A note on the concept of obfuscation

N. P. Varnovsky

Abstract. In this paper we address the issue of defining security of program obfuscation. We argue that requirements to obfuscated programs may be different and are dependent on potential applications. Therefore three distinct models are suggested for studying the issues of obfuscation, namely obfuscation for software protection, total obfuscation and constant hiding. We also introduce a definition of weak obfuscation based on "grey-box" paradigm and show this weak form of obfuscation to be impossible.

1. Introduction

Obfuscation is a relatively new topic in computer science. To the best of our knowledge there exists unique paper of Barak et al. [1] which provides mathematically rigorous treatment of this concept. However, an informal idea of obfuscation emerges naturally from investigation of intelligibility of program codes. It is mentioned (without using the term "obfuscation") already in the seminal paper by Diffie and Hellman [2].

From theoretical point of view the concept of obfuscation is related to the concept of learnability [3]. In the learning theory setting learner has access to some partial information about an unknown particular concept chosen in a predetermined class of concepts and is required to identify it more or less accurately. It is evident that obfuscated program has to be unlearnable in a certain sense, but this condition is by no means sufficient. Any reasonable definition of secure obfuscation should be based on stronger requirements.

It is also obvious that obfuscation is related to cryptography. Obfuscation can be considered as a special case of encryption. One minor difference is that plaintext (original program) needs not be efficiently extractable from cryptogram (obfuscated program). The main difference seems to be decisive: cryptogram itself must be executable code equivalent to original program.

If secure obfuscation were possible it could be used for protecting software as intellectual property, i. e. for hiding principal ideas behind the algorithm as opposed to protecting program code only against unauthorized copying.

In our opinion, however, the main challenge to obfuscation has its origin in philosophy: can one human being produce useful program code that is practically

unintelligible to any other human being?

In section 3 of the present paper we briefly overview main definitions and results of Barak et al. [1]. Reader, familiar with the paper [1] may skip this section.

In section 4 we examine the notion of obfuscation. We argue that requirements to obfuscated programs may be different and are dependent on potential applications. Therefore we suggest two novel definitions of secure obfuscation.

Strong impossibility results [1] leave open the following question: whether there exist weaker forms of universal (i. e. applicable to any program) obfuscators. In section 5 we introduce a definition of secure obfuscator based on "grey-box" paradigm. Namely, an adversary, trying to extract some information from obfuscated program $\mathcal{O}(P)$ is compared with simulating machine having access to an oracle that on input x returns not only output $P(x)$ but also a trace of execution of P on this input. We show this weak form of obfuscation to be impossible as well.

2. Some notation

We use TM as a shorthand for Turing machine. PPT denotes probabilistic polynomial-time Turing machine. For PPT A and any input x, $A(x)$ is a random variable. When we write "for any $A(x)$" we mean a universal quantifier over the support of $A(x)$.

For a pair of TMs A and B, $A \approx B$ denotes their equivalence, i. e. for any input x, $A(x) = B(x)$.

Function $\nu : N \to [0,1]$ is negligible if it decreases faster than any inverse polynomial, i. e. for any $k \in N$ there exists n_0 such that for all $n \geq n_0$, $\nu(n) < 1/n^k$.

3. Definitions and main results of Barak et al [1]

In computer science there is a number of definitions of a program. Two of them can be regarded as commonly accepted ones. The first says that program is a Turing machine (TM), while the second defines program as a Boolean circuit. Accordingly, in Barak et al [1] there are two definitions of secure obfuscation.

Definition 1 [1]. A probabilistic algorithm $\mathcal{O}$ is a TM obfuscator if the following holds:

— For every TM M any output $\mathcal{O}(M)$ of $\mathcal{O}$ on input M describes a TM that computes the same function as M.

— The description length and running time of any TM $\mathcal{O}(M)$ are at most polynomially larger than that of M. I. e., there exists a polynomial p such that for every TM M and any $\mathcal{O}(M)$, $|\mathcal{O}(M)| \leq p(|M|)$ and if M halts in t steps on

some input x then $\mathcal{O}(M)$ halts within $p(t)$ steps on x.

— For any PPT A there is a PPT S and a negligible function ν such that for all TMs M

$$|Pr\{A(\mathcal{O}(M)) = 1\} - Pr\{S^M(1^{|M|}) = 1\}| = \nu(|M|).$$

Definition 2 [1]. A probabilistic algorithm $\mathcal{O}$ is a circuit obfuscator if the following holds:

— For every circuit C any output $\mathcal{O}(C)$ of $\mathcal{O}$ on input C describes a circuit that computes the same function as C.

— There is a polynomial p such that for every circuit C and any $\mathcal{O}(C)$, $|\mathcal{O}(C)| \leq p(|C|)$.

— For any PPT A there is a PPT S and a negligible function ν such that for all circuits C

$$|Pr\{A(\mathcal{O}(C)) = 1\} - Pr\{S^C(1^{|C|}) = 1\}| = \nu(|C|).$$

The main result of Barak et al [1] is negative: secure obfuscation is impossible. To state this precisely one needs the next definition.

Definition 3 [1]. An unobfuscatable function ensemble is an ensemble $\{H_k\}_{k\in N}$ of distributions H_k on finite functions (from, say, $\{0,1\}^{n(k)}$ to $\{0,1\}^{m(k)}$) satisfying:

— there exists polynomial Q such that every function in the support of H_k is computable by a circuit of size at most $Q(k)$. Moreover, there exists a probabilistic polynomial time algorithm that samples uniformly from the distribution on circuits consistent with H_k.

— There exists a function $\pi : \cup_{k\in N}\mathrm{Supp}(H_k) \to \{0,1\}$ such that

1. $\pi(f)$ is hard to compute with black-box access to f: for any PPT S

$$Pr\{S^f(1^k) = \pi(f)\} \leq 1/2 + \nu(k),$$

where ν is a negligible function.

2. $\pi(f)$ is easy to compute with access to any circuit that computes f: there exists a PPT A such that for any $f \in \cup_{k\in N}\mathrm{Supp}(H_k)$ and for any circuit C that computes f, $A(C) = \pi(f)$.

Theorem 1 [1]. If one-way functions exist, then there exists an unobfuscatable function ensemble

This result is essentially a corollary to the next theorem.

Theorem 2 [1]. If one-way functions exist then circuit obfuscators (in the sense of definition 2) do not exist.

In the case of Turing machines an analogous result is proven unconditionally

Theorem 3 [1]. Turing machine obfuscators (in the sense of definition 1) do not exist.

4. Discussion

4.1. Motivating examples

Any mathematically correct definition is useless unless one keeps in mind some potential applications of the notion being defined. In the paper [1] the authors list several possible applications of obfuscators. The first one is in software protection. Suppose one party invents a fast algorithm for factoring integers and wishes to sell to another party a program for breaking the RSA cryptosystem. The goal is to have this program transformed in such a way that it will be hard to derive factorization algorithm from the text of transformed program.

Another particularly challenging potential application is obfuscation of encryption programs of private-key cryptosystems. This would allow to transform private-key cryptosystems, e. g. DES, into public-key ones.

However the "virtual black-box" paradigm underlying the definitions 1 and 2 does not seem to be well-suited for these intended applications. Indeed, it is hardly possible to find a client who will buy a black-box as a piece of software. Instead, any program product on the market must have a user guide specifying its functionality.

4.2. Classification

We suggest the following three models for studying the issues of obfuscation.

Obfuscation for software protection. In this setting an adversary knows the function computed by the program in question. The most straightforward way to formalize this is to fix some program P_0 equivalent to the original program P and give P_0 to the adversary as an additional input.

E. g., if P is a program totally breaking the RSA cryptosystem, i. e. finding its private key given a public one, then P_0 might be an easy-to-understand program which accomplishes the same task by exhaustive search.

The simulating machine S guaranteed by definitions 1 and 2 also has access to the text of the program P_0. The modified definition 1 is as follows.

Definition 4. A probabilistic algorithm $\mathcal{O}$ is a TM obfuscator if the following holds:

— For every TM M any output $\mathcal{O}(M)$ of $\mathcal{O}$ on input M describes a TM that computes the same function as M.

— The description length and running time of any TM $\mathcal{O}(M)$ are at most polynomially larger than that of M. I. e., there exists a polynomial p such that for every TM M and any $\mathcal{O}(M)$, $|\mathcal{O}(M)| \leq p(|M|)$ and if M halts in t steps on some input x then $\mathcal{O}(M)$ halts within $p(t)$ steps on x.

— For any PPT A there is a PPT S and a negligible function ν such that for all pairs of TMs $(M, \widetilde{M})$, $M \approx \widetilde{M}$,

$$|Pr\{A(\mathcal{O}(M), \widetilde{M}) = 1\} - Pr\{S^M(1^{|M|}, \widetilde{M}) = 1\}| = \nu(|M|).$$

The machines A and S are polynomial in the lengths of their first inputs, $\mathcal{O}(M)$ and $1^{|M|}$ respectively.

Note that in the case when algorithm $\widetilde{M}$ is efficient simulating machine S needs no access to the oracle M.

Remark. There exist program properties (predicates) π invariant under equivalent program transformations. I. e., for any program P' functionally equivalent to P, $\pi(P') = \pi(P)$. Under our definition such properties are only trivially obfuscatable. Note, however, that from the practical point of view obfuscating invariant properties is useless.

Impossibility results of Barak et al. [1] do not apply in this setting. Moreover, our definition has the following peculiar feature. If one manages to extend the proof technique of the paper [1] to show that secure obfuscation in the sense of definition 4 is impossible, then this will immediately imply that some weak form of obfuscation does exist! Indeed, counterexample of Barak et al. is based on a specific invariant property of a particular family of programs. Suppose there exists an infinite sequence of pairs $(M, \widetilde{M})$ of TMs such that some invariant property π is learnable efficiently given the text of the program $\mathcal{O}(M)$ but is unlearnable when we have the text of the program $\widetilde{M}$ and black-box access to M. But this means that $\widetilde{M}$ is a (provably) secure obfuscation of $\mathcal{O}(M)$ with respect to the property π.

We pose the question of existence of secure obfuscation in the sense of definition 4 as the main research problem in the theory of obfuscation.

Total obfuscation. This is just the concept of Barak et al [1]. An example of potential application of this brand of obfuscation is given in [1], namely removing random oracles from cryptographic schemes. Note, however, that in this case obfuscation is not completely total. Adversary knows a priori that random oracle has to compute some function from, say, $\{0,1\}^{2n}$ to $\{0,1\}^n$ which looks random. Therefore the method of constructing counterexamples to secure obfuscation suggested by Barak et al [1] does not work in the case being considered. This method is based on asking the adversary to guess whether the function computed by obfuscated program is constant. For random oracles the

answer is always "no". This means that at least when application in removing random oracles is concerned, security requirements to obfuscation are weaker and impossibility results of [1] do not rule out such an application. However, the most obvious approach to this problem, namely one based on obfuscating pseudorandom function families does not work [1].

As another possible application of total obfuscation consider the following scenario. An abonent of computer network runs on her PC certain programs and does not want to let hackers to know what kind of data processing these programs do.

Constant hiding. In this setting everything to be hidden from adversary is certain constant c_0 used by a program P. This constant is assumed to be chosen randomly in sufficiently large set C. One can safely assume that the original program P is completely known to the adversary except for the constant c_0.

It is easy to see that constant hiding is essentially a particular case of obfuscation for software protection. Definition 4 can be easily adopted to the present case by setting $\widetilde{M}$ to be the same TM as M except for the constant c_0 replaced by a constant c chosen randomly and independently in C.

Details follow. Let P be a program which uses a constant c. For simplicity we assume that this constant is chosen uniformly at random in the set $\{0,1\}^n$. For any given constant value $c_0 \in \{0,1\}^n$ we denote by $P(c_0)$ the corresponding instantiation of the program P. We assume the length of the program P to be polynomial in n.

Definition 5. A probabilistic algorithm $\mathcal{O}$ is a TM obfuscator if the following holds:

— For every TM M any output $\mathcal{O}(M)$ of $\mathcal{O}$ on input M describes a TM that computes the same function as M.

— The description length and running time of any TM $\mathcal{O}(M)$ are at most polynomially larger than that of M. I. e., there exists a polynomial p such that for every TM M and any $\mathcal{O}(M)$, $|\mathcal{O}(M)| \leq p(|M|)$ and if M halts in t steps on some input x then $\mathcal{O}(M)$ halts within $p(t)$ steps on x.

— For any PPT A there is a PPT S and a negligible function ν such that for any TM M and any constant value $c_0 \in \{0,1\}^n$

$$|Pr\{A[\mathcal{O}(M)(c_0), M(c)] = 1\} - Pr\{S^{M(c_0)}[1^{|M(c_0)|}, M(c)] = 1\}| = \nu(|M(c_0)|),$$

where $c \in_R \{0,1\}^n$.

Note that one can define a weak form of constant hiding in which adversary A and simulating machine S instead of outputting a single bit have to guess the constant c_0.

Nontrivial constant hiding does exist (at least in the weak form) under cryptographic assumptions for certain restricted classes of programs. In fact, any public-key cryptosystem gives such an example since its encryption program can be regarded as hiding a particular constant, namely a private key.

4.3. Weak obfuscation

In this section we examine one specific approach to program obfuscation. The main idea is to divide the search for secure obfuscation into two stages. Since an adversary having access to the obfuscated program can always obtain not only input-output pairs but also the corresponding traces one could first pose a problem of whether it is possible to transform a program in such a way that these traces are essentially the only useful information available to adversary. If this were possible then the next problem is how one can guarantee that the traces themselves give away no useful information.

We confine ourselves here with the model for total obfuscation.

Definition of this new brand of obfuscation which from now on we will call weak obfuscation differs from definition 1 in two aspects. The first is specification of the oracle ("black-box") used by simulating machine S. When the machine S issues query x to the oracle it gets as a response not only the output word y but also the trace of execution of M on input x. Note that in our setting it makes difference which particular program equivalent to M is used by the oracle. We insist that this is just the original program M. This means that we do not require obfuscator to hide any property that is learnable efficiently from traces of execution of TM M. Notice also that any obfuscator can be deemed secure only if it does not reveal any properties that remain hidden given access to the traces of original program.

The second is definition of program equivalence. We consider programs with "memory", i. e. their output depends not only on the current input but also on the preceding ones.

To distinguish the oracle used in definition 1 from the oracle provided to simulating machine S in this new setting we denote the latter by $Tr(M)$. On input x this oracle outputs a pair $(y, tr_M(x))$, where y is the output of TM M on input x and $tr_M(x)$ is the trace of execution of M on this input.

The string tr_M is defined as concatenation of all successive instructions executed by M when running on input x.

Definition 6. A probabilistic algorithm $\mathcal{O}$ is a weak TM obfuscator if the following holds:

— For every TM M any output $\mathcal{O}(M)$ of $\mathcal{O}$ on input M describes a TM that is equivalent to M in the following sense. For any sequence of inputs $x_1, \ldots, x_u$

the corresponding sequences of outputs of TMs $\mathcal{O}(M)$ and M coincide, i. e., $\mathcal{O}(M)(x_i) = M(x_i)$ for any $i = 1, \ldots u$.

— The description length and running time of any TM $\mathcal{O}(M)$ are at most polynomially larger than that of M. I. e., there exists a polynomial p such that for every TM M and any $\mathcal{O}(M)$, $|\mathcal{O}(M)| \leq p(|M|)$ and if M halts in t steps on some input x then $\mathcal{O}(M)$ halts within $p(t)$ steps on x.

— For any PPT A there is a PPT S and a negligible function ν such that for all TMs M

$$|Pr\{A(\mathcal{O}(M)) = 1\} - Pr\{S^{Tr(M)}(1^{|M|}) = 1\}| = \nu(|M|).$$

Theorem 4. If one-way functions exist then weak obfuscation is impossible.

Proof. The intuition behind proof is quite simple. The proof of theorem 3 is by counterexample. Barak et al. [1] constructed an infinite family of TMs such that certain predicate $\pi(M)$ defined on this family is unlearnable with oracle access to the function computed by M but can be decided easily given a text of any program equivalent to M. It suffices to modify this construction in such a way that the following two requirements hold simultaneously. First, learnability of predicate $\pi(M)$ given a text of any program equivalent to M should be preserved. Second, traces of execution of M must provide adversary with no additional useful information as compared to the output-only oracle treated by definition 1. An adversary having access to traces of execution of M is apparently more powerful then one bounded to see input-output pairs only. To defeat this powerful adversary we make use of cryptographic tools. Namely, instead of comparing input x to the fixed string α as in Barak et al [1] we test first whether $f(x) = f(\alpha)$, where f is a one-way function. Only if this test passes we check whether $x = \alpha$.

Typical trace $tr_M(x)$ consists of instructions used to compute $f(x)$ and to check whether $f(x) = f(\alpha)$. Most of the time this check fails. Moreover, if the equality $f(x) = f(\alpha)$ holds with nonnegligible probability then this contradicts the fact that f is one-way function.

Note that one cannot replace one-way function f in this construction by e. g. encryption function of private-key cryptosystem since encryption algorithm uses private key and traces of its execution become available to adversary.

We stress that the above discussion has to be considered as motivating. The actual construction is somewhat more complicated.

Now we turn to formal proof.

The counterexample of Barak et al. [1] uses two families of TMs. For any pair of strings $\alpha, \beta \in \{0,1\}^n$ Turing machine $C'_{\alpha,\beta}(x)$ outputs β if $x = \alpha$ and 0^n otherwise. For the same parameters α, β Turing machine $D'_{\alpha,\beta}(C)$ outputs 1 if

$C(\alpha) = \beta$ and 0 otherwise.

Let $\{f : \{0,1\}^n \to \{0,1\}^n\}_n$ be a one-way function. Our construction depends also on integer parameter $t \geq 2$ which can be e. g. a constant or a value of arbitrary but fixed polynomial (in n).

First we choose $2t$ strings $\alpha_1, \ldots, \alpha_t, \beta_1, \ldots, \beta_t \in \{0,1\}^n$ uniformly at random. Denote this $2t$-tuple by γ. Now define TM $C_\gamma(x)$ as follows. On input x this TM computes $f(x)$ and tests the result against precomputed values $f(\alpha_i)$, $i = 1, \ldots, t$. If $f(x) \neq f(\alpha_i)$ for all $i = 1, \ldots, t$ then C_γ outputs 0^n and halts. In the case when $f(x) = f(\alpha_i)$ for some i, C_γ checks whether $x = \alpha_i$ and if so outputs β_i, otherwise it outputs 0^n and in either case halts.

Next we define TM D_γ. It stores two arrays of strings $\alpha_1, \ldots, \alpha_t$ and $\beta_1, \ldots, \beta_t$. Let i be a pointer to both this arrays which is initially set to 0.

On input a TM C, D_γ runs as follows.

1. Check whether $i = t - 1$. If so, output 0 and halt.

2. Advance the pointer $i = i + 1$ and then feed C with input α_i.

3. If $C(\alpha_i) = \beta_i$ then check the current value of the pointer. If $i = t$ then output 1 and halt, else goto 2.

4. If $C(\alpha_i) \neq \beta_i$ then output 0 and halt.

Note that simulating machine having access to C_γ and D_γ as oracles may query D_γ with arbitrary TM C and extract α_1 from the trace, then query C_γ with α_1 and obtain β_1 (even if we have a modified TM D_γ that hides the values β_i) and so on. Therefore simulating machine is granted only $t - 1$ queries to D_γ (in fact further queries are allowed but result invariably in zero responses and thus can be ignored). The unobfuscatable property is the existence of t distinct strings $\alpha_1, \ldots, \alpha_t \in \{0,1\}^n$ such that output of a given TM on each of them is nonzero.

Pair of TMs (C_γ, D_γ) replaces TMs $(C'_{\alpha,\beta}, D'_{\alpha,\beta})$ used in the proof of theorem 3. Analysis of this proof shows that to adopt it to our case one needs only to prove the next claim.

Claim. For any PPT S

$$|Pr\{S^{C_\gamma, D_\gamma}(1^n) = 1\} - Pr\{S^{Z_\gamma, D_\gamma}(1^n) = 1\}|$$

is negligible.

TM Z_γ differs from C_γ only in that on input α_t it outputs 0^n.

The probabilities are taken over uniform choice of $\alpha_1, \ldots, \alpha_t$, $\beta_1, \ldots, \beta_t$ in $\{0,1\}^n$ and coin tosses of S.

Proof of claim (*sketch*). It suffices to show that any PPT has only negligible

probability to get nonzero response to any of its oracle queries, no matter which TM, C_γ or Z_γ, is used as the first oracle.

For simplicity we assume $t = 2$ for the rest of proof. In this case machine S can issue unique query to the second oracle. Let $a_1, \ldots, a_s \in \{0,1\}^n$ be all the queries of S to the first oracle prior to issuing its only query to the second one (in fact s is a random variable). Suppose that nonzero string appears with nonnegligible probability among responses to these s queries. We construct a PPT T inverting the function f.

Let $z \in_R \{0,1\}^n$ and $y = f(z)$ be input to T. Machine T flips a coin and decides which of values, $f(\alpha_1)$ or $f(\alpha_2)$ will be set to y. Without loss of generality let it be α_1. Then T chooses $\alpha_2, \beta_1, \beta_2 \in \{0,1\}^n$ uniformly at random and calls S as a subroutine. All queries to the first oracle are intercepted by T. For a query $x \in \{0,1\}^n$, T computes $f(x)$ and checks whether $f(x) = y$ or $f(x) = f(\alpha_2)$. If neither of these equalities holds, T outputs 0^n. In the case $f(x) = y$, T outputs β_1, and in the case $f(x) = f(\alpha_2)$ it proceeds in the same way as TM C_γ does.

The crucial observation is that the probability of seeing a nonzero response from this simulated oracle T is at least as high as in the case of real oracle.

Therefore for some $j \in 1, \ldots s$ the response of T is nonzero with nonnegligible probability. For this j, α_j is in the preimage of y with probability $1/2$. Thus T inverts f with nonnegligible probability which contradicts the fact that f is one-way function.

If nonzero string appears among responses to the first s queries with negligible probability then the probability of nonzero response to a unique query of S to the second oracle is negligible as well.

For the remaining oracle queries (after the query to the second oracle) the same argument as above shows that nonnegligible probability of success would imply existence of efficient algorithm for inverting the function f. This contradiction proves the claim.

References

[1] Barak B., Goldreich O., Impagliazzo R., Rudich S., Sahai A., Vadhan S., Ke Yang. On the (im)possibility of obfuscating programs. J. Kilian ed. Advances in Cryptology — Crypto'01. Lecture Notes in Computer Science. Springer–Verlag, v. 2139, 2001, 1–18

[2] Diffie W., Hellman M. New directions in cryptography. IEEE Transactions om Information Theory, IT-22(6), 1976, 644–654

[3] Valiant L. A theory of learnable. Comm. ACM, 1984, v. 27, N 11, 1134–1142