On the (Im)possibility of Obfuscating Programs

(Extended Abstract)

Boaz Barak[1], Oded Goldreich[1], Rusell Impagliazzo[2], Steven Rudich[3], Amit Sahai[4], Salil Vadhan[5], and Ke Yang[3]

[1] Department of Computer Science, Weizmann Institute of Science, Rehovot, ISRAEL. {boaz,oded}@wisdom.weizmann.ac.il

[2] Department of Computer Science and Engineering, University of California, San Diego, La Jolla, CA 92093-0114. russell@cs.ucsd.edu

[3] Computer Science Department, Carnegie Mellon University, 5000 Forbes Ave. Pittsburgh, PA 15213. {rudich,yangke}@cs.cmu.edu

[4] Department of Computer Science, Princeton University, 35 Olden St. Princeton, NJ 08540. sahai@cs.princeton.edu

[5] Division of Engineering and Applied Sciences, Harvard University, 33 Oxford Street, Cambridge, MA 02138. salil@eecs.harvard.edu

Abstract. Informally, an *obfuscator* $\mathcal{O}$ is an (efficient, probabilistic) "compiler" that takes as input a program (or circuit) P and produces a new program $\mathcal{O}(P)$ that has the same functionality as P yet is "unintelligible" in some sense. Obfuscators, if they exist, would have a wide variety of cryptographic and complexity-theoretic applications, ranging from software protection to homomorphic encryption to complexity-theoretic analogues of Rice's theorem. Most of these applications are based on an interpretation of the "unintelligibility" condition in obfuscation as meaning that $\mathcal{O}(P)$ is a "virtual black box," in the sense that anything one can efficiently compute given $\mathcal{O}(P)$, one could also efficiently compute given oracle access to P.

In this work, we initiate a theoretical investigation of obfuscation. Our main result is that, even under very weak formalizations of the above intuition, obfuscation is impossible. We prove this by constructing a family of functions $\mathcal{F}$ that are *inherently unobfuscatable* in the following sense: there is a property $\pi : \mathcal{F} \to \{0,1\}$ such that (a) given *any program* that computes a function $f \in \mathcal{F}$, the value $\pi(f)$ can be efficiently computed, yet (b) given *oracle access* to a (randomly selected) function $f \in \mathcal{F}$, no efficient algorithm can compute $\pi(f)$ much better than random guessing.

We extend our impossibility result in a number of ways, including even obfuscators that (a) are not necessarily computable in polynomial time, (b) only *approximately* preserve the functionality, and (c) only need to work for very restricted models of computation ($\mathbf{TC_0}$). We also rule out several potential applications of obfuscators, by constructing "unobfuscatable" signature schemes, encryption schemes, and pseudorandom function families.

1 Introduction

The past few decades of cryptography research has had amazing success in putting most of the classical cryptographic problems — encryption, authentication, protocols — on complexity-theoretic foundations. However, there still remain several important problems in cryptography about which theory has had little or nothing to say. One such problem is that of *program obfuscation.* Roughly speaking, the goal of (program) obfuscation is to make a program "unintelligible" while preserving its functionality. Ideally, an obfuscated program should be a "virtual black box," in the sense that anything one can compute from it one could also compute from the input-output behavior of the program.

The hope that some form of obfuscation is possible arises from the fact that analyzing programs expressed in rich enough formalisms is hard. Indeed, any programmer knows that total unintelligibility is the natural state of computer programs (and one must work hard in order to keep a program from deteriorating into this state). Theoretically, results such as Rice's Theorem and the hardness of the Halting Problem and Satisfiability all seem to imply that the only useful thing that one can do with a program or circuit is to run it (on inputs of one's choice). However, this informal statement is, of course, an overgeneralization, and the existence of obfuscators requires its own investigation.

To be a bit more clear (though still informal), an *obfuscator* $\mathcal{O}$ is an (efficient, probabilistic) "compiler" that takes as input a program (or circuit) P and produces a new program $\mathcal{O}(P)$ satisfying the following two conditions:

- (functionality) $\mathcal{O}(P)$ computes the same function as P.
- ("virtual black box" property) "Anything that can be efficiently computed from $\mathcal{O}(P)$ can be efficiently computed given oracle access to P."

While there are heuristic approaches to obfuscation in practice (cf., Figure 1 and [CT00]), there has been little theoretical work on this problem. This is unfortunate, since obfuscation, if it were possible, would have a wide variety of cryptographic and complexity-theoretic applications.

```
#include<stdio.h> #include<string.h> main(){char*O,l[999]=
"'`acgo\177~|xp .-\0R^8)NJ6%K4O+A2M(*0ID57$3G1FBL";while(O=
fgets(l+45,954,stdin)){*l=O[strlen(O)[O-1]=0,strspn(O,l+11)];
while(*O)switch((*l&&isalnum(*O))-!*l){case-1:{char*I=(O+=
strspn(O,l+12)+1)-2,O=34;while(*I&3&&(O=(O-16<<1)+*I---'-')<80);
putchar(O&93?*I&8||!(  I=memchr( l , O , 44 ) ) ?'?':I-l+47:32);
break;case 1: ;}*l=(*O&31)[l-15+(*O>61)*32];while(putchar(45+*l%2),
(*l=*l+32>>1)>35);case 0:putchar((++O,32));}putchar(10);}}
```

Fig. 1. The winning entry of the 1998 *International Obfuscated C Code Contest*, an ASCII/Morse code translator by Frans van Dorsselaer [vD98] (adapted for this paper).

In this work, we initiate a theoretical investigation of obfuscation. We examine various formalizations of the notion, in an attempt to understand what we can and cannot hope to achieve. Our main result is a negative one, showing that obfuscation (as it is typically understood) is *impossible*. Before describing this result and others in more detail, we outline some of the potential applications of obfuscators, both for motivation and to clarify the notion.

1.1 Some Applications of Obfuscators

Software Protection . The most direct applications of obfuscators are for various forms of software protection. By definition, obfuscating a program protects it against reverse engineering. For example, if one party, Alice, discovers a more efficient algorithm for factoring integers, she may wish to sell another party, Bob, a program for apparently weaker tasks (such as breaking the RSA cryptosystem) that use the factoring algorithm as a subroutine without actually giving Bob a factoring algorithm. Alice could hope to achieve this by obfuscating the program she gives to Bob.

Intuitively, obfuscators would also be useful in *watermarking* software (cf., [CT00, NSS99]). A software vendor could modify a program's behavior in a way that uniquely identifies the person to whom it is sold, and then obfuscate the program to guarantee that this "watermark" is difficult to remove.

Homomorphic Encryption. A long-standing open problem is whether *homomorphic* encryption schemes exist (cf., [RAD78, FM91, DDN00, BL96, SYY99]). That is, we seek a secure public-key cryptosystem for which, given encryptions of two bits (and the public key), one can compute an encryption of any binary Boolean operation of those bits. Obfuscators would allow one to convert any public-key cryptosystem into a homomorphic one: use the secret key to construct an algorithm that performs the required computations (by decrypting, applying the Boolean operation, and encrypting the result), and publish an obfuscation of this algorithm together with the public key.[1]

Removing Random Oracles. The *Random Oracle Model* [BR93] is an idealized cryptographic setting in which all parties have access to a truly random function. It is (heuristically) hoped that protocols designed in this model will remain secure when implemented using an efficient, publicly computable cryptographic hash function in place of the random function. While it is known that this is not true in general [CGH98], it is unknown whether there exist efficiently computable functions with strong enough properties to be securely used in place

[1] There is a subtlety here, caused by the fact that encryption algorithms must be *probabilistic* to be semantically secure in the usual sense [GM84]. However, both the "functionality" and "virtual black box" properties of obfuscators become more complex for probabilistic algorithms, so in this work, we restrict our attention to obfuscating deterministic algorithms. This restriction only makes our main (impossibility) result stronger.

of the random function in various *specific* protocols (e.g., in Fiat-Shamir type schemes [FS87]). One might hope to obtain such functions by obfuscating a family of pseudorandom functions [GGM86], whose input-output behavior is by definition indistinguishable from that of a truly random function.

Transforming Private-Key Encryption into Public-Key Encryption. Obfuscation can also be used to create new public-key encryption schemes by obfuscating a private-key encryption scheme. Given a secret key K of a private-key encryption scheme, one can publish an obfuscation of the encryption algorithm Enc_K.[2] This allows everyone to encrypt, yet only one possessing the secret key K should be able to decrypt.

1.2 Our Results

The Basic Impossibility Result. Most of the above applications rely on the intuition that an obfuscated program is a "virtual black box." That is, anything one can efficiently compute from the obfuscated program, one should be able to efficiently compute given just oracle access to the program.

Our main result shows that it is impossible to achieve this notion of obfuscation. We prove this by constructing (from any one-way function) a family $\mathcal{F}$ of functions which is *inherently unobfuscatable* in the sense that there is some property $\pi : \mathcal{F} \to \{0,1\}$ such that:

– Given *any* program (circuit) that computes a function $f \in \mathcal{F}$, the value $\pi(f)$ can be efficiently computed;
– Yet, given oracle access to a (randomly selected) function $f \in \mathcal{F}$, no efficient algorithm can compute $\pi(f)$ much better than by random guessing.

Thus, there is no way of obfuscating the programs that compute these functions, even if (a) the obfuscation is meant to hide only one bit of information about the function (namely $\pi(f)$), and (b) the obfuscator itself has unbounded computation time.

We believe that the existence of such functions shows that the "virtual black box" paradigm for obfuscators is inherently flawed. Any hope for positive results about obfuscator-like objects must abandon this viewpoint, or at least be reconciled with the existence of functions as above.

Approximate Obfuscators. The basic impossibility result as described above applies to obfuscators $\mathcal{O}$ for which we require that the obfuscated program $\mathcal{O}(P)$ computes exactly the same function as the original program P. However, for some applications it may suffice that, for every input x, $\mathcal{O}(P)$ and P agree on x with high probability (over the coin tosses of $\mathcal{O}$). Using some additional ideas, our impossibility result extends to such *approximate obfuscators.*

[2] This application involves the same subtlety pointed out in Footnote 1. Thus, our results regarding the (un)obfuscatability of private-key encryption schemes (described later) refer to a relaxed notion of security in which multiple encryptions of the same message are not allowed (which is consistent with a deterministic encryption algorithm).

Impossibility of Applications. To give further evidence that our impossibility result is not an artifact of definitional choices, but rather that there is something inherently flawed in the "virtual black box" idea, we also demonstrate that several of the applications of obfuscators are also impossible. We do this by constructing *inherently unobfuscatable* signature schemes, encryption schemes, and pseudorandom functions. These are objects satisfying the standard definitions of security (except for the subtlety noted in Footnote 2), but for which one can efficiently compute the secret key K from *any program* that signs (or encrypts or evaluates the pseudorandom function, resp.) relative to K. (Hence handing out "obfuscated forms" of these keyed-algorithms is highly insecure.)

In particular, we complement Canetti et. al.'s critique of the Random Oracle Methodology [CGH98]. They show that there exist (contrived) protocols that are secure in the idealized Random Oracle Model (of [BR93]), but are *insecure* when the random oracle is replaced with *any* (efficiently computable) function. Our results imply that for even for *natural* protocols that are secure in the random oracle model (e.g., Fiat-Shamir type schemes [FS87]), there exist (contrived) pseudorandom functions, such that these protocols are insecure when the random oracle is replaced with *any program* that computes the contrived function.

Obfuscating restricted complexity classes. Even though obfuscation of general programs/circuits is impossible, one may hope that it is possible to obfuscate more restricted classes of computations. However, using the pseudorandom functions of [NR97] in our construction, we can show that the impossibility result holds even when the input program P is a constant-depth threshold circuit (i.e., is in $\mathbf{TC_0}$), under widely believed complexity assumptions (e.g., the hardness of factoring).

Obfuscating Sampling Algorithms. Another way in which the notion of obfuscators can be weakened is by changing the functionality requirement. Until now, we have considered programs in terms of the functions they compute, but sometimes one is interested in other kinds of behavior. For example, one sometimes considers *sampling algorithms*, i.e. probabilistic programs that take no input (other than, say, a length parameter) and produce an output according to some desired distribution. We consider two natural definitions of obfuscators for sampling algorithms, and prove that the stronger definition is impossible to meet. We also observe that the weaker definition implies the nontriviality of statistical zero knowledge.

Software Watermarking. As mentioned earlier, there appears to be some connection between the problems of software watermarking and code obfuscation. In the full version of the paper [BGI$^+$01], we consider a couple of formalizations of the watermarking problem and explore their relationship to our results on obfuscation.

1.3 Discussion

Our work rules out the standard, "virtual black box" notion of obfuscators as impossible, along with several of its applications. However, it does not mean that there is no method of making programs "unintelligible" in some meaningful and precise sense. Such a method could still prove useful for software protection.

Thus, we consider it to be both important and interesting to understand whether there are alternative senses (or models) in which some form of obfuscation is possible. Towards this end, in the full version of the paper we suggest two weaker definitions of obfuscators that avoid the "virtual black box" paradigm (and hence are not ruled out by our impossibility proof). These definitions could be the subject of future investigations, but we hope that other alternatives will also be proposed and examined.

As is usually the case with impossibility results and lower bounds, we show that obfuscators (in the "virtual black box" sense) do not exist by supplying a somewhat contrived counterexample of a function ensemble that cannot be obfuscated. It is interesting whether obfuscation is possible for a restricted class of algorithms, which nonetheless contains some "useful" algorithms. If we try to restrict the algorithms by their computational complexity, then there's not much hope for obfuscation. Indeed, as mentioned above, we show that (under widely believed complexity assumptions) our counterexample can be placed in $\mathbf{TC_0}$. In general, the complexity of our counterexample is essentially the same as the complexity of pseudorandom functions, and so a complexity class which does not contain our example will also not contain many cryptographically useful algorithms.

1.4 Additional Related Work

There are a number of heuristic approaches to obfuscation and software watermarking in the literature, as described in the survey of Collberg and Thomborson [CT00]. A theoretical study of software protection was previously conducted by Goldreich and Ostrovsky [GO96], who considered *hardware-based* solutions.

Hada [Had00] gave some definitions for code obfuscators which are stronger than the definitions we consider in this paper, and showed some implications of the existence of such obfuscators. (Our result rules out also the existence of obfuscators according to the definitions of [Had00].)

Canetti, Goldreich and Halevi [CGH98] showed another setting in cryptography where getting a function's description is provably more powerful than black-box access. As mentioned above, they have shown that there exist protocols that are secure when executed with black-box access to a random function, but insecure when instead the parties are given a description of any hash function.

1.5 Organization of the Paper

In Section 2, we give some basic definitions along with (very weak) definitions of obfuscators. In Section 3, we prove the impossibility of obfuscators by constructing an inherently unobfuscatable function ensemble. Other extensions and results are deferred to the full version of the paper [BGI$^+$01].

2 Definitions

2.1 Preliminaries

TM is shorthand for Turing machine. *PPT* is shorthand for probabilistic polynomial-time Turing machine. For algorithms A and M and a string x, we denote by $A^M(x)$ the output of A when executed on input x and oracle access to M. If A is a probabilistic Turing machine then by $A(x;r)$ we refer to the result of running A on input x and random tape r. 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 $x \stackrel{\mathrm{R}}{\leftarrow} D$ we mean that x is a random variable distributed according to D. If S is a set then by $x \stackrel{\mathrm{R}}{\leftarrow} S$ we mean that x is a random variable that is distributed uniformly over the elements of S. $\mathrm{Supp}(D)$ denotes the *support* of distribution D, i.e. the set of points that have nonzero probability under D. A function $\mu : \mathbb{N} \to \mathbb{N}$ is called *negligible* if it grows slower than the inverse of any polynomial. That is, for any positive polynomial $p(\cdot)$ there exists $N \in \mathbb{N}$ such that $\mu(n) < 1/p(n)$ for any $n > N$. We'll sometimes use $\mathrm{neg}(\cdot)$ to denote an unspecified negligible function. We will identify Turing machines and circuits with their canonical representations as strings in $\{0,1\}^*$.

2.2 Obfuscators

In this section, we aim to formalize the notion of obfuscators based on the "virtual black box" property as described in the introduction. Recall that this property requires that "anything that an adversary can compute from an obfuscation $\mathcal{O}(P)$ of a program P, it could also compute given just oracle access to P." We shall define what it means for the adversary to successfully compute something in this setting, and there are several choices for this (in decreasing order of generality):

- (computational indistinguishability) The most general choice is not to restrict the nature of what the adversary is trying to compute, and merely require that it is possible, given just oracle access to P, to produce an output distribution that is computationally indistinguishable from what the adversary computes when given $\mathcal{O}(P)$.
- (satisfying a relation) An alternative is to consider the adversary as trying to produce an output that satisfies an arbitrary (possibly polynomial-time) relation with the original program P, and require that it is possible, given just oracle access to P, to succeed with roughly the same probability as the adversary does when given $\mathcal{O}(P)$.
- (computing a function) A weaker requirement is to restrict the previous requirement to relations which are functions; that is, the adversary is trying to compute some function of the original program.

– (computing a predicate) The weakest is to restrict the previous requirement to $\{0,1\}$-valued functions; that is, the adversary is trying to decide some property of the original program.

Since we will be proving impossibility results, our results are strongest when we adopt the weakest requirement (i.e., the last one). This yields two definitions for obfuscators, one for programs defined by Turing machines and one for programs defined by circuits.

Definition 2.1 (TM obfuscator). *A probabilistic algorithm $\mathcal{O}$ is a* TM obfuscator *if the following three conditions hold:*

- *(functionality) For every TM M, the string $\mathcal{O}(M)$ describes a TM that computes the same function as M.*
- *(polynomial slowdown) The description length and running time of $\mathcal{O}(M)$ are at most polynomially larger than that of M. That is, there is a polynomial p such that for every TM M, $|\mathcal{O}(M)| \le p(|M|)$, and if M halts in t steps on some input x, then $\mathcal{O}(M)$ halts within $p(t)$ steps on x.*
- *("virtual black box" property) For any PPT A, there is a PPT S and a negligible function α such that for all TMs M*

$$\left| \Pr\left[A(\mathcal{O}(M)) = 1\right] - \Pr\left[S^M(1^{|M|}) = 1\right] \right| \le \alpha(|M|).$$

We say that $\mathcal{O}$ is efficient *if it runs in polynomial time.*

Definition 2.2 (circuit obfuscator). *A probabilistic algorithm $\mathcal{O}$ is a* (circuit) obfuscator *if the following three conditions hold:*

- *(functionality) For every circuit C, the string $\mathcal{O}(C)$ describes a circuit that computes the same function as C.*
- *(polynomial slowdown) There is a polynomial p such that for every circuit C, $|\mathcal{O}(C)| \le p(|C|)$.*
- *("virtual black box" property) For any PPT A, there is a PPT S and a negligible function α such that for all circuits C*

$$\left| \Pr\left[A(\mathcal{O}(C)) = 1\right] - \Pr\left[S^C(1^{|C|}) = 1\right] \right| \le \alpha(|C|).$$

We say that $\mathcal{O}$ is efficient *if it runs in polynomial time.*

We call the first two requirements (functionality and polynomial slowdown) the *syntactic requirements* of obfuscation, as they do not address the issue of security at all.

There are a couple of other natural formulations of the "virtual black box" property. The first, which more closely follows the informal discussion above, asks that for every predicate π, the probability that $A(\mathcal{O}(C)) = \pi(C)$ is at most the probability that $S^C(1^{|C|}) = \pi(C)$ plus a negligible term. It is easy to see that this requirement is equivalent to the ones above. Another formulation refers to the distinguishability between obfuscations of two TMs/circuits: ask that for every C_1 and C_2, $|\Pr[A(\mathcal{O}(C_1)) = 1] - \Pr[A(\mathcal{O}(C_2))]|$ is approximately equal to $|\Pr\left[S^{C_1}(1^{|C_1|}, 1^{|C_2|}) = 1\right] - \Pr\left[S^{C_2}(1^{|C_1|}, 1^{|C_2|})\right]|$. This definition appears to be slightly weaker than the ones above, but our impossibility proof also rules it out.

Note that in both definitions, we have chosen to simplify the definition by using the size of the TM/circuit to be obfuscated as a security parameter. One can always increase this length by padding to obtain higher security.

The main difference between the circuit and TM obfuscators is that a circuit computes a function with finite domain (all the inputs of a particular length) while a TM computes a function with infinite domain. Note that if we had not restricted the size of the obfuscated circuit $\mathcal{O}(C)$, then the (exponential size) list of all the values of the circuit would be a valid obfuscation (provided we allow S running time $\text{poly}(|\mathcal{O}(C)|)$ rather than $\text{poly}(|C|)$). For Turing machines, it is not clear how to construct such an obfuscation, even if we are allowed an exponential slowdown. Hence obfuscating TMs is intuitively harder. Indeed, it is relatively easy to prove:

Proposition 2.3. *If a TM obfuscator exists, then a circuit obfuscator exists.*

Thus, when we prove our impossibility result for circuit obfuscators, the impossibility of TM obfuscators will follow. However, considering TM obfuscators will be useful as motivation for the proof.

We note that, from the perspective of applications, Definitions 2.1 and 2.2 are already too weak to have the wide applicability discussed in the introduction. The point is that they are nevertheless *impossible* to satisfy (as we will prove).

3 The Main Impossibility Result

To state our main result we introduce the notion of inherently unobfuscatable function ensemble.

Definition 3.1. *An* inherently unobfuscatable function ensemble *is an ensemble $\{\mathcal{H}_k\}_{k\in\mathbb{N}}$ of distributions $\mathcal{H}_k$ on finite functions (from, say, $\{0,1\}^{l_{\text{in}}(k)}$ to $\{0,1\}^{l_{\text{out}}(k)}$) such that:*

- *(efficiently computable) Every function $f \stackrel{R}{\leftarrow} \mathcal{H}_k$ is computable by a circuit of size* $\text{poly}(k)$. *(Moreover, a distribution on circuits consistent with $\mathcal{H}_k$ can be sampled uniformly in time* $\text{poly}(k)$.*)*
- *(unobfuscatability) There exists a function $\pi : \bigcup_{k\in\mathbb{N}} \text{Supp}(\mathcal{H}_k) \to \{0,1\}$ such that*
  1. *$\pi(f)$ is hard to compute with black-box access to f: For any PPT S*

$$\Pr_{f \stackrel{R}{\leftarrow} \mathcal{H}_k} [S^f(1^k) = \pi(f)] \leq \frac{1}{2} + \text{neg}(k)$$

2. *$\pi(f)$ is easy to compute with access to any circuit that computes f: There exists a PPT A such that for any $f \in \bigcup_{k \in \mathbb{N}} \mathrm{Supp}(\mathcal{H}_k)$ and for any circuit C that computes f*

$$A(C) = \pi(f)$$

We prove in Theorem 3.9 that, assuming one-way functions exist, there exists an inherently unobfuscatable function ensemble. This implies that, under the same assumption, there is no obfuscator that satisfies Definition 2.2 (actually we prove the latter fact directly in Theorem 3.6). Since the existence of an *efficient* obfuscator implies the existence of one-way functions (Lemma 3.7), we conclude that efficient obfuscators do not exist (unconditionally).

However, the existence of inherently unobfuscatable function ensemble has even stronger implications. As mentioned in the introduction, these functions can not be obfuscated even if we allow the following relaxations to the obfuscator:

1. As mentioned above, the obfuscator does not have to run in polynomial time — it can be any random process.
2. The obfuscator has only to work for functions in $\mathrm{Supp}(\mathcal{H}_k)$ and only for a non-negligible fraction of these functions under the distributions $\mathcal{H}_k$.
3. The obfuscator has only to hide an *a priori* fixed property π from an *a priori* fixed adversary A.

Structure of the Proof of the Main Impossibility Result. We shall prove our result by first defining obfuscators that are secure also when applied to several (e.g., two) algorithms and proving that they do not exist. Then we shall modify the construction in this proof to prove that TM obfuscators in the sense of Definition 2.1 do not exist. After that, using an additional construction (which requires one-way functions), we will prove that a circuit obfuscator as defined in Definition 2.2 does not exist if one-way functions exist. We will then observe that our proof actually yields an unobfuscatable function ensemble (Theorem 3.9).

3.1 Obfuscating two TMs/circuits

Obfuscators as defined in the previous section provide a "virtual black box" property when a single program is obfuscated, but the definitions do not say anything about what happens when the adversary can inspect more than one obfuscated program. In this section, we will consider extensions of those definitions to obfuscating two programs, and prove that they are impossible to meet. The proofs will provide useful motivation for the impossibility of the original one-program definitions.

Definition 3.2 (2-TM obfuscator). *A* 2-TM obfuscator *is defined in the same way as a TM obfuscator, except that the "virtual black box" property is strengthened as follows:*

– *("virtual black box" property) For any PPT A, there is a PPT S and a negligible function α such that for all TMs M, N*

$$\left|\Pr\left[A(\mathcal{O}(M), \mathcal{O}(N)) = 1\right] - \Pr\left[S^{M,N}(1^{|M|+|N|}) = 1\right]\right| \leq \alpha(\min\{|M|, |N|\})$$

2-circuit obfuscators are defined by modifying the definition of circuit obfuscators in an analogous fashion.

Proposition 3.3. *Neither 2-TM nor 2-circuit obfuscators exist.*

Proof. We begin by showing that 2-TM obfuscators do not exist. Suppose, for sake of contradiction, that there exists a 2-TM obfuscator $\mathcal{O}$. The essence of this proof, and in fact of all the impossibility proofs in this paper, is that there is a fundamental difference between getting black-box access to a function and getting a program that computes it, no matter how obfuscated: A program is a succinct description of the function, on which one can perform computations (or run other programs). Of course, if the function is (exactly) learnable via oracle queries (i.e., one can acquire a program that computes the function by querying it at a few locations), then this difference disappears. Hence, to get our counterexample, we will use a function that cannot be exactly learned with oracle queries. A very simple example of such an unlearnable function follows. For strings $\alpha, \beta \in \{0,1\}^k$, define the Turing machine

$$C_{\alpha,\beta}(x) \stackrel{\text{def}}{=} \begin{cases} \beta & x = \alpha \\ 0^k & \text{otherwise} \end{cases}$$

We assume that on input x, $C_{\alpha,\beta}$ runs in $10 \cdot |x|$ steps (the constant 10 is arbitrary). Now we will define a TM $D_{\alpha,\beta}$ that, given the code of a TM C, can distinguish between the case that C computes the same function as $C_{\alpha,\beta}$ from the case that C computes the same function as $C_{\alpha',\beta'}$ for any $(\alpha',\beta') \neq (\alpha,\beta)$.

$$D_{\alpha,\beta}(C) \stackrel{\text{def}}{=} \begin{cases} 1 & C(\alpha) = \beta \\ 0 & \text{otherwise} \end{cases}$$

(Actually, this function is uncomputable. However, as we shall see below, we can use a modified version of $D_{\alpha,\beta}$ that only considers the execution of $C(\alpha)$ for $\mathrm{poly}(k)$ steps, and outputs 0 if C does not halt within that many steps, for some fixed polynomial $\mathrm{poly}(\cdot)$. We will ignore this issue for now, and elaborate on it later.) Note that $C_{\alpha,\beta}$ and $D_{\alpha,\beta}$ have description size $\Theta(k)$.

Consider an adversary A, which, given two (obfuscated) TMs as input, simply runs the second TM on the first one. That is, $A(C,D) = D(C)$. (Actually, like we modified $D_{\alpha,\beta}$ above, we also will modify A to only run D on C for $\mathrm{poly}(|C|,|D|)$ steps, and output 0 if D does not halt in that time.) Thus, for any $\alpha, \beta \in \{0,1\}^k$,

$$\Pr\left[A(\mathcal{O}(C_{\alpha,\beta}), \mathcal{O}(D_{\alpha,\beta})) = 1\right] = 1 \tag{1}$$

Observe that any $\mathrm{poly}(k)$-time algorithm S which has oracle access to $C_{\alpha,\beta}$ and $D_{\alpha,\beta}$ has only exponentially small probability (for a random α and β) of querying either oracle at a point where its value is nonzero. Hence, if we let Z_k be a Turing machine that always outputs 0^k, then for every PPT S,

$$\left|\Pr\left[S^{C_{\alpha,\beta},D_{\alpha,\beta}}(1^k) = 1\right] - \Pr\left[S^{Z_k,D_{\alpha,\beta}}(1^k) = 1\right]\right| \leq 2^{-\Omega(k)}, \tag{2}$$

where the probabilities are taken over α and β selected uniformly in $\{0,1\}^k$ and the coin tosses of S. On the other hand, by the definition of A we have:

$$\Pr\left[A(\mathcal{O}(Z_k), \mathcal{O}(D_{\alpha,\beta})) = 1\right] = 0 \tag{3}$$

The combination of Equations (1), (2), and (3) contradict the fact that $\mathcal{O}$ is a 2-TM obfuscator.

In the above proof, we ignored the fact that we had to truncate the running times of A and $D_{\alpha,\beta}$. When doing so, we must make sure that Equations (1) and (3) still hold. Equation (1) involves executing (a) $A(\mathcal{O}(D_{\alpha,\beta}),\mathcal{O}(C_{\alpha,\beta}))$, which in turn amounts to executing (b) $\mathcal{O}(D_{\alpha,\beta})(\mathcal{O}(C_{\alpha,\beta}))$. By definition (b) has the same functionality as $D_{\alpha,\beta}(\mathcal{O}(C_{\alpha,\beta}))$, which in turn involves executing (c) $\mathcal{O}(C_{\alpha,\beta})(\alpha)$. Yet the functionality requirement of the obfuscator definition assures us that (c) has the same functionality as $C_{\alpha,\beta}(\alpha)$. By the polynomial slowdown property of obfuscators, execution (c) only takes $\mathrm{poly}(10 \cdot k) = \mathrm{poly}(k)$ steps, which means that $D_{\alpha,\beta}(\mathcal{O}(C_{\alpha,\beta}))$ need only run for $\mathrm{poly}(k)$ steps. Thus, again applying the polynomial slowdown property, execution (b) takes $\mathrm{poly}(k)$ steps, which finally implies that A need only run for $\mathrm{poly}(k)$ steps. The same reasoning holds for Equation (3), using Z_k instead of $C_{\alpha,\beta}$.[3] Note that all the polynomials involved are *fixed* once we fix the polynomial $p(\cdot)$ of the polynomial slowdown property.

The proof for the 2-circuit case is very similar to the 2-TM case, with a related, but slightly different subtlety. Suppose, for sake of contradiction, that $\mathcal{O}$ is a 2-circuit obfuscator. For $k \in \mathbb{N}$ and $\alpha, \beta \in \{0,1\}^k$, define Z_k, $C_{\alpha,\beta}$ and $D_{\alpha,\beta}$ in the same way as above but as circuits rather than TMs, and define an adversary A by $A(C,D) = D(C)$. (Note that the issues of A and $D_{\alpha,\beta}$'s running times go away in this setting, since circuits can always be evaluated in time polynomial in their size.) The new subtlety here is that the definition of A as $A(C,D) = D(C)$ only makes sense when the input length of D is larger than the size of C (note that one can always pad C to a larger size). Thus, for the analogues of Equations (1) and (3) to hold, the input length of $D_{\alpha,\beta}$ must be larger than the sizes of the *obfuscations* of $C_{\alpha,\beta}$ and Z_k. However, by the polynomial slowdown property of obfuscators, it suffices to let $D_{\alpha,\beta}$ have input length $\mathrm{poly}(k)$ and the proof works as before.

■

3.2 Obfuscating one TM/circuit

Our approach to extending the two-program obfuscation impossibility results to the one-program definitions is to combine the two programs constructed above into one. This will work in a quite straightforward manner for TM obfuscators, but will require new ideas for circuit obfuscators.

Combining functions and programs. For functions, TMs, or circuits $f_0, f_1 : X \to Y$, define their *combination* $f_0 \# f_1 : \{0,1\} \times X \to Y$ by $(f_0 \# f_1)(b,x) \stackrel{\mathrm{def}}{=} f_b(x)$. Conversely, if we are given a TM (resp., circuit) $C : \{0,1\} \times X \to Y$, we can

[3] Another, even more minor subtlety that we ignored is that, strictly speaking, A only has running time polynomial in the description of the *obfuscations* of $C_{\alpha,\beta}$, $D_{\alpha,\beta}$, and Z_k, which could conceivably be shorter than the original TM descriptions. But a counting argument shows that for all but an exponentially small fraction of pairs $(\alpha,\beta) \in \{0,1\}^k \times \{0,1\}^k$, $\mathcal{O}(C_{\alpha,\beta})$ and $\mathcal{O}(D_{\alpha,\beta})$ must have description size $\Omega(k)$.

efficiently decompose C into $C_0 \# C_1$ by setting $C_b(x) \stackrel{\text{def}}{=} C(b,x)$; note that C_0 and C_1 have size and running time essentially the same as that of C. Observe that having oracle access to a combined function $f_0 \# f_1$ is equivalent to having oracle access to f_0 and f_1 individually.

Theorem 3.4. *TM obfuscators do not exist.*

Proof Sketch: Suppose, for sake of contradiction, that there exists a TM obfuscator $\mathcal{O}$. For $\alpha, \beta \in \{0,1\}^k$, let $C_{\alpha,\beta}$, $D_{\alpha,\beta}$, and Z_k be the TMs defined in the proof of Proposition 3.3. Combining these, we get the TMs $F_{\alpha,\beta} = C_{\alpha,\beta} \# D_{\alpha,\beta}$ and $G_{\alpha,\beta} = Z_k \# C_{\alpha,\beta}$.

We consider an adversary A analogous to the one in the proof of Proposition 3.3, augmented to first decompose the program it is fed. That is, on input a TM F, algorithm A first decomposes F into $F_0 \# F_1$ and then outputs $F_1(F_0)$. (As in the proof of Proposition 3.3, A actually should be modified to run in time $\operatorname{poly}(|F|)$.) Let S be the PPT simulator for A guaranteed by Definition 2.1. Just as in the proof of Proposition 3.3, we have:

$$\Pr\left[A(\mathcal{O}(F_{\alpha,\beta})) = 1\right] = 1 \text{ and } \Pr\left[A(\mathcal{O}(G_{\alpha,\beta})) = 1\right] = 0$$
$$\left|\Pr\left[S^{F_{\alpha,\beta}}(1^k) = 1\right] - \Pr\left[S^{G_{\alpha,\beta}}(1^k) = 1\right]\right| \leq 2^{-\Omega(k)},$$

where the probabilities are taken over uniformly selected $\alpha, \beta \in \{0,1\}^k$, and the coin tosses of A, S, and $\mathcal{O}$. This contradicts Definition 2.1. □

There is a difficulty in trying to carry out the above argument in the circuit setting. (This difficulty is related to (but more serious than) the same subtlety regarding the circuit setting discussed earlier.) In the above proof, the adversary A, on input $\mathcal{O}(F_{\alpha,\beta})$, attempts to evaluate $F_1(F_0)$, where $F_0 \# F_1 = \mathcal{O}(F_{\alpha,\beta}) = \mathcal{O}(C_{\alpha,\beta} \# D_{\alpha,\beta})$. In order for this to make sense in the circuit setting, the size of the circuit F_0 must be at most the input length of F_1 (which is the same as the input length of $D_{\alpha,\beta}$). But, since the output $F_0 \# F_1$ of the obfuscator can be polynomially larger than its input $C_{\alpha,\beta} \# D_{\alpha,\beta}$, we have no such guarantee. Furthermore, note that if we compute F_0, F_1 in the way we described above (i.e., $F_b(x) \stackrel{\text{def}}{=} \mathcal{O}(F_{\alpha,\beta})(b,x)$) then we'll have $|F_0| = |F_1|$ and so F_0 will necessarily be larger than F_1's input length.

To get around this, we modify $D_{\alpha,\beta}$ in a way that will allow A, when given $D_{\alpha,\beta}$ and a circuit C, to test whether $C(\alpha) = \beta$ even when C is larger than the input length of $D_{\alpha,\beta}$. Of course, oracle access to $D_{\alpha,\beta}$ should not reveal α and β, because we do not want the simulator S to be able to test whether $C(\alpha) = \beta$ given just oracle access to C and $D_{\alpha,\beta}$. We will construct such functions $D_{\alpha,\beta}$ based on pseudorandom functions [GGM86].

Lemma 3.5. *If one-way functions exist, then for every $k \in \mathbb{N}$ and $\alpha, \beta \in \{0,1\}^k$, there is a distribution $\mathcal{D}_{\alpha,\beta}$ on circuits such that:*

1. *Every $D \in \operatorname{Supp}(\mathcal{D}_{\alpha,\beta})$ is a circuit of size* $\operatorname{poly}(k)$.

2. *There is a polynomial-time algorithm A such that for any circuit C, and any $D \in \mathrm{Supp}(\mathcal{D}_{\alpha,\beta})$, $A^D(C, 1^k) = 1$ iff $C(\alpha) = \beta$.*
3. *For any PPT S, $\Pr\left[S^D(1^k) = \alpha\right] = \mathrm{neg}(k)$, where the probability is taken over $\alpha, \beta \stackrel{R}{\leftarrow} \{0,1\}^k$, $D \stackrel{R}{\leftarrow} \mathcal{D}_{\alpha,\beta}$, and the coin tosses of S.*

Proof. Basically, the construction implements a private-key "homomorphic encryption" scheme. More precisely, the functions in $\mathcal{D}_{\alpha,\beta}$ will consist of three parts. The first part gives out an encryption of the bits of α (under some private-key encryption scheme). The second part provides the ability to perform binary Boolean operations on encrypted bits, and the third part tests whether a sequence of encryptions consists of encryptions of the bits of β. These operations will enable one to efficiently test whether a given circuit C satisfies $C(\alpha) = \beta$, while keeping α and β hidden when only oracle access to C and $D_{\alpha,\beta}$ is provided.

We begin with any one-bit (probabilistic) private-key encryption scheme $(\mathrm{Enc}, \mathrm{Dec})$ that satisfies indistinguishability under *chosen plaintext* and *non-adaptive chosen ciphertext* attacks. Informally, this means that an encryption of 0 should be indistinguishable from an encryption of 1 even for adversaries that have access to encryption and decryption oracles prior to receiving the challenge ciphertext, and access to just an encryption oracle after receiving the challenge ciphertext. (See [KY00] for formal definitions.) We note that such encryptions schemes exist if one-way functions exist; indeed, the "standard" encryption scheme $\mathrm{Enc}_K(b) = (r, f_K(r) \oplus b)$, where $r \stackrel{R}{\leftarrow} \{0,1\}^{|K|}$ and f_K is a pseudorandom function, has this property.

Now we consider a "homomorphic encryption" algorithm Hom, which takes as input a private-key K and two ciphertexts c and d (w.r.t. this key K), and a binary boolean operation $\odot$ (specified by its 2×2 truth table). We define

$$\mathrm{Hom}_K(c, d, \odot) \stackrel{\mathrm{def}}{=} \mathrm{Enc}_K(\mathrm{Dec}_K(c) \odot \mathrm{Dec}_K(d)).$$

It can be shown that such an encryption scheme retains its security even if the adversary is given access to a Hom oracle. This is formalized in the following claim:

Claim. For every PPT A,

$$\left|\Pr\left[A^{\mathrm{Hom}_K, \mathrm{Enc}_K}(\mathrm{Enc}_K(0)) = 1\right] - \Pr\left[A^{\mathrm{Hom}_K, \mathrm{Enc}_K}(\mathrm{Enc}_K(1)) = 1\right]\right| \leq \mathrm{neg}(k).$$

Proof of claim: Suppose there were a PPT A violating the claim. First, we argue that we can replace the responses to all of A'S Hom_K-oracle queries with encryptions of 0 with only a negligible effect on A's distinguishing gap. This follows from indistinguishability under chosen plaintext and ciphertext attacks and a hybrid argument: Consider hybrids where the first i oracle queries are answered according to Hom_K and the rest with encryptions of 0. Any advantage in distinguishing two adjacent hybrids must be due to distinguishing an encryption of 1 from an encryption of 0. The resulting distinguisher can be implemented using oracle access to encryption and decryption oracles prior to receiving the challenge ciphertext (and an encryption oracle afterwards).

Once we have replaced the Hom_K-oracle responses with encryptions of 0, we have an adversary that can distinguish an encryption of 0 from an encryption of 1 when given access to just an encryption oracle. This contradicts indistinguishability under chosen plaintext attack. □

Now we return to the construction of our circuit family $\mathcal{D}_{\alpha,\beta}$. For a key K, let $E_{K,\alpha}$ be an algorithm which, on input i outputs $\mathrm{Enc}_K(\alpha_i)$, where α_i is the i'th bit of α. Let $B_{K,\beta}$ be an algorithm which when fed a k-tuple of ciphertexts $(c_1, \ldots, c_k)$ outputs 1 if for all i, $\mathrm{Dec}_K(c_i) = \beta_i$, where $\beta_1, \ldots, \beta_k$ are the bits of β. A random circuit from $\mathcal{D}_{\alpha,\beta}$ will essentially be the algorithm

$$D_{K,\alpha,\beta} \stackrel{\mathrm{def}}{=} E_{K,\alpha} \# \mathrm{Hom}_K \# B_{K,\beta}$$

(for a uniformly selected key K). One minor complication is that $D_{K,\alpha,\beta}$ is actually a *probabilistic* algorithm, since $E_{K,\alpha}$ and Hom_K employ probabilistic encryption, whereas the lemma requires deterministic functions. This can be solved in the usual way, by using pseudorandom functions. Let $q = q(k)$ be the input length of $D_{K,\alpha,\beta}$ and $m = m(k)$ the maximum number of random bits used by $D_{K,\alpha,\beta}$ on any input. We can select a pseudorandom function $f_{K'} : \{0,1\}^q \to \{0,1\}^m$, and let $D'_{K,\alpha,\beta,K'}$ be the (determinstic) algorithm, which on input $x \in \{0,1\}^q$ evaluates $D_{K,\alpha,\beta}(x)$ using randomness $f_{K'}(x)$.

Define the distribution $\mathcal{D}_{\alpha,\beta}$ to be $D'_{K,\alpha,\beta,K'}$, over uniformly selected keys K and K'. We argue that this distribution has the properties stated in the lemma. By construction, each $D'_{K,\alpha,\beta,K'}$ is computable by circuit of size $\mathrm{poly}(k)$, so Property 1 is satisfied.

For Property 2, consider an algorithm A that on input C and oracle access to $D'_{K,\alpha,\beta,K'}$ (which, as usual, we can view as access to (deterministic versions of) the three separate oracles $E_{K,\alpha}$, Hom_K, and $B_{K,\alpha}$), proceeds as follows: First, with k oracle queries to the $E_{K,\alpha}$ oracle, A obtains encryptions of each of the bits of α. Then, A uses the Hom_K oracle to do a gate-by-gate emulation of the computation of $C(\alpha)$, in which A obtains encryptions of the values at each gate of C. In particular, A obtains encryptions of the values at each output gate of C (on input α). It then feeds these output encryptions to $D_{K,\beta}$, and outputs the response to this oracle query. By construction, A outputs 1 iff $C(\alpha) = \beta$.

Finally, we verify Property 3. Let S be any PPT algorithm. We must show that S has only a negligible probability of outputting α when given oracle access to $D'_{K,\alpha,\beta,K'}$ (over the choice of K, α, β, K', and the coin tosses of S). By the pseudorandomness of $f_{K'}$, we can replace oracle access to the function $D'_{K,\alpha,\beta,K'}$ with oracle access to the probabilistic algorithm $D_{K,\alpha,\beta}$ with only a negligible effect on S's success probability. Oracle access to $D_{K,\alpha,\beta}$ is equivalent to oracle access to $E_{K,\alpha}$, Hom_K, and $B_{K,\beta}$. Since β is independent of α and K, the probability that S queries $B_{K,\beta}$ at a point where its value is nonzero (i.e., at a sequence of encryptions of the bits of β) is exponentially small, so we can remove S's queries to $B_{K,\beta}$ with only a negligible effect on the success probability. Oracle access to $E_{K,\alpha}$ is equivalent to giving S polynomially many encryptions of each of the bits of α. Thus, we must argue that S cannot compute α with nonnegligible probability from these encryptions and oracle access to Hom_K. This follows from the fact that the encryption scheme remains secure in the presence of a Hom_K oracle (Claim 3.2) and a hybrid argument. ■

Theorem 3.6. *If one-way functions exist, then circuit obfuscators do not exist.*

Proof. Suppose, for sake of contradiction, that there exists a circuit obfuscator $\mathcal{O}$. For $k \in \mathbb{N}$ and $\alpha, \beta \in \{0,1\}^k$, let Z_k and $C_{\alpha,\beta}$ be the circuits defined in the proof of Proposition 3.3, and let $\mathcal{D}_{\alpha,\beta}$ be the distribution on circuits given by Lemma 3.5. Fer each $k \in \mathbb{N}$, consider the following two distributions on circuits of size $\mathrm{poly}(k)$:

$\mathcal{F}_k$**:** Choose α and β uniformly in $\{0,1\}^k$, $D \stackrel{\text{R}}{\leftarrow} \mathcal{D}_{\alpha,\beta}$. Output $C_{\alpha,\beta}\#D$.
$\mathcal{G}_k$**:** Choose α and β uniformly in $\{0,1\}^k$, $D \stackrel{\text{R}}{\leftarrow} \mathcal{D}_{\alpha,\beta}$. Output $Z_k\#D$.

Let A be the PPT algorithm guaranteed by Property 2 in Lemma 3.5, and consider a PPT A' which, on input a circuit F, decomposes $F = F_0\#F_1$ and evaluates $A^{F_1}(F_0, 1^k)$, where k is the input length of F_0. Thus, when fed a circuit from $\mathcal{O}(\mathcal{F}_k)$ (resp., $\mathcal{O}(\mathcal{G}_k)$), A' is evaluating $A^D(C, 1^k)$ where D computes the same function as some circuit from $\mathcal{D}_{\alpha,\beta}$ and C computes the same function as $C_{\alpha,\beta}$ (resp., Z_k). Therefore, by Property 2 in Lemma 3.5, we have:

$$\Pr\left[A'(\mathcal{O}(\mathcal{F}_k)) = 1\right] = 1 \qquad \Pr\left[A'(\mathcal{O}(\mathcal{G}_k)) = 1\right] = 0.$$

We now argue that for any PPT algorithm S

$$\left|\Pr\left[S^{\mathcal{F}_k}(1^k) = 1\right] - \Pr\left[S^{\mathcal{G}_k}(1^k) = 1\right]\right| \leq 2^{-\Omega(k)},$$

which will contradict the definition of circuit obfuscators. Having oracle access to a circuit from $\mathcal{F}_k$ (respectively, $\mathcal{G}_k$) is equivalent to having oracle access to $C_{\alpha,\beta}$ (resp., Z_k) and $D \stackrel{\text{R}}{\leftarrow} \mathcal{D}_{\alpha,\beta}$, where α, β are selected uniformly in $\{0,1\}^k$. Property 3 of Lemma 3.5 implies that the probability that S queries the first oracle at α is negligible, and hence S cannot distinguish that oracle being $C_{\alpha,\beta}$ from it being Z_k. ■

We can remove the assumption that one-way functions exist for *efficient* circuit obfuscators via the following (easy) lemma (proven in the full version of the paper).

Lemma 3.7. *If efficient obfuscators exist, then one-way functions exist.*

Corollary 3.8. *Efficient circuit obfuscators do not exist (unconditionally).*

As stated above, our impossibility proof can be cast in terms of "inherently unbfuscatable functions":

Theorem 3.9 (inherently unobfuscatable functions). *If one-way functions exist, then there exists an inherently unobfuscatable function ensemble.*

Proof. Let $\mathcal{F}_k$ and $\mathcal{G}_k$ be the distributions on functions in the proof of Theorem 3.6,and let $\mathcal{H}_k$ be the distribution that, with probability 1/2 outputs a sample of $\mathcal{F}_k$ and with probability 1/2 outputs a sample of $\mathcal{G}_k$. We claim that $\{\mathcal{H}_k\}_{k\in\mathbb{N}}$ is an inherently unobfuscatable function ensemble.

The fact that $\{\mathcal{H}_k\}_{k\in\mathbb{N}}$ is efficiently computable is obvious. We define $\pi(f)$ to be 1 if $f \in \bigcup_k \mathrm{Supp}(\mathcal{F}_k)$ and 0 otherwise (note that $(\bigcup_k \mathrm{Supp}(\mathcal{F}_k)) \cap (\bigcup_k \mathrm{Supp}(\mathcal{G}_k)) = \emptyset$ and so $\pi(f) = 0$ for any $f \in \bigcup_k \mathrm{Supp}(\mathcal{G}_k)$). The algorithm A' given in the proof of Theorem 3.6 shows that $\pi(f)$ can be computed in polynomial time from any circuit computing $f \in \mathrm{Supp}(\mathcal{H}_k)$. Because oracle access to $\mathcal{F}_k$ cannot be distinguished from oracle access to $\mathcal{G}_k$ (as shown in the proof of Theorem 3.6), it follows that $\pi(f)$ cannot be computed from an oracle for $f \stackrel{\mathrm{R}}{\leftarrow} \mathcal{H}_k$ with probability noticeably greater than 1/2. ■

Acknowledgments

We are grateful to Luca Trevisan for collaboration at an early stage of this research. We also thank Dan Boneh, Ran Canetti, and Yacov Yacobi for helpful discussions and comments.

This work was partially supported by the following funds: Oded Goldreich was supported by the Minerva Foundation, Germany; Salil Vadhan (at the time at MIT) was supported by a DOD/NDSEG Graduate Fellowship and an NSF Mathematical Sciences Postdoctoral Research Fellowship.

References

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. Technical report, Electronic Colloquium on Computational Complexity, 2001. `http://www.eccc.uni-trier.de/eccc`.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the First Annual Conference on Computer and Communications Security*. ACM, November 1993.

[BL96] Dan Boneh and Richard Lipton. Algorithms for black-box fields and their applications to cryptography. In M. Wiener, editor, *Advances in Cryptology—CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 283–297. Springer-Verlag, August 1996.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 209–218, Dallas, 23–26 May 1998.

[CT00] Christian Collberg and Clark Thomborson. Watermarking, tamper-proofing, and obfuscation – tools for software protection. Technical Report TR00-03, The Department of Computer Science, University of Arizona, February 2000.

[DDN00] Danny Dolev, Cynthia Dwork, and Moni Naor. Nonmalleable cryptography. *SIAM Journal on Computing*, 30(2):391–437 (electronic), 2000.

[FM91] Joan Feigenbaum and Michael Merritt, editors. *Distributed computing and cryptography*, Providence, RI, 1991. American Mathematical Society.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Advances in cryptology—CRYPTO '86 (Santa Barbara, Calif., 1986)*, pages 186–194. Springer, Berlin, 1987.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the Association for Computing Machinery*, 33(4):792–807, 1986.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM*, 43(3):431–473, 1996.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, April 1984.

[Had00] Satoshi Hada. Zero-knowledge and code obfuscation. In T. Okamoto, editor, *Advances in Cryptology – ASIACRYPT ' 2000*, Lecture Notes in Computer Science, pages 443–457, Kyoto, Japan, 2000. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany.

[KY00] Jonathan Katz and Moti Yung. Complete characterization of security notions for private-key encryption. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, pages 245–254, Portland, OR, May 2000. ACM.

[NSS99] David Naccache, Adi Shamir, and Julien P. Stern. How to copyright a function? In H. Imai and Y. Zheng, editors, *Public Key Cryptography—PKC '99*, volume 1560 of *Lecture Notes in Computer Science*, pages 188–196. Springer-Verlag, March 1999.

[NR97] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th Annual Symposium on Foundations of Computer Science*, pages 458–467, Miami Beach, Florida, 20–22 October 1997. IEEE.

[RAD78] Ronald L. Rivest, Len Adleman, and Michael L. Dertouzos. On data banks and privacy homomorphisms. In *Foundations of secure computation (Workshop, Georgia Inst. Tech., Atlanta, Ga., 1977)*, pages 169–179. Academic, New York, 1978.

[SYY99] Thomas Sander, Adam Young, and Moti Yung. Non-interactive cryptocomputing for NC^1. In *40th Annual Symposium on Foundations of Computer Science*, pages 554–566, New York, NY, 17–19 October 1999. IEEE.

[vD98] Frans van Dorsselaer. Obsolescent feature. Winning entry for the *1998 International Obfuscated C Code Contest*, 1998. `http://www.ioccc.org/`.