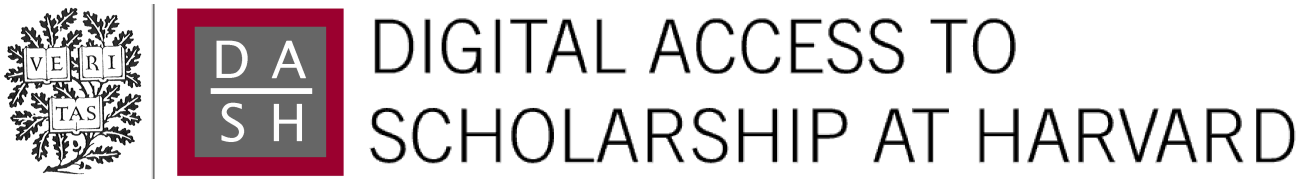

On the (Im)possibility of Obfuscating Programs

On the (Im)possibility of Obfuscating Programs[1]

Boaz Barak, Microsoft Research New England, Camrbidge, MA 02142. E-mail: b@boazbarak.org
and Oded Goldreich, Department of Computer Science, Weizmann Institute of Science, Rehovot, ISRAEL. E-mail: oded.goldreich@weizmann.ac.il
and Russell Impagliazzo, Department of Computer Science and Engineering, University of California, San Diego, La Jolla, CA 92093-0114. E-mail: russell@cs.ucsd.edu
and Steven Rudich, Computer Science Department, Carnegie Mellon University, 5000 Forbes Ave. Pittsburgh, PA 15213. E-mail: rudich@cs.cmu.edu
and Amit Sahai, Department of Computer Science, UCLA, Los Angeles, CA 90095. Email: sahai@cs.ucla.edu
and Salil Vadhan, School of Engineering and Applied Sciences and Center for Research on Computation and Society, Harvard University, 33 Oxford Street, Cambridge, MA 02138. E-mail: salil@seas.harvard.edu
and Ke Yang, Google Inc., Moutain View, CA 94043. E-mail: yangke@gmail.com

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 efficient programs $\mathcal{P}$ that are *unobfuscatable* in the sense that (a) given *any* efficient program P' that computes the same function as a program $P \in \mathcal{P}$, the "source code" P can be efficiently reconstructed, yet (b) given *oracle access* to a (randomly selected) program $P \in \mathcal{P}$, no efficient algorithm can reconstruct P (or even distinguish a certain bit in the code from random) except with negligible probability.

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.

Categories and Subject Descriptors: F.1.3 [**Theory of Computation**]: Complexity Measures and Classes; D.4.6 [**Software**]: Security and Protection—*Cryptographic Controls*

General Terms: Theory

Additional Key Words and Phrases: complexity theory, cryptography, homomorphic encryption, pseudorandom functions, Rice's Theorem, software protection, software watermarking, statistical zero knowledge

[1]A preliminary version of this paper appeared in *CRYPTO'01* [BGI+].

1. INTRODUCTION

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

The hope that some form of obfuscation is possible arises from the fact that analyzing programs expressed in rich enough formalisms is hard. Indeed, any programmer knows that total unintelligibility is the natural state of computer programs (and one must work hard in order to keep a program from deteriorating into this state). Theoretically, results such as Rice's Theorem and the hardness of the HALTING PROBLEM and SATISFIABILITY all seem to imply that the only useful thing that one can do with a program or circuit is to run it (on inputs of one's choice). However, this informal statement is, of course, highly speculative, 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 [CT]), there has been little theoretical work on this problem. This is unfortunate, since obfuscation, if it were possible, would have a wide variety of cryptographic and complexity-theoretic applications.

```
#include<stdio.h> #include<string.h>
main(){char*O,l[999]="''‘acgo\177~|xp .
-\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);}}
```

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

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

1.1. Some Applications of Obfuscators

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

Intuitively, obfuscators would also be useful in *watermarking* software (cf., [CT; NSS]). 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.

Removing Random Oracles.. The *Random Oracle Model* [BR] 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 [CGH], 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[2]. One might hope to obtain such functions by obfuscating a family of pseudorandom functions [GGM], whose input-output behavior is by definition indistinguishable from that of a truly random function.

Transforming Private-Key Encryption into Public-Key Encryption.. Obfuscation can also be used to create new public-key encryption schemes by obfuscating a private-key encryption scheme. Given a secret key K of a private-key encryption scheme, one can publish an obfuscation of the encryption algorithm Enc_K. 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 [DH], 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]

---

[2]We note that the results of [CGH] can also be seen as ruling out a very strong "virtual black box" definition of obfuscators. This is because their result implies that no obfuscator applied to any pseudorandom function family could work for all protocols, while a very strong virtual black box definition would guarantee this. We note, however, that our main results rule out a seemingly much weaker definition of obfuscation. Also, we note that ruling out strong virtual black box definitions is almost immediate: For example, one thing that can be efficiently computed from $\mathcal{O}(P)$ is the program $\mathcal{O}(P)$ itself. However, for any program P corresponding to a function that is hard to learn from queries, it would be infeasible to produce any program equivalent to P in functionality given only oracle access to P.

[3]From [DH]: "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

Homomorphic Encryption.. A long-standing open problem in cryptography is whether *homomorphic* encryption schemes exist (cf., [RAD; FM; DDN; BL; SYY]). 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, by using ideas as in the previous paragraph. Specifically, 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 along with the public key.

1.2. Our Main 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 $\mathcal{P}$ of efficient programs (in the form of Boolean circuits) that is *unobfuscatable* in the sense that

— Given *any* efficient program P' that computes the same function as a program $P \in \mathcal{P}$, the "source code" P can be reconstructed very efficiently (in time roughly quadratic in the running time of P').
— Yet, given oracle access to a (randomly selected) program $P \in \mathcal{P}$, no efficient algorithm can reconstruct P (or even distinguish a certain bit in the code from random) except with negligible probability.

Thus, there is no way of obfuscating the programs that compute these functions, even if (a) the obfuscator itself has unbounded computation time, and (b) the obfuscation is meant to hide only one bit of information about the function. We also note that the family $\mathcal{P}$ is a family of *circuits*, which means they take inputs of a specific bounded size and produce outputs of a specific bounded size, which could be known to a potential obfuscator, making the job of obfuscation seemingly easier.

We believe that the existence of such functions shows that the "virtual black box" paradigm for general-purpose 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, the programs $\mathcal{O}(P)$ 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*.

Impossibility of Applications.. To give further evidence that our impossibility result is not an artifact of definitional choices, but rather is inherent in the "virtual black box"

---

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 that 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."

idea, we also demonstrate that several of the applications of obfuscators are impossible. We do this by constructing *unobfuscatable* signature schemes, encryption schemes, and pseudorandom functions. These are objects satisfying the standard definitions of security, 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 [CGH]. They show that there exist (contrived) protocols that are secure in the idealized Random Oracle Model (of [BR]), 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 [FS]), 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. We mention that, subsequent to our work, Barak [Bar1] constructed arguably natural protocols that are secure in the random oracle model (e.g. those obtained by applying the Fiat–Shamir heuristic [FS] to his public-coin zero-knowledge arguments) but are insecure when the random oracle is replaced by any efficiently computable 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 [NR] 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* — probabilistic programs that, when fed a uniformly random string (of some length) as input, 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 results). 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 TC^0. In general, the complexity of our counterexample is essentially the same as the complexity of pseudorandom functions, and so a complexity class that does not contain our example will also not contain many cryptographically useful algorithms.

For further (nontechnical) discussion and interpretation of our results, the interested reader is referred to [Bar2].

1.4. Related Work

A fair number of heuristic approaches to obfuscation and software watermarking have been proposed in the past, as described in the survey of Collberg and Thomborson [CT]. A theoretical study of software protection, based on some tamper-proof hardware, was previously conducted by Goldreich and Ostrovsky [GO].

Hada [Had] gave some definitions for code obfuscators that 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 [Had].)

Canetti, Goldreich and Halevi [CGH] showed another setting in cryptography where getting a function's description is provably more powerful than black-box access. As mentioned above, they showed 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.

Related work that was done subsequent to the original publication [BGI$^+$] of our results is reviewed in Section 9.

1.5. Organization of the Paper

In Section 2, we give some basic definitions along with (very weak) definitions of obfuscators (within the virtual black box paradigm). In Section 3, we prove the impossibility of obfuscators by constructing an unobfuscatable family of programs. In Section 4, we give a number of extensions of our impossibility result, including impossibility results for obfuscators that 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.) This completes the main part of our paper.

Various ramifications are pursued in the rest of the paper. 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, as well as progress subsequent to the original versions of our paper [BGI$^+$].

2. DEFINITIONS

2.1. Preliminaries

In addition to the notation mentioned below, we refer to numerous standard concepts from cryptography and complexity theory. These can be found in [Gol1; Gol2] and [Sip], respectively.

Standard computational notation.. We use the shorthand *TM* for Turing machine, and the shorthand *PPT* 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 $x \in \{0,1\}^n$, it holds that $C(x) = f(x)$. 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, and by $A(x)$ we refer to the distribution induced by choosing r uniformly and running $A(x;r)$. By a *randomized* circuit, we mean a circuit that may additionally make use of special gates called *randomness gates*. Each randomness gate in a randomized circuit takes no input, and outputs a uniformly and independently chosen bit. If C is a randomized circuit, then $C(x)$ will refer to the distribution obtained by evaluating the randomized circuit C on input x. We will sometimes abuse notation and also use $C(x)$ to refer to a value y sampled from the distribution $C(x)$. We will identify Turing machines and circuits with their canonical representations as strings in $\{0,1\}^*$.

For a circuit C, we denote by $[C]$ the function computed by C. Similarly if M is a TM, then we denote by $[M]$ the (possibly partial) function computed by M.

Non-standard notation regarding computation with oracles.. We need to deviate from the standard conventions regarding using subroutines (or oracles), because a user having access to a code can run the code on a selected input *for a number of steps of her choice*.[4] Thus, black-box access to a program should mean access to the function that returns the result of running the program on a given input for a given number of steps. Thus, for a TM M, we denote by $\langle M \rangle$ the function defined as

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

This convention is unnecessary when dealing with circuits, since the "running time" of a circuit on each adequate input equals the predetermined size of the circuit. For simplicity, in both cases, we denote by $A^P(x)$ the output of algorithm A when executed on input x and oracle access to P. If P is a TM, then $A^P(x)$ is actually a shorthand for the standard notation $A^{\langle P \rangle}(x)$, which represents the output of algorithm A when executed on input x and oracle access to the function $\langle P \rangle$. If P is a circuit, then $A^P(x)$ is a shorthand for the standard notation $A^{[P]}(x)$, which represents the output of algorithm A when executed on input x and oracle access to the function $[P]$ (i.e., the function computed by P).

Probabilistic notation.. If D is a distribution, then by $x \stackrel{\text{R}}{\leftarrow} D$ we mean that x is a random variable distributed according to D. If S is a set, then by $x \stackrel{\text{R}}{\leftarrow} S$ we mean that x is a random variable that is distributed uniformly over the elements of S. The *support* of distribution D, i.e., the set of points that have nonzero probability under D,

[4]The point is that the actual running-time of a TM on a particular input may be smaller than the TM's *a priori* known time-bound. In this case, it is conceivable that a user having access to the code can learn the actual running-time of the TM on inputs of its choice, and we need to reflect this ability in the oracle-aided computation.

is denoted $\mathrm{Supp}(D)$. (Thus, $x \stackrel{R}{\leftarrow} D$ and $x \stackrel{R}{\leftarrow} \mathrm{Supp}(D)$ are the same only if D is distributed uniformly over its support.)

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 will sometimes use $\mathrm{neg}(\cdot)$ to denote an unspecified negligible function.

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) A weaker alternative is to consider the adversary as trying to produce an output that satisfies a predetermined (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) An even weaker requirement is to restrict the previous requirement to relations that are functions; that is, the adversary is trying to compute some predetermined function of the original program.
— (computing a predicate) The weakest alternative is obtained by restricting the previous requirement to $\{0,1\}$-valued functions; that is, the adversary is trying to decide some predetermined property of the original program.

An equivalent[5] formulation is obtained by following the first alternative, *with the crucial difference of allowing only one-bit outputs*. That is, we require that it is possible, given just oracle access to P, to produce an output distribution that is statistically close to the *single-bit distribution* that the adversary produces when given $\mathcal{O}(P)$.

The first two requirements are easily seen to be impossible to meet, in general. Consider a relation (or a distinguisher) R such that $R(P, P')$ accepts if programs P' and P agree on many randomly chosen inputs (say from $\{0,1\}^k$, where k is the security parameter). An adversary given an obfuscation $\mathcal{O}(P)$ can easily satisfy this relation by outputting $P' = \mathcal{O}(P)$. But it is infeasible to satisfy the relation given oracle access to P, if P is a program that is hard to learn from queries (even approximately, with respect to the uniform distribution on $\{0,1\}^k$); for example, if P comes from a family of pseudorandom functions.

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 for the collection $\mathfrak{F}$ of Turing machines* if the following three conditions hold:

[5] See Footnote 7.

— (functionality) For every TM $M \in \mathfrak{F}$, 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 $M \in \mathfrak{F}$, $|\mathcal{O}(M)| \leq p(|M|)$, 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 $M \in \mathfrak{F}$, it holds that

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

We say that $\mathcal{O}$ is *efficient* if it runs in polynomial time. If we omit specifying the collection $\mathfrak{F}$, then it is assumed to be the collection of all Turing machines.

We stress that Turing machines merely provide a formalization of "programs" (and that we could have alternatively considered programs defined by some generalized C-like programming language). In our context, the size of the program is the size of the Turing machine. Thus, unlike many other contexts in cryptography or complexity theory, for us the sizes of the Turing machines to be obfuscated are not meant to be thought of as "constants" (which would have made the notion of obfuscation trivial).[6]

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

— (functionality) For every circuit $C \in \mathfrak{F}$, 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 $C \in \mathfrak{F}$, we have that $|\mathcal{O}(C)| \leq p(|C|)$.
— ("virtual black box" property) For any PPT A, there is a PPT S and a negligible function α such that for all circuits $C \in \mathfrak{F}$, it holds that

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

We say that $\mathcal{O}$ is *efficient* if it runs in polynomial time. If we omit specifying the collection $\mathfrak{F}$, then it is assumed to be the collection of all circuits.

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 was mentioned in the foregoing informal discussion, requires 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. Clearly, the simulation of any single-bit output distribution implies the ability to match the probability of guessing the value of any predicate, and the converse holds by considering a predicate that is undetermined on every set of functionally equivalent programs.[7]

Another formulation refers to the distinguishability between obfuscations of two TMs/circuits: Specifically, for every C_1 and C_2, it holds

[6] This is similar to the way in which Turing machine sizes are treated when studying forms of the Bounded Halting problem (e.g. given a Turing machine M and a time bound t, does M accept the empty string within t steps?), which are often trivialized if the Turing machine is restricted to be of constant size. In general, one should not view program sizes as constant when the programs themselves are inputs (as in obfuscation and in the Bounded Halting problem).

[7] Needless to say, a rigorous proof requires a rigorous definition of the alternative condition, which we avoided in the main text. Loosely speaking, such a formulation may require that for every predicate π and every PPT A, there is a PPT S and a negligible function α such that for any distribution on programs

that $|\Pr[A(\mathcal{O}(C_1))=1] - \Pr[A(\mathcal{O}(C_2))=1]|$ is approximately equal to $|\Pr\left[S^{C_1}(1^{|C_1|},1^{|C_2|})=1\right] - \Pr\left[S^{C_2}(1^{|C_1|},1^{|C_2|})=1\right]|$. 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 $\mathrm{poly}(|\mathcal{O}(C)|)$ rather than $\mathrm{poly}(|C|)$). 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 quite easy to prove:[8]

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, cf. [HMS; HRSV]. The point is that they are nevertheless *impossible* to satisfy (as we will prove).[9]

We also note that the definitions are restricted to obfuscating deterministic algorithms; this restriction only makes our negative results stronger (since any obfuscator for probabilistic algorithms should also work for deterministic algorithms). Nevertheless, in Section 6 we discuss possible definitions of obfuscation for a special type of probabilistic algorithms (i.e., sampling algorithms, which get no "real" input).

3. THE MAIN IMPOSSIBILITY RESULT

To state our main result we introduce the notion of an unobfuscatable circuit ensemble.[10]

Definition 3.1. An *unobfuscatable circuit ensemble* is an ensemble $\{\mathcal{H}_k\}_{k\in\mathbb{N}}$ of distributions $\mathcal{H}_k$ on circuits (from, say, $\{0,1\}^{l_{\mathrm{in}}(k)}$ to $\{0,1\}^{l_{\mathrm{out}}(k)}$) satisfying:

— (efficient computability) Every circuit $C \in \mathrm{Supp}(\mathcal{H}_k)$ is of size $\mathrm{poly}(k)$. Moreover, $C \xleftarrow{R} \mathcal{H}_k$ can be sampled uniformly in $\mathrm{poly}(k)$-time.

C, it holds that

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

Note that it suffices to consider all ("degenerate") distributions C that have support size 1. Now, it is clear that the simulation of the distribution $A(\mathcal{O}(C))$ allows to match, for every predicate π, the probability that $A(\mathcal{O}(C)) = \pi(C)$. On the other hand, consider a predicate π such that for every program C there exist functionally equivalent programs C_0, C_1 such that $|C_0| = |C_1| = |C|$ and $\pi(C_\sigma) = \sigma$. Then, the ability to simulate $A(\mathcal{O}(C))$ follows from the hypothesis that there exists an S such that for every C and every $\sigma \in \{0,1\}$ it holds that $\Pr[A(\mathcal{O}(C_\sigma))=\pi(C_\sigma)]$ is upper-bounded by $\Pr\left[S^{C_\sigma}(1^{|C_\sigma|})=\pi(C_\sigma)\right] + \alpha(|C_\sigma|)$, since $S^C(1^{|C|}) = S^{C_\sigma}(1^{|C_\sigma|})$ for both σ.

[8]Given a circuit (on n-bit inputs) to be obfuscated, construct a Turing machine that emulates it, apply the TM-obfuscator to this TM, and output a circuit that emulates the latter (on inputs of length n).

[9]These definitions or even weaker ones, may still be useful when considering obfuscation as an end in itself, with the goal of protecting software against reverse-engineering, cf. [GR].

[10]In the preliminary version of our paper [BGI+], this is referred to as a *totally unobfuscatable function ensemble*.

— (unlearnability)[11]
There exists a polynomial-time computable function $\pi : \bigcup_{k\in\mathbb{N}} \mathrm{Supp}(\mathcal{H}_k) \to \{0,1\}^*$ such that $\pi(C)$ is pseudorandom given black-box access to $C \stackrel{R}{\leftarrow} \mathcal{H}_k$. That is, for every PPT S

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

— (reverse-engineerability)[12] C is easy to reconstruct given any equivalent circuit: There exists a polynomial-time algorithm A such that for every $C \in \bigcup_k \mathrm{Supp}(\mathcal{H}_k)$ and every circuit C' that computes the same function as C it holds that $A(C') = C$.

The reverse-engineerability condition says that the source code C can be completely reverse-engineered given any program computing the same function. On the other hand the unlearnability condition implies that it is infeasible to reverse-engineer the source code given only black-box access to C. We note that for the purpose of the main impossibility result, it would have sufficed for the function π to have a single bit output. (We will make use of the longer pseudorandom output of π in later results.) Putting the two items together, it follows that, when given only black-box access to C, it is hard to find *any* circuit computing the same function as C. In the language of learning theory (see [KV]), this says that an unobfuscatable circuit ensemble constitutes a concept class that is *hard to exactly learn with queries*. On the other hand, any concept class that can be exactly learned with queries is trivially obfuscatable (according even to the strongest definitions of obfuscation), because we can use the output of the learning algorithm when given oracle access to a function C in the class as an obfuscation of C.

We prove in Theorem 3.10 that, assuming one-way functions exist, there exists an unobfuscatable circuit 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 3.7). Since the existence of an *efficient* obfuscator implies the existence of one-way functions (Lemma 3.8), we conclude that efficient obfuscators do not exist (unconditionally).

However, the existence of unobfuscatable circuit ensemble has even stronger implications. As mentioned in the introduction, these programs cannot be obfuscated even if we allow the following relaxations to the obfuscator:

(1) The obfuscator does not have to run in polynomial time — it can be any random process.
(2) The obfuscator only has to work for programs in $\mathrm{Supp}(\mathcal{H}_k)$ and only for a nonnegligible fraction of these programs under the distributions $\mathcal{H}_k$.
(3) The obfuscator only has to hide an *a priori* fixed property (e.g. the first bit of $\pi(C)$) 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 such obfuscators do not exist. Next, we shall modify the

[11]The term "learnable" is used here in an intuitive sense that is somewhat different from its meaning in computational learning theory (see [KV]). On the one hand, we only require the learner to distinguish $\pi(C)$ from random. On the other hand, the definition allows $\pi(C)$ to be a function of the source code C rather than its functionality. However, in our construction $\pi(C)$ will in fact depend only on the functionality of C, making the task more akin to learning.

[12]We assume that any circuit has size that is greater than the number of its inputs. Thus, the length of the description of C' is at least $l_{\mathrm{in}}(k)$, which in our construction will be polynomially related to the length of the description of C (so A has at least enough time to write C).

construction in this proof to prove that TM obfuscators in the sense of Definition 2.1 do not exist. Then, 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 circuit ensemble (Theorem 3.10).

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 and N, it holds that

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

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 and D, it holds that

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

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

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

We assume that on input x, machine $C_{\alpha,\beta}$ 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 $(\alpha',\beta')\neq(\alpha,\beta)$.

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

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

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

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

Observe that any $\mathrm{poly}(k)$-time algorithm S that 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, it holds that

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

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:

$$\Pr\left[A(\mathcal{O}(Z_k),\mathcal{O}(D_{\alpha,\beta}))=1\right]=2^{-k}, \tag{3}$$

since $D_{\alpha,\beta}(Z_k)=1$ if and only if $\beta=0^k$. The combination of Equations (1), (2), and (3) contradict the hypothesis that $\mathcal{O}$ is a 2-TM obfuscator.

In the foregoing 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 the functionality requirement of the obfuscator, (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 assures us that (c) has the same functionality as $C_{\alpha,\beta}(\alpha)$. By the polynomial slowdown property of obfuscators, execution (c) only takes $\mathrm{poly}(10\cdot k)=\mathrm{poly}(k)$ steps, which means that $D_{\alpha,\beta}(\mathcal{O}(C_{\alpha,\beta}))$ need only run for $\mathrm{poly}(k)$ steps. Thus, again applying the polynomial slowdown property, execution (b) takes $\mathrm{poly}(k)$ steps, which finally implies that A need only run for $\mathrm{poly}(k)$ steps. The same reasoning holds for Equation (3), using Z_k instead of $C_{\alpha,\beta}$.[13] 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 (as before) an adversary A by $A(C,D)=D(C)$. (Note that the issues of A and $D_{\alpha,\beta}$'s running times disappears 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 at least as large as the size of C (note that one can always pad the description of C to a longer length). Thus, for the analogues of Equations (1) and (3) to hold, the input length of $D_{\alpha,\beta}$ must be at least as large as the sizes of the *obfuscations* of $C_{\alpha,\beta}$ and Z_k. However, by the polynomial slowdown property of

[13] 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 $\Omega(k)$.

obfuscators, it suffices to let $D_{\alpha,\beta}$ have input length $\mathrm{poly}(k)$ and the proof works as before.

■

3.2. Obfuscating one TM/circuit

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

Combining functions and programs.. For functions, TMs, or circuits $f_0, f_1 : X \to Y$, define their *combination* $f_0 \# f_1 : \{0,1\} \times X \to Y$ by $(f_0 \# f_1)(b,x) \stackrel{\mathrm{def}}{=} f_b(x)$. Conversely, if we are given a TM (resp., circuit) $C : \{0,1\} \times X \to Y$, we can efficiently decompose C into $C_0 \# C_1$ by setting $C_b(x) \stackrel{\mathrm{def}}{=} C(b,x)$; 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 $G_{\alpha,\beta} = Z_k \# D_{\alpha,\beta}$.

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 $\mathrm{poly}(|F|)$-time.) Let S be the PPT simulator for A guaranteed by Definition 2.1. Just as in the proof of Proposition 3.4, we have:

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

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. □

The difficulty in the circuit setting.. 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 at the end of the proof of Proposition 3.4.) In the proof of Theorem 3.5, 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 and F_1 in the way we described above (i.e., $F_b(x) \stackrel{\mathrm{def}}{=} \mathcal{O}(F_{\alpha,\beta})(b,x)$), then we shall 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 the modified $D_{\alpha,\beta}$ should not reveal α and β, because we do not want the simulator S to be able to test whether $C(\alpha) = \beta$ when given just oracle access to C and $D_{\alpha,\beta}$. We will construct such functions $D_{\alpha,\beta}$ based on pseudorandom functions [GGM]. The construction, captured in the following lemma, is the technical core of our main results.

LEMMA 3.6. *If one-way functions exist, then for every $k \in \mathbb{N}$ and $\alpha, \beta \in \{0,1\}^k$, there exists a distribution $\mathcal{D}_{\alpha,\beta}$ on circuits such that the following conditions hold.*

(*1*) *Every $D \in \mathrm{Supp}(\mathcal{D}_{\alpha,\beta})$ is a circuit of size $\mathrm{poly}(k)$. Furthermore, there exists a probabilistic polynomial-time algorithm that, for every $k \in \mathbb{N}$, on input $\alpha, \beta \in \{0,1\}^k$, samples the distribution $\mathcal{D}_{\alpha,\beta}$.*
(*2*) *There is a polynomial-time algorithm A such that for every $k \in \mathbb{N}$, $\alpha, \beta \in \{0,1\}^k$ and $D \in \mathrm{Supp}(\mathcal{D}_{\alpha,\beta})$, and for every circuit C, if $C(\alpha) = \beta$, then $A^D(C, 1^k) = \alpha$.*
(*3*) *For any PPT S, it holds that $\Pr\left[S^D(1^k) = \alpha\right] = \mathrm{neg}(k)$, 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.*

The crucial aspect about this lemma is that Item 2 refers also to circuits C that are larger than the length of the input to D. Note that Item 2 is seemingly stronger than required for our applications. For starters, the algorithms considered in our applications will get the code of D, whereas the algorithm A asserted in Item 2 only uses D as a black-box (i.e., A is given oracle access to D). Second, the algorithm A in Item 2 is able to obtain α whenever $C(\alpha) = \beta$, whereas for proving the impossibility of circuit obfuscators (as per Definition 2.2) it suffices to be able to distinguish $C_{\alpha,\beta}$ from Z_k (which is easy to do, provided that $\beta \neq 0^k$, by invoking C on A's output).[14] We note that the stronger version of Item 2 will be used to construct unobfuscatable circuits (as per Definition 3.1).

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 (and gives out α if this is the case). These operations will enable efficiently testing whether a given circuit C satisfies $C(\alpha) = \beta$, while keeping α and β hidden from parties that are only provided with oracle access to C and $D_{\alpha,\beta}$.

We shall use an arbitrary (probabilistic) private-key encryption scheme $(\mathrm{Enc}, \mathrm{Dec})$ that encrypts a single bit in a way that is secure 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. We call such schemes *CCA1-secure*, and refer the reader to formal definition provided in [KY; Gol2]. We note that CCA1-secure private-key encryptions schemes exist if (and only if) one-way functions exist; indeed, the "standard" (PRF-based) encryption scheme $\mathrm{Enc}_K(b) = (r, f_K(r) \oplus b)$, where $r \stackrel{R}{\leftarrow} \{0,1\}^{|K|}$ and $f_K : \{0,1\}^{|K|} \to \{0,1\}$ is a pseudorandom function, is CCA1-secure.

Now we consider a "homomorphic encryption" oracle Hom_K, which depends on a private-key K, and note that such an oracle can be implemented by a polynomial-size circuit (which depends on K). When queried on two ciphertexts c and d (w.r.t this key K) and a binary Boolean operation $\odot$ (specified by its 2×2 truth table), the oracle returns an encryption of $\mathrm{Dec}_K(c) \odot \mathrm{Dec}_K(d)$ under the key K. That is, we define

$$\mathrm{Hom}_K(c, d, \odot) \stackrel{\mathrm{def}}{=} \mathrm{Enc}_K(\mathrm{Dec}_K(c) \odot \mathrm{Dec}_K(d)). \tag{4}$$

[14] Indeed, if $C(\alpha) = \beta$ (e.g., if $C = C_{\alpha,\beta}$), then $C(A^D(C, 1^k)) = C(\alpha) = \beta$, whereas $Z_k(A^D(Z_k, 1^k)) = 0^k$.

It can be shown that any CCA1-secure encryption scheme is secure in a setting in which the adversary is given access to the corresponding Hom oracle.[15] This is formalized in the following claim:

CLAIM 3.6.1. *Let* $(\mathrm{Enc}, \mathrm{Dec})$ *be a CCA1-secure private-key encryption scheme and let* Hom_K *be as in Equation* (4)*. Then, for every PPT* A*, it holds that*

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

where $K \stackrel{R}{\leftarrow} \{0,1\}^k$.

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 by a hybrid argument, which relies on the CCA1-security of $(\mathrm{Enc}, \mathrm{Dec})$. For each $\sigma \in \{0,1\}$, consider a computation of A on input $\mathrm{Enc}_K(\sigma)$, when given oracle access to both Hom_K and Enc_K. Consider hybrids such that, in the i^{th} hybrid, the first i oracle queries are answered according to Hom_K, and the rest are answered with encryptions of 0. Then, any gap between the output distributions in the i^{th} and $i+1^{\mathrm{st}}$ hybrids must be due to the way the $i+1^{\mathrm{st}}$ query is answered, where in the i^{th} hybrid the answer is always an encryption of 0 and in the $i+1^{\mathrm{st}}$ hybrid (by the existence of a gap) the answer is an encryption of 1. Thus, we derive a contradiction to CCA1-security as follows. Prior to even receiving the challenge ciphertext, we invoke A on input $\mathrm{Enc}_K(\sigma)$, and answer the first i queries to Hom_K by emulating its operation via Dec_K and Enc_K queries (which are made before we get the challenge ciphertext). Next, we obtain the challenge ciphertext (which is either $\mathrm{Enc}_K(0)$ or $\mathrm{Enc}_K(1)$) and use it as our answer to the $i+1^{\mathrm{st}}$ query of A (to Hom_K), and finally we answer all subsequent queries to Hom_K by querying Enc_K for encryptions of 0. Thus, a gap between the i^{th} and $i+1^{\mathrm{st}}$ hybrids translates to a gap that violates the CCA1-security of Enc.

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, which in particular contradicts the CCA1-security of Enc. □

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

$$D_{K,\alpha,\beta} \stackrel{\mathrm{def}}{=} E_{K,\alpha}\#\mathrm{Hom}_K\#B_{K,\alpha,\beta} \tag{5}$$

(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 a standard 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

[15] Note that the Hom oracle can be used to implement an encryption oracle (by feeding it with a constant operation), but for sake of clarity we use a redundant phrasing in the following claim.

select a pseudorandom function $f_{K'} : \{0,1\}^q \to \{0,1\}^m$, and let $D'_{K,\alpha,\beta,K'}$ be the (deterministic) algorithm that 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 a circuit of size $\mathrm{poly}(k)$, and sampling $\mathcal{D}_{\alpha,\beta}$ is easy, so Property 1 is satisfied.

For Property 2, consider an algorithm A that on input a circuit C and oracle access to $D'_{K,\alpha,\beta,K'}$ (which provides access to (deterministic versions of) the three separate oracles $E_{K,\alpha}$, Hom_K, and $B_{K,\alpha,\beta}$), proceeds as follows: First, with k oracle queries to the $E_{K,\alpha}$ oracle, A obtains encryptions of each of the bits of α. Next, 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 (when C is evaluated on input α). Finally, A feeds these output encryptions to $B_{K,\alpha,\beta}$, and outputs the response to this oracle query. By construction, A outputs α if $C(\alpha) = \beta$, since in this case A feeds $B_{K,\alpha,\beta}$ with a sequence that decrypts to β.

Last, 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 $B_{K,\alpha,\beta}$. Since β is independent of α and K (whereas only $B_{K,\alpha,\beta}$ depends on β), the probability that S queries $B_{K,\alpha,\beta}$ 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 $B_{K,\alpha,\beta}$ 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 3.6.1), by a hybrid argument: Specifically, using Claim 3.6.1, a hybrid argument shows that access to the oracles $E_{K,\alpha}$ and Hom_K can be replaced with access to the oracles $E_{K,0^k}$ and Hom_K, while causing only a negligible difference in the success probability of S. (The hybrids range over each bit of α and each of the polynomially many queries S can make to the $E_{K,\alpha}$ oracle.) Once this replacement is done, S has no information regarding α, which was chosen uniformly at random from $\{0,1\}^k$. Thus, the probability that S outputs α is negligible. ■

On the impossibility of circuit obfuscators.. Using Lemma 3.6, we obtain our main impossibility result.

THEOREM 3.7. *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 $\{0,1\}^k$, $D \xleftarrow{R} \mathcal{D}_{\alpha,\beta}$. Output $C_{\alpha,\beta}\#D$.

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

Let A be the PPT algorithm guaranteed by Property 2 in Lemma 3.6, and consider a PPT A' that, on input a circuit F, decomposes $F = F_0\#F_1$ and outputs 1 if

$F_0(A^{F_1}(F_0, 1^k)) \neq 0^k$, 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 $C(A^D(C, 1^k))$ 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, and accounting for the probability that $\beta = 0^k$, we have that:

$$\begin{aligned} \Pr\left[A'(\mathcal{O}(\mathcal{F}_k)) = 1\right] &= 1 - 2^{-k}, \text{ and} \\ \Pr\left[A'(\mathcal{O}(\mathcal{G}_k)) = 1\right] &= 0. \end{aligned}$$

We now argue that for any PPT algorithm S, it holds that

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

which will contradict the definition of circuit obfuscators. This claim holds since 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 $\{0,1\}^k$. Now, 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 the case that this oracle is $C_{\alpha,\beta}$ from it being Z_k. ■

Avoiding the assumption.. When wishing to prove the impossibility of *efficient* circuit obfuscators, we can avoid the assumption that one-way functions exist. This is the case, since the existence of the former implies the existence of the latter.

LEMMA 3.8. *If efficient circuit 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

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

Now define $f_k(\alpha, b, r) \stackrel{\mathrm{def}}{=} \mathcal{O}(C_{\alpha,b}; r)$, i.e., the obfuscation of $C_{\alpha,b}$ using coin tosses r. We will show that $f = \bigcup_{k\in\mathbb{N}} f_k$ is a one-way function. Since $\mathcal{O}$ is efficient, it follows that f_k can be evaluated in $\mathrm{poly}(k)$-time. Next note that the functionality property of $\mathcal{O}$ implies that the bit b is (information-theoretically) determined by $f_k(\alpha, b, r)$, which in turn implies that if b is a hardcore bit of f_k, then f_k must be (strongly) hard to invert (since any f_k-preimage of $f_k(\alpha, b, r)$ must have the form $(\cdot, b, \cdot)$). To prove that b is a hardcore bit, we first observe that for any PPT S, it holds that

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

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

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

The lemma follows. □

COROLLARY 3.9. *Efficient circuit obfuscators do not exist (unconditionally).*

PROOF. Assuming, towards the contradiction, that such obfuscators exist, we infer (by Lemma 3.8) that one-way functions exist, reaching a contradiction to Theorem 3.7. ■

On the existence of unobfuscatable circuit ensembles.. We now strengthen our result to not only rule out circuit obfuscators, but actually yield unobfuscatable programs.

THEOREM 3.10 (UNOBFUSCATABLE PROGRAMS). *If one-way functions exist, then there exists an unobfuscatable circuit ensemble.*

PROOF. Again, for $k \in \mathbb{N}$ and $\alpha, \beta \in \{0,1\}^k$, let $C_{\alpha,\beta}$ 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. Our unobfuscatable circuit ensemble $\mathcal{H}_k$ is defined as follows.

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

(Indeed, $C_{\alpha,(D,\gamma)}$ is the circuit that on input α outputs (D,γ), and on all other inputs outputs $0^{|(D,\gamma)|}$.)[16]

Efficiency is clearly satisfied. For unlearnability, we define $\pi(C_{\alpha,\beta}\#D\#C_{\alpha,(D,\gamma)}) = \gamma$. Let's verify that γ is pseudorandom when given oracle access to $C_{\alpha,\beta}\#D\#C_{\alpha,(D,\gamma)}$. That is, for every PPT S, we prove that

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

Having oracle access to a circuit from $\mathcal{H}_k$ is equivalent to having oracle access to $C_{\alpha,\beta}$, D, and $C_{\alpha,(D,\gamma)}$, where $D \xleftarrow{R} \mathcal{D}_{\alpha,\beta}$ and α, β, and γ are selected uniformly in $\{0,1\}^k$. Property 3 of Lemma 3.6 implies that the probability that any PPT S queries either of the two $C_{\alpha,\cdot}$-oracles at α and thus gets a nonzero response is negligible. Note that this holds even if the PPT S is given γ as input, because the probabilities in Lemma 3.6 are taken over only α, β, and $D \xleftarrow{R} \mathcal{D}_{\alpha,\beta}$, so we can view S as choosing γ on its own. Thus,

$$\begin{aligned}
\Pr_{f \xleftarrow{R} \mathcal{H}_k} [S^f(\pi(f)) = 1] &= \Pr_{\alpha,\beta,\gamma \xleftarrow{R} \{0,1\}^k, D \xleftarrow{R} \mathcal{D}_{\alpha,\beta}} [S^{C_{\alpha,\beta}\#D\#C_{\alpha,(D,\gamma)}}(\gamma) = 1] \\
&= \Pr_{\alpha,\beta,\gamma,\gamma' \xleftarrow{R} \{0,1\}^k, D \xleftarrow{R} \mathcal{D}_{\alpha,\beta}} [S^{C_{\alpha,\beta}\#D\#C_{\alpha,(D,\gamma')}}(\gamma) = 1] \pm \mathrm{neg}(k) \\
&= \Pr_{f \xleftarrow{R} \mathcal{H}_k, z \xleftarrow{R} \{0,1\}^k} [S^f(z) = 1] \pm \mathrm{neg}(k).
\end{aligned}$$

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 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. EXTENSIONS

4.1. 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.

[16]Indeed, we could have used $C_{\alpha,(\beta,D,\gamma)}\#D$ instead of the two C-oracles, but the chosen alternative is more convenient in the rest of the proof.

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.

4.1.1. 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 only on the coin tosses of the obfuscating algorithm (rather than over the choice of a randomly chosen input).

Definition 4.1. For any function $f : \{0,1\}^n \to \{0,1\}^k$, $\epsilon > 0$, the random variable D is called an *ϵ-approximate implementation* of f if the following holds:

(1) D ranges over circuits from $\{0,1\}^n$ to $\{0,1\}^k$
(2) For any $x \in \{0,1\}^n$, it holds that $\Pr_D[D(x) = f(x)] \geq 1 - \epsilon$

We then define a strongly unobfuscatable circuit ensemble to be an unobfuscatable circuit ensemble where the original circuit C can be reconstructed not only from any circuit that computes the same function as C but also from any approximate implementation of C.

Definition 4.2. A *strongly unobfuscatable circuit ensemble* $\{\mathcal{H}_k\}_{k\in\mathbb{N}}$ is defined in the same way as an unobfuscatable condition ensemble, except that the "reverse-engineerability" condition is strengthened as follows:

— (strong reverse-engineerability) C is easy to reconstruct given an approximate implementation: There exists a PPT A and a polynomial $p(\cdot)$ such that for every $C \in \bigcup_{k\in\mathbb{N}} \mathrm{Supp}(\mathcal{H}_k)$ and for every random variable C' that is an ϵ-approximate implementation of the function computed by C, we have

$$\Pr[A(C') = C] \geq 1 - \epsilon \cdot p(k)$$

Our main theorem in this section is the following:

THEOREM 4.3. *If one-way functions exist, then there exists a strongly unobfuscatable circuit ensemble.*

Similarly to the way that Theorem 3.10 implies Theorem 3.7, Theorem 4.3 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 in 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. (This is implied by, but apparently weaker than, the requirement of Definition 4.1 — if C is an ε-approximate implementation of f, then for any fixed distribution D on inputs, C and f agree on a $1 - \sqrt{\varepsilon}$ fraction of D with probability at least $1 - \sqrt{\varepsilon}$.) We do not know whether approximate obfuscators with respect to this weaker notion exist, and leave it as an open problem.

The natural strategy towards proving Theorem 3.10 is to generalize the proof of Theorem 3.10. We shall first see why the proof of Theorem 3.10 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.

4.1.2. Generalizing the Proof of Theorem 3.10 to the Approximate Case. The first question is whether the proof of Theorem 3.10 already shows that the ensemble $\{\mathcal{H}_k\}_{k\in\mathbb{N}}$ defined there is actually a *strongly* unobfuscatable circuit ensemble. As we explain below, the answer is no.

To see why, let us recall the definition of the ensemble $\{\mathcal{H}_k\}_{k\in\mathbb{N}}$ given in the proof of Theorem 3.10. The distribution $\mathcal{H}_k$ was defined by choosing $\alpha,\beta,\gamma \stackrel{\mathrm{R}}{\leftarrow} \{0,1\}^k$ and a function $D \stackrel{\mathrm{R}}{\leftarrow} \mathcal{D}_{\alpha,\beta}$, and outputting $C_{\alpha,\beta}\#D\#C_{\alpha,(D,\gamma)}$. The proof gave an algorithm, denoted A', that reconstructs $C \in \mathcal{H}$ given any circuit that computes exactly the same function as C. Let us see why A' might fail when given only an approximate implementation of C. On input a circuit F, algorithm A' works as follows: It decomposes F into three circuits $F = F_1\#F_2\#F_3$. Then F_2 and F_3 are used only in a black-box manner, but the queries A' makes to F_2 depend on the gate structure of the circuit F_1.

The problem is that a vicious approximate implementation for a function $C_{\alpha,\beta}\#D\#C_{\alpha,(D,\gamma)} \in \mathrm{Supp}(\mathcal{H}_k)$ may work in the following way: Choose a random circuit F_1 out of some set $\mathcal{C}$ of exponentially many circuits that compute $C_{\alpha,\beta}$, take F_2 that computes D, and F_3 that computes $C_{\alpha,(D,\gamma)}$. Then see at which points A' queries F_2 when given $F_1\#F_2\#F_3$ as input.[17] Since these places depend on F_1, it is possible that for each $F_1 \in \mathcal{C}$, there exists a string $x(F_1)$ such that A' will query F_2 at $x(F_1)$, 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 $F_1\#F_2\#F_3$ 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. (This already holds for F_3, which is only queried at α in a correct computation.) If A' had this property, then given an ϵ-approximate implementation of $C_{\alpha,\beta}\#D\#C_{\alpha,(D,\gamma)}$, 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 $\mathcal{D}_{\alpha,\beta}$ that is defined in the proof of Lemma 3.6. We defined there the algorithm

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

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 $D'_{K,\alpha,\beta,K'} = E'_{K,\alpha,K'}\#\mathrm{Hom}'_{K,K'}\#B_{K,\alpha,\beta}$ 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, denoted E, H, and B, such that if F_2 computes $D = E'_{K,\alpha,K'}\#\mathrm{Hom}'_{K,K'}\#B_{K,\alpha,\beta}$, 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 $B_{K,\alpha,\beta}$. 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 that A' makes to the H oracle is 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, algorithm A' can generate a random encryption of the same bit (i.e., distributed

[17] Recall that A' is not an arbitrary algorithm (which we must treat as a black-box), but rather a very specific algorithm (postulated in Theorem 3.10, and presented in its proof).

according to $\mathrm{Enc}_K(\mathrm{Dec}_K(c), r)$ for uniformly selected r). For instance, this would be true if the encryption scheme were rerandomizable (or in other words, "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* $(c, d, \odot)$ *depends on the circuit* F. The reason is that the distribution of the modified query $(c', d', \odot)$ depends only on $(\mathrm{Dec}_K(c), \mathrm{Dec}_K(d), \odot) \in \{0,1\} \times \{0,1\} \times \{0,1\}^{2 \cdot 2}$, and there are only 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 such that A' will essentially have this desired ability of rerandomizing 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:

$$\mathrm{Hom}'_{K,K'}(c, d, \odot, r) = \mathrm{Enc}_K(\mathrm{Dec}_K(c) \odot \mathrm{Dec}_K(d); f_{K'}(c, d, \odot, r)) \tag{6}$$

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

4.1.3. Invoker-Randomizable Pseudorandom Functions. The property we would like the pseudorandom function $f_{K'}$ to possess is one that makes $f_{K'}(x, r)$ random when only r is random.

Definition 4.4. A function ensemble $\{f_{K'}\}_{K' \in \{0,1\}^*}$ such that $f_{K'} : \{0,1\}^{q+n} \to \{0,1\}^n$, where n and q are 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 every K' and $x \in \{0,1\}^q$, the mapping $r \mapsto f_{K'}(x, r)$ is a permutation over $\{0,1\}^n$.

Property 2 implies that, for every fixed K' and $x \in \{0,1\}^q$, if r is chosen uniformly in $\{0,1\}^n$, then the value $f_{K'}(x, r)$ is distributed uniformly in $\{0,1\}^n$ (and in particular is independent of x).

LEMMA 4.5. *If pseudorandom functions exist, then there exist invoker-randomizable pseudorandom functions.*

Proof Sketch: Let $\{g_{K'} : \{0,1\}^q \to \{0,1\}^{|K'|}\}_{K' \in \{0,1\}^*}$ be a pseudorandom function ensemble and $\{p_S : \{0,1\}^n \to \{0,1\}^n\}_{S \in \{0,1\}^*}$ be a pseudorandom function ensemble such that, for any $S \in \{0,1\}^{|K'|}$, the function p_S is a permutation over $\{0,1\}^n$. (The existence of the latter ensembles is implied by the existence of ordinary pseudorandom function ensembles [LR].) We define the function ensemble $\{f_{K'} : \{0,1\}^{q+n} \to \{0,1\}^n\}_{K' \in \{0,1\}^*}$ 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 4.4. What needs to be shown is that it is a pseudorandom function ensemble. This is done by using a hybrid argument, in which we consider the following intermediate hybrids.

(1) The function ensemble $\{f'_G : \{0,1\}^{q+n} \to \{0,1\}^n\}_G$ such that $f'_G(x,r) \stackrel{\text{def}}{=} p_{G(x)}(r)$, where $G : \{0,1\}^q \to \{0,1\}^{|K'|}$ is a random function.

(2) The function ensemble $\{f''_{G,\overline{F}} : \{0,1\}^{q+n} \to \{0,1\}^n\}_{G,\overline{F}}$ such that $f''_{G,\overline{F}}(x,r) \stackrel{\text{def}}{=} F_{G(x)}(r)$, where $\overline{F} = (F_{0^{|K'|}}, ..., F_{1^{|K'|}})$ such that the F_S's are random functions from $\{0,1\}^n$ to $\{0,1\}^n$.

The indistinguishability of our main function ensemble (i.e., $\{f_{K'}\}$) and $\{f'_G\}$ follows from the pseudorandomness of the ensemble $\{g_{K'}\}$. The indistinguishability of $\{f'_G\}$ and $\{f''_{G,\overline{F}}\}$ follows from the pseudorandomness of the ensemble $\{p_S\}$. Finally, note that $\{f''_{G,\overline{F}}\}$ is identical to a random function from $\{0,1\}^{q+n}$ to $\{0,1\}^n$. □

4.1.4. Finishing the Proof of Theorem 4.3. 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. Recall that H is an ϵ-approximate implementation of $\mathrm{Hom}'_{K,K'}$ as defined in Equation (6), whereas $\mathrm{Hom}'_{K,K'}$ answers such a (randomized) query with a *random* encryption of $\mathrm{Dec}_K(c) \odot \mathrm{Dec}_K(d)$. Therefore, with probability at least $1 - p(k) \cdot \epsilon$ (for some polynomial $p(\cdot)$), algorithm A' gets correct answers for all its queries to $F_2 = E\#H\#B$. This holds because of the following considerations.

(1) The queries made to E are fixed, and therefore independent of the gate structure of F_1. Thus, each such query is answered correctly with probability at least $1 - \epsilon$.

(2) Assuming all answers received so far are correct, we consider each query made to the H oracle. Such a query is 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. Therefore, with probability at least $1 - 64\epsilon$, this query will be answered correctly by an ϵ-approximator of $\mathrm{Hom}'_{K,K'}$, even if it knows $(a,b,\odot)$.

(3) Assuming A' never gets an incorrect answer from the E and H oracles, its last query (i.e., its 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.

The claim follows, and this completes the proof of Theorem 4.3.

One point to note is that we have converted our deterministic algorithm A' of Theorem 3.10 into a *probabilistic* algorithm.

4.2. Impossibility of the Applications

So far, we have 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 will show now that this is not the case for some 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 (i.e., an algorithm satisfying the syntactic requirements of Definition 2.2) that can be used to achieve them in the ways

that are described in Section 1.1. Our results in this section also extend to approximate obfuscators.

Consider, for example, the application to transforming private-key encryption schemes into public-key ones. Recall that the transformation consists of obfuscating the private-key encryption algorithm E_K, and releasing its obfuscation $\widetilde{E_K}$ as a public key. This approach will fail for a private-key encryption scheme (G, E, D) that is *unobfuscatable* in the sense that there exists a polynomial-time algorithm A such that for every key $K \in \mathrm{Supp}(G(1^k))$ and every circuit $\widetilde{E_K}$ that computes the encryption function E_K it holds that $A(\widetilde{E_K}) = K$.

The foregoing text refers implicitly to deterministic encryption algorithms, whereas such schemes cannot offer a robust notion of security (i.e., semantic security under chosen-plaintext attack). Indeed, as noted in [GM], a robust notion of security requires the encryption algorithm to be probabilistic. Thus, our definition of unobfuscatable encryption schemes should apply to probabilistic encryption, and refer to the distribution[18] generated by the original encryption process E_K and ditto its potential obfuscation, denoted $\widetilde{E_K}$.

Definition 4.6. A probabilistic private-key encryption scheme (G, E, D) is called *unobfuscatable* if there exists a polynomial-time algorithm A such that, for every key $K \in \mathrm{Supp}(G(1^k))$ and for every randomized circuit $\widetilde{E_K}$ such that $\widetilde{E_K}(m)$ and $E_K(m)$ are identically distributed for each message m, it holds that $A(\widetilde{E_K}) = K$.

The requirement that $\widetilde{E_K}(m)$ and $E_K(m)$ be identically distributed can be relaxed to only require that they are statistically close (and our result can be extended to these cases).[19] We mention that related definitions of obfuscators for probabilistic circuits are discussed in Section 6.

In Theorem 4.10 below, we prove that if secure private-key encryption schemes exist, then so do unobfuscatable encryption schemes that satisfy the same security requirements. This means that any definition of an obfuscator that will be strong enough to allow the conversion of an arbitrary secure private-key encryption scheme into a secure public-key encryption scheme will be impossible to meet (because there exist unobfuscatable encryption schemes). 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 generic way to transform a private-key encryption scheme into a public-key encryption scheme by obfuscating the encryption algorithm.

We present analogous definitions for unobfuscatable signature schemes, MACs, and pseudorandom functions. (For these, the restriction to deterministic schemes is insignificant, since any probabilistic signature/MAC scheme can be converted into a deterministic one; see [Gol2, Sec. 6.1.5.2].)

---

[18] An alternative approach could consider E_K as a deterministic function of its message m and coin tosses r, and require $\widetilde{E_K}$ to compute the same deterministic function. But then unobfuscatable encryption schemes would exist for trivial and uninteresting reasons: Observe that if E'_K is a secure private-key encryption scheme, then so is $E_K(m; r_1, r_2) = (E'_K(m; r_1), r_2 \oplus K)$, but we can recover K from any circuit (or even oracle) that computes the function $(m, r_1, r_2) \mapsto E_K(m; r_1, r_2)$. Indeed, this case provides a good demonstration of the difference between obfuscating a distribution and obfuscating the function underlying the distribution's generation process. This topic is further pursued in Section 6.

[19] Furthermore, our results would also extend if the requirement were relaxed to only require that $(m, K, \widetilde{E_K}(m))$ and $(m, K, E_K(m))$ be computationally indistinguishable.

Definition 4.7. A deterministic signature scheme (G, S, V) is called *unobfuscatable* if there exists a polynomial-time algorithm A such that for every key $(SK, VK) \in \mathrm{Supp}(G(1^k))$ and every circuit $\widetilde{S_{SK}}$ that computes the signature function with signing key SK, it holds that $A(\widetilde{S_{SK}}) = SK$.

Definition 4.8. A deterministic message authentication scheme (G, S, V) is called *unobfuscatable* if there exists a polynomial-time algorithm A such that for every key $K \in \mathrm{Supp}(G(1^k))$ and every circuit $\widetilde{S_K}$ that computes the tagging function with tagging key K, it holds that $A(\widetilde{S_K}) = K$.

Definition 4.9. A pseudorandom function ensemble $\{h_K\}_{K\in\{0,1\}^*}$ is called *unobfuscatable* if there exists a polynomial-time algorithm A such that for every key $K \in \{0,1\}^*$ and every circuit $\widetilde{h_K}$ that computes h_K, it holds that $A(\widetilde{h_K}) = K$.

THEOREM 4.10 (IMPOSSIBILITY OF SOME BASIC APPLICATIONS).

(*1*) *If there exist secure probabilistic private-key encryption schemes, then there exist ones that are unobfuscatable.*
(*2*) *If there exist secure deterministic signature schemes, then there exist ones that are unobfuscatable.*
(*3*) *If there exist secure deterministic message authentication schemes, then there exist ones that are unobfuscatable.*
(*4*) *If there exist secure pseudorandom function ensembles, then there exist ones that are unobfuscatable.*

Proof Sketch: First note that the existence of any one of these primitives implies the existence of one-way functions [IL]. Therefore, Theorem 3.10 gives us a totally unobfuscatable circuit ensemble $\mathcal{H} = \{\mathcal{H}_k\}$.

Now, we begin with the construction of the unobfuscatable signature schemes. 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 $C \stackrel{R}{\leftarrow} \mathcal{H}_\ell$, where $\ell = |SK|$. The new signing key SK' is (SK, C) while the verification key VK' is the same as VK.

We can now define

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

where π is the function from the unlearnability condition in Definition 3.1.

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

We claim that (G', S', V') is an unobfuscatable, yet secure, signature scheme. To see that (G', S', V') is unobfuscatable, observe that given any circuit that computes $S'_{SK,f}$, one can obtain the string $SK \oplus \pi(C)$ as well as a circuit that computes the same function as C. By the reverse-engineering condition of $\mathcal{H}_\ell$, possession of the latter enables the reconstruction of the original circuit C itself, from which $\pi(C)$ and then SK can be computed.

To see that scheme (G', S', V') retains the security of the scheme (G, S, V), observe that given oracle access to $S'_{SK,C}$ is equivalent to being given oracle access to S_{SK} and C along with the string $\pi(C) \oplus SK$ itself. Using the facts that $\pi(C)$ is indistinguishable from random, when given oracle access to C, and that C is chosen independently of

SK, it can be shown that the presence of C and $\pi(C) \oplus SK$ does not help an adversary break the signature scheme.

We now turn to construct unobfuscatable pseudorandom functions and MACs. The key observation here is that the proof of Theorem 3.10 can be modified to give an unobfuscatable circuit ensemble that is also a family of pseudorandom functions (and hence also a secure message authentication code). Recall that $\mathcal{H}_k = C_{\alpha,\beta}\#D\#C_{\alpha,(D,\gamma)}$, where $\alpha, \beta, \gamma \stackrel{R}{\leftarrow} \{0,1\}^k$ and $D \stackrel{R}{\leftarrow} \mathcal{D}_{\alpha,\beta}$, and that D is a deterministic version of $E_{K,\alpha}\#\mathrm{Hom}_K\#B_{K,\alpha,\beta}$, where $K \stackrel{R}{\leftarrow} \{0,1\}^k$ (and the derandomization is obtained by using an auxiliary pseudorandom function). We first observe that we may assume that the encryption scheme in use (i.e., in $E_{K,\alpha}$ and Hom_K) has pseudorandom ciphertexts. Indeed, this is the case in the scheme (cited for demonstration) in the proof of Lemma 3.6. Thus, the outputs of $E_{K,\alpha}$ and Hom_K are pseudorandom. As for the outputs of the other parts of the circuit, in the original proof they were set to produce zero on almost all inputs, and it was shown that a PPT machine having oracle access to them can hit an exceptional input (which was assigned a non-zero value) only with negligible probability. Thus, our modification will consist of using another pseudorandom function, and defining the value of most inputs (i.e., those set to zero before) to equal the value of that pseudorandom function (on the corresponding input). Clearly, the argument used in the proof of Theorem 3.10 remain valid, and so it follows that the modified ensemble is both unobfuscatable and pseudorandom.

Last, we turn to construct an unobfuscatable private-key encryption scheme. To this end, we will modify (a simplified version of) the construction from the proof of Theorem 3.10. Recall that in that construction (or actually in the proof of Lemma 3.6), the algorithm $D_{K,\alpha,\beta}$ was converted from a probabilistic algorithm into a deterministic one because the goal there was to rule out obfuscation for ordinary (deterministic) circuits. Since we are now considering randomized circuits, we consider a simplified version where $D_{K,\alpha,\beta}$ is implemented by a randomized circuit where the randomness required for the encryptions are provided by randomness gates in the circuit, instead of using pseudorandom functions applied to the input.[20] Let $T_{\alpha,\beta,\gamma,K}$ denote the randomized circuit $C_{\alpha,\beta}\#D_{K,\alpha,\beta}\#C_{\alpha,(D_{K,\alpha,\beta},\gamma)}$ that is the final result of the construction.

Now let (G, E, D) be a semantically secure private-key encryption scheme, and define a new scheme (G', E', D') as follows: A key for the new scheme is generated (by G') by obtaining a secret key SK from G, and selecting uniformly $\alpha, \beta, \gamma, K \in \{0,1\}^k$, where $k = |SK|$. Encryption, based on the key $(SK, \alpha, \beta, \gamma, K)$, is defined by

$$E'_{SK,\alpha,\beta,\gamma,K}(m) \;=\; (E_{SK}(m), T_{\alpha,\beta,\gamma,K}(m), \gamma \oplus SK).$$

The decryption procedure D' simply runs D on the first component of the ciphertext.

The semantic security of (G', E', D') follows from the semantic security of the private-key encryption used in the construction, and from the fact that for almost all inputs, the output of the randomized circuit $C_{\alpha,\beta}\#D_{K,\alpha,\beta}\#C_{\alpha,(D_{K,\alpha,\beta},\gamma)}$ computationally hides all information about the input. The only inputs for which this is not necessarily true are a negligibly small fraction of inputs that depend on the values α, β, and K. Since these values are part of the secret key of the scheme, this does not affect semantic security. □

[20] A minor issue that arises refers to the reverse-engineering condition, which utilizes an algorithm that is given the obfuscated circuit. This algorithm is supposed to emulate the computation of the circuit in order to recover the string α; see description at the end of the proof of Theorem 3.10. The issue is that in the current case the circuit has randomness gates, but we assert a deterministic recovering algorithm. This is possible because the recovering algorithm has perfect correctness, so it can treat all of the random bits of the circuit as zero.

Implication for applying obfuscation as a means to implement the Random Oracle Methodology.. A natural application of obfuscation would be to "implement" a random oracle by obfuscating a function chosen randomly from a pseudo-random function ensemble. Our construction of unobfuscatable pseudorandom function ensembles can be used to show that this approach would fail in the following sense: For many *natural* protocols that are secure in the random oracle model, there exists a (contrived) pseudorandom function ensemble such that if the random oracle is replaced with *any* public circuit that computes any function in that ensemble then the resulting protocol is insecure. One such example is provided by the Fiat–Shamir signature scheme [FS], which is obtained by removing interaction from a 3-round (public-coin) identification protocol, which in turn is derived from a honest-verifier zero-knowledge (hvZK) proof of knowledge of some secret that refers to a public record associated with the legitimate signer. Recall that the interaction is removed (in the Random Oracle Model) by applying the random function to the first message (and that the Random Oracle Methodology calls for replacing this random function by a public function).

PROPOSITION 4.11 (FAILURE OF A GENERIC VERSION OF THE FIAT–SHAMIR SIGNATURE). *Let (P,V) be an arbitrary 3-round public-coin hvZK protocol, and consider the signature scheme in which message m is signed by sending the transcript that corresponds to an interaction of $(P,V)(\rho)$, where ρ is the signer's public record/key and the verifier message in the interaction is replaced by the application of the Random Oracle to the pair (m,a) such that a is the first message sent by P. Suppose that one-way functions exists.*[21] *Then, there exists a pseudorandom function ensemble $\{h_K\}_{K\in\{0,1\}^*}$ such that replacing the random oracle in the foregoing scheme by* any *public circuit that computes any h_K, yields an insecure scheme, in the strong sense that an attacker can forge a valid signature given only the signer's public record/key.*

Proof Sketch: Starting with an arbitrary pseudorandom function ensemble, denoted $\{f_s\}$, we consider the function ensemble $\{f'_{s,m_0,r_0}\}$ defined (for polynomially related $|s|$, $|m_0|$, and $|r_0|$) by

$$f'_{s,m_0,r_0}(x\circ y,z)\stackrel{\rm def}{=}\begin{cases} S(y,r_0)_2 & \text{if } x=m_0 \text{ and } S(y,r_0)_1=z\\ f_s(x\circ y,z) & \text{otherwise}\end{cases}$$

where $S(y,r_0)_i$ is the $i^{\rm th}$ element in the transcript produced by the (honest-verifier) simulator on input y and using randomness r_0. Note that the resulting ensemble preserves the pseudorandomness of the original one, since the modified inputs are extremely rare (and the adversary lacks any information regarding their identity). On the other hand, given the seed of a function (i.e., s,m_0,r_0), it is easy to forge a signature for the message $m_0\circ\rho$ in the resulting signature proof (by letting $(a,b,c)\leftarrow S(\rho,r_0)$, where ρ is the prover/signer's public record/key).[22] Now we apply the construction outlined in the proof of Theorem 4.10, while using the pseudorandom ensemble $\{f'_{s,m_0,r_0}\}$ in order to assign pseudorandom values to the inputs that were set to zero in the proof of Theorem 3.10. The resulting unobfuscatable pseudorandom ensemble $\{h_K\}$ still allows the foregoing forging, since with high probability (over m_0) h_K will agree with f'_{s,m_0,r_0} on the input $z=(m_0\circ\rho,S(\rho,r_0))$. Indeed, by choosing m_0 a bit more carefully that this occurs with probability 1 (by making sure that all inputs with prefix m_0 lead to appying $C_{\alpha,\beta}$ on an input different than α). □

[21] Recall that if no one-way functions exist, then there exist no signature schemes anyhow.

[22] Note that in this case $b=f'_{s,m_0,r_0}(m_0\circ\rho,a)$, since $S(\rho,r_0)_1=a$. Thus, $(a,f'_{s,m_0,r_0}(m_0\circ\rho,a),c)$ is an accepting transcript of $(P,V)(\rho)$, which means that it constitutes a valid signature to m_0 (relative to the signer's record/key ρ).

We comment that the foregoing proof only relies on a very weak notion of simulation that only requires that on input y the simulator always outputs an accepting transcript. Furthermore, the proof extends to any public-coin protocol (with the foregoing weak simulation condition), implying that, *although it may be infeasible to generate accepting transcripts of the execution in the Random Oracle model, it is feasible to generate accepting transcripts of an execution that refers to replacing the random oracle by any public circuit computing any function in* $\{h_K\}$.[23] In Appendix C, we show that a similar, but weaker, attack applies when considering the Fiat–Shamir transformation applied to arbitrary public-coin identification protocols (including ones that do not admit the weak simulation property needed for the attack above).

4.3. 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 4.12. *If "factoring Blum integers is hard",*[24] *then there is a family $\mathcal{H}_k$ of unobfuscatable circuits such that every $C \stackrel{R}{\leftarrow} \mathcal{H}_k$ is a constant-depth threshold circuit of size* $\mathrm{poly}(k)$.

Below, we shall say that a circuit ensemble $\{\mathcal{H}_k\}$ is in $\mathbf{TC^0}$ if there exists a constant c such that, for every k, every $C \in \mathrm{Supp}(\mathcal{H}_k)$ is a threshold circuit of size at most $c \cdot k^c$ and depth at most c.

Proof Sketch: Naor and Reingold [NR] showed that under the stated assumption, there exists a family of pseudorandom functions computable in $\mathbf{TC^0}$. Thus, we simply need to check that we can build our unobfuscatable circuits from such a family without a substantial increase in depth. Recall that the unobfuscatable circuit ensemble $\mathcal{H}_k$ constructed in the proof of Theorem 3.10 consists of functions of the form $C_{\alpha,\beta}\#D\#C_{\alpha,(D,\gamma)}$, where D is from the family $\mathcal{D}_{\alpha,\beta}$ of Lemma 3.6. It is easy to see that $C_{\alpha,\beta}$ and $C_{\alpha,(D,\gamma)}$ 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 $f_{K'}$ that 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 (as detailed in the proof of Lemma 3.6),[25] then the resulting circuit is in $\mathbf{TC^0}$. □

4.4. Relativization

In this section, we address the question of whether or not 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 programs being obfuscated and produced, and including the adversary, have oracle access to F.

---

[23] In the proof of Proposition 4.11, we referred to a protocol in which the random oracle is applied to strings of the form $(m \circ \rho, \cdot)$, where ρ was the input to the protocol and m was an auxiliary input (representing an external message). In the general setting, we may refer to inputs of the form $x \circ y$ and to the task of generating an accepting transcript of $(P, V)(x \circ y)$, where y is fixed and the choice of x is at the adversary's discretion. We also mention that the argument extends to any public-coin protocol that satisfies the weak simulation condition, regardless of the number of rounds.

[24] We refer to the standard formulation of this assumption (see, e.g., [NR]). As shown in [NR], this assumption allows to construct pseudorandom functions in $\mathbf{TC^0}$. An alternative assumption that was shown in [NR] to suffice refers to the Decisional Diffie–Hellman problem.

[25] Recall that the basic scheme is $\mathrm{Enc}_K(b) = (r, f_K(r) \oplus b)$, where r is uniformly chosen in $\{0,1\}^{|K|}$.

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 impossibility results (i.e., Proposition 3.4 and Theorem 3.5) hold relative to any oracle, since they involve only simulation and diagonalization-like arguments.

PROPOSITION 4.13. *Proposition 3.4* (impossibility of 2-circuit obfuscators) *and Theorem 3.5* (impossibility of TM obfuscators) *hold relative to any oracle.*

Interestingly, our main impossibility results (i.e., Theorem 3.7, Theorem 3.10, and Corollary 3.9) do not relativize.

PROPOSITION 4.14. *There is an oracle relative to which efficient circuit obfuscators do exist. Thus, Theorems 3.7 and 3.10 and Corollary 3.9 do not relativize.*

This proposition can be viewed both as evidence that the foregoing 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 $F_k = \mathcal{O}_k \# \mathcal{E}_k$, where $\mathcal{O}_k : \{0,1\}^k \times \{0,1\}^k \to \{0,1\}^{6k}$ is a random (injective) function and $\mathcal{E}_k : \{0,1\}^{6k} \times \{0,1\}^k \to \{0,1\}^k$ is defined as follows: If there exists a (C, r) such that $\mathcal{O}_k(C, r) = x$, then $\mathcal{E}_k(x, y) = C^F(y)$, where C is viewed as the description of a circuit. Otherwise, $\mathcal{E}_k(x, y) = \perp$. 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 function $\mathcal{E}_k$ offers efficient evaluation of C on y, when given x and y. Formally, we should define the obfuscation of C to be a circuit that has $x = \mathcal{O}_k(C, r)$ hardwired in it, and makes an oracle query to $\mathcal{E}_k$; that is, on input y, the circuit issues the query (x, y) to the $\mathcal{E}_k$-part of F, and returns the answer received.

So it remains 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.[26]

CLAIM 4.14.1. *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 the choice of F, it holds that*

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

The rest of the proof is devoted to prove Claim 4.14.1. Fixing any PPT A, we define the simulator S such that, on input 1^k, it chooses $x \xleftarrow{R} \{0,1\}^{6k}$ and simulates $A^F(x)$. In this simulation, S uses its own F-oracle to answer A's oracle queries, *except that A's queries to $\mathcal{E}_{k'}$, for $k' \geq k$, are answered as follows*. The query (x', y') to $\mathcal{E}_{k'}$, where $k' \geq k$, is answered with z such that:

(1) If $x' = x$, then $z = C^F(y')$, where this value is computed using oracle access to C^F.

[26] 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.

(2) Else, if $x' = \mathcal{O}_{k'}(C', r')$ for some previous query (C', r') to the $\mathcal{O}_{k'}$-oracle, then $z = (C')^F(y')$, where $(C')^F(y')$ is computed recursively by emulating the computation of C' while using these very rules.
(3) Else, $z = \perp$.

Note that the queries generated in the handling of Case 2 have a total length that is smaller than the size of C'. It follows that the recursive evaluation of $(C')^F(y')$ only incurs a polynomial overhead in running time.[27] Also note that S never queries the $\mathcal{E}_{k'}$ 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 $x' \in \text{Image}(\mathcal{O}_{k'})$, $x' \neq x$, 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 $\text{poly}(k) \cdot 2^{2k}/2^{6k}$, taken over the choice of F and a random r (recall that $x = \mathcal{O}_k(C, r)$).[28] 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 4.14.1 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 $G(r) \stackrel{\text{def}}{=} \mathcal{O}_k(C, r)$ 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 $\mathcal{E}_{k'}$-oracle for any $k' \geq k$. This enables us to use the following claim, proven in Appendix B.

CLAIM 4.14.2. *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 Claim 4.14.2 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 $\mathcal{E}_{k'}$ 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, Claim 4.14.2 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. □

[27] The complexity of evaluating C', which is charge to the query (C', r'), is $\text{poly}(|C'|)$ plus the complexity of evaluating queries of total length smaller than $|C'|$. Thus, each level at the tree of recursion calls, which has depth at most $|C'|$, incurs a cost that is upper bounded by $\text{poly}(|C'|)$.

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

Bounded relativization.. While our main impossibility results do not relativize in the usual sense (see Proposition 4.14), their proofs can be modified to work for a slightly different form of relativization, which we refer to as *bounded relativization*.[29] In *bounded relativization*, an oracle is a *finite* function with fixed input length ℓ (which is polynomially related to the security parameter k), and all algorithms/circuits in the context being discussed can have running time larger than ℓ (but still polynomial in k). In particular, in the context of obfuscation, this means that the circuit to be obfuscated can have size larger than ℓ (but still polynomial in in it).

PROPOSITION 4.15. *Theorems 3.10 and 3.7* (one-way functions imply unobfuscatable circuits and impossibility of circuit obfuscators)*, and Corollary 3.9* (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. Suppose that the oracle F has input length ℓ and output length 1 (without loss of generality). We augment the Hom_K algorithm 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 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, $[C]$ denotes the function computed by a circuit C.

Rice's Theorem *Let $L \subseteq \{0,1\}^*$ be any language such that, for every two functionally equivalent machines M and M'* (i.e., $[M] \equiv [M']$)*, it holds that $M \in L$ if and only if $M' \in L$. Then, if L is decidable, then it is trivial* (i.e., 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. As Borchert and Stephan [BS] observe, simply replacing the word "Turing machine" with "circuit" and "decidable" with "polynomial time" does not work. One counterexample is the non-trivial language $L = \{C \in \{0,1\}^* : C(0) = 0\}$ that can be decided in polynomial time, although whenever $[C] \equiv [C']$ holds it holds that $C \in L$ iff $C' \in L$. Yet, there is a sense in which L is "somewhat trivial" in the sense that deciding whether or not $C \in L$ is feasible without using C itself; having oracle access to C suffices. This motivates the following conjecture:

CONJECTURE 5.1 (SCALED-DOWN RICE S THEOREM). *Let $L \subseteq \{0,1\}^*$ be any language such that, for every two functionally equivalent circuits C and C'* (i.e., $[C] \equiv [C']$)*, it holds that $C \in L$ if and only if $C' \in L$. If $L \in$ **BPP**, then there exists a PPT S such that*

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

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

Put differently, the conjecture states that any semantic property of circuits that is infeasible to decide when only given oracle access to the circuit is also infeasible to decide

[29] Bounded relativization reflects the way that the Random Oracle Model is sometimes interpreted in cryptography.

when given the circuit. We mention that Borchert and Stephan [BS] take a different approach to finding a complexity analogue of Rice's theorem: Instead of restricting the scope to properties that are infeasible to decide given oracle access, they restrict it to properties that only depend on the number of satisfying assignments of the circuit (and some variants of this idea). They show that any such property is **UP**-hard, and thus is unlikely to be tractable. Improved lower bounds are given in [HR; HT].

Not being able to settle Conjecture 5.1, we consider a generalization of it to *promise problems* [ESY], *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 that comes to mind is the following one, where we extend the definition of **BPP** to promise problems in the natural way.

CONJECTURE 5.2. *Let $\Pi = (\Pi_Y, \Pi_N)$ be any promise problem such that, for every two functionally equivalent circuits C and C'* (i.e., $[C] \equiv [C']$)*, it holds that $C \in \Pi_Y$ if and only if $C' \in \Pi_Y$, and similarly $C \in \Pi_N$ if and only if $C' \in \Pi_N$. If $\Pi \in$ **BPP**, then 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 circuits implies that the last 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 circuit ensemble given by Theorem 3.10, and let $\pi' : \bigcup_k \mathrm{Supp}(\mathcal{H}_k) \to \{0,1\}$ be the *first bit* of the function guaranteed by the unlearnability condition. Consider the following promise problem $\Pi = (\Pi_Y, \Pi_N)$:

$$\Pi_Y = \left\{ C' : \exists C \in \bigcup_k \mathrm{Supp}(\mathcal{H}_k) \text{ s.t. } [C] \equiv [C'] \text{ and } \pi'(C) = 1 \right\}$$
$$\Pi_N = \left\{ C' : \exists C \in \bigcup_k \mathrm{Supp}(\mathcal{H}_k) \text{ s.t. } [C] \equiv [C'] \text{ and } \pi'(C) = 0 \right\}$$

By the reverse-engineering condition, C can be recovered from the code of any functionally equivalent circuit C', and π is easy to evaluate. Thus, $\Pi \in$ **BPP**. But since $\pi(C)$ is pseudorandom with black-box access to C, no S satisfying Conjecture 5.2 can exist. □

Discussion.. It is an interesting problem to weaken or even remove (from Theorem 5.3) the hypothesis that one-way functions exist. Reasons to believe that this may be possible are: (1) The fact that conjectures refers only to worst-case complexity (and not average case), and (2) the fact that the conjectures imply some sort of computational difficulty. For instance, if **NP** $\subseteq$ **BPP**, then both conjectures are false, since CIRCUIT SATISFIABILITY is not decidable using black-box access (e.g., using black-box access, one cannot distinguish a circuit that is satisfied on exactly one randomly chosen input from an unsatisfiable circuit). Thus, if we could weaken the hypothesis of Theorem 5.3 to **NP** $\not\subseteq$ **BPP**, then 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 somewhat 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. We mention that the generalization that succeeds (i.e., is valid) is closest in spirit to Conjecture 5.2, indicating that the failure of the latter is more surprising.

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. One such case is the case of *sampling algorithms*, considered in this section. That is, we consider probabilistic algorithms that when fed a uniformly random string (of some length) as input, produce an output according to some desired distribution. We stress that we are interested in the distribution that is defined by such algorithms, and not in the strict effect of these algorithms as functions (from random inputs to samples in the distribution). (At the end of the section, we also discuss definitions of obfuscation for general probabilistic algorithms.)

For simplicity, we only work with sampling algorithms described 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, upon 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 (for a class of circuits)* if the following three conditions hold:

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

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

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

Note that Definition 6.1 differs from Definition 2.2 firstly in the functionality condition and secondly in the type of oracle considered in the virtual black-box condition.

We do not know whether Definition 6.1 can be met, but we can rule out the following (seemingly) stronger definition, which essentially allows the adversary to output arbitrarily long strings, instead of just one bit as in the definition above.

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

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

Note that this definition is analogous to the strongest definition considered at the beginning of Section 2.2. However, at the current context, this type of definition seems harder to rule out.

PROPOSITION 6.3. *If one-way functions exist, then strong sampling obfuscators do not exist. In particular, there exist no efficient strong sampling obfuscators, unconditionally.*

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 strong sampling obfuscator $\mathcal{O}$, and consider the PPT adversary A defined by $A(C) = C$. By the definition of a strong sampling obfuscator, there exists a PPT simulator S that, 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 that is indistinguishable from one that generates random messages with their 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 that 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.

The claim regarding the nonexistence of efficient (strong sampling) obfuscators follows by showing that their existence implies the existence of one-way functions. A stronger claim, which refers to (efficient) non-strong sampling obfuscators, is proved in the following proof of Proposition 6.4. □

Turning back to the sampling obfuscators in the sense of Definition 6.1, we can show that they imply the nontriviality of **SZK** (i.e., the class of promise problems possessing statistical zero-knowledge proofs).

PROPOSITION 6.4. *If efficient sampling obfuscators exist, then* **SZK** *is not contained in* **BPP**.

PROOF. It is known that the following promise problem $\Pi = (\Pi_Y, \Pi_N)$ is in **SZK** [SV] (and in fact has a noninteractive perfect zero-knowledge proof system [DDPY; GSV]):

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

where n denotes the output length of the circuit C and U_n is the uniform distribution on $\{0,1\}^n$. Assuming, towards the contradiction, that efficient sampling obfuscators exist, we will show that Π is not in **BPP**.

Suppose that an efficient sampling obfuscator $\mathcal{O}$ exists. Analogous to Lemma 3.8, such obfuscators imply the existence of one-way functions (we can obtain a one-way function f by defining $f(\alpha, r) = \mathcal{O}(C_\alpha, r)$ where C_α is a circuit that outputs α with probability $1/2^n$ and 0 otherwise). Thus, there also exists a length-doubling pseudorandom generator G [HILL]. Let $G_n : \{0,1\}^{n/2} \to \{0,1\}^n$ denote the circuit that evaluates G on inputs of length $n/2$.

Now consider any PPT algorithm A. By the definition of a sampling obfuscator, there exists a PPT machine S such that $Pr[A(\mathcal{O}(G_n)) = 1]$ is negligibly close to $\Pr[S^{\langle\langle G_n\rangle\rangle}(1^{|G_n|}) = 1]$, and $Pr[A(\mathcal{O}(U_n)) = 1]$ is negligibly close to $\Pr[S^{\langle\langle U_n\rangle\rangle}(1^{|G_n|}) = 1]$, where here U_n means the trivial padded circuit that samples uniformly from U_n but has the same size as G_n. But by the definition of pseudorandom generators and a hybrid argument (over each sampling access), it follows that $\Pr[S^{\langle\langle G_n\rangle\rangle}(1^{|G_n|}) = 1]$ and $\Pr[S^{\langle\langle U_n\rangle\rangle}(1^{|G_n|}) = 1]$ are also negligibly close. Thus, for any PPT algorithm A, it holds that $|Pr[A(\mathcal{O}(G_n)) = 1] - Pr[A(\mathcal{O}(U_n)) = 1]|$ is negligible (and, in particular, smaller than $1/3$).

However, 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 $\Pi \notin$ **BPP**. ■

Remark 6.5. Assuming the existence of one-way functions, we can extend the result of Proposition 6.4 to the natural notion of *approximate sampling obfuscators*, in which $\mathcal{O}(C)$ only needs to sample a distribution of small statistical difference from that of C. This is done by using STATISTICAL DIFFERENCE, the complete problem for **SZK** from [SV], in place of the promise problem Π. (We assume the existence of one-way functions, because we do not know whether approximate sampling obfuscators imply their existence.)

Definition of obfuscation for general probabilistic algorithms.. We note that combining Definitions 2.2 and 6.1 yields a natural definition of obfuscator for general probabilistic algorithms in the form of randomized circuits C that take an input x, and produce an output according to some desired distribution that depends on x. (Consider, for example, your favorite randomized primality tester, or a probabilistic encryption scheme.) For the functionality requirement, we can require that $\mathcal{O}(C)$ outputs a randomized circuit C' such that, for every input x, it holds that $C'(x)$ is identically distributed (or statistically close) to $C(x)$. For the virtual black-box property, we can give the simulator S access to a probabilistic oracle, that on every query x, gives an answer distributed according to $C(x)$. Of course, any obfuscator meeting this definition also meets Definition 2.2 (as a special case), and hence the negative result of Theorem 3.7 applies. The point of the current section was to study restricted types of obfuscators (i.e., ones that are supposed to be applied only to randomized algorithm that have no "real" inputs).

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 results).

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 and C_2 that compute the same function and are of the same size k, it holds that

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

It is instructive to contrast the notion of unobfuscatable circuit ensemble (i.e., Definition 3.1) with the notion of indistinguishability obfuscator. The unobfuscatability of the former (as well as the non-existence of general circuit obfuscators) is not due to the method used for converting the original circuit into a possibly obfuscated one, but is rather due to the fact that the *code* of *any* functionally equivalent circuit allows for reverse-engineering of the original circuit! In contrast, Definition 7.1 does not consider something that can be extracted from the code of *any* functionally equivalent circuit (of a certain size) as breach of the obfuscation condition. Only distinguishing between (the obfuscated forms of) functionally equivalent circuits (of a certain size) is considered a breach of the obfuscation condition. This contrast is demonstrated in 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, we will show that no $\mathcal{O}'$ (even an inefficient one) makes C_1 and C_2 much more indistinguishable than $\mathcal{O}$ does. Intuitively, this will say that an indistinguishability obfuscator is "as good" as any other obfuscator that exists. For example, it will imply that if "differing-input obfuscators" (as we will define later) exist, then any indistinguishability obfuscator is essentially also a differing-input obfuscator.

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 ("COMPETITIVENESS"). *Suppose $\mathcal{O}$ is an efficient indistinguishability obfuscator. Let $\mathcal{O}'$ be any* (possibly inefficient) *algorithm that satisfies the syntactic requirements of obfuscation as well as the condition that $|\mathcal{O}'(C)| = q(|C|)$ for some fixed polynomial q. Then, for every PPT A, there exists a PPT A' and a negligible function α such that for all circuits C_1, C_2 of size k, it holds that*

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

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

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

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

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

The proposition follows. ■

Even with this 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 that compute the corresponding functions.

Definition 7.4 (*differing-inputs obfuscator*). A *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

$$\varepsilon \stackrel{\text{def}}{=} \left|\Pr\left[A(\mathcal{O}(C_1)) = 1\right] - \Pr\left[A(\mathcal{O}(C_2)) = 1\right]\right| > \alpha(k).$$

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 $\mathrm{poly}(k, 1/(\varepsilon - \alpha(k)))$.

Indeed, this definition implies 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+; PAK].) Here we concentrate on watermarking *programs*, as in [CT; NSS]. A watermarking scheme should consist of a *marking* algorithm that embeds a message m into a given program, and an *extraction* algorithm that 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 various heuristic methods for software watermarking in the literature (*cf.,* [CT]), but as with obfuscation, there has been little theoretical work on this problem in general. 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 arises, of course, from the need to properly capture the fragility property. Note that it is easy to remove a watermark from programs for functions that are exactly learnable with 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 $(\mathrm{Mark}, \mathrm{Extract})$ satisfying the following properties:

— (functionality) For every circuit C, key K, and message m, the string $\mathrm{Mark}_K(C, m)$ describes a circuit that computes the same function as C.
— (polynomial slowdown) There is a polynomial p such that for every circuit C, it holds that $|\mathrm{Mark}_K(C, m)| \leq p(|C| + |m| + |K|)$.
— (extraction) For every circuit C, key K, and message m, it holds that $\mathrm{Extract}_K(\mathrm{Mark}_K(C, m)) = m$.
— (meaningfulness) For every circuit C, it holds that $\Pr_K[\mathrm{Extract}_K(C) \neq \perp] = \mathrm{neg}(|C|)$.
— (fragility) For every PPT A, there is a PPT S such that for every C and m, it holds that

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

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 $\mathrm{Extract}$ run in polynomial time.

Actually, a stronger fragility property than the foregoing one is probably desirable; the foregoing definition does not exclude the possibility that the adversary can remove the watermark by changing the value the function at a single location. However, as shown next, even the former minimal 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 unobfuscatable circuit ensemble guaranteed by Theorem 3.10. No matter how we try to produce a marked circuit from $C \stackrel{R}{\leftarrow} \mathcal{H}$, the algorithm guaranteed by the reverse-engineerability condition in Definition 3.1 can reconstruct the source code C, which (by the meaningfulness property) is unmarked with high probability. On the other hand, any potential simulator, given just oracle access to C, will be unable to produce any circuit computing the same function (since if it could, then it could compute $\pi(C)$, which is pseudorandom). □

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

Proof Sketch: As usual, it suffices to show that the existence of an efficient watermarking scheme implies the existence of one-way function. Consider, for example, the mapping of $K, \alpha \in \{0,1\}^n$ to $\mathrm{Mark}_K(C_\alpha, 0^n)$, where $C_\alpha(x) = 1$ iff $x = \alpha$. Observe that no simulator S, which is given oracle access to C_α, can find a circuit that is computationally equivalent to C_α, but inverting this map enables the recovery of α, which in turn allows to construct an unmarked circuit that is computationally equivalent to C_α. □

Given these impossibility results, we are led to seek the weakest possible formulation of the fragility condition, requiring that any adversary *occasionally* fails to remove the mark (i.e., there exists a circuit and a message such that the adversary fails to remove the corresponding marking with noticeable probability).

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

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

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

Interestingly, in contrast to the connection suggested by the proof of Theorem 8.2, 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, here we require the watermarking scheme to satisfy the additional natural condition that* $|\mathrm{Mark}_K(C,m)| = q(|C|)$ *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, it holds that $\mathcal{O}(\mathrm{Mark}_K(C, 1^{|C|}))$ is a circuit computing the same function as C. Let C' be a padding of C to the same length as $\mathrm{Mark}_K(C, 1^{|C|})$. By fragility, $\mathrm{Extract}_K(\mathcal{O}(\mathrm{Mark}_K(C,1))) = 1$ with nonnegligible probability. By meaningfulness, $\mathrm{Extract}_K(\mathcal{O}(C')) = 1$ with negligible probability. Thus, $\mathrm{Extract}_K$ distinguishes between $\mathcal{O}(C')$ and $\mathcal{O}(\mathrm{Mark}_K(C, 1^{|C|}))$, contradicting the indistinguishability property of $\mathcal{O}$. ■

Note that this proposition fails if we allow $\mathrm{Mark}_K(C,m)$ to instead be an *approximate implementation* of C in the sense of Definition 4.1. Indeed, in such a case it seems that obfuscators would be useful in constructing watermarking schemes, because a 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 (possible strengthening of) the fragility condition, because it is desirable to prevent adversaries from producing unmarked approximate implementations of the marked program.

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

9. RESEARCH DIRECTIONS AND SUBSEQUENT 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 4.1, we have ruled out a obfuscators with approximately preserve functionality in a stronger sense; see discussion after Theorem 4.3.)
— Obfuscators for a restricted, yet still nontrivial, class of functions. By Theorem 4.12, 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 mention that the constructions of [Can; CMR] can be viewed as some form of obfuscators for "point functions" (i.e., functions $f : \{0,1\}^n \to \{0,1\}$ that 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, unlike Definition 8.1, is not ruled out by our constructions) seems to be challenging, whereas for homomorphic encryption, the definitions are (more) straightforward, but the question of existence seems very challenging.

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

Subsequent Work.. Subsequent to the original version of this paper [BGI$^+$], a number of other works have continued to develop our theoretical understanding of the possibility and limitations of obfuscation. The paper [GK] provides negative results for obfuscating natural and cryptographically useful functionalities (as opposed to our contrived functionalities), with respect to a stronger definition of security. The papers [LPS; Wee; DS; NS1; NS2] explore the possibility of obfuscating simple but useful functionalities such as "point functions" and generalizations, a line of work begun in the work of [Can; CMR], which preceded our work. The papers [HMS; HRSV] propose definitions of obfuscation that are suitable for cryptographic applications (strengthening Definition 2.2 in some respects and weakening it in others), and [HRSV] shows how to obfuscate a specific "re-encryption" functionality with respect to one of these definitions. The paper [GR] proposes and explores a definition of obfuscation that does not fall within the scope of our impossibility result (and is closely related to our notion of indistinguishability obfuscators from Definition 7.1).

A. GENERALIZING RICE'S THEOREM TO PROMISE PROBLEMS

We say that an algorithm A *decides* the promise problem $\Pi = (\Pi_Y, \Pi_N)$ if

$$x \in \Pi_Y \;\Rightarrow\; A(x) = 1$$
$$x \in \Pi_N \;\Rightarrow\; A(x) = 0$$

In such a case, we say that Π is *decidable*. We say that Π is closed under $[\cdot]$ if, for all pairs (M, M') such that $[M] \equiv [M']$, it holds that $M \in \Pi_Y$ iff $M' \in \Pi_Y$, and similarly $M \in \Pi_N$ iff $M' \in \Pi_N$. 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 any 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)

$$\Pi_Y = \{M : \ M \text{ always halts and } M(0) = 1\}$$
$$\Pi_N = \{M : \ M \text{ always halts and } M(0) = 0\}$$

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 saying that "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 any promise problem closed under $[\cdot]$. If Π is decidable, then there exists a Turing machine S such that*

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

Note the similarity to Conjecture 5.2.

PROOF. Suppose that $\Pi = (\Pi_Y, \Pi_N)$ is decided by the Turing machine T. We will build a machine S that 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 that 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. In other words, it is natural to consider also the following way to generalize Rice's Theorem:

CONJECTURE A.3 (RICE'S THEOREM — THIRD GENERALIZATION). *Let $\Pi = (\Pi_Y, \Pi_N)$ be any promise problem closed under $[\cdot]$. If Π is decidable, then 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 $\Pi = (\Pi_Y, \Pi_N)$:

$$\Pi_Y = \{M : M \text{ always halts and } \exists x < \mathrm{KC}([M]) \text{ s.t. } [M](x) = 1\}$$
$$\Pi_N = \{M : M \text{ always halts and } \forall x\ M(x) = 0\}$$

where $\mathrm{KC}(f)$ denotes the description length of the smallest Turing machine that computes the partial recursive function 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 that 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 $N_{n,r}$ that 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 B.1 (CLAIM 4.14.2, RESTATED). *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,*

$$\left| \Pr_{x \in [K]} \left[D^G(G(x)) = 1 \right] - \Pr_{y \in [L]} \left[D^G(y) = 1 \right] \right| \leq \frac{1}{K^\delta}. \tag{7}$$

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 [GT]. Specifically, we will show that "most" G for which Inequality (7) 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 (7) 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 $x \in S_G$, none of the oracle queries made in the execution of $D^G(G(x))$ are at points in S_G, 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.1.1. *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 $G \stackrel{R}{\leftarrow} \mathcal{G}$ 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.1.2. *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.1.1, 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.1.1, 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

$$\Pr_{x\in S_G}\left[M(G(x))=1\right]-\Pr_{y\in L_G}\left[M(y)=1\right]>\frac{1}{2K^\delta} \tag{8}$$

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 (8), which can be rewritten as:

$$\Pr_{y\in T}\left[M(y)=1\right]-\Pr_{y\in L_G}\left[M(y)=1\right]>\frac{1}{2K^\delta} \tag{9}$$

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 9.

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.1.2, $\mathcal{G}'$ 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.

C. OBFUSCATION AND THE FIAT–SHAMIR TRANSFORMATION

In this section, we briefly revisit the question considered in Proposition 4.11: Namely, whether the random oracle used for the Fiat–Shamir transformation can be "instantiated" by obfuscating a function chosen randomly from a pseudo-random function ensemble. In Proposition 4.11, we showed that for any (3-round) public-coin honest-verifier zero-knowledge identification protocol, there exists a contrived pseudorandom function ensemble such that if the random oracle is replaced with any public circuit that computes any function in that ensemble, then the signature scheme obtained by applying the Fiat–Shamir transformation is insecure. In fact, we showed that the resulting signature scheme is insecure in the strong sense that an adversary can forge a signature given only the public record of the signer.

Here, we show that a similar but weaker attack applies when applying the Fiat–Shamir transformation to *any* public-coin identification scheme. The attack is weaker in that the adversary must first obtain a valid signature on a fixed message, and only then it is able to produce a forged signature on another message.[30] At a high level, the attack proceeds similarly to our previous attack but instead of "implanting" a hidden simulation into the pseudorandom function ensemble, it implants a known collision into the pseudorandom function ensemble.

PROPOSITION C.1. *Let (P,V) be an arbitrary 3-round public-coin identification protocol, and consider the signature scheme in which message m is signed by sending the transcript that corresponds to an interaction of $(P,V)(\rho)$, where ρ is the signer's public record/key and the verifier message in the interaction is replaced by the application of the Random Oracle to the pair (m,a) such that a is the first message sent by P. Suppose that one-way functions exists.Then, there exists a pseudorandom function ensemble $\{h_K\}_{K\in\{0,1\}^*}$ such that replacing the random oracle in the foregoing scheme by* any *public circuit that computes any h_K, yields an insecure scheme, in the sense that an attacker can forge a valid signature given only the signer's public record/key and a single signature on a fixed message.*

Proof Sketch: Starting with an arbitrary pseudorandom function ensemble, denoted $\{f_s\}$, we consider the function ensemble $\{f'_{s,m_0}\}$ defined (for polynomially related $|s|$ and $|m_0|$) by

$$f'_{s,m_0}(x \circ y, z) \stackrel{\text{def}}{=} \begin{cases} f_s(0^{|m_0|} \circ y, z) & \text{if } x = m_0 \\ f_s(x \circ y, z) & \text{otherwise} \end{cases}$$

[30] While this attack is weaker than the one obtained in Proposition 4.11, we note that nevertheless the current attack shows that the resulting signature scheme is not even a one-time signature scheme.

Note that the resulting ensemble preserves the pseudorandomness of the original one, since the modified inputs are extremely rare (and the adversary lacks any information regarding their identity). On the other hand, given the seed of a function (i.e., s, m_0), note that any valid signature (a, b, c) on the message $0^{|m_0|}$ is itself a forged signature for the message $m_0 \circ \rho$ since $b = f'_{s,m_0}(0^{|m_0|} \circ \rho, a) = f'_{s,m_0}(m_0 \circ \rho, a)$ by construction. The remainder of the proof is identical to the proof of Proposition 4.11. □

Remark C.2. We first note that the foregoing argument extends to any multi-round public-coin identification protocols. Secondly we note that, while (in contrast to Proposition 4.11) the foregoing attack applies to any public-coin identification scheme, this wider applicability comes with some inherent limitations. To see this, let (G, S, V) be a secure signature scheme and consider the following simple public-coin identification protocol. The signer uses G to generate a signing key SK and a verification key VK. The public record of the identification record is $\rho = VK$. The identification protocol starts by having the signer send VK to the verifier. The verifier responds with a long random string r. The final message of the protocol is a signature on r using the signing key SK. Clearly, this protocol is a secure identification protocol. We make two observations:

(1) The foregoing identification protocol is *not* zero-knowledge in a strong sense: Given only the public record VK, it is not possible to generate an accepting transcript of the protocol (as this would contradict the security of the signature scheme). This shows that if one applied the Fiat–Shamir transformation to this protocol with *any* function replacing the random oracle, the resulting signature scheme would not allow for forgeries given only the public record. This is in strong contrast to the situation in Proposition 4.11, where we show that forgery is possible given only the public record.
(2) On the other hand, the foregoing identification protocol (unsurprisingly) yields a completely secure signature scheme when the Fiat–Shamir transformation is applied with the random oracle replaced with a randomly chosen hash function from any collision-resistant hash function ensemble. As such, the foregoing attack can also be seen as ruling out the generic use of obfuscation to transform a pseudorandom function ensemble into a collision-resistant hash function ensemble. Again, however, this is in contrast to the case of applying Fiat–Shamir to general public-coin honest-verifier zero-knowledge protocols, where no general secure method of instantiating the random oracle is known (see [DNRS] for further discussion).

ACKNOWLEDGMENT

We are grateful to Luca Trevisan for collaboration at an early stage of this research. We also thank Dan Boneh, Ran Canetti, Manoj Prabhakaran, Michael Rabin, Emanuele Viola, Yacov Yacobi, and the anonymous reviewers of *CRYPTO'01* and *JACM* for helpful discussions and comments.

Most of this work was done when Boaz Barak was a graduate student in Weizmann Institute of Science, Amit Sahai was a graduate student at MIT (supported by an DOD/NDSEG Graduate Fellowship), Salil Vadhan was a graduate student and a postdoctoral fellow at MIT (supported by a DOD/NDSEG Graduate Fellowship and an NSF Mathematical Sciences Postdoctoral Research Fellowship), and Ke Yang was a graduate student at CMU. Further support for this work was provided to Boaz Barak by NSF grants 0627526 and 0426582, US-Israel BSF grant 2004288 and Packard and Sloan fellowships, to Oded Goldreich by the Minerva Foundation (Germany) and the Israel Science Foundation (grant No. 460/05), to Amit Sahai by a Sloan Research Fellowship, an Okawa Research Award, and NSF grants 0205594, 0312809, 0456717, 0627781, 0716389, 0830803, 0916574, 1065276, 1118096, and 1136174, and to Salil Vadhan by NSF grants 0430336 and 0831289, a Guggenheim Fellowship, and the Miller Institute for Basic Research in Science.

REFERENCES

B. Barak. How to go beyond the black-box simulation barrier. In *42nd IEEE Symposium on Foundations of Computer Science (Las Vegas, NV, 2001)*, pages 106–115. IEEE Computer Soc., Los Alamitos, CA, 2001.

B. Barak. Can We Obfuscate Programs? Essay, 2002. `http://www.cs.princeton.edu/~boaz/Papers/obf_informal.html`.

B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (Im)possibility of Obfuscating Programs. In J. Kilian, editor, *Advances in Cryptology—CRYPTO '01*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18. Springer-Verlag, 19–23 August 2001. Preliminary full versions appeared in the ECCC and eprint archives.

M. Bellare and P. 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.

D. Boneh and R. 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, Aug. 1996.

B. Borchert and F. Stephan. Looking for an analogue of Rice's theorem in circuit complexity theory. *Mathematical Logic Quarterly*, 46(4):489–504, 2000.

R. Canetti. Towards Realizing Random Oracles: Hash Functions That Hide All Partial Information. In B. S. K. Jr., editor, *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 455–469. Springer, 1997.

R. Canetti, O. Goldreich, and S. 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.

R. Canetti, D. Micciancio, and O. 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.

C. Collberg and C. Thomborson. Watermarking, Tamper-Proofing, and Obfuscation – Tools for Software Protection. Technical Report TR00-03, The Department of Computer Science, University of Arizona, Feb. 2000.

A. De Santis, G. Di Crescenzo, G. Persiano, and M. 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.

W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

Y. Dodis and A. Smith. Correcting errors without leaking partial information. In H. N. Gabow and R. Fagin, editors, *STOC*, pages 654–663. ACM, 2005.

D. Dolev, C. Dwork, and M. Naor. Nonmalleable cryptography. *SIAM Journal on Computing*, 30(2):391–437 (electronic), 2000.

C. Dwork, M. Naor, O. Reingold, L. J. Stockmeyer. Magic Functions. *J. ACM* 50(6): 852-921 (2003)

S. Even, A. L. Selman, and Y. Yacobi. The complexity of promise problems with applications to public-key cryptography. *Information and Control*, 61(2):159–173, 1984.

J. Feigenbaum and M. Merritt, editors. *Distributed computing and cryptography*, Providence, RI, 1991. American Mathematical Society.

A. Fiat and A. 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.

H. N. Gabow and R. Fagin, editors. *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*. ACM, 2005.

R. Gennaro and L. Trevisan. Lower Bounds on the Efficiency of Generic Cryptographic Constructions. In *41st Annual Symposium on Foundations of Computer Science*, Redondo Beach, CA, 17–19 Oct. 2000. IEEE.

O. Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, Cambridge, 2001.

O. Goldreich. *Foundations of Cryptography: Basic Applications*. Cambridge University Press, Cambridge, 2004.

O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the Association for Computing Machinery*, 33(4):792–807, 1986.

O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM*, 43(3):431–473, 1996.

O. Goldreich, A. Sahai, and S. 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 Aug. 1999. To appear.

S. Goldwasser and Y. T. Kalai. On the Impossibility of Obfuscation with Auxiliary Input. In *FOCS*, pages 553–562. IEEE Computer Society, 2005.

S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and System Sciences*, 28(2):270–299, Apr. 1984.

S. Goldwasser and G. N. Rothblum. On Best-Possible Obfuscation. In *TCC*, pages 194–213, 2007.

S. 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.

J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396 (electronic), 1999.

L. A. Hemaspaandra and J. Rothe. A second step towards complexity-theoretic analogs of Rice's Theorem. *Theoretical Compututer Science*, 244(1–2):205–217, 2000.

L. A. Hemaspaandra and M. Thakur. Lower bounds and the hardness of counting properties. *Theoretical Computer Science*, 326(1-3):1–28, 2004.

D. Hofheinz, J. Malone-Lee, and M. Stam. Obfuscation for Cryptographic Purposes. In *TCC*, pages 214–232, 2007.

S. Hohenberger, G. N. Rothblum, A. Shelat, and V. Vaikuntanathan. Securely Obfuscating Re-encryption. In *TCC*, pages 233–252, 2007.

R. Impagliazzo and M. 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 Oct.–1 Nov. 1989. IEEE.

J. Katz and M. 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.

M. J. Kearns and U. V. Vazirani. *An introduction to computational learning theory*. MIT Press, Cambridge, MA, 1994.

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

B. Lynn, M. Prabhakaran, and A. Sahai. Positive Results and Techniques for Obfuscation. In C. Cachin and J. Camenisch, editors, *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, pages 20–39. Springer, 2004.

L. R. Matheson, S. G. Mitchell, T. G. Shamoon, R. E. Tarjan, and F. 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, Feb. 1998.

D. Naccache, A. Shamir, and J. 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, Mar. 1999.

M. Naor and O. 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 Oct. 1997. IEEE.

A. Narayanan and V. Shmatikov. Obfuscated databases and group privacy. In V. Atluri, C. Meadows, and A. Juels, editors, *ACM Conference on Computer and Communications Security*, pages 102–111. ACM, 2005.

A. Narayanan and V. Shmatikov. On the Limits of Point Function Obfuscation. Cryptology ePrint Archive, Report 2006/182, 2006. `http://eprint.iacr.org/`.

F. A. P. Petitcolas, R. J. Anderson, and M. J. Kuhn. Information Hiding — A Survey. *Proceedings of the IEEE*, 87(7):1062–1078, 1999.

R. L. Rivest, L. Adleman, and M. 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.

A. Sahai and S. Vadhan. A complete problem for statistical zero knowledge. *Journal of the ACM*, 50(2):196–249, March 2003.

T. Sander, A. Young, and M. Yung. Non-interactive Cryptocomputing for NC^1. In *40th Annual Symposium on Foundations of Computer Science*, pages 554–566, New York, NY, 17–19 Oct. 1999. IEEE.

M. Sipser. *Introduction to the Theory of Computation*. Course Technology, 2nd edition, 2005.

F. van Dorsselaer. Obsolescent Feature. Winning entry for the *1998 International Obfuscated C Code Contest*, 1998. `http://www.ioccc.org/`.

H. Wee. On obfuscating point functions. In H. N. Gabow and R. Fagin, editors, *STOC*, pages 523–532. ACM, 2005.