On the (Im)possibility of Obfuscating Programs*

Boaz Barak† Oded Goldreich† Russell Impagliazzo‡ Steven Rudich§
Amit Sahai¶ Salil Vadhan‖ Ke Yang§

November 13, 2001

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 *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.

Keywords: cryptography, complexity theory, software protection, homomorphic encryption, Rice's Theorem, software watermarking, pseudorandom functions, statistical zero knowledge

*An extended abstract of this work is to appear in CRYPTO 2001 [BGI+01].

†Department of Computer Science, Weizmann Institute of Science, Rehovot, ISRAEL. E-mail: {boaz,oded}@wisdom.weizmann.ac.il

‡Department of Computer Science and Engineering, University of California, San Diego, La Jolla, CA 92093-0114. E-mail: russell@cs.ucsd.edu

§Computer Science Department, Carnegie Mellon University, 5000 Forbes Ave. Pittsburgh, PA 15213. E-mail: {rudich,yangke}@cs.cmu.edu

¶Department of Computer Science, Princeton University, 35 Olden St. Princeton, NJ 08540. E-mail: sahai@cs.princeton.edu

‖Division of Engineering and Applied Sciences, Harvard University, 33 Oxford Street, Cambridge, MA 02138. E-mail: salil@eecs.harvard.edu. URL: http://eecs.harvard.edu/~salil.

Contents

1 Introduction **2**
- 1.1 Some Applications of Obfuscators . . . 2
- 1.2 Our Results . . . 4
- 1.3 Discussion . . . 5
- 1.4 Additional Related Work . . . 6
- 1.5 Organization of the Paper . . . 6

2 Definitions **6**
- 2.1 Preliminaries . . . 6
- 2.2 Obfuscators . . . 7

3 The Main Impossibility Result **9**
- 3.1 Obfuscating two TMs/circuits . . . 10
- 3.2 Obfuscating one TM/circuit . . . 12

4 Extensions **16**
- 4.1 Totally unobfuscatable functions . . . 16
- 4.2 Approximate obfuscators . . . 17
- 4.3 Impossibility of the applications . . . 21
- 4.4 Obfuscating restricted circuit classes . . . 23
- 4.5 Relativization . . . 24

5 On a Complexity Analogue of Rice's Theorem **26**

6 Obfuscating Sampling Algorithms **28**

7 Weaker Notions of Obfuscation **30**

8 Watermarking and Obfuscation **32**

9 Directions for Further Work **34**

A Generalizing Rice's Theorem to Promise Problems. **37**

B Pseudorandom Oracles **40**

1 Introduction

The past two 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 ones choice). However, this informal statement is, of course, an over-generalization, 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.

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.

```
#include<stdio.h> #include<string.h>
main(){char*O,l[999]="’‘acgo\177~|xp .
-\OR^8)NJ6%K4O+A2M(*OID57$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);}}
```

Figure 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).

Homomorphic Encryption. A long-standing open problem in cryptography 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 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

[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(except in Section 6). This restriction only makes our main (impossibility) result stronger.

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.

Interestingly, in the original paper of Diffie and Hellman [DH76], the above was the reason given to believe that public-key cryptosystems might exist even though there were no candidates known yet. That is, they suggested that it might be possible to obfuscate a private-key encryption scheme.[3]

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 *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).

[3] From [DH76]: "A more practical approach to finding a pair of easily computed inverse algorithms E and D; such that D is hard to infer from E, makes use of the difficulty of analyzing programs in low level languages. Anyone who has tried to determine what operation is accomplished by someone else's machine language program knows that E itself (i.e. what E does) can be hard to infer from an algorithm for E. If the program were to be made purposefully confusing through the addition of unneeded variables and statements, then determining an inverse algorithm could be made very difficult. Of course, E must be complicated enough to prevent its identification from input-output pairs.

Essentially what is required is a one-way compiler: one which takes an easily understood program written in a high level language and translates it into an incomprehensible program in some machine language. The compiler is one-way because it must be feasible to do the compilation, but infeasible to reverse the process. Since efficiency in size of program and run time are not crucial in this application, such compilers may be possible if the structure of the machine language can be optimized to assist in the confusion."

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 *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 pseudorandom 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. Up to 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. 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. Toward this end, 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 presenting 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. This restriction should not be confined to the computational complexity of the 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 explicit 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 unobfuscatable function ensemble. In Section 4, we give a number of extensions of our impossibility result, including impossibility results for obfuscators which only need to approximately preserve functionality, for obfuscators computable in low circuit classes, and for some of the applications of obfuscators. We also show that our main impossibility result does not relativize. In Section 5, we discuss some conjectural complexity-theoretic analogues of Rice's Theorem, and use our techniques to show that one of these is false. In Section 6, we examine notions of obfuscators for *sampling* algorithms. In Section 7, we propose weaker notions of obfuscation that are not ruled out by our impossibility results. In Section 8, we discuss the problem of software watermarking and its relation to obfuscation. Finally, in Section 9, we mention some directions for further work in this area.

2 Definitions

2.1 Preliminaries

Standard Notations. *TM* is shorthand for Turing machine. *PPT* is shorthand for probabilistic polynomial-time Turing machine. By *circuit* we refer to a standard boolean circuit with AND,OR and NOT gates. If C is a circuit with n inputs and m outputs, and $x \in \{0,1\}^n$ then by $C(x)$ we denote the result of applying C on input x. We say that C computes a function $f : \{0,1\}^n \to \{0,1\}^m$

if for any $x \in \{0,1\}^n$, $C(x) = f(x)$. 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. When M is a circuit, this carries the standard meaning (on answer to oracle query x, A receives $M(x)$). When M is a TM, this means that A can make oracle queries of the form $(x, 1^t)$ and receive in response either the output of M on input x (if M halts within t steps on x), or $\perp$ (if M does not halt within t steps on x).[4] 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 $x \stackrel{R}{\leftarrow} D$ we mean that x is a random variable distributed according to D. If S is a set then by $x \stackrel{R}{\leftarrow} S$ 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 $\{0,1\}^*$.

Nonstandard Notations. If M is a TM then we denote by $\langle M \rangle$ the function $\langle M \rangle : 1^* \times \{0,1\}^* \to \{0,1\}^*$ given by:

$$\langle M \rangle(1^t, x) \stackrel{\mathrm{def}}{=} \begin{cases} y & M(x) \text{ halts with output } y \text{ after at most } t \text{ steps} \\ \perp & \text{otherwise} \end{cases}$$

If C is a circuit then we denote by $[C]$ the function it computes. Similarly if M is a TM then we denote by $[M]$ the (possibly partial) function it computes.

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.

[4] In typical cases (i.e., when the running time is *a priori* bounded), this convention makes our definitions of obfuscator even weaker since it allows A to learn the actual running-time of M on particular inputs. This seems the natural choice because a machine given the code of M can definitely learn its actual running-time on inputs of its own choice.

- (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, $|\mathcal{O}(M)| \leq p(|M|)$, and if M halts in t steps on some input x, then $\mathcal{O}(M)$ halts within $p(t)$ steps on x.*
- *("virtual black box" property) For any PPT A, there is a PPT S and a negligible function α such that for all TMs M*

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

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, $|\mathcal{O}(C)| \leq p(|C|)$.*
- *("virtual black box" property) For any PPT A, there is a PPT S and a negligible function α such that for all circuits C*

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

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 one 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 unobfuscatable function ensemble.

Definition 3.1 *An* 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 $\{0,1\}^{l_{\text{out}}(k)}$) satisfying:*

- *(efficient computability) 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 $f \in \bigcup_{k\in\mathbb{N}} \text{Supp}(\mathcal{H}_k)$ and for any circuit C that computes f*

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

We prove in Theorem 3.11 that, assuming one-way functions exist, there exists an 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 3.8). Since the existence of an *efficient* obfuscator implies the existence of one-way functions (Lemma 3.9), we conclude that efficient obfuscators do not exist (unconditionally).

However, the existence of 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 3.11).

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*

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

Definition 3.3 (2-circuit obfuscator) *A* 2-circuit obfuscator *is defined in the same way as a circuit obfuscator, except that the "virtual black box" property is replaced with the following:*

- *("virtual black box" property) For any PPT A, there is a PPT S and a negligible function α such that for all circuits C, D*

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

Proposition 3.4 *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 $\text{poly}(k)$ steps, and outputs 0 if C does not halt within that many steps, for some fixed polynomial $\text{poly}(\cdot)$. 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 $\text{poly}(|C|, |D|)$ 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 $\text{poly}(k)$-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 $\text{poly}(10 \cdot k) = \text{poly}(k)$ steps, which means that $D_{\alpha,\beta}(\mathcal{O}(C_{\alpha,\beta}))$ need only run for $\text{poly}(k)$ steps. Thus, again applying the polynomial slowdown property, execution (b) takes $\text{poly}(k)$ steps, which finally implies that A need only run for $\text{poly}(k)$ steps. The same reasoning

holds for Equation (3), using Z_k instead of $C_{\alpha,\beta}$.[5] 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 $\text{poly}(k)$ 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 $(f_0 \# f_1)(b, x) \stackrel{\text{def}}{=} f_b(x)$. Conversely, if we are given a TM (resp., circuit) $C : \{0,1\} \times X \to Y$, we can 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 3.5 *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 3.4. 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 3.4, 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 3.4, A actually should be modified to run in time $\text{poly}(|F|)$.) Let S be the PPT simulator for A guaranteed by Definition 2.1. Just as in the proof of Proposition 3.4, we have:

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

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. □

[5] 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 $\Omega(k)$.

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 3.6 *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 $D \in \mathrm{Supp}(\mathcal{D}_{\alpha,\beta})$ is a circuit of size $\mathrm{poly}(k)$.*
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 *nonadaptive 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 $r \stackrel{\mathrm{R}}{\leftarrow} \{0,1\}^{|K|}$ 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

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

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 3.7 *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 afterward).

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 (deterministic) 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 3.7) and a hybrid argument. ■

Now we can prove the impossibility of circuit obfuscators.

Theorem 3.8 *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 3.4, and let $\mathcal{D}_{\alpha,\beta}$ be the distribution on circuits given by Lemma 3.6. For 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 $\{0,1\}^k$, $D \stackrel{\mathrm{R}}{\leftarrow} \mathcal{D}_{\alpha,\beta}$. Output $C_{\alpha,\beta}\#D$.

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

Let A be the PPT algorithm guaranteed by Property 2 in Lemma 3.6, 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 3.6, we have:

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 $D \stackrel{\mathrm{R}}{\leftarrow} \mathcal{D}_{\alpha,\beta}$, where α, β are selected uniformly in $\{0,1\}^k$. Property 3 of Lemma 3.6 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.

Lemma 3.9 *If efficient obfuscators exist, then one-way functions exist.*

Proof Sketch: Suppose that $\mathcal{O}$ is an efficient obfuscator as per Definition 2.2. For $\alpha \in \{0,1\}^k$ and $b \in \{0,1\}$, let $C_{\alpha,b} : \{0,1\}^k \to \{0,1\}$ be the circuit defined by

$$C_{\alpha,b}(x) \stackrel{\mathrm{def}}{=} \begin{cases} b & x = \alpha \\ 0 & \text{otherwise.} \end{cases}$$

Now define $f_k(\alpha, b, r) \stackrel{\text{def}}{=} \mathcal{O}(C_{\alpha,b}; r)$, i.e. the obfuscation of $C_{\alpha,b}$ using coin tosses r. We will argue that $f = \bigcup_{k\in\mathbb{N}} f_k$ is a one-way function. Clearly f_k can be evaluated in time poly(k). Since the bit b is information-theoretically determined by $f_k(\alpha, b, r)$, to show that f is one-way it suffices to show that b is a *hard-core bit* of f. To prove this, we first observe that for any PPT S,

$$\Pr_{\alpha,b}\left[S^{C_{\alpha,b}}(1^k) = b\right] \leq \frac{1}{2} + \text{neg}(k).$$

By the virtual black box property of $\mathcal{O}$, it follows that for any PPT A,

$$\Pr_{\alpha,b,r}\left[A(f(\alpha, b, r)) = b\right] = \Pr_{\alpha,b,r}\left[A(\mathcal{O}(C_{\alpha,b}; r)) = b\right] \leq \frac{1}{2} + \text{neg}(k).$$

This demonstrates that b is indeed a hard-core bit of f, and hence that f is one-way. □

Corollary 3.10 *Efficient circuit obfuscators do not exist (unconditionally).*

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

Theorem 3.11 (unobfuscatable functions) *If one-way functions exist, then there exists an unobfuscatable function ensemble.*

Proof: Let $\mathcal{F}_k$ and $\mathcal{G}_k$ be the distributions on functions in the proof of Theorem 3.8,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 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 $f \in \bigcup_k \text{Supp}(\mathcal{F}_k)$ and 0 otherwise (note that $(\bigcup_k \text{Supp}(\mathcal{F}_k)) \cap (\bigcup_k \text{Supp}(\mathcal{G}_k)) = \emptyset$ and so $\pi(f) = 0$ for any $f \in \bigcup_k \text{Supp}(\mathcal{G}_k)$). The algorithm A' given in the proof of Theorem 3.8 shows that $\pi(f)$ can be computed in polynomial time from any circuit computing $f \in \text{Supp}(\mathcal{H}_k)$. Because oracle access to $\mathcal{F}_k$ cannot be distinguished from oracle access to $\mathcal{G}_k$ (as shown in the proof of Theorem 3.8), it follows that $\pi(f)$ cannot be computed from an oracle for $f \stackrel{\text{R}}{\leftarrow} \mathcal{H}_k$ with probability noticeably greater than 1/2. ■

4 Extensions

4.1 Totally unobfuscatable functions

Some of the extensions of our impossibility result require a somewhat stronger form of unobfuscatable functions, in which it is not only possible to compute $\pi(f)$ from any circuit for f, but even to recover the "original" circuit for f. This can be achieved by a slight modification of our construction. It will also be useful to extend the construction so that not only the one bit $\pi(f)$ is unpredictable given oracle access to f, but rather that there are many bits of information about f which are completely pseudorandom. These properties are captured by the definition below. In this definition, it will be convenient to identify the functions f in our family with the canonical circuits that compute them.

Definition 4.1 *A* totally unobfuscatable function ensemble *is an ensemble* $\{\mathcal{H}_k\}_{k\in\mathbb{N}}$ *of distributions* $\mathcal{H}_k$ *on* circuits *(from, say,* $\{0,1\}^{l_{\text{in}}(k)}$ *to* $\{0,1\}^{l_{\text{out}}(k)}$*) satisfying:*

- *(efficient computability) Every circuit* $f \in \text{Supp}(\mathcal{H}_k)$ *is of size* poly(k)*. Moreover,* $f \stackrel{R}{\leftarrow} \text{Supp}(\mathcal{H}_k)$ *can be sampled uniformly in time* poly(k)*.*

- *(unobfuscatability) There exists a poly-time computable function* $\pi : \bigcup_{k\in\mathbb{N}} \mathrm{Supp}(\mathcal{H}_k) \to \{0,1\}^*$, *such that*

  1. $\pi(f)$ *is pseudorandom given black-box access to* f*: For any PPT* S

  $$\left| \Pr_{f \xleftarrow{R} \mathcal{H}_k} [S^f(\pi(f)) = 1] - \Pr_{f \xleftarrow{R} \mathcal{H}_k, z \xleftarrow{R} \{0,1\}^k} [S^f(z) = 1] \right| \leq \mathrm{neg}(k)$$

  2. f *is easy to reconstruct given any other circuit for* f*: There exists a PPT* A *such that for any* $f \in \bigcup_k \mathrm{Supp}(\mathcal{H}_k)$ *and for any circuit* C *that computes the same function as* f

  $$A(C) = f$$

  ,

Note that totally unobfuscatable functions imply unobfuscatable functions: given oracle access to a totally unobfuscatable f, pseudorandomness implies that the first bit of $\pi(f)$ cannot be computed with probability noticeably more than $1/2$, and given any circuit for f, one can efficiently find the canonical circuit for f, from which one can compute $\pi(f)$ (and in particular, its first bit).

Theorem 4.2 (totally unobfuscatable functions) *If one-way functions exist, then there exists a totally unobfuscatable function ensemble.*

Proof Sketch: The first step is to observe that the ensemble $\mathcal{D}_{\alpha,\beta}$ of Lemma 3.6 can be modified so that Property 2 instead says $A^D(C, 1^k) = \alpha$ if $C(\alpha) = \beta$ and $A^D(C, 1^k) = 0^k$ otherwise. (To achieve this, replace $B_{K,\beta}$ with $B'_{K,\alpha,\beta}$ which outputs α when fed a sequence of ciphertexts $(c_1, \ldots, c_k)$ whose decryptions are the bits of β and outputs 0^k otherwise.)

Now our totally unobfuscatable function ensemble $\mathcal{H}_k$ is defined as follows.

$\mathcal{H}_k$**:** Choose α, β, γ uniformly in $\{0,1\}^k$, $D \xleftarrow{\mathrm{R}} \mathcal{D}_{\alpha,\beta}$. Output $C_{\alpha,\beta}\#D\#C_{\alpha,(D,\gamma)}$.

(Above, $C_{\alpha,(D,\gamma)}$ is the circuit which on input α outputs (D, γ), and on all other inputs outputs $0^{|(D,\gamma)|}$.)

Efficiency is clearly satisfied. For unobfuscatability, we define $\pi(C_{\alpha,\beta}\#D\#C_{\alpha,(D,\gamma)}) = \gamma$. Let's verify that γ is pseudorandom given oracle access. As in the proof of Theorem 3.11, it follows from Property 3 of Lemma 3.6 that a PPT algorithm given oracle access to $C_{\alpha,\beta}\#D\#C_{\alpha,(D,\gamma)}$. will only query $C_{\alpha,(D,\gamma)}$ with negligible probability and hence γ is indistinguishable from uniform.

Finally, let's show that given any circuit C' computing the same function as $C_{\alpha,\beta}\#D\#C_{\alpha,(D,\gamma)}$, we can reconstruct the latter circuit. First, we can decompose $C' = C^1\#D'\#C^2$. Since D' computes the same function as D and $C^1(\alpha) = \beta$, we have $A^{D'}(C^1) = \alpha$, where A is the algorithm from (the modified) Property 2 of Lemma 3.6. Given α, we can obtain $\beta = C^1(\alpha)$ and $(D, \gamma) = C^2(\alpha)$, which allows us to reconstruct $C_{\alpha,\beta}\#D\#C_{\alpha,(D,\gamma)}$. □

4.2 Approximate obfuscators

One of the most reasonable ways to weaken the definition of obfuscators, is to relax the condition that the obfuscated circuit must compute *exactly* the same function as the original circuit. Rather, we can allow the obfuscated circuit to only *approximate* the original circuit.

We must be careful in defining "approximation". We do not want to lose the notion of an obfuscator as a *general purpose* scrambling algorithm and therefore we want a definition of approximation that will be strong enough to guarantee that the obfuscated circuit can still be used in the place of the original circuit in *any application*. Consider the case of a signature verification algorithm V_K. A polynomial-time algorithm cannot find an input on which V_K does not output 0 (without knowing the signature key). However, we clearly do not want this to mean that the constant zero function is an approximation of V_K.

4.2.1 Definition and Impossibility Result

In order to avoid the above pitfalls we choose a definition of approximation that allows the obfuscated circuit to deviate on a particular input from the original circuit only with negligible probability and allows this event to depend on only the coin tosses of the obfuscating algorithm (rather than over the choice of a randomly chosen input).

Definition 4.3 *For any function $f : \{0,1\}^n \to \{0,1\}^k$, $\epsilon > 0$, the random variable C is called an* ϵ-approximate implementation *of f if the following holds:*

1. *C ranges over circuits from $\{0,1\}^n$ to $\{0,1\}^k$*
2. *For any $x \in \{0,1\}^n$, $\Pr_C[C(x) = f(x)] \geq 1 - \epsilon$*

We then define a strongly unobfuscatable function ensemble to be an unobfuscatable function ensemble where the hard property $\pi(f)$ can be computed not only from any circuit that computes f but also from any approximate implementation of f.

Definition 4.4 *A* strongly unobfuscatable function ensemble $\{\mathcal{H}_k\}_{k\in\mathbb{N}}$ *is defined in the same way as an unobfuscatable function ensemble, except that Part 2 of the "unobfuscatability" condition is replaced with the following:*

2. *$\pi(f)$ is easy to compute with access to a circuit that approximates f: There exists a PPT A and a polynomial $p(\cdot)$ such that for any $f \in \bigcup_{n\in\mathbb{N}} \mathrm{Supp}(\mathcal{H}_n)$ and for any random variable C that is an ϵ-approximate implementation of f*

$$\Pr[A(C) = \pi(f)] \geq 1 - \epsilon \cdot p(n)$$

Our main theorem in this section is the following:

Theorem 4.5 *If one-way functions exist, then there exists a strongly unobfuscatable function ensemble.*

Similarly to the way that Theorem 3.11 implies Theorem 3.8, Theorem 4.5 implies that, assuming the existence of one-way functions, an even weaker definition of circuit obfuscators (one that allows the obfuscated circuit to only approximate the original circuit) is impossible to meet. We note that it some (but not all) applications of obfuscators, a weaker notion of approximation might suffice. Specifically, in some cases it suffices for the obfuscator to only approximately preserve functionality with respect to a particular distribution on inputs, such as the uniform distribution. We do not know whether such obfuscators exist, and leave it as an open problem.

We shall prove this theorem in the following stages. First we will see why the proof of Theorem 3.11 does not apply directly to the case of approximate implementations. Then we shall define a construct called *invoker-randomizable pseudorandom functions*, which will help us modify the original proof to hold in this case.

4.2.2 Generalizing the Proof of Theorem 3.11 to the Approximate Case

The first question is whether the proof of Theorem 3.11 already shows that the ensemble $\{\mathcal{H}_k\}_{k\in\mathbb{N}}$ defined there is actually a *strongly* unobfuscatable function ensemble. As we explain below, the answer is no.

To see why, let us recall the definition of the ensemble $\{\mathcal{H}_k\}_{k\in\mathbb{N}}$ that is defined there and uses the distributions $\mathcal{F}_k$ and $\mathcal{G}_k$ that are defined in the proof of Theorem 3.8. The distribution $\mathcal{H}_k$ is defined by taking an element from $\mathcal{F}_k$ or $\mathcal{G}_k$, with probability 1/2 each. The distribution $\mathcal{F}_k$ is defined by choosing $\alpha, \beta \stackrel{R}{\leftarrow} \{0,1\}^k$, a function $D \stackrel{R}{\leftarrow} \mathcal{D}_{\alpha,\beta}$ and outputting $C_{\alpha,\beta}\#D$. Similarly, $\mathcal{G}_k$ is defined by choosing $\alpha, \beta \stackrel{R}{\leftarrow} \{0,1\}^k$, $D \stackrel{R}{\leftarrow} \mathcal{D}_{\alpha,\beta}$ and outputting $Z_k\#D$. The property π is defined simply to distinguish functions in $\mathcal{F}_k$ from those in $\mathcal{G}_k$.

That proof gave an algorithm A' which computes $\pi(f)$ given a circuit computing any function f from $\mathcal{H}$. Let us see why A' might fail when given only an approximate implementation of f. On input a circuit F, A' works as follows: It decomposes F into two circuits $F = F_1\#F_2$. F_2 is used only in a black-box manner, but the queries A' makes to it depend on the gate structure of the circuit F_1. The problem is that a vicious approximate implementation for a function $C_{\alpha,\beta}\#D \in \mathrm{Supp}(\mathcal{F}_k)$ may work in the following way: choose a random circuit F_1 out of some set $\mathcal{C}$ of exponentially many circuits that compute $C_{\alpha,\beta}$, and take F_2 that computes D. Then see at which points A' queries F_2 when given $F_1\#F_2$ as input.[6] As these places depend on F_1, it is possible that for each $F_1 \in \mathcal{C}$, there is a point $x(F_1)$ such that A' will query F_2 at the point $x(F_1)$, but $x(F_1) \neq x(F_1')$ for any $F_1' \in \mathcal{C} \setminus \{F_1\}$. If the approximate implementation changes the value of F_2 at $x(F_1)$, then A''s computation on $F_1\#F_2$ is corrupted.

One way to solve this problem would be to make the queries that A' makes to F_2 *independent* of the structure of F_1. If A' had this property, then given an ϵ-approximate implementation of $C_{\alpha,\beta}\#D$, each query of A' would have only an ϵ chance to get an incorrect answer and overall A' would succeed with probability $1 - \epsilon \cdot p(k)$ for some polynomial $p(\cdot)$. (Note that the probability that $F_1(\alpha)$ changes is at most ϵ.)

We will not be able to achieve this, but something slightly weaker that still suffices. Let's look more closely at the structure of $\mathcal{D}_{\alpha,\beta}$ which is defined in the proof of Lemma 3.6. We defined there the algorithm

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

and turned it into a deterministic function by using a pseudorandom function f'_K and defining $D'_{K,\alpha,\beta,K'}$ to be the deterministic algorithm that on input $x \in \{0,1\}^q$ evaluates $D_{K,\alpha,\beta}(x)$ using randomness $f_{K'}(x)$. We then defined $\mathcal{D}_{\alpha,\beta}$ to be $D'_{K,\alpha,\beta,K'} = E'_{K,\alpha,K'}\#\mathrm{Hom}'_{K,K'}\#B_{K,\beta}$ for uniformly selected private key K and seed K'.

Now our algorithm A' (that uses the algorithm A defined in Lemma 3.6) treats F_2 as three oracles: E, H, and B , where if F_2 computes $D = E'_{K,\alpha,K'}\#\mathrm{Hom}'_{K,K'}\#B_{K,\beta}$ then E is the oracle to $E'_{K,\alpha,K'}$, H is the oracle to $\mathrm{Hom}'_{K,K'}$ and B is the oracle to $B_{K,\beta}$. The queries to E are at the places $1,\ldots,k$ and so are independent of the structure of F_1. The queries that A makes to the H oracle, however, do depend on the structure of F_1.

Recall that any query A' makes to the H oracle are of the form $(c,d,\odot)$ where c and d are ciphertexts of some bits, and $\odot$ is a 4-bit description of a binary boolean function. Just for motivation, suppose that A' has the following ability: given an encryption c, A' can generate a random encryption of the same bit (i.e., distributed according to $\mathrm{Enc}_K(\mathrm{Dec}_K(c),r)$ for uniformly

[6] Recall that A' is not some given algorithm that we must treat as a black-box but rather a specific algorithm that we defined ourselves.

selected r). For instance, this would be true if the encryption scheme were "random self-reducible." Suppose now that, before querying the H oracle with $(c, d, \odot)$, A' generates c', d' that are random encryptions of the same bits as c, d and query the oracle with $(c', d', \odot)$ instead. We claim that if F_2 is an ϵ-approximate implementation of D, then for any such query, there is at most a 64ϵ probability for the answer to be wrong *even if* $(c, d, \odot)$ *depend on the circuit* F. The reason is that the distribution of the modified query $(c', d', \odot)$ depends only on $(\mathrm{Dec}_K(c), \mathrm{Dec}_K(d), \odot)$, and there are only $2 \cdot 2 \cdot 2^4 = 64$ possibilities for the latter. For each of the 64 possibilities, the probability of an incorrect answer (over the choice of F) is at most ϵ. Choosing $(\mathrm{Dec}_K(c), \mathrm{Dec}_K(d), \odot)$ after F to maximize the probability of an incorrect answer multiplies this probability by at most 64.

We shall now use this motivation to fix the function D so that A' will essentially have this desired ability of randomly self-reducing any encryption to a random encryption of the same bit. Recall that $\mathrm{Hom}'_{K,K'}(c, d, \odot) = \mathrm{Enc}_K(\mathrm{Dec}_K(c) \odot \mathrm{Dec}_K(d); f_{K'}(c, d, \odot))$. Now, a naive approach to ensure that any query returns a random encryption of $\mathrm{Dec}_K(c) \odot \mathrm{Dec}_K(d)$ would be to change the definition of Hom' to the following: $\mathrm{Hom}'_{K,K'}(c, d, \odot, r) = \mathrm{Enc}_K(\mathrm{Dec}_K(c) \odot \mathrm{Dec}_K(d); r)$. Then we change A' to an algorithm A'' that chooses a uniform $r \in \{0,1\}^n$ and thereby ensures that the result is a random encryption of $\mathrm{Dec}_K(c) \odot \mathrm{Dec}_K(d)$. The problem is that this construction would no longer satisfy Property 3 of Lemma 3.6 (security against a simulator with oracle access). This is because the simulator could now control the random coins of the encryption scheme and use this to break it. Our solution will be to redefine Hom' in the following way:

$$\mathrm{Hom}'_{K,K'}(c, d, \odot, r) = \mathrm{Enc}_K(\mathrm{Dec}_K(c) \odot \mathrm{Dec}_K(d); f_{K'}(c, d, \odot, r))$$

but require an additional special property from the pseudorandom function $f_{K'}$.

4.2.3 Invoker-Randomizable Pseudorandom Functions

The property we would like the pseudorandom function $f_{K'}$ to possess is the following:

Definition 4.6 *A function ensemble* $\{f_{K'}\}_{K' \in \{0,1\}^*}$ *(*$f_{K'} : \{0,1\}^{q+n} \to \{0,1\}^n$ *,* n *,*q *polynomially related to* $|K'|$*) is called an* invoker-randomizable pseudorandom function ensemble *if the following holds:*

1. $\{f_{K'}\}_{K' \in \{0,1\}^*}$ *is a pseudorandom function ensemble*
2. *For any* $x \in \{0,1\}^q$ *, if* r *is chosen uniformly in* $\{0,1\}^n$ *then* $f_{K'}(x, r)$ *is distributed uniformly (and so independently of* x*) in* $\{0,1\}^n$*.*

Fortunately, we can prove the following lemma:

Lemma 4.7 *If pseudorandom functions exist then there exist invoker-randomizable pseudorandom functions.*

Proof Sketch: Suppose that $\{g_{K'}\}_{K' \in \{0,1\}^*}$ is a pseudorandom function ensemble and that $\{p_S\}_{S \in \{0,1\}^*}$ is a pseudorandom function ensemble in which for any $S \in \{0,1\}^*$, p_S is a permutation (the existence of such ensembles is implied by the existence of ordinary pseudorandom function ensembles [LR88]).

We define the function ensemble $\{f_{K'}\}_{K' \in \{0,1\}^*}$ in the following way:

$$f_{K'}(x, r) \stackrel{\mathrm{def}}{=} p_{g_{K'}(x)}(r)$$

It is clear that this ensemble satisfies Property 2 of Definition 4.6 as for any x, the function $r \mapsto f_{K'}(x,r)$ is a permutation.

What needs to be shown is that it is a pseudorandom function ensemble. We do this by showing that for any PPT D, the following probabilities are identical up to a negligible factor.

1. $\Pr_{K'}[D^{f_{K'}}(1^k) = 1]$ (where $k = |K'|$).
2. $\Pr_G[D^{(x,R)\mapsto p_{G(x)}(R)}(1^k) = 1]$, where G is a true random function.
3. $\Pr_{P_1,\ldots,P_t}[D^{P_1,\ldots,P_t}(1^k) = 1]$, where $t = t(k)$ is a bound on the number of queries that D makes and each time D makes a query with a new value of x we use a new random function P_i. (This requires a hybrid argument).
4. $\Pr_F[D^F(1^k) = 1]$, where F is a truly random function.

□

4.2.4 Finishing the Proof of Theorem 4.5

Now, suppose we use a pseudorandom function $f_{K'}$ that is invoker-randomizable, and modify the algorithm A' so that all its queries $(c, d, \odot)$ to the H oracle are augmented to be of the form $(c, d, \odot, r)$, where r is chosen uniformly and independently for each query. Then the result of each such query is a *random* encryption of $\mathrm{Dec}_K(c) \odot \mathrm{Dec}_K(d)$. Therefore, as argued above, A' never gets a wrong answer from the H oracle with probability at least $1 - p(k) \cdot \epsilon$, for some polynomial $p(\cdot)$. Indeed, this holds because aside from the first queries which are fixed and therefore independent of the gate structure of F_1, all other queries are of the form $(c, d, \odot, r)$ where c and d are uniformly distributed and independent encryptions of some bits a and b, and r is uniformly distributed. Only $(a, b, \odot)$ depend on the gate structure of F_1, and there are only 64 possibilities for them. Assuming A' never gets an incorrect answer from the H oracle, its last query to the B oracle will be a uniformly distributed encryption of $\beta_1, \ldots, \beta_k$, which is independent of the structure of F_1, and so has only an ϵ probability to be incorrect. This completes the proof.

One point to note is that we have converted our deterministic algorithm A' of Theorem 3.11 into a *probabilistic* algorithm.

4.3 Impossibility of the applications

So far, we have only proved impossibility of some natural and arguably minimalistic definitions for obfuscation. Yet it might seem that there's still hope for a different definition of obfuscation, one that will not be impossible to meet but would still be useful for some intended applications. We'll show now that this is not the case for many of the applications we described in the introduction. Rather, any definition of obfuscator that would be strong enough to provide them, will be impossible to meet.

Note that we do not prove that the applications themselves are impossible to meet, but rather that there does not exist an obfuscator[7] that can be used to achieve them in the ways that are described in Section 1.1. Our results in the section also extend to approximate obfuscators.

Consider, for example, the application to transforming private-key encryption to public-key ones. The circuit $\widetilde{E_k}$ in the following definition can be viewed as an encryption-key in the corresponding public-key encryption scheme.

[7]By this, we mean any algorithm that satisfies the syntactic requirements of Definition 2.2 (functionality and polynomial slowdown).

Definition 4.8 *A private-key encryption scheme (G, E, D) is called* unobfuscatable *if there exists a PPT A such that*

$$\Pr_{K \stackrel{R}{\leftarrow} G(1^k)}[A(\widetilde{E_K}) = K] \geq 1 - \mathrm{neg}(k)$$

where $\widetilde{E_K}$ is any circuit that computes the encryption function with private key K.

Note that an unobfuscatable encryption scheme is unobfuscatable in a very strong sense. An adversary is able to completely break the system given *any* circuit that computes the encryption algorithm.

We prove in Theorem 4.12 that if encryption schemes exist, then so do unobfuscatable encryption schemes that satisfy the same security requirements.[8] This means that any definition of an obfuscators that will be strong enough to allow the conversion of private-key encryption schemes into public-key encryption schemes mentioned in Section 1.1, would be impossible to meet (because there exist unobfuscatable encryption schemes).[9]

We present analogous definitions for unobfuscatable signature schemes, MACs, and pseudorandom functions.

Definition 4.9 *A signature scheme (G, S, V) is called* unobfuscatable *if there exists a PPT A such that*

$$\Pr_{(SK,VK) \stackrel{R}{\leftarrow} G(1^k)}[A(\widetilde{S_{SK}}) = SK] \geq 1 - \mathrm{neg}(k)$$

where $\widetilde{S_{SK}}$ is any circuit which computes the signature function with signing key SK.

Definition 4.10 *A message authentication scheme (G, S, V) is called* unobfuscatable *if there exists a PPT A such that*

$$\Pr_{K \stackrel{R}{\leftarrow} G(1^k)}[A(\widetilde{S_K}) = K] \geq 1 - \mathrm{neg}(k)$$

where $\widetilde{S_K}$ is any circuit which computes the tagging function with tagging key K.

Definition 4.11 *A pseudorandom function ensemble $\{h_K\}_{K \in \{0,1\}^*}$ is called* unobfuscatable *if there exists a p.p.t A such that*

$$\Pr_{K \stackrel{R}{\leftarrow} \{0,1\}^k}[A(\widetilde{H_K}) = K] \geq 1 - \mathrm{neg}(k)$$

where $\widetilde{H_K}$ is any circuit that computes h_K.

One implication of the existence of unobfuscatable pseudorandom function ensembles is that for many *natural* protocols that are secure in the random oracle model (such as the Fiat–Shamir authentication protocol [FS87]), one can find a pseudorandom function ensemble $\{h_k\}_{k \in \{0,1\}^*}$ such that if the random oracle is replaced with *any* circuit that computes h_k, the protocol would not be secure.

[8] Recall that, for simplicity, we only consider deterministic encryption schemes here and relaxed notions of security that are consistent with them (cf., Footnote 2).

[9] Of course, this does not mean that public-key encryption schemes do not exist, nor that there do not exist private-key encryption schemes where one can give the adversary a circuit that computes the encryption algorithm without loss of security (indeed, any public-key encryption scheme is in particular such a private-key encryption). What this means is that there exists no general purpose way to transform a private key encryption scheme into a public key encryption by obfuscating the encryption algorithm.

Theorem 4.12 *1. If signature schemes exist, then so do unobfuscatable signature schemes.*

2. If private-key encryption schemes exist, then so do unobfuscatable encryption schemes.

3. If pseudorandom function ensembles exist, then so do unobfuscatable pseudorandom function ensembles.

4. If message authentication schemes exist, then so do unobfuscatable message authentication schemes.

Proof Sketch: First note that the existence of any one of these primitives implies the existence of one-way functions [IL89]. Therefore, Theorem 4.2 gives us a totally unobfuscatable function ensemble $\mathcal{H} = \{\mathcal{H}_k\}$.

Now, we shall sketch the construction of the unobfuscatable signature scheme. All other constructions are similar. Take an existing signature scheme (G, S, V) (where G is the key generation algorithm, S the signing algorithm, and V the verification algorithm). Define the new scheme (G', S', V') as follows:

The generator G' on input 1^k uses the generator G to generate signing and verifying keys $(SK, VK) \stackrel{R}{\leftarrow} G(1^k)$. It then samples a circuit $f \stackrel{R}{\leftarrow} \mathcal{H}_\ell$, where $\ell = |SK|$. The new signing key SK' is (SK, f) while the verification key VK' is the same as VK.

We can now define

$$S'_{SK,f}(m) \stackrel{\text{def}}{=} (S_{SK}(m), f(m), SK \oplus \pi(f)),$$

where π is the function from the unobfuscatability condition in Definition 4.1.

$$V'_{VK}(m, (\tau, x)) \stackrel{\text{def}}{=} V_{VK}(m, \tau)$$

We claim that (G', S', V') is an unobfuscatable, yet secure, signature scheme. Clearly, given any circuit that computes $S'_{SK,f}$, one can obtain $SK \oplus \pi(f)$ and a circuit that computes the same function as f. Possession of the latter enables one to reconstruct the original circuit f itself, from which $\pi(f)$ and then SK can be computed.

To see that scheme (G', S', V') retains the security of the scheme (G, S, V), observe that being given oracle access to $S'_{SK,f}$ is equivalent to being given oracle access to S_{SK} and f, along with being given the string $\pi(f) \oplus SK$. Using the facts that $\pi(f)$ is indistinguishable from random given oracle access to f and that f is chosen independently of SK, it can be easily shown that the presence of f and $\pi(f) \oplus SK$ does not help an adversary break the signature scheme.

The construction of an unobfuscatable encryption scheme and pseudorandom function ensemble is similar. The only detail is that when we construct the pseudorandom function ensemble, we need to observe that Theorem 4.2 can be modified to give $\mathcal{H}$ which is also a family of pseudorandom functions. (To do this, all places where the functions f in $\mathcal{H}$ were defined to be zero should instead be replaced with values of a pseudorandom function.) □

4.4 Obfuscating restricted circuit classes

Given our impossibility results for obfuscating general circuits, one may ask whether it is easier to obfuscate computationally restricted classes of circuits. Here we argue that this is unlikely for all but very weak models of computation.

Theorem 4.13 *If factoring Blum integers is "hard"*[10] *then there is a family $\mathcal{H}_k$ of unobfuscatable functions such that every $f \stackrel{R}{\leftarrow} \mathcal{H}_k$ is computable by a constant-depth threshold circuit of size* $\mathrm{poly}(k)$ *(i.e., in* $\mathbf{TC^0}$*).*

Proof Sketch: Naor and Reingold [NR97] showed that under the stated assumptions, there exists a family of pseudorandom functions computable in $\mathbf{TC^0}$. Thus, we simply need to check that we can build our unobfuscatable functions from such a family without a substantial increase in depth. Recall that the unobfuscatable function ensemble $\mathcal{H}_k$ constructed in the proof of Theorem 3.11 consists of functions of the form $C_{\alpha,\beta}\#D$ or $Z_k\#D$, where D is from the family $\mathcal{D}_{\alpha,\beta}$ of Lemma 3.6. It is easy to see that $C_{\alpha,\beta}$ and Z_k are in $\mathbf{TC^0}$, so we only need to check that $\mathcal{D}_{\alpha,\beta}$ consists of circuits in $\mathbf{TC^0}$. The computational complexity of circuits in the family $\mathcal{D}_{\alpha,\beta}$ is dominated by performing encryptions and decryptions in a private-key encryption scheme $(\mathrm{Enc}, \mathrm{Dec})$ and evaluating a pseudorandom function $f_{K'}$ which is used to derandomize the probabilistic circuit $D_{K,\alpha,\beta}$. If we use the Naor–Reingold pseudorandom functions both for $f_{K'}$ and to construct the encryption scheme (in the usual way, setting $\mathrm{Enc}_K(b) = (r, f_K(r) \oplus b)$), then the resulting circuit is in $\mathbf{TC^0}$. □

4.5 Relativization

In this section, we discuss whether our results relativize. To do this, we must clarify the definition of an obfuscator relative to an oracle $F : \{0,1\}^* \to \{0,1\}^*$. What we mean is that all algorithms in the definition, including the one being obfuscated and including the adversary, have oracle access to F. For a circuit, this means that the circuit can have gates for evaluating F. We fix an encoding of (oracle) circuits as binary strings such that a circuit described by a string of length s can only make oracle queries of total length at most s.

By inspection, our initial (easy) impossibility results hold relative to any oracle, as the involve only simulation and diagonalization.

Proposition 4.14 *Proposition 3.4 (impossibility of 2-circuit obfuscators) and Theorem 3.5 (impossibility of TM obfuscators) hold relative to any oracle.*

Interestingly, however, our main impossibility results do not relativize.

Proposition 4.15 *There is an oracle relative to which efficient circuit obfuscators exist. Thus, Theorem 3.8,3.11, and Corollary 3.10 do not relativize.*

This can be viewed both as evidence that these results are nontrivial, and as (further) evidence that relativization is not a good indication of what we can prove.

Proof Sketch: The oracle $F = \bigcup_k F_k$ will consist of two parts $F_k = \mathcal{O}_k\#\mathrm{E}_k$, where $\mathcal{O}_k : \{0,1\}^k \times \{0,1\}^k \to \{0,1\}^{6k}$, and $\mathrm{E}_k : \{0,1\}^{6k} \times \{0,1\}^k \to \{0,1\}^k$. $\mathcal{O}_k$ is simply a uniformly random injective function of the given parameters. $\mathrm{E}_k(x,y)$ is defined as follows: If there exists a (C,r) such that $\mathcal{O}_k(C,r) = x$, then $\mathrm{E}_k(x,y) = C^F(y)$ (where C is viewed as the description of a circuit). Otherwise, $\mathrm{E}_k(x,y) = \perp$. Note that this definition of F_k is not circular, because C can only make oracle queries of size at most $|C| = k$, and hence can only query $F_{k'}$ for $k' \leq k/2$.

Now we can view $x = \mathcal{O}_k(C,r)$ as an obfuscation of C using coin tosses r. This satisfies the syntactic requirements of obfuscation, since $|x| = O(|C|)$ and the E_k allows one to efficiently

[10] This result is also implied if the Decisional Diffie–Hellman problem is "hard"; see [NR97] for precise statements of these assumptions.

evaluate $C(y)$ given just x and y. (Technically, we should define the obfuscation of C to be a circuit which has x hardwired in and makes an oracle query to E_k.)

So we only need to prove the virtual black-box property. By a union bound over polynomial-time adversaries A of description size smaller than $k/2$ and circuits C of size k, it suffices to prove the following claim.[11]

Claim 4.16 *For every PPT A there exists a PPT S such that for every circuit C of size k, the following holds with probability at least $1 - 2^{-2k}$ over F:*

$$\left| \Pr_{r \xleftarrow{R} \{0,1\}^k} \left[A^F(\mathcal{O}_k(C,r)) = 1 \right] - \Pr \left[S^{F,C}(1^k) = 1 \right] \right| \leq 2^{-\Omega(k)}$$

Fix a PPT A. We define the simulator S as follows. $S^{F,C}(1^k)$ chooses $x \xleftarrow{R} \{0,1\}^{6k}$ and simulates $A^F(x)$, using its own F-oracle to answer A's oracle queries, *except* A's queries to $\mathrm{E}_{k'}$ for $k' \geq k$. On A's query (x', y') to $\mathrm{E}_{k'}$, S feeds A the response z computed as follows:

1. If $x' = x$, then set $z = C(y')$ (computed using oracle access to C).
2. Else if $x' = \mathcal{O}_{k'}(C', r')$ for some previous query (C', r') to the $\mathcal{O}_{k'}$-oracle, then set $z = (C')^F(y')$ (computed recursively using these same rules).
3. Else set $z = \perp$.

From the fact that a circuit of size s can only make oracle queries of total length s, it follows that the recursive evaluation of $(C')^F(y)$ only incurs a polynomial overhead in running time. Also note that S never queries the $\mathrm{E}_{k'}$ oracle for $k' \geq k$.

Let us denote the execution of the above simulation for a particular x by $S^{F,C}(x)$. Notice that when $x = \mathcal{O}_k(C, r)$ for some r, then $S^{F,C}(x)$ and $A^F(x)$ have exactly the same behavior *unless* the above simulation produces some query (x', y') such that $x' \in \mathrm{Image}(\mathcal{O}_{k'})$, $x' \neq x$, and x' was not obtained by a previous query to $\mathcal{O}_{k'}$. Since $\mathcal{O}$ is a random length-tripling function, it follows that the latter happens with probability at most $\mathrm{poly}(k) \cdot 2^{2k}/2^{6k}$, taken over the choice of F and a random r (recall that $x = \mathcal{O}_k(C, r)$).[12] Thus, with probability at least $1 - 2^{-3k}$ over the choice of F, $S^{F,C}(\mathcal{O}_k(C,r)) = A^F(\mathcal{O}_k(C,r))$ for all but a $2^{-\Omega(k)}$ fraction of r's.

Thus, proving Claim 4.16 reduces to showing that:

$$\left| \Pr_{r \xleftarrow{R} \{0,1\}^k} \left[S^{F,C}(\mathcal{O}_k(C,r)) = 1 \right] - \Pr_{x \xleftarrow{R} \{0,1\}^{6k}} \left[S^{F,C}(x) = 1 \right] \right| \leq 2^{-\Omega(k)}$$

with high probability (say, $1 - 2^{3k}$) over the choice of F.

In other words, we need to show that the function $G(r) \stackrel{\mathrm{def}}{=} \mathcal{O}_k(C, r)$ is a pseudorandom generator against S. Since G is a random function from $\{0,1\}^k \to \{0,1\}^{6k}$, this would be obvious were it not for the fact that S has oracle access to F (which is correlated with G). Recall, however, that we made sure that S does not query the $\mathrm{E}_{k'}$-oracle for any $k' \geq k$. This enables us to use the following lemma, proven in Appendix B.

[11] Note that we are only proving the virtual black-box property against adversaries of "bounded nonuniformity," which in particular includes all uniform PPT adversaries. Presumably it can also be proven against nonuniform adversaries, but we stick to uniform adversaries for simplicity.

[12] Technically, this probability (and later ones in the proof) should also be taken over the coin tosses of A/S.

Lemma 4.17 *There is a constant $\delta > 0$ such that the following holds for all sufficiently large K and any $L \geq K^2$. Let D be an algorithm that makes at most K^δ oracle queries and let G be a random injective function $G : [K] \to [L]$. Then with probability at least $1 - 2^{-K^\delta}$ over G,*

$$\left| \Pr_{x \in [K]} \left[D^G(G(x)) = 1 \right] - \Pr_{y \in [L]} \left[D^G(y) = 1 \right] \right| \leq \frac{1}{K^\delta}.$$

Let us see how Lemma 4.17 implies what we want. Let $K = 2^k$ and associate $[K]$ with $\{0,1\}^k$. We fix all values of $\mathcal{O}_{k'}$ for all $k' \neq k$ and $\mathrm{E}_{k'}$ for all $k' < k$. We also fix the values of $\mathcal{O}_k(C', r)$ for all $C' \neq C$, and view $G(r) \stackrel{\text{def}}{=} \mathcal{O}_k(C, r)$ as a random injective function from $[K]$ to the remaining $L = K^6 - (K-1) \cdot K$ elements of $\{0,1\}^{6k}$. The only oracle queries of S that vary with the choice of G are queries to $\mathcal{O}_k$ at points of the form (C, r), which is equivalent to queries to G. Thus, Lemma 4.17 implies that the output of G is indistinguishable from the uniform distribution on some subset of $\{0,1\}^{6k}$ of size L. Since the latter has statistical difference $(K^6 - L)/K^6 < 1/K^4$ from the uniform distribution on $\{0,1\}^{6k}$, we conclude that G is ε-pseudorandom (for $\varepsilon = 1/K^\delta + 1/K^4 = 2^{-\Omega(k)}$) against S with probability at least $1 - 2^{-K^\delta} > 1 - 2^{-3k}$, as desired. □

While our result does not relativize in the usual sense, the proof does work for a slightly different form of relativization, which we refer to as *bounded relativization* (and is how the Random Oracle Model is sometimes interpreted in cryptography.) In *bounded relativization*, an oracle is a *finite* function with fixed input length (polynomially related to the security parameter k), and all algorithms/circuits in the protocol can have running time larger than this length (but still polynomial in k). In particular, in the context of obfuscation, this means that the circuit to be obfuscated can have size polynomial in this length.

Proposition 4.18 *Theorems 3.11 and 3.8 (one-way functions imply unobfuscatable functions and impossibility of circuit obfuscators), and Corollary 3.10 (unconditional impossibility of efficient circuit obfuscators) hold under bounded relativization (for any oracle).*

Proof Sketch: The only modification needed in the construction is to deal with oracle gates in the Hom algorithm in the proof of Lemma 3.6. Let's call say the oracle F has input length ℓ and output length 1 (without loss of generality). We augment the Hom_K to also take inputs of the form $(c_1, \ldots, c_\ell, \texttt{oracle})$ (where $(c_1, \ldots, c_\ell)$ are ciphertexts), on which it naturally outputs $\mathrm{Enc}_K(F(\mathrm{Dec}_K(c_1), \mathrm{Dec}_K(c_2), \ldots, \mathrm{Dec}_K(c_\ell)))$. The rest of the proof proceeds essentially unchanged. □

5 On a Complexity Analogue of Rice's Theorem

Rice's Theorem asserts that the only properties of partial recursive functions that can be decided from their representations as Turing machines are trivial. To state this precisely, we denote by $[M]$ the (possibly partial) function that the Turing Machine M computes. Similarly, for $[C]$ denotes the function computed by a circuit C.

Rice's Theorem *Let $L \subseteq \{0,1\}^*$ be a language such that for any M, M' , $[M] \equiv [M']$ implies that $M \in L \iff M' \in L$. If L is decidable, then L is trivial in the sense that either $L = \{0,1\}^*$ or $L = \emptyset$.*

The difficulty of problems such as SAT suggest that perhaps Rice's theorem can be "scaled-down" and that deciding properties of finite functions from their descriptions as circuits is intractable.

Simply replacing the word "Turing machine" with "circuit" and "decidable" with "polynomial time" does not work. A counterexample is the language $L = \{C \in \{0,1\}^* \mid C(0) = 0\}$ that can be decided in polynomial time, even though $[C] \equiv [C']$ implies $(C \in L \iff C' \in L)$, and both $L \neq \{0,1\}^*$ and $L \neq \emptyset$. Yet, there is a sense in which L is trivial — to decide whether $C \in L$ efficiently one does not need to use C itself, but rather one can do with oracle access to C only. This motivates the following conjecture:

Conjecture 5.1 (Scaled-down Rice's Theorem) *Let $L \subseteq \{0,1\}^*$ be a language such that for circuits C, C', $[C] \equiv [C']$ implies that $C \in L \iff C' \in L$. If $L \in \mathbf{BPP}$, then L is trivial in the sense that there exists a PPT S such that*

$$C \in L \Rightarrow \Pr[S^{[C]}(1^{|C|}) = 1] > \frac{2}{3}$$

$$C \notin L \Rightarrow \Pr[S^{[C]}(1^{|C|}) = 0] > \frac{2}{3}$$

We now consider a generalization of this conjecture to *promise problems* [ESY84], *i.e.,* decision problems restricted to some subset of strings. Formally, a promise problem Π is a pair $\Pi = (\Pi_Y, \Pi_N)$ of disjoint sets of strings, corresponding to YES and NO instances, respectively. The generalization of Conjecture 5.1 we seek is the following, where **BPP** is the generalization of **BPP** to promise problems:

Conjecture 5.2 *Let $\Pi = (\Pi_Y, \Pi_N)$ be a promise problem such that for circuits C, C', $[C] \equiv [C']$ implies that both $C \in \Pi_Y \iff C' \in \Pi_Y$ and $C \in \Pi_N \iff C' \in \Pi_N$. If $\Pi \in \mathbf{BPP}$, then Π is trivial in the sense that there exists a PPT S such that*

$$C \in \Pi_Y \Rightarrow \Pr[S^{[C]}(1^{|C|}) = 1] > \frac{2}{3}$$

$$C \in \Pi_N \Rightarrow \Pr[S^{[C]}(1^{|C|}) = 0] > \frac{2}{3}$$

Our construction of unobfuscatable functions implies that the latter conjecture is false.

Theorem 5.3 *If one-way functions exist, then Conjecture 5.2 is false.*

Proof Sketch: Let $\mathcal{H} = \{\mathcal{H}_k\}_{k \in \mathbb{N}}$ be the unobfuscatable function ensemble given by Theorem 3.11, and let $\pi : \bigcup_k \mathrm{Supp}(\mathcal{H}_k) \to \{0,1\}$ be the property guaranteed by the unobfuscatability condition.

Consider the following promise problem $\Pi = (\Pi_Y, \Pi_N)$:

$$\Pi_Y = \left\{ C : [C] \in \bigcup_k \mathrm{Supp}(\mathcal{H}_k) \text{ and } \pi([C]) = 1 \right\}$$

$$\Pi_N = \left\{ C : [C] \in \bigcup_k \mathrm{Supp}(\mathcal{H}_k) \text{ and } \pi([C]) = 0 \right\}$$

$\Pi \in \mathbf{BPP}$ because $\pi(f)$ is easy to compute with access to a circuit that computes f. But since $\pi(f)$ is hard to compute with black-box access to f, no S satisfying Conjecture 5.2 can exist. $\square$

It is an interesting problem to weaken or even remove the hypothesis that one-way functions exist. Reasons to believe that this may be possible are: 1. The conjecture is only about worst case complexity and not average case, and 2. The conjectures imply some sort of computational difficulty. For instance, if $\mathbf{NP} \subseteq \mathbf{BPP}$ then both conjectures are false, as Circuit Satisfiability is not decidable using black-box access. (Using black-box access, one cannot distinguish a circuit that is satisfied on exactly one randomly chosen input from an unsatisfiable circuit.) So if we could weaken the hypothesis of Theorem 5.3 to $\mathbf{NP} \not\subseteq \mathbf{BPP}$, Conjecture 5.2 would be false *unconditionally*.

We have shown that in the context of complexity, the generalization of Scaled-down Rice's Theorem (Conjecture 5.1) to promise problems (i.e., Conjecture 5.2) fails. When trying to find out what this implies about Conjecture 5.1 itself, one might try to get intuition from what happens in the context of computability. This direction is pursued in Appendix A. It turns out that the results in this context are inconclusive. We explore three ways to generalize Rice's Theorem to promise problems. The first, naive approach fails, and there are two non-naive generalizations, of which one succeeds and one fails.

What do our results say about the claim "the best thing you can do with a circuit/program is run it"? To answer this question, we must first interpret this sentence in a more formal way. The interpretation we suggest is "deciding any non-trivial semantic property of circuits is intractable" where "nontrivial" is defined above and by "semantic property" we mean a property of the *function* that the circuit computes, rather than a property of the particular circuit. This interpretation is expressed in Conjectures 5.1 and 5.2.

Since we haven't disproved Conjecture 5.1, how can we say that obfuscation is impossible? The answer is that obfuscation needs much more than Conjecture 5.1. Informally, Conjecture 5.1 only says that for every nontrivial property (*i.e.,* one which cannot be decided with oracle access), there exist circuits from which it is hard to decide the property. Obfuscation, on the other hand, requires that for every nontrivial property *and every function f* (for which the property is hard to decide given oracle access), there exist circuits *that compute the function f* from which it is hard to decide the property. Still, it may be within reach to also disprove Conjecture 5.1, and we leave this as an open problem.

6 Obfuscating Sampling Algorithms

In our investigation of obfuscators thus far, we have interpreted the "functionality" of a program as being the function it computes. However, sometimes one is interested in other aspects of a program's behavior, and in such cases a corresponding change should be made to the definition of obfuscators. In this section, we consider programs that are *sampling algorithms*, i.e. are probabilistic algorithms that take no input (other than possibly a length parameter), and produce an output according to some desired distribution.

For simplicity, we only work with sampling algorithms given by circuits — a circuit C with m input gates and n output gates can be viewed as a sampling algorithm for the distribution $\langle\langle C \rangle\rangle$ on $\{0,1\}^n$ obtained by evaluating C on m uniform and independent random bits. If A is an algorithm and C is a circuit, we write $A^{\langle\langle C \rangle\rangle}$ to indicate that A has *sampling access* to C. That is, A can obtain, on request, independent and uniform random samples from the distribution defined by C. The natural analogue of the definition of circuit obfuscators to sampling algorithms follows.

Definition 6.1 (sampling obfuscator) *A probabilistic algorithm $\mathcal{O}$ is a* sampling obfuscator *if, for some polynomial p, the following three conditions hold:*

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

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

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

We do not know whether this definition is impossible to meet, but we can rule out the following (seemingly) stronger definition.

Definition 6.2 (strong sampling obfuscator) *A* strong sampling obfuscator *is defined in the same way as a sampling obfuscator, expect that the "virtual black box" property is replaced with the following.*

- *("virtual black box" property) For any PPT A, there is a PPT S such that the ensembles $\{A(\mathcal{O}(C))\}_C$ and $\{S^{\langle\langle C\rangle\rangle}(1^{|C|})\}_C$ are computationally indistinguishable. That is, for every PPT D, there is a negligible function α such that*

$$\left|\Pr\left[D(C, A(\mathcal{O}(C))) = 1\right] - \Pr\left[D(C, S^{\langle\langle C\rangle\rangle}(1^{|C|})) = 1\right]\right| \leq \alpha(|C|).$$

Proposition 6.3 *If one-way functions exist, then strong sampling obfuscators do not exist.*

Proof Sketch: If one-way functions exist, then there exist message authentication codes (MACs) that are existentially unforgeable under chosen message attack. Let Tag_K denote the tagging (i.e., signing) algorithm for such a MAC with key K, and define a circuit $C_K(x) = (x, \mathrm{Tag}_K(x))$. That is, the distribution sampled by C_K is simply a random message together with its tag. Now suppose there exists a sampling obfuscator $\mathcal{O}$, and consider the PPT adversary A defined by $A(C) = C$. By the definition of a sampling obfuscator, there exists a PPT simulator S which, when giving sampling access to $\langle\langle C_K\rangle\rangle$, produces an output computationally indistinguishable from $A(\mathcal{O}(C_K)) = \mathcal{O}(C_K)$. That is, after receiving the tags of polynomially many random messages, S produces a circuit which is indistinguishable from one which generates random messages with its tags. This will contradict the security of the MAC.

Let $q = q(|K|)$ be a polynomial bound on the number of samples received from $\langle\langle C_K\rangle\rangle$ obtained by S, and consider a distinguisher D which does the following on input (C_K, C'): Recover the key K from C_K. Obtain $q+1$ random samples $(x_1, y_1), \ldots, (x_{q+1}, y_{q+1})$ from C'. Output 1 if the x_i's are all distinct and $y_i = \mathrm{Tag}_K(x_i)$ for all i.

Clearly, D outputs 1 with high probability on input $(C_K, A(\mathcal{O}(C_K)))$. (The only reason it might fail to output 1 is that the x_i's might not all be distinct, which happens with exponentially small probability.) On the other hand, the security of the MAC implies that D outputs 1 with negligible probability on input $(C_K, S^{\langle\langle C_K\rangle\rangle}(1^{|K|}))$ (over the choice of K and the coin tosses of all algorithms). The reason is that, whenever D outputs 1, the circuit output by S has generated a valid message-tag pair not received from the $\langle\langle C_K\rangle\rangle$-oracle. □

For sampling obfuscators in the sense of Definition 6.1, we do not know how to prove impossibility. Interestingly, we can show that they imply the nontriviality of **SZK**, the class of promise problems possessing statistical zero-knowledge proofs.

Proposition 6.4 *If efficient sampling obfuscators exist, then* $\mathbf{SZK} \neq \mathbf{BPP}$.

Proof: It is known that the following promise problem $\Pi = (\Pi_Y, \Pi_N)$ is in **SZK** [SV97] (and in fact has a noninteractive perfect zero-knowledge proof system [DDPY98, GSV99]):

$$\begin{aligned} \Pi_Y &= \{C : \langle\langle C \rangle\rangle = U_n\} \\ \Pi_N &= \{C : |\text{Supp}(C)| \leq 2^{n/2}\}, \end{aligned}$$

where n denotes the output length of the circuit C and U_n is the uniform distribution on $\{0,1\}^n$. Now suppose that an efficient sampling obfuscator $\mathcal{O}$ exists. Since, analogous to Lemma 3.9, such obfuscators imply the existence of one-way functions, there also exists a length-doubling pseudorandom generator G [HILL99]. Let $G_n : \{0,1\}^{n/2} \to \{0,1\}^n$ denote the circuit that evaluates G on inputs of length $n/2$.

Now, by the definition of pseudorandom generators and a hybrid argument, sampling access to $\langle\langle G_n \rangle\rangle$ is indistinguishable from sampling access to U_n. Thus, by the definition of a sampling obfuscator, $\mathcal{O}(G_n)$ is computationally indistinguishable from $\mathcal{O}(U_n)$, where by U_n we mean the trivial circuit that samples uniformly from U_n. By functionality, $\mathcal{O}(U_n)$ is always a YES instance of Π and $\mathcal{O}(G_n)$ is always a NO instance. It follows that $\Pi \notin \mathbf{BPP}$. ■

Remark 6.5 By using STATISTICAL DIFFERENCE, the complete problem for **SZK** from [SV97], in place of the promise problem Π, the above proposition can be extended to the natural definition of *approximate sampling obfuscators*, in which $\mathcal{O}(C)$ only needs to sample a distribution of small statistical difference from that of C.

7 Weaker Notions of Obfuscation

Our impossibility results rule 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. In this section, we suggest two weaker definitions of obfuscators that avoid the "virtual black box" paradigm (and hence are not ruled out by our impossibility proof).

The weaker definition asks that if two circuits compute the same function, then their obfuscations should be indistinguishable. For simplicity, we only consider the circuit version here.

Definition 7.1 (indistinguishability obfuscator) *An* indistinguishability obfuscator *is defined in the same way as a circuit obfuscator, except that the "virtual black box" property is replaced with the following:*

- *(indistinguishability) For any PPT A, there is a negligible function α such that for any two circuits C_1, C_2 which compute the same function and are of the same size k,*

$$|\Pr[A(\mathcal{O}(C_1))] - \Pr[A(\mathcal{O}(C_2))]| \leq \alpha(k).$$

Some (very slight) hope that this definition is achievable comes from the following observation.

Proposition 7.2 *(Inefficient) indistinguishability obfuscators exist.*

Proof: Let $\mathcal{O}(C)$ be the lexicographically first circuit of size $|C|$ that computes the same function as C. ■

While it would be very interesting to construct even indistinguishability obfuscators, their usefulness is limited by the fact that they provide no *a priori* guarantees about obfuscations of circuits C_1 and C_2 that compute different functions. However, it turns out that, if $\mathcal{O}$ is efficient, then it is "competitive" with respect to any pair of circuits. That is, no $\mathcal{O}'$ makes C_1 and C_2 much more indistinguishable than $\mathcal{O}$ does. To state this precisely, for a circuit C of size at most k, we define $\mathrm{Pad}_k(C)$ to be a trivial padding of C to size k. Feeding $\mathrm{Pad}_k(C)$ instead of C to an obfuscator can be thought of as increasing the "security parameter" from $|C|$ to k. (We chose not to explicitly introduce a security parameter into the definition of obfuscators to avoid the extra notation.) For the proof, we also need to impose a technical, but natural, constraint that the size of $\mathcal{O}'(C)$ only depends on the size of C.

Proposition 7.3 *Suppose $\mathcal{O}$ is an efficient indistinguishability obfuscator. Let $\mathcal{O}'$ be any algorithm satisfying the syntactic requirements of obfuscation, also satisfying the condition that $|\mathcal{O}'(C)| = q(|C|)$ for some fixed polynomial q. Then for any PPT A, there exists a PPT A' and a negligible function α such that for all circuits C_1, C_2 of size k,*

$$\begin{aligned}&\left|\Pr\left[A(\mathcal{O}(\mathrm{Pad}_{q(k)}(C_1))=1\right]-\Pr\left[A(\mathcal{O}(\mathrm{Pad}_{q(k)}(C_2))=1\right]\right|\\ &\quad\leq\ \left|\Pr\left[A'(\mathcal{O}'(C_1))=1\right]-\Pr\left[A'(\mathcal{O}'(C_2))=1\right]\right|+\alpha(k).\end{aligned}$$

Proof: Define $A'(C) \stackrel{\mathrm{def}}{=} A(\mathcal{O}(C))$. Then, for any circuit C_i of size k, we have

$$\begin{aligned}&\left|\Pr\left[A(\mathcal{O}(\mathrm{Pad}_{q(k)}(C_i)))=1\right]-\Pr\left[A'(\mathcal{O}'(C_i))=1\right]\right|\\ &\quad=\ \left|\Pr\left[A(\mathcal{O}(\mathrm{Pad}_{q(k)}(C_i)))=1\right]-\Pr\left[A(\mathcal{O}(\mathcal{O}'(C_i)))=1\right]\right|\\ &\quad\leq\ \mathrm{neg}(q(k))=\mathrm{neg}(k),\end{aligned}$$

where the inequality is because $\mathrm{Pad}_{q(k)}(C_i)$ and $\mathcal{O}'(C_i)$ are two circuits of size $q(k)$ which compute the same function and because $\mathcal{O}$ is an indistinguishability obfuscator. Thus,

$$\begin{aligned}&\left|\Pr\left[A(\mathcal{O}(\mathrm{Pad}_{q(k)}(C_1))=1\right]-\Pr\left[A(\mathcal{O}(\mathrm{Pad}_{q(k)}(C_2)))=1\right]\right|\\ &\quad\leq\ \left|\Pr\left[A(\mathcal{O}(\mathrm{Pad}_{q(k)}(C_1))=1\right]-\Pr\left[A'(\mathcal{O}'(C_1))=1\right]\right|\\ &\qquad+\left|\Pr\left[A'(\mathcal{O}'(C_1))=1\right]-\Pr\left[A'(\mathcal{O}'(C_2))=1\right]\right|\\ &\qquad+\left|\Pr\left[A'(\mathcal{O}'(C_2))=1\right]-\Pr\left[A(\mathcal{O}(\mathrm{Pad}_{q(k)}(C_2)))=1\right]\right|\\ &\quad\leq\ \mathrm{neg}(k)+\left|\Pr\left[A'(\mathcal{O}'(C_1))=1\right]-\Pr\left[A'(\mathcal{O}'(C_2))=1\right]\right|+\mathrm{neg}(k).\end{aligned}$$

■

Even with the competitiveness property, it still seems important to have explicit guarantees on the behavior of an obfuscator on circuits that compute different functions. We now give a definition that provides such a guarantee, while still avoiding the "virtual black box" paradigm. Roughly speaking, it says that if it is possible to distinguish the obfuscations of a pair of circuits, then one can find inputs on which they differ given any pair of circuits which compute the same functions. This definition is motivated in part by our impossibility proof, in which the functions constructed are such that it is possible to efficiently find inputs on which they differ given any pair of circuits computing them.

Definition 7.4 (differing-inputs obfuscator) *An* differing-inputs obfuscator *is defined in the same way as an indistinguishability obfuscator, except that the "indistinguishability" property is replaced with the following:*

- *(differing-inputs property) For any PPT A, there is a probabilistic algorithm A' and a negligible function α such that the following holds. Suppose C_1 and C_2 are circuits of size k such that*
$$\varepsilon \stackrel{\text{def}}{=} |\Pr[A(\mathcal{O}(C_1)) = 1] - \Pr[A(\mathcal{O}(C_2)) = 1]| > \alpha(k).$$
Then, for any C_1', C_2' of size k such that C_i' computes the same function as C_i for $i = 1, 2$, $A'(C_1', C_2')$ outputs an input on which C_1 and C_2 differ in time $\text{poly}(k, 1/(\varepsilon - \alpha(k)))$.

This definition is indeed stronger than that of indistinguishability obfuscators, because if C_1 and C_2 compute the same function, then A' can never find an input on which they differ and hence ε must be negligible.

8 Watermarking and Obfuscation

Generally speaking, *(fragile) watermarking* is the problem of embedding a message in an object such that the message is difficult to remove without "ruining" the object. Most of the work on watermarking has focused on watermarking *perceptual* objects, *e.g.,* images or audio files. (See the surveys [MMS+98, PAK99].) Here we concentrate on watermarking *programs*, as in [CT00, NSS99]. A watermarking scheme should consist of a *marking* algorithm which embeds a message m into a given program, and an *extraction* algorithm which extracts the message from a marked program. Intuitively, the following properties should be satisfied:

- (functionality) The marked program computes the same function as the original program.
- (meaningfulness) Most programs are unmarked.
- (fragility) It is infeasible to remove the mark from the program without (substantially) changing its behavior.

There are a various heuristic methods for software watermarking in the literature (*cf.,* [CT00]), but as with obfuscation, there has been little rigorous work on this problem. Here we do not attempt to provide a thorough definitional treatment of software watermarking, but rather consider a couple of weak formalizations which we relate to our results on obfuscation. The difficulty in formalizing watermarking comes, of course, in capturing the fragility property. As with obfuscation, it is easy to remove a watermark from programs for functions that are (exactly) learnable with membership queries (by using the learning algorithm to generate a new program (for the function) that is independent of the marking). A natural question is whether learnable functions are the only ones that cause problems. That is, can the following definition be satisfied?

Definition 8.1 (software watermarking) *A* (software) watermarking scheme *is a pair of (keyed) probabilistic algorithms* $(\text{Mark}, \text{Extract})$ *satisfying the following properties:*

- *(functionality) For every circuit C, key K, and message m, the string* $\text{Mark}_K(C, m)$ *describes a circuit that computes the same function as C.*
- *(polynomial slowdown) There is a polynomial p such that for every circuit C,* $|\text{Mark}_K(C, m)| \leq p(|C| + |m| + |K|)$.

- *(extraction) For every circuit C, key K, and message m,* $\text{Extract}_K(\text{Mark}_K(C,m)) = m$.
- *(meaningfulness) For every circuit C,* $\Pr_K[\text{Extract}_K(C) \neq \perp] = \text{neg}(|C|)$.
- *(fragility) For every PPT A, there is a PPT S such that for every C and m*

$$\begin{aligned}&\Pr_K\left[A(\text{Mark}_K(C,m)) = C' \text{ s.t. } C' \equiv C \text{ and } \text{Extract}_K(C') \neq m\right] \\ &\leq \Pr\left[S^C(1^{|C|}) = C' \text{ s.t. } C' \equiv C\right] + \text{neg}(|C|),\end{aligned}$$

where K is uniformly selected in $\{0,1\}^{\max(|C|,|m|)}$, and $C' \equiv C$ means that C' and C compute the same function.

We say that the scheme is efficient *if* Mark *and* Extract *run in polynomial time.*

By using our construction of totally unobfuscatable functions, we can prove that this definition is impossible to meet.

Theorem 8.2 *If one-way functions exist, then no watermarking scheme in the sense of Definition 8.1 exists.*

Proof Sketch: Consider the totally unobfuscatable function ensemble guaranteed by Theorem 4.2. No matter how we try to produce a marked circuit from $f \stackrel{R}{\leftarrow} \mathcal{H}$, the algorithm A given by the unobfuscatability condition in Definition 4.2 can reconstruct the canonical circuit f, which by the meaningfulness property is unmarked with high probability. On the other hand, the simulator, given just oracle access to f, will be unable produce any circuit computing the same function (since if it could, then it could compute $\pi(f)$, which is pseudorandom). □

Corollary 8.3 Efficient *watermarking schemes in the sense of Definition 8.1 do not exist (unconditionally).*

Given these impossibility results, we are led to seek the weakest possible formulation of the fragility condition — that the any adversary *occasionally* fails to remove the mark.

Definition 8.4 (occasional watermarking) *An* occasional *software watermarking scheme is defined in the same way as Definition 8.1, except that the fragility condition is replaced with the following:*

- *For every PPT A, there exists a circuit C and a message m such that*

$$\Pr_K\left[A(\text{Mark}_K(C,m)) = C' \text{ s.t. } C' \equiv C \text{ and } \text{Extract}_K(C') \neq m\right] \leq 1 - 1/\text{poly}(|C|),$$

where K is uniformly selected in $\{0,1\}^{\max(|C|,|m|)}$.

Interestingly, in contrast to the usual intuition, this weak notion of watermarking is inconsistent with obfuscation (even the weakest notion we proposed in Section 7).

Proposition 8.5 *Occasional software watermarking schemes and efficient indistinguishability obfuscators (as in Definition 7.1) cannot both exist. (Actually, we require the watermarking scheme to satisfy the additional natural condition that $|\text{Mark}_K(C,m)| = q(|C|)$ for some fixed polynomial q and all $|C| = |m| = |K|$.)*

Proof: We view the obfuscator $\mathcal{O}$ as a "watermark remover." By functionality of watermarking and obfuscation, for every circuit C and key K, $\mathcal{O}(\text{Mark}_K(C, 1^{|C|}))$ is a circuit computing the same function as C. Let C' be a padding of C to the same length as $\text{Mark}_K(C, 1^{|C|})$. By fragility, $\text{Extract}_K(\mathcal{O}(\text{Mark}_K(C, 1))) = 1$ with nonnegligible probability. By meaningfulness, $\text{Extract}_K(\mathcal{O}(C')) = 1$ with negligible probability. Thus, Extract_K distinguishes $\mathcal{O}(C')$ and $\mathcal{O}(\text{Mark}_K(C, 1^{|C|}))$, contradicting the indistinguishability property of $\mathcal{O}$. ■

Note that this proposition fails if we allow $\text{Mark}_K(C, m)$ to instead be an *approximate implementation* of C in the sense of Definition 4.3. Indeed, in such a case it seems that obfuscators would be useful in constructing watermarking schemes, for the watermark could be embedded by changing the value of the function at a random input, after which an obfuscator is used to "hide" this change. Note that approximation may also be relevant in the fragility condition, for it would be nice to prevent adversaries from producing unmarked approximate implementations of the function.

As with obfuscation, positive theoretical results about watermarking would be very welcome. One approach, taken by Naccache, Shamir, and Stern [NSS99], is to find watermarking schemes for specific useful families of functions.

9 Directions for Further Work

We have shown that obfuscation, as it is typically understood (*i.e.,* satisfying a virtual black-box property), is impossible. However, we view it as an important research direction to explore whether there are alternative senses in which programs can be made "unintelligible." These include (but are not limited to) the following notions of obfuscation which are not ruled out by our impossibility results:

- Indistinguishability (or differing-input) obfuscators, as in Definition 7.1 (or Definition 7.4, respectively).
- Sampling obfuscators, as in Definition 6.1.
- Obfuscators that only have to approximately preserve functionality with respect to a *specified* distribution on inputs, such as the uniform distribution. (In Section 4.2, we have ruled out obfuscators that preserving functionality on *each* input with high probability.)
- Obfuscators for a restricted, yet still nontrivial, class of functions. By Theorem 4.13, any such class of functions should not contain $\mathbf{TC_0}$. That leaves only very weak complexity classes (e.g., $\mathbf{AC_0}$, read-once branching programs), but the class of functions need not be restricted only by "computational" power: syntactic or functional restrictions may offer a more fruitful avenue. We note that the constructions of [CMR98] can be viewed as some form of obfuscators for "delta functions" (i.e., functions $f : \{0,1\}^n \to \{0,1\}$ which take on the value 1 at exactly one point in $\{0,1\}^n$.)

In addition to obfuscation, related problems such as homomorphic encryption and software watermarking are also little understood. For software watermarking, even finding a reasonable formalization of the problem (which is not ruled out by our constructions, unlike Definition 8.1) seems to be challenging, whereas for homomorphic encryption, the definitions are (more) straightforward, but existence is still open.

Finally, our investigation of complexity-theoretic analogues of Rice's theorem has left open questions, such as whether Conjecture 5.1 holds.

Acknowledgments

We are grateful to Luca Trevisan for collaboration at an early stage of this research. We also thank Dan Boneh, Ran Canetti, Michael Rabin, Yacov Yacobi, and the anonymous CRYPTO reviewers 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 Impagaliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In J. Kilian, editor, *Advances in Cryptology—CRYPTO '01*, Lecture Notes in Computer Science. Springer-Verlag, 2001, August 2001. To appear.

[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.

[CMR98] Ran Canetti, Daniele Micciancio, and Omer Reingold. Perfectly one-way probabilistic hash functions. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 131–140, 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.

[DDPY98] Alfredo De Santis, Giovanni Di Crescenzo, Giuseppe Persiano, and Moti Yung. Image Density is complete for non-interactive-SZK. In *Automata, Languages and Programming, 25th International Colloquium*, Lecture Notes in Computer Science, pages 784–795, Aalborg, Denmark, 13–17 July 1998. Springer-Verlag. See also preliminary draft of full version, May 1999.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

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

[ESY84] Shimon Even, Alan L. Selman, and Yacov Yacobi. The complexity of promise problems with applications to public-key cryptography. *Information and Control*, 61(2):159–173, 1984.

[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.

[GT00] Rosario Gennaro and Luca Trevisan. Lower bounds on the efficiency of generic cryptographic constructions. In *41st Annual Symposium on Foundations of Computer Science*, Redondo Beach, CA, 17–19 October 2000. IEEE.

[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.

[GSV99] Oded Goldreich, Amit Sahai, and Salil Vadhan. Can statistical zero-knowledge be made non-interactive?, or On the relationship of SZK and NISZK. In *Advances in Cryptology—CRYPTO '99*, Lecture Notes in Computer Science. Springer-Verlag, 1999, 15–19 August 1999. To appear.

[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.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396 (electronic), 1999.

[IL89] Russell Impagliazzo and Michael Luby. One-way functions are essential for complexity based cryptography (extended abstract). In *30th Annual Symposium on Foundations of Computer Science*, pages 230–235, Research Triangle Park, North Carolina, 30 October–1 November 1989. IEEE.

[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.

[LR88] Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2):373–386, 1988. Special issue on cryptography.

[MMS+98] Lesley R. Matheson, Stephen G. Mitchell, Talal G. Shamoon, Robert E. Tarjan, and Francis Zane. Robustness and security of digital watermarks. In H. Imai and Y. Zheng, editors, *Financial Cryptography—FC '98*, volume 1465 of *Lecture Notes in Computer Science*, pages 227–240. Springer, February 1998.

[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.

[PAK99] Fabien A. P. Petitcolas, Ross J. Anderson, and Markus J. Kuhn. Information hiding — a survey. *Proceedings of the IEEE*, 87(7):1062–1078, 1999.

[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.

[SV97] Amit Sahai and Salil P. Vadhan. A complete promise problem for statistical zero-knowledge. In *38th Annual Symposium on Foundations of Computer Science*, pages 448–457, Miami Beach, Florida, 20–22 October 1997. IEEE.

[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/`.

A Generalizing Rice's Theorem to Promise Problems.

We say that a Turing machine M *decides* the promise problem $\Pi = (\Pi_Y, \Pi_N)$ if

$$x \in \Pi_Y \Rightarrow M(x) = 1$$

$$x \in \Pi_N \Rightarrow M(x) = 0$$

In such a case, we say that Π is is *decidable*. We say that Π is closed under $[\cdot]$ if for all M, M', if $[M] \equiv [M']$ then both $M \in \Pi_Y \iff M' \in \Pi_Y$ and $M \in \Pi_N \iff M' \in \Pi_N$ hold.

The straightforward way to generalize Rice's Theorem to promise problems is the following:

Conjecture A.1 (Rice's Theorem — naive generalization) *Let $\Pi = (\Pi_Y, \Pi_N)$ be a promise problem closed under $[\cdot]$. If Π is decidable, then Π is trivial in the sense that either $\Pi_Y = \emptyset$ or $\Pi_N = \emptyset$.*

This generalization is really too naive. Consider the following promise problem (Π_Y, Π_N)

$$\Pi_Y = \{M \mid M \text{ always halts, } M(0) = 1\}$$

$$\Pi_N = \{M \mid M \text{ always halts, } M(0) = 0\}$$

It is obviously decidable, non-trivial, and closed under $[\cdot]$.

Our next attempt at generalizing Rice's Theorem to promise problems is based on the idea of a simulator, which we use to formalize the interpretation of Rice's Theorem as "the only useful thing you can do with a machine is run it." Recall that for a Turing machine M, the function $\langle M \rangle(1^t, x)$ is defined to be y if $M(x)$ halts within t steps with output y, and $\perp$ otherwise.

Theorem A.2 (Rice's Theorem — second generalization) *Let $\Pi = (\Pi_Y, \Pi_N)$ be a promise problem closed under $[\cdot]$. Suppose that Π is decidable, then Π is trivial in the sense that there exists a Turing machine S such that*

$$M \in \Pi_Y \Rightarrow S^{\langle M \rangle}(1^{|M|}) = 1$$
$$M \in \Pi_N \Rightarrow S^{\langle M \rangle}(1^{|M|}) = 0$$

Proof: Suppose that $\Pi = (\Pi_Y, \Pi_N)$ is decided by the Turing machine T. We will build a machine S which will satisfy the conclusion of the theorem.

We say that a machine N is *n-compatible* with a machine M if $\langle N \rangle(1^t, x) = \langle M \rangle(1^t, x)$ for all $|x|, t \leq n$. Note that:

1. n-compatibility with M can be decided using oracle access to $\langle M \rangle$.
2. M is n-compatible with itself for all n.
3. If $[M] \not\equiv [N]$ then there exists a number n' such that N is not n-compatible with M for all $n > n'$.
4. It may be the case than $[M] \equiv [N]$ but N is not n-compatible with M for some n.

With oracle $\langle M \rangle$ and input $1^{|M|}$, S does the following for $n = 0, 1, 2, \ldots$:

1. Compute the set S_n which consists of all the machines of size $|M|$ that are n-compatible with M (this can be done in finite time as there are only finitely many machines of size $|M|$).
2. Run T on all the machines in S_n for n steps. If T halts on all these machines and returns the same answer σ, then halt and return σ. Otherwise, continue.

It is clear that if S halts then it returns the same answer as $T(M)$. This is because M is n-compatible with itself for all n and so $M \in S_n$ for all n.

We claim that S always halts. For any machine N of size $|M|$ such that $[N] \not\equiv [M]$, there's a number n' such that n is not in S_n for all $n > n'$. Since there are only finitely many such machines, there's a number n'' such that all the machines $N \in S_n$ for $n > n''$ satisfy $[N] \equiv [M]$. For any such machine N with $[N] \equiv [M]$, T halts after a finite number of steps and outputs the same answer as $T(M)$. Again, since there are only finitely many of them , there's a number $n > n''$ such that T halts on all the machines of S_n in n steps and returns the same answer as $T(M)$. ■

Our previous proof relied heavily on the fact that the simulator was given an upper bound on the size of the machine M. While in the context of complexity we gave this length to the simulator to allow it enough running time, one may wonder whether it is justifiable to give this bound to the simulator in the computability context. That is:

Conjecture A.3 (Rice's Theorem — third generalization) *Let $\Pi = (\Pi_Y, \Pi_N)$ be a promise problem closed under $[\cdot]$. Suppose that Π is decidable. Then $\Pi =$ is trivial in the sense that there exists a Turing machine S such that*

$$M \in \Pi_Y \Rightarrow S^{\langle M \rangle}() = 1$$

$$M \in \Pi_N \Rightarrow S^{\langle M \rangle}() = 0$$

It turns out that this small change makes a difference.

Theorem A.4 *Conjecture A.3 is false.*

Proof: Consider the following promise problem $\Pi = (\Pi_Y, \Pi_N)$:

$$\Pi_Y = \{M \mid M \text{ always halts, } \exists x < \mathrm{KC}([M]) \text{ s.t. } [M](x) = 1\}$$

$$\Pi_N = \{M \mid M \text{ always halts, } \forall x\ M(x) = 0\}$$

where for a partial recursive function f, $\mathrm{KC}(f)$ is the description length of the smallest Turing machine that computes f. It is obvious that Π is closed under $[\cdot]$.

We claim that Π is decidable. Indeed, consider the following Turing machine T: On input M, T invokes $M(x)$ for all $x < |M|$ and returns 1 iff it gets a non-zero answer. Since any machine in $\Pi_Y \cup \Pi_N$ always halts, T halts in finite time. If T returns 1 then certainly M is not in Π_N. If $M \in \Pi_Y$ then $M(x) = 1$ for some $x < \mathrm{KC}([M]) \leq |M|$ and so T returns 1.

We claim that Π is not trivial in the sense of Conjecture A.3. Indeed, suppose for contradiction that there exists a simulator S such that

$$M \in \Pi_Y \Rightarrow S^{\langle M \rangle}() = 1$$

$$M \in \Pi_N \Rightarrow S^{\langle M \rangle}() = 0$$

Consider the machine Z which reads its input and then returns 0. We have that

$$\langle Z \rangle(1^t, x) = \begin{cases} \perp & t < |x| \\ 0 & \text{otherwise} \end{cases}$$

As $Z \in \Pi_N$, we know that $S^{\langle Z \rangle}()$ will halt after a finite time and return 0. Let n be an upper bound on $|x|$ and t over all oracle queries $(1^t, x)$ of $S^{\langle Z \rangle}()$.

Let r be a string of Kolmogorov complexity $2n$. Consider the machine $N_{n,r}$ which computes the following function,

$$N_{n,r}(x) = \begin{cases} 0 & x \leq n \\ 1 & x = n+1 \\ r & x \geq n+2 \end{cases}$$

and runs in time $|x|$ on inputs x such that $|x| \leq n$.

For any $t, |x| \leq n$, $\langle Z \rangle(1^t, x) = \langle N_{n,r} \rangle(1^t, x)$. Therefore $S^{\langle N_{n,r} \rangle}() = S^{\langle Z \rangle}() = 0$. But $N_{n,r} \in \Pi_Y$ since $N_{n,r}(n+1) = 1$ and $\mathrm{KC}([N_{n,r}]) > n+1$. This contradicts the assumption that S decides Π. ■

B Pseudorandom Oracles

In this section, we sketch a proof of the following lemma, which states that a random function is a pseudorandom generator relative to itself with high probability.

Lemma 4.17 *There is a constant $\delta > 0$ such that the following holds for all sufficiently large K and any $L \geq K^2$. Let D be an algorithm that makes at most K^δ oracle queries and let G be a random injective function $G : [K] \to [L]$. Then with probability at least $1 - 2^{-K^\delta}$ over G,*

$$\left| \Pr_{x \in [K]} \left[D^G(G(x)) = 1 \right] - \Pr_{y \in [L]} \left[D^G(y) = 1 \right] \right| \leq \frac{1}{K^\delta}. \tag{4}$$

We prove the lemma via a counting argument in the style of Gennaro and Trevisan's proof that a random permutation is one-way against nonuniform adversaries [GT00]. Specifically, we will show that "most" G for which Inequality (4) fails have a "short" description given D, and hence there cannot be too many of them.

Let $\mathcal{G}$ be the collection of G's for which Inequality (4) fails (for a sufficiently small δ, whose value is implicit in the proof below). We begin by arguing that, for every $G \in \mathcal{G}$, there is a large set $S_G \subset [K]$ of inputs on which D's behavior is "independent," in the sense that for $x \in S$, none of the oracle queries made in the execution of $D^G(G(x))$ are at points in S, yet D still has nonnegligible advantage in distinguishing $G(x)$ from random. Actually, we will not be able to afford specifying S_G when we "describe" G, so we actually show that there is a fixed set S (independent of G) such that for most G, the desired set S_G can be obtained by just throwing out a small number of elements from S.

Claim B.1 *There is a set $S \subset [K]$ with $|S| = K^{1-5\delta}$, and $\mathcal{G}' \subset \mathcal{G}$ with $|\mathcal{G}'| = |\mathcal{G}|/2$ such that for all $G \in \mathcal{G}'$, there is a set $S_G \subset S$ with the following properties:*

1. *$|S_G| = (1 - \gamma)|S|$, where $\gamma = K^{-3\delta}$.*
2. *If $x \in S_G$, then $D^G(G(x))$ never queries its oracle at an element of S_G.*
3. $$\left| \Pr_{x \in S_G} \left[D^G(G(x)) = 1 \right] - \Pr_{y \in L_G} \left[D^G(y) = 1 \right] \right| > \frac{1}{2K^\delta},$$

   where $L_G \stackrel{\text{def}}{=} [L] \setminus G([K] \setminus S_G)$. (Note that L_G contains more than a $1 - K/L$ fraction of L.)

Proof: First consider choosing both a random $G \stackrel{\text{R}}{\leftarrow} \mathcal{G}$ and a random S (among subsets of $[K]$ of size $K^{1-5\delta}$). We will show that with probability at least $1/2$, there is a good subset $S_G \subset S$ satisfying Properties 1–3. By averaging, this implies that there is a fixed set S for which a good subset exists for at least half the $G \in \mathcal{G}$, as desired. Let's begin with Property 2. For a random G, S, and a random $x \in S$, note that $D^G(G(x))$ initially has no information about S, which is a random set of density $K^{-5\delta}$. Since D makes at most K^δ queries, the probability that it queries its oracle at some element of S is at most $K^\delta \cdot K^{-5\delta} = K^{-4\delta}$. Thus, with probability at least $3/4$ over G and S, $D^G(G(x))$ queries its oracle at an element of S for at most a $4/K^{-4\delta} < \gamma$ fraction of $x \in S$. Throwing out this γ fraction of elements of S gives a set S_G satisfying Properties 1 and 2.

Now let's turn to Property 3. By a Chernoff-like bound, with probability at least $1 - \exp(\Omega(K^{1-5\delta} \cdot (K^{-\delta})^2)) > 3/4$ over the choice of S,

$$\left| \Pr_{x \in S} \left[D^G(G(x)) = 1 \right] - \Pr_{x \in [K]} \left[D^G(G(x)) = 1 \right] \right| \leq \frac{1}{4K^\delta}.$$

Then we have:

$$\begin{aligned}
&\left|\Pr_{x\in S_G}\left[D^G(G(x))=1\right]-\Pr_{y\in L_G}\left[D^G(y)=1\right]\right| \\
&\geq \left|\Pr_{x\in[K]}\left[D^G(G(x))=1\right]-\Pr_{y\in[L]}\left[D^G(y)=1\right]\right| \\
&\quad-\left|\Pr_{x\in S_G}\left[D^G(G(x))=1\right]-\Pr_{x\in[S]}\left[D^G(G(x))=1\right]\right| \\
&\quad-\left|\Pr_{x\in S}\left[D^G(G(x))=1\right]-\Pr_{x\in[K]}\left[D^G(G(x))=1\right]\right| \\
&\quad-\left|\Pr_{y\in[L]}\left[D^G(y)=1\right]-\Pr_{y\in L_G}\left[D^G(y)=1\right]\right| \\
&> 1/K^\delta-\gamma-1/4K^\delta-K/L \\
&> 1/2K^\delta
\end{aligned}$$

■

Now we show how the above claim implies that every $G\in\mathcal{G}'$ has a "small" description.

Claim B.2 *Every $G\in\mathcal{G}'$ can be uniquely described by $(\log B)-\Omega(K^{1-7\delta})$ bits given D, where B is the number of injective functions from $[K]$ to $[L]$.*

Proof: For starters, the description of G will contains the set S_G and the values of $G(x)$ for all $x\notin S_G$. Now we'd like to argue that this information is enough to determine $D^G(y)$ for all y. This won't exactly be the case, but rather we'll show how to compute $M^G(y)$ for some M that is "as good" as D. From Property 3 in Claim B.1, we have

$$\Pr_{x\in S_G}\left[D^G(G(x))=1\right]-\Pr_{y\in L_G}\left[D^G(y)=1\right]>\frac{1}{2K^\delta}.$$

(We've dropped the absolute values. The other case is handled analogously, and the only cost is one bit to describe which case holds.) We will describe an algorithm M for which the same inequality holds, yet M will only use the information in our description of G instead of making oracle queries to G. Specifically, on input y, M simulates $D(y)$, except that it handles each oracle query z as follows:

1. If $z\notin S_G$, then M responds with $G(z)$ (This information is included in our description of G).
2. If $z\in S_G$, then M halts and outputs 0. (By Property 2 of Claim B.1, this cannot happen if $y\in G(S_G)$, hence outputting 0 only improves M's distinguishing gap.)

Thus, given S_G and $G|_{[K]\setminus S_G}$, we have a function M satisfying

$$\Pr_{x\in S_G}\left[M(G(x))=1\right]-\Pr_{y\in L_G}\left[M(y)=1\right]>\frac{1}{2K^\delta} \tag{5}$$

To complete the description of G, we must specify $G|_{S_G}$, which we can think of as first specifying the image $T=G(S_G)\subset L_G$ and then the bijection $G:S_G\to T$. However, we can save in our description because T is constrained by Inequality (5), which can be rewritten as:

$$\Pr_{y\in T}\left[M(y)=1\right]-\Pr_{y\in L_G}\left[M(y)=1\right]>\frac{1}{2K^\delta} \tag{6}$$

Chernoff Bounds say that most large subsets are good approximators of the average of a boolean function. Specifically, at most a $\exp(-\Omega((1-\gamma)K^{1-5\delta} \cdot (K^{-\delta})^2)) = \exp(-\Omega(K^{1-7\delta}))$ fraction of sets $T \subset L_G$ of size $(1-\gamma)K^{1-5\delta}$ satisfy Equation 6.

Thus, using M, we have "saved" $\Omega(K^{1-7\delta})$ bits in describing $G(S_G)$ (over the standard "truth-table" representation of a function G). However, we had to describe the set S_G itself, which would have been unnecessary in the truth-table representation. Fortunately, we only need to describe S_G as a subset of S, and this only costs $\log \binom{K^{1-5\delta}}{(1-\gamma)K^{1-5\delta}} = O(H_2(\gamma)K^{1-5\delta}) < O(K^{1-8\delta} \log K)$ bits (where $H_2(\gamma) = O(\gamma \log(1/\gamma))$ denotes the binary entropy function). So we have a net savings of $\Omega(K^{1-7\delta}) - O(K^{1-8\delta} \log K) = \Omega(K^{1-7\delta})$ bits. ■

From Claim B.2, $\mathcal{G}'$ can consist of at most an $\exp(-\Omega(K^{1-7\delta})) < K^{-\delta}/2$ fraction of injective functions $[K] \to [L]$, and thus $\mathcal{G}$ has density smaller than $K^{-\delta}$, as desired.