On the Possibility of Provably Secure Obfuscating Programs*

Nikolay P. Varnovsky[1,3] and Vladimir A. Zakharov[2,3]

[1] Laboratory for Cryptography,
Lomonosov State University, Moscow, RU-119899, Russia
[2] Faculty of Computational Mathematics and Cybernetics,
Lomonosov State University, Moscow, RU-119899, Russia
zakh@cs.msu.su
[3] Institute for System Programming, Russian Academy of Sciences,
B. Kommunisticheskaya, 25, 109004 Moscow, Russia

Abstract. By obfuscation we mean any efficient semantic-preserving transformation of computer programs aimed at bringing a program into such a form, which impedes the understanding of its algorithm and data structures or prevents the extracting of some valuable information from the plaintext of a program. The main difficulty in designing an effective program obfuscator is to guarantee security, i.e. to prove that no algorithm can break software protection in reasonable time. All obfuscation techniques and tools developed so far rely on the informal concept of security and therefore can't be regarded as provably secure. In this paper we (1) introduce for the first time a formal information-theoretic definition of obfuscation security, (2) present a new obfuscation technique which takes advantage of cryptographic primitives (one-way functions, hard-core predicates), and (3) demonstrate, taking a conventional password identification scheme as a case study, how to prove security of the obfuscating transformations.

Keywords: program transformation, obfuscation, security, mutual information, one-way function, hard-core predicate.

1 Introduction

Protection of software against intelligent tampering and unauthorized purposeful modifications is one of the central issues in computer security. Almost every software-controlled system faces threats from potential adversaries, from Internet-aware client applications running on PCs, to complex telecommunications and power systems accessible over Internet, to commodity software with copy protection mechanisms. Various methods and tools are widely used for the purpose of software protection, including sophisticated security policies, network filters, cryptosystems, tamper-resistant hardware, etc. [1,8,10]. But no matter

* This work was supported by the Russian Foundation for Basic Research (grant 03-01-00880)

M. Broy and A.V. Zamulin (Eds.): PSI 2003, LNCS 2890, pp. 91–102, 2003.

how powerful these techniques may be, they don't cover the case when an adversary, having in mind to make an illegal modification of a program or to gain some valuable knowledge about algorithms or data structures, gets an access to the plaintext of a program. The current trends in software engineering and communication technology make it common to distribute software in such a form that retains most of information presented in the program source code. This increases drastically the risk of reverse engineering attacks aimed at extracting secret information from a program.

An important example is Java bytecode. Java applications are distributed as Java class files, hardware-independent virtual machine codes that retains almost all information of the original Java source. The customary cryptographic tools can be very effective for protecting programs from illegal usage at the stage of their distribution. But when a program is decrypted, it becomes extremely vulnerable to software pirates seeking for private information (passwords, data-keys, etc.) or valuable pieces of code to incorporate them in their own applications.

In these cases the only way to prevent such malicious activity is to convert a program into some tamper-resistant form, which has the property that understanding and making purposeful modifications to it are rendered difficult while its original functionality and efficiency are preserved. Program transformations of this kind are called *obfuscating transformations*. The plaintext of an obfuscated program becomes itself the "ultimate defensive line" of the program.

Apart from software protection, program obfuscators could enjoy wide application in cryptography. When introducing the concept of public-key cryptosystem in the seminal paper [9], Diffie and Hellman noticed that, given any means for obscuring data structures in a symmetric-key encryption algorithm, one could convert this algorithm into a public-key cryptosystem. Obfuscators would also allow one to convert any public-key cryptosystem into a homomorphic one, i.e. a public-key cryptosystem which, given encryption of two bits, can securely compute an encryption of any Boolean operation of these bits. The existence of homomorphic encryption schemes is a long-standing open problem in cryptography.

The concept of obfuscating transformation was introduced in [4]. In this paper an obfuscator is defined informally as any efficient probabilistic compiler $\mathcal{O}$ which transforms any source program π into a new program $\mathcal{O}(\pi)$ satisfying the following requirements:

- $\mathcal{O}(\pi)$ has the same observable behavior, and
- $\mathcal{O}(\pi)$ is substantially less intelligible (readable) than π.

To get a formal definition of obfuscating transformation one has to clarify the terms "the same observable behavior" and "less intelligible". Now it is commonly accepted that when dealing with sequential deterministic programs the first term means that programs π and $\mathcal{O}(\pi)$ compute the same input-output relation and the complexity of $\mathcal{O}(\pi)$ (its size, time and space complexity, etc.) is at most polynomially larger than that of π.

The second term is still a topic for discussions. The authors of the most papers on program obfuscation restrict their consideration only to the expound-

ing of the intuitive meaning of this term. Thus, Collberg and Thomborson [7] require an obfuscating transformation $\mathcal{O}$ to increase the *obscurity* of π so that the understanding and reverse engineering of $\mathcal{O}(\pi)$ will be strictly more time-consuming than the understanding and reverse engineering of π. Since static analysis of programs is in wide use for the purposes of program understanding and reverse engineering, this line of research on obfuscating transformations regards obfuscator as a tool for obstructing static analysis (see [18]). There have been many ad hoc obfuscating techniques proposed: a taxonomy of approaches is discussed in [4,5,18,19]. The most advanced of them are based on the following principle: find some NP-hard problem $\mathcal{P}$, define a class $\mathcal{F}$ of program fragments whose efficient analysis implies resolution of $\mathcal{P}$ in polynomial time, and develop a technique for inserting safely fragments from $\mathcal{F}$ into an arbitrary program. Collberg et al. [5,6] exploit NP-hardness of pointer alias analysis [13,14,17] for constructing opaque predicates, i.e. predicates whose behavior is difficult for understanding. The hard cases of points-to analysis problem is also used in [18] for increasing software tamper-resistance. In [3] a universal technique was proposed which makes it possible to "implant" uniformly and safely an arbitrary PSPACE-complete problem into any program.

The principal drawback of all these techniques is that none of them has a formal basis for making claims about the difficulty of understanding an obfuscated program. Therefore it is hardly possible to estimate to what extent such methods serve the purpose — only the reference to the intractability of some static analysis problems or to the hardness of combinatoric problem embedded into obfuscated programs is not sufficient. Meanwhile, the need for formalization of security requirements was realized in the earliest studies on foundation of cryptography, and a number of suitable security criteria were developed. M. Blum (see [15]) noticed that some ideas and techniques from cryptography are worthy of being adapted for program obfuscation. Unfortunately, the straightforward application of these techniques to the obfuscation problem is hardly possible, since obfuscating transformations ought to preserve functionality of source programs, whereas semantic-preserving encryptions are few and far between.

A formal investigation of the very concept of program obfuscation in the context of present-day cryptography was initiated in [2,12]. To formalize the "unreadability" requirement Barak, Goldreich, et al. introduced in [2] the concept of "perfect" obfuscation. Intuitively, a program obfuscator $\mathcal{O}$ is called *perfect* if it transforms any program π into a "virtual black box" $\mathcal{O}(\pi)$ in the sense that anything one can efficiently compute given $\mathcal{O}(\pi)$, one should be able to efficiently compute given just oracle access to π. The main result of [2] implies that perfect obfuscation is impossible. This is proved by constructing a family $\mathcal{F}$ of functions and a predicate $P : \mathcal{F} \to \{0, 1\}$ such that, given any program that computes a function $f \in \mathcal{F}$, the value $P(f)$ can be efficiently computed, whereas no efficient algorithm, being given only oracle access to a randomly selected $f \in \mathcal{F}$, can compute $P(f)$ much better than by random guessing.

Our brief review shows that there is an amazing gap between the well-proved negative result of [2], which excludes the possibility of omnipotent obfuscators,

and numerous heuristic algorithms [3,4,5,6,7,18,19] attempting some vague kind of obfuscation. The aim of our paper is to fill (at least partially) this gap. We put forward an alternative meaningful definition of obfuscation security which allows to estimate in information-theoretic terms the capability of a semantic-preserving program transformation $\mathcal{O}$ to obfuscate an individual property P for a given class of programs $\mathcal{H}$. Informally, $\mathcal{O}$ obfuscates securely a property P if any adversary A, being granted an access to programs from $\mathcal{O}(\mathcal{H})$, can extract in polynomial time only a "negligibly small" amount of information on P. To show that the new formalization of obfuscation security enables one to design provably secure program obfuscators we consider one meaningful example. We take for $\mathcal{H}$ a class of programs that implement a simple fixed password identification scheme, and for P the property of a program to execute actual password checking. Then, assuming that one-way permutations exist, we design a program compiler $\mathcal{O}$ and prove that $\mathcal{O}$ securely obfuscates P. Moreover, by demonstrating that a secure obfuscation of P implies the existence of one-way functions, we also reveal close relationships between obfuscation problem and basic cryptographic primitives.

The paper is organized as follows. In Section 2 we give some basic definitions along with some facts from complexity theory that are necessary for proving our main result. In Section 3 we introduce an information-theoretic definition of obfuscation security. Next (Section 4) we consider an obfuscation problem for a fixed password identification scheme, present an obfuscating transformation $\mathcal{O}$ and prove its security. Finally (Section 5) we show how to apply $\mathcal{O}$ in practice to strengthen the tamper resistance of an arbitrary program.

2 Preliminaries

We denote by B the set $\{0,1\}$, by B^n the set of all binary strings of length n, and by B^* the set of all finite binary strings. If x and y are binary strings of the same length, say n, then we write $\bigoplus(x,y)$ for $\bigoplus_{i=1}^{n} x_i y_i$. PPA is shorthand for probabilistic polynomial time algorithms. Each PPA A is associated with a pair of polynomials p_1, p_2. If A is a PPA then by $A(x,r)$ we refer to the result of running A within $p_1(|x|)$ steps on input x, $x \in B^*$, and a random string r choosing uniformly from $B^{p_2(|x|)}$. By $A(x)$ we refer to the distribution induced by choosing r uniformly and running $A(x,r)$. If D is a distribution then by $x \stackrel{R}{\leftarrow} D$ we mean that x is randomly distributed according to D. When writing $x \stackrel{R}{\leftarrow} B^n$ we mean that x is a random binary string distributed uniformly over the elements of B^n.

We denote by $\mathcal{U}$ some universe of computer programs. The nature of $\mathcal{U}$ is of minor importance; for definiteness sake we consider ALGOL-style programs. Each program π computes some recursive function $F_\pi : \mathit{Dom}_{In} \to \mathit{Dom}_{Out}$. Programs π_1 and π_2 are said to be equivalent ($\pi_1 \sim \pi_2$ in symbols) iff $F_{\pi_1} = F_{\pi_2}$. We denote by comp_π some complexity measure of program π; for example, $\mathit{comp}_\pi(x)$ may be thought of as the number of basic actions (steps) to be executed by π for computing $F_\pi(x)$. When a program π is analyzed by an algorithm A,

we assume that some suitable encoding is used to represent π as a binary string x, and by writing $A(\pi)$ we refer to the result of running A on such x. The length of such binary string x is denoted by $size(\pi)$.

A function $\mu : \mathbb{N} \to \mathbb{N}$ is called *negligible* if for any positive polynomial $p(\cdot)$ there exists $N \in \mathbb{N}$ such that $\mu(n) < 1/p(n)$ for any $n > N$. We will sometimes use $neg(\cdot)$ and $poly(\cdot)$ to denote unspecified negligible function and positive polynomial, respectively.

A sequence $\varphi = \{\varphi_n\}_{n\in\mathbb{N}}$ of functions $\varphi_n : B^n \to B^{m(n)}$ is called *one-way function* if the following conditions hold

1. $n < poly(m(n))$,
2. there is a deterministic polynomial time algorithm C such that $C(x) = \varphi_n(x)$ for every $x \in B^n,\ n \in \mathbb{N}$,
3. $\Pr_{x\overset{R}{\leftarrow}B^n}[\varphi_n(A(\varphi_n(x))) = \varphi_n(x)] = neg(n)$, for any PPA A.

This means that it is "easy" to compute φ for all binary strings x but for "essentially all" elements $y \in Im(\varphi)$ it is "computationally infeasible" to find any string x such that $\varphi(x) = y$. If every φ_n is a bijection then φ is called a *one-way permutation*. It is still unknown whether one-way functions exist since the existence of one-way functions implies P≠NP. But nevertheless, some functions (discrete logarithm, RSA-function, etc., see [16]) are strongly believed to be one-way. One-way functions are used widely in cryptography for designing public-key cryptosystems, pseudorandom generators, hash-functions, etc. A cryptographic method is said to be *provably secure* if its defeating can be shown to be essentially as hard as solving a well-known and supposedly difficult problem (such as inverting one of the above functions).

A function $h : B^* \to B$ is said to be a *hard-core predicate* for a one-way function φ if the following conditions hold

1. there is a deterministic polynomial time algorithm C' which computes h,
2. if there is a PPA A_1 such that

$$\Pr_{x\overset{R}{\leftarrow}B^m}[A_1(\varphi_m(x)) = h(x)] > \frac{1}{2} + 1/poly(m)$$

holds for infinitely many $m \in \mathbb{N}$ then there exists a PPA A_2 such that

$$\Pr_{x\overset{R}{\leftarrow}B^n}[\varphi_n(A_2(\varphi_n(x))) = \varphi_n(x)] > 1/poly(n)$$

holds for infinitely many $n \in \mathbb{N}$.

In other words, an oracle which computes $h(x)$ with probability significantly greater than $1/2$, given only $\varphi(x)$, can be used to invert φ efficiently.

Goldreich and Levin [11] showed that any one-way function φ can be transformed into a one-way function ψ which has a hard-core predicate. Their construction is as follows. Define the function ψ by $\psi(u, x) = (u, \varphi(x))$, where u is a binary string of the same length as x. Then ψ is also a one-way function and $h(u, x) = \bigoplus(u, x)$ is a hard-core predicate for ψ.

3 Program Obfuscators

By program obfuscation we mean any semantic-preserving transformation aimed at bringing a program into such a form which impedes as much as possible the extracting of some valuable information from the plaintext of a program.

Definition 1. *A probabilistic algorithm $\mathcal{O}$ is a* program obfuscator *if for every program π, $\pi \in \mathcal{U}$, the following conditions hold:*

(1) $\pi \sim \mathcal{O}(\pi)$ *(functionality preserving);*
(2) $size(\mathcal{O}(\pi)) \le poly(size(\pi))$ *(polynomial expansion);*
(3) $comp(\mathcal{O}(\pi)) \le poly(comp(\pi))$ *(polynomial slowdown).*

But the most important issue which characterizes the capability of $\mathcal{O}$ to hide the specific properties of programs is security. Informally, an obfuscator $\mathcal{O}$ is called *secure* w.r.t. an ensemble of programs $\mathcal{H}$ and a secret property P if any (possible) adversary, when being granted a random access to obfuscated programs $\mathcal{O}(\pi)$, $\pi \in \mathcal{H}$, can extract in reasonable time only a negligibly small amount of information on P.

An *ensemble* of programs is a sequence $\mathcal{H} = \{(S_n, D_n)\}_{n\in\mathbb{N}}$, where S_n, $n \in \mathbb{N}$, is a sample set of programs from $\mathcal{U}$, and D_n is a probability distribution on S_n. Denote by $S_{\mathcal{H}}$ the set $\bigcup_{n\in\mathbb{N}} S_n$ of all programs that appear in $\mathcal{H}$.

Let $\mathcal{H}$ be an ensemble of programs. We call a *secret property* any predicate P defined on $S_{\mathcal{H}}$. For every $n \in \mathbb{N}$ such P can be considered as a random variable defined on S_n. We write $\Pr_{\pi \overset{R}{\leftarrow} D_n}[P(\pi) = \sigma]$, $\sigma \in B$, for the probability of a randomly chosen program π from S_n satisfying (case $\sigma = 1$) or not satisfying (case $\sigma = 0$) the secret property P. If for every pair of programs π_1, π_2, the equivalence $\pi_1 \sim \pi_2$ implies $P(\pi_1) = P(\pi_2)$ then P is called a *semantic property*.

By an *adversary* we mean any set $\mathcal{A}$ of PPAs; the elements from $\mathcal{A}$ will be called *attacks*. Each attack takes programs from $\mathcal{U}$ as inputs and outputs one bit. Thus, an attack A can be viewed as a randomized checker of a secret property of programs. To hamper the deducing of the secret property one could apply some obfuscator $\mathcal{O}$ to the source programs. When an ensemble $\mathcal{H}$ is fixed, this forces an adversary $\mathcal{A}$ to deal with the set of programs $\{\mathcal{O}(\pi) : \pi \in S_{\mathcal{H}}\}$. The result $A(\mathcal{O}(\pi))$ of an attack A on a randomly chosen obfuscated program $\mathcal{O}(\pi)$ can be considered as a random variable defined on the sample set of triples (π, x, y), where π is a program from $\mathcal{O}(S_n)$ and x, y are random bit strings used by the PPAs A and $\mathcal{O}$. We denote by $\Pr_{\pi \overset{R}{\leftarrow} D_n}[A(\mathcal{O}) = \delta]$, $\delta \in B$, the probability of the attack A completing with the result δ on obfuscated programs $\mathcal{O}(\pi)$, where π is distributed over S_n. In what follows we will write $\Pr_n$ instead of $\Pr_{\pi \overset{R}{\leftarrow} D_n}$ to simplify notation.

Definition 2. *Let $\mathcal{H}$ be an ensemble of programs, P be a secret property, $\mathcal{A}$ be an adversary, and $\mathcal{O}$ be an obfuscator. We say that $\mathcal{O}$* securely obfuscates *P* w.r.t. *$\mathcal{H}$* and *$\mathcal{A}$ if for every attack $A \in \mathcal{A}$ the following condition holds*

(4) $I_n(P, A(\mathcal{O})) = neg(n)$, *(security)*

where $I_n(P, A(\mathcal{O}))$ *is the mutual information of random variables* P *and* $A(\mathcal{O})$,

$$I_n(P, \mathcal{A}(\mathcal{O})) = \sum_{\alpha=0}^{1} \sum_{\beta=0}^{1} \Pr_n[P(\pi) = \alpha, A(\mathcal{O}(\pi)) = \beta] \times \\ \times \log \frac{\Pr_n[P(\pi)=\alpha, A(\mathcal{O}(\pi))=\beta]}{\Pr_n[P(\pi)=\alpha]\Pr_n[A(\mathcal{O}(\pi))=\beta]}.$$

Clearly, the possibility of designing secure obfuscators depends highly on the secret properties to be obfuscated and the capability of an adversary. For example, if $P(\pi)$ is specified by the assertion "the program π uses the variable PSI" then P can be readily and securely obfuscated w.r.t. any ensemble of programs and any set of attacks by renaming variables in π. Nontrivial cases are that of a secret property P being semantic, since such predicates are invariant with respect to any obfuscation $\mathcal{O}$. It is easy to see that some semantic properties P admit no obfuscation at all (e.g. when $P(\pi)$ is specified by $F_\pi(0) = 0$).

4 The Main Result

To demonstrate that the requirements (1)–(4) can be satisfied by some compiler we consider the obfuscation problem for the fixed password identification scheme as the case study.

Suppose that a software engineer Alice eventually decides to protect from unauthorized access certain of the programs she designed. When selecting a program to be protected she chooses randomly a binary string for a password and adds to the program a fixed password identification scheme. A tamper engineer Bob, given a program designed by Alice, attempts to compromise the software protection. To obstruct his activity Alice may apply some obfuscator to her programs so that Bob could not distinguish the programs that actually use password identification from those having free access.

To help Alice to attain these ends we consider an ensemble of programs $\mathcal{H}_0 = \{(S_n, D_n)\}_{n\in\mathbb{N}}$, where $S_n = \{\pi_0\} \cup \{\pi_1^w : \ w \in B^n\}$, and the probability distribution D_n is defined by $D_n(\pi_0) = 1/2$ and $D_n(\pi_1^w) = 1/2^{n+1}$. The programs π_0 and π_1^w are depicted in Fig. 1. The program π_0 simulates free access, whereas the programs π_1^w implement a simple identification scheme.

```
prog π0;
var x: string, y: bit;
input(x);
y:=0; output(y);
end of prog;
```

```
prog π1^w;
var x: string, y: bit;
input(x);
if x=w then y := 1 else y:=0;
output(y);
end of prog;
```

Fig. 1.

Bob is aimed at checking whether a program contains some identification scheme. Therefore, the secret property $P_0(\pi)$ to be obfuscated is specified by the

predicate $\exists x(F_\pi(x) = 1)$. Clearly, $P_0(\pi_0) = 0$ and $P_0(\pi_1^w) = 1$. Bob is allowed to apply any algorithm running in polynomial time to achieve his goal. Hence, by the adversary $\mathcal{A}_0$ we mean the set of all PPAs.

Theorem 1. *If one-way permutations exist then the secret property P_0 can be securely obfuscated w.r.t. $\mathcal{H}_0$ and $\mathcal{A}_0$.*

Proof. Suppose φ is a one-way permutation computed by some deterministic polynomial time algorithm ONE_WAY. We will assume that it is implemented as a built-in function. A compiler $\mathcal{O}$, given a program π from S_n, computes a quadruple (w, u, v, σ) and builds the program $\Pi^{w,u,v,\sigma}$ (see Fig. 2). This program uses values u, v and σ as constants. The quadruple (w, u, v, σ) is computed as follows. If $\pi = \pi_0$ then $\mathcal{O}$ chooses uniformly at random a pair of bit strings $w, u \in B^n$ and sets $v = \varphi(w)$, $\sigma = \bigoplus(w, u)$. If $\pi = \pi_1^w$ then $\mathcal{O}$ chooses uniformly at random a string $u \in B^n$ and sets $v = \varphi(w)$, $\sigma = 1 \oplus \bigoplus(w, u)$.

```
prog Π^{w,u,v,σ};
var x: string, y: bit;
const u, v : string, σ : bit;
input(x);
if ONE_WAY(x) = v then
    if SUM(x,u) = σ then y:=0
    else y:=1
else y:=0
output(y);
end of prog;
```

```
function SUM (X,Y : string);
var Z: bit, i : integer;
Z:=0;
for i=0 to n do Z:=Z ⊕ X[i] * Y[i];
return Z;
end of function;
```

Fig. 2.

It is easy to see that $\mathcal{O}$ is an obfuscator: for every program π from $S_{\mathcal{H}_0}$ a program $\mathcal{O}(\pi)$ is equivalent to π, the size of $\mathcal{O}(\pi)$ is almost the same as that of π, and the time complexity of an obfuscated program $\mathcal{O}(\pi)$ is at most polynomially larger than that of π.

To show that $\mathcal{O}$ securely obfuscates P_0 w.r.t. $\mathcal{H}_0$ and $\mathcal{A}_0$ one should notice that $\mathcal{O}(\pi) = \Pi^{w,u,v,\sigma}$ implies $P_0(\pi) = \sigma \oplus \bigoplus(w, u)$. Hence, if by applying some attack A to programs $\Pi^{w,u,v,\sigma}$ one could extract a "significant" amount of information on P_0 then A could be used for computing $h(u, w) = \bigoplus(w, u)$ efficiently on a "significant" amount of inputs $u, \varphi(w)$. But the latter is impossible, since $h(u, w)$ is a hard-core predicate for the one-way function ψ associated with the one-way function φ (see Sect. 2).

These intuitive considerations give rise to the formal proof as follows.

Suppose the obfuscator $\mathcal{O}$ doesn't satisfy security requirement (4) (see Definition 2), i.e. there exists a PPA A such that $I_n(P_0, A(\mathcal{O})) > 1/poly(n)$ holds for infinitely many $n \in \mathbb{N}$. In view of the definition of $I_n(P_0, A(\mathcal{O}))$ and the facts that

$$\Pr_n[P_0(\pi) = 0] = \Pr_n[P_0(\pi) = 1] = 1/2$$

and

$$\Pr_n[A(\mathcal{O}(\pi))=\delta] = \Pr_n[P_0(\pi)=0, A(\mathcal{O}(\pi))=\delta] + \Pr_n[P_0(\pi)=1, A(\mathcal{O}(\pi))=\delta]$$

this implies that for some pair $\alpha, \beta \in B$

$$\Pr_n[P_0(\pi) = \alpha, A(\mathcal{O}(\pi)) = \beta] > 1/poly(n),$$
$$\Pr_n[P_0(\pi) = \alpha, A(\mathcal{O}(\pi)) = \beta] - \Pr_n[P_0(\pi) = 1-\alpha, A(\mathcal{O}(\pi)) = \beta] > 1/poly(n)$$

hold infinitely often. Consider an attack A' which operates as follows. Given a program $\pi' \in \mathcal{O}(S_{\mathcal{H}_0})$, it first applies A to π'. If $A(\pi', x) = 1 - \beta$ (recall that x is a random string used by A) then A' decides by tossing a coin. Otherwise, A' outputs α. It is easy to see that $\Pr_n[A'(\mathcal{O}(\pi)) = P_0(\pi)] > 1/2 + 1/poly(n)$. Thus, A' can be used for computing $\bigoplus(w, u)$ with a "significant" probability on infinitely many inputs $u, \varphi(w)$. Taking into account the result of [11] (see Section 2), we arrive at the contradiction with the assumption that φ is a one-way permutation.

Thus, $\mathcal{O}$ is a secure obfuscator of the secret property P_0 w.r.t. the ensemble $\mathcal{H}_0$ of the password identification programs and the set $\mathcal{A}_0$ of the probabilistic polynomial-time attacks. □

Theorem 2. *If the secret property P_0 can be securely obfuscated in polynomial time w.r.t. $\mathcal{H}_0$ and $\mathcal{A}_0$ then one-way functions exist.*

Proof. Suppose that PPA $\mathcal{O}$ securely obfuscates P_0 w.r.t. $\mathcal{H}_0$ and $\mathcal{A}_0$. Consider a function φ_0 defined as follows:

$$\varphi_0(\sigma, w, z) = \begin{cases} \mathcal{O}(\pi_0, z), \text{ if } \sigma = 0, \\ \mathcal{O}(\pi_1^w, z), \text{ if } \sigma = 1, \end{cases}$$

where $\sigma \in B$ is a bit, and $w, z \in B^*$ are bit strings. Clearly, $Im(\varphi_0) = \mathcal{O}(S_{\mathcal{H}_0})$, and, moreover, for every pair w, z we have $\sigma = P_0(\varphi_0(\sigma, w, z))$. Suppose that some PPA A correctly inverts φ_0 for a "significant" amount of elements from $\mathcal{O}(S_{\mathcal{H}_0})$. In such an event, there exists a PPA A' which on inputs $\mathcal{O}(\pi)$ computes $\sigma = P_0(\pi)$ more accurately than by random guessing. This implies that the mutual information $I_n(P_0, A'(\mathcal{O}))$ is non-negligible in contrary to the assumption that $\mathcal{O}$ is a secure obfuscator.

More formally.

Suppose φ_0 is not a one-way function. Then there exists a PPA A such that

$$\Pr_{\sigma \stackrel{R}{\leftarrow} B, w \stackrel{R}{\leftarrow} B^n, z \stackrel{R}{\leftarrow} B^n}[\varphi_0(A(\varphi_0(\sigma, w, z))) = \varphi_0(\sigma, w, z)] > 1/poly(n)$$

holds for infinitely many $n \in \mathbb{N}$. This inequality can be also written as follows:

$$\Pr_n[\varphi_0(A(\mathcal{O}(\pi))) = \mathcal{O}(\pi)] > 1/poly(n). \qquad (*)$$

Now consider a PPA A' which operates as follows. Given a program $\pi' \in \mathcal{O}(S_{\mathcal{H}_0})$, it first applies A to π'. If A inverts φ_0 on input π' correctly (this can be checked

effectively) then A' outputs the first bit of the string $A(\pi')$ which is $\sigma = P_0(\pi')$. Otherwise A' tosses a coin to output a random bit. It follows from $(*)$ that

$$\Pr_n[A'(\mathcal{O}(\pi)) = P(\pi)] > 1/2 + 1/poly(n)$$

holds for infinitely many n. Hence, at least for one value $\alpha \in B$ the attack A' guesses the secret property P_0 of obfuscated programs with a significant accuracy, i.e.

$$\Pr_n[P(\pi) = \alpha, A'(\mathcal{O}(\pi)) = \alpha] - \Pr_n[P(\pi) \neq \alpha, A'(\mathcal{O}(\pi)) = \alpha] > 1/poly(n)$$

holds for infinitely many $n \in \mathbb{N}$. In view of the facts that

$$\Pr_n[P_0(\pi) = \alpha] = 1/2$$

and

$$\Pr_n[A'(\mathcal{O}(\pi)) = \alpha] = \Pr_n[P_0(\pi) = \alpha, A'(\mathcal{O}(\pi)) = \alpha] + \Pr_n[P_0(\pi) \neq \alpha, A'(\mathcal{O}(\pi)) = \alpha]$$

holds, this implies

$$\Pr_n[P(\pi) = \alpha, A'(\mathcal{O}(\pi)) = \alpha] \times \log \frac{\Pr_n[P(\pi) = \alpha, A'(\mathcal{O}(\pi)) = \beta]}{\Pr_n[P(\pi) = \alpha]\Pr_n[A'(\mathcal{O}(\pi)) = \beta]} > 1/poly(n)$$

for infinitely many n. It immediately follows that $I_n(P, A'(\mathcal{O})) > 1/poly(n)$ in contrary to the assumption that $\mathcal{O}$ is a secure obfuscator.

Thus, φ_0 is a one-way function. □

5 Conclusions

We introduce a new formal definition of program obfuscation security which enables us to judge the capability of tools intended for obscuring individual semantic properties of programs. The essential feature required of a secure obfuscation is that there be only a negligible "leakage" of information on the secret property under the "pressure" of polynomial time algorithms. The new definition of obfuscation security agrees well with intuition, but at the same time it is much weaker than that from [2]. This opens up fresh opportunities for invoking complexity-theoretic and cryptographic techniques to software protection. To be certain that our approach has some success we consider a meaningful example of programs implementing a password identification scheme and demonstrate with the commonly accepted assumptions that some key property of such programs admits secure obfuscation. This simple case study results in the "perfect" obfuscator for the class of computer programs supplied with a password identification scheme.

The obfuscating transformation presented in the proof of Theorem 1 can be extended and adapted for strengthening tamper-resistance of an arbitrary computer program which uses fixed password identification scheme.

Consider some one-way function φ. The extended obfuscator $\mathcal{O}_{ext}$, given a program π protected by an identification scheme with a password w, operates as follows:

1. $\mathcal{O}_{ext}$ decomposes π into separate fragments $B_1, B_2, \ldots, B_n$ and forms a set of spurious fragments $S = \{B'_1, \ldots, B'_k\}$;
2. $\mathcal{O}_{ext}$ computes $v = \varphi(w)$ and adds to the program a new bit string variable x;
3. for every i, $1 \leq i \leq n$, the extended obfuscator $\mathcal{O}_{ext}$ chooses randomly a bit string u_i such that $|u_i| = |w|$, computes $\sigma_i = \bigoplus(u_i, w)$ and replaces the fragment B_i with a statement

   `if` $\bigoplus(u_i, x)$ `then` B^1 `else` B^0

   such that $B^{\sigma_i} = B_i$ and $B^{1-\sigma_i} \in S$;
4. protects the program through the use of the conventional identification scheme

   `input(`x`);`
   `if` $\varphi(x)$ $=$ v `then` $ACCESS\_GRANTED$ `else` $ACCESS\_DENIED$;

If we consider the limiting case, when each separate fragment B_i represents a single bit operation, then Theorem 1 guarantees that no adversary can extract any "significant" information on π from the plaintext of thus obfuscated program. This means that the "perfect" secure obfuscation of programs supplied with password identification scheme is possible. One could trace the parallels between the above obfuscating transformation and one-time pad symmetric-key cryptosystems [16].

References

1. Amoroso E.G. *Fundamentals of Computer Security Technology.* Englewood Cliffs, NJ: Prentice Hall PTR, 1994.
2. Barak B., Goldreich O., Impagliazzo R., Rudich S., Sahai A., Vedhan S., Yang K., On the (Im)possibility of obfuscating programs. *CRYPTO'01 — Advances in Cryptology*, Lecture Notes in Computer Science, **2139**, 2001, p. 1–18.
3. Chow S., Gu Y., Johnson H., Zakharov V., An approach to the obfuscation of control flow of sequential computer programs. *Information Security Conference*, Lecture Notes in Computer Science, **2200**, 2001, p. 144–156.
4. Collberg C., Thomborson C., Low D., A taxonomy of obfuscating transformations, Tech. Report, N 148, Dept. of Computer Science, Univ. of Auckland, 1997.
5. Collberg C., Thomborson C., Low D., Manufacturing cheap, resilient and stealthy opaque constructs. *Symposium on Principles of Programming Languages*, 1998, p.184–196.
6. Collberg C., Thomborson C., Low D. Breaking abstraction and unstructuring data structures. *IEEE International Conference on Computer Languages*, 1998, p.28–38.
7. Collberg C., Thomborson C., Watermarking, tamper-proofing and obfuscation — tools for software protection. *IEEE Transactions on Software Engineering*, **28**, N 2, 2002, p. 735–746.
8. Devanbu P.T., Stubblebine S. Software engineering for security: a roadmap. *Future of SE Track*, 2000, 227–239.
9. Diffie W., Hellman M.E., New directions in cryptography. *IEEE Transactions in Information Theory*, **22**, 1976, p. 644–654.

10. Gollmann D. *Computer Security*. New York: Willey, 1999.
11. Goldreich O., Levin L.A., A hard-core predicate for all one-way functions. *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, 1989, p. 25–32.
12. Hada S., Zero-knowledge and code obfuscation. *ASIACRYPT'2000 — Advances in Cryptology*, 2000.
13. Horwitz S., Precise flow-insensitive may-alias analysis is NP-hard. *ACM Transactions on Programming Languages and Systems*, **19**, N 1, 1997, p. 1–6.
14. Landi W. Undecidability of static analysis. *ACM Letters on Programming Languages and Systems*, **1**, N 4, 1992, p. 323–337.
15. MacDonald J. On program security and obfuscation. Technical Report, University of California, 1998.
16. Menezes A.J., Van Oorschot P.C., Vanstone S.A., *Handbook of Applied Cryptography*. CRC Press, 1997.
17. Ramalingam G., The undecidability of aliasing. *ACM Transactions on Programming Languages and Systems*, **16**, N 5, 1994, p. 1467–1471.
18. Wang C., Hill J., Knight J. Davidson J., Software tamper resistance: obstructing static analysis of programs, Tech. Report, N 12, Dep. of Comp. Sci., Univ. of Virginia, 2000.
19. Wroblewski G., General method of program code obfuscation, in *Proceedings of the International Conference on Software Engineering Research and Practice (SERP)*, 2002, p. 153–159.