ISSN 0361-7688, Programming and Computer Software, 2015, Vol. 41, No. 6, pp. 361–372. © Pleiades Publishing, Ltd., 2015.
Original Russian Text © N.P. Varnovskiy, V.A. Zakharov, N.N. Kuzyurin, A.V. Shokurov, 2015, published in Programmirovanie, 2015, Vol. 41, No. 5.

RUBRIKA

The Current State of Art in Program Obfuscations: Definitions of Obfuscation Security

N. P. Varnovskiy[a], V. A. Zakharov[b], N. N. Kuzyurin[c], and A. V. Shokurov[c]

[a] *Moscow State University, Information Security Institute, Michurinskii pr. 1, Moscow, 119333 Russia*
[b] *Higher School of Economics, National Research University, ul. Myasnitskaya 20, Moscow, 101000 Russia*
[c] *Institute of System Programming, Russian Academy of Sciences, ul. Solzhenitsyna 25, Moscow, 109004 Russia*
e-mail: barnaba.np@gmail.com, zakh@cs.msu.su, nnkuz@ispras.ru, shok@ispras.ru

Received December 15, 2014

Abstract—Program obfuscation is a semantic-preserving transformation aimed at bringing a program into a form that impedes understanding of its algorithm and data structures or prevents extracting certain valuable information from the text of the program. Since obfuscation may find wide use in computer security, information hiding and cryptography, security requirements to program obfuscators have become a major focus of interest in the theory of software obfuscation starting from the pioneering works in this field. In this paper we give a survey of various definitions of obfuscation security and basic results that establish possibility or impossibility of secure program obfuscation under certain cryptographic assumptions.

DOI: 10.1134/S0361768815060079

1. INTRODUCTION

Obfuscation of a program is any transformation of the program that preserves the function computed by the program (equivalent transformation) but brings the program at a form that makes it difficult to extract key information from the program text about the algorithm and data structures implemented by the program. In contrast to refactoring, program obfuscation is aimed at impeding understanding of the program and preventing its modification. Hence, program obfuscation is one of the system programming problems similar to other program transformation problems, such as translation, optimization, refactoring, and parallelization. On the other hand, the obfuscation may be viewed as a special case of program encryption. Unlike encryption systems used for storing and transmitting data, the obfuscation does not suggest constructing efficient decryption algorithms, i.e., recovering the original program text; instead, it requires preserving semantics of the encrypted message—function computed by the obfuscated program. Therefore, the program obfuscation problem may be classified as that of cryptography or cryptoanalysis. It is the duality of this problem that explains the fact that the study of this problem during last 15 years proceeded in the frameworks of two different fields—system programming and cryptography. The goal of this paper is to acquaint the researchers working in the field of system programming with the results of studying the program obfuscation problem in the field of mathematical cryptography. This paper opens a series of works on mathematical aspects of the program obfuscation problem. In the first paper from this series, we consider and analyze definitions of the program obfuscation security concepts. In a subsequent work, we plan to study system programming and cryptography problems that can be efficiently solved by applying program obfuscation and evaluate how well modern obfuscation methods solve them. Finally, in the last paper of the series, we discuss the state of the art in the field of construction of secure program obfuscators.

The first mentioning of the obfuscation problem (without explicit use of the term "obfuscation") can be found in the seminal work by Diffie and Hellman [1] in 1976. They wanted to illustrate the concept of public key encryption and suggested the following simple scheme of its implementation. Given an arbitrary cryptosystem with a private key, the encryption procedure inserts the private key. Then, the encryption program initialized by this key is entangled such that the extraction of the private key from its text was an extremely difficult task. Thus, the modified encryption procedure becomes the public key of the new cryptosystem. The entangling of the encryption procedure aimed at preventing extraction of the private key from its text is one of possible applications of program obfuscation for solving some cryptography and computer security problems.

The term "program obfuscation" was first introduced in the work by Collberg, Thomborson, and Low [2] in 1997. The authors considered program obfusca-

tion, first of all, as a mean to protect intellectual property in algorithms implemented in open-source programs. They suggested simplest kinds of obfuscating program transformations, classified them, and established relationships of the program obfuscation problem with some known system programming problems.

The results of study of the program obfuscation problem during last 18 years can be briefly summarized as follows.

1. The class of problems that can be solved by means of program obfuscation algorithms is quite wide, and the goals of the obfuscation use can be opposite. The obfuscation can be used both to protect programs from virus attacks [3] and to mask computer viruses [4, 5]. When used in cryptography, the goal of masking is to hide data (private key) rather than algorithms. However, when obfuscation is used to ensure computer security, the purpose of masking is to hide algorithms rather than processed data. Thus, the program obfuscation problem includes a family of problems of program masking, each of which needs special requirements on the obfuscation security.

2. There is a large gap between theoretical requirements on program obfuscation security and techniques and means for solving this problem that are used in practice. There are many works where various practical obfuscation methods are proposed (see [6] for detail). Some of these methods were implemented in commercial software (see, for example, [7–12]). However, these developments are only slightly affected by fundamental theoretical results [13–16]: security requirements studied in the context of the cryptographic applications are either too strong or are not adequate to the software protection tasks arising in practice.

3. There is a large gap between negative and positive results of solving the obfuscation problem. It is proved in [13] that, for some strictly formalized definitions of the obfuscation security (security in the black-box model), there exist families of effectively computed functions that do not admit secure obfuscation. Secure obfuscation was obtained for essentially weaker security requirements and for only very simple—point and similar—functions [17–21]. Although the positive results were extended to wider function classes [22–24], the (im)possibility of efficient obfuscation for general cryptographic protocols, or for any significant class of programs (e.g., for finite state machines), under standard cryptographic assumptions has not been proved yet.

The discussion above shows that program obfuscation is a very complicated multifaceted problem, which hardly can be solved by means of a universal method. We hope that further progress in study of this problem will result in the formulation of mathematical foundations for creating a wide variety of formal concepts and methods of program obfuscation in various applications. The creation of such mathematical apparatus should begin with the development of various definitions of obfuscation security and study of the relationship between the suggested definitions and appropriate concepts and models of discrete mathematics, mathematical cryptography and complexity theory. This will make it possible to identify the most important properties inherent in program obfuscation of all kinds. The availability of a number of different definitions of obfuscation security will make it easier to understand the security requirements that one or another obfuscating transformation should satisfy. Finally, the incorporation of new formal requirements imposed on program obfuscation security opens new possibilities for adaptation of formal methods of theoretical informatics to solving program protection tasks.

The papers [1, 2] laid foundations for two directions of program obfuscation studies, namely, program obfuscation as a tool for solving certain cryptographic problems and program obfuscation as one of the methods for ensuring computer security. The main attention in this study is focused on the first direction. However, to make the picture complete, in the next section, we briefly present basic achievements in solving the program obfuscation problem and some similar problems in the field of system programming. Further, in Section 3, we consider the most frequently used formal definitions of program obfuscation security and relationships between these definitions.

2. PROGRAM OBFUSCATION FROM THE POINT OF VIEW OF SYSTEM PROGRAMMING

Program obfuscation could be very useful for solving many system programming and computer security problems. In early works devoted to program obfuscation, it was shown that obfuscating transformations could be used for protection of intellectual property, as a mean to prevent from recovery of original algorithms from open-source program codes [2, 25–27] and removal of watermarkings and fingerprintings [6, 8, 28–31], to protect software against attacks of malicious hosts (computer viruses), to ensure security of mobile agents in computer networks [32–36] and secure search in streaming data [37], and to protect databases [38] and VLSI designs in manufacturing of microelectronic circuits [39, 40]. The other side of the advantageous features of the obfuscation is that it can be used to impede detection of malicious hosts [4, 5, 41] and to hide vulnerabilities in computer security systems [42].

For the first time, the simplest techniques of program obfuscation were listed and systemized in [2, 43]. The authors of these works came to a number of conclusions that further determined several basic lines of research of the program obfuscation problem by methods and means of system programming and program analysis theories.

1. The purpose of program obfuscation is to resist algorithms of static and dynamic analysis of programs. Therefore, the obfuscation quality can be evaluated experimentally depending on the efficiency of application of program analysis tools to programs subjected to obfuscation. Thus, program obfuscation tools should be developed in close relationship with improving program deobfuscation means relying on the principle of shield and sword competition. The evolution of this competition, which was embodied in the annual competition of obfuscated programs, can be traced from a series of works [44–60].

2. Theoretical estimates of the obfuscation quality can be obtained by means of metric characteristics describing complexity of the program structure: an obfuscation of a program has to considerably increase these characteristics. Quality of obfuscating transformations based on program complexity metrics is analyzed in [61–65].

3. Since the goal of obfuscation is to resist static analysis algorithms, the security of obfuscating transformations can also be estimated with respect to complexity of program models that are used in algorithms of static analysis designed for revealing and removing consequences of program masking: the obfuscation is to be considered securer if more complicated program models are required for the deobfucscating static analysis. Security of obfuscating transformations with respect to static analysis methods was studied in [66–73]. These studies resulted, in particular, in the development of specialized algorithms for detection of polymorphic computer viruses, which use obfuscation procedures in the course of replication [74–76].

4. It was noted in [2] that the key element of many program obfuscation techniques is the so-called opaque predicates, the predicates the post-conditions of which are difficult to calculate by means of static analysis algorithms. Methods for constructing opaque predicates and estimating their security were studied in [29, 31, 77–79]. The authors of paper [2] also noted that security of obfuscating transformations can be improved through the use of computationally hard combinatorial problems by constructing on their base opaque predicates in such a way that the understanding of the behavior of such a predicate would be equivalent to solution of the problem. This expedient was used in works [25, 80–83] to demonstrate that, in some cases, the deobfuscation problem may turn out computationally hard.

5. Some variant of program obfuscation problem assumes that only a certain part of the program is open to the adversary for analysis and modifications, while the other program components run on a secure computer. Such a statement of the program protection problem was first discussed in paper [84], where a computation model consisting of a secure memory device and a public processor was considered. Successful solution of this problem resulted in the development of some hardware-based obfuscation methods [85, 86].

Results of theoretical and experimental studies of the program obfuscation problem were implemented in dozens of software tools for protecting programs by means of obfuscating transformations of one or another type. The majority of the suggested approaches are heuristics (some of them are quite sophisticated) designed to impede operation of programs for analysis of algorithms. The disadvantage of all these approaches is that they do not guarantee the desired security. Some of these obfuscation means successfully withstand automatic reverse engineering tools. However, the security of the existing obfuscation means turns out insufficient when dynamic analysis methods are employed and qualified experts in system programming are involved. Thus, currently available practical obfuscation methods and algorithms can impede (sometimes, significantly) understanding and modifications of a program but cannot be considered as protection means for classified information in a program code similar to encryption system.

3. PROGRAM OBFUSCATION FROM THE POINT OF VIEW OF MATHEMATICAL CRYPTOGRAPHY

Program obfuscation has important applications not only in system programming but also in cryptography. It was noted in early works [1, 13] that program obfuscation allows one to transform cryptosystems with a private key to those with a public key, with the open key being the obfuscated encryption procedure with the private key inserted in it. By means of program obfuscation, one can construct homomorphic encryption systems [13], functional encryption systems [87], trusted re-encryption schemes [23] and digital signatures; get rid of a random oracle when proving security of cryptographic protocols [2]; shuffle randomly encrypted messages in secret ballot schemes [89]; and create schemes of deniable encryption [90, 91] and one-sided functions with secrets [91]. These applications of program obfuscation will be discussed in more detail in our future paper. Note that, in order that the above-listed applications possess certain information security features, the obfuscations applied should also meet certain security requirements. Therefore, in the study of the program obfuscation problem from the mathematical cryptography standpoint, the security requirement is the most important one. In this section, we consider various definitions of the obfuscation security known from the literature and present the most important results on possibility or impossibility of construction of secure obfuscating transformations.

Definitions of the program obfuscation security will be classified in accordance with the commonly accepted scheme “threat–attack.” Under the threat, we mean the goal the adversary aims to achieve. This

may be an attempt to find out one or another feature of the obfuscated program or to obtain information about algorithms and constants used in the program, and the like. The attack is a model of the adversary, which includes its computational resources and data available to the adversary.

In the literature on the complexity theory and mathematical cryptography, two commonly accepted formalizations of the concept of a "computational program" are used. According to one of them, programs are represented as Turing machines, while the other formalization represents a program as a logical circuit. When the computation complexity and program sizes are estimated up to polynomial transformations, these two definitions are equivalent [93]. For the sake of uniformity, we confine our consideration to the former case, i.e., representations in the form of Turing machines.

The adversary models in cryptography are most often represented by probabilistic algorithms, which use random variables (random numbers generators, random binary strings, etc.). The computations carried out by the adversary must terminate in time polynomial in size of the data available to the adversary. Thus, the adversary is also modeled by a Turing machine (TM); however, this machine is probabilistic and bounded by a polynomial time. The machines of this class are denoted as PPT. In the complexity theory, along with ordinary and probabilistic Turing machines, Turing machines with an oracle are used. An oracle may be any function or predicate, which are considered to be auxiliary computation means. A TM with an oracle F may appeal to the oracle to compute a function value $F(y)$ for an arbitrary string y written on a special oracle tape. This value is computed in one step independent of the complexity of the function F.

For a TM π and a binary string x used as the input data for the TM, we will use the notation $|\pi|$ to denote the size of the TM π, notation $\pi(x)$ to denote the result of computation of the TM π on the input data x, and notation $time(\pi(x))$ to denote the computation time (the number of steps) of the TM π on the input data x. TMs π_1 and π_2 are considered to be equivalent if $\pi_1(x) = \pi_2(x)$ for any input data x. A function $\nu: N \to [0, 1]$ is considered to be negligibly small if it decreases faster than any function inverse to a polynomial; i.e., for any $k \in N$, there exists a $n_0 \in N$ such that, for all $n > n_0$, the inequality $\nu(n) < 1/n^k$ holds. An arbitrary polynomial and an arbitrary negligibly small function are conventionally denoted as $poly(\cdot)$ and $neg(\cdot)$, respectively.

3.1. Program Obfuscation in the Black-box Model

A formal definition of program obfuscation was first formulated in paper [13], which relied on results of the earlier work [92]. An obfuscation of a program is to meet three basic requirements: preservation of program functionality, insignificant change of program size and performance, and the security requirement. A program obfuscation is said to be secure in the black-box model if the adversary with unlimited access to the text of the obfuscated program is capable of extracting from this text only that part of information about the original program that can be obtained based on only test experiments with the program without access to its text (the so-called black-box testing strategy). The threat here is extraction of any information about the obfuscated program except that about the function implemented by this program. The formal mathematical definition of the above-specified requirements is as follows.

Definition 1 (black-box model) ([13]). An obfuscator in the black-box model for a TM family is a PPT O satisfying the following three conditions:

1. **Functional equivalence.** For any TM π, $\pi \in M$, the application of the obfuscator O to π results in a TM that is functionally equivalent to the original machine π.

2. **Polynomial costs.** For any TM π, $\pi \in M$, the size and performance of any TM $O(\pi)$ differ from the size and performance of the TM π polynomially at most; i.e., $|O(\pi)| = poly(|\pi|)$ and $time(O(\pi(x))) = poly(time((\pi(x))))$.

3. **Virtual black-box property.** For any PPT A (adversary), there exists a PPT S (simulator) such that the relation

$$|\Pr[A(O(\pi)) = 1] - \Pr[S^{\pi}(1^{|\pi|}) = 1]| \leqslant neg(|\pi|)$$

holds for any TM π, $\pi \in M$ (the first probability depends on random variables used by the obfuscator O and adversary A, and the second probability depends on random variables used by the simulator S).

One of the basic results of the work [13] is formulated in the following theorem.

Theorem 1 ([13]). There are TM families for which obfuscation in the black-box model cannot be constructed.

The idea of the proof is simple. Consider two TMs $\pi_{a,b}$ and $\pi_{c,d}$. The first TM, given an input string x, outputs string b if $x = 1$ and zero string otherwise. The second TM considers the input string x as a description of some program, applies this program to string c and outputs 1 if this computation terminates in a polynomial time (with respect to length of x) with result d and 0 otherwise. These two TMs can be combined into a TM $\pi_{c,d}$ that carries out computation like TM $\pi_{a,b,c,d}(x, y)$ if $y = 0$ and like TM $\pi_{a,b}(x)$ otherwise. The goal (threat) of the adversary is to find out whether it is true that $a = c$ and $b = d$ for the given TM $\pi_{c,d}$. This goal is easily achieved by a deterministic TM A that, having received an obfuscated program $\pi' = O(\pi_{a,b,c,d})$, substitutes 0 or 1 for the second component of the input data and, then, computes the value $\pi'(\pi'(\cdot, 0), 1)$. Clearly, the equalities $a = c$ and $b = d$ hold if and only if that value calculated by the adversary A is equal to 1. At the same time, any simulator S that runs in a polynomial time and has only oracle access to the program $\pi_{a,b,c,d}$ can cor-

rectly check whether the conditions $a = c$ and $b = d$ hold only if it guess right by chance appropriate queries to the oracle, the probability of which is negligibly small.

The authors of paper [13] managed to construct *hereditary unobfuscatable family of functions*, i.e., a set of functions such that any program of computation of any function from this set does not admit obfuscation in the black-box model. It is interestingly to note that the class of hereditary unobfuscatable families of functions includes some families of pseudorandom functions, encryption functions with a private key, and schemes of digital signature. In addition, it was shown in [13] that the obfuscation in the black-box model is impossible for certain families of programs that can be represented by logical schemes from class TC_0, which are constructed from threshold elements and have depth bounded by a constant.

On the whole, results of paper [13] show that the program obfuscation problem has no simple solution: there exist families of programs computing relatively simple functions for which a secure obfuscation in the black-box model cannot be constructed. Due to this, further studies of the obfuscation problem were focused on finding answers to the following questions. Are there exist other definitions of the obfuscation security that impose less restrictive requirements on the security compared to the virtual black box model and still make it possible finding appropriate cryptographic applications for the obfuscation? For what classes of programs, it is possible to construct obfuscators satisfying one or another reasonable security requirement? What necessary and sufficient conditions a program must meet in order to be unobfuscatable.

3.2. Variations of the Black-box Model

From the very definition of the obfuscation in the black-box model, it can be seen that it admits certain modifications if we change threats and attacks, i.e., constraints imposed on the adversary A and simulator S.

For example, we may assume that computational capabilities of the adversary are unbounded. Then, the class of programs admitting obfuscation can be completely described. A family of programs M is called effectively learnable [94] if there exists a PPT L such that, for any program π, $\pi \in M$, the TM L^{π} outputs a program π' that is equivalent to the program π. It was established in [20] that a family of programs admits obfuscation in the black-box model with an unbounded adversary if and only if this family of programs is effectively learnable.

One can consider a variant of Definition 1 where computational capabilities of the simulator are unbounded, with the number of accesses of the simulator to the oracle being bounded by some polynomial in the size of the obfuscated program π. This kind of obfuscation was introduced in paper [95]; we will call it obfuscation with weakly bounded simulator. It was shown in that work that the obfuscation security requirements in the model with weakly bounded simulator are considerably weaker than those in the black-box model.

Theorem 2 ([95]). If there exist one-sided permutations, then there exists a TM family that admits obfuscation in the model with weakly bounded simulator and does not admit obfuscation in the virtual black-box model.

The proof of the theorem is based on the same idea as that of Theorem 1.

The authors of paper [95] managed to prove existence of TM families for which obfuscation in the model with weakly bounded simulator cannot be constructed.

The capabilities of the simulator can also be strengthened in another way. Since the adversary with the access to the obfuscated program can observe not only input–output pairs but also computation traces, it makes sense to check whether it is possible to obtain an obfuscated program whose computation traces would be the only source of useful information for the adversary carrying out experiments with the program.

To achieve this goal, we modify the definition of obfuscation in the black-box model. The new obfuscation, which was introduced and studied in [14], is called obfuscation in the gray-box model. The gray-box model differs from the black-box model in two aspects.

First, in response to a query x, the oracle of the simulator outputs not only the result of computation of the program $\pi(x)$ being obfuscated but also the trace of computation of program π for input data x. Thus, in this case, it is important to know which exactly program among the equivalent ones was selected to be the oracle. We require that such a program be the original program π. This means that the obfuscator is not obliged to hide any properties of program π that can efficiently be established based on computation traces of the TM π.

Second, instead of terminating programs, we consider reactive ones. Unlike the terminating programs, which are designed for calculating relations between the input and output data, the reactive programs perform transformations of streams of events–queries coming to the program input into a stream of responses–reactions to these events produced in the program output. Thus, the reactive program computes a function mapping an infinite sequence of input data $x_1, x_2, \ldots, x_n$ (a sequence of queries) into an infinite sequence of outputs $y_1, y_2, \ldots, y_n$ (a sequence of responses) such that each output y_n depends on only inputs $x_1, x_2, \ldots, x_n$. Examples of the reactive programs are network protocols (including cryptographic ones), imbedded systems, operating systems, and the like. The reactive program can formally be defined as an

ordinary TM that uses input and output tapes, as well as some auxiliary tapes, such that a current input word x_{n+1} is not read until the output word y_n is completely written. To distinguish such machines from the ordinary TMs, we denote the former as RTMs.

To distinguish the oracle used in the definition of the obfuscator in the black-box model (Definition 1) from the oracle used in the new definition, we denote the latter as $Tr(\pi)$. In response to a current query x_n, the oracle $Tr(\pi)$ outputs a pair $\langle y_n, tr(x_n)\rangle$, where y_n is a result of work of machine π given the input x_n (note that y_n depends not only on x_n but also on all previous inputs used earlier as queries to the oracle) and $tr(x_n)$ is the trace of execution of $tr(x_n)$ for this input.

Definition 2 (gray-box model) [14]. An obfuscator in the gray-box model for the RTM family M is a PPT O that satisfies conditions of functional equivalence and polynomial costs, as well as the following security requirement:

Virtual gray box property. For any PPT A (adversary), there exists a PPT S (simulator) such that the relation

$$|\Pr[A(O(\pi)) = 1] - \Pr[S^{Tr(\pi)}(1^{|\pi|}) = 1] \leqslant neg(|\pi|)$$

holds for any RTM π, $\pi \in M$ (the first probability depends on random variables used by the obfuscator O and adversary A, and the second probability depends on random variables used by the simulator S).

Theorem 3 ([14]). If there exist one-sided functions, then there exist RTM families the obfuscation of which in the gray-box model is impossible.

One more definition of obfuscation similar to that in the black-box model was proposed in [96]. In practice, the adversary may analyze an obfuscated program having at hand some additional data, which may have or have no relation to this program. For example, the adversary may have, along with the obfuscated program π, a simplified demo π' of this program. Then, the black-box model may be modified.

Definition 3 (black-box model with auxiliary input) [96]. An obfuscator in the black-box model with auxiliary input for a TM family M is a PPT O that satisfies conditions of functional equivalence and polynomial costs, as well as the following condition:

Property of a virtual black box with auxiliary input. For any PPT A (adversary), there exists a PPT S (simulator) such that the relation

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

holds for any TM π, $\pi \in M$, and a binary string z (the first probability depends on r random variables used by the obfuscator O and adversary A, and the second probability depends on for random variables used by the simulator S).

In this definition, the additional input data z may depend or not depend on the obfuscated program. Therefore, there are two variants of obfuscation with auxiliary input; for both of them, the relationship with above-introduced kinds of obfuscation was established.

Theorem 4 ([95, 96]). If there exists an obfuscator in the black-box model for a TM family, then, for the same TM family, there exists an obfuscator with auxiliary input not depending on the obfuscated program. If, for a TM family M, there exists an obfuscator in the black-box model with weakly bounded simulator, then, for the same TM family, there exists an obfuscator with auxiliary input depending on the obfuscated program.

Paper [96] describes also a class of logical circuits—circuits with superpolynomial entropy calculating functions with NP-complete filter—for which it is impossible to construct obfuscators with dependent or independent auxiliary input. The authors of that paper showed, in particular, that the class of circuits with superpolynomial entropy includes any family of logical circuits computing pseudorandom functions, as well as any family of logical circuits implementing cryptosystems that use pseudorandom functions.

The results obtained demonstrate that simple weakening of the black-box obfuscation security requirement does not imply considerable simplification of the program obfuscation problem. Moreover, it turned out that, for many cryptographic functions, no program of their computation can completely be protected by means of obfuscation. Therefore, it makes sense to consider other kinds of program obfuscation that are less restrictive than the obfuscation in the black-box model.

3.3. Algorithm Obfuscation

The most obvious application of obfuscation for protection of software suggests the following scenario. Suppose that somebody invented a fast algorithm for solving a complex and practically important problem. The purpose of obfuscation in this case is to find a transformation of the program implementing this algorithm that would not allow one to easily extract the algorithm from the obfuscated program. The black-box obfuscation is not suitable in this case, since it is unlikely that someone would like to buy a black box as a part of software. Moreover, any software product on the market must have a manual with a description of the product functionality. If the function computed by the program is known, the purpose of the obfuscation is to imped understanding of the original features of the algorithm implemented by the program. Such obfuscation better suits the goal of algorithm protection than the total black-box obfuscation. The simplest way to formalize the concept of algorithm obfuscation is to provide the adversary with some (arbitrary) program π_0 that computes the same function as the obfuscated program π. Thus, we arrive

at several definitions of an algorithm obfuscator, which were suggested in works [13, 14, 16].

Definition 4 (algorithm obfuscation) [14]. An algorithm obfuscator for a TM family M is a PPT O that satisfies conditions of functional equivalence and polynomial costs, as well as the following security requirement:

Property of algorithm suppression. For any PPT A (adversary), there exists a PPT S (simulator) such that the relation

$$|\Pr[A(O(\pi), \pi_0) = 1] - \Pr[S(1^{|\pi|}, \pi_0) = 1]| \leqslant neg(|\pi|)$$

holds for any pair of equivalent TMs π, π_0 from the class M satisfying the condition $|\pi_0| = poly(|\pi|)$ (the first probability depends on random variables used by the obfuscator O and adversary A, and the second probability depends on random variables used by the simulator S).

In addition to the above definition of the algorithm obfuscator, two similar alternative definitions of obfuscation for algorithm hiding were proposed in [13, 16]. These definitions introduce indistinguishable obfuscation and the best possible obfuscation and are based on the concept of computationally indistinguishable distributions of random variables.

Definition 5 (indistinguishable obfuscation) [13]. An indistinguishable obfuscator for a TM family M is a PPT O that satisfies conditions of functional equivalence and polynomial costs, as well as the following condition:

Property of effective program indistinguishability. For any pair of equivalent TMs π_1 and π_2 of the same size from the class M, probability distributions of random variables $O(\pi_1)$ and $O(\pi_2)$ are computationally indistinguishable; i.e., for any PPT D, the following relation holds:

$$|\Pr[D(O(\pi_1)) = 1] - \Pr[D(O(\pi_2)) = 1]| \leqslant neg(|\pi|).$$

It is not difficult to see that the property of program indistinguishability follows from the virtual black box property; however, the example used in Theorem 1 for proving impossibility of construction of an obfuscator in the black-box model is not applicable in the case of indistinguishable obfuscation. The disadvantage of Definition 5 is that it does not give us intuitively clear guarantee that the obfuscated program does mask the implemented algorithm. Therefore, an alternative definition based on indistinguishability of the obfuscated program from any equivalent one was suggested in [16].

Definition 6 (the best possible obfuscation) [16]. The best possible obfuscator for a TM family M is a PPT O that satisfies conditions of functional equivalence and polynomial costs, as well as the following condition:

Property of constrained effective learnability. For any PPT L (learner), there exists a PPT S (simulator) such that, for any pair of equivalent TMs π_1 and π_2 of the same size from the class M, probability distributions of random variables $L(O(\pi_1))$ and $S(\pi_2)$ are computationally indistinguishable.

The property of constrained learnability suggests that the adversary cannot extract from the text of an obfuscated program more useful information than could be extracted from the text of any equivalent program.

In Definitions 5 and 6, the conditions of computational indistinguishability of the probability distributions can be strengthened if we require that the probability distributions under consideration coincide (absolute indistinguishability) or statistical distance between them be not greater than a prescribed constant (statistical indistinguishability). Although the relationship between Definitions 1, 5, and 6 is not clear, the following theorem is valid.

Theorem 5 ([16, 95]). The three kinds of program obfuscation— indistinguishable obfuscation, the best possible obfuscation, and obfuscation in the black-box model with a simulator with unlimited computational capabilities—are equivalent.

It can be seen from this theorem that obfuscation of any class of programs in the black-box model implies the best possible obfuscation for the same class of programs. Therefore, the requirement of the efficiency of simulator S in Definition 6 is redundant. Moreover, obfuscation of algorithms for any class of programs implies the best possible obfuscation for the same class of programs. However, the question of whether the inverse inclusion is valid, as well as the question of existence of a family of programs for which the best possible obfuscation does not exist, is still open. The next theorem was proved in [16].

Theorem 6 ([16]). If a family of programs represented in a 3-CNF form has the best possible (in the sense of statistical indistinguishability) obfuscation, than the polynomial hierarchy of complexity classes collapses on the second level.

Currently, the question of structure of polynomial hierarchy remains open, although the majority of experts in computer science believe that it has infinitely many levels.

Recent studies show that, most likely, it is Definition 6 that can be used for constructing obfuscators the security of which can be proved in the framework of certain commonly accepted in cryptography assumptions on difficulty of solving some mathematical problems by means of homomorphic encryption systems.

The important particular case of obfuscation of algorithms is obfuscation of constants. Let a parameterized TM family $M(C) = \{\pi(c) \mid c \in C\}$ be given. One TM differs from another only by a value of one secrete parameter used as a constant. Such a parameter could be, for example, a private key in an encryption procedure or in a digital signature. The algorithm implemented by programs from family $M(C)$ is not a secrete one and does not need hiding.

In the papers on program obfuscation, several definitions of parameterized programs have been suggested. One of them is a simple adaptation of the definition of the obfuscator in the black-box model

Definition 7 (obfuscation of constants in the black-box model) [14, 21]. An obfuscator of constants for a TM family $M(C)$ is a PPT O that satisfies conditions of functional equivalence and polynomial costs, as well as the following security requirement:

Property of suppression of constants. For any PPT A (adversary), there exists a PPT S (simulator) such that the relation

$$|\Pr[A(O(\pi(c_0)), \pi(c)) = 1] - \Pr[S^{\pi(c_0)}(1^{|\pi(c_0)|}, \pi(c)) = 1]| \leqslant neg(|\pi|)$$

holds for any pair of TMs $\pi(c)$, $\pi(c_0)$ from the class $M(C)$ satisfying the condition $|\pi(c_0)| = poly(|\pi(c)|)$ (the first probability depends on random variables used by the obfuscator O and adversary A, and the second probability depends on random variables used by the simulator S).

The existence of a program for which obfuscation in the black-box model cannot be constructed implies that, for some families of parameterized programs, obfuscation of constants cannot be done as well.

One more definition is just a modification of the definition of an indistinguishable obfuscator. Let a probability distribution be given on a set of constants.

Definition 8 (indistinguishable obfuscation of constants) ([21]). An obfuscator of constants for a TM family $M(C)$ is a PPT pair (G, O) satisfying the following three conditions:

1. **Functional equivalence.** For any constant c, $c \in C$, the output $G(c)$ is a constant d, $d \in D$, and the output $O(G(c), c)$ is a TM π' such that the TM $\pi(c)$ is equivalent to the TM $\pi'(d)$.

2. **Polynomial costs.** For any constant c, $c \in C$, the size of any constant $G(c)$ differs from the size of constant c polynomially at most, and the size and time of execution of any TM $O(\pi)$ differs from the size of TM π polynomially at most.

3. **Property of constant indistinguishability.** For any PPT D (distinguisher), there exists a PPT S (simulator) such that the relation

$$|\Pr[D^{\pi(c)}(1^{|c|}, G(c)) = 1] - \Pr[D^{\pi(c)}(1^{|c|}, S^{\pi(c)}(1^{|c|})) = 1]| \leqslant neg(|\pi|),$$

holds, where the constant c is selected in accordance with the above-mentioned probability distribution and both probabilities on the right-hand side of the relation depend on random variables used by the obfuscator O, distinguisher D, and simulator S.

Here, the very constant is subjected to obfuscation, and the program that uses it is only modified to meet the functional equivalence condition. This definition of obfuscation of constants was introduced specially for solving tasks of transformation of cryptosystems with private key to those with public key. However, there exist programs that cannot be obfuscated in the sense of Definition 8.

Theorem 7 ([21]). A family of programs $M(C)$ implementing a pseudorandom function cannot be obfuscated in the sense of Definition 8.

3.4. Obfuscation of Predicates

The series of formal definitions of program obfuscation ends with the definition of obfuscation of predicates. This obfuscation is designed for hiding a specified feature of programs. Let a predicate P be given on a TM set M.

Definition 9 (obfuscation of predicates) [14]. An obfuscator of predicate P for a TM family M is a PPT O that satisfies conditions of functional equivalence and polynomial costs, as well as the following security requirement:

Property of predicate suppression. For any PPT A (adversary), there exists a PPT S (simulator) such that the relation

$$|\Pr[A(O(\pi)) = P(\pi)] - \Pr[S^{\pi}(1^{|\pi(c_0)|}) = P(\pi)]| \leqslant neg(|\pi|)$$

holds for any TM π from the class M, with the first probability depending on random variables used by the obfuscator O and adversary A and the second probability depending on random variables used by the simulator S.

Comparing Definitions 1 and 8, one can see that program obfuscation in the black-box model has to hide all possible predicates on a given family of programs, whereas Definition 8 admits construction of different obfuscators for different predicates. The existence of predicates that cannot be obfuscated for some families of programs follows from Theorem 1. At the same time, there exist classes of programs for which some predicates can efficiently be obfuscated.

4. CONCLUSIONS

As can be seen from the presented ontology of definitions of program obfuscation, the obfuscators can conditionally be divided into two classes: "strong" obfuscators (Definitions 1–3) designed to mask unlearnable program features and "weak" obfuscators (Definitions 4–8) designed for hiding only some program features. For every definition of obfuscation, it is proved that there exist representative classes of functions (including functions used in cryptography) that cannot be implemented by the obfuscated programs. Mathematical technique developed in works [13, 14, 16, 20–22, 96] makes it possible to construct examples of this kind for any "reasonable" obfuscation definition. Much more difficult task is to prove that some practically significant programs and functions can be obfuscated.

Achievements in cryptography during last five years forced the researchers to revise the above definitions of program obfuscation security. In 2009, Gentry [97, 98] managed to construct (using earlier known assumptions of difficulty of solving some algebraic problems) secure systems of fully homomorphic encryption (see, also, [99–101]). Such a cryptosystem allows one to carry out any computations on encrypted data. Having at hand a public key pk, one can encode input data x and program π. Then, the cyphertexts $Enc(x, pk)$ and $Enc(\pi, pk)$ obtained can be fed to the input of a universal procedure, which will compute the cyphertext of the result of execution of program π on the above-specified input data: $Enc(\pi, pk) = Eval(Enc(\pi, pk), Enc(x, pk))$. Further, having the secrete key sk, one can decode the cyphertext and obtain the result $\pi(x) = Dec(Enc(\pi, pk), sk)$. As can be seen from the above brief description, this scheme of computation on encrypted data can be used for obfuscation of programs as well, under the condition that there is an appropriate way to mask the decryption procedure. The decryption algorithm itself does not need protection; only one parameter used by the algorithm—secrete key sk—is to be protected. Thus, in order to construct secure program obfuscation—it is sufficient to ensure secure obfuscation of constants in a special family of programs. Efforts of several groups of leading researchers in the field of mathematical cryptography have been focused on solving this problem during last three years [87, 102–104].

REFERENCES

1. Diffie, W. and Hellman, M., New directions in cryptography, *IEEE Trans. Inf. Theory*, 1976, vol. **22**, no. 6, pp. 644–654.
2. Collberg, C., Thomborson, C., and Low, D., A taxonomy of obfuscating transformations, *Tech. Report*, no. 148, Dept. of Computer Science, Univ. of Auckland, 1997.
3. Cohen, F., Operating system protection through program evolution, *Comput. Security*, 1993, vol. **12**, no. 6, pp. 565–584.
4. Chess, D. and White, S., An undetectable computer virus, *Proc. of the 2000 Virus Bulletin Conf.*, Orlando, 2000.
5. Szor, P. and Ferrie, P., Hunting for metamorphic, *Proc. of the 2001 Virus Bulletin Conf.*, 2001, pp. 123–144.
6. Collberg, C. and Nagra, J., *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Program Protection*, Addison-Wesley, 2009.
7. Aucsmith, D., Tamper resistant software: An implementation, *Lect. Notes Comput. Sci.*, 1996, vol. **1174**, pp. 317–333.
8. Scud, T.T., ObjObf—x86/Linux ELF relocateable object obfuscator. http://packetstormsecurity.org/files/31524/objobf-0.5.0.tar.bz2.
9. Solutions, P., DashO—the premier Java obfuscator and efficiency enhancing tool. http://www.preemptive.com/products/dasho/.
10. Solutions, P., Dotfuscator—the premier. NET obfuscator and efficiency enhancing tool. http://www.preemptive.com/products/dotfuscator/.
11. KlassMaster, Z., The second generation Java obfuscator. http://www.zelix.com/.
12. Ge, J., Chaudhuri, S., and Tyagi, A., Control flow based obfuscation, *Proc. of the Digital Rights Management Workshop*, Alexandria, VA, 2005, pp. 83–92.
13. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., and Ke Yang, On the (im)possibility of obfuscating programs, *Lect. Notes Comput. Sci.*, 2001, vol. **2139**, pp. 1–18.
14. Varnovsky, N.P., A note on the concept of obfuscation, *Tr. Inst. Sistemnogo Program. Ross. Akad. Nauk*, 2004, no. 6. pp. 127–137.
15. Kuzurin, N.N., Shokurov, A.V., Varnovsky, N.P., and Zakharov, V.A., On the concept of software obfuscation in computer security, *Lect. Notes Comput. Sci.*, 2007, vol. **4779**, pp. 281–298.
16. Goldwasser, S. and Rothblum, G.N., On best possible obfuscation, *Lecture Notes Comput. Sci.*, 2007, vol. **4392**. pp. 194–213.
17. Canetti, R., Towards realizing random oracles: hash functions that hide all partial information, *Lect. Notes Comput. Sci.*, 1997, vol. **1294**, pp. 455–469.
18. Varnovsky, N.P. and Zakharov V.A. On the possibility of provably secure obfuscating programs, *Lect. Notes Comput. Sci.*, 2004, vol. **2890**. pp. 91–102.
19. Lynn, B., Prabhakaran, M., and Sahai, A., Positive results and techniques for obfuscation, *Lect. Notes Comput. Sci.*, 2004, vol. **3027**, pp. 20–39.
20. Wee, H., On obfuscating point functions, *Proc. of the 37th Symp. on Theory of Computing*, 2005, pp. 523–532.
21. Hofheinz, D., Malone-Lee, J., and Stam, M., Obfuscation for cryptographic purpose, *Lect. Notes Comput. Sci.*, 2007, vol. **4392**, pp. 214–232.
22. Canetti, R. and Dakdouk, R.R., Obfuscating point functions with multibit output, *Lect. Notes Comput. Sci.*, 2008, vol. **4965**, pp. 489–508.
23. Hohenberger, S., Rothblum, G.N., Shelat, A., and Vaikuntanathan, V., Securely obfuscating re-encryption, *Proc. of the 4th Conf. on Theory of Cryptography*, 2007, pp. 233–252.
24. Canetti, R., Rothblum, G.N., and Varia, M., Obfuscation of hyperplane membership, *Proc. of the 7th Conf. on Theory of Cryptography*, 2010, pp. 72–89.
25. Collberg, C., Thomborson, C., and Low, D., Manufacturing cheap, resilient and stealthy opaque constructs, *Proc. of the Symp. on Principles of Programming Languages*, 1998, pp. 184–196.
26. de Oor, A. and van der Oord L., Stealthy obfuscation techniques: Misleading pirates, *Tech. report of Dept. of Computer Science Univ. of Twente Enschede*, 2003.
27. Naumovich, G, and Memon, N., Preventing piracy, reverse engineering, and tampering, *IEEE Comput.*, 2003, vol. **36**, no. 7, pp. 64–71.
28. Collberg, C. and Thomborson, C., Watermarking, tamper-proofing, and obfuscation—tools for software protection, *IEEE Trans. Software Eng.*, 2002, vol. **28**, no. 6, pp. 735–746.

29. Arboit, G., A method for watermarking Java programs via opaque predicates, *Proc. of the Int. Conf. on Electronic Commerce Research (ICECR-5)*, Montreal, 2002, pp. 1–8.
30. Zhu, W., Thomborson, C., and Wang, F.-Y., A survey of software watermarking, *Lect. Notes Comput. Sci.*, 2005, vol. **3495**, pp. 454–458.
31. Myles, G. and Collberg, C., Software watermarking via opaque predicates: Implementation, analysis, and attacks, *Electronic Commerce Research*, 2006, vol. **6**, no. 2, pp. 155–171.
32. Sander, T. and Tchudin, C.F., Protecting mobile agents against malicious hosts, *Lect. Notes Comput. Sci.*, 1997, pp. 44–60.
33. Hohl, F., Time limited blackbox security: Protecting mobile agents from malicious hosts, *Lect. Notes Comput. Sci.*, 1998, vol. **1419**, pp. 92–113.
34. D'Anna, L., Matt, B., Reisse, A., van Vleck, T., Schwab, S., and LeBlanc, P., Self-protecting mobile agents obfuscation report, *Tech. report no. 03-015*, Network Associates Laboratories, 2003.
35. Wu, J., Zhang, Y., Wang, X., *et al.*, A scheme for protecting mobile agents based on combining obfuscated control flow and time checking technology, *Proc. of the Conf. on Computational Intelligence and Security*, Harbin, 2007, pp. 912–916.
36. Roeder, T. and Schneider, F.B., Proactive obfuscation, *ACM Trans. Comput. Syst.*, 2010, vol. **28**, no. 2.
37. Ostrovsky, R. and Skeith, W.E., Private searching on streaming data, *Lect. Notes Comput. Sci.*, 2005, vol. **3621**, pp. 223–240.
38. Narayanan, A. and Shmatikov, V., Obfuscated databases and group privacy, *Proc. of the 12th ACM Conf. on Computer and Communications Security*, 2005, pp. 102–111.
39. Ivannikov, V.P., Varnovsky, N.P., Zakharov, V.A., Kuzyurin, N.N., Shokurov, A.V., Kononov, A.N., and Kalinin, A.V., Methods of information protection of project solutions in manufacturing of microelectronic circuits, *Izv. Taganrogskogo Radiotekhnicheskogo Univ.*, 2005, vol. **4**, pp. 112–119.
40. Varnovsky, N.P., Zakharov, V.A., Kuzyurin, N.N., Cherov, A.V., and Shokurov, A.V., Problems and methods for ensuring information security in manufacturing of microelectronic circuits, *Tr. Inst. Sistemnogo Program. Ross. Akad. Nauk*, 2006, vol. **11**, pp. 29–61.
41. Borello, J.M. and Me, L., Code obfuscation technique for metamorphic viruses, *J. Comput. Virology*, 2008, vol. **4**. pp. 211–220.
42. Bhatkar, S., Du Varney, D.C., and Sekar, R., Efficient techniques for comprehensive protection from memory error exploits, *Proc. of the 14th Conf. on USENIX Security Symp.*, 2005, vol. 14, p. 17.
43. Wroblewski, G., General method of program code obfuscation, *Proc. Int. Conf. on Software Engineering Research and Practice*, 2002.
44. Linn, C. and Debray, S., Obfuscation of executable code to improve resistance to static disassembly, *Proc. of the 10th ACM Conf. on Computer and Communication Security*, 2003, pp. 290–299.
45. Sosonkin, M, Naumovich, G, and Memon, N., Obfuscation of design intent in object-oriented applications, *Proc. of the Digital Rights Management Workshop*, Washington, DC, 2003, pp. 142–153.
46. Collberg, C., Myles, G., and Huntwort, A., Sandmark—a tool for software protection research, *IEEE Security Privacy*, 2003, vol. **1**, no. 4, pp. 40–49.
47. Heffner, K. and Collberg, C., The obfuscation executive, *Lect. Notes Comput. Sci.*, 2004, vol. **3225**, pp. 428–440.
48. Chan, J.T. and Yang, W., Advanced obfuscation techniques for Java bytecode, *J. Syst. Software*, 2004, vol. **71**, nos. 1–2, pp. 1–10.
49. Cimato, S., De, S.A., and Petrillo, U.F., Overcoming the obfuscation of Java programs by identifier renaming, *J. Systems Software*, 2005, vol. **78**, no. 1, pp. 60–72.
50. Madou, M., Anckaert, B., de Sutter, B., and de Bosschere, K., Hybrid static-dynamic attacks against software protection mechanisms, *Proc. of the 5th ACM Workshop on Digital Rights Management*, 2005, pp. 75–82.
51. Udupa, S.K., Debray, S.K., and Madou, M., Deobfuscation: Reverse engineering obfuscated code, *Proc. of the 12th Working Conf. on Reverse Engineering*, Pittsburgh, 2005, pp. 45–54.
52. Ge, J., Chaudhuri, S., and Tyagi, A., Control flow based obfuscation, *Proc. of the Digital Rights Management Workshop*, Alexandria, VA, 2005, pp. 83–92.
53. Chen, K. and Chen, J.B., On instrumenting obfuscated Java bytecode with aspects, *Proc. of the 2006 Int. Workshop on Software Engineering for Secure Systems*, Shanghai, 2006, pp. 19–26.
54. Madou, M., Anckaert, B., de Sutter, B., de Bosschere, K., Cappaert, J., and Preenel, B., On the effectiveness of source code transformations for binary obfuscation, *Proc. Int. Conf. on Software Engineering Research and Practice*, 2006, pp. 527–533.
55. Madou, M., Anckaert, B., Moseley, P., Debray, S., de Sutter, B., and de Bosschere, K., Software protection through dynamic code mutation, *Proc. of the 6th Int. Conf. on Information Security Applications*, 2006, pp. 194–206.
56. Drape, S., Majumdar, A., and Thomborson, C., Slicing aided design of obfuscating transforms, *Proc. of the Int. Computing and Information Systems Conf. (ICIS 2007)*, Melbourne, 2007, pp. 1019–1024.
57. Majumdar, A., Drape, S., and Thomborson, C., Slicing obfuscations: Design, correctness, and evaluation, *Proc. of the 2007 ACM Workshop on Digital Rights Management*, Alexandria, 2007, pp. 70–81.
58. Batchelder, M. and Hendren, L., Obfuscating Java: The most pain for the least gain, *Proc. of the Compiler Construction*, Braga, Portugal, 2007, pp. 96–110.
59. Ceccato, M., Di, P.M., Nagra, J., *et al.*, Towards experimental evaluation of code obfuscation techniques, *Proc. of the 4th ACM Workshop on Quality of Protection*, Alexandria, 2008, pp. 39–46.
60. Darwish, S.M., Guirguis, S.K., and Zalat, M.S., Stealthy code obfuscation technique for software security, *Proc. of the Int. Conf. on Computer Engineering and Systems*, 2010, pp. 93–99.
61. Chernov, A.V., A method of program masking, *Tr. Inst. Sistemnogo Program. Ross. Akad. Nauk*, 2003, vol. **4**, pp. 85–119.

62. Majumdar, A., Drape, S., Thomborson, C., *et al.*, Metrics-based evaluation of slicing obfuscations, *Proc. of the 3d Int. Symp. on Information Assurance and Security*, Manchester, 2007, pp. 472–477.

63. Naeem, N.A., Batchelder, M., and Hendren, L., Metrics for measuring the effectiveness of decompilers and obfuscators, *Proc. of the 15th IEEE Int. Conf. on Program*, Banff, Canada, 2007, pp. 253–258.

64. Anckaert, B., Madou, M., De, S.B., *et al.*, Program obfuscation: A quantitative approach, *Proc. of the 2007 ACM Workshop on Quality of Protection*, Alexandria, USA, 2007, pp. 15–20.

65. Tsai, H.Y., Huang, Y.L., and Wagner, D., A graph approach to quantitative analysis of control-flow obfuscating, *IEEE Trans. Information Forensics Security*, 2009, vol. **4**, no. 2, pp. 257–267.

66. Cousot, P. and Cousot, R., An abstract interpretation-based framework for software watermarking, *Proc. of 31st ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, 2004, pp. 173–185.

67. Zakharov, V.A. and Ivanov, K.S., Program obfuscation as obstruction of program static analysis, *Tr. Inst. Sistemnogo Program. Ross. Akad. Nauk*, 2004, vol. **6**, pp. 141–161.

68. Zakharov, V.A. and Ivanov, K.S., On counteraction to some algorithms of program static analysis, *Proc. of Conf. "Mathematics and Safety of Information Technologies"*, 2003, pp. 282–286.

69. Dalla Preda, M. and Giacobazzi, R., Semantic-based code obfuscation by abstract interpretation, *Lecture Notes Comput. Sci.*, 2005, vol. **3580**, pp. 1325–1336.

70. Zakharov, V.A. and Ivanov, K.S., Program models related to the problem of counteraction to algorithms of program static analysis, *Tr. Inst. Sistemnogo Program. Ross. Akad. Nauk*, 2006, vol. 11.

71. Varnovsky, N.P., Zakharov, V.A., Kuzyurin, N.N., Podlovchenko, R.I., Shokurov, A.V., and Shcherbina, V.L., On the use of program deobfuscation methods for detecting complex computer viruses, *Izv. Taganrogskogo Radiotekhnicheskogo Univ.*, 2005, vol. 4, pp. 2006, vol. 6, pp. 18–27.

72. Kuzurin, N.N., Podlovchenko, R.I., Scherbina, V.L., and Zakharov, V.A., Using algebraic models of programs for detecting metamorphic malwares, *Tr. Inst. Sistemnogo Program. Ross. Akad. Nauk*, 2007, vol. **12**, pp. 77–94.

73. Della Preda, M. and Giacobazzi, G., Semantic-based code obfuscation by abstract interpretation, *J. Comput. Security*, 2009, vol. **17**, no. 6, pp. 855–908.

74. Christodorescu, M. and Jha, S., Static analysis of executables to detect malicious patterns, *Proc. of the 12th Security Symp.*, 2003, pp. 169–186.

75. Della Preda, M., Christodorescu, M., Jha, S., and Debray, S., A semantic-based approach to malware detection, *Proc. of the 34th Annu. ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, 2007, pp. 377–388.

76. Della Preda, M., Giacobazzi, G., Debray, S., Coogan, K., and Townsend, G., Modelling metamorphism by abstract interpretation, *Lect. Notes Comput. Sci.*, 2010, vol. **6337**, pp. 218–235.

77. Majumdar, A. and Thomborson, C., On the use of opaque predicates in mobile agent code obfuscation, *Proc. of the ISI 2005*, Altanta, 2005, pp. 648–649.

78. Majumdar, A. and Thomborson, C., Manufacturing opaque predicates in distributed systems for code obfuscation, *Proc. of the 4th Int. Conf. on Information Security*, Hobart, Australia, 2006, pp. 187–196.

79. Della Preda, M., Giacobazzi, G., Madou, M., and de Bosschere, K., Opaque predicate detection by abstract interpretation, *Proc. of the 11th Int. Conf. on Algebriac Methodology and Software Technology, Lecture Notes Comput. Sci.*, 2006, vol. **4019**, pp. 81–95.

80. Wang, C., Davidson, J., Hill, J., and Knight, J., Protection of software-based survivability mechanisms, *Proc. of the Int. Conf. of Dependable Systems and Networks*, 2001.

81. Chow, S., Gu, Y., Johnson, H., and Zakharov, V., An approach to obfuscation of control-flow of sequential programs, *Lecture Notes Comput. Sci.*, 2001, vol. **2000**, pp. 144–155.

82. Ogiso, T., Sakabe, Y., Soshi, M., and Miyaji, A., Software obfuscation on a theoretical basis and its implementation, *Inst. Electron., Inf. Commun. Eng., Trans., Sect. E*, 2003, E86-A(1).

83. Varnovskii, N.P., Zakharov, V.A., Kuzyurin, N.N., and Shokurov, A.V., On prospcts of solving program obfuscation problem, *Proc. of Conf. "Mathematics and Safety of Information Technologies"*, 2003, pp. 344–351.

84. Ostrovsky, R., Efficient computation on oblivious RAMs, *Proc. of the 22nd Annu. ACM Symp. on Theory of Computing*, 1990, pp. 514–523.

85. Zhuang, X., Zhang, T., Lee, H.-H.S., and Pande, S., Hardware assisted control flow obfuscation for embedded processes, *Proc. of the 2004 Int. Conf. on Compilers, Architecture, and Synthesis for Embedded Systems*, 2004, pp. 292–302.

86. Bhatkar, S., Du Varney, D.C., and Sekar, R., Address obfuscation: An efficient approach to combat a broad range of memory error exploits, *Proc. of the 12th Conf. on USENIX Security Symp.*, 2003, vol. 8, pp. 105–120.

87. Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., and Waters, B., Candidate indistinguishability obfuscation and functional encryption for all circuits, *Proc. of the 2013 IEEE 54nd Annu. Symp. on Foundations of Computer Science*, 2013, pp. 40–49.

88. Hada, S., Secure obfuscation for encrypted signatures, *Lect. Notes Comput. Sci.*, 2010, vol. **6110**, pp. 92–112.

89. Adida, B. and Wikstrom, D., How to shuffle in public, *Lect. Notes Comput. Sci.*, 2007, vol. **4392**, pp. 555–574.

90. Canetti, R., Dwork, C., Naor, M., and Ostrovsky, R., Deniable encryption, *Lect. Notes Comput. Sci.*, 1997, vol. **1294**, pp. 90–104.

91. Sahai, A. and Waters, B., How to use indistinguishability obfuscation: Deniable encryption, and more, *Proc. of the 22nd Annu. ACM Symp. on Theory of Computing*, 2014, pp. 475–484.

92. Hada, S., Zero-knowledge and code obfuscation, *Lect. Notes Comput. Sci.*, 2000, vol. **1976**, pp. 443–457.

93. Savage, J., *Models of Computation: Exploring the Power of Computing*, Boston: Addison-Wesley, 1997.

94. Valiant, L., A theory of learnable, *Commun. ACM*, 1984, vol. **27**, no. 11, pp. 1134–1142.

95. Bitansky, N. and Canetti, R., On obfuscation with strong simulators, *Lect. Notes Comput. Sci.*, 2010, vol. **6223**, pp. 520–537.

96. Goldwasser, S. and Kalai, T.Y., On the impossibility of obfuscation with auxiliary input, *Proc. of the 46th IEEE Annu. Symp. on Foundations of Computer Science*, 2005, pp. 553–562.

97. Gentry, C., Fully homomorphic encryption using ideal lattices, *Proc. of the 41st ACM Symp. on Theory of Computing (STOC 2009)*, 2009, pp. 169–178.

98. Gentry, C., Computing arbitrary functions of encrypted data, *Commun. ACM*, 2010, vol. **53**, no. 3, pp. 97–105.

99. Gentry, C. and Halevi, S., Implementing Gentry's fully-homomorphic encryption scheme, *Lect. Notes Comput. Sci.*, 2011, vol. **6632**, pp. 129–148.

100. Brakerski, Z. and Vaikuntanathan, V., Efficient fully homomorphic encryption from (standard) LWE, *Proc. of the 2011 IEEE 54nd Annu. Symp. on Foundations of Computer Science*, 2011, pp. 97–106.

101. Gentry, C., Lewko, A.L., and Waters, B., Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based, *Lect. Notes Comput. Sci.*, 2013, vol. **8042**, pp. 75–92.

102. Brakerski, Z. and Rothblum, G.N., Virtual black-box obfuscation for all circuits via generic graded encoding, *Lect. Notes Comput. Sci.*, 2014, vol. **8349**, pp. 1–25.

103. Barak, B., Garg, S., Kalai, Y.T., Paneth, O., and Sahai, A., Protecting obfuscation against algebraic attacks, *Lect. Notes Comput. Sci.*, 2014, vol. **8441**, pp. 221–238.

104. Canetti, R., Kalai, Y.T., Paneth. O., On obfuscation with random oracles, *Lect. Notes Comput. Sci.*, 2015, vol. **9015**, pp. 456–467.

Translated by A. Pesterev

SPELL: OK