Program Intent Protection Using Circuit Encryption

J. Todd McDonald, *Member, IEEE*, and Alec Yasinsac, *Senior Member, IEEE*

***Abstract*—The question of whether obfuscators enjoy well defined security is an important issue to the security and software development communities. It is widely believed that general obfuscators do not exist. We consider circuit obfuscators that do not have either the functionality property or black box property described by Barak *et al.*: namely the obfuscated circuit is semantically different from the original circuit but the output of the obfuscated circuit is recoverable. Moreover, we show that a general obfuscator can be created in our model that is not subject to Barak's proof. We show the usefulness of such a security model for defining intent-protected programs. We assert the usefulness of a combined black-box and white-box protection based on the indistinguishability of circuits in a large selection class and present initial results for defining obfuscators with such qualities.**

***Index Terms*—Communication system security, Circuit obfuscation, tamper resistant software, code entropy**

I. Introduction

One definition of obfuscation is the ability to efficiently rewrite a program so that an adversary who possesses the obfuscation gains no advantage beyond having observable program input/output behavior. This intuition has received substantial research and applied attention, yet a gap currently exists between practical and theoretical obfuscation security [1,2,3,4,5,6].

The current de facto standard theoretical obfuscation model is the Virtual Black Box (VBB) paradigm [7]. Barak *et al.* prove that there is a family of functions that cannot be obfuscated in VBB, and thus that efficient, general obfuscators do not exist. Wee [8] proved that particular classes of point functions, whose result is true on one and only one input and false otherwise, are obfuscatable given certain complexity assumptions. Lynn *et al.* [9] provide variations for protecting point functions based on random oracles while Canetti [The current de facto standard theoretical obfuscation model is the Virtual Black Box (VBB) paradigm [10]. Barak *et al.* prove that there is a family of functions that cannot be obfuscated in VBB, and thus that efficient, general obfuscators do not exist. Wee [11] proved that particular classes of point functions, whose result is true on one and only one input and false otherwise, are obfuscatable given certain complexity assumptions. Lynn *et al.* [12] provide variations for protecting point functions based on random oracles while Canetti [13] demonstrates cases where oracles are replaced by hash functions. Goldwasser and Kalai [14] show that you cannot efficiently obfuscate function families with respect to a priori information given to adversaries—giving unconditional impossibility results under VBB unrelated to one-way functions.

Barak *et al.* claim that the virtual black box paradigm is inherently flawed. Researchers suggest that two directions are left to pursue based on these foundations:

1) Are there weaker or alternative models for obfuscation that provide meaningful results?

2) Can we construct obfuscators for restricted but non-trivial/interesting classes of programs?

In other words, can practical obfuscation methods be proven secure against some threats and attacks, but not necessarily all? In this paper, we offer an alternative model for describing obfuscation security strength based on the complementary notions of random programs and black-box semantic transformation [15,16]. We provide a basis for understanding *intent-protected* programs using this paradigm and consider obfuscators that make random selections from a set of black-box protected programs. As a result, we relax both the hiding property and the class of programs that are obfuscated. We purposefully produce obfuscated programs that are not semantically equivalent to the original version so that $M(x) \neq O(M(x))$ and we show that a general obfuscator exists in our model that is not subject to Barak's impossibility proof.

A. Intent Protection

We begin with an intuitive description of program intent protection, and define the model in the next section. We can think of intent protection as a game between a developer and an adversary that desires to (1) manipulate code in order to attain a known output effect, (2) manipulate input to attain a known output effect, or (3) understand the program's function for use with contextual information.

Consider an industrial application on a stolen laptop. An adversary may desire to know how the laptop owner generates

Manuscript received August 9, 2006. This material is based upon work supported in part by the U.S. Army Research Laboratory and the U.S. Army Research Office under grant number DAAD19-02-1-0235.

J. Todd McDonald is an Assistant Professor in the Dept. of Electrical and Computer Engineering, Air Force Institute of Technology, 2950 Hobson Way, Bldg 640, Rm 304-A, Wright Patterson AFB, OH 45433-7765 USA. The views expressed in this article are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

Alec Yasinsac is an Associate Professor in the Computer Science Department at Florida State University, 262 James Jay Love Building, Tallahassee, Florida 32306-4530, USA 850.644.6407, 850.644.0058 (fax), yasinsac@fsu.edu,

business or financial estimates, how their decision process works, or other business or organizational information. For a black box protected program, the enemy cannot determine the device's function from an arbitrary number of input-output pairs. However, if the enemy is sophisticated, they examine the executable code structure or analyze the application's control flow and data manipulations as they occur. Programs that are white box protected prevent the enemy from learning the program's intent by watching its execution. We consider models for expressing the security strength of such intent-protection obfuscators in this paper and next discuss our definition for program understanding.

B. Program Understanding

We consider four distinct, but related program understanding paradigms. In the first, we consider the generic, intuitive notion of "understanding" that an adversary's ability to anticipate a program's operational manifestation(s) reflects their program understanding. Secondly, an adversary may gain intent indications by comparing the obfuscated code, or segments, to known code libraries. Third, we recognize VBB's theoretical and practical importance. Finally, information content in program code is our primary focus.

There is a mountain of data encryption that provides important insights into understanding information and its representations. We contend that programs are no more than a special information class with well-defined syntax and semantics. Moreover, scrambling techniques are limited because the final form must adhere to this rigid syntax and semantics. However, program code information content is otherwise equivalent to information content in any other type of bit stream. For example, program code that is statistically indistinguishable from a random bit stream has negligible information content, as is shown in [12]. Of course, meaningful applications cannot simulate random bit streams because they must adhere to the target machine architecture. Thus, from an information theoretic viewpoint, the best we can hope for is to create an obfuscation that is indistinguishable from any other program in the target architecture.

II. Defining Obfuscation

Obfuscation's goal is to prevent an adversary from using the program's code to better understand the original program's intent. We say "better understand" because there are few, if any, applications where a client would execute completely unknown mobile code. So, obfuscation cannot prevent contextual understanding, just as encryption, by itself, can not prevent, say traffic analysis. However, there is much that we can do. One security approach is to identify threats or attack types, then engineer systems that overcome those known and potential threats. The two classical approaches for defeating program obfuscation are black-box and white-box analysis. These attacks are independent in that one need not exercise one in order to leverage the other and complementary in the sense that they can be used together to identify program properties and intent. We contend that an obfuscator that retains semantic equivalence to the original program cannot obfuscate a program that reveals its intent through black box analysis. That is, no matter how scrambled the code, any reasonable adversary can reveal the program's intent.

We characterize intent-protection schemes based upon black-box and white-box definitions. This framework defines random programs and leverages the inability to distinguish circuits of an appreciably large size. In the next subsection we give our definition for protecting against input/output analysis.

A. Black-box Protection

A natural way to try to identify a program's intent is to analyze known input/output parings. This approach is widely known as black-box analysis since the program is treated as an oracle (black-box) without any insight into its internal workings. Definition 1 describes black-box understandability based on the ability of the adversary to guess or compute the input of a program based on the observed output.

Definition 1. Program $P \rightarrow \{X,Y\}$ is *black-box understandable* if and only if, given an arbitrarily large set of pairs $IO = (x_i, y_i)$ such that $y_i = P(x_i)$ and y_j an arbitrary element of Y with (x_j, y_j) not an element of IO), an adversary can efficiently guess x_j such that $y_j = P(x_j)$ with greater than negligible probability. Otherwise, we say P is *black-box obfuscated***.**

Traditionally, obfuscation involved producing program versions, where one version is understandable, but the obfuscated version is not. As noted earlier, obfuscation in this model is impossible. We introduce a different model and consider obfuscators that take a program (p) and generate a new program (p') with an associated recovery program (r). The recovery program has the property that $p(x) = r(p'(x))$. The fundamental property of the model is that output of the executably obfuscated code (p') is not equivalent to the output of the original program (p), a property reflected in Sander and Tschudin's homomorphic encryption scheme [17] and others. Our obfuscation process uses a key that provides security control and allows correlation with data encryption paradigms, since its security is only dependent on the key.

In Figure 1, transformation process t generates a black-box unrecognizable program p' based on composing program p with a strong data encryption algorithm e so that for all $x \in X$, $p'(x) = e(p(x))$. This process protects p's intent against black-box analysis because p' is one-way.

Lemma 1. Any program that implements a cryptographically strong data encryption algorithm is black-box obfuscated.

Proof: Arbitrarily select the crypto-graphically strong data encryption algorithm E, a plaintext message m, and encryption key k. Assume E is black-box understandable. Then there exists $y = E(m, k)$ where an adversary can guess m given y with negligible probability. This violates strong data encryption.

Black-box obfuscated programs are the foundation to protect against out-of-band, white-box analysis.

Lemma 2. Programs that are not one-way cannot be obfuscated by an obfuscator where O(p) = p' if p'(x) = p(x) for all x.

Proof: Follows directly from Definition 1.

Theorem 1. Let t(p, E, k) = (p', r) be a process that creates program p' by composing the output of program p to the input of a data encryption program E. Then p' is black box obfuscated.

Proof: Follows directly from Definition 1 and Lemma 1. If E is black-box obfuscated, then p' is also black-box obfuscated since the output of p' is [also] the output of E.

By ensuring that every obfuscation is a strong one-way function, our model alleviates all black box analysis threats. An interesting and important side effect is that this property also simply and absolutely insulates our model against the Barak impossibility result. In [7] the impossibility proof technique relies on a Turing machine decider that can detect whether a given function is a certain type. In our model, this proof technique cannot apply, since every obfuscation is a one way function. There are no point functions (the type of function that Barak *et al.* use), nor are there any other functional program categories.

B. White-Box Protection

Classic security research reveals many reasons to seek strong program obfuscation theory and technology. One important application is to protect secret keys in untrusted host environments [18], which is one goal of our research. Protecting the seam between two composed programs is another canonical obfuscation goal that is central to our model. Recall that we compose the protected function (A) with an encryption function (E) to provide black box protection. If the adversary can identify the seam between A and E through white box analysis, the black box protection provided by E disappears.

White box security encompasses both cases. White-box security is the ability to shield program intent from code analysis. Thus, white box protected obfuscations protect against analysis intended to reveal embedded data, seams between functions, the number of functions, or other such program properties.

Traditional obfuscation applies data and control flow confusion techniques to complicate attacks, with little measurable protection. We intend to provide systematic, measurable defenses against white-box threats.

C. Comparing Random Data and Random Programs

Unbiased selection is one way to think of randomness. Generally, if we select an element from a population without bias (i.e. each population member was equally likely to be selected), that element is a randomly selected element of the population. The element itself is no more “random” than any other element; only the unbiased selection gave the element the random property. More specifically, we can only select a random bit if we can construct an unbiased selection process, where 1 and 0 are selected with equal likelihood.

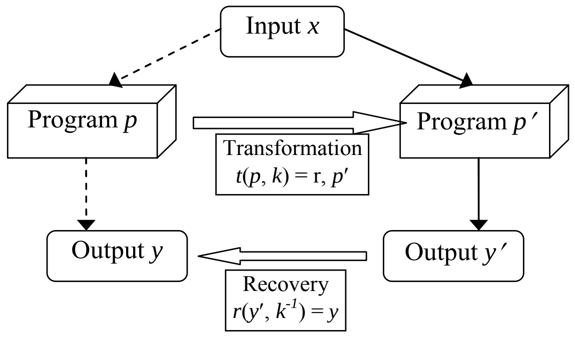

Figure 1: Black-box Obfuscation with

Unfortunately, this problem is impossible in practice (we cannot create a perfect coin, etc.) but science has rendered many excellent simulators that provide nearly random bit selection.

We stress here that only selections that are absolutely without bias produce random selections; if those selections are bits, we refer to their conglomeration as random bit stream. Perfect data encryption rests on generating cipher text that is indistinguishable from a random bit stream of the same length. The [accurate] intuition here is that cipher text that closely simulates randomness is unlikely to give away any hints about the corresponding plaintext.

We extend that notion to expect that a stream that reflects strong randomness properties also has high entropy and low information content and reveals only confusion, thus protecting secrecy, under inspection and cryptanalysis. We leverage this paradigm and transfer its notions from data encryption and protecting information secrecy to programs and code, and to protect program intent.

Random *programs* [15] are similar to randomized *data*

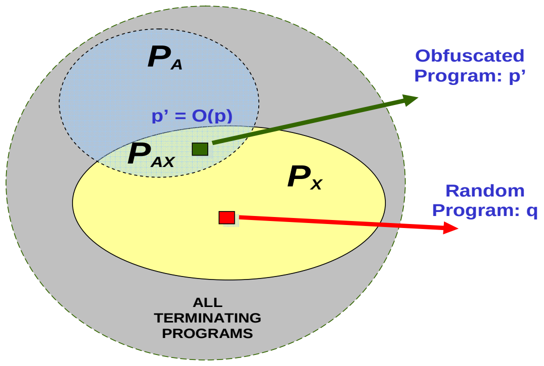

Figure 2: Random Program Selection

produced by strong data encryption algorithms. Digitized random data, for example, has no discernible patterns and in whose bit representations each bit is equally likely to be zero or one.

Similarly, given the infinite set P_A that contains all programs that implement some functionality A and the large, but finite set P_X that contains all programs of length X, the intersection set P_{AX} contains all programs that are of size X and that implement A. If we randomly choose q from P_X, we consider q to be a *random* program. Figure 2 illustrates the relationship between P_X, P_A, P_{AX}, and the selection of random program q.

Random selection is only valuable if it provides or ensures entropy in some form. Just as randomness properties for strings (no patterns or lengthy uniform sections, similar # of zero/one, etc.) only emerge as string length increases, so program entropy only emerges as program size increases. Intuitively, there are many more ways to write an unrecognizable program, e.g. to write in unintelligible spaghetti code, than there are to write versions that reveal their intent through static analysis.

Hypothesis 1. Entropy of randomly selected programs increases exponentially on the program size.

We accept this hypothesis without proof, though we make an argument relative to random circuits later. Given Hypothesis 1, we may characterize program encryption strength as its ability to select a program randomly from a set of equivalent, bounded implementations. Classically, such mechanisms are measured based on an adversary's ability to distinguish an executably encrypted (i.e. randomly selected) program *p'* of size *x* that implements A from a random program *q* of size *x*, that does not implement A. If the adversary can distinguish *p'* from *q*, then the program obfuscation / encryption may leak the intent of *p*. A secure obfuscation produces a program *p'* that is indistinguishable from a random program selected from the set of all programs the same size as *p'*.

To fully protect intent under white-box protection, an obfuscator must systematically confuse p' so that an adversary cannot learn anything about program intent by analyzing the static code structure or by observing program execution. The confusion must make the code and all possible execution paths that it produces display random program properties. For example, if a sophisticated adversary can distinguish between the functional program and the composite encryption program, they may be able to extract valuable intent information. Definition 3 extends the white-box protection definition and defines full intent protection as preventing all combined means of analysis to discover programmatic intent.

Definition 2. Given access to a random program oracle which transforms any program *p* using algorithm *E*(*p*) into an encrypted version *p'*, and given full access to any encrypted program p'_x: After knowing any *n* pairs of original and encrypted programs $\{(p_1, p'_1), (p_2, p'_2), \ldots, (p_{n-1}, p'_{n-1}), (p_n, p'_n))\}$, an adversary that supplies a subsequent program p_{n+1} will receive p_{n+1}' from the oracle which is either: a random program (P_R) or the encrypted version of the program $p_{n+1}' = E(p_{n+1})$. The program $E(p)$ provides white-box protection if and only if the probability that an adversary is able to distinguish the encrypted program (p_{n+1}') from a random program (P_R) is ½ + ε, where ε is a negligible constant.

$$p_{n+1}' = \begin{cases} P_R & \Pr[p_{n+1}' = P_R] \leq \frac{1}{2} + \varepsilon \\ E(p_{n+1}) & \Pr[p_{n+1}' = E(p_{n+1})] \leq \frac{1}{2} + \varepsilon \end{cases}$$

Definition 3. Program *P* is *intent protected* if and only if it is protected against black-box analysis and white box analysis.

D. Comparing Data and Program Encryption

Data cipher security properties are analyzed in one of two viewpoints: 1) information theory or 2) complexity. Data encryption strength is often reflected by properties such as whether possible breaks are reducible to known hard problems (e.g., factoring). Asymmetric ciphers use trapdoor one-way functions based on algebraic groups or rings. Symmetric cipher security proofs, on the other hand, do not rely on number theory. Confusion, diffusion, and composition operations form the foundation for the Data Encryption Standard (DES), AES, RC4, etc. Security proofs leverage Shannon's perfect secrecy [19], though security confidence relies on the fundamental theory of cryptography, i.e. that no *easy* attacks on symmetric schemes like DES have been found despite voluminous research efforts over the years[1]. Symmetric cryptosystems rely on brute force exhaustive search as their strength metric. Yet, symmetric ciphers are widely accepted as strong, despite absence of mathematical proof formulations.

There are two analogous threads in program obfuscation research. The Virtual Black Box is the de facto standard "provable security" approach, pitting the ability of a Turing machine given obfuscated code against one with only oracle access to the original function. Conversely, we use *random programs* as a baseline to measure program intent protection through entropy. Figure 3 summarizes these notions.

Practical program obfuscation techniques are, in large part, observation generated. Software engineers have known for decades that certain program structures reveal more about program intent than others. These intuitions led to obfuscation techniques such as adding ruse code, eliminating structured constructs, generating "elegant" algorithms, type casting, code reordering, code interleaving, and many others. The foundation was that if structured, concise code is easy to understand, then non-structured, elaborate code must be difficult to understand.

Unfortunately, the software engineering model that seeks to understand code and the security model whose goal is to protect intent do not correspond well at their extremes. Specifically, to protect intent against sophisticated intruders is fundamentally different than revealing intent to maintenance programmers. Thus, program obfuscation techniques focus on confusing code, with little theory or evidence that independent mechanisms are complementary, or even that they are not counter-productive.

Asymmetric Data Encryption	Symmetric Data Encryption
Based on mathematical algebraic primitives	Based on repetitive permutation/substitution
Provably secure relative to mathematical theory	Time-tested, secure based on limited resources
Key-based, systematic, recoverable	
Seeks to create ciphered data with discernible randomness properties	

Program Obfuscation	Program Encryption
Spurious, heuristic, limited	Based on repetitive, heuristic use of Composed permutation/substitution primitives
Not provably secure in the general case (VBB); secure in limited contexts [8]	Time-tested / complexity (secure based on limited resources)
Mechanism-specific, non-generalized	Key-based, systematic, recoverable
Seeks to protect programs against specific attacks using specific techniques	Seeks to create ciphered programs with discernible properties of randomness

Figure 3: Comparing Data Ciphers with Program Obfuscation/ Encryption

We contend that we can measure confusion by comparing our systematically obfuscated code (hereafter referred to as "encrypted code") or circuits against random code or circuits. We adhere to Kerckhoffs' security principle and leverage substitution and permutation engines similar to symmetric key encryption techniques, thus consider our techniques program "encryption" rather than program "obfuscation". Our obfuscated modules are key-based and executable, unlike Aucsmith's approach [20] that utilizes a key to generate pseudorandom blocks of encrypted code that are decrypted just prior to execution).

Program (or *circuit*) *encryption* mechanisms are key-based functions, with corresponding recovery mechanisms. These algorithms produce programs with well understood randomness properties. Program protection algorithms which rely on confusion, diffusion, and composition strategies are not necessarily *weaker* than mathematically based functional-transformations such as homomorphic encryption schemes. To illustrate, consider permutation and substitution data ciphers. Permutation, or transposition shuffles the order, where the key dictates the shuffle order; when used alone as a data cipher, permutation diffuses data across cipher text but is not cryptographically strong alone. When applied by itself, it might be viewed as a method of data *obfuscation*. Data substitution (or replacement), when used as a lone cipher technique, confuses bits within a ciphertext but is not individually cryptographically strong either—it can also be rightly considered a form of data *obfuscation*. When these techniques are strategically composed, they can create strong encryption, evidenced in well-known symmetric ciphers like DES. Even though DES strength is difficult to mathematically express in other than brute force terms, it is a recognized strong cipher that has no known attacks *significantly* more efficient than brute force key discovery.

We leverage the *program encryption* analogy that uses confusion and diffusion that are not strong themselves, but when composed in systematic, round-based algorithms produce executably encrypted code. Program encryption security analysis is not tied to VBB; instead, we use the random program model and offer an alternative view for security analysis.

III. Integrating Circuit and Program Encryption

Under Definition 1, the goal of black-box intent discovery by an adversary is to establish the I/O relationship that exists for an obfuscated program *p'*. If the adversary cannot find the functionality class A given runtime analysis of the obfuscated version *p'*, black-box protection is achieved. By definition, the family of all programs that implement one-way functions consists of programs whose input/output behavior is hard to learn. The security game played with an adversary involves not knowing or being able to determine a program's the I/O class or functional category.

In Definition 2 we express how to measure whether an adversary has an advantage when given the obfuscated program (code) or circuit over oracle-only access to the original program. We analyze whether the adversary distinguishes the obfuscated program from a randomly selected program of the same size. This includes an adversary who not only performs black-box analysis but also performs static or dynamic analysis of the code itself, specifically to determine *program intent.* To reiterate, we do not attempt to prove *general* security against all-powerful adversaries—rather we seek a more narrowly defined goal of intent protection and a framework to evaluate security of practical obfuscation techniques.

A. Black-Box Program Intent Protection

In our model, obfuscation must protect against black box analysis, essentially preventing an adversary from gaining intent understanding by examining an arbitrary number of input-output pairs. A natural intuition is to consider only obfuscating one-way functions, whose output is inherently black box protected. We capture this notion in Definition 1.

[1] Observation from RSA Security, http://www.rsasecurity.com

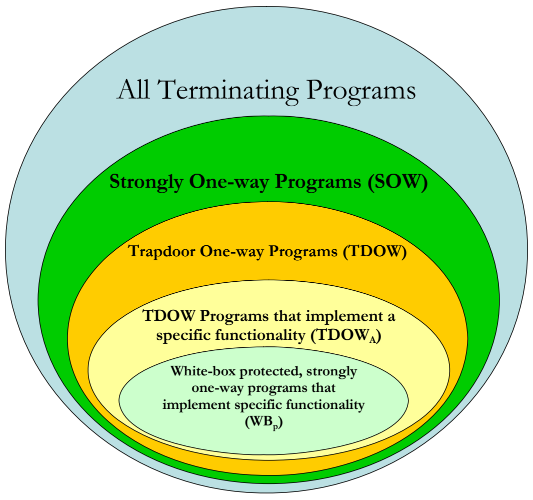
All Terminating Programs
Strongly One-way Programs (SOW)
Trapdoor One-way Programs (TDOW)
TDOW Programs that implement a specific functionality (TDOW$_A$)
White-box protected, strongly one-way programs that implement specific functionality (WB$_P$)

Figure 4: White Box Protected Programs

The term "trapdoor one-way function" describes cryptographically strong data ciphers (e) that use a plaintext x and key K to return a recoverable ciphertext $y = e(x,K)$.

We recognize a subclass of trapdoor one-way programs that has a special input/output relationship defined by a functionality class A and a program $P \in A$. Specifically, if we compose a program P that has a specific functionality A ($P \in A$) with a trapdoor one-way program ($e(x,K)$), we have programs consistent with those described in Theorem 1. We show this family of programs as the subset $TDOW_A$. In other words, given the transformation process t of Theorem 1 that creates a specific subclass of programs in $TDOW$, a recovery algorithm r recovers the intended output y of any program $P(x)$, given the output of $y' = p_i'(x)$, where $p_i' \in TDOW_A$. This set of programs is black-box intent protected under Definition 1.

From a compositional approach, black-box obfuscators O (under our Definition 1) that implement Theorem 1 are compilers that produce $p_i' = O(P)$ from an original program P and a strong, trapdoor one-way program e such that $p_i'(x) = e(P(x),K)$. Here $p_i' \in P'$ and indicates that the set P' contains all programs whose input/output relationship accommodates the domain of A and produces the range of E. A black-box obfuscator that meets Theorem 1 thus produces obfuscated programs whose input/output characteristics are consistent with E. We clarify that the selection of the particular class of functions E is a key-based decision part of the obfuscation process. Thus, E is randomized along with other parameters and the class A may itself include strongly one-way programs, trapdoor one-way programs, or data encryption algorithms.

B. *Intent Protection with White-Box Transformations*

At this point we refer specifically to Boolean circuits (using P' to refer to a set of circuits) and of obfuscators that algorithmically manipulate circuits. Considering the stronger form of protection (from Definition 2 and 3), we investigate obfuscators that perform circuit transformations based on indistinguishability from a random circuit. Such white-box obfuscators assume circuits $p_i' \in$ P' as a starting point. Since P' is infinitely large, we bound the possibilities by specifying only circuits with a maximum size N or less. For example, if E were the N-bounded family of Boolean circuits that implement the DES algorithm, all elements in E are circuits of size N or less that produce the mapping $E_{DES,K}$: $\{0,1\}^{64} \rightarrow \{0,1\}^{64}$ based on 64 bit input and a 56-bit key K.

The specified maximum circuit size N represents the desired obfuscated circuit efficiency; we consider obfuscators that randomize a circuit in a way that circuit blow up is exponential unless bounded otherwise. The lower bound size of circuits in P' is based on the size of the most efficiently reduced circuits that implement $p_i'(x) = E(P(x),K)$. A maximum circuit size N bounds the number of circuits that implement E. Likewise, N bounds the number of circuits in the set of all trapdoor one-way functions. Figure 4 illustrates the relationship of sets SOW, $TDOW$, E, and P'.

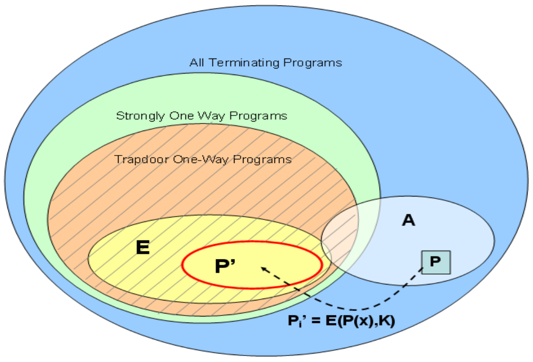

Figure 5: One Way Functions

We base white-box protection on an indistinguishability argument. As Definition 2 states, white-box intent protection is achieved if a circuit obfuscator (encryptor) produces an obfuscation of P that is indistinguishable from a random circuit P_R,. We use a random program (circuit) model as the basis for security and ask whether obfuscators exist that achieve intent protection.

We again leverage the well understood notion of traditional data ciphers to illuminate our paradigm. Strong data encryption produces ciphertext that is indistinguishable from a string chosen randomly from the set of all strings of the same size. Cryptographically strong data ciphers that use permutation, substitution combinations accomplish this successfully. Our desire is to design or find obfuscators that utilize circuit permutation and substitution to produce randomized circuits; these randomized circuits are indistinguishable with respect to P from any other circuit of comparable size chosen randomly. If random circuit selection provides white-box protection, as we contend, then our effort is reduced to finding mechanisms that produce suitably randomized "cipher code" (to coin a phrase). So, black-box analysis is defeated by semantic transformation on P to p_i' while static analysis is prevented by randomization of p_j'.

In Figure 5, we highlight the initial steps of an obfuscator that takes circuit p_i' and produces another semantically equivalent circuit $p_j' \in$ P'. The obfuscator must ensure that the selection

of p_j' from the set *P'* is uniform, random, key-based, and repeatable. Our claim is that if we randomly select a circuit from *P'*, this selection is indistinguishable from a random selection from the set *E*. We further investigate whether the selection of p_j' is indistinguishable from a random circuit selection taken from *TDOW* and from *SOW*.
We have two goals based on these foundations. First, if an obfuscator randomly selects a bounded size circuit from *P'*, this selection is indistinguishable from a bounded size circuit randomly selected from E. Secondly, we investigate whether obfuscators exist that randomly select circuits from *P'*. The secondary goal has to do with practical implementation of the first and we discuss our initial results toward that aim. In a sense, the second step corresponds with classic efforts to confuse code. While other approaches lack structure, in our approach, there is a well understood goal (randomization) and a metric (non-linearity).

C. *Distinguishing Random Selections of P' from P_R*

A circuit's behavior is reflected by its input/output mappings. Such mappings are commonly summarized either by truth table or the characteristic Boolean function of the circuit in some reduced, canonical form. By definition, circuits in *P'* are not analyzable by their input/output mappings—they are indeed hard to learn based on their membership in the set of all one-way functions. Given a circuit $\{0,1\}^{64} \rightarrow \{0,1\}^{64}$ with an appreciably large input size (64 bits) and appreciably large output size (64 bits), the truth table for such a circuit has 2^{64} rows. Without being able to analyze the input/output pairs of circuits in *P'*, no link to an original *P* is possible on the basis of input/output analysis alone. An adversary must then analyze circuits in *P'* using combined static and dynamic techniques.

There are uncountably many circuits in the unbounded sets *E* and *P'*. Given only circuits of size *N, E* and *P'* are finite and allow possible uniform selection. We assume a standard Boolean circuit definition as a directed acyclic graph that uses nodes for gates and edges that correspond to signal connections between gates. Given some number of possible Boolean gate types, there is a large but countable set of circuits of size *N* or less that implement *E* and that ultimately compose *P'*. *P'* is extremely large, but finite, and *E* is by implication much larger. We stipulate that at least one element of the set P' is selected by the obfuscation algorithm: the circuit created by black-box protection using the Theorem 1 (p_i') transformation. However, by applying both sub-circuit confusion and diffusion to p_i' in a round-based, repetitive manner, we select an equivalent, random circuit from *P'* which we refer to as p_j'.

Given a mechanism (obfuscator) that randomly selects a circuit from the set *P'*, we assert that such a circuit is indistinguishable from a randomly chosen circuit from the set *E*. The group of all permutations of $\{0,1\}^{64}$ is considered large enough to satisfy a brute-force discovery of the key (having $2^{64}!$ elements), even though some attacks on DES slightly reduce the number of plaintext/ciphertext pairs required to be successful. We draw a parallel and say that the number of

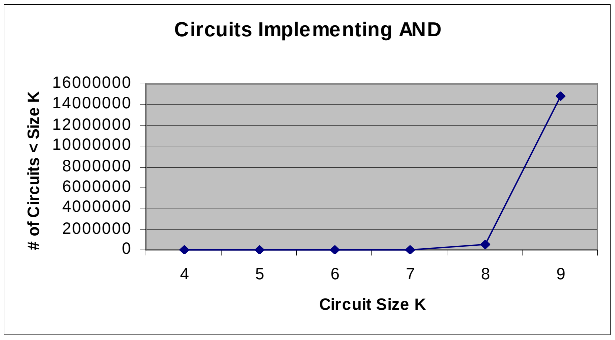

Table 1: Exponential Circuit Equivalence

representations for circuits that implement *P'* with characteristically large input/output $\{0,1\}^{64} \rightarrow \{0,1\}^{64}$ form a pool for random selection. Recall that selection from P' does not preserve thte original functionality, but preserves the black box protected functionality. This selection distinction allows successful intent protection.

To amplify the size of sets *P'* and *E*, consider the set *B* of all circuits that implement the AND function. If we fix input size to be two (x_0, x_1) and output size to be one (x_{K-1}), *B* contains circuits of size 3 or above, where size is the number of edges in a directed acyclic graph representing the circuit with a truth table output of [0, 0, 0, 1]. We assume logic gates are binary functions for AND, OR, XOR, XNOR, NOR, or NAND. We can enumerate all node arrangement possibilities and with N or fewer edges. Through experimentation, we count the number of circuit representations that produce the characteristic [0, 0, 0, 1] function, demonstrating exponential blowup in the *possible* number of AND function representations as N increases. For example, when N is 4, there are 66 total possible circuit combinations (circuits of size 4 composed of any legal combination of AND/OR/XOR/XNOR /NOR/NAND gates) ; three of these circuits constitute the set B: [x_2=x_1 AND x_0] , [x_2=x_1 XNOR x_0, x_3=x_2 AND x_1], and [x_2=x_1 XNOR x_0, x_3=x_2 AND x_0]. Table 1 shows the increase of |B| as 81 (N=5); 971 (N=6); 22,881 (N=7); 581,203 (N=8); 14,793,117 (N=9); and so forth. The set B of circuits size 9 or less that all implement AND contains nearly 15 million circuits. Any selection from this set gives a circuit with equivalent logical *AND* functionality.

This illustrates that for *complex* functionality, the number of circuits implementing that functionality is large, but can be bounded. *E* and *TDOW* are much larger than *P'*, Yu *et al.* [21] demonstrate the ability to hide redundant gates that are part of a (small) garbled circuit by creating a uniform circuit topology independent of output size. Such uniform topology indicates that circuit transformations can prevent the adversary from identifying redundant gates. Assuming a linear increase to the circuit size, we see an increase in the complexity/understandability of a Boolean circuit by converting all gates to an atomic gate type such as NAND or NOR.

D. Creating Obfuscators that Randomly Select Programs

Given a binary string representing a circuit, our process selects legal sub-circuit substitutions-permutations that preserve circuit functionality. The resultant binary representation reflects these transformations and mimics data plaintext replacement with an equivalent cipher-text substring. Security strengty comes the ability to perform key-based operations that are random and uniform across the plaintext. This is normally accomplished one plaintext block at a time and returning recoverable ciphertext blocks. In confusion-diffusion approaches, each block is transformed by a series key-based operations that include some type of non-linear substitution on small portions of the string (4 bits for example) and then permutation across the entire string.

We leverage non-linear substitution for sub-circuit replacement within a parent circuit. Even thought circuits in P' are large, we deal with fixed (small) size sub-circuits and create sets (substitution boxes) of circuits that preserve functionality (i.e., produce the same truth table). The intuition is that given a bounded size circuit, if sub-circuits are randomly chosen and replaced repetitively (up to some number of rounds), the resultant circuit has properties consistent with a randomly selected circuit from the pool of circuits P'.

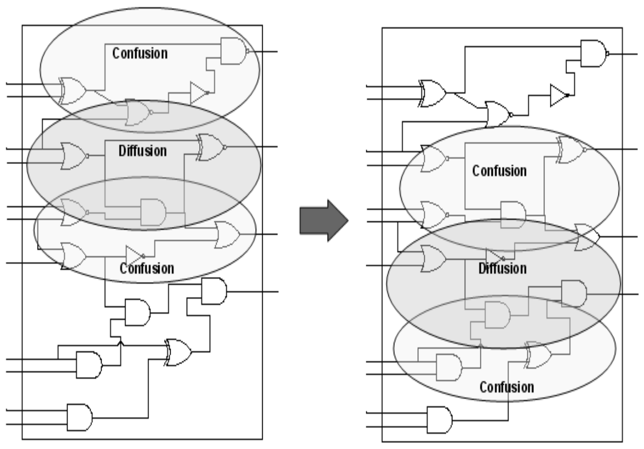

Figure 6: Circuit Substitution and Permutation

In the case of data S-boxes, bit strings are transformed from larger to smaller sizes. In the case of circuits, we replace a circuit of some (small) size with an equivalent circuit of closely smaller, equal, or greater size that has equal number of inputs and outputs. We assume initially that circuit substitution boxes produce equivalent sized circuits. Cryptographic algorithms based on the strength of non-linear substitution also rely on a given number of confusion/diffusion rounds. We define a circuit substitution operation as a non-linear equivalent replacement of a sub-circuit and a circuit diffusion operation as a substitution that comes as a result of two different replacement operations.

Figure 6 shows a notional circuit transformation where two other sub-circuit replacements diffuse the original functionality. We start with p_i' and apply round-based sub-circuit selection- replacement so that all p_i' gates are considered for replacement at least once. Each selection-replacement round within *P'* is key-based. Unlike block-ciphers, not all circuit definition blocks are contiguous. This dictates multiple selection/replacement rounds using various (small) input size and output size sub-circuits. A one-time, up-front cost is required to create equivalence classes for circuits—much like the requirement to design S-boxes part of symmetric data ciphers.

An obfuscator that takes a circuit p_i' and produces an equivalent circuit p_j' based on this process produces a string representation of p_j' that properties consistent with a random circuit. In particular, the binary string representations of p_i' compared to p_j' would map closely to the plaintext/ciphertext pair produced by a symmetric data cipher like DES. If the obfuscator functions in this manner, the resultant circuit is indeed indistinguishable from a random circuit. Figure 7 illustrates our vision for incorporating such a circuit obfuscator into a higher level algorithm that provides program intent protection.

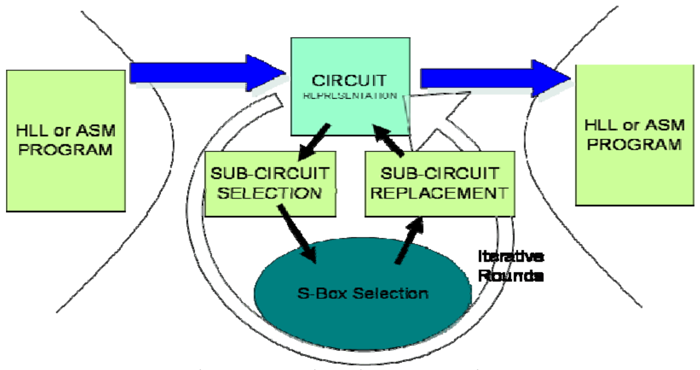

Figure 7: Circuit Encryption Process

IV. Conclusions

We offer a new program intent protection paradigm. We provide program obfuscation evaluation mechanism based on random program indistinguishability and prove how to attain absolute black-box protection. An important contribution is our white box protection approach. By developing obfuscators that randomize circuits through permutation-substitution rounds, the resultant circuit is indistinguishable from any circuit chosen from a collection of circuits in that size range. We appeal to both the indistinguishability of a circuit chosen from a large group and the inability of an adversary to realize the original input/output relationships of an original program *P* from its obfuscated version found in *P'*. Our methodology for considering obfuscation is analogous to the way in which ciphertext strings are analyzed in traditional data ciphers. Its proof or disproof is linked to the same way in which cryptographic primitives based on confusion/diffusion primitives are considered.

Most recent work in this area is based on algebraic/combinatorial primitives and they are considered more viable for proving security than their symmetric data cipher counterparts. While obfuscation is impossible in that model, our model is not subject to the same limitation. Our research leverages confusion technology that many symmetric-based algorithms have proven to be as hard to break and more efficient number theoretic approaches. Obfuscators, though they have been used for a number of years, face an uphill battle in understanding their theoretical value for security. We attempt to bridge that gap by posing an alternative means for viewing security and laying groundwork to see whether obfuscators exist that achieve program intent protection.

References

[1] G. Naumovich and N. Memon, "Preventing piracy, reverse engineering, and tampering," *Computer*, vol. 36, 2003, pp. 64-71.

[2] M. J. Atallah, E. D. Bryant, and M. R. Stytz, "A survey of anti-tamper technologies," *Crosstalk*, 2004.

[3] C. S. Collberg and C. Thomborson, "Watermarking, tamper-proofing, & obfuscation - tools for software protection," *IEEE Trans. on Software Engin.*, vol. 28, 2002, pp. 735-746.

[4] C. Linn and S. Debray, "Obfuscation of executable code to improve resistance to static disassembly," in *Proc. of the 10th ACM Conference on Computer and Communications Security*, 2003, pp. 290-299.

[5] N. Varnovsky and V. Zakharov, "On the possibility of provably secure obfuscating programs," *Perspectives of System Informatics, LNCS 2890*, Springer-Verlag, 2003, pp. 91-102.

[6] S. Hada, "Zero-knowledge and code obfuscation," in *Proc. of the 6th Int'l Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, vol. 1976: Springer-Verlag, 2000, pp. 443 - 457.

[7] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang, "On the (Im)possibility of obfuscating programs," in *Proc. of the 21st Annual Int'l Cryptology Conference on Advances in Cryptology, LNCS 2139*, Springer-Verlag, 2001, pp. 1-18.

[8] H. Wee, "On obfuscating point functions," in *Proc. of the 37th Annual ACM Symp. on Theory of Computing*, ACM Publishers, 2005, pp. 523-532.

[9] B. Lynn, M. Prabhakaran, and A. Sahai, "Positive results and techniques for obfuscation," *Eurocrypt'04,* 2004.

[10] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang, "On the (Im)possibility of obfuscating programs," in *Proc. of the 21st Annual Int'l Cryptology Conference on Advances in Cryptology, LNCS 2139*, Springer-Verlag, 2001, pp. 1-18.

[11] H. Wee, "On obfuscating point functions," in *Proc. of the 37th Annual ACM Symp. on Theory of Computing*, ACM Publishers, 2005, pp. 523-532.

[12] B. Lynn, M. Prabhakaran, and A. Sahai, "Positive results and techniques for obfuscation," *Eurocrypt'04,* 2004.

[13] R. Canetti., "Towards realizing random oracles: Hash functions that hide all partial information," In *Proc. of CRYPTO'97*, *LNCS 1294*, Springer-Verlag, 1997, pp. 455-469.

[14] S. Goldwasser and Y. Kalai, "On the impossibility of obfuscation with auxiliary input," in *Proc. of the 46th Annual IEEE Symp. on Foundations of Comptuer Science*, 2005.

[15] A. Yasinsac and J. T. McDonald, "Of unicorns and random programs," in *Proc. of the 3rd IASTED International Conference on Communications and Computer Networks (IASTED/CCN),* Marina del Rey, CA, 2005.

[16] W. Thompson, A. Yasinsac, and J. T. McDonald, "Semantic encryption transformation scheme," in *Proc. of the Int'l Workshop on Security in Parallel and Distributed Systems (PDCS 2004)*, San Francisco, CA, 2004.

[17] T. Sander and C. F. Tschudin, "On software protection via function hiding," *LNCS 1525*, 1998, pp. 111-123.

[18] S. Chow, P. Eisen, H. Johnson, and P. van Oorschot, "A white-box DES implementation for DRM applications," *Proc. of the 2nd ACM WS on Digital Rights Management, LNCS 2696*, Springer-Verlag, 2003, pp. 1-15.

[19] C. E. Shannon, "Communication Theory of Secrecy Systems", *Bell System Tech Jour*, v.28-4, pp 656--715, 1949

[20] D. Aucsmith, "Tamper-resistant software: an implementation," in *Proc. of the 1st Int'l Workshop on Information Hiding, LNCS 1174*, Springer-Verlag, 1996, pp. 317-333.

[21] Y. Yu, J. Leiwo, and B. Premkumar, "Securely utilizing external computing power," in *Proc. of the Int'l Conf. on Information Technology: Coding and Computing (ITCC '05)*, IEEE Publishers, 2005.