Evaluating Component Hiding Techniques in Circuit Topologies

Jeffrey T. McDonald, *Member, IEEE*, Yong C. Kim, *Member, IEEE*, Daniel J. Koranek, James D. Parham,

***Abstract*—Security for Cyber physical systems includes not only guaranteeing operational security of data they process, but preventing malicious alteration of their execution due to knowledge of their underlying structure. With the advent of software in the form of reprogrammable hardware descriptions, protection of field programmable units from malicious reverse engineering and subversion becomes more critical. We compare four different white-box transformation algorithms aimed at hindering adversarial reverse engineering by changing component and signal configurations within combinational logic programs. We present security and efficiency analysis for these techniques and show positive results for achieving measurable hiding of signal and component information.**

***Index Terms*—Circuit Analysis, Combinational Circuits, Programmable Logic Devices, Computer Hacking, Computer Security.**

I. Introduction

COMPUTING technology embodied in both hardware circuitry and software represents a significant portion of both military and commercial development budgets. Adversaries target design understanding for this technology to find exploitable flaws, manipulate control for nefarious purposes, or clone system components for other uses. While cryptography provides adequate protection for data security, protection of design information inherent in programmatic specifications is not as firm. Nohl et al. [1], for example, showed the relative ease of reverse engineering the Milfare Classic RFID tag based on gate structure analysis in order to compromise cipher implementations. Popular hardware such as the Apple iPhone are sold at reduced versions overseas because cloning of underlying intellectual property can be accomplished [2]. In general, it can be much more economical for a foreign government to reverse engineer existing technology rather than procure a similar product from scratch [3]. Chikofsky and Cross [4] define reverse engineering as an analysis task of a subject system that creates equivalent representations at higher abstraction levels. Though industry and military analysts use reverse engineering to recover lost design information for legal, economical, and defensive purposes, we consider its nefarious use for malicious purposes.

We report results in this paper on techniques that involve combinatorial circuit designs. Namely, can we generate variants of such logic programs which convey less information about a circuit's original function and structure so that an adversaries ability to reverse engineer the circuit is significantly degraded? Algorithms that create variants are commonly known as *obfuscators* and theoretical study of obfuscation has received much attention over the last decade [5]–[7]. Given the theoretical impossibility of producing a circuit variant that hides all information leakage relative to an original circuit (other than its black-box behavior), we consider efficient obfuscation algorithms that produce semantically equivalent versions of circuits where the elimination of some functional information or some structural information can be empirically measured.

To discuss our finding, Section II introduces the scope of our obfuscation study and defines security based on functional and structural hiding related to component identification. Section III defines four different methods for achieving component hiding that can deter a malicious reverse engineer. Section IV presents empirical results using these methods on sample combinational logic programs. Section V summarizes our results and gives point for future work.

II. Variation and Circuit Protection

A. Defining Circuits, Signals, and Components

We consider combinational logic programs and the effect of structural changes at the syntactic/gate level where semantics of the overall circuit are preserved. Combinational logic represents a large class of software constructs where straight-line logic is involved. Sequential logic represents the more robust category of programmatic logic seen in traditional software where looping constructs are allowed; translation between high level languages and general circuitry provides

Manuscript received September 19, 2011. This work was supported in part by the by the U.S. Air Force Office of Scientific Research under grant number F1ATA07337J001.

Jeffrey T. McDonald is an Associate Professor with the School of Computer and Information Science at the University of South Alabama, Mobile, AL, 36688, USA (phone 251-460-7555; fax:251-460-7274; e-mail: jtmcdonald@usouthal.edu).

Yong C. Kim is with the Air Force Research Laboratory, Wright-Patterson, AFB, OH, 45433 (e-mail: engafit@yahoo.com).

James. D. Parham is with Air Force Research Laboratory, Kirtland AFB, NM (e-mail: James.Parham@kirtland.af.mil).

Daniel J. Koranek is with the Air Force Research Laboratory, Wright-Patterson AFB, OH 45433 (e-mail: daniel.koranek@wpafb.af.mil).

correlation of results using either software or hardware-based program descriptions [8]. We motivate our study of hiding properties using combinational logic for two reasons: 1) reverse engineering of circuits occurs frequently at the gate or ASIC level where combinatorial logic is prevalent; and 2) sequential circuits may be decomposed into combinational components for analysis and synthesis, separating state and memory decisions from allocation of basic building blocks.

For simplicity, we define a *circuit* as a directed, acyclic graph *D(V,E)* of inputs, logic gates, and outputs where 1) *V* describes a set of nodes (or vertices) representing logic gates and 2) *E* represents the set of edges representing wires connecting the output of some logic gate (or an input to the circuit) to the input of some other logic gate (or an output of the circuit). We define a *subcircuit* as a subgraph of some circuit *C*. The logic gates themselves map to known Boolean functions in a given basis set Ω such as Ω ∊ {*AND, OR, XOR, NAND, NOR, XNOR*}. We may represent circuit *semantics* as a Boolean function, f_C:$\{0,1\}^n \rightarrow \{0,1\}^m$, where *n* is the input size and *m* is the output size in bits.

Enumerating the full *truth-table* (a collection of all possible input/output pairs for a circuit) remains infeasible for most circuits because of the exponential (2^n) blow-up. Fig. 1 illustrates the truth-table for a 3-input, 2-output NOR-gate full-adder circuit. We define a *signal* as the truth table value for an intermediate gate within the graph of circuit. In Fig. 1, gates 4–10 are intermediate gates and their possible values form a truth-table (column) based on possible inputs and propagation of predecessor values. The signals of output gates (Sum, C_{out} as seen in Fig. 1), when associated with their respective inputs, create the semantics of the entire circuit. Fig. 2 illustrates the full-adder circuit and its corresponding signal values.

In order to discuss component properties, we define a *candidate component* based on White's definition [9] as a specified subcircuit which is fully specified, fully contained, or both. White's algorithm on subcircuit enumeration and identification forms the basis for our empirical study on component hiding. As a more suitable extension to this definition, we define a *component* as a subgraph with a specific input/output pattern whose semantics (truth table input/output mapping) are known.

	A	B	Cin	4	5	6	7	8	9	10	Sum	Cout
0	0	0	0	1	0	0	1	0	1	0	0	0
1	0	0	1	1	0	0	1	0	0	0	1	0
2	0	1	0	0	1	0	0	1	0	0	1	0
3	0	1	1	0	1	0	0	0	0	1	0	1
4	1	0	0	0	0	1	0	1	0	0	1	0
5	1	0	1	0	0	1	0	0	0	1	0	1
6	1	1	0	0	0	0	1	0	1	0	0	1
7	1	1	1	0	0	0	1	1	0	0	1	1

Fig. 1. Truth-table for 9-gate, NOR full-adder: *n* = 3, *m* = 2

B. *Analysis Activities*

In considering the goals of a reverse-engineering adversary, we make several educated assumptions about the strategies used to recover higher level design specifications of a circuit. We assume that a reverse-engineering adversary can perform *black-box analysis* on a circuit *C* by applying values to its *n* input gates and observing values of its *m* output gates, which is to say, they have oracle access to *C*. Because full truth-table analysis remains intractable for large input circuits (*n* > 50), we assume that adversaries are driven to combined analysis that looks at white-box topology and various gate signals.

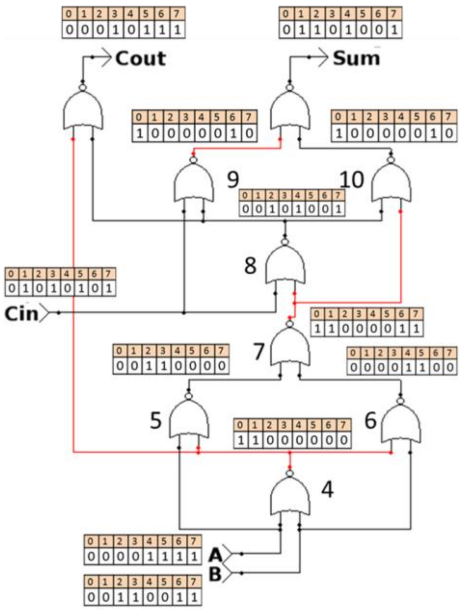

Fig. 2. NOR full-adder circuit topology and signal values

Because reverse engineers are particularly interested in the white-box information leaked by a circuit, we assume an adversary driven to *white-box analysis* attempts to identify components and the communication (signals and wiring) between them as their primary method to attain a higher-level circuit abstraction.

C. *Semantic Preserving Polymorphic White-Box Variation*

An obfuscator *O* is a polynomial-time algorithm that takes as input circuit *C* and returns a semantically equivalent version *C'*. Obfuscators produce a distribution of circuit variants that are functionally equivalent to a given *C*; we analyze variants in the distributions empirically on the basis of whether a variant exhibits a hiding property of interest. Variants are polymorphic by nature because they represent alternative forms of an original. We consider obfuscating algorithms that iteratively change the internal structure of a circuit in a semantic preserving manner. The obfuscating algorithm makes unique pseudorandom and deterministic choices in performing the structural change for a given number of iterations. Iteration count determines the overall run-time of the algorithm based on transformation techniques that are chosen by the obfuscator. We consider the effect of how much randomness or determinism is needed to produce hiding properties of interest. For this paper, two properties are in view: signal and component hiding.

D. Signal Hiding

Reverse engineering adversaries consider the value of signals within a circuit to establish key control flow and to establish component boundaries. To measure the effect of hiding in polymorphic variants, we ask the question "Do the original internal signals of the circuit *C* exist in a variant *C'*?" We acquire this metric primarily by analyzing truth-table values for every gate in both the original and variant circuit. In order to measure signals for large circuits, we must choose a reduced input size vector set must. Controllability and observability are well studied historical research areas [10] with the goal of finding the most suitable input vectors for testing and equivalence checking of large Boolean circuits. We can therefore measure signal hiding using reduced-size input vector sets that are produced by controllability analysis. For full signal hiding verification, we only consider circuits with reasonable input size.

E. Component Hiding

A combinational logic circuit *C* can be decomposed into a smaller set of combinational subcircuits $s = \{s_1, s_2, \ldots, s_n\}$ that are used individually to compute smaller functions. Typically, such components are well known and defined as well being implemented with the same gate structures. Typical component identification algorithms involve 1) finding some subcircuit s_{ident} with the same input/output flow as a known circuit s_{known}; 2) validating the semantics of s_{known} and s_{ident} are equivalent. Given a set of known components, the reverse engineer would identify the intent of *C* from the topology relationships between components within of *C*. In one sense, component identification is an extension to signal identification because components have distinctive outputs based on their respective functions. Since outputs of any given component come from the set of all signals within *C*, it follows that component hiding builds upon the ability to hide signals.

For measuring component hiding, we ask "Do the original component boundaries in *C* exist in a variant *C'*?" More properly, "Are there structures within the variant *C'* that map to known component input/output relationships in the original *C*?" To compute this metric we use a Java-based implementation [11] of the White algorithm described in [9].

F. Efficiency

We consider efficiency of various polymorphic obfuscation algorithms along the lines of speed and power/area of candidate circuit variants. Algorithm runtime of the obfuscator itself also bears on efficiency. Level count measures the largest number of hierarchical levels between inputs and output within a circuit and reflects increased computation delay/speed when level size is increased. Gate count is the number of logic gates within a circuit and reflects increased power consumption, space, and analysis space for the adversary. We compare metrics based on gate size (speed) and level count (power/area) when comparing different techniques for producing variants.

III. Transformation Techniques

For brevity, we present high level overviews for four different obfuscating algorithms. The basic transformation engine we use operates on a simple algorithm that takes circuits from a set of all possible combinational logic circuits, δ_Ω, based on a basis set Ω. We define our general obfuscating algorithm, $O(C) = C'$, as:

```
Given a circuit $C \in \delta_\Omega$, let $C_0 = C$
FOR $i = 0$ to $z$:
    1) SELECT a set of gates $G_i$ as a subcircuit within $C_i$ and
       let $f_{Gi}$ represent its function
    2) REPLACE subcircuit $G_i$ with a version $G_{i+1}$ such that
       $\forall x: f_{Gi}(x) = f_{Gi+1}(x)$
    3) REMOVE subcircuit $G_i$ from $C_i$ and replace with $G_{i+1}$
    4) let $C_{i+1} = C_i$
```

In this general algorithm, *z* represents an upper bound of iterations that provide a time-constraint or a goal-constraint for the obfuscator. For example, the algorithm may finish when it achieves the goal of replacing all original gates at least once. The particular methods in which we choose subcircuits (SELECT) and find a suitable replacement (REPLACE/ REMOVE) form the basis of algorithm comparison.

A. Pseudorandom Selection and Replacement (SSR)

Original work [12] with our polymorphic engine strived to maximize randomness in the obfuscator. We based pseudorandom algorithms on different options which governed the selection of up to *x* number of gates. For each iteration of the algorithm, we choose a different selection strategy randomly or use some single strategy to evaluate its efficacy. We let selection size of the chosen subcircuit (*x)* range from 1 to 4 gates and we choose gates in the subcircuit based on 1) level within the circuit (random, largest, fixed, or output level); 2) random choices for all gates; 3) random choices for all gates from within an updateable subset (i.e., gates not chosen yet); 4) gates with a specified metric such as maximum fan-out; or 5) gates part of identified component boundaries. *X-Y* SSR refers to *X* gate selection and *Y* gate replacement.

To maximize randomness in the replacement, a selected subcircuit G_i forms a functional family based on f_{Gi}. We let the size of the replacement subcircuit G_{i+1} vary between 0 to 2 gate sizes larger than G_i. Once we choose a gate size of the replacement subcircuit, we query a static library of all possible circuits with the same input/output/gate size and choose a uniform, random replacement from the subset of circuits that have the function f_{Gi}. We generate static libraries out of band and note that such libraries produce maximum randomness for the obfuscator at the expense of disk space and construction time beforehand. Because we use all possible circuit combinations, the circuit family size grows on an exponential order. On empirical observation, we find that using a 6-gate basis set Ω results in library sizes that match the integer series A000366 multiplied by 6^{g-1}, where *g* is gate size. Interesting hiding properties emerge for larger selection and replacement sizes, unfortunately limiting the utility of truly random-based selection and replacement. To expand the nature of polymorphic variation, we consider also deterministic methods of selection and replacement.

B. *Deterministic Boundary Blurring*

Boundary blurring [11] is based on targeting output gates of known components that are part of an original circuit C. The algorithm requires that a partitioned set of gates be defined for circuit C that represents original components. For each iteration of the obfuscator, a component g_i and an output gate of the component is chosen, $g_{orig.}$ 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 versions of the blur: 1) multi-level blurring generates new combinational logic using the originals signals of the two selected gates g_{orig} and $g_{rec.}$; 2) don't care blurs generate new combination logic depending on signals randomly chosen from other locations within the circuit. Once the combinational logic is built, g_{orig} and all gates between it and g_{rec} are replaced with the new logic. By choosing recovery gates outside of a component boundary, the blur allows for original component configurations to be modified while still maintaining overall circuit semantics.

C. *Deterministic Component Fusion*

Component fusion [13] ensures replacement of the entire circuit during every iteration of the algorithm. It accomplishes this by partitioning the circuit into subcircuits and then targets hiding of known information based on original component definitions. We describe the component fusion algorithm as follows:

```
Given a circuit $C \in \delta_\Omega$, let $C_0 = C$
Let $G$ = the gate set of $C$ and let $G_{UNUSED} = \varnothing$
Let $M = \{m_1, m_2, m_3, \ldots, m_p\}$, a component set of $C$
REPEAT
    1) SELECT a component $m_i \in M$
    2) PARTITION unused gates into connected subcircuits
       to produce component $m_i$'
    3) MERGE component $m_i$' into $C_i$ and add any changed
       gates to $G_{UNUSED}$
    4) let $C_{i+1} = C_i$
UNTIL $G_{UNUSED} = G$
```

Component fusion uses random choices during the SELECT and PARTITION operations, guaranteeing that the obfuscator produces unique variants. PARTITION produces a subcircuit that forms the basis for replacement used by the MERGE operation. MERGE uses a deterministic approach to create a replacement by: 1) choosing a random gate basis Ω; 2) choosing a random Product of Sum/Sum of Products canonical form; and 3) applying ESPRESSO's Quine McCluskey algorithm to logically reduce the component $m_{i'}$.

D. *Deterministic Component Encryption*

Component encryption targets specific components for protection directly and does not concern itself with other gates within the circuit. For each iteration, this method targets the wires and signals that join two components. It semantically changes the components themselves, but leaves the overall functionality of the circuit intact. The notion of component encryption comes from [14] where the overall semantics of circuits with small input sizes may be fully protected by concatenating an encryption or permutation cipher to the output of the original circuit. In this form, the obfuscator O takes P with small input size and produces a variant P' such that $\forall x$: $P'(x) = E_K(P(x))$.The modified variant is synthesized using a standard Quine McCluskey reduction. Fig. 3 illustrates the selection of two components and the additional of an encryption and decryption component between them.

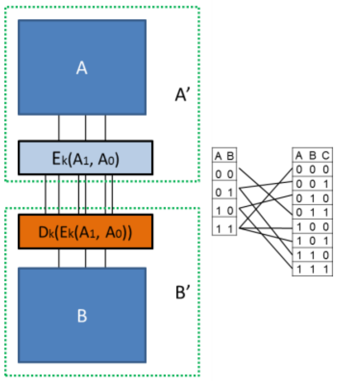

Fig. 3. Component Encryption of Two Selected Components

The component encryption algorithm works by reading a circuit description C and its associated component definition set M. Certain gates are not eligible to be encrypted using this approach and the algorithm groups those that are eligible into a candidate set of signals. The algorithm then generates encoding and decoding functions for these internal signals and produces new components. Once new combinational logic is inserted to reconnect new components, the algorithm selects the next set of candidate signals. Fig. 3 illustrates a one-to-many mapping for signals to their encoded values which is then used for decoding. Mappings are unique for every iteration of the component encryption process.

IV. Experimental Results

To analyze security and efficiency of these four algorithms, we create test cases based on three different candidate circuits: 1) *c6288* 16-bit multiplier, which contains a large number of identical components: 16 half-adders and 224 full-adders; 2) *c264* 4-bit multiplier with same components as c6288 on smaller scale: 4 half-adder and 8 full-adders; and 3) a *polymorphic full-adder* circuit with 5 4-input/1-output multiplexors composed of 15 2-input/1-output multiplexors. These circuits have distinguishing features which allow efficient component identification using the White algorithm [9]. We generated 50 variants of each test case circuit using all four algorithms and then applied security and efficiency metrics to compute an average. We also verified the functional equivalence of all generated circuit variants created by the four different algorithms. Fig. 4 illustrates our results for signal hiding based on truth table analysis metrics and percentage of original circuit signals hidden. Fig. 5 represents the perceived component hiding based on negative component identification metrics (100% means no original components were identified). Fig 6. illustrates our results for efficiency analysis based on the gate size and number of levels in variants and the algorithm runtime in seconds.

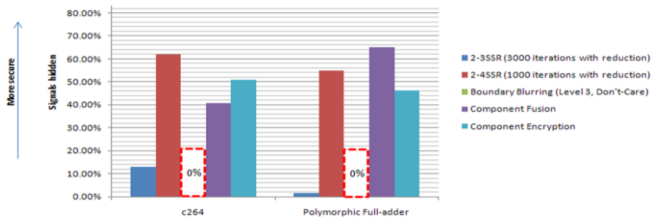

Fig. 4. Variant signal hiding per algorithm.

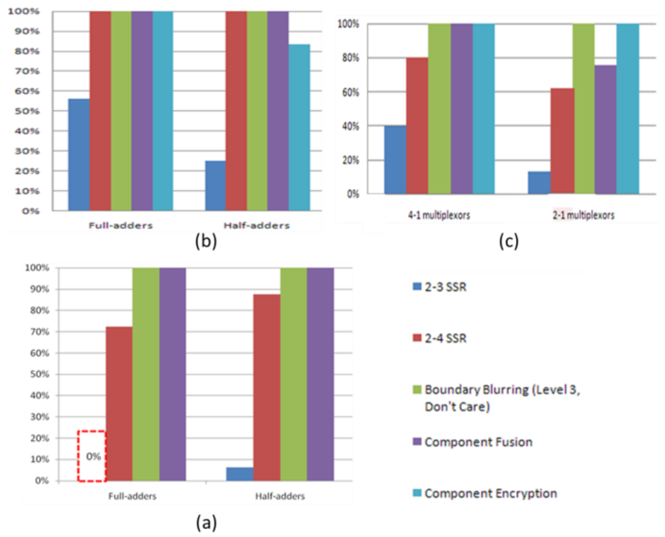

Fig. 5. Component hiding per algorithm for (a) c6288 variants, (b) c264 variants, and (c) polymorphic full-adder

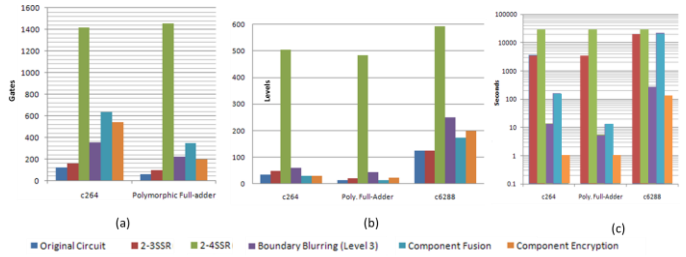

Fig. 6. Efficiency per algorithm based on (a) average gate size per variant, (b) average level size per variant, and (c) obfuscator runtime per variant

V. Conclusions And Future Work

Based on experimental results, we observe that component encryption has the fastest runtime of the four algorithms and produces better signal hiding results than boundary blurring. Component fusion also produces better hiding than blur or random selection/replacement based on component identification metrics. In some cases, boundary blurring accomplishes no signal hiding and random selection/replacement with 2-gate selection and 3 or 4 gate replacement provides poor signal hiding and worst-case increases in level size. No algorithm accomplished full signal hiding and all algorithms produced *some* component hiding based on results of running a component identification tool. We demonstrate that efficiency can be maintained using some algorithms (component fusion and encryption) within reasonable bounds while frustrating efforts of reverse engineering adversaries.

Future work will seek to understand the effect of varying all four techniques on the same circuit on both efficiency and security metrics with expanding metrics to other algorithms and implementations. Other circuits with additional component configurations will be in view for future experiments. Research on component identification and recovery has moved toward large-scale Boolean matching [15] which seeks to determine whether two Boolean functions are semantically equivalent when inputs and outputs are reordered or negated. Future work will involve utilizing Boolean matching tools as an adversarial component identification analysis technique for comparative purposes.

References

[1] K. Nohl, D. Evans, S. Pltz, and H. Pltz, “Reverse- Engineering a Cryptographic RFID Tag,” *USENIX Security Symposium*, July 2008.

[2] D. Koeppel, “China’s iClone ,” Popular Science, August, 2007. URL: http://www.popsci.com/iclone

[3] Select Committee on U.S. National Security, “HOUSE REPORT 105-851 Chapter 1: PRC Acquisition of U.S. Technology”, January 1999, URL: http://www.gpo.gov/congress/house/hr105851-html/ch1bod.html.

[4] E. Chikofsky and J. Cross II, “Reverse engineering and design recovery: a taxonomy,” *IEEE Software*, 7(1):13–17, Jan 1990.

[5] B. Barak, O. Goldreich, *et al.*, “On the (im)possibility of obfuscating programs,” *Proc. of CRYPTO ’01*, Aug 2001, 1–18.

[6] N. Ding and D. Gu, “A general and efficient obfuscation for programs with tamper-proof hardware,” *Proceedings of ISPEC'11*, Feng Bao and Jian Weng (Eds.). Springer-Verlag, Berlin, Heidelberg, 401-416.

[7] S. Goldwasser and G. Rothblum, “On best-possible obfuscation,” LNCS 4392, *TCC 2007*, Springer-Verlag, (21-24 Feb 2007), 194–213.

[8] N. Wirth, “Hardware Compilation: Translating Programs into Circuits,” *IEEE Computer*, 31(6):25–31, June, 1998.

[9] J. White, A.Wojcik, *et al.*, “Candidate sub-circuits for functional module identification in logic circuits,” *Proc. of the 10th Great Lakes Symposium on VLSI (2000)*, Chicago, IL, 34–38.

[10] L. H. Goldstein, “Controllability / observability analysis of digital circuits,” *IEEE Trans on Circuits and Systems*, CAS-26(9):685–693, September 1979.

[11] J. Parham, Y. Kim, *et al*., “Hiding Circuit Components Using Boundary Blurring Techniques,” *Proc. of IEEE Annual Symposium on VLSI*, (Jul. 5-7, 2010), Cephalonia, Greece.

[12] J. McDonald, Y. Kim, and M. Grimaila, “Protecting Reprogrammable Hardware with Polymorphic Circuit Variation,” *Proc. of the 2nd Cyber Research Wrkshp* (June 2009), Shreveport, LA.

[13] J. McDonald, Y. Kim, and D. Koranek, “Deterministic Circuit Variation for Anti-Tamper Applications,” to appear, *Proc. of 7th Annual CSIIRW* (12—14 Oct, 2011), Oak Ridge, TN.

[14] J. McDonald and A. Yasinsac, “Program intent protection using circuit encryption,” *Proc of the 8th Intl Symposium on System and Information Security*, IEEE Computer Society, 8-10 Nov 2006.

[15] H. Katebi and I. Markov, “Large-scale Boolean matching,” *Design, Automation & Test in Europe* (8-12 Mar 2010), 771—776.