DYNAMIC POLYMORPHIC RECONFIGURATION FOR ANTI-TAMPER CIRCUITS

Roy Porter, Samuel J. Stone*, Yong C. Kim*, J. Todd McDonald*, and LaVern A. Starman*,*

Dept. of Electrical and Computer Engineering
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433 USA
email: {roy.porter.2, samuel.stone}@us.af.mil, {ykim, jmcdonal, and lstarman}@afit.edu

ABSTRACT

The susceptibility of digital systems to tampering is of immense concern to military and commercial organizations. Current defenses against such techniques as reverse engineering and side channel analysis are limited and don't address the underlying vulnerable characteristics of digital circuits. In this paper, we propose a generalized defense methodology named Dynamic Polymorphic Reconfiguration that significantly reduces the probability of successful tampering. We achieve protection through targeted component hiding and the introduction of run-time autonomous defense adaptations. As a result, we establish the feasibility of self-protecting circuits.

1. INTRODUCTION

The protection of sensitive information and intellectual property that is processed on digital systems is generally regarded as the responsibility of the user. This allows designers to focus on optimizing performance parameters without being limited to an innumerable amount of security concerns. The problem with this approach is that there currently exists a gap between the knowledge of exploitation techniques and the sophistication of applications that these systems are used in. Thus, there is a critical need to provide users with dependable protection methods that meet the level of current threats.

Dynamic Polymorphic Reconfiguration (DPR) offers defense against a large class of vulnerabilities. Dynamic reconfiguration is a relatively new concept to the area of circuit security, due in large part to the lack of threat generalization. That generalization is finally achieved in [1] where it is shown that vulnerabilities to such techniques as reverse engineering and side-channel analysis stem from the ability to recognize component functionality or structure. Thus, the case is made for a defense framework that targets both characteristics. DPR achieves this through a mix of obfuscating transformations and dynamic structure reconfiguration.

The views expressed in this article are those of the authors and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

2. TAMPERING

2.1. Reverse Engineering

Reverse engineering (RE) encompasses a range of techniques to identify circuit intent and structure using a small set of known characteristics. Black-box and white-box methods are the two classifications that make up RE. The associated techniques correspond to the amount of information that is available for analysis. We may define successful black-box analysis as follows:

> Given a circuit $\mathbf{C}\rightarrow\{\mathbf{X}, \mathbf{Y}\}$, an arbitrarily large set of pairs $\mathbf{IO} = \{x_i, y_i \mid y_i = \mathbf{C}(x_i)\}$ where $x \in X$, $y \in Y$, and an arbitrary element $y_j \notin \mathbf{IO}$, we may consider black-box understandability as the ability to efficiently find x_j such that $y_j = \mathbf{C}(x_j)$ [2].

This approach involves exercising input combinations and mapping the associated outputs. Examples of black-box analysis include trying all input combinations to determine full functionality, inputting random vectors (also known as *fuzzing*), or inputting invalid combinations that make the circuit malfunction. A common misconception regarding black-box RE on modern circuits is that it is a futile effort, owing to the increasing I/O space and circuit complexity. However, in [1] it is shown that this technique is an efficient method even with an I/O space numbering in the thousands. The key idea established in that research is the ability to recognize patterns in a partial truth table that are characteristic of common functions. Thus, we have the first requirement for successful defense, introducing confusion into a circuit's truth table.

White-box analysis is not hindered by such information limitations. Under this method, given the previously defined circuit $\mathbf{C}$ and element y_j, it is possible to efficiently compute x_j by analyzing the circuit structure. The structure can be available in the form of a design netlist, circuit imagery, etc. Avery *et al.* have conducted work in which optical and electron beam microscopy, pattern recognition, and netlist extraction techniques are combined to reconstruct logic diagrams and VHDL designs from

ASICs [3]. In [4], Nohl *et al.* apply white-box analysis to the reconstruction of an embedded proprietary cryptographic circuit. Key to the success of their approach is the fact that cryptographic algorithms rely heavily on bit-wise logical operations. Nohl *et al.* exploit this fact by creating a library of the 70 different gate types found on-chip and focusing their efforts on areas dense in exclusive-or (XOR) functions. Thus, the second requirement for successful defense is established, namely obfuscating the recognizable structure of common components.

2.2. Side-Channel Analysis

Operation of electronic devices can be characterized by measurements of power, time and electromagnetic emissions. The exploitation of these attributes is known as a side-channel attack. Information gathered from side-channel measurements can be used to deduce areas where specific data is stored or when certain information is made available for theft. Miyamoto *et al.* present a unique method of side-channel analysis against RSA cryptosystems that does not require substantial knowledge of the circuit itself [5]. The attack relies on the identification of functional components that are necessary to carry out the multiplication and exponentiation operations of the RSA algorithm. Given a specific implementation, along with a chosen input value, secret key information that leads to the factorization of the RSA prime can be obtained. The process is simple in that it solely requires measuring the voltage drop across a low-value resistor that is inserted between the circuit ground and external ground.

Timing related attacks exploit the fact that operations can be differentiated by the delay between presented input and output availability, i.e. addition takes less time than multiplication. This characteristic was first exploited by Kocher in 1996 [6] and has since been applied to various implementations of cryptographic systems. Schindler *et al.* have shown that it is possible to extract the 512-bit RSA secret key using 5,000 measurements of total running time [7].

Taking into account the different methods of side-channel analysis, it is clear that the common characteristic is the ability to compare actual measurements to assumed values. If those measurements can be altered to bear no resemblance to the expected, it becomes more difficult to carry out this attack. Additionally, if the measurements continuously change over time, the chances of creating a correlation are further reduced. This establishes the third goal of DPR which is to carry out continuous reconfiguration at a rate faster than analysis can be carried out.

3. POLYMORPHISM AND OBFUSCATION

In [8] and [2], the case is made for using systematic techniques of structural substitution and permutation to obfuscate digital circuits, targeting intent protection in both black and white-box domains to produce "executably encrypted" circuit variants. Polymorphic transformations are at the heart of the encryption processes, which aim at making an obfuscated variant indistinguishable from the original in both I/O relationships and white-box structure. These transformations take place at various levels of a design's hierarchy – function, component, gate, signal [9]. The line that separates different levels is contextually dependent and not considered to be definitive. In all cases equivalence is defined in terms of I/O fanning or semantic representation. Obfuscation at a higher level necessarily builds upon changes at lower levels.

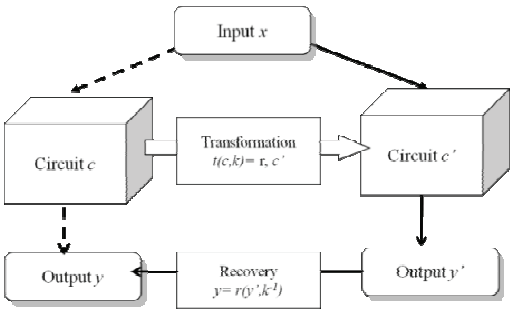

Figure 1: General Black-Box Transformation. A transformation *t* takes circuit *c* and black-box transform key *k*, producing circuit variant *c'* and recovery function *r*. Give output *y'* of the circuit *c'* based on input *x*, *r* provides recovery to original output *y*.

In [2], two goals for black-box intent protection are established. Firstly, the obfuscated circuit, *C',* must exhibit a different functional output pattern than the original circuit, *C*. As a consequence of achieving the first goal, *C'* by itself is not useful in the design. Accordingly, the second goal is to provide a capability to recover the original output of *C* from *C'*. As Figure 1 illustrates, black-box polymorphism results through transforms which produce circuits with different, but recoverable, input/output patterns. We use this approach on a small, component basis as the foundation for Dynamic Polymorphic Reconfiguration (DPR), which we elaborate next.

4. POLYMORPHIC TRANSFORMATIONS

For our consideration, we consider a circuit to be a small component or sub-circuit of a larger circuit. DPR transformations use circuit graph cut-sets that have a specific form. We may define a sub-circuit or cut-set

component G as a graph $G(V,E)$ of Boolean logic gates where the node set V represents gates and the edge set E represents wire connections between gates. DPR cut-sets specifically take the following form:

- A set of nodes $V = \{A, B, C\}$
- A set of edges $E = \{a, b, c, d, e, f, g\}$
- The edges a, b, c, and d are inputs to the system while g is the output
- a, b, c, and d must be distinct

Each element N of the node set V has a corresponding Boolean function, f_N: $\{0,1\}X\{0,1\} \rightarrow \{0,1\}$. For our specific cutest configuration, we define the node functions as related to the edges a,b,c,d,e,f,g as follows: $f_A(a, b) \rightarrow e$, $f_B(c, d) \rightarrow f$, $f_C(e, f) \rightarrow g$. Since each function has two Boolean inputs, we describe a bit signature as the elaboration of all possible inputs applied to the appropriate logic function. Given a gate function $g(a,b)$ with input edges a and b, we describe its corresponding bit signature as a 4-bit vector $[g(0,0), g(0,1), g(1,0), g(1,1)]$ or $[g_0,g_1,g_2,g_3]$. For the DPR cut-set, we describe the bit signature for non-input edges as follows: e = $[e_0,e_1,e_2,e_3]$, f = $[f_0,f_1,f_2,f_3]$, and g = $[g_0,g_1,g_2,g_3]$.

DPR creates polymorphic reconfigurations such that (1) there is a guarantee that additional signals are integral to circuit operation and (2) the original I/O mapping is changed. Specifically, we delineate four basic operations that may be performed on a circuit cut-set: gate replacement, signal hiding, input signal/gate addition, and output signal addition. We explain each of these techniques next and note that each one may result in changing the original function while providing specific means for recovering the output.

4.1. Gate Replacement

Cut-sets are localized groupings of gates targeted for reconfiguration by the DPR algorithm and are made up of two levels, noted as the input level and output level (Figure 2). We define a gate replacement as a substitution of one functional node type for another (different) one. DPR invokes replacement on gates at the input level, giving us thirty possible replacement combinations corresponding to six possible gate types (AND, OR, XOR, NAND, NOR, XNOR) and the fact that a gate cannot be replaced with itself. The choice for replacement is left up to random selection from the thirty possible.

4.2. Signal Hiding

We define targeted signal hiding as a white-box transformation that affects the intermediate semantic representation of a circuit. Signal hiding refers to the information that the signal transfers, not the instantiation of the signal itself [2]. For DPR cut-sets, signal e, f, and g are of interest for signal hiding purposes where we wish to change or eliminate the original bit signatures. Using the cut-set notation in Figure 2, the four possible output bits of signal e are e_0, e_1, e_2, e_3. After signal hiding, signature e is transformed to e' and the possible states are e'_0, e'_1, e'_2, e'_3.

Signal hiding occurs iff $e_0e_1e_2e_3 \not\equiv e'_0e'_1e'_2e'_3$. Gate replacement induces signal hiding when the gate that we choose to replace corresponds to the signal that we want to hide. For example, gate replacement of on Gate A will induce signal hiding on the signature e, while gate replacement on Gate B will induce signal hiding on the signature of f. Following replacement, the cut-set will exhibit a new structural (white-box) make-up and potentially, a new functional (black-box) operation.

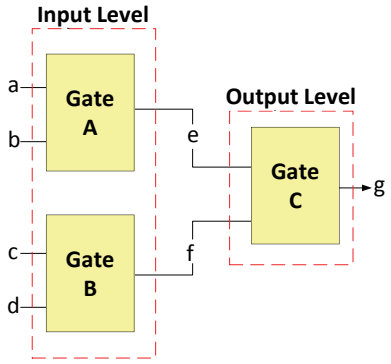

Figure 2: Cut-set. Cut-sets are made up of two levels of logic (input and output), three gates and up to seven signals.

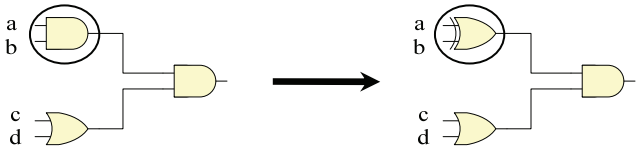

Figure 3: Gate replacement. The AND gate in the first cut-set is replaced with an XOR.

4.3. Input Signal Addition or Gate Addition

We can describe the addition of an input signal or a single gate as the same operation, with the new signal representing the output signature of the new gate. When DPR adds a signal, the gate and signal is integrated into the circuit in such a way so that we affect the semantics of the original component. We can illustrate this concept through truth table representations of circuits, where a new input adds an additional column to the input enumeration and induces a change to the output column or other intermediate gate columns of the component.

Consider the addition of a signal to a cut-set. As the first step, we choose how that signal (h) will be integrated. The method chosen by DPR is to add a gate (D) between the input and output levels for which the new signal will be one input and the other input will come from the first level. DPR chooses the type of intermediary gate randomly. The second step is to ensure that the output of the cut-set is changed. This is accomplished by connecting the output of

the new gate (*i*) to the input of the gate that serves as the cut-set output (*C*): see Figure 4.

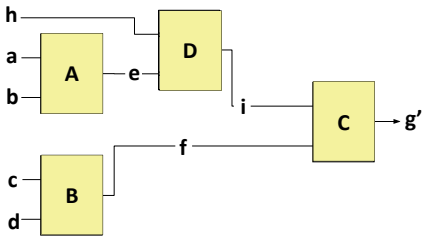

Figure 4: Result of signal/gate addition. Adding signal *h* requires adding a new gate, *D*, between the input and output levels, introducing corresponding signal *i*.

4.4. Output Signal Addition

Adding an additional output is a necessary capability so that new inputs can originate internally instead of only at the circuit boundary. Consequently, steps for adding an output signal are straight-forward: take any collection of cut-set signals and combine them using a randomly chosen gate. The gate output becomes a component output.

5. RECOVERY KEY

The second part of dynamic polymorphic reconfiguration concerns the remainder of the system within which we re-integrate the reconfigured component. It also addresses the issue of maintaining overall semantic equivalence of the circuit in which the component resides. We use the term *convergence* to describe how all related changes induced by a reconfigured component have to be reversed at a single point. For this, we employ the idea of a *recovery key* [2]. Without the key, the remainder of the circuit will see only the reconfigured output (which has a different signature than that of the original component), with which it cannot operate correctly. The intent of the key then, is to recover the original semantic representation of the component and therefore allow the overall circuit to retain functional equivalence.

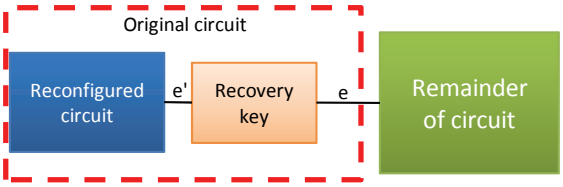

Figure 5: Standalone key. Although the original circuit has been reconfigured to hide signal *e*, the added security is insufficient. The reconfigured circuit and recovery key can be combined to form the original circuit.

We observe one notable characteristic of DPR related to these recovery keys: only the last original cut-set in a ser series of component reconfigurations is needed to create a key. This property reduces the size of the final reconfigured circuit because we can make several changes without creating keys for them individually. Here, we formulate two methods for key development: standalone and encoded. Standalone key offers a less complex approach that is easier to analyze. The encoded key scheme is an extension of the standalone approach and offers a preferred method for extending black-box component changes to other parts of the circuit with degrees of variability.

5.1. Standalone Key

A standalone key component is a component that takes as input the transformed output signal of a reconfigured component and then outputs the recovered / original output signal of the component. To accomplish this, DPR uses the result from a comparison between the full output signature of the original and reconfigured cut-sets. In basic terms, the key outputs the necessary function to XOR with the new output. The XOR function is chosen for its unique property of having no single controlling value and so it can provide any desired output regardless of the fixed value on one input.

Convergence becomes an issue with the use of standalone keys because the original signal is locatable if a component that uses the reconfigured output can be identified (Figure 5). The key can then be considered part of the reconfigured component. A considerable amount of white-box obfuscation may have taken place, but the black-box obfuscation is negligible. The solution to this problem is encoding the key.

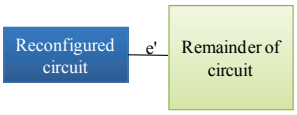

Figure 6: Reconfigured 5(a) using encoded key. Encoding key eliminates possibility of locating *e*.

5.2. Encoded Key

Using encoding, a component's reconfiguration takes place with knowledge of changes to other cut-sets that produce its inputs. Consider a circuit that is made up of two connected components, *A* and *B*, which can be distinguished from one another. Once *A* has been reconfigured so that its output is changed, its key is encoded into *B*. The difference between the reconfiguration processes on *A* and *B* is that the latter is necessarily more deterministic. Determinism in this case does not mean that reconfiguration is predetermined; there remains a random quality to all the transforming operations. What *is* predetermined is that at some point in *B*, the changes will converge. The natural break between components no longer exists which is the limitation of standalone keys (Figure 6). The original function of *A* cannot be recovered unless it is separated from *B* and all of

B's previous transformations, which may involve other components that *A* is not associated with.

6. AUTONOMOUS / DYNAMIC OPERATION

With the proper entry and terminating conditions, each of the transformations becomes programmatically employable. Therefore, we may use algorithms to implement each transformation. The creation of an encoded key is dependent on the components that are connected to the reconfigured output. Specifically, a cut-set in a connected component must be identified as the location for convergence. This is the point at which the original functionality of the circuit will be preserved, i.e. where a standalone key will be created. The fewer points of convergence created, the better the obfuscation.

Polymorphic reconfiguration is easily extended to a dynamic operation when utilizing a FPGA that enables the targeting of specific look-up tables (LUTs). DPR adds protection in the sense that a design is stable only for the length of time necessary to carry out an operation. Stability in this case can be expressed both in terms of location and structure. Because the transformations used by DPR are at the gate level, it is natural to look at this technique in terms of changing LUT definitions.

7. EVALUATION PLATFORM

A custom reconfigurable platform is required to handle the communication requirements of dynamic reconfiguration. An array of VHDL coded, 1-bit LUTs is created on a Xilinx Virtex-II Pro FPGA. Each LUT takes as input two 1-bit operands and a 4-bit configuration identification code. The combination of the inputs dictates the 1-bit output. In lieu of building complex networks to handle information transfer, an embedded PowerPC core and software defined registers are utilized. The registers are VHDL instantiated constructs, four assigned to each LUT. It is in these registers that a LUT's input, output, and configuration are communicated. Control is provided by a C program running on the PowerPC core.

8. REVERSE ENGINEERING MITIGATION

DPR's effect on reverse engineering is first described in terms of proofs. This method is used because it eliminates the need for exhaustive case study analysis. The proofs are important because they show that the deterministic use of DPR in fact achieves effective obfuscation.

8.1. Gate Replacement

A gate replacement operation changes the gate type of *A*. The reconfiguration results in two black-box and one white-box changes. The first is *e* is given a new signature: $f_{A'}(a, b) \rightarrow e'$. Because *A* has changed and necessarily exhibits a different function, *e* will differ from e' in at least one bit location.

Proof Sketch A: Given $f_A(a, b) \rightarrow e$ and $f_{A'}(a, b) \rightarrow e'$. Then *e'* is identical to *e* iff $[e_0, e_1, e_2, e_3] \equiv [e'_0, e'_1, e'_2, e'_3]$. But by definition, $A' \not\equiv A$ so $[e_0, e_1, e_2, e_3] \not\equiv [e'_0, e'_1, e'_2, e'_3]$ and $e' \not\equiv e$.

The second change is a potential black-box change resulting from the fact that a replaced gate may affect the output of the cut-set. When the input-level of a cut-set contains a gate (*B*) that produces a controlling value regardless of the input, changes in the other gate (*A*) will be masked. However, if there exists an input combination for which *B* does not produce a controlling value, then the change in *A* will be reflected in the cut-set output signature.

Proof Sketch B: Given $f_A(a, b) \rightarrow e$, $f_B(c, d) \rightarrow f$, and $f_C(e, f) \rightarrow g$. Let the controlling input of *C* be e_x and the non-controlling value f_y for some $0 \leq x, y \geq 3$. After a gate replacement operation on *A*, $g' \not\equiv g$ *iff* e_x is not changed to or from f_y. However, we know from Proof A that $e' \not\equiv e$ after a gate replacement and thus $g' \not\equiv g$.

The final change that results from gate replacement is a white-box change to the structure of the cut-set, which follows by definition.

8.2. Signal Hiding

Signal hiding is achieved through gate replacement, with the cut-set chosen in such a way that the gate from which the targeted signal originates is the gate which is replaced. Proof A confirms the correctness of this operation in meeting the established definition of hiding a signal.

8.3. Input Signal Addition or Gate Addition

Signal/gate addition results in an increase in the number of input signals and a change in the netlist connections. From an I/O perspective, the effort to identify a complete cut-set mapping is increased from 2^4 to 2^5. Secondly, in the case that *B* does not always produce the controlling value for *C*, it is not possible to recover *g* from *g'* using black-box techniques. This is proven as follows:

Proof Sketch C: Given $f_A(a, b) \rightarrow e$, $f_B(c, d) \rightarrow f$, $f_D(e, h) \rightarrow i$ and $f_C(i, f) \rightarrow g$. Let g_z be the output for which f_y is not the controlling value. g_z is recoverable from g'_z *iff* $e \equiv i$ for both values of *h* (0,1). This requires *e* to always be the controlling value for *D*. However, this is impossible with the implementable set of gates.

9. SIDE-CHANNEL ANALYSIS MITIGATION

9.1. Timing Analysis

During the examination of DPR's operational impact, it is shown that the timing is affected in terms of the gate delay

from input to output. This is evident in the cut-sets that are reconfigured. Because timing analysis makes use of statistical comparisons between expected and measured functional delays, varying the time it takes to complete those particular operations is a strong defense. The addition of a gate with the AddGate routine results in an increase of two gate delays to a cut-set. The Xilinx ISE synthesis software reports a delay of 0.086ns per LUT used in implementation. There is also an additional delay associated with adding the routing information, which varies with the particular instance of reconfiguration. Since each LUT corresponds to a single gate, the delay is at least 0.172ns per call of the AddGate routine.

9.2. Power Analysis

Power analysis uses statistical comparisons to infer information about a particular implementation, measured in terms of the power required to carry out operations. DPR affects power in two ways: the dynamic power used during computation and the static power that is consistently drawn. Here the dynamic power is of interest. For each input that is added with the *AddInput* routine, there is a 1.4% increase in dynamic power. For each gate that is added to the computation path, the change is 0.006%.

On the evaluation platform, the dynamic power difference between an 8-bit adder/subtractor and 8-bit multiplier is 0.03279. Multiplication requires less power because it utilizes a built-in component. The power signature of the multiplier can be made to resemble that of the adder/subtractor with the addition of 33 inputs. This is possible even though the multiplier is built-in because white-box obfuscation is not needed and the signals can be added indiscriminately.

10. OPERATIONAL IMPACT

Here an analysis is made of DPRs impact in terms of size and computation time. Size is measured by gate count and time is based on the increased latency from input to output. Both gate replacement and signal hiding do not result in a gate count increase. Gate addition and input addition lead to an increase of one gate. The range of gate increase as a result of key recovery is 2 to 2^{n-1}, where n is the number of inputs to the target cut-set. On the minimal end, this corresponds to transformations that result in all 1's or 0's in the obfuscated signature. At the other extreme, are transformations that change 2^{n-1}-1 bit locations of the original signature. Gate replacement and signal hiding both add at most one gate delay from input to output whereas the addition of either a gate or input results in two extra gate delays.

A timing delay is also incurred with each reconfiguration operation. This is minimized by the fact that a separate clock can be used to control changes to LUTs and the steps prior to making a physical change can be done in parallel with the normal circuit operation. Thus, DPR can be optimized to add one clock delay per reconfiguration.

11. CONCLUSION

The core of this research is in defending critical technology from tampering. Commercial devices offer little protection against the exploitation of vulnerabilities such as reverse engineering and side-channel analysis. The idea of using DPR to improve that defense offers significant possibilities. By continuously employing obfuscating transformations, DPR injects confusion into a circuit's truth table and thereby reduces the likelihood of successful reverse engineering. Secondly, the ability to provide this protection autonomously and in a dynamic fashion allows for a circuit to protect itself from tampering without the need for user interaction. Finally, the use of transforming algorithms that mix random and deterministic methods ensures that the protective measures cannot be easily undone, increasing any analysis effort above that which would be required for an exhaustive functional mapping.

12. REFERENCES

[1] Roy Porter, "FPGA Tamper Protection Through Dynamic Polymorphic Reconfiguration," M.S. Thesis, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, MS Thesis AFIT/GE/ENG/09-08 March 2009.

[2] J. Todd McDonald, Yong C Kim, and Alec Yasinsac, "Software Issues in Digital Forensics," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 3, pp. 29-40, 2008.

[3] J.S. Crabbe, S. Al Sofi, H. Ahmed, J.R.A. Cleaver, D.J. Weaver L.R. Avery, "Reverse Engineering Complex Application-Specific Integrated Circuits (ASICs)," in *Diminishing Manufacturing Sources and Material Shortages*, 2002.

[4] Karsten Nohl, David Evans, and Starburg and Henryk Plotz, "Reverse-Engineering a Cryptographic RFID Tag," in *USENIX Security Symposium*, San Jose, 2008.

[5] Atsushi Miyamoto, Naofumi Homma, Takafumi Aoki, and Akashi Satoh, "Enhanced Power Analysis Attack Using Chosen Message against RSA Hardware Implementations," in *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on*, pp. 3282-3285.

[6] Paul C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and other Systems," in *Advances in Cryptology*, 1996, pp. 104-113.

[7] Werner Schindler, Francois Koeune, and Jean-Jacues Quisquater, "Unleashing the full power of timing attack," UCL Crypto Group, Louvain-la-Neuve, Technial Report Series B-1348, 2001.

[8] Samuel J. Stone, Roy Porter, Yong C. Kim, and Jason V. Paul, "A Dynamically Reconfigurable Field Programmable Gate Array Hardware Foundation for Security Applications," *IEEE International Conference on Field Programmable Technology*, pp.305-308, Dec., 2008, Taipei, Taiwan.

[9] J.T. McDonald and A Yasinac, "Program intent protection using circuit encryption," in *Proc of the 8th Intl Symp. on System and Information Security. IEEE Computer Society*, 2006.