the circuit, the algorithm chooses a recovery gate g_{rec}. The algorithm then implements one of two methods: 1) a *multi-level blur* which adds new combinational logic based on signals of the gates g_{orig} and $g_{rec.}$; or 2) a ***don't care blur*** which creates new combination logic based on randomly chosen signals from other circuit locations. Once the combinational logic is built, g_{orig} and all gates between it and g_{rec} are replaced with the new logic.

3.2.2 Component Fusion

Component fusion ensures replacement of the entire circuit and also requires partitioning the circuit into known component subcircuits. During each iteration *i*, the algorithm selects a known component s_i and selects a set of gates from adjacent components, g_i, to form a sub-circuit m_i. The technique then makes two choices: 1) a random gate basis Ω; and 2) a random Product of Sum/Sum of Products canonical form. It then applies ESPRESSO's Quine McCluskey algorithm to logically reduce the component, replacing the selected sub-circuit m_i with a new version m_i'.

3.2.3 Component Encryption

Component encryption targets specific components for protection directly by specifically changing the wires and signals that join two components. The technique involves concatenating a permutation cipher (encryption) at the output boundary of one component and prepending a recovery permutation (decryption) at the input boundary of the connected component. Once logic for the permutations is inserted, the modified sub-circuit is synthesized using a standard Quine McCluskey reduction. The algorithm identifies eligible components into a candidate set of signals

and then generates encoding and decoding functions for these internal signals to produce new components. New combinational logic is inserted to reconnect components, continuing until all candidates are transformed.

Figure 8 illustrates the high level concepts of deterministic component hiding techniques: boundary blurring (a) targets a gate, replaces it, and then incorporates recovery logic at some point lower in the circuit toward the output; component fusion (b) selects a component and some number of predecessor gates from previous components, replacing the selected sub-circuit with a minimized version; component encryption (c) selects al component signal and wire configurations and replaces all logic with synthesized versions that incorporate encryption and decryption logic.

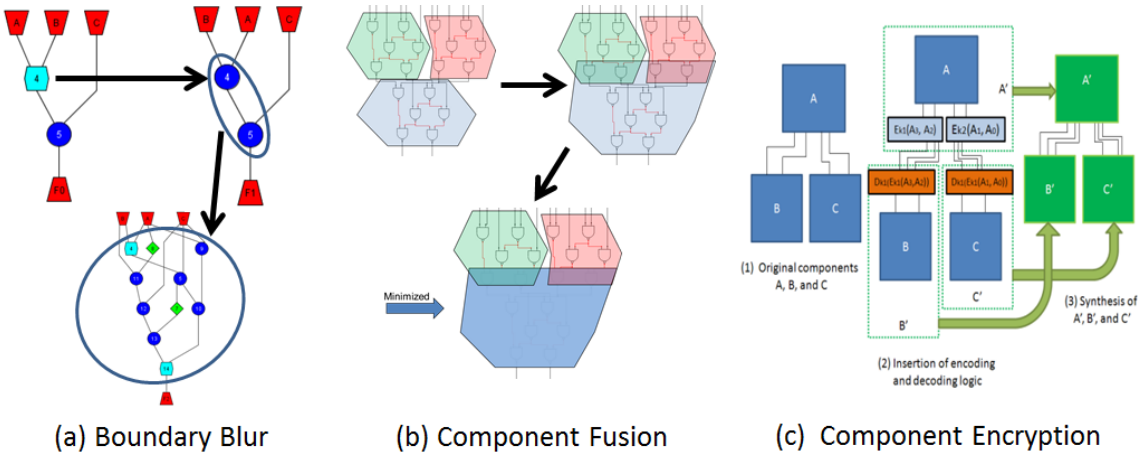

Figure 8: Deterministic component hiding techniques

4. Limitations and Trade-off Space

After several years of investigating candidate obfuscation algorithms *O* and measuring their measured effect on deterring topology, signal, and component recovery, we have encountered limitations and observe two distinct trade-off spaces as a result. Despite the overall theoretic limitation that no general, efficient obfuscator can provide *perfect* (virtual black box) protection, we have focused on whether achievable security in limited contexts can be achieved. In this regard, we define two goals that at times are antithetical to each other.

4.1 Maximizing Randomness in the Obfuscation Algorithm

On one hand, we desire *O* to be a publically known algorithm that randomly mutates a circuit to produce a functionally equivalent variant. This principle is consistent with Kerckhoff's assertion (Kerckhoff 1883) that the security of a cryptosystem should not rely on secrecy of the algorithm, but rather, secrecy of the key (in our case, the specific random choices made by the obfuscator in creating a variant). Intuitively, the greater the selection space defined by an obfuscator, the less advantage an adversary has to create pattern and rule-based approaches that would reverse or undo transformations. To maximize randomness (and therefore security) of the obfuscator, we would prefer *O* to make a uniform, random choice from the set of all possible circuits that are functionally equivalent to *C* and that are within some reasonable size (number of gates or levels) of *C*. Such sets are not tractable to create by the method we describe in section 3.1.1 where we enumerate all possible circuit configurations for a given input/output and gate size class. Though we cannot *choose* an obfuscated variant by such a method, we have postulated whether we can *approach* a uniform, random distribution of variants by making uniformly random small-scale structural changes in an iterative process, which section 3.1.1 describes.

Even for small-scale replacement libraries, however, we approach intractability limits quickly. An important result produced by Simonaire (2008) was the empirical measurement of the size of certain circuit families through enumeration. These enumerated circuit families were enumerated by input count, output count, and gate count, without regard to the function computed by any particular circuit. The circuit families consisting of 1-input, 1-output, n-gate circuits (the δ_{1-1-n} family) constructed from a one-gate basis were found to be identical in size to corresponding integers in the AT&T Research Labs series A000366. This correlation provided one of our first theoretical results quantifying the cost of circuit family enumeration. The integer series A005439 (counting the number of Boolean functions of *n* variables whose reduced-order binary decision diagram contains at least *n* branch nodes, one for each variable) is equivalent to A000366 multiplied by 2^{n-1}, and Simonaire's empirical count of circuit family constructed using a six-gate basis (AND, OR, NAND, NOR, XOR, and XNOR) is equivalent to A000366 multiplied by 6^{n-1}. This is because the counting of possible BDD configurations correlates to the counting of gate families based on only two possible gate identities. The first seven values of these three integer series are shown in Table 1.

Table 1: AT&T Research Lab numeric series for Boolean functions

g	1	2	3	4	5	6	7
A000366	1	2	7	38	295	3098	42271
A005439	2	8	56	608	9440	198272	5410688
A000366 * 6^{n-1}	6	72	1,512	49,248	2,293,920	144,540,288	11,833,174,656

Other empirical studies from creating circuit libraries for replacement bear out the exponential limits in disk space and time. As input, output, and gate size increase, tradeoffs also emerge in terms of indexing the circuits by their input/output function (which we use to choose replacements): static generation time is increased by a polynomial order when indexing is done before obfuscation runtime, but runtime of the obfuscator is significantly decrease for each iteration if a library has to be queried for functionally equivalents at run time. We have used different flat file, database, and persistent object configurations for storage mechanisms, each bringing limitations in terms of access time and storage limits for the libraries we wish to create. Table 2 summarizes size for a small δ_{2-1-n} library with gate size up to 6 and Table 3 shows disk space limitation we face for circuit families with 5 gate circuits (based on a 4-gate selection which normally has 8 inputs).

Table 2: Number of circuits, creation time, and disk size for 2-input, 1-output circuit libraries

In	Out	Sz	Basis	Generation	Selection	Replacement	# of Circuits	# of Signatures	Creation Time (min)	Size (bytes)	Size (GB)
2	1	1	6GATE	FFFTTT	1-Gate	2-Gate	6	12	0.00	91	0.000
2	1	2	6GATE	FFFTTT	1-Gate	2-Gate	64	12	0.00	1,880	0.000
2	1	3	6GATE	FFFTTT	1-Gate	3-Gate	1,132	14	0.00	49,354	0.000
2	1	4	6GATE	FFFTTT	1-Gate	4-Gate	27,032	14	0.02	1,563,052	0.001
2	1	5	6GATE	FFFTTT	1-Gate	5-Gate	837,924	14	0.86	60,374,270	0.056
2	1	6	6GATE	FFFTTT	1-Gate	6-Gate	31,917,464	14	47.07	1,914,290,859	1.783

Table 3: Disk space for replacement libraries based on selection and replacement gate size

Transformation Library	DB Size
1 to 2 Gates	23.7 KB
2 to 3 Gates	53.6 MB
3 to 4 Gates	166.9 GB
4 to 5 Gates	934.9 TB

4.2 Maximizing Security of Obfuscated Variants

As another tradeoff space, we desire the distribution of circuits produced by *O* to demonstrate security properties of interest: a variant should reveal less information in regards to recovery analysis vectors described in section 2. Whereas our first trade-off space defines the quality of the *obfuscator* itself, the latter distinction defines the properties of the circuit *variants* we want the obfuscator to produce. Random approaches do not necessarily guarantee security properties in variants, but empirical studies have demonstrated that some variable choices are better than others in manifesting hiding properties. For example, using a random algorithm, 2-gate selection, 4-gate replacement method manifests hiding properties in variants more than a random single-gate selection, 2-gate replacement does. Because replacement libraries are limited in size, random-based algorithms are limited in the possible variants reachable. In other words, the incremental circuit forms produced during the variation process (seen in Figure 6) may cause other viable circuits within the possible set of replacements to no longer be reachable (seen in Figure 1).

As a result of these limitations, we have considered different methods to create replacements that are more deterministic in nature. The deterministic methods described in section 3 represent our initial work along these lines (McDonald *et al.* 2011). Such methods target the recovery processes involved in reverse engineering. The tradeoff, of course, is that adversaries are more likely to use knowledge of these methods and target the features that are produced as artifacts that appear in circuit variants. The search to find methods that would support larger sub-circuits for replacement led us to consider the Boolean algebra approach described in section 3.1.2 and our future work involves expansion of this technique.

5. Conclusion and Future Work

In this paper we present the methods and measurement techniques that can be used to protect intellectual property in combinational level synthesis solutions that are predominant in hardware environments today. We show that a tradeoff exist between a publicly known obfuscation algorithm that maximizes random choices versus an algorithm that targets hiding properties of circuit variants specifically. Our future work will continue to develop deterministic methods for hiding properties and integration of all random and deterministic techniques into an overall algorithm space.

Acknowledgements

This material is based upon work supported in part by the U.S. Air Force Office of Scientific Research under grant number F1ATA09048G001. The views expressed in this article are those of the authors and do not reflect the official policy or position of the Unites States Air Force, Department of Defense, or the U.S. Government.

Bibliography

Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., and Yang, K. (2001) "On the (im)possibility of obfuscating programs(extended abstract)", *Advances in cryptology—CRYPTO 2001* (Santa Barbara, CA), volume 2139 of Lecture Notes in Computer Science, 1–18. Springer.

Kerckhoff, A. (1883) "La Cryptographie Militaire", **Journal des sciences militaires**, Vol 9, pp 5–83, Jan. 1883.

Kim, Y. and McDonald, J. (2009), "Considering Software Protection for Embedded Systems", **Crosstalk: The Journal of Defense Software Engineering**, Sept/Oct, Vol 22, No 6, pp 4–8.

McDonald, J., Kim, Y., Koranek, D., and Parham, J. (2011) "Evaluating Component Hiding Techniques in Circuit Topologies," submitted to, *ICC-CISS-2012*, June, 2012, Ottawa, Canada.

Nohl, K., Evans D., Pltz, S., and Pltz, H. (2008) "Reverse-Engineering a Cryptographic RFID Tag", *USENIX Security Symposium*, July 2008.

Reich, C.A. (1995) **The Greening of America**. Crown Trade Paperbacks.

Robert, M., Rouzeyre, B., Piguet, C., and Flottes, M., editors (2002) **SOC Design**

Methodologies: **IFIP Advances in Information and Communication Technology**, Vol. 90, Springer.
Simonaire, E. (2008) “Sub-circuit Selection and Replacement Algorithms Modeled as Term Rewriting Systems”, Master’s thesis, Air Force Institute of Technology (AU), December 2008.