Applications for Provably Secure Intent Protection with Bounded Input-Size Programs

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

Abstract—**The de facto standard program obfuscation security model, termed the virtual black box (VBB), declares a program to be securely obfuscated if and only if an adversary can prove no more when given the obfuscated code than it can when only given oracle access to the original program. In this paper, we define and give methodology for a perfectly secure program intent obfuscation that is general and practical for bounded input-size programs, including those with input/output relationships that are easily learned. We also lay foundations for how to embed a key securely in a private-key encryption setting using such constructions.**

Index Terms—**Program obfuscation, virtual black box, software protection, information security, embedded keys**

I. Introduction

IN their well known result, Barek, Goldreich, *et al.* [1] state that no efficient, general obfuscators exist in the virtual black box (VBB) model. Under VBB, any candidate obfuscator $O(\cdot)$, when given input program P, must in polynomial time produce a semantically equivalent version P' that is roughly similar in efficiency to P. According to VBB, any predicate which is polynomially computed from P' must also be polynomially computed given oracle-only access to P. Though Barak *et al.* speculate that obfuscators *may* exist for *some* classes of programs, Goldwasser and Kalai [2] expand the impossibility result by covering cases where an adversary has some additional a priori information when given P'. They show that many natural classes of functions cannot be obfuscated with respect to auxiliary input, both when the auxiliary input is dependent of the function being obfuscated and even when the auxiliary input is independent of the function being obfuscated.

In this paper, we show how to produce a semantically secure obfuscation for $\{P_n\}_{n \in N}$, which is the class of programs with input size n. Unlike other results, the only definition we give for P_n is a polynomially-related bound b on the input size such that $n, b \in \mathcal{N}$ and $2^n \leq n^b$. Given such a bound, we show how to produce circuits that are efficient, semantically recoverable, and virtual black box protected with regards to the original program. The algorithmic complexity of the obfuscation is exponential, but, when bounded polynomially, is practical for a relevant class of programs—discussed next.

Manuscript received December 1, 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,

II. Motivating Examples

In order to frame our formulation, we illustrate first the class of programs we are interested in obfuscating, which are those with small (bounded) input size. Our construction is not comprehensive to all programs because the obfuscator or obfuscated circuit is not efficient for all input-size programs; yet we present four potential application categories that can naturally leverage the strength of our approach, though there are many other such potential application categories.

1. **Sensor nets.** Sensors (depicted in Figure 1) are canonically resource constrained and typically process small sized input, e.g. 16 bits. A manufacturer could create a perfect VBB obfuscation to field such a sensor to protect their intellectual property.

2. **Location information.** Positioning devices utilize numerically intensive functions. Mathematical input can often be very efficiently represented. Thus, location finding or tracking devices are potential perfect obfuscation applications.

3. **Financial transactions.** There is a clear need to protect programs that compute financial data. Many important financial programs take small mathematical input and, thus can be target applications for perfectly secure obfuscation.

4. **Protecting embedded keys.** Our second most important contribution is to recognize that *the perfect obfuscation we introduce can absolutely protect embedded key encryption algorithms in executing program code.* For an application with suitable input size, simply compose the code with an appropriate encryption algorithm and apply our construction to create the obfuscation. The key (and

the seam between the application and the encryption algorithm) are provably hidden within the obfuscation.

Take for example a sensor that is deployed in a remote operating location as illustrated in Figure 1. The output of the sensor is a broadcast stream of binary digits (64 bits at a time) that is carried by some means to a remote processing facility. If the sensor were captured by an adversary who had the capability to disassemble it and look at its internal structure, it may become obvious to the adversary (after some reverse hardware engineering process) that the sensor uses temperature readings and motion sensor related data. For temperature, an input size of 8 bits is used (capturing a range from -100^{O} C to 100^{O} C) and, for motion sensor data, 24 bits is required. The software inside the sensor thus takes in 32 bits of input and outputs 64 bits of data every time a reading is taken (all of which are observable by the adversary).

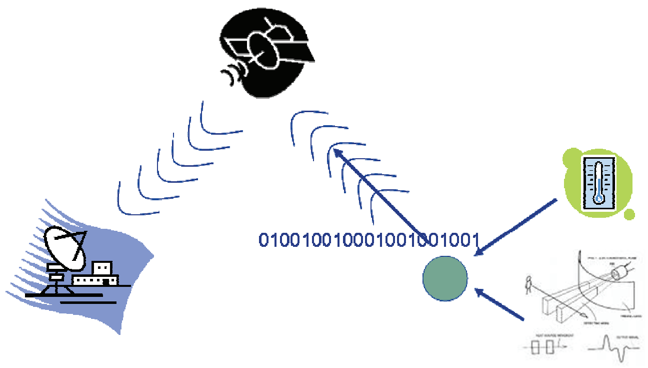

Fig. 1. Application Example for Bounded Input Size Program. A sensor node receives some number of (possibly small) input bits from its environment (temperature, motion data, etc.). The node performs some algorithm (that we want to intent protect) and produces an output stream that is sent to some (external) processing facility.

We want to protect the intent of the application software embedded in the sensor so that the adversary cannot foil the detection properties of the sensor or even understand what processed information is being relayed back to the processing facility based on the input. In other words, we want to ensure that both the input/output (black-box) relationships of the sensor and the algorithmic information (white-box) of the sensor's embedded circuitry are securely and provably protected. We produce the construction for an obfuscator that accomplishes these goals for the embedded program software.

III. Obfuscation Security Models

There are generally two appeals for measuring cryptographic security strength: information-theoretic and computational-complexity. The former is strongest and is based on whether breaks are *possible* (unconditionally) while the latter is based on whether breaks are *feasible*. In terms of *data* ciphers, an encryption scheme is considered insecure in the information-theoretic sense if the ciphertext contains *any* information about the plaintext. In the computational-complexity model, it only matters whether information about the plaintext that is contained in the ciphertext can be efficiently extracted.

With information-theoretic secrecy, an *ideal* security model is used to show that any candidate security solution is nearly as good as the ideal one. This implicit approach is quite different from the explicit complexity method which must define an adversary task and then show that the task is computationally difficult. Heuristic techniques and some computational approaches are deemed a form of "fuzzy" security (neither well defined nor precise) because they rely on capturing all possible adversarial actions. Defining such actions is difficult and computational/heuristic approaches may suffer from a use/break/tweak/use cycle. These foundational differences in defining security apply directly to the discussion of how we can securely obfuscate a program.

For some time, obfuscation researchers have found results based on both computational and information-theoretic models. The security characterization of obfuscation has been described as NP-easy [3], derivable in limited contexts [4,5], and proven to be NP-hard [6,7,8] / PSPACE-hard [9] based on specific protection mechanism. Heuristic approaches include techniques based on the hardness of interprocedural analysis [8], key-based generation of pseudorandom encrypted cope (decrypted just prior to execution) [10], and applying cryptographic primitives for constant hiding [4]. Collberg defines several complexity metrics that are designed to analyze the "hard to understand" quality of practical techniques [6]. In [11], Drake characterizes obfuscation as a refinement/proof process on data structures (versus algorithms).

Yu and his colleagues have recently found several positive results for completely hiding circuit topology in the information theoretic sense [12,13]. Canetti [14] and Lynn *et al.* [5] provide formulations for point-function obfuscation under the random oracle model while Wee [15] provides a secure point-function construction under VBB. We have developed a provably secure black-box program protection mechanism in [16] similar to that of Ostrovsky and Skeith's recent work [17] based on public-key obfuscation that produces encrypted, recoverable program output. Other research has focused on hardware supported program security [18], protection of embedded keys [19], and protecting mobile programs [20].

The VBB model of measuring *obfuscation* security essentially levies an information theoretic requirement: an adversary should learn no more when given the obfuscated version (i.e., executable ciphertext) of a program than it should when given black-box access to the original (executable plaintext) version of the program. Because of the impossibility results under VBB, it has been very hard (impossible) for any practical implementations of obfuscation to demonstrate *measurable* security properties.

We illustrate next a generalized obfuscation technique that produces perfectly protected circuits from *any* program with bounded (small) input size. The circuits are unconditionally and perfectly secure (at least from the "notion" of a virtual black box). Though the process is not efficient for all input-size programs, we use it nonetheless to illustrate that virtual black-box (perfectly secure) protection can be achieved for a relevant, real-world class of programs.

IV. Bridging Theory and Practice

Even though Turing machines are not physically constructible, they represent the theoretical underpinning of computer science; any best-case implementation of a Turing machine would require a (bounded) limit on infinitely defined tapes. If we desire a true information-theoretic proof that an obfuscated program does not leak any information regarding the original program, then we must show that the obfuscated program behaves exactly (and only) like an oracle for the original program would. By definition, an obfuscated circuit *P'* should not leak any more information about *P* than the oracle of *P* reveals.

A. Bounded Input-Size Obfuscation

Obfuscators that use *only* oracle-access to a function *P*, and not the original function *P* itself, have possibility for achieving information theoretic security. We state in Definition 1 the notion of a generalized program obfuscator related only to bounded program input-size.

Definition 1. (bounded input-size program obfuscator) *An algorithm O is an obfuscator for the class of b-bounded input size programs* $\{\mathbb{P}_n\}_{n,\, b\in N,\, 2^n \le n^b}$, *where* $P \in \mathbb{P}_n$ *if:*

1. **Semantic Equivalence:** *$\forall x$, $P(x) = P'(x)$, where $P'=O(P)$*
2. **Efficiency**: *There is a polynomial $l(\cdot)$ such that for every n, $b \in N$ where $2^n \le n^b$, and for every P in $\mathbb{P}$, $|O(P)| \le l(|P|)$*
3. **Perfectly Secure Obfuscation:** *For any PPT A, there is a PPT simulator S and a negligible function α such that for every n, $b \in N$ where $2^n \le n^b$, and for every $P \in \mathbb{P}_n$,*

$$\left|\Pr[A(O(P)) = 1] - \Pr[S^P(1^n)) = 1]\right| \le \alpha(n)$$

A function α: $\mathcal{N} \rightarrow \mathcal{R}^+$ is negligible if, for any positive polynomial p, there exists $N \in \mathcal{N}$ such that $\alpha(n) < p(n)^{-1}$ for any $n > N$.

In the information theoretic sense, perfectly secure obfuscation is defined by information gained by a PPT simulator S^P that has oracle-only access to some original program *P*. If a PPT algorithm uses only the information gained from an oracle of *P* to construct a semantically equivalent circuit/program *P'* for *P*, then it is impossible for any circuit/program *P'* created in a such a manner to leak *more* information than what the oracle for *P* could give. In particular, an oracle for *P* may be *simulated* by an algorithm that utilizes the truth table of *P*. The existence of such an oracle simulator for *P* assumes that the possible input range of *P* and its corresponding output can be fully enumerated, stored, and accessed.

We pause to clarify and amplify an oracle's capability. Classically, an oracle answers questions with no notion, reference, or intuition on our part as to how it knows the answer; we universally accept that the oracle's answers are correct. We utilize truth tables in our arguments because they capture the oracle's capability for answering function queries, since each answer, essentially, fills in a space in the function's truth table.

In their argument formulation, Barak *et al.* acknowledge a valid obfuscation exists for circuits in the following manner:

> *"Note that if we had not restricted the size of the obfuscated circuit O(C), then the (exponential size) list of all the values of the circuit would be a valid obfuscation (provided we allow S running time poly(|O(C)|) rather than poly(|C|))."* [1]

We explore this statement and define explicitly the constructions related to this possibility. The VBB impossibility proofs in general deal with (contrived) functions where the input size is too large for practical truth table enumeration—therefore a simulator with oracle access to an original program *P* (defined as S^P) can do no better than guessing based on oracle-queries. We consider instead the family of functions whose input size is small and therefore whose input/output behavior is not prohibitive for a simulator to enumerate.

Barak *et al.* also state that the foundation of (all) of their proofs derive from the "fundamental difference between getting black-box access to a function and getting a program that computes it, no matter how obfuscated" [1]. They go on to state that this difference *disappears* if the function is learnable completely from oracle (black-box) queries. Our interest in bounded input-size programs is that their truth tables can be obtained efficiently when they have a sufficiently limited input size.

Some functions are easily learnable in that they can be learned from partial truth tables. Our results address functions whose truth tables can be completely constructed in polynomial time from oracle access, and point out that even for functions whose complexity grows exponentially, truth table construction complexity simulates polynomial growth for small input sizes. This function class provides the opportunity to observe provably VBB protected circuit implementations.

A natural question to ask is: "How (can) protecting a circuit/program whose truth table can be computed provide security?" As we mention in our review of obfuscation security models, the value of an obfuscation model where the obfuscated version of a program/circuit is not semantically equivalent to the original program/circuit has already been demonstrated. In our ideal construction (under Theorem 3), the obfuscated program's truth table is black box protected and, thus, does not reveal anything about I/O-based intent. Moreover, the canonical circuit construction described in Theorem 1, when used as an obfuscation technique, reveals nothing about the original circuit structure, thus providing perfect white box protection.

Theorem 1: *Perfectly secure obfuscators exist for b-bounded input-size programs (under Definition. 1).*

Proof: Our proof is by construction. We give a three step obfuscator *O(P)* that takes any executable program **P**, generates the truth table from oracle access to *P*, and applies a Boolean canonical reduction on the truth table to produce a circuit that is semantically equivalent to P. Assume *n* is the input size of P and let $2^n \leq n^b$, for some user specified *b*.

Then: *O is a b-bounded input-size program obfuscator for the class of programs* $\{P_n\}_{n,b\in N,\ 2^n \leq n^b}$, *for any* $P \in P_n$, *under the following construction*:

Step 1. Using *P*, acquire or create S^P as an efficient oracle emulation of *P*.
Step 2. Generate the truth table for *P*, *T(P)*, by running S^P on all 2^n inputs of *P*. Assuming $P: \{0,1\}^n \rightarrow \{0,1\}^m$, *T(P)* is the $m{\cdot}2^n$ size matrix of input/output pairs obtained in the following manner: $\forall x$, $[x,y] = [x, S^P(x)]$, where S^P is a PPT simulator with oracle access to *P*.
Step 3. Create circuit *P'* by applying the algorithm for canonical complete-sum of products [21,22] to *T(P)*. $P'=\sum_{i=1,\ldots,n} \pi_i$ is in disjunctive normal form (DNF) where each product π_i is a conjunct of literals and each literal is either an input variable x_j or its negation x'_j $(1 \leq j \leq n)$. Minimize *P'* via heuristic minimal-sum of products algorithm such as Blake's reduction based on Shannon's recursive expansion.

1. *P'* is perfectly secure with respect to *P*. Since $P' = O(P)$, *T(P)* is *fully* derivable given *P* assuming some polynomially bound *b* on input size *n*. Given bounded size, the following relationship holds between any PPT simulator S^P and obfuscator *O*. Both can derive *T(P)* and thus a canonical circuit for P in polynomially bounded time.

$$\left|\Pr[A(O(P))=1]-\Pr[S^P(1^n))=1]\right| \leq \alpha(n),$$

for bounded n.

2. For $\forall x$, $P(x) = P'(x)$. By construction, *P'* precisely implements *T(P)*.
3. There is a polynomial $l(\cdot)$ such that for every $n,b\in N$ where $2^n \leq n^b$, and for every *P* in P, $|O(P)| \leq l(|P|)$. In the worst case, a *complete* sum-of-products expansion is composed of *m* outputs consisting of up to 2^n minterms composed of up to *n-1* products (AND) and up to 2^n*-1* summations (OR). The maximum size, $m2^n(n-1)(2^n-1)$, is $O(2^n)$ while the minimal possible size is $\Omega(m)$—representing where each output is constant 0. By bounding the input size of program P with *b*, the size for the complete sum of products expansion circuit becomes $O(n^b)$. We would not (in practice), use the complete sum of products expansion because much more efficient representations are possible. From the security aspect alone, however, any *more-efficient* derivation of the complete sum of products circuit retains the perfectly secure obfuscation (hiding) property.
4. The minimal SOP expression of *P'* is polynomially equivalent in input-size to the original *P* related to some polynomial bound *b*, because $n = |x_P|$ and $|P'| \leq n^b$.

We point out that obfuscators constructed under Theorem 1 produce perfectly white-box protected circuits (in the information theoretic sense) from bounded input-size programs, but assume nothing about the hardness or difficulty of *learning* the original program *P*. If the input/output of *P* (and thus any semantically equivalent version of *P* such as *P'*), reveals the intent or function of *P*, then no degree of white-box hiding can prevent the adversary from learning the function of *P* from the input/output relationships of *P'*. The truth-table derived construction of Theorem 1 perfectly hides only the *algorithmic* construction of *P*—and nothing more.

B. Protecting Embedded Private-Key Ciphers

In the VBB constructions, *P* is assumed to be a function whose input/output behavior is hard to learn to begin with. However, constructions under Theorem 1 point out two useful practical realizations when used in context to hard-to-learn, one-way, pseudorandom functions: truth-table-based circuit derivations provide a method to hide embedded encryption keys programmatically and perfectly secure obfuscated private-key encryption schemes are possible where the (unpadded) input size (of the plaintext) is bounded.

A block cipher is a function $E: \{0,1\}^k \times \{0,1\}^m \rightarrow \{0,1\}^m$ that take a *k*-bit key and an *m*-bit (block length) plaintext input and returns an *m*-bit ciphertext string. The inverse function *D*: $\{0,1\}^k \times \{0,1\}^m \rightarrow \{0,1\}^m$ takes the *k*-bit key and an *m*-bit ciphertext string and returns an original *m*-bit plaintext string. We let $E_K(M)$ denote the encryption of message $M\in \{0,1\}^m$ with a specific key $K \in \{0,1\}^k$ and let $D_K(C)$ denote the decryption (inverse encryption) of message $C\in\{0,1\}^m$ with a specific key $K \in \{0,1\}^k$. We assume that any block cipher *E* of interest to us is a strongly pseudorandom function that is a permutation on $\{0,1\}^m$, as defined for example by Goldreich in his textbook [23].

Several block-cipher-based, private-key encryption schemes exist with pseudorandom properties. DES with fixed message size has been characterized as a candidate one-way function (assuming one-way functions exist) among other algorithms such as RSA. The hardness of key recovery and the one-way properties of ciphers such as DES are well established and pseudorandom properties of the DES family is discussed by Bellare *et al.* in [24] and Goldreich in [23]. Our interest in the DES family of functions, including variants such as 3-DES, is the comparatively small block size of the plaintext (64 bits). Though the virtual key size of 3-DES is larger than 56 bits, we focus on DES nonetheless with its standard 56-bit key space.

In Definition 2, we specify the requirements for an obfuscator of block-based private-key encryption schemes (such as DES), that provides a semantically secure hiding of an encryption key. In essence, the obfuscator *O(K,E)*, under this definition, takes a private-key *K* and block encryption algorithm *E(K,·)* and returns $E_K(\cdot)$ such that no *key-recovery* attack can reveal the key *K* based on *analysis of the source code/gate structure* of E_K. Theorem 2 now gives the formulation for obfuscating a key-embedded block cipher under the construction of Theorem 1.

Definition 2. (private-key block encryption program obfuscator) *The tuple of PPT algorithms (KG,E,D,O) enforces perfectly secure obfuscation in the private-key setting with security parameter k and block-size m for the class of programs* $\{\mathbb{E}_{k,m}\}$ *where* $E \in \mathbb{E}_{k.m}$ *if:*

1. **Private Key Encryption:** *(KG,E,D) defines a pseudorandom private-key block encryption scheme with block-size m and security parameter k*:

   KG: a probabilistic algorithm which picks K (on input 1^k*, produces key K); assume KG never produces "weak" keys*

   E: $\{0,1\}^k \times \{0,1\}^m \rightarrow \{0,1\}^m$*, on input* $K \in \{0,1\}^k$ *and plaintext message* $M \in \{0,1\}^m$*, produces ciphertext* $C \in \{0,1\}^m$

   D: $\{0,1\}^k \times \{0,1\}^m \rightarrow \{0,1\}^m$*, for all* $K \xleftarrow{R} KG(1^k)$ *and* $M \in \{0,1\}^m$*,* $D_K(E_K(M)) = M$*,*
2. **Semantic Equivalence:** *Given* $K \xleftarrow{R} KG(1^k)$ and program $E \in \mathbb{E}_{k,m}$*,* $\forall x$*,* $E(K,x) = E'(x)$*, where* $E' = O(K,E) = E_K(\cdot)$
3. **Efficiency**: *There is a polynomial l(·) for every E in* $\mathbb{E}_{k.m}$ $|O(K,E)| \leq l(|E|)$
4. **Perfectly Secure Obfuscation:** *For any PPT A, there is a PPT simulator S and a negligible function* α *such that for every for every* $E \in \mathbb{E}_{k.m}$ and for every $K \xleftarrow{R} KG(1^k)$

$$\left|\Pr[E, A(O(K,E)) = 1] - \Pr[E, S^{E_K}(1^m)) = 1]\right| \leq \alpha(n)$$

We assume any distinguisher does not have access to the private key K but has knowledge of the encryption program E.

Theorem 2: *Perfectly secure obfuscators exist for b-bounded input-size private-key block encryption programs.*

Proof: Our proof is by construction. We give a three step obfuscator *O(·)* that takes $K \xleftarrow{R} KG(1^k)$ and block-cipher program *E* with block-size *m* and key-size *k*, generates the truth table from oracle access to *E(K, ·)*, and applies a Boolean canonical reduction on the truth table to produce a circuit *E'* that is semantically equivalent to *E(K, ·)*.

Let circuit $E' = O(T_{EK}) = O(K,E)$ be an obfuscation of the encryption program *E* with embedded key *K* (i.e. *E'* retains the functionality of *E(K,·)*) where T_{EK} is the truth table of *E(K, ·)*. Assume *m* is the input size of E and *n* is the virtual (unpadded) input size of the plaintext where $n \leq m$ and let $2^n \leq n^b$, for some user specified *b*. Let T_{EK} be generated through the PPT simulator S^E.

Then: *O is a b-bounded input-size private-key block encryption program obfuscator for the class of programs* $\{\mathbb{E}_n\}_{n,k,b \in N,\ n \leq m,\ 2^n \leq n^b}$*, for any* $E \in \mathbb{E}_{k.m}$

Given $E \in \mathbb{E}_{k,m}$ and $K \xleftarrow{R} KG(1^k)$

Step 1. Acquire an efficient implementation of *E*, S^E, to use as oracle emulation.

Step 2. Generate the truth table for *E(K,·)*, T_{EK} by running S^E on all 2^n inputs of *E*, where *n* is related to the polynomial efficiency bound *b*. Where $n < m$, pad each input with *m-n* zeros.

Step 3. Create circuit *E'* by applying the algorithm for canonical complete-sum of products to T_{EK}. $E' = \sum_{i=1,\ldots,n} \pi_i$, is in disjunctive normal form (DNF) where each product π_i is a conjunct of literals and each literal is either an input variables x_j or its negation x'_j $(1 \leq j \leq n)$. Minimize *E'* via minimal-sum of products algorithm such as Blake's reduction based on Shannon's recursive expansion.

From Theorem 1, *E*' has the same characteristics based on its construction and meets requirements for semantic equivalence, efficiency, and perfectly secure obfuscation.

We distinguish between the block-size of cipher *E* which is *m* and our desired (bounded) input-size *n*. We establish that $n \leq m$ and $2^n \leq n^b$ for some user specified *b*. Where $n = m$, no padding of the input is necessary for *E(K,M)*. Where $n = m$ is too large for a user chosen bound *b* (meaning there are not enough computational resources available to achieve truth table elaboration or the reduced sum-of-products circuit derivation), an input size reduction is necessarily in order to meet the efficiency requirements for a polynomial bounded circuit size on *E'* or polynomial time speed for O. Where $n < m$, we must choose whether to pad with *m-n* zeros or to pad with a (randomly) chosen *m-n* bit string. We assume padding with 0 for simplicity at this point but point out that our plaintext message space is now $\{0,1\}^n$ as opposed to $\{0,1\}^m$. The security ramifications where the adversary knows that a (possibly) reduced (virtual) block size is being used is a separate but related discussion to whether the adversary can recover the key *K* when given the source code (gate structure) of *E'*.

Consider, for example, that we could easily encrypt the *output* of our sensor from Figure 1 using an embedded-key DES program because the sensor outputs 64 bits of data at a time (which matches the block input size of DES). The encrypted computational result $\text{DES}_K(\textit{sensor}(x))$ could then be sent back to the processing facility, decrypted using the private key *K*, and then analyzed. The only stipulation given under Theorem 2 is that we have computational resources related to the bound *b* such that $2^{32} < 32^b$. The primary limiting factor is the input size of the sensor since the size of circuit is only polynomially related to the number of outputs (which would be a factor of the encryption algorithm *E*). If there are adequate computational resources to accomplish the truth table enumeration for the 32-bit input/64-bit output matrix, then circuit *E'* can be constructed and a perfectly secure key-embedded circuit can be used in the sensor.

Assume that the *output* of the sensor described in Figure 1 were 32 bits instead of 64. We are confronted with the fact that only 32 bits (not the total 64 bit block size) of DES are in view. For a DES program that takes messages that are 32 bits long, the job of the adversary is to find the one truth table out of the approximately 2^{56} possible truth tables (excluding those based on weak keys) that is based upon the specific *K* that is embedded in *E'*. For our specific sensor example, each truth table is a possibly 2^{32} enumeration (versus a 2^{64} enumeration) of entries corresponding to each message/ciphertext pair.

The obfuscators defined under Theorem 2 need only produce one of these truth tables in order to embed the key with perfect semantic protection in the circuit *E'*. The adversary (on the other hand), must enumerate up to 2^{56} such truth tables in order to use the circuit *E'* to pinpoint the particular key *K* embedded within. Because of the construction process for circuit *E'*, which is based only on the input/output relationships of an embedded-key encryption operation, the adversary cannot discover the key *K* by examination of the actual gates of circuit *E'*. In fact, the gates of *E'* only yields semantic information concerning the input/output behavior of E_K, and nothing more. The adversary can do no more than observe input/output pairs which are obtained from execution of *E'* itself.

We pause here to mention that an adversary that has possession of an (embedded) key-based cipher and knowledge of the underlying cipher/algorithm can actually pinpoint the particular key based on input/output pairs. If the key-space is within reasonable computational range, the adversary need only enumerate all keys and run a single input into the (known) encryption algorithm to produce a corresponding ciphertext output. This pair may then be compare with the input/output of the embedded-key cipher. Because of the one-way nature of the cipher, the plaintext/ciphertext pairs for all possible keys would uniquely pinpoint the embedded key—assuming the adversary can enumerate all key space possibilities. In the case of embedded-key ciphers, the adversary can perform a wide range of cryptanalysis attacks not possible when they only have possession of ciphertext or plaintext alone.

Given a possibly reduced message space for DES, the security of the circuit *E'* is certainly related more to the key-space of DES than the reduced message space 2^n versus 2^m. We can leverage this observation and replace the DES_{56} program with $3DES_{56}$, AES_{128}, AES_{512}, RSA_{512}, RSA_{1024}, or even an RSA_{2048} variant. In each replacement just mentioned, the efficiency of the obfuscator under Theorem 2 given a bounded input size (32-bits in our example) increases only in relationship to the additional running time incurred by the oracle for each prospective encryption algorithm to generate one truth table. The circuit size of *E'* does not vary based on the encryption algorithm chosen other than a linear variation based on additional output bits (64-bits versus 128, 512, 1024, etc.).

We can use public key encryption algorithms under this same construction with *both* public and private keys held *private*. This fits especially well with the computational model of a remote processing sensor data because the execution environment of the program (the sensor) does not require decryption of the data it is processing. We can also introduce randomness or variation into the way in which we construct the cipher itself; we may consider a key-based method of concatenating ciphers with other ciphers as a viable alternative

C. *Generalized Program Intent Protection*

Consider now a sensor that takes in 32 bits of data and produces 32 bits of input: an adversary may observe *much less* than 2^{32} input/output pairs of the sensor in order to adequately determine the programmatic intent of the sensor and therefore find an (effective) way to subvert it. Theorem 3 provides a basis to consider *any* bounded input-size program *P* that has (easily) learnable input/output patterns versus one-way relationships (like DES_{56}) that are assumed provably hard to learn. Figure 2 gives a notional/ specific view of this

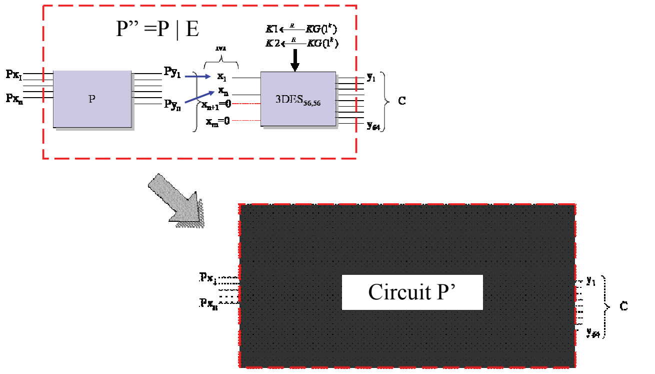

Fig. 2. Fully Generalized Bounded Input-Size Program Obfuscation. By composing program P with a (strong) data encryption cipher E. for all x, *P"(x) = E(P(x),K)*. By generating the truth table of P" for all inputs x, we produce circuit P' by applying standard Sum-of-Products derivation and simplification. P' is provably black-box and white-box intent protected (VBB) with respect to P.

construction using a program *P* and a 3DES encryption algorithm.

In this construction, a circuit *P'* is constructed from the catenation of the output of program *P* with a data encryption cipher *E* (which is a 3DES cipher that uses 2 keys in an encrypt-decrypt-encrypt relationship). As illustrated, *P* is a function $P: \{0,1\}^{|xP|} \rightarrow \{0,1\}^{|yP|}$ and *E* is a function $3DES_{K1,K2}$: $\{0,1\}^{64} \rightarrow \{0,1\}^{64}$ with two embedded keys. We assume the output size of *P*, $|y_P|$, is less than or equal to the input size of *E* (which for 3DES is 64 bits). The circuit *P'* is a concatenation of *P* and *E* that then becomes a virtual black box, such that for all input *x*, $P'(x) = 3DES_{K1,K2}(P(x))$.

We note that Ostravsky and Skeith define similar *public key* encryption-program-padding obfuscators in [17] with follow on work by Adida and Wikström [25] that implements such constructions in obfuscated mixnet programs. Our own semantic transformation obfuscation technique described in [16], which precedes these results, leaves open the possibility for either a symmetric or asymmetric encryption algorithm to effectively mask the original input/output patterns of an original program via concatenation of program output. These techniques of course *rely* on the underlying semantic security properties of the encryption algorithm that is used. We adapt such padding schemes to our construction to provide fully general bounded input-size program protection and prove such obfuscators exist for them in Theorem 3. We provide now a definition and theoretical construction that would take any bounded-input size program *P* that is (easily) learnable and concatenate the output of that program with an embedded-key strongly pseudorandom encryption algorithm. For brevity, we only specify the symmetric/private-key block cipher variant and follow the construction for obfuscated mixnets given in [25].

For notational purposes, let P|E refer to the concatenation of program P with the program E such that $(P|E_K)(x) = E_K(P(x))$*, for all x. Let P be defined as function* $P:\{0,1\}^n \rightarrow \{0,1\}^{|y_P|}$ *and* $E: \{0,1\}^k \times \{0,1\}^m \rightarrow \{0,1\}^m$*. Let* $P|E_K$ *for encryption algorithm E with embedded key K be defined as* $P|E_K: \{0,1\}^n \rightarrow \{0,1\}^m$.

Definition 3. (general obfuscator for easily learned programs with bounded input-size) *For PPT algorithms KG,E,D,O, obfuscator O provides perfectly secure obfuscation for the class of b-bounded programs* $\{\mathbb{P}_n\}_{n,k,b\in N,\ n<m,\ 2^n \le n^b}$ *where* $P \in \mathbb{P}_n$ *if:*

1. **Private Key Encryption:** *(KG,E,D) defines a pseudorandom private-key block encryption scheme with block-size m and security parameter k under Definition 2.*
2. **Semantic Equivalence:** *Given* $K \xleftarrow{R} KG(1^k)$ and program $P \in \mathbb{P}_n$, $\forall x,\ P(x) = D_K(P'(x))$ *where* $P' = O(K,P,E)$*. Furthermore,* $\forall x,\ P'(x) = E_K(P(x))$.
3. **Generality:** $(|x_P| = n) \le m$*, for all* $E \in \mathbb{E}_{k,m}$ *under Definition 2*
4. **Efficiency**: *There is a polynomial* $l(\cdot)$ *for every P in* $\mathbb{P}_n$, $|O(K,P,E)| \le l(|P|)$
5. **Perfectly Secure Obfuscation:** *For any PPT A, there is a PPT simulator S and a negligible function* α *such that for every* $n,b \in N$ *where* $2^n \le n^b$*, and for every* $P \in \mathbb{P}_n$ *and for every* $K \xleftarrow{R} KG(1^k)$
$$\left|\Pr[E, A(O(K,P,E)) = 1] - \Pr[E, S^{E_K}(1^n)) = 1]\right| \le \alpha(n)$$

Theorem 3: *Perfectly secure obfuscators exist for b-bounded input-size programs with easily learned I/O relationships.*

Proof: Our proof is by construction. We give a three-step obfuscator $O(\cdot)$ that takes as input $K \xleftarrow{R} KG(1^k)$, a block-cipher encryption program *E* with block-size *m* and key-size/security parameter *k*, and a *b*-bounded program *P* with input size *n* and output size $|y_P| \le m$, and where $2^n \le n^b$, for some user defined *b*. Let circuit $P' = O(K,P,E)$ be an obfuscation of any general program P with these constraints such that $\forall x,\ P(x) = D_K(P'(x))$ and, $\forall x,\ P'(x) = E_K(P(x))$.

Construct $P' = O(K,P,E)$ in the following manner:

Step 1. Given $K \xleftarrow{R} KG(1^k)$, let $P'' = P \mid E_K$. Acquire an efficient implementation of $ORACLE_{P''}$ to use as oracle emulation.
Step 2. Generate the truth table $T(P'')$ by executing $ORACLE_{P''}(x) = E(K,P(x))$ for all 2^n possible inputs *x* of *P*. Where $|y_P| < m$, pad the output of $P(x)$ with $m - |y_P|$ zeros.
Step 3. Create circuit *P'* by applying the algorithm for canonical complete-sum of products to $T_{P''}$, as defined in Theorem 1. Minimize *P'* via standard heuristic 2-level reduction techniques.

Then:

1. *E* is hard to learn and therefore *P''/P'* are hard to learn from black-box observation *alone*. However, recovery of any *intended* output of *P* (which is easy to learn) is possible because $\forall x,\ P(x) = D_K(P'(x)) = D_K(E_K(P(x)))$. Thus, the semantic equivalence between *P'* and *P* is established.
2. P' is a perfectly secure obfuscation with respect to P *and* the embedded-key encryption algorithm E_K because *P'* is produced only from oracle access to $P'' = P \mid E_K$.
3. $|P'|$ is *poly-(n)* given bound *b*.
4. O is a general, efficient obfuscator for *any* program P that runs in poly-(*n*) time, given a bound *b* related to the input size of *P*. *P'* is roughly equivalent in efficiency to *P*. The minimal SOP expression of *P'* is polynomially equivalent in size to *P* related to some bound *b*, because $|P'| \le n^b$. Note that the size characteristics of *P'* are related to the input size of *P* and not the possible input size of *E*.

D. Specifying Bounds

The generalized protection technique described in Theorem 3 depends clearly on the user-defined bound on program input size. The parameter *b* is clearly based on computational resources available to the user and reflects an acceptable limit on exponential growth. For truth table derivation and sum-of-product derivation, the storage available for the truth table *T(P'')* and the size of the resulting circuit *P'* become the primary barometer for determining bound b. An input size *n=32* bits reflects a bounded input parameter *b=7*, since $2^n \le n^b$, $2^{32} \le 32^7$. For a nominal desktop PC with 512 MB of memory, 60 GB of disk space, and a medium sized process (~1GhZ class), we have found *n=32* (bits) to be a reasonable program input size for truth table derivation / canonical circuit creation within a day. To illustrate the magnitude of resources (time and space) required for various input sizes, Table 1 shows the relationship between the computational bound parameter *b* and input size (*n*). We expect with adequate resources, input sizes of 128-256 could be realized for specific applications of interest.

V. Conclusions

The techniques demonstrated in this paper illustrate that *any* program, with small input size, is a candidate for practical and provably secure obfuscation. They also reaffirm a primary

TABLE I
RELATIONSHIP BETWEEN (PROGRAM) INPUT SIZE AND (COMPUTATIONAL) USER BOUND

Input Size n	Bound b	2^n	n^b
16	4	65,536	65,536
24	6	1.67×10^{7}	1.91×10^{8}
32	7	4.29×10^{9}	3.44×10^{10}
56	10	7.21×10^{16}	3.03×10^{17}
64	11	1.84×10^{19}	7.38×10^{19}
128	19	3.40×10^{38}	1.09×10^{40}
256	32	1.16×10^{77}	1.16×10^{77}
512	57	1.34×10^{154}	2.68×10^{154}

negative result under VBB that no *general*, *secure*, *efficient* obfuscator exists for *all* programs (any input size). However, the primary contribution we point out is that there are *relevant* program classes that can enjoy secure obfuscation under VBB.

As Barak states in [26] concerning the existence of unobfuscatable function families, “We believe the existence of such functions shows that the virtual black box paradigm for obfuscators is inherently flawed.” We believe also that a security analysis model that characterizes *practical* obfuscators is needed; particularly, a model that is not inherently flawed and evaluates the *relative* strength of current techniques that are predominantly based on confusion. The model would of course acknowledge that (absolute or unconditional) information theoretic security of the original source code under VBB for general programs is not possible. Such a better model has been elusive to realize, but we address alternate possibilities for measuring obfuscation strength in other work [27, 28].

In this work, we establish the possibility of generalized, perfect obfuscation security in the information theoretic sense for a limited, but useful and relevant, class of programs. More importantly, we lay the foundations for how to embed a key securely in a private-key encryption setting. Such constructions can be used as a basis to securely transform a private-key system into a public-key system—a long standing problem in computer science.

REFERENCES

[1] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang, "On the (im)possibility of obfuscating programs," in *Proc. of CRYPTO '01*, J. Kilian, Ed. Santa Barbara, California: Springer-Verlag, *LNCS 2139*, Aug. 19-23 2001, pp. 1-18

[2] 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.

[3] A. Appel, "Deobfuscation is in NP," unpublished manuscsript, preprint available from http://www.cs. princeton.edu/~appel/papers/deobfus.pdf, 2002.

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

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

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

[7] C. Wang, "A security architecture for survivability mechanisms," PhD thesis, Department of Computer Science, University of Virginia, 2000.

[8] T. Ogiso, Y. Sakabe, M. Soshi, and A. Miyaji, “Software obfuscation on a theoretical basis and its implementation,” *IEICE Trans. Fundamentals,* vol E86-A, no. 1, January 2003.

[9] S. Chow, Y. Gu, H. Johnson and V. A. Zakharov, “An approach to the obfuscation of control-flow of sequential computer programs,” *Proc. of ISC 2001, LNCS 2200*, pp. 144-155, 2001.

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

[11] S. Drape, “Obfuscation of abstract data types,” Doctoral thesis, Department of Computer Science, St. John’s College, University of Oxford, UK.

[12] Y. Yu, J. Leiwo, and B. Premkumar, “Hiding circuit topology from unbounded reverse engineers,” in L. Batten and R. Safavai-Naini (Eds.): *Proc. of ACISP 2006, LNCS 4058*, pp. 171-182, 2006.

[13] Y. Yu, J. Leiwo, and B. Premkumar, “Securely utilizing external computing power,” *Proc. of the IEEE Int’l Conf. on Information Technology: Coding and Computing (ITCC’05)*, 2005.

[14] R. Canetti, “Towards realizing random oracle: Hash functions that hide all partial information,” *CRYPTO’97*, pp. 455-469.

[15] H. Wee, “On obfuscating point functions,” *Proc. of ACM STOC’05*, pp. 523-532, May 22-24, 2005.

[16] W. Thompson, A. Yasinsac, and J. 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] R. Ostrovsky and W. Skeith, “Private searching on streaming data,” CRYPTO ‘2005, 2005.

[18] D. Lie, J. Mitchell, C. A. Thekkath, and M. Horowitz, "Specifying and verifying hardware for tamper-resistant software,” in Proc. of IEEE Symposium on Security and Privacy, Berkeley, CA., 2003.

[19] S. Chow, P. Eisen, H. Johnson, and P. C. van Oorschot, "A white-box DES implementation for DRM applications," *Proc. of the 2nd ACM Workshop on Digital Rights Management (DRM 2002), LNCS 2696*, pp. 1-15, 2003.

[20] T. Sander and C. F. Tschudin, "Protecting mobile agents against malicious hosts," in *Mobile Agent Security, LNCS 1648*, G. Vigna, Ed.: Springer-Verlag, 1998, pp. 44-60.

[21] J. Gregg. Ones and Zeros: Understanding Boolean Algebra, Digital Circuits, and the Logic of Sets. IEEE Press. 1998.

[22] E. Mendelson. Schaum’s Outline Series: Theory and Problems of Boolean Algreba & Switching Circuits. McGraw-Hill Book Company, 1970.

[23] O. Goldreich. Foundations of Cryptography. Cambridge University Press, 2001.

[24] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway, “A concrete security treatment of symmetric encryption: Analysis of the DES modes of operation,” *Proc. of the 38th Symp. on FOCS*, IEEE, 1997.

[25] B. Adida and D. Wikström, “Obfuscated ciphertext mixing,” *IACR Eprint Archive*, no. 394, 2005.

[26] B. Barak, “Non-black-box techniques in cryptography,” PhD Thesis, Weizmann Institute, 2003.

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

[28] J. McDonald and A. Yasinsac, “Program Intent Protection Using Circuit Encryption,” to appear, *Proc. of 8th Int’l Symposium on Systems and Information Security*, Sao Paulo, Brazil, Nov. 8-10, 2006.