On the Concept of Software Obfuscation in Computer Security*

Nikolay Kuzurin[1], Alexander Shokurov[1],
Nikolay Varnovsky[2], and Vladimir Zakharov[2]

[1] Institute for System Programming, Moscow, Russia
[2] Lomonosov Moscow State University, Russia

Abstract. Program obfuscation is a semantic-preserving transformation aimed at bringing a program into such a form, which impedes the understanding of its algorithm and data structures or prevents extracting of some valuable information from the text of a program. Since obfuscation could find wide use in computer security, information hiding and cryptography, security requirements to program obfuscators became a major focus of interests for pioneers of theory of software obfuscation. In this paper we also address the issue of defining security of program obfuscation. We argue that requirements to obfuscation may be different and dependent on potential applications. Therefore, it makes sense to deal with a broad spectrum of security definitions for program obfuscation. In this paper we analyze five models for studying various aspects of obfuscation: "black box" model of total obfuscation, "grey box" model of total obfuscation, obfuscation for software protection, constant hiding, and predicate obfuscation. For each of these models we consider the applications where the model may be valid, positive and negative results on the existence of secure obfuscation in the framework of the model, and relationships with other models of program obfuscation.

Keywords: Program obfuscation, security, Turing machine, encryption.

1 Introduction

To obfuscate a program means to bring it into such a form which hampers as much as possible the extraction of some valuable information concerning algorithms, data structures, secrete keys, etc. from the text of a program. Obfuscation can be viewed 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: a cryptogram itself must be an executable code equivalent to the original program. The latter is the reason why obfuscation attracts considerable interest of many researchers in cryptography and computer security.

Obfuscation is a relatively new topic in computer science. It was first mentioned (without using the term "obfuscation") in the seminal paper by Diffie and

* This work is supported by RFBR grant 06-01-00584.

J. Garay et al. (Eds.): ISC 2007, LNCS 4779, pp. 281–298, 2007.

Hellman [15]. When introducing the concept of public-key cryptosystem, they noticed that, given any means for obscuring data structures in a private-key encryption scheme, one could convert this algorithm into a public-key encryption scheme. Software engineering community became acquainted with obfuscation through the paper by Collberg et al. [10] where it was presented as an effective means for protecting software intellectual property against reverse engineering. These papers marked the beginning of two lines of research in program obfuscation, namely, obfuscation for the purposes of cryptography and obfuscation for software engineering and computer security.

Program obfuscators could enjoy wide application in cryptography; a secure program obfuscation would allow one to convert private-key encryption schemes into public-key ones, to design homomorphic public-key cryptosystems, to remove random oracles from cryptographic protocols, etc. But in all these cases program obfuscation should comply with some security requirements used in cryptography. Barak et al. initiated in [3] a theoretical investigation of obfuscation. They introduced the concept of "virtual black box" security: an obfuscation $\mathcal{O}$ is secure iff anything one can efficiently compute given an obfuscated program $\mathcal{O}(M)$ one could also efficiently compute given only oracle access to the original program M. They also proved that some families of functions $\mathcal{F}$ are inherently unobfuscatable in the following sense: there is a property P such that the value $P(f)$ can be efficiently computed, given any program that computes a function $f \in \mathcal{F}$, but no efficient algorithm can compute $P(f)$ when it is given only oracle access to f. This negative result was strengthened in [18,22,32] for some other variants of "virtual black box" security. Positive results were obtained also: a secure obfuscation is possible (under some rather strong cryptographic assumptions) for programs that compute point functions evaluating to zero almost everywhere (see [22,24,29,32]).

There are also a lot of problems in software engineering and computer security where program obfuscation would be very much helpful. Over the years it was found that obfuscation as a low cost means may be used for preventing reverse engineering [10,11], defending against computer viruses [9], protecting software watermarks and fingerprints [2,12], providing security of mobile agents [14,21], maintaining private searching on streaming data [26]. Unfortunately, obfuscation is also well-suited for some malevolent purposes, e.g. for obscuring malwares [7,27] and some types artificial vulnerabilities in protection systems [4]. These and some other applications have given impetus to the development of numerous obfuscation techniques (see, e.g. [1,8,11,13,23,25,31]). For the most part these approaches are no more than heuristics (sometime rather sophisticated) intended for distracting program analysis algorithms and impeding thus the understanding and reverse engineering of obfuscated programs. The principle drawback of all known obfuscation techniques is that they do not refer to any formal definition of obfuscation security and do not have a firm ground for estimating to what extent such methods serve the purpose.

This brief overview of the current state of art in software obfuscation allows some conclusions.

- The applications of program obfuscation are notably diversified, and the objectives to be pursued in these applications are also different. When obfuscation is intended to turn a private-key cryptosystem into a public-key one it is expected that such transformation hides a secret key but not necessarily an implementation of the encryption algorithm. On the contrary, when obfuscation is designed for software protection against reverse engineering it is often bound to hide only the implementation of the most valuable algorithms but not a processed data. This means that the term "obfuscation problem" is but a generic name for the set of particular problems, each being determined by the specific security requirement.
- There is considerable gap between negative and positive results. Actually, the authors of [3] proved the impossibility of a universal obfuscating program by presenting a family of functions such that every program which computes any of these functions readily betrays a secret when being applied to itself (cf. self-applicability problem for Turing machines). The positive results were obtained in [5,24,32] only for a very small class of functions under strong cryptographic assumptions. Although these results were extended in [16] to the more interesting case of proximity queries, nothing is known about the (im)possibility of effective obfuscation for common cryptographic algorithms, or any meaningful class of programs (say, finite automata) under standard cryptographic assumptions against reasonably weak security requirements.
- There is considerable gap between theory and practice of program obfuscation. There are many papers which suggest various approaches to designing obfuscating transformations of programs; some of them are implemented in academic or commercial toolkits for program obfuscation. But the influence of fundamental results from [3,18,24,32] on this branch of software engineering is minor: security requirements studied in the context of cryptographic applications are either too strong or inadequate to many software protection problems emerged in practice.

We believe that further progress in the study of program obfuscation may ensue primarily from the development of a solid framework which makes it possible to set up security definitions for program obfuscation in the context of various applications. This would help us to reveal the most important common properties required of any type of program obfuscation. Having access to a variety of security definitions one may also understand better what security requirements one or other obfuscating transformation complies with and thus estimate both the potency and the drawbacks of a particular program obfuscation technique. Intuition suggests that some weakly secure program obfuscation is possible: everybody dealing with program understanding knows that in many cases even small programs require considerable efforts to reveal their meaning. A variety of new formal security requirements for program obfuscation would offer a clearer view of this observation. Finally, when new formalizations of security requirements for software obfuscation are brought into service, this opens new

channels for adopting formal methods from computer science and cryptography to the problems of software protection.

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. To this end we consider five models for studying various aspects of obfuscation. We take a simulation paradigm and "black box" model of total obfuscation defined in [3] as a basis. Strong impossibility results obtained in [3] leave open the following question: whether there exist some weaker meaningful forms of security requirements that admits the existence of provably secure obfuscator. In attempt to find an answer to this question we try to adopt this model to various applications where program obfuscation may be used. This bring us to four new formal models of program obfuscation, namely, "grey box" model of total obfuscation, obfuscation for software protection, constant hiding, and predicate obfuscation. We study the basic properties of these models, positive and negative results on the existence of secure obfuscation in the framework of the models, and relationships with other models of program obfuscation. The key notions of program obfuscation studied in this paper have been introduced in preliminary form in Technical Report [30]. We believe that these new models of program obfuscation would provide a good framework for analyzing the potency of obfuscating transformations developed in software engineering for the purpose of program protection.

2 Notation

In complexity theory 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, whereas the second defines program as a Boolean circuit. Accordingly, a secure obfuscation can be defined either in terms of Turing machines, or in terms of circuits (see [3]). For the sake of uniformity we restrict our consideration to Turing machines only.

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

For a TM we will write $|A|$ to denote the size of A. For a pair of TMs A and B, $A \approx B$ denotes their equivalence, i.e. $A(x) = B(x)$ holds for any input 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 $\nu(n) < 1/n^k$ holds for all $n \geq n_0$. We will sometimes write $neg(\cdot)$ and $poly(\cdot)$ to denote unspecified negligible function and positive polynomial, respectively.

We write $x \in_R D$ to indicate that x is chosen uniformly at random in the set D.

3 Total Obfuscation

3.1 "Black Box" Model

This concept of obfuscation was introduced and investigated by Barak et al. in the paper [3]. Ideally, a totally obfuscated program should be a "virtual black box", in the sense that anything one can deduce from its text could be also computed from its input-output behavior. This notion admits various formalizations according to what an adversary party regards as a success. The weakest form is that when an adversary tries to compute any predicate. Below we recall briefly basic definitions of "virtual black box" obfuscation security.

Definition 1 ([3]). *A probabilistic algorithm $\mathcal{O}$ is a* "virtual black box" obfuscator *if it complies with the following requirements:*

1. (functionality) *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, i.e. $M \approx \mathcal{O}(M)$.*
2. (polynomial slow-down) *The description length (size) 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.*
3. (security) *For any PPT A (adversary) there is a PPT S (simulator) and a negligible function ν such that*

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

holds for all TMs M.

The main result of Barak et al. [3] is negative: secure obfuscation is impossible.

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

The key idea of the proof is as follows. For strings $\alpha, \beta \in \{0,1\}^n$ define the TMs $C_{\alpha,\beta}, D_{\alpha,\beta}, Z_n$. On every input x a TM $C_{\alpha,\beta}$ always terminates within $10|x|$ steps; it outputs β if $x = \alpha$ and 0^n otherwise. A TM $D_{\alpha,\beta}$ considers an input x as the code of some TM M, executes $M(\alpha)$ for $poly(n)$ steps, and outputs 1 if $M(\alpha) = \beta$ and 0 otherwise. A TM Z_n always outputs 0^n. Let $M_0 \# M_1$ denotes a TM which takes the pairs (δ, x), $\delta \in \{0,1\}$ as inputs and operates on x as M_0 if $\delta = 0$ and as M_1 if $\delta = 1$. Clearly, it is possible to distinguish efficiently $O(C_{\alpha,\beta} \# D_{\alpha,\beta})$ from $O(Z_n \# D_{\alpha,\beta})$ with probability 1. To this end, given a TM M on input, an adversary A first decomposes M into $M_0 \# M_1$ and then outputs $M_1(M_0)$. On the other hand, for every PPT S the outputs $S^{C_{\alpha,\beta} \# D_{\alpha,\beta}}(x)$ and $S^{Z_n \# D_{\alpha,\beta}}(x)$ will be the same with a high enough probability.

Remark 1. Programs that are obfuscatable in the sense of Definition 1 always exist. Clearly, if on every input x a TM M attaches its program to the output, i.e. $M(x) = (y, \text{"}M\text{"})$, then (1) holds trivially for any semantic-preserving

transformation $\mathcal{O}$. Note, that in this case we deal with a *learnable* program (in the sense of [28]). The question is whether there exists a class of unlearnable programs which admits a total obfuscation.

Even before "virtual black box" concept of obfuscation emerged in [3], Canetti [5] and Canetti et al. [6] essentially obfuscated point functions under various computational assumptions. A point funtion $I_\alpha(x)$ is evaluted to 1 if $x = \alpha$ and to 0 otherwise. In [24] it was shown that a total obfuscation is possible in the random oracle model for programs computing point and multi-point functions. The problem of total obfuscation of point functions was also studied by Wee in [32]. In this paper it was shown that an efficient obfuscation for the family of point functions with multi-bit output is possible in the standard model of computation provided that some very strong one-way permutations exist. Using the construction from [6] Dodis and Smith demonstrated in [16] how to obfuscate proximity queries where points are selected at random from a distribution with min-entropy.

For the most applications of program obfuscation in cryptography (including the converting of private-key encryption algorithms into public-key cryptosystems, the designing of homomorphic public-key cryptosystems, etc.) the "virtual black box" security paradigm is crucial. Another particularly challenging potential application is the removing of 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. [3] 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 [3] 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 yet (see [3]).

3.2 "Grey Box" Model

Since an adversary having access to an 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 an adversary. If this were possible then the next problem is how one can guarantee that the traces themselves give away no useful information.

To this end we modify the "virtual black box" obfuscation model. Definition of this new brand of obfuscation which from now on we will call "virtual grey box" obfuscation differs from Definition 1 in two aspects. The first is the 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 aspect concerns the programs to be obfuscated. We consider so called reactive programs. Unlike ordinary programs intended for computing some input-output relation, reactive programs are intended to interact with the environment. This interaction displays itself in the responses a reactive program computes on the requests from the environment. Thus, a reactive program computes a function which maps the infinite sequences of inputs $x_1, x_2, \ldots, x_n, \ldots$ (sequences of requests) into the infinite sequences of outputs $y_1, y_2, \ldots, y_n, \ldots$ (sequences of replies) so that every output y_n depends on the inputs $x_1, x_2, \ldots, x_n$ only. Examples of reactive programs may be found in network protocols (including cryptographic ones), embedded systems, operating systems, etc. A reactive program may be formalized as a conventional TM which operates on read-only input tape and write-only output tape (and using some auxiliary tapes) in such a way that it does not proceed to read next input word x_{n+1} until it completes writing the output y_n. We will denote these machines as RTM to distinguish them from the ordinary TM. It should be noticed that every TM may be viewed as an RTM which resets its control into some predefined initial state s_0 and erases its auxiliary tapes every time before reading the next input x_n.

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 RTM M on input x (note that y depends not only on x but also on all previous inputs that have been used as requests to the oracle) 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 2. *A probabilistic algorithm $\mathcal{O}$ is a* "virtual grey box" RTM obfuscator *if it complies with the following requirements:*

1. *For every RTM M any output $\mathcal{O}(M)$ of $\mathcal{O}$ on input M describes a RTM that is equivalent to M in the following sense: for any sequence of inputs $x_1, \ldots, x_n, \ldots$ the corresponding sequences of outputs of RTMs $\mathcal{O}(M)$ and M coincide.*
2. *The description length and running time (on every finite sequence of inputs $x_1, x_2, \ldots, x_n$) of any RTM $\mathcal{O}(M)$ are at most polynomially larger than that of M.*
3. *For any PPT A (adversary) there is a PPT S (simulator) and a negligible function ν such that*

$$|Pr\{A(\mathcal{O}(M)) = 1\} - Pr\{S^{Tr(M)}(1^{|M|}) = 1\}| = \nu(|M|). \tag{2}$$

holds for all RTMs M.

Theorem 2. *If one-way functions exist then "virtual grey box" RTM obfuscators do not exist.*

The intuition behind proof is quite simple. Barak et al. [3] proved Theorem 1 by constructing 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 a simulator with no additional useful information as compared to the output-only oracle treated by Definition 1. A simulator S 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 simulator we make use of cryptographic tools. Namely, instead of comparing input x to the fixed string α as in Barak et al [3] we test first whether $f(x) = f(\alpha)$, where f is a one-way function. Only if this test passes we check whether $x = \alpha$.

A 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 a 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 the formal proof.

Proof. The counterexample of Barak et al. from [3] involves 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 RTM $C_\gamma(x)$ as follows. On input x this RTM 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 RTM 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 RTM 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 RTM 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 RTM 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 RTM on each of them is nonzero.

Pair of RTMs (C_γ, D_γ) replaces TMs $(C'_{\alpha,\beta}, D'_{\alpha,\beta})$ used in the proof of Theorem 1 presented in [3]. Analysis of this proof shows that to adopt it to our case one needs only to prove the next claim.

Claim: *Suppose that RTM Z_γ differs from C_γ only in that on input α_t it outputs 0^n. Then for any PPT S the amount*

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

is negligible. 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. (*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 RTM, C_γ or Z_γ, is used as the first oracle.

For the sake of 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 RTM 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. □

As it may be seen from the proof, we lean to a large measure upon the ability of an RTM to keep in memory a "history" of its computation on previous inputs. This makes such programs non-resettable. It is unclear yet, whether this theorem can be extended to ordinary TMs. This remains as an open question.

4 Obfuscation for Software Protection

As the most apparent application of obfuscation for software protection consider the following scenario. 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. However, the "black box" obfuscation model underlying Definition 1 does not seem to be well-suited for this application. 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. In this setting an adversary knows the function computed by the program in question. The aim of obfuscation in this case is not to hide any property of the program which refers to its functionality (we may assume that such properties are known to an adversary party in advance from, say, users manual of the program), but to make unintelligible the implementation of these functional properties in a particular program. We think that this view of obfuscation is more adequate for the needs of software engineering community than total obfuscation. The most straightforward way to formalize this concept is to fix some program M_0 equivalent to the original program M and give M_0 to the adversary as an additional input. For example, if M is a program totally breaking the RSA cryptosystem, i.e. finding its private key given a public one, then M_0 might be an easy-to-understand program which accomplishes the same task by exhaustive search.

The simulating machine S guaranteed by Definition 1 also has access to the text of the program M_0. The modified Definition 3 is as follows.

Definition 3. *A probabilistic algorithm $\mathcal{O}$ is a* software protecting obfuscator *if it complies with* functionality *and* polynomial slow-down *requirements from Definition 1, and with the following* security *requirement:*

3) For any PPT A there is a PPT S such that

$$|Pr\{A(\mathcal{O}(M), \widetilde{M}) = 1\} - Pr\{S^M(1^{|M|}, \widetilde{M}) = 1\}| = neg(|M|). \qquad (3)$$

holds for every pair of TMs $(M, \widetilde{M})$, where $M \approx \widetilde{M}$, and $|\widetilde{M}| = poly(|M|)$.

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 2. Some program properties (predicates) P are invariant under equivalent program transformations, i.e. $P(M) = P(M')$ holds for any program M' functionally equivalent to M. Clearly, such properties are only trivially obfuscatable. Note, however, that in the context of software protection it is useless to obfuscate invariant properties as soon as functionality of a program is known.

Security requirement in Definition 3 is much different from that in total obfuscation. The negative results of [3] do not rule out the possibility of "black box" secure obfuscation for many important general classes of machines. Consider as an example the obfuscation problem for deterministic finite state automata (DFA). Secure total obfuscation of DFAs still remains one of the most challenging problem in the theory of program obfuscation. But if one manages to estimate the obfuscation security following Definition 3 then a surprisingly simple obfuscation of DFAs can be obtained.

Theorem 3. *There exists a secure software protecting obfuscator for the family of DFAs.*

Proof. A DFA is but a one-way TM. We may regard any efficient DFA minimizing algorithm $\mathcal{O}$ as a TM obfuscator. Indeed, it is well known that for every DFA M there exists the unique minimal DFA M_0 such that $M \approx M_0$. Therefore, a simulator S, given a TM $\widetilde{M}$ only, can readily compute $\mathcal{O}(M)$ by applying any efficient minimizing algorithm to $\widetilde{M}$ and then mimic an adversary A on the pair $(M, \widetilde{M})$ to satisfy (3). □

The possibility of such effect has been noted in [3] and also studied in more details in [19]. It appears in software engineering as well: if a program compilation includes an advanced optimization (minimization) then executables become far less intelligible by means of reverse engineering. As it can be seen from Definition 3, software protecting obfuscators are intended to remove all individual specific features from a program to be protected. Hence, to obfuscate a program it is sufficient to reduce it to some normal form. If a family of programs $\mathcal{H}$ admits strong and effective normalization, i. e. there exists an algorithm which reduces any two equivalent programs to the same normal form, then Theorem 3 can be extended to this family: any efficient normalization algorithm becomes an obfuscator $\mathcal{O}$ for programs from $\mathcal{H}$. It is worth noting that the authors of [31] implemented a sort of normalization, namely, flattening of program control flow, as the basic obfuscating transformation in their software protection toolkit.

Unfortunately, efficient normalization is possible only for a few natural classes of programs. Therefore, we qualify TM obfuscation through normalization as trivial and pose the following question:

Open problem. Is there any class of programs which admits non-trivial secure obfuscation in the sense of Definition 3?

Another distinctive feature of TM obfuscation is that impossibility results of Barak et al. [3] do not apply in this setting. Moreover, if one attempts to extend the proof technique of the paper [3] to show that secure obfuscation in the sense of Definition 3 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.

It makes sense to juxtapose the notion of TM obfuscation (Definition 3) with the notion of obfuscation w.r.t. auxiliary inputs introduced in the paper [18]. Goldwasser and Kalai modified the "virtual black box" property by admitting both an adversary A and a simulator S an access to an auxiliary input z which may be dependent as well as independent from a program to be obfuscated. When being adapted to TM notation it is as follows.

Definition 4 ([18]). *A probabilistic algorithm $\mathcal{O}$ is an* obfuscator w.r.t. dependent auxiliary input *for the family of TM $\mathcal{F}$ if it complies with* functionality *and* polynomial slow-down *requirements from Definition 1, and with the following* security *requirement:*

3) For any PPT A there is a PPT S such that

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

holds for every TM $M \in \mathcal{F}$, and every auxiliary input z of size poly$(|M|)$ (z may depend on M).

In [18] it was shown that many natural classes of functions (so called filter functions based on circuits with super-polynomial pseudo entropy on inputs from NP-complete language) cannot be obfuscated. The security requirement in the sense of Definition 3 is much weaker than that from [18] since in the case of TM obfuscation we are bound to check (3) only for those auxiliary inputs that are TMs $\widetilde{M}$ equivalent to an obfuscated TM M. Therefore, negative results and examples of [18] cannot testify the impossibility of universal software protecting obfuscation, and we may pose the following question.

Open problem. Does there exist a class of programs that are unobfuscatable in the sense of Definition 3?

Software protecting obfuscation has much in common with best possible obfuscation advanced by Goldwasser and Rothblum in [19]. An obfuscator O is judged as best possible if it transforms any program so that anything that can be computed given an access to the obfuscated program $O(M)$ should also be computable from any other equivalent program (of some related size).

Definition 5 ([19]). *A probabilistic algorithm $\mathcal{O}$ is a* best possible obfuscator *if it complies with* functionality *and* polynomial slow-down *requirements from Definition 1, and with the following* security *requirement:*

3) For any PPT A (learner) there is a PPT S (simulator) such that for every pair of equivalent TMs $(M, \widetilde{M})$ *of the same large enough size the distributions* $A(O(M))$ *and* $S(\widetilde{M})$ *are indistinguishable.*

The aim of software protection is to hide all specific properties of any particular implementation M of some function known to all (the latter is formalized by giving an adversery a description of this function as an arbitrary program $\widetilde{M}$ equivalent to M). Intuitively, this may be achieved iff for every pair of implementations M_1 and M_2 of the same function the distributions $O(M_1)$ and $O(M_2)$ are indistinguishable. But, as it was proved in [19], this is equivalent (in the case of efficient O) to the claim that O is a best possible obfuscator. Thus, it may be conjectured that software protection obfuscation is equivalent to best possible obfuscation. But to prove this conjecture formally one has to elaborate Definitions 3 and 5 to circumvent the discrepancy in the admissible size of an auxiliary program $\widetilde{M}$.

5 Constant Hiding

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

It is easy to see that constant hiding is closely related with obfuscation for software protection. Therefore we adopt Definition 3 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. Let M be a program which uses a constant c. For simplicity we assume that $c \in_R \{0,1\}^n$ for every integer n. For any given constant value $c \in \{0,1\}^n$ we denote by M_c the corresponding instantiation of the program M. Thus, we deal with a parameterized family of programs $\mathcal{F} = \{M_c \ : \ c \in \{0,1\}^n, \ n \geq 1\}$. The obfuscation of $\mathcal{F}$ is intended to prevent the extraction of information about a constant c from the program M_c.

Definition 6. *Let* $\mathcal{F}$ *be a parameterized family of TMs* M_c*. A probabilistic algorithm* $\mathcal{O}$ *is a* constant hiding obfuscator *for the family* $\mathcal{F}$ *if it complies with* functionality *and* polynomial slow-down *requirements from Definition 1, and with the following* security *requirement:*

3) For any PPT A there is a PPT S such that

$$|Pr\{A[\mathcal{O}(M_{c_0}), M_c] = 1\} - Pr\{S^{M_{c_0}}[1^{|M_{c_0}|}, M_c] = 1\} = neg(n) \quad (4)$$

holds for any constant value $c_0 \in \{0,1\}^n$ *and* $c \in_R \{0,1\}^n$*.*

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

Remark 3. Suppose that U is a universal TM. Then a constant c may be interpreted as an encoded description of some other TM M simulated by the instantiation U_c. In this case a constant hiding obfuscator for a universal TM is some specific variant of obfuscation w.r.t. dependent auxiliary input introduced in [18] (see Definition 4). At the same time, if all TMs M_c from the family $\mathcal{F}$ do not refer to a constant c along any execution then every semantic-preserving transformation $\mathcal{O}$ is a trivial constant hiding obfuscator for $\mathcal{F}$. 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. The obfuscations of point functions from [24,32] can be also viewed as constant hiding obfuscations (see also [18]).

Yet another variant of constant hiding obfuscation is defined in [22]. An obfuscator $\mathcal{O}$ for a family of programs $\mathcal{F}$ is regarded as a composition of two algorithms: a key transformation routine T that takes a key c and returns an obfuscated key $c' = T(c)$, and a polynomial time machine G which on input an obfuscated key c' and x outputs $y = M_c(x)$. The authors of [22] proved that if a secure secrete-key encryption scheme can be obfuscated according to their definition, then the result is a secure public-key encryption scheme. On the other hand, they also showed that there exists a secure probabilistic secret-key cryptosystem that cannot be obfuscated. Definition 6 specifies a more general model of constant hiding obfuscation than that of [22], since in our model it is assumed that obfuscating transformations may be applied not only to a constant c_0 to be hidden but also to the program M_{c_0} which is instantiated by the constant. Instead of using some fixed universal machine G we grant an adversary an access to a typical program M_c, since all TM from the family $\mathcal{F}$ are the same except for the constants.

6 Predicate Obfuscation

This is one of the weakest model of obfuscation. It is intended to hide only a particular distinguished property of a program (predicate). Nevertheless, predicate obfuscation may appear in numerous applications in computer security. Assume that some programs are infected with a virus which is triggered off at some executions. There are several approaches to a virus detection. A sandbox approach emulates the operating system, runs executables in this simulation and analyzes the sandbox for any changes which might indicate a virus. This approach is resource consuming and normally it takes place only during on-demand scans. The most effective virus detection technique is based on virus dictionary: an antivirus program tries to find virus-patterns inside ordinary programs by scanning them for so-called *virus signatures*. To avoid detection some viruses attempt to hide their signatures by using complex obfuscating transformations and rewriting their bodies completely each time they are to infect new executables. Also the viruses tend to "weave" their code into the host program making detection by traditional heuristics almost impossible since the virus share a great deal of its instructions with a host program code. A virus-detection problem may be

specified by a predicate $\pi(M)$ which evaluates to *true* whenever the program M includes some malware piece of code. An obfuscating transformation $\mathcal{O}$ used by a metamorphic malware could be regarded secure if any anti-virus scanner is no more effective in detecting an obfuscated virus signature (i.e. in evaluating $\pi(\mathcal{O}(M))$) than some sandbox virus detector.

To formalize this idea consider some predicate P defined on a family of TMs $\mathcal{F}$.

Definition 7. *A probabilistic algorithm $\mathcal{O}$ is an* obfuscator of the predicate π *on the family $\mathcal{F}$ if it complies with* functionality *and* polynomial slow-down *requirements from Definition 1, and with the following* security *requirement:*

3) For any PPT A there is a PPT S such that

$$|Pr\{A[\mathcal{O}(M)] = \pi(M)\} - Pr\{S^M[1^{|M|}] = \pi(M)\}| = neg(|M|) \qquad (5)$$

holds for every TM M from $\mathcal{F}$ and its obfuscation $\mathcal{O}(M)$.

Predicate obfuscation is related intrinsically with total obfuscation: a "virtual black box" obfuscator needs to hide all predicates, whereas Definition 7 allows to build different obfuscators for different predicates. As it may be seen from the proof of Theorem 1 given in [3], the impossibility of total obfuscation is certified by presenting an unobfuscatable predicate as a counterexample. On the other hand, predicate obfuscation is much weaker than total obfuscation. In [29] it was shown that if every TM from a family $\mathcal{F}$ computes either zero function, or a point function, then secure obfuscation of the predicate "$M(x) \not\equiv 0$" on the family $\mathcal{F}$ is possible under assumption that a one-way permutation exists. This obfuscation is based on the *hard-core predicate* construction suggested by Goldreich and Levin [17]. As it can be seen from theorem below, predicate obfuscation offers some nice properties: composition of predicate obfuscators increases the potency of obfuscation.

We say that a predicate π on TMs is a *functional property* if $\pi(M_1) = \pi(M_2)$ holds for every pair of equivalent TMs M_1, M_2.

Theorem 4. *Let $\mathcal{O}_1$ and $\mathcal{O}_2$ be obfuscator of functional properties π_1 and π_2, respectively. Suppose also that the range of O_2 is contained in the domain of O_1. Then the composition $\widehat{\mathcal{O}} = \mathcal{O}_1\mathcal{O}_2$ is an obfuscator of both predicates π_1 and π_2.*

Proof. We may assume that each obfuscator $\mathcal{O}_i$, $i = 1, 2$, is supplied with a polynomial $q_i(\cdot)$, and for every TM M the size of $\mathcal{O}_i(M)$ is exactly $q_i(|M|)$. This may be achieved by padding an obfuscated program with a sufficient amount of dummy instructions. Clearly, $\widehat{\mathcal{O}}$ satisfies the functionality and polynomial slow-down requirements.

Let M be an arbitrary TM.

1. To show that $\widehat{\mathcal{O}}$ obfuscates π_1 consider an arbitrary adversary $\widehat{A}$. Since $\mathcal{O}_1$ is an obfuscator of π_1, there exists a PPT S such that

$$|Pr\{\widehat{A}[\mathcal{O}_1(\mathcal{O}_2(M))] = \pi_1(\mathcal{O}_2(M))\} - Pr\{S^{\mathcal{O}_2(M)}[1^{|\mathcal{O}_2(M)|}] = \pi_1(\mathcal{O}_2(M))\}| = neg(|M|)$$

holds for the predicate π_1, TM $\mathcal{O}_2(M)$ and its obfuscation $\mathcal{O}_1(\mathcal{O}_2(M))$. Now we may consider a simulator $\widehat{S}$ which operates as follows: given an input 1^x, it computes $y = q_2(x)$, and applies the simulator S to 1^y. Since $\pi_1(\mathcal{O}_2(M)) = \pi_1(M)$ and $\mathcal{O}_2(M) \approx M$, we arrive at the equation

$$|Pr\{\widehat{A}[\widehat{\mathcal{O}}(M)] = \pi_1(M)\} - Pr\{\widehat{S}^M[1^{|M|}] = \pi_1(M)\}| = neg(|M|),$$

which is exactly the security requirement for π_1, TM M and its obfuscation $\widehat{\mathcal{O}}(M)$. Hence, $\widehat{\mathcal{O}}$ obfuscates π_1.

2. To show that $\widehat{\mathcal{O}}$ obfuscates π_2 consider an arbitrary adversary $\widehat{A}$ and the composition $A = \widehat{A}\mathcal{O}_1$. Since $\mathcal{O}_2$ is an obfuscator of π_2, there exists a PPT $\widehat{S}$ which simulates the adversary A so that

$$|Pr\{A[\mathcal{O}_2(M)] = \pi_2(M)\} - Pr\{S^M[1^{|M|}] = \pi_2(M)\}| = neg(|M|)$$

holds. Hence, $\widehat{\mathcal{O}}$ satisfies security requirement to predicate obfuscation for π_2. □

7 Conclusion

The models of program obfuscation presented in this paper evolved naturally from the "black box" simulation principle offered by Barak et al. in [3]. Security requirements given in Definitions 3, 6, and 7 make it possible to cover those cases of program obfuscation that do not fall into the "virtual black box" security paradigm. Although we have made only preliminary study of these new models, even such simple results as Theorems 3 and 4 show that as security requirements become weaker program obfuscation displays some new interesting features. On the other hand, Theorem 2 demonstrates that total program obfuscation remains impossible even in a framework of a weak "virtual grey box" obfuscation model under standard cryptographic assumptions.

Acknowledgements

We thank the anonymous reviewers for their insightful comments which help us to improve the paper.

References

1. Aucsmith, D.: Tamper resistant software: an implementation. In: Anderson, R. (ed.) Information Hiding. LNCS, vol. 1174, pp. 317–333. Springer, Heidelberg (1996)
2. Arboit, G.: A method for watermarking java programs via opaque predicates. 5-th International Conference on Electronic Commerce Research (2002)

3. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K.: On the (im)possibility of obfuscating programs. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 1–18. Springer, Heidelberg (2001)
4. Bhatkar, S., DuVarney, D.C., Sekar, R.: Efficient techniques for comprehensive protection from memory error exploits. USENIX Security (2005)
5. Canetti, R.: Towards realizing random oracles: hash functions that hide all partial information. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 455–469. Springer, Heidelberg (1997)
6. Canetti, R., Micciancio D.D., Reingold, O: Perfectly one-way probabilistic hash functions. 30-th ACM Symposium on Theory of Computing, 131–140 (1998)
7. Chess, D., White, S.: An undetectable computer virus. In: 2000 Virus Bulletin Conference (2000)
8. Chow, S., Gu, Y., Johnson, H., Zakharov, V.: An approach to obfuscation of control-flow of sequential programs. In: Wilhelm, R. (ed.) Informatics. LNCS, vol. 2000, pp. 144–155. Springer, Heidelberg (2001)
9. Cohen, F.: Operating system protection through program evolution. Computers and Security 12(6), 565–584 (1993)
10. Collberg, C., Thomborson, C., Low, D.: A Taxonomy of Obfuscating Transformations. Tech. Report, N 148, Univ. of Auckland (1997)
11. Collberg, C., Thomborson, C., Low, D., Collberg, C., Thomborson, C., Low, D.: Manufacturing cheap, resilient and stealthy opaque constructs. In: Symposium on Principles of Programming Languages, pp. 184–196 (1998)
12. Collberg, C., Thomborson, C.: Watermarking, Tamper-Proofing, and Obfuscation - Tools for Software Protection. IEEE Transactions on Software Engineering 28(6) (2002)
13. Dalla Preda, M., Giacobazzi, R.: Semantic-based code obfuscation by abstract interpretation. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 1325–1336. Springer, Heidelberg (2005)
14. D'Anna, L., Matt, B., Reisse, A., Van Vleck, T., Schwab, S., LeBlanc, P.: Self- Protecting Mobile Agents Obfuscation Report, Report #03-015, Network Associates Laboratories (June 2003)
15. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Transactions om Information Theory IT-22(6), 644–654 (1976)
16. Dodis, Y., Smith, A.: Correcting errors without leaking partial informtion. 37-th ACM Symposium on Theory of Computing, 654–663 (2005)
17. Goldreich, O., Levin, L: A hard-core predicate to any one-way function. 21-th ACM Symposium on Theory of Computing, 210–217 (1989)
18. Goldwasser, S., Tauman Kalai, Y.: On the impossibility of obfuscation with auxiliary input. 46-th IEEE Symposium on Foundations of Computer Science, 553–562 (2005)
19. Goldwasser, S., Rothblum, G.N.: On best possible obfuction. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 194–213. Springer, Heidelberg (2007)
20. Hada, S.: Zero-knowledge and code obfuscation. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 443–457. Springer, Heidelberg (2000)
21. Hohl, F.: Time limited blackbox security: protecting mobile agents from malicious hosts. In: Vigna, G. (ed.) Mobile Agents and Security. LNCS, vol. 1419, pp. 92–113. Springer, Heidelberg (1998)
22. Hofheinz, D., Malone-Lee, J., Stam, M.: Obfuscation for cryptographic purpose. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 214–232. Springer, Heidelberg (2007)

23. Linn, C., Debray, S.: Obfuscation of executable code to improve resistance to static disassambly. In: 10-th ACM Conference on Computer and Communication Security, pp. 290–299 (2003)
24. Lynn, B., Prabhakaran, M., Sahai, A.: Positive results and techniques for obfuscation. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 20–39. Springer, Heidelberg (2004)
25. Ogiso, T., Sakabe, Y., Soshi, M., Miyaji, A.: Software obfuscation on a theoretical basis and its implementation. IEEE Trans. Fundamentals E86-A(1) (2003)
26. Ostrovsky, R., Skeith, W.E.: Private searching on streaming data. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 223–240. Springer, Heidelberg (2005)
27. Szor, P., Ferrie, P.: Hunting for metamorphic. In: 2001 Virus Bulletin Conference, pp. 123–144 (2001)
28. Valiant, L.: A theory of learnable. Communications of the ACM 27(11), 1134–1142 (1984)
29. Varnovsky, N.P., Zakharov, V.A.: On the possibility of provably secure obfuscating programs. In: Broy, M., Zamulin, A.V. (eds.) PSI 2003. LNCS, vol. 2890, pp. 91–102. Springer, Heidelberg (2004)
30. Varnovsky, N.P.: A note on the concept of obfuscation. In: Proceedings of Institute for System Programming, Moscow, vol. 6, pp. 127–137 (2004)
31. Wang, C., Davidson, J., Hill, J., Knight, J.: Protection of software-based survivability mechanisms. In: International Conference of Dependable Systems and Networks (2001)
32. Wee, H.: On obfuscating point functions. In: 37-th Symposium on Theory of Computing, pp. 523–532 (2005)