Protecting Reprogrammable Hardware with Polymorphic Circuit Variation[*]

J. Todd McDonald
Air Force Institute of Technology
jmcdonal@afit.edu

Yong C. Kim
Air Force Institute of Technology
ykim@afit.edu

Michael R. Grimaila
Air Force Institute of Technology
mgrimail@afit.edu

Abstract

Cyperspace is constantly threatened by attackers and malware that focus their attacks on a set of known vulnerabilities. When a sequence of software code or hardware structure is exposed, it can reveal new vulnerabilities and weaken embedded protections. Attacks on existing code sequences or hardware structure will be less effective if we can provide sufficient protection. Though software protection is an open problem with known theoretical limits, practitioners seek to find ways of expressing time or cost metrics induced by various techniques on malicious reverse engineers and adversarial analysis. In this paper we consider the nature of circuit transformation algorithms that operate on programmatic logic using iterative sequences of probabilistic and deterministic transforms. We consider such algorithms from the perspective of the kinds of information relative to circuits we are interested in hiding or protecting and experimental results along those lines.

1 Introduction

One approach to protecting software or circuits from reverse engineering is *obfuscation*: obscuring programmatic logic or original source code information so that an adversary may not subvert, copy, or understand some original version [4]. We observe that general programs typically have collections of straight-line logic (no loops and discrete input/output relationships) and basic programs are themselves abstractions of Boolean primitives [8]. Accordingly, we may represent an interesting class of programmatic syntax as Boolean logic circuits. We also note that reprogrammable hardware environments such as Field Programmable Gate Arrays offer possibility for software-like configuration in a wide variety of modern embedded systems. This provides great context for the Cyber realm and gives us motivation to understand the limits of circuit variation because more and more cryptographic operations and critical technology now find their way into reprogrammable environments.

Leveraging this correlation, we present in this paper an experimental environment that gives insight into the fundamental nature of whitebox variation where functional semantics of a circuit are preserved. Namely, at what point does a polymorphic circuit[1] variant exhibit a hiding property of interest, or obfuscation? We consider this question by analyzing the effect of systematic and iterative changes (variation) to small parts of a circuit where we allow large variability within the design of specific experiments. Such experiments allow us to introduce large numbers of user-driven goals, ran-

[*]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

[1]Other established definitions of polymorphism in virology refer to multifunctional circuits that perform two or more functions under different conditions. We use the term polymorphism to highlight the fact that functionally equivalent circuits have *many* (poly) different *forms* or *kinds* (morph) that are all semantically interchangeable.

dom/probabilistic choices, and criteria-based deterministic options. Thus, we can consider end-to-end effects of small syntactic level changes that manifest not only as whitebox structural variations, but possibly protection metrics of interest.

2 Background

As a measure of security, circuit obfuscation has theoretical boundaries if we desire to prevent *all* leakage in the information theoretic sense [1] or if we want to obtain a best possible alternative [6]. However, if we allow transformations that change blackbox behavior but use a recovery function to return the intended output, other possibilities exist. If we have small input-size functions, we can combine canonical minimization and encryption function composition to fully hide the intent of intermediate gate logic [12]; likewise, if we have circuits with behavior that falls into special classes such as rational functions, we may use homomorphic transformation schemes to provide the hiding [14]. If we limit our measurement scope to specific properties such as side-channel analysis [9, 20, 15] or topology hiding [19], several heuristic and theoretical models come into view as well.

Cohen [3] was one of the first researchers to link Shannon's concepts of confusion and diffusion with programmatic transformations. Most modern obfuscation algorithms use one or more of the program evolution techniques suggested by him: equivalent instruction sequences, instruction reordering, variable substitution, jump addition/removal, call addition/removal, garbage insertion, program encoding, redundancy, program interleaving, and anti-debugger mutations.

More recently, researchers have appealed to formal software models to express certain properties related to obfuscation. Term rewriting systems [2, 16], abstract interpretation [5, 13], and program encryption [17, 11] have all been used to analyze and characterize the effect of structural variation and syntactic changes. These frameworks may either characterize the difficulty of finding and normalizing malicious transformations or attempt to measure the strength of friendly protection schemes based on variation.

Table 1: RPM Notations

Variable	Meaning
C	A combinational Boolean circuit
C'_i	Original circuit C after i iterations of randomization
C', C'_n	Original circuit C after n-iteration randomization is finished
Ω	circuit *basis*. Ω is a set of Boolean functions such that $\Omega \subseteq \{\text{AND, NAND, OR, NOR, XOR, XNOR, NOT}\}$
$C_{X\text{-}Y\text{-}S\text{-}\Omega}$	the *class* of a circuit, indicating inputs (X), outputs (Y), size (S = maximum number of gates), and basis (Ω)
δ, $\delta_{X\text{-}Y\text{-}S\text{-}\Omega}$	circuit *family*, i.e., the set containing all circuits $C_{X\text{-}Y\text{-}S\text{-}\Omega}$
δ_C	family of circuits semantically equivalent to C ($\delta_C \subset \delta$)

The hardness of reverse engineering or its suitability to hide some original program information is normally linked with *unintelligibility* or *understandability*. The use of these terms has unfortunately not promoted robust theoretical discussion of actual/practical obfuscating transformations because intelligence and understandability remain human-centered concepts. Collberg and his colleagues [4] use metrics that in almost all cases correlate larger size and numbers of artifacts to the specific cost in time or resources of various software reverse engineering tasks. We prefer the ability to measure *indistinguishability* and *randomness* as more precise since both terms have context in traditional cryptography and information theory. In order to understand the fundamental/mathematical nature of heuristic-based syntactic transformations, our experimental environment considers the effects of large numbers and large classes of random choices applied to the structural level of a circuit. As a motivating context, we probe assertions of one obfuscation definition known as the *random program model* [17], which we review next.

2.1 Notation

In our context, we model programs specifically as Boolean circuits. A circuit over Ω is a directed acyclic graph (DAG) having either nodes mapping to functions in Ω (referred to as *gates*) or having nodes with in-degree 0 being termed *inputs*. We also distinguish one (or more) intermediate nodes as *outputs*. The basis is complete if and only if all functions f are computable by a circuit over Ω. The basis

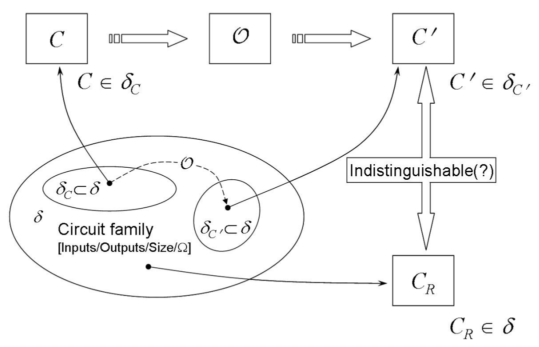


Figure 1: Random Program Model (RPM) [17]

sets {AND, OR, NOT}, {AND, NOT}, {OR, NOT}, {NAND}, and {NOR} are all known to be complete. One example of a complete 6-gate basis is Ω = {AND, OR, NOR, NAND, XOR, NXOR} which has basis size $|\Omega| = 6$. We summarize our notational style in Table 1.

2.2 Randomness as an Obfuscation Metric

When considering circuits, we typically use two primary analysis paradigms to describe them: how they *behave* and how they are *constructed*. We rightly consider software "behavior" as the blackbox functional characteristics (denotational semantics) of a circuit reflected by all possible input/output pairs while we can define circuit "construction" as the representation of its whitebox internal structure (the collection of language statements that define its topography).

We may define an obfuscating transformation $O(\cdot)$ as an efficient, terminating program which takes a circuit C as input and returns another circuit C': $O(C) = C'$. Of this assertion, all theoreticians and practitioners (that we are aware of) would agree. Beyond that, the majority of theoretical and practical models for obfuscation have at least two other requirements for the obfuscating program $O(\cdot)$, where $O(C) = C'$: *semantic equivalence* and *security*.

- **Semantic Equivalence**: $\forall x \in \{0,1\}^n : C(x) = C'(x)$, where n is the input size of C and $C' = O(C)$.
- **Efficiency**: There is a polynomial l such that for every circuit C, $|O(C)| \leq l(|C|)$.
- **Security**: A property that expresses some notion of information "hiding" or security guaranteed by $O(\cdot)$ for every possible circuit under consideration. The expression and measurement of the property varies from model to model: blackbox [1], indistinguishability [1], best-possible [6].

In [18, 17], a theoretical and practical understanding of obfuscation based on the *random program model* (RPM) is given. RPM posits that an *intent-protected* circuit when compared with any other circuit randomly chosen from a similar family (i.e., the same $\delta_{X\text{-}Y\text{-}S\text{-}\Omega}$, where $C \in \delta_{X\text{-}Y\text{-}S\text{-}\Omega}$) are indistinguishable as possible variants of the original circuit. Figure 1 gives our visual understanding of RPM. Intent protection itself is expressed as adversarial software exploitation for three main purposes:

1. Tampering with code in order to get specific results
2. Manipulating input in order to get specific results
3. Correlating input/output with environmental context

Compared to other theoretical understandings, RPM differs in the requirement for semantic equivalence and its definition of security. For its security property, RPM posits that if 1) the behavioral (blackbox) information gleaned from the obfuscated circuit C' has no correlation with the original circuit's behavior and 2) the structural (whitebox) topology of C' has no *more* correlation with the original circuit than any randomly chosen circuit of similar kind, then the intent of the original circuit has been protected. RPM also allows a different input/ouput semantics in the obfuscated circuit, as long as the intended, original output is recoverable.

To achieve this effect, RPM uses both semantically preserving whitebox and semantically recoverable blackbox transformations. In general, an obfuscating function has *only*

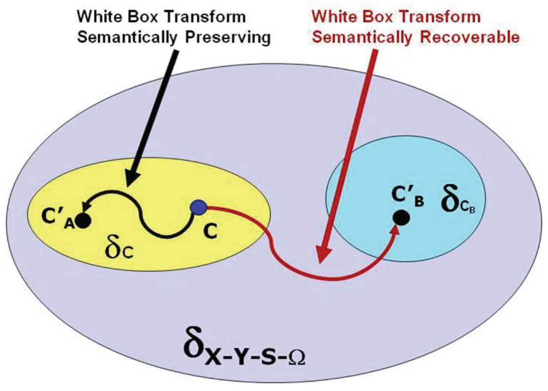


Figure 2: Obfuscation as Set Selection

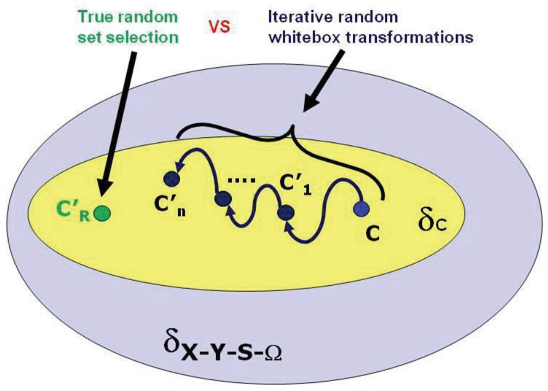


Figure 3: Random Uniform Set Selection versus Iterative Random Selections

two possibilities: whitebox changes which induce a blackbox transformation on the input/output and whitebox changes which preserve blackbox semantics. An obfuscator may change the whitebox structure of a circuit so that blackbox input/output relationships of the original circuit C are changed. Likewise, an obfuscator may change whitebox structure in such a way so that semantic equivalence with C is preserved. We illustrate this distinction in Figure 2 and note that we can alternatively view obfuscation as a set selection process.

2.3 Uniform Set vs. Iterative Selection

We design a framework that supports both semantic preserving/semantic recoverable transformations. For sake of brevity, we limit our discussion in this paper to the *whitebox, semantic-preserving* component. In other words, we only consider experiments where algorithms are sequenced, semantic-preserving structural transformations based on random or deterministic choices arranged in some random or deterministic manner. As Figure 2 illustrates, we can view an obfuscator as a program that selects programs from a set of functionally equivalent variations (i.e., polymorphic versions).

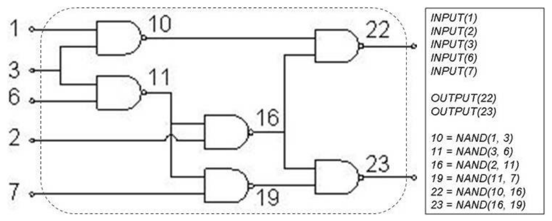
For example, all semantic-preserving obfuscators that produce a variant of circuit C, where $C \in \delta_C$ and $\delta_C \subset \delta_{X\text{-}Y\text{-}S\text{-}\Omega}$, will select some (other) element of δ_C, regardless of the theoretical model we choose to describe its security. We may conceive of one obfuscation goal and measurement criteria as whether we have maximized the randomness between the intermediate gate structure of C and the intermediate gate structure of its variant (C'_A in Figure 2). This translates to the goal of creating the best variant (in terms of confusion) that still accomplishes the same function as C.

RPM assumes that the best-possible obfuscator under this criteria would be one that chooses a circuit variant C' from the entire set of functional equivalents (δ_C in Figure 2) in a random, uniform manner. This random choice would represent our best attempt at producing a variant with random properties, or saying it another way, our best attempt at producing a variant that has confused and diffused the topological structure of the original circuit C. Even if we bound the size of the circuit family for (which is the primary factor in determining the set size of $\delta_{X\text{-}Y\text{-}S\text{-}\Omega}$ and thus the subset size of δ_C), enumerating all possible circuits with such a configuration is super-factorial in running time and storage requirements. However, if the circuit size is reasonably small, enumeration is feasible and we can select functional alternatives in a random, uniform manner. We leverage this fact in the construction of one half of our experimental framework (see Section 6) which deals with finding replacements for very small subcircuits.

As Figure 3 depicts, we summarize an ideal obfuscation selection process under RPM compared to achievable, practical obfuscation processes that we can build currently. RPM posits that we can do no better than an obfuscator which chooses an element in a uniform, random fashion from the set of semantically equivalent alternatives (i.e, δ_C). In Figure 3, C'_R represents such a choice. Current obfuscation techniques that perform iterative forms of confusion and diffusion, at best, only produce variants that are structurally close to each other. We are interested in how far we may alter an original circuit structure through small changes before it becomes indistinguishable from a truly random variant. After performing some sequence of small transformations, we focus on how much intermediate gate information of the original C is revealed by the final variant (C'_n in Figure 3).

One motivating reason for developing a whitebox variation environment is to explore whether random iterative selections might eventually approach a truly uniform selection from a large circuit family. If it is possible, we would expect the distribution of obfuscated circuits that come from an iterative random selection sequence obfuscator to be indistinguishable from a random uniform set selection obfuscator. In either case, we base the ideal variant to be one that has least correlation (according to some definable metric) with an original circuit.

Since we are not concerned with function hiding itself, we limit our concern to measuring how effective we can create a randomized variant of some original circuit. Our framework provides us a way to represent circuits and design experiments with a large option space in how small random alternatives are created. We present next the environment itself with a description of how we carry out experiments.



```
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(6)
INPUT(7)

OUTPUT(22)
OUTPUT(23)

10 = NAND(1, 3)
11 = NAND(3, 6)
16 = NAND(2, 11)
19 = NAND(11, 7)
22 = NAND(10, 16)
23 = NAND(16, 19)
```

Figure 4: ISCAS Benchmark Circuit c17

2.4 Circuit Representation

Circuits have several manifest properties. We let $SIZE(C) = n{-}m{-}s$ denote that circuit C has input size n, output size m, and intermediate gate size s. For example, in Figure 4, $SIZE(c17) = 5{-}2{-}4$. Output gates are distinguished intermediate gates and together with the inputs define the denotational semantics of the circuit. We let Ω(C)={`NAND`, `XOR`} denote that circuit C has basis $\Omega = \{\texttt{NAND}, \texttt{XOR}\}$. For example, in Figure 4, Ω(c17)={`NAND`} and we would refer to $c17$ as a `NAND`-*only* circuit. For notational purposes, let Φ represent the set of all gates (intermediate or output) in a circuit C and let g_x represent a gate $g_x \in \Phi$. For example, in Figure 4, Φ = {10, 11, 16, 19, 22, 23}.

In addition, we indicate the level of a gate g within a circuit by $level(g)$, which is synonymous with the trace level of a gate within a signal propagation hierarchy, assuming that every output signal has a final level of 0 and some virtual level where its Boolean logic signal is computed. Also, we let $|level(g)|$ represent the number of gates belonging to a particular level within the circuit. For example, in Figure 4, all inputs ({1,2,3,6,7}) are at level 3, $level(10) = level(11) = 2$, $level(16) = level(19) = 1$, $|level(16)| = |level(19)| = 2$. Each node within the DAG of a circuit C constitutes a specialized node with an associated Boolean logic function, derived from $\Omega(C)$.

We define any subset of gates $\alpha \subseteq \Phi$ as a *subcircuit* of circuit C, and we use C_{sub} to help identify algorithmic selections. We designate an k-gate subcircuit as a selection by $[a_1, a_2, ..., a_k]$. As a final property of interest, a circuit (and by definition, any subcircuit) readily express its blackbox behavior by enumeration of all inputs, subsequent evaluation and propagation of signals on all intermediate gates, and recording of the corresponding output.

We refer to the full list of input/output pairs of the circuit as the *truth table*. The blackbox behavior of such a circuit may be succinctly expressed by the output signals corresponding to a canonical ordering of the 2^n inputs, which we refer to as the circuit *signature*. For instance, the signature for a 2-

input and 1-output Boolean logic gate with AND functionality has a signature $< 0001 >$ while a 2-input OR logic gate has signature $< 1110 >$.

3 Experimental Configuration

We derive experiments based on textual descriptions of Boolean logic in BENCH format [7] and utilize a Java-based graph library package to support graph-based manipulation of the associated circuit DAG. Using this common DAG form, we compute a variety of graph-based, circuit-based, and semantics-based metrics. Our variation algorithm incorporate Kerckhoff's principles of cryptographic systems design [10]: namely, we give every possible choice made by the obfuscator as public knowledge while keeping only the precise set of steps used for a given obfuscation secret (much like the only secret part of a secure cipher should be the encryption key).

To perform whitebox transformation, we use a two-step *iteration* process which includes subcircuit *selection* followed by subcircuit *replacement*. Figure 5 illustrates the general notion, in two different views, of how we take an original circuit C and apply iterative changes to it that produce intermediate versions, C'_i. Each intermediate version, C'_i becomes the starting point for the next iteration which will produce the intermediate version C'_{i+1}. When we complete some n iterations of selection and replacement, the final variant becomes our candidate obfuscation variant, C'.

The large number of experiments which we may create using this approach derives from the nuance of each selection and replacement component. We say that a selection or replacement activity is *random* if we leave the choices of the algorithm completely open to a probabilistic dice-roll made by the algorithm (pseudo-random number generators suffice for this purpose). We say that a selection or replacement activity is *smart* if some criteria or user preference is used to guide or replace a probabilistic choice made by the algorithm. In the case of our selection/replacement algorithmic framework, the *obfuscation key* consists of the combined composition of all random and smart choices made during an experiment.

1. *Random selection*: Select a subcircuit $C_{sub} \subset C$ at random.

2. *Random replacement*: Select a replacement circuit $C_{rep} \in \delta_{C_{rep}}$ at random.

3. *Smart selection*: Only select subcircuits which have a particular property. If the subset of allowable selections contains more than one subcircuit, then one may be selected at random or based on another user-specified criteria.

4. *Smart replacement*: Similar to smart selection, only select replacement circuits from the library which have a particular property. If the subset of allowable selections contains more than one subcircuit, then one may be selected at random or based on another user-specified criteria.

We define a deterministic obfuscation *experiment* to be an 5-tuple: $(C, n, \xi, \sigma, \tau)$. We define the tuple as follows: C is an original circuit, n is the number of iterations, ξ is a set of selection algorithms with cardinality $|\xi| = n$ where $s_i \in \xi$ indicates the selection algorithm performed during iteration i, σ is a set of selection algorithms with cardinality $|\sigma| = n$ where $r_i \in \sigma$ indicates the replacement algorithm performed during iteration i, and τ is a set of gates that are are given selection priority during the incremental execution of the experiment. The *trace* of an experiment records all pertinent information and metrics across all iterations of the experiment as well. It would, for example, indicate for each iteration, which specific set of gates or subcircuit was chosen for selection and which specific set of gates or subcircuit was chosen for replacement. It is possible, for example, that no suitable replacement could be found for a given selected subcircuit and given the constraints of the replacement criteria. Thus, some iterations of an experiment may return the same original circuit.

Since we design each selection/replacement iteration as independent, atomic operations, we use τ to represent the notion of a global experiment state where we may target some gates of interest in the original circuit. For example, we may have a smart (criteria) based *experiment* that stipulates at least one original gate be considered by every selection algorithm, until all original gates are replaced. This criteria would, over some number of iterations close to the original circuit size, guarantee that all original gates of a circuit are replaced at least once. If, after accomplishing such criteria, we reset τ to be all gates in the current iteration variant, we would then guarantee (after some number of iterations) that all original gates with their newly introduced gates would be replaced at least once as well.

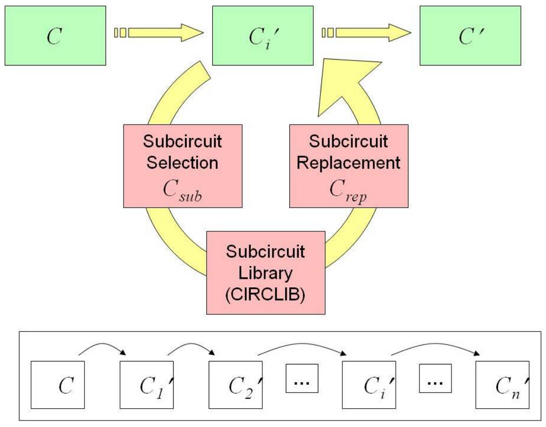

C
C_i'
C'
Subcircuit Selection C_{sub}
Subcircuit Replacement C_{rep}
Subcircuit Library (CIRCLIB)
C
C_1'
C_2'
...
C_i'
...
C_n'

Figure 5: Iterative Substitution and Replacement

4 Subcircuit Selection

Given an *experiment* defined as the tuple $(C, n, \xi, \sigma, \tau)$, ξ represents a set of selection algorithms and $s_i \in \xi$ indicates the selection algorithm used during iteration i. We define a subcircuit selection operation $C_{sub} = s(C, x, \gamma, \tau)$ with several characteristic attributes. The input to the algorithm is a circuit C, the (intermediate gate) size of the selection subcircuit x, the particular strategy $\gamma \in S$ (whether *smart* or *random*), and an optional set of gates τ that provide limiting criteria for the selection strategy itself. The set S of possible selection strategies (described below) must have all members defined a priori and we use the following set currently: S = {RandomSingleGate, RandomTwoGates, RandomLevelTwoGates, LargestLevelTwoGates, OutputLevelTwoGates, FixedLevelTwoGates, RandomAlgorithm}. The output of the algorithm C_{sub} is a circuit whose signature and $SIZE(C_{sub})$ forms the basis for functionally equivalent alternatives and replacement. As an example, iteration i that uses the `RandomSingleGate` strategy would be delineated as $s_i = s(C, 1,$ `RandomSingleGate`, $\emptyset)$, if we assume no experiment level criteria for selection/replacement.

4.1 Selection Strategies

In terms of selection approaches, we presently experiment with six different subcircuit-selection strategies. The `RandomAlgorithm` strategy chooses any possible selection strategy for a single iteration of the experiment and we define five as follows:

- `RandomSingleGate` $\longrightarrow$ Choose $g_1 \in \Phi$ in a random, uniform manner.
- `RandomTwoGates` $\longrightarrow$ Choose $g_1 \in \Phi$ in a random, uniform manner. Choose $g_2 \in \Phi$ where $g_2 \neq g_1$.
- `RandomLevelTwoGates` $\longrightarrow$ Choose $g_1 \in \Phi$ in a random, uniform manner. Choose $g_2 \in \Phi$ where $g_2 \neq g_1$ and where $level(g_2) = level(g_1) \pm 1$ or $level(g_2) = level(g_1)$.
- `LargestLevelTwoGates` $\longrightarrow$ Choose $g_1 \in \Phi$ such that $|level(g_1)| = \ell_{max}$ where ℓ_{max} represents the maximum size of all levels within the circuit: $\ell_{max} = \sqcup\{|level(g_x)| \mid g_x \in \Phi\}$. Choose $g_2 \in \Phi$ where $g_2 \neq g_1$ and where $level(g_2) = level(g_1) - 1$ or $level(g_2) = level(g_1)$.
- `OutputLevelTwoGates` $\longrightarrow$ Choose $g_1 \in \Phi$ where g_1 is a distinguished intermediate gate (i.e, an output of the circuit). Choose $g_2 \in \Phi$ where $g_2 \neq g_1$ and where $level(g_2) = level(g_1) - 1$ or $level(g_2) = level(g_1)$.
- `FixedLevelTwoGates` $\longrightarrow$ Choose $g_1 \in \Phi$ where, for some user-provided level criteria k, $level(g_1) = k$. Choose $g_2 \in \Phi$

where $g_2 \neq g_1$ and where $level(g_2) = level(g_1) - 1$ or $level(g_2) = level(g_1)$.

- `RandomAlgorithm` $\longrightarrow$ Choose any selection strategy $\gamma \in S$ in a random, uniform manner.

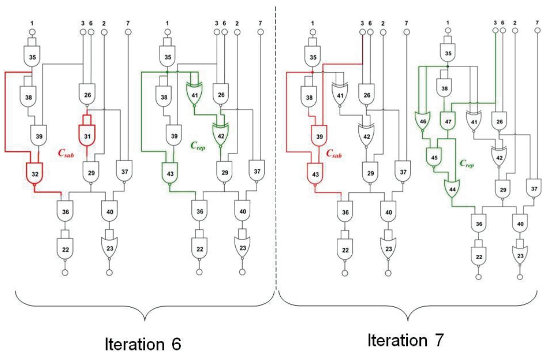


Figure 6: Iteration Example

Every random or smart selection strategy may be guided by criteria-based rules at the experiment level. When $\tau \neq \emptyset$, we modify the strategies listed by limiting the possibilities of at least the first gate chosen by the strategy. For example, an experiment that guarantees replacement of all original circuit gates would provide $\tau \subseteq \Phi$ to each iteration selection, which is to say that the strategy would make its first gate selection from the subset. If we used a `RandomSingleGate` strategy, the algorithm would instead choose $g_1 \in \tau$ in a random, uniform manner. Depending on the result of the replacement operation, if we *effectively* replace an original gate $g_x \in \tau$ (i.e., change fan-in, fan-out, or gate type), then we remove that gate from the set of possible first choices for the next iteration: $\tau = \tau \setminus \{g_x\}$.

4.2 Smart Strategy Limitations

A number of future, possible "smart" subcircuit selections can lead to NP-complete problems in generating the appropriate set of selectable subcircuits. For instance, a smart selection strategy based on subgraph isomorphism creates an NP-complete search, which is too computationally involved for large circuits. We may also develop selection strategies that look for specific Boolean logic functions (adders, multiplexer, decoder, comparator, etc.) for replacement. These would introduce greater than polynomial complexity to the obfuscator and would warrant heuristic options for the search.

5 Subcircuit Replacement

Given an *experiment* defined as the tuple $(C, n, \xi, \sigma, \tau)$, σ represents a set of replacement algorithms and $r_i \in \sigma$ indicates the replacement algorithm used during iteration i. We define a subcircuit replacement operation $C_{rep} = r(C_{sub},\ z,\ \psi, \Omega)$ with several characteristic attributes. C_{sub} is the circuit chosen for replacement, z is the requested gate size of the replacement circuit, ψ represents criteria that governs how we generate the replacement circuit library (described in Section 6.1), and Ω represents the basis choice of the replacement circuit. Given access to a selected circuit, we can derive the key characteristics that determine a replacement circuit library. $SIZE(C_{sub})$ gives us input size n, output size m, circuit (intermediate gate) size s. Combining this with knowledge of the basis, Ω, we have enough information to *create* or *query* a circuit family.

As we mention previously, the replacement component of our experimental environment actually accomplishes for small subcircuits what we would desire to do for large circuits. Recalling Figure 2, the subcircuit library generator (seen as `CIRCLIB` in Figure 5) first creates a set of circuits $\delta_{n\text{-}m\text{-}s\text{-}\Omega}$. From this set of circuits, we choose randomly and uniformly an alternative variant for C_{sub} from the functionally equivalent subset $\delta_{C_{sub}} \subset \delta_{n\text{-}m\text{-}s\text{-}\Omega}$. Therefore, $C_{rep} \in \delta_{C_{sub}}$ and, ideally, $\delta_{C_{sub}} \neq \emptyset$. Based on the circuit selected and the criteria for replacement, there are a countless number of configurations in which there are no alternative replacements and thus $\delta_{C_{sub}} = \emptyset$. For example, there are no [2-1-1-{`NAND`}] circuits that implement the `AND` Boolean logic function with signature $< 0001 >$. Likewise, we could also design many experiments that, when given a circuit C, only return the original circuit C.

Figure 6 illustrates two iterations (6 and 7) from an experiment with the `c17` circuit from

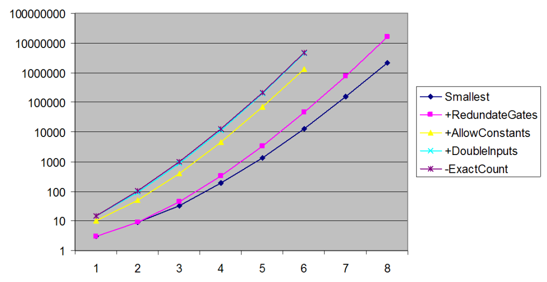


Figure 7: Circuit Library Sizes

Figure 4. The figure shows that in iteration 6, CORGI uses a two-gate selection strategy to choose the subcircuit $C_{sub} = [32, 31]$ and then, once it removes the subcircuit from the original, replaces it with $C_{rep} = [41, 42, 43]$. Both of these circuits belong to the C_{4-2-X} family. We note that the replacement increases the gate size of the overall circuit by one and increases the levelization also. Other effects of replacement may include changes to fan-in, fan-out, link length, unique input/output paths, unique paths through node, average paths per node, nodes per level, largest level, link length per node, average link length, and average nodes per level. We also note that, in iteration 6, gate 43 of C_{rep} is essentially the same gate 32 of C_{sub}: though renumbered, the gate has the same logic function, fan-in, and fan-out relationship. Figure 6 also demonstrates how the next iteration (7) of the experiment use a two-gate selection strategy to choose the subcircuit $C_{sub} = [39, 43]$, which resides in the C_{3-1-2} family and replaces it with a functionally equivalent $C_{rep} = [44, 45, 46, 47]$ which belongs to the C_{3-1-4} family. This example illustrates that we may grow the circuit size by virtue of replacing a circuit of size s with one of $s + 1$, $s + 2$, and so forth.

We note here that, if viable replacements were possible, we could easily replace size s subcircuits with functionally equivalent versions of size s, $s - 1$, or $s - 2$. It should make sense, that there are no single-gate replacement circuits for single-gate selection circuits, and there are some, but few, numbers of single-gate replacement circuits for two-gate selections. We discuss some of these relationships next, but point out that gate size and basis type drive the size of potential library classes. Currently our primary experimentation centers on single and two gate selection strategies, thus limiting our ability to report on optimizing or identical-size replacements at this time.

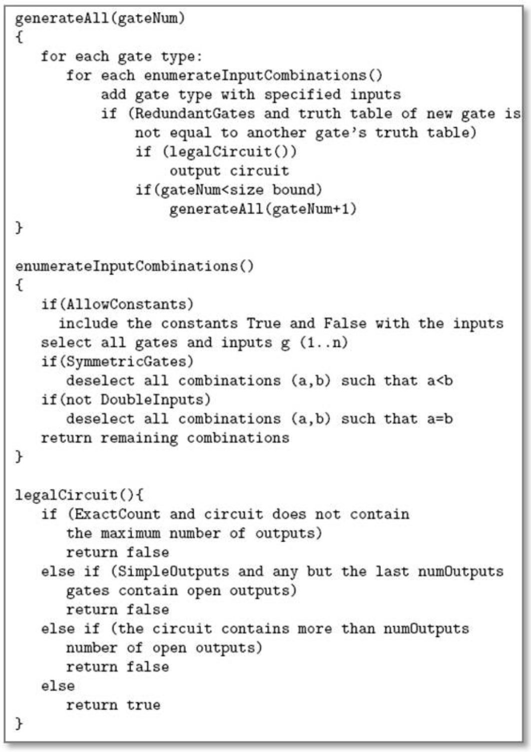
5.1 Library Generation Algorithm

Currently, we define only one iterative replacement algorithm but provide several basis-transforming operations (NAND-*only* , NOR-*only*) and structure-transforming operations (decompose multiple fan-in gates to dual fan-in, convert to sum-of-minterms form, convert to product-of-maxterms) that work at the whole-circuit level or do gate-by-gate replacement for all gates within the circuit. We focus currently on the use of purely random replacement choices versus smart options and describe next our recursive algorithm that enumerates circuit possibilities to produce a characteristic circuit family. We conceptually view this as the creation process for the circuit family $\delta_{X\text{-}Y\text{-}S\text{-}\Omega}$ seen in Figure 2, where we start with the knowledge of input size, output size, gate size, and basis.

We begin with a discuss of what constitutes a "legal circuit", because the generation algorithm must enumerate all possible graphs which conform to a set of combinational logic constraints. Assuming that all circuits consist of inputs and a set of one or more gates with exactly two inputs each (2-input/1-output logic function gates), some of which we treat as outputs, there are still a few questions to ask. We characterize these questions as true/false queries that form a Boolean 6-tuple, which we define as ψ in the description of a replacement operation: $r(C_{sub}, z, \psi, \Omega)$. We may vary these options for every replacement opportunity in an experiment, but typically choose a set of options ψ that remain constant for the entire sequence of iterations. Each option determines also how many circuits are produced, and we show in Figure 7 the exponential growth of library sizes (based on intermediate gates), based on different generation options for the C_{3-1-X} family as reference.

- SymmetricGates $\longrightarrow$ Are gates symmetric?

- `RedundantGates` $\longrightarrow$ Should we allow gates that are identical to other gates based on the inputs?
- `AllowConstants` $\longrightarrow$ Should we allow the circuit immediate access to the constants *True* and *False*?
- `DoubleInputs` $\longrightarrow$ Should we allow both inputs to a gate to originate in the same place?
- `ExactCount` $\longrightarrow$ Does the set contain all circuits within a certain size bound or only all circuits of an *exact* size?
- `SimpleOutputs` $\longrightarrow$ Which gates may be outputs?



```
generateAll(gateNum)
{
   for each gate type:
      for each enumerateInputCombinations()
          add gate type with specified inputs
          if (RedundantGates and truth table of new gate is
              not equal to another gate's truth table)
              if (legalCircuit())
                  output circuit
              if(gateNum<size bound)
                  generateAll(gateNum+1)
}

enumerateInputCombinations()
{
   if(AllowConstants)
     include the constants True and False with the inputs
   select all gates and inputs g (1..n)
   if(SymmetricGates)
      deselect all combinations (a,b) such that a<b
   if(not DoubleInputs)
      deselect all combinations (a,b) such that a=b
   return remaining combinations
}

legalCircuit(){
   if (ExactCount and circuit does not contain
      the maximum number of outputs)
      return false
   else if (SimpleOutputs and any but the last numOutputs
      gates contain open outputs)
      return false
   else if (the circuit contains more than numOutputs
      number of open outputs)
      return false
   else
      return true
}
```

Figure 8: Circuit Enumeration Algorithm

These options are the primary way we may make smart choices about the libraries that we choose to make random selections from. Our first initial generation algorithm was very basic. However, by accounting for the six creation options listed above, we present a final refined version of the recursive algorithm in Figure 8. Several of the creation options govern what we refer to as *practical* versus *theoretical* constraints on circuit construction. For example, it is highly unusual for real-world logic circuits to have gates with inputs both coming from the same source (the `DoubleInputs` option). The `SimpleOutputs` option also gives ability to preclude circuit replacement options that have dangling intermediate gates that are never actually used. We observe from running many (5000+) experiments with varying number of iterations that randomly chosen alternatives of two, three, and four gate size typically are considered "bad" from the perspective of normal VLSI/ASIC circuit design. As a first goal, we want to consider the effect of purely random replacements while learning what metrics best reflect either hiding properties of interest for reverse engineering purposes.

Table 2: Transformation Library 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

Once the enumeration algorithm generates (or locates) a circuit library with the appropriate circuit typology, it can find circuits within the family that match a particular (functional) signature. For our current experiments with 1 and 2 gate selection using 2, 3, or 4 gate replacement, we discover that it is more efficient to enumerate such libraries in memory versus access them from persistent data stores. As expected, we find that generation and retrieval of replacement candidates remains constant regardless of the circuit under consideration or the number of experiment iterations.

5.2 Library Creation and Size

Because of the recursive nature of the algorithm, we can see the factorial blowup in Figure 7 of possible circuit numbers, using the 3-1-X family as an example. We also note that there are orders of magnitude in size differ-

Table 3: Library Efficiency

	Usable Circuits	Total Circuits	Efficiency
1 Input - 1 Output	1,512	1,512	100.00%
1 Input - 2 Outputs	3,240	3,240	100.00%
2 Inputs - 1 Output	9,720	9,720	100.00%
2 Inputs - 2 Outputs	22,468	27,216	82.55%
3 Inputs - 1 Output	4,752	33,696	14.10%
3 Inputs - 2 Outputs	26,820	116,640	22.99%
4 Inputs - 1 Output	0	86,400	0.00%
4 Inputs - 2 Outputs	5,184	356,400	1.45%
Total	73,696	634,824	11.61%

ence based on the creation options. We have found from numerous experiments that ψ = (`SymmetricGates` = *true*, `RedundantGates` = *false*, `AllowConstants` = *false*, `DoubleInputs` = *false*, `ExactCount` = *true*, `SimpleOutputs` = *true*) produces circuits most like those we expect to see in traditional VLSI designs. We have discovered that certain option combinations produce gates which may be degenerate (all 1 or all 0), easily optimized away by a linear search algorithm, or produce redundant copies of either inputs, intermediate gates, or output gates. By using this approach, we also see the intractability of efficiently producing variants of larger circuits in a truly random way (if we want to use larger gate selection with full enumeration of the replacement alternatives). Table 2 illustrates the recorded disk space or memory requirements for several typical selection/replacement requests to the `CIRCGEN` library. To support 5 gate circuit replacements, we need almost 1 Petabyte.

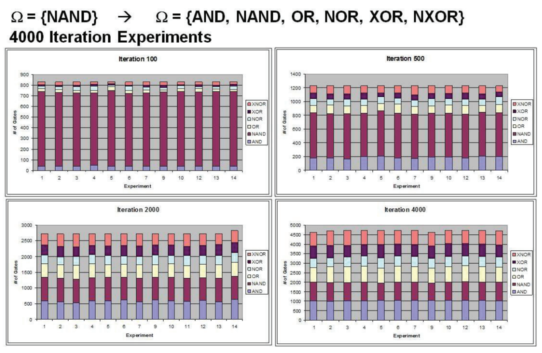


Figure 9: Uniform Gate Distribution Experiment

To improve efficiency further, we have the ability to cull out from a library circuits that have no expectation of every being used. This ability comes as an artifact of the way in which we select subcircuits to begin with and with the particular library creation options available to us. In particular, choosing a certain number of gates will result in variance between the actual circuit classes that contain equivalent circuits. For example, choosing two gates might result in a circuit with one input, two inputs, three inputs, or four inputs. In Table 3, we show the efficiency of choosing a subcircuit containing 2 gates and replacing it with a subcircuit containing 3 gates. We show the percentage of the generated subcircuits containing 3 gates which participate in transformation rules, meaning those which the algorithm would actually use in a replacement query. We note that the cost of storing only those circuits which can be used for replacements would be significantly less than the cost of storing all sub-circuits. We also note that a large body of future work remains to cull out circuit replacements which are by nature easy to find and reverse, though we leave the valid discuss of circuit reduction and logic minimization for another time. We are currently integrating various optimizing algorithms into the experimental framework as part of the variation process.

6 Obfuscating Measures of Interest

In thousands of experiments in our environment, we have ran various types of single and two gate selection and replacement experiments. Most of our experimental circuits come from ISCAS-85 Benchmark set or custom designed variants of comparators, carry-look-ahead adders, ripple-carry adders, multiplexors, decoders, and randomly generated circuits. Our maximum iteration run is 10000 currently, our largest *effective* selection and replacement size choice is 2 gates replaced with 2/3/4, and our largest real-world circuit for consideration has 3500 gates (we have processed randomly generated circuits with 10000 gates as well).

Besides understanding basic metrics that we may collect from circuits undergoing structural change, we find interest in properties of the circuit that point to effective information

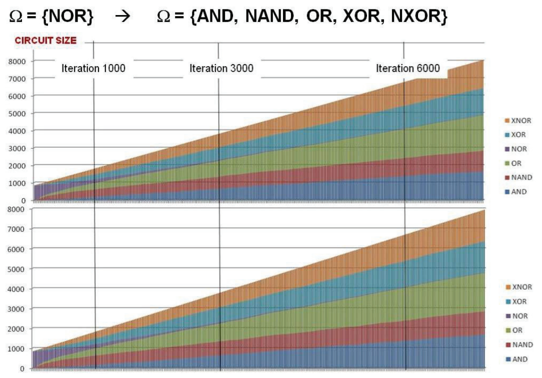


Figure 10: Full Replacement Experiment

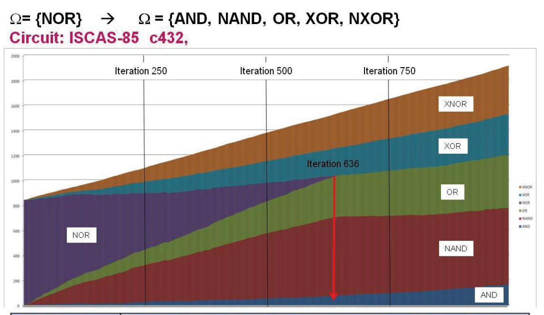
hiding or beneficial mutations that foster real-world circuit protection goals. There are several information hiding properties of interest if we focus on the hiding of intermediate gate signals. We observe that the power of a random iterative algorithm with small selections size (1 or 2) to accomplish signal hiding is very small: mainly because 1-gate and 2-gate selections cannot physically or logically support hiding (regardless of any replacement we may use). Single-gate circuits, for example, will never hide the original signal because there must be a gate in the replacement circuit (regardless of gate size) that keeps the output behavior of the original gate. Two-gate circuits will only provide opportunity for hiding when gates are arranged in more than level (within the virtual circuit create by the selection itself, not their level within the circuit). If two gates chosen are independently related, then on average, random and criteria-based random selection strategies will not on average choose the structure that is suitable for signal hiding.

As another facet of information hiding, we have particular interest in whether the algorithm effectively replaces gates of the original circuit itself. We note that hiding an original *signal* is only one possible side effect of replacing an original *gate*: other possibilities include copying the original gate signal (redundancy), inverting the original gate signal, copying an input/output signal, inverting an input/output signal, or introducing degenerate gates that always produce either 1 or 0. As an example, an verb"AND" gate that has dual inputs originating from the same source will always duplicate the input signal (producing a buffer) while an `XOR` gate that has dual inputs from the same source will always output a 0, (producing a degenerate gate). We focus here simply on the nature of the algorithm to completely remove an original gate from the final version and leave for future analysis the reversibility properties of the replacement.

6.1 Measuring Replacement

We report on three forms of experiments designed to measure gate replacement. We define gate replacement as the case where a gate chosen for selection does not appear in the replacement circuit in some renumbered form. This means that there is no gate in the replacement circuit with the same logic function (gate type), fan-in, and fan-out. We leverage the ability of our algorithm to choose the basis type of its replacement circuits to measure replacement. Given a replacement operation $r(C_{sub}, z, \psi, \Omega)$, we can vary Ω and thus guide the types of Boolean gates within the circuit over the course of the experiment. If we count the gate types of all gates within the circuit, over each iteration, we can tell when one type of gate no longer appears. If, for example, a circuit were a `NAND`-*only* circuit and we design an experiment where for all $r_i \in \sigma$, $\Omega =$ {`NOR`, `OR`, `AND`, `XOR`, `XNOR`}, then the we know when we find the iteration where the number of `NAND` gates in the variant circuit $= 0$, we have replaced all the original gates of the circuit.

Figure 9 illustrates the first type of experiment where we begin with a `NAND`-*only* circuit of around 700 gates. We set the replacement basis Ω to be all six possible types: Ω = {`NAND`, `NOR`, `OR`, `AND`, `XOR`, `XNOR`}. What we expect to see is that the circuit will manifest a fairly even distribution of gate types, assuming the selection/replacement operations are uniform as we expect. After conducting 14 separate 4000-iteration experiments using a `RandomTwoGates` selection strategy and 3-gate random replacement, Figure 9 shows the relative distribution of gate types for each experiment at iteration 100, 500, 2000, and 4000. What we observe are uniform distribution of gate types. Even for those gates that are NAND, they may not be originalNAND gates either, but we do not account for those in this experiment.

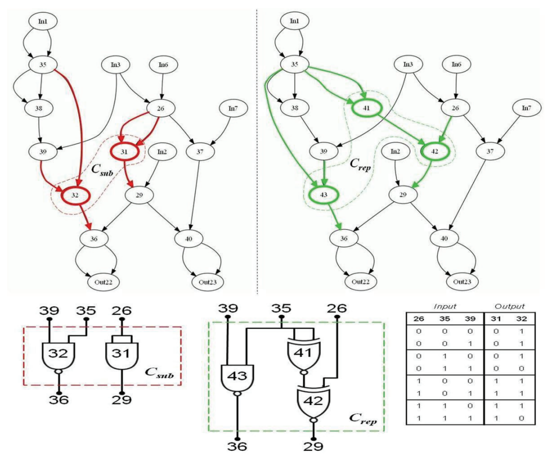


c432	120 gates (4 ANDs + 79 NANDs + 19 NORs + 18 XORs + 40 inverters)
Decomposed	230 gates (60 ANDs + 151 NANDs + 19 NORs + 40 inverters)
Decomposed NOR	843 gates (843 NORs)

Figure 11: Smart Experiment/Full Replacement Experiment

Figure 10 illustrates a second experiment using a NOR-*only* circuit of around 850 gates (the decomposed, NOR-*only* variant of the ISCAS-85 c880 circuit referenced in Figure 11). We show two (typical) results from 15 separate 7400-iteration experiments using a RandomAlgorithm selection strategy (weighted 75% towards RandomTwoGates selection strategy) and 3-gate random replacement. We use $\Omega = \{$NAND, OR, AND, XOR, XNOR$\}$ and expect to see an asymptotic decrease in NOR gates over time. In all of our experiments, purely random selection with no smart criteria at the experiment level always leaves some small number of original gates. This of course can be attributed to the fact that as the circuit grows in size, the small remaining original gate types become less likely to be chosen for replacement.

Figure 11 uses the same NOR-*only* circuit as a starting point and illustrates the results of a (typical) single experiment out of 25 where we chose a smart selection approach at the experiment level (reference Section 4). In this case, we set $\tau = \Phi$ and indicate that selection algorithms should favor original gates as their first selection choice. Each experiment was a 1000-iteration experiments using a smart RandomTwoGates selection strategy and 3-gate random replacement. We use $\Omega = \{$NAND, OR, AND, XOR, XNOR$\}$ and expect to see all NOR



Input			Output	
26	35	39	31	32
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	0

Figure 12: Crossover Example Context

gates to be removed from the circuit over time. As expected, in every experiment we saw all original gates removed from the circuit, on average, around iteration 630. In Figure 11, we know that the variant at iteration 636 contains no original gates. This illustrates the usefulness of smart-based variants of strategies which may be affected at the experiment level.

6.2 Control Diffusion and Redundancy

We conclude with a brief discussion of another circuit artifact of interest in both reverse engineering and mission assurance. By virtue of the two-gate selection strategies we specify, when gates in independent control paths are chosen for selection and replacement, the replacement circuit induces a control flow or diffusion within the circuit that did not exist before. Using the old adage that one man's trash is another man's treasure, the criticisms we give for small two-gate, cut set selection/replacements to provide signal hiding do on the other hand foster the ability to duplicate signals. When signals become duplicated in new control flows, this property may further goals such as fault tolerance or open up new methods of producing modular redundancy. Figure 12 illustrates this behavior, which occurs in nearly 95% of all RandomTwoGates selection strategy experiments. We highlight in this iteration example, $C_{sub} = [32, 31]$, which resides in the

C_{3-2-2} family. Gates 31 and 32 have no dependency or control flow between them before the selection and replacement operation. Once chosen for selection, however, their replacement induces a new control flow when $C_{rep} = [41, 42, 43]$. We leave for future work and results more extensive analysis of this phenomenon, but note here that the current set of our experimental strategies for two-gate selection create this behavior with high probability. We also leave for future analysis the resilience of such constructions to detection or removal.

7 Conclusion

In this paper we present a framework for whitebox circuit variation and describe our efforts to understand the effect of random and deterministic subcircuit selection and replacement on hiding properties of interest. We show the value of the framework for answering questions related to randomness as an obfuscation metric in considering circuit variants that may be used in reprogrammable hardware environments such as FPGAs. We give results of initial experimentation in support of specific questions such as gate replacement and gate diffusion/crossover. For brevity, we do not discuss all initial findings here but do expect in future work to report results related to a wide variety of questions: larger gate selection strategies, alternate possibilities for circuit library generation, impact of reduction or reversal algorithms, attempts for larger circuit library generation and storage, optimization and steady-state circuit replacements, and measurements of physical characteristics with real-world circuits.

References

[1] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. *Electronic Colloquium on Computational Complexity*, 8, 2001.

[2] Mohamed R. Chouchane, Andrew Walenstein, and Arun Lakhotia. Statistical signatures for fast filtering of instruction-substituting metamorphic malware. In *WORM '07: Proceedings of the 2007 ACM workshop on Recurring malcode*, pages 31–37, New York, NY, USA, 2007. ACM.

[3] Frederick B. Cohen. Operating system protection through program evolution. *Comput. Secur.*, 12(6):565–584, 1993.

[4] Christian S. Collberg and Clark Thomborson. Watermarking, tamper-proofing, and obfuscation: tools for software protection. *IEEE Trans. Softw. Eng.*, 28(8):735–746, 2002.

[5] Patrick Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Theor. Comput. Sci.*, 277(1-2):47–103, 2002.

[6] S. Goldwasser and G. N. Rothblum. *On best-possible obfuscation*, pages 194–213. 4th Theory of Cryptography Conference, TCC 2007. Proceedings (LNCS Vol. 4392). Springer-Verlag, Germany; Berlin, 21-24 Feb 2007.

[7] M.C. Hansen, H. Yalcin, and J.P. Hayes. Unveiling the iscas-85 benchmarks: a case study in reverse engineering. *Design & Test of Computers, IEEE*, 16(3):72–80, 1999.

[8] Michael Huth and Mark Ryan. *Logic in computer science: Modelling and Reasoning about Systems.* Cambridge University Press, 2004.

[9] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. pages 463–481. CRYPTO, 2003.

[10] Auguste Kerckhoffs. La cryptographie militaire. 9:5–38.

[11] J. Todd McDonald, Yong C. Kim, and Alec Yasinsac. Software issues in digital forensics. *ACM Operating Systems Review*, 42(3), April 2008.

[12] J. Todd McDonald and Alec Yasinsac. Applications for provably secure intent protection with bounded input-size programs. In *Proceedings of the International Conference on Availability, Reliability and Security (ARES 2007)*. IEEE Computer Society, 10-13 April 2007.

[13] Mila Dalla Preda and Roberto Giacobazzi. Semantic-based code obfuscation by abstract interpretation. In *ICALP*, pages 1325–1336, 2005.

[14] T. Sander and C. F. Tschudin. On software protection via function hiding. *Information Hiding*, pages 111–123, 1998.

[15] Kris Tiri and Ingrid Verbauwhede. Design method for constant power consumption of differential logic circuits. In *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*, pages 628–633, Washington, DC, USA, 2005. IEEE Computer Society.

[16] Andrew Walenstein, Rachit Mathur, Mohamed R. Chouchane, and Arun Lakhotia. Normalizing metamorphic malware using term rewriting. In *SCAM '06: Proceedings of the Sixth IEEE*, pages 75–84, Washington, DC, USA, 2006. IEEE Computer Society.

[17] Alec Yasinsac and J. Todd McDonald. Tamper resistant software through intent protection. *The International Journal of Network Security*, 7(3):370–382.

[18] Alec Yasinsac and J. Todd McDonald. Of unicorns and random programs. In *Proceedings of the 3rd IASTED International Conference on Communications and Computer Networks (IASTED/CCN)*, 8-10 Nov 2005.

[19] Yu Yu, Jussipekka Leiwo, and Benjamin Premkumar. Hiding circuit topology from unbounded reverse engineers. In *ACISP*, pages 171–182, 2006.

[20] Yu Yu, Jussipekka Leiwo, and Benjamin Premkumar. Private stateful circuits secure against probing attacks. In *ASIACCS*, pages 63–69, 2007.