On the Limits of Point Function Obfuscation

Arvind Narayanan and Vitaly Shmatikov
The University of Texas at Austin
{arvindn,shmat}@cs.utexas.edu

May 30, 2006

Abstract

We study the problem of circuit obfuscation, *i.e.*, transforming the circuit in a way that hides everything except its input-output behavior. Barak *et al.* showed that a universal obfuscator that obfuscates every circuit class cannot exist, leaving open the possibility of special-purpose obfuscators. Known positive results for obfuscation are limited to point functions (boolean functions that return 1 on exactly one input) and simple extensions thereof in the random oracle model, *i.e.*, assuming black-box access to a true random function. It was also shown by Wee how to instantiate random oracles so as to achieve a slightly weaker form of point function obfuscation. Two natural questions arise: (i) what circuits have obfuscators whose security can be reduced in a black-box way to point function obfuscation? and (ii) for what circuits obfuscatable in the random oracle model can we instantiate the random oracles to build obfuscators in the plain model?

We give partial answers to these questions: there is a circuit in AC^0 which can be obfuscated in the random oracle model, but not secure when random oracles are instantiated with Wee's construction. More generally, we present evidence for the impossibility of a black-box reduction of the obfuscatability of this circuit to point function obfuscation. Conversely, this result shows that to instantiate random oracles in general obfuscators, one needs to utilize properties of the instantiation that are not satisfied by point function obfuscators.

1 Introduction

Obfuscation is a long-standing problem in computer security and cryptography. To obfuscate a function f is to create an implementation of f that reveals nothing about f except its input-output behavior. Intuitively, a circuit *obfuscator* $\mathcal{O}$ is an efficient algorithm that, given a circuit C implementing some function f, outputs another circuit $\mathcal{O}(C)$ such that (i) its size is within a polynomial factor of c, (ii) it computes (perhaps approximately) the same function as f, and, (iii) for any efficient adversary that computes some predicate on $\mathcal{O}(C)$, there exists an efficient simulator that computes the same predicate with black-box access to an oracle that evaluates f. The latter requirement was formalized by Canetti [Can97] (for point functions) and Barak *et al.* [BGI$^+$01] (for any circuit) as the "virtual black-box" property of secure obfuscation.

Barak *et al.* showed that a universal obfuscator that obfuscates *every* circuit cannot exist. This does not rule out the existence of special-purpose obfuscators for many useful circuit classes. In particular, obfuscation of point functions has received some attention. A *point function* (also known as a delta function) $f_x : D \to \{0,1\}$ returns 1 on some input $x \in D$, and 0 on every other input. For example, the password checking routine in Unix can be thought of as a point function that returns 1 on input the correct password, and 0 otherwise. Although it is not known whether point functions are obfuscatable in the virtual black-box model of [BGI$^+$01] under general complexity assumptions, there exist constructions for point function obfuscators under restrictions on the adversary [CMR98, DS05] (i.e, the input distribution must have high min entropy), or under a slightly weakened definition of obfuscation [Wee05].

Obfuscated point functions are closely related to *random oracles* [BR93]. Random oracles trivially obfuscate point functions because the output of the random oracle hides all information about the input that produced it. This was observed in the early work on perfectly one-way probabilistic hash functions [Can97, CMR98] (which are essentially point function obfuscators) that aimed to provide an implementation of random oracles. Also, the only known positive results for obfuscation in the random oracle model are simple compositions of obfuscated point functions, implemented as (programmable) random oracles [LPS04, NS05].

Separation between random oracles and point function obfuscators. We investigate the power of random oracles in enabling obfuscation. While the random oracle heuristic has caused some controversy due to its generic insecurity in many cryptographic constructions [CGH04a, CGH04b, GK03, BBP04], it appears that random oracles are potentially useful for practical obfuscation and, most importantly, they *can* be securely instantiated in some obfuscation contexts. Wee demonstrated how to implement random oracles for the specific purpose of point function obfuscation [Wee05]. It is tempting to do the same for other obfuscations in the random oracle model. There is no known general method to prove all such constructions secure, because it is impossible to simulate the view of the adversary [Wee05, theorem 5.3]. Nonetheless, instantiating random oracles with point function obfuscators appears more promising than instantiating them with hash functions such as SHA-1, which are not even randomized. Furthermore, one might hope to prove such constructions secure for most concrete circuits that one is interested in.

We show that this is likely to be impossible. We present evidence that it is impossible to take an obfuscation that is provably secure in the random oracle model, instantiate the random oracles with point function obfuscators, and prove the obfuscation secure by black-box reduction to the security of point function obfuscation. More generally, there are circuits obfuscatable in the random oracle model whose obfuscatability in the plain model cannot be established by black-box reduction to the security of point function obfuscators (regardless of whether the latter obfsucator works by instantiating the random oracles in the output of the former obfuscator.)

We start by defining "point oracles" to be oracles computing point functions. To isolate the power of oracles (either point function oracles, or random oracles, depending on the context) from computational feasibility, we reformulate the virtual black box property in terms of *statistical security*. In this model, the adversary (and the adversary simulator) are computationally unbounded, but are limited to a polynomial number of oracle queries. (All known positive results for obfuscation in the random oracle model in fact achieve the stronger notion of statistical security.)

Techniques. Our key technical tool is the *lensing lemma*. Intuitively, if the obfuscated circuit leaks a small amount of information (which may not by itself be sufficient to break obfuscation if the input distribution has a high enough min-entropy), then an adversary with a superlogarithmic advice (or "lens") has a non-negligible advantage over the simulator who has the same advice. This technique is reminiscent of the proof of the Valiant-Vazirani theorem [VV85]. The other main new technique is the *probable queries lemma*, which, very roughly, demonstrates that only a polynomial number of oracle inputs "matter" in the evaluation of the obfuscated circuit.

Some of our results also apply to the plain (computationally bounded) model where all algorithms are required to be efficient, *i.e.*, probabilistic polynomial-time. Specifically, we use the lensing lemma to demonstrate that Wee's implementation of random oracles [Wee05], which is secure when used to instantiate random oracles in the point function obfuscation, results in an *insecure* obfuscation under $\omega(\log n)$-way self composition (i.e, when the adversary is allowed to see $\omega(\log n)$ obfuscations of the same input), n being the input size. We expect that the lensing lemma will find applications beyond those considered in this paper.

Interpretation. Analogous to the proof of Impagliazzo and Rudich [IR89] that there is no black box reduction from secret-key agreement to one-way functions, our proof technique can be modified to show that there is no black box reduction from obfuscation of some circuit to point function obfuscation. This can be done by weakening the adversary to be computationally bounded, but assuming that $\mathsf{P} = \mathsf{NP}$. This would show that a black box reduction would be as hard as proving that $\mathsf{P} \neq \mathsf{NP}$.

The separation between random oracles and point oracles shows us that there are obfuscators in the random oracle model in which we cannot hope to securely instantiate the random oracles with circuits that do no more than obfuscate point functions. It is not clear what extra properties of an instantiation would be useful for proving the security of specific obfuscators. It would be interesting to investigate whether the circuit that we prove in Section 5 to be (i) obfuscatable in the random oracle model, and (ii) has no point oracle obfuscator with statistical security, is in fact unobfuscatable in the plain model. Such a result would be stronger than the Barak et al. impossiblity result because our circuit is in AC^0.

However, it appears more likely that there is in fact an instantiation for random oracles that makes the above circuit obfuscatable in the plain model. Such a result would be the first positive result for obfuscation that does not follow from the obfuscatability of point functions. This dual promise makes the circuit of Section 5 an exciting avenue for future research.

2 Preliminaries

Throughout the paper we will use the letter ψ to denote an oracle.

Definition 1 (Point oracle) *A point oracle ψ behaves as follows: on a query x, if it has seen the query x before, ψ outputs 1, else 0.*

Compare this definition with a random oracle, which generates and returns a uniformly random string if it has not seen x before, and the same string when it sees x again. An alternative way to define a point oracle would be to give it explicit get and set interfaces. Our formulation is equivalent, and makes it easy to write oracle algorithms in exactly the same way regardless of whether they are querying random oracles or point oracles.

Analogous to the random oracle model, the way in which we make use of point oracles is to assume that all parties have access to such an oracle. The intended use of a point oracle is for the obfuscator to query and thus "program" it on a single point, so that to the adversary it behaves like an oracle computing a point function. It is perfectly legal and sometimes meaningful for the obfuscator to query it on multiple points.

Regardless of the analogy above, there is a fundamental difference between a random oracle and a point oracle: we are using the latter to model an object that would be the *output* of a "real" obfuscator. To avoid a formulation where algorithms actually output oracles, we are forced to define a point oracle as a reactive or stateful functionality. While obfuscation in the random oracle model can be defined by quantifying over oracle sets and asserting that an insignificant fraction (more precisely, Lebesgue measure 0, in the style of [BG81]) of oracles are "bad", such a formulation is not possible with point oracles. However, the notion of black box access to functionalities is not new, and might likely be familiar to the reader from Secure Multiparty Computation [Bea92, Can01].

Definition 2 (Oracle algorithms and oracle circuits) *An oracle algorithm (or oracle machine) A^ψ is a Turing machine which can execute queries to the oracle ψ in a single step. An oracle circuit C^ψ is a boolean circuit in which some of the gates are queries to the oracle ψ.*

In the following definitions, we make no restrictions on the running time of any of the algorithms. While proving positive results, however, we will be interested in computationally efficient obfuscators.

Definition 3 (Query bounded algorithm) *A (randomized) oracle algorithm A^ψ is said to be query bounded if the number of queries it makes to the oracle ψ is bounded by some polynomial (of the size of A's input).*

Definition 4 (Oracle obfuscation with statistical security) *A query bounded oracle algorithm $\mathcal{O}^\psi$ which takes a function as input and produces an oracle circuit is said to be an obfuscator of the family $\mathcal{F} = \cup_k \mathcal{F}_k$ if:*

Approximate functionality: *There exists a negligible function ν such that for all k, for all $F \in \mathcal{F}_k$, $\mathbf{P}[\exists x \in \{0,1\}^* \; \mathcal{O}^\psi(F)(x) \neq F(x)] \leq \nu(k)$.*

Polynomial slowdown: *There exists a polynomial p such that for all k, for all $F \in \mathcal{F}_k$ $|\mathcal{O}(F)| \leq p(k)$.*

Virtual black box: *For every query bounded adversarial oracle algorithm A, there exists a query bounded simulator oracle algorithm S and a negligible function ν such that for all k, for all $F \in \mathcal{F}_k$*
$|\mathbf{P}[A^\psi(\mathcal{O}^\psi(F, 1^k)) = 1] - \mathbf{P}[S^F(1^k) = 1]| \leq \nu(k)$.

Efficiency: *The obfuscator $\mathcal{O}^\psi$ is probabilistic polynomial time.*

The probabilities are taken over the randomness of ψ as well as the coin tosses of the obfuscator, the adversary and the simulator.

The fourth property is optional; we take no position on whether it should be enforced or not, since our negative results do not assume it and our positive result demonstrates an efficient obfuscator.

Terminology and notation. A function $\nu(k)$ is *negligible* if for every polynomial p there is a n such that $\nu(k) < \frac{1}{p(k)}$ for $k > n$. By $negl(k)$ we mean the class of functions negligible in k. A function is *significant* if it is not negligible. By $poly(k)$ we mean the class of all functions that grow slower than some polynomial. By $poly(k)g(k)$ we mean $\{f(k)g(k) : \; f(k) \in poly(k)\}$. Thus, for example, $poly(k)2^{-k} \subset negl(k)$.

Point function obfuscation. We briefly review here the constructions of [LPS04] and [Wee05]. For the definitions and proofs we refer the reader to those papers.

Let $\mathcal{O}^\psi(\alpha)$ be a program which stores $r = \psi(\alpha)$ and on input x checks if $\psi(x) = r$, and outputs 1 or 0 accordingly. Then $\mathcal{O}$ is a point function obfuscator in the random oracle model.

This can be converted into a point function obfuscator in the plain model by instantiating the random oracle ψ with the hash function

$$h(x;\tau_1,\tau_2,\ldots\tau_{3k}) = (\tau_1,\tau_2,\ldots\tau_k,\langle x,\tau_1\rangle,\langle\pi(x),\tau_2\rangle,\ldots\langle\pi^{3k-1}(x),\tau_{3k}\rangle)$$

where π is a one-way permutation satisfying certain properties. The obfuscated circuit has the output of the hash function (including the randomness) hard-wired, and on input y verifies if $\langle y,\tau_i\rangle = \langle x,\tau_i\rangle$ for each i.

3 The lensing lemma and an application

The motivation for the lensing lemma is as follows. Many obfuscators will leak some bits of information about the input, which are not leaked by oracle access. This does not necessarily contradict the virtual black box property because the bits that are leaked are dependent on the randomness of the obfuscator. However, one intuitively feels that if "too many" bits are leaked, then simulatability must necessarily be compromised. How many bits have to be leaked before this happens? The trivial answer is that if almost as many bits are leaked as the size of the input, then the (computationally unbounded) adversary can uniquely determine the input. Surprisingly, it turns out that a small (superlogarithmic) number of leaked bits is sufficient. We formalize this below in terms of a "distinguisher" that attempts to determine, given the obfuscation of an unknown function, if it is an obfuscation of a given function.

Lemma 1 (lensing) *If $\mathcal{O}$ is a query bounded oracle obfuscator for the point function family $\mathcal{F} = \cup_k\mathcal{F}_k$, then there exists no distinguisher algorithm $h(\mathcal{O}(F),F')$ such that for every F:*

- $\mathbf{P}[h(\mathcal{O}(F),F)] = 1] \notin negl(k)$
- $\mathbf{P}_{F'\in\mathcal{F}_k}[h(\mathcal{O}(F),F') = 1] \in negl(k)$

The first probability is taken over the randomness of $\mathcal{O}$ and h, and the second probability is taken also over a uniformly chosen $F' \in \mathcal{F}_k$. Observe that we need h *not* to be an oracle algorithm, i.e, it cannot query the oracles in the obfuscated oracle circuit $\mathcal{O}(F)$.

Proof. Let $\mu(k)$ be a negligible function such that $\mathbf{P}_{F'}[h(\mathcal{O}(F),F') = 1] < \mu$. Assume, for convenience, that $\mathcal{F}_k$ is a finite field. Let $H = H(F)$ be the random variable $\{F' \in \mathcal{F}_k : h(\mathcal{O}(F),F') = 1\}$. H can be thought of as the subset of $\mathcal{F}_k$ whose members the distinguisher h thinks $\mathcal{O}(F)$ could be an obfuscation of. Consider an adversary $\mathcal{A}_\sigma$ that has the advice $\sigma = (a,b) \in \mathcal{F}_k^2$ hardwired and behaves on input $\mathcal{O}(F)$ as follows:

Let $L = L(\sigma) = L(\sigma,\mu) = \{ai_{\mathcal{F}} + b\}_{1\le i\le\frac{1}{\mu}}$, where $i_{\mathcal{F}}$ is an embedding of the integers in $\mathcal{F}_k$ (we call L the lens set). Output a random element of $H \cap L$.

Claim 1 *There exists an adversary A_σ that, on any input $\mathcal{O}(F)$ where $F \in L$, outputs F with a non-negligible probability.*

Proof. By a direct counting argument, $E_{H,\sigma}[|H\cap L(\sigma)| \mid F \in H\cap L(\sigma)] = 1 + \frac{|H||L(\sigma)|(|\mathcal{F}_k|-1)}{|\mathcal{F}_k|^2}] < 2$. By an averaging argument, $\exists\sigma$ s.t $E_H[|H\cap L(\sigma)| \mid F \in H\cap L(\sigma)] < 2$. For this σ, it is clear that A_σ outputs F with probability $> \frac{1}{2}\mathbf{P}[h(\mathcal{O}(F),F)] = 1] \notin negl(k)$ (recall that A_σ outputs F with a probability $\frac{1}{|H\cap L|}$ provided that $F \in |H\cap L(\sigma)|$). $\square$

On the other hand, any query bounded oracle simulator, given σ, μ[1] and oracle access to F such that $F \in L(\sigma, \mu)$, can output F with a probability of only $poly(k)\mu(k)$ which is a negligible quantity. This completes the proof of the lensing lemma. $\square$

Note that even though the lemma as stated only applies to point function obfuscators, it is actually relevant to a much larger class of query bounded oracle obfuscators, including everything considered in this paper. The way to apply the lemma is to fix most of the input bits of the function so that the $\mathcal{O}$ behaves like a point function obfuscator on the rest of the bits.

3.1 Application to the plain model

The only adversarial computation in the proof of the above lemma that is inefficient is the need to loop through the elements of L. However, if the nature of h is such that there were a way to generate uniform elements of the set $H \cap L$, then the adversary would be computationally efficient. Therefore it is possible to prove a version of the lemma for the plain model that states that such an algorithm for generating uniform elements cannot exist.

Lemma 2 (lensing, plain model) *If $\mathcal{O}$ is an obfuscator for the point function family $\mathcal{F} = \cup_k \mathcal{F}_k$, and there exists a function $h(\mathcal{O}(F), F')$ such that for every F:*

- $\mathbf{P}[h(\mathcal{O}(F), F)] = 1] \notin negl(k)$
- $\mathbf{P}_{F' \in \mathcal{F}_k}[h(\mathcal{O}(F), F') = 1] \in negl(k)$

then there exists no $L \subset \mathcal{F}_k$ and polynomial q such that

- $|L| \notin poly(k)$
- $\forall F \in L,\ \mathbf{P}[|H(F) \cap L| < q(k) \bigm| F \in H] > \frac{1}{2}$ *where* $H(F) = \{F' \in \mathcal{F}_k : h(\mathcal{O}(F), F') = 1\}$
- *There exists a PPT algorithm to generate uniform elements of $H \cap L$.*

Note that the sampling algorithm above is nonuniform and is restricted to polynomial advice, even though L is superpolynomial.. Also note that h can be any function now; the adversary will never need to compute it.

Proof. Omitted. $\square$

It appears difficult to characterize h for which such an L exists; however, we can answer the question in the affirmative for some specific h such as a system of linear equations. This enables us to make some observations about the construction of [Wee05]. Recall that Wee constructs a point function obfuscator by instantiating a random oracle in the non self-composing construction of [LPS04] (see end of Section 2 above). However, [LPS04] do give a self-composing obfuscation (actually they give a composable obfuscation for a point function with general output; we recall here the special case where the output of the point function is a single bit):

On input P_α, the point function that outputs 1 when its input is α, the obfuscator $\mathcal{O}^\psi$ generates a uniformly random string σ of length k and computes $\beta = \psi(\alpha, \sigma)$; it outputs a circuit which has the values σ and β hardwired and on input x computes the equality test $\psi(x, \sigma) = \beta$.

[1] We could alternately assume that the simulator simply knows μ, since it is a property of the obfuscator.

It is natural to ask what happens when we instantiate the random oracle in the above obfuscator by Wee's construction: for what m is the resulting plain-model obfuscator m-self-composing (m being some function of k)? To answer this question, we observe that if the m point functions being obfuscated are $P_{a_1}, P_{a_2}, \ldots P_{a_m}$ respectively, then adversary learns the values of the following dot products:

$$(\langle a_1 \| \sigma_1, \tau_1^1 \rangle, \langle a_2 \| \sigma_2, \tau_1^2 \rangle, \ldots \langle a_m \| \sigma_m, \tau_1^m \rangle)$$

where $\|$ denotes concatenation (we are assuming fresh randomness for each instantiation). Observe that this is equivalent to learning r_i and b_i in the equations

$$\langle a_i, r_i \rangle = b_i, 1 \leq i \leq m$$

for some $b_1, b_2, \ldots b_m$ where r_i represents the first k bits of τ_1^i.

It is easy to see that, if $m \geq k$, then the construction is insecure, because it allows the adversary to distinguish the case when all the a_i are equal – if $a_1 = a_2 = \ldots a_m = a$(say), then the adversary gets m linear equations in k variables over $\mathbb{GF}(2)$ (each bit of a being a variable).[2] When m is smaller, it is not clear whether or not the instantiation leads to a valid obfuscation.

Using the lensing technique, we can show that it is not a valid obfuscation as long as $m = \omega(\log k)$, which is a considerable improvement over $m \geq k$. The adversary's lensing set is $L(k) = \{x : \langle x, r_i \rangle = 0\}_{m < i \leq k}$, where r_i are arbitrary linearly independent vectors of length k. Each independent linear equation cuts the space of solutions exactly in half, and so $|L(k)| = 2^{k-(k-m)} = 2^m \notin poly(k)$. The distinguisher corresponding to the leaked information can be written as $h(x) = 1$ if $\langle x, r_i \rangle = 0$ for each $1 \leq i \leq m$, and therefore $H = \{x : \langle x, r_i \rangle = 0\}_{1 \leq i \leq m}$.

To show that L and H satisfy the properties of the lensing lemma (Lemma 2), we will show thatit is possible to efficiently generate uniform elements of $H \cap L$ and its size is polynomial (specifically, quadratic) with high probability. The first property is clearly true because $H \cap L$ is simply the solution to a system of linear equations ($(\langle a_i, r_i \rangle = b_i, 1 \leq i \leq k)$, and the normal method of solving the system allows us to generate uniform elements of the solution space, no matter how large it is. Note that the system cannot be inconsistent when $F \in H$. It is also clear that when the dimension of the system is at least $k - 2\log k$, the size of $H \cap L$ is at most $2^{2\log k} = k^2$. We lower bound this probability as follows. Consider the sequence of random variables $\{Dim_n\}_n$ where Dim_n is the rank of the constraint matrix $[r_1 r_2 \ldots r_n]^T$. Dim_{n+1} is either Dim_n or $Dim_n + 1$; the former happens when $r_{n+1} \in \mathsf{Span}(r_1, r_2, \ldots r_n)$. Therefore $\mathbf{P}[Dim_n = Dim_{n+1}] = \frac{|\mathsf{Span}(r_1, r_2, \ldots r_n)|}{2^k} = 2^{n-k}$. By the union bound, $\mathbf{P}[Dim_{k-2\log k} = k - 2\log k] \geq 1 - (k - 2\log k)2^{(k-2\log k)-k} \geq \frac{k-1}{k}$. Thus, $\mathbf{P}[Dim_k \geq k - 2\log k] \geq \mathbf{P}[Dim_{k-2\log k} = k - 2\log k] \geq \frac{k-1}{k}$, as required. Since the lensing lemma asserts that no such L can exist, it follows that the construction is not an obfuscator. □

Exactly the same argument as above shows that if we take the natural obfuscation of the m-way OR function

$$F_{a_1, a_2, \ldots a_m}(x_1, x_2, \ldots x_m) = \bigvee_{i=1}^{m} x_i = a_i$$

[2] Not all these equations are necessarily linearly independent, but it can be shown that with high probability there are sufficiently many independent equations that there are only a polynomial number of solutions to the system. The technique is the same as the one used in the sequel.

(where each a_i is a k-bit string) in the random oracle model, and instantiate the random oracles with Wee's construction, then the resulting algorithm is not an obfuscator for $m = \omega(\log k)$.

4 Tools

We now describe the technical tools that we will need in the rest of the paper. The most important idea is "shadow" evaluation of an oracle circuit. A shadow oracle can be thought of as oracle together with the cached history of queries to it. This technique will allow the query bounded adversary to keep track of past oracle queries so as not to "waste" queries by querying the oracle multiple times on the same input.

Definition 5 *For an oracle ψ, the* shadow oracle *of ψ is an oracle $\mathsf{Shadow}(\psi)$ together with a set $\mathsf{History}(\psi)$, which is the set of all queries that have been made to $\mathsf{Shadow}(\psi)$ so far. $\mathsf{Shadow}(\psi)$ behaves as follows on an input q: if $q \notin \mathsf{History}(\psi)$, then it adds q to $\mathsf{History}(\psi)$, queries $\psi(q)$, returns whatever ψ returns, and stores its response; if $q \in \mathsf{History}(\psi)$, then it returns the same response as it returned the last time it saw q.*

Shadow evaluation of an *oracle circuit* involves "simulating" the output of the oracle on inputs where it has not been queried before. The algorithms below must be thought of as being executed by the adversary, *i.e.*, the obfuscator has already "programmed" the oracle by querying it on certain inputs.

Algorithm 1 (shadow-eval) Input: an oracle circuit C^{ψ}, an input x to the circuit, and a shadow oracle $(\mathsf{Shadow}(\psi), \mathsf{History}(\psi))$.

All gates except oracle gates are evaluated in the normal way. A query q to the oracle ψ is evaluated as follows: if $q \in \mathsf{History}(\psi)$ then the answer is $\mathsf{Shadow}(\psi)(q)$. Otherwise the answer is computed as follows: if ψ is a random oracle, then the answer is a random string of the appropriate size. If ψ is a point oracle, then the answer is simply 0.

Observe that $\mathsf{shadow\text{-}eval}$ does not necessarily compute the same function as $C^{\psi}(x)$ because an oracle gate may be evaluated differently from the answer programmed by the obfuscator on an input not in the history.

Algorithm 2 (build-cache) Input: an oracle circuit $\mathcal{O}^{\psi}(F)$ (which is the output of the obfuscator on the function F), a set $X \subset \{0,1\}^*$ and black-box access to a procedure F_X which will evaluate F on any input in X.

Initialize the shadow oracle of ψ with no history. For each input $x \in X$, evaluate $F_X(x)$ and $\mathsf{shadow\text{-}eval}(\mathcal{O}(F), x)$. If $\mathsf{shadow\text{-}eval}(\mathcal{O}(F), x) \neq F_X(x)$, then query $\mathsf{Shadow}(\psi)$ on the set $Q_x \setminus \mathsf{History}(\psi)$, where Q_x is the set of queries to ψ internally evaluated by $\mathsf{shadow\text{-}eval}(\mathcal{O}(F))(x)$.

Lemma 3 *If $\mathcal{O}^{\psi}$ is an oracle obfuscator with statistical security, $\mathsf{build\text{-}cache}$ is a query bounded oracle algorithm.*

Remark. We don't include X when measuring the input size of $\mathsf{build\text{-}cache}$. Thus, we need to prove that the number of oracle queries it makes is polynomial in k.

Proof. To prove that $\mathsf{build\text{-}cache}$ makes no more than polynomially many oracle queries, we classify the iterations of the algorithm (*i.e.*, each x) into two types. Let $Q_{\mathcal{O}}$ be the set of all oracle queries to ψ made by the obfuscator, and Q_x as above. If $Q_x \cap Q_{\mathcal{O}} \subset \mathsf{History}(\psi)$, then $x \in X_1$, else $x \in X_2$. (Here $\mathsf{History}(\psi)$ is understood to be the history at the beginning of the iteration corresponding to x.)

Claim 2 $\mathbf{P}[\exists x \in X_1 : \textit{shadow-eval}(\mathcal{O}(F), x) \neq F_X(x)] \leq \nu(k)$.

The probability is taken over the randomness of ψ and the coin tosses of the obfuscator and shadow-eval.

Proof. When $x \in X_1$, consider two cases. If $x \in Q_{\mathcal{O}}$, then $\mathsf{Shadow}(\psi)(x) = \psi(x)$. If not, then $\psi(x)$ and $\mathsf{Shadow}(\psi)(x)$ are both uniformly distributed. The claim now follows from the approximate functionality property of $\mathcal{O}(F)$. □

Claim 3 $|\{x \in X_2 : \textit{Shadow}(\mathcal{O}(F))(x) \neq F_X(x)\}| \leq |Q_{\mathcal{O}}|$

Proof. During each iteration with $x \in X_2 \wedge \mathsf{shadow\text{-}eval}(\mathcal{O}(F), x) \neq F_X(x)$, the size of $Q_{\mathcal{O}} \setminus \mathsf{History}(\psi)$ decreases by at least 1, since $Q_x \cap (Q_{\mathcal{O}} \setminus \mathsf{History}(\psi))$ is nonempty and the algorithm queries ψ on the inputs in $Q_x \setminus \mathsf{History}(\psi)$. □

The following observations complete the proof of the lemma: $|Q_{\mathcal{O}}|$ is polynomial in k, build-cache makes no oracle queries during any iteration where $x \in X_1$ with high probability, and makes at most $|\mathcal{O}(F)|$ oracle queries during each iteration where $x \in X_2$ (and, therefore, makes a total of at most $|Q_{\mathcal{O}}||\mathcal{O}(F)|$ queries over inputs $x \in X_2$). □

Algorithm 3 (find-error) Input: $\mathcal{O}^{\psi}(F)$, a set $X \subset \{0,1\}^*$, oracle access to a procedure $\hat{F}_X$ which evaluates $\hat{F}$ on any input in X, and a shadow oracle $(\mathsf{Shadow}(\psi), \mathsf{History}(\psi))$.

The algorithm tries to find an input on which F and $\hat{F}$ differ. It does not always succeed, but whenever it produces an output, it is correct.

For each input x in X, evaluate $\hat{F}_X(x)$ and $\mathsf{shadow\text{-}eval}(\mathcal{O}(F), x)$. Let Q_x be the set of queries to ψ internally evaluated by $\mathsf{shadow\text{-}eval}(\mathcal{O}(F), x)$. If $\mathsf{shadow}(\mathcal{O}(F), x) \neq \hat{F}_X(x)$, then query $\mathsf{Shadow}(\psi)$ on the set $Q_x \setminus \mathsf{History}(\psi)$. Evaluate $\mathcal{O}(F)(x)$. If $\mathcal{O}(F)(x) \neq \hat{F}_X(x)$, then output x and exit.

Lemma 4 *Algorithm find-error is a query bounded oracle algorithm.*

Proof. We classify the inputs x into three sets as follows. If $F(x) = \hat{F}(x)$ and $Q_x \cap Q_{\mathcal{O}} \subset \mathsf{History}(\psi)$, then $x \in X_1$. If $F(x) = \hat{F}(x)$ and $Q_x \cap Q_{\mathcal{O}} \not\subset \mathsf{History}(\psi)$, then $x \in X_2$. If $F(x) \neq \hat{F}(x)$ then $x \in X_3$.

Claim 4 $\mathbf{P}[\exists x \in X_1 : \mathcal{O}(F)_T(x) \neq \hat{F}_X(x)] \leq \nu(k)$.

Proof. Identical to proof of claim 2 because $F(x) = \hat{F}(x) \forall x \in X_1$. □

Claim 5 $|\{x \in X_2 : \mathcal{O}(F)_T(x) \neq \hat{F}_X(x)\}| \leq |Q_{\mathcal{O}}|$

Proof. Identical to proof of claim 3 because $F(x) = \hat{F}(x) \forall x \in X_2$. □

Claim 6 *The probability that find-error makes a nonzero number of oracle queries for more than one $x \in X_3$ is negligible.*

Proof. Consider the first $x \in X_3$. By approximate functionality, $\mathcal{O}(F)(x) = F(x)$ w.h.p. Since $F(x) \neq \hat{F}(x)$, build-cache will output this x and exit with high probability. □

The lemma now follows from claims 4, 5 and 6.

We will now prove a theorem (proof in Appendix A) which, while perhaps not very interesting in itself, is a demonstration of the power of the tools we have built above. It demonstrates a function family for which no obfuscation that embeds only a single random oracle gate in the obfuscated circuit can achieve statistical security. We note that there are no impossibility results so far in the literature that derive from statements about the structure of the obfuscated circuit.

Theorem 1 *There is no oracle obfuscator achieving statistical security for the function family $\mathcal{F} = \cup_k \mathcal{F}_k$ where*

$$\mathcal{F}_k = \{F_{a,b}(x, y) := (x = a \vee y = b)\}_{a,b \in \{0,1\}^k}$$

if the obfuscated circuit is required to have at most one random oracle call. □

5 Separation result

In this section we show that random oracles can obfuscate strictly more than point functions in the query bounded oracle algorithm model. To do this we define a family $\mathcal{F}^{SEP} = \cup_k \mathcal{F}_k^{SEP}$ of functions where $\mathcal{F}_k^{SEP}$ has 2^{2k^2} elements. A member of $\mathcal{F}_k^{SEP}$ will be denoted $F_{\mathbf{a},\mathbf{b}}^{SEP}$ where $\mathbf{a} = \langle a_1, a_2, \ldots a_k \rangle, \mathbf{b} = \langle b_1, b_2, \ldots b_k \rangle$, each a_i and each b_i being a k-bit string. $F_{\mathbf{a},\mathbf{b}}^{SEP}$ is defined as

$$F_{\mathbf{a},\mathbf{b}}^{SEP}(x_1, y_1, z_1, x_2, y_2, z_2, \ldots, x_k, y_k, z_k) = (\bigwedge_{i=1}^{k} MUX(x_i = a_i, y_i = b_i, z_i)) \bigvee_{i=1}^{k} (x_i = a_i \wedge y_i = b_i)$$

The x_i's and y_i's are k-bit strings while the z_i's are single bits. MUX is a multiplexer – it returns its first or its second argument according as its third argument (the "selector" bit) is 0 or 1.

Intuition. A conjunction of many point functions can be expressed as a single point function and can be obfuscated using a single point oracle, and a disjunction of point functions can be obfuscated using multiple point oracles. We suspect that these are essentially all the functions that can be obfuscated with statistical security in the point oracle model. In particular, the function $F_{\mathbf{a},\mathbf{b}}(x_1, y_1, x_2, y_2, \ldots, x_k, y_k) = \wedge_{i=1}^{k}(x_i = a_i \vee y_i = b_i)$ is a first attempt at constructing a function that separates the power of random oracles and point oracles. However, it is not clear how to obfuscate this function family in the random oracle model either. Therefore we make our task easier using MUXes as above, and arrive at $F_{\mathbf{a},\mathbf{b}}(x_1, y_1, z_1, x_2, y_2, z_2, \ldots, x_k, y_k, z_k) = \wedge_{i=1}^{k} MUX(x_i = a_i, y_i = b_i, z_i)$. The obvious way of obfuscating this family in the random oracle model leaks the answer to $(x_i = a_i \wedge y_i = b_i)$, and therefore we modify the function to be obfuscated to leak this bit of information as well.

Theorem 2 *There exists a query bounded random oracle obfuscator for the function family $\mathcal{F}^{SEP}$ described above. This obfuscator is computationally efficient, and further, the virtual black box property is achieved via a PPT uniformly black box simulator.*

What we mean is that the simulator is PPT when viewed as executing the adversary as an oracle, the adversary itself being computationally unbounded.

Proof.

Obfuscator. The following is an obfuscator in the query bounded random oracle model. We have assumed, for convenience, that the obfuscator has access to an unlimited number of distinct, "named" random oracles – two random oracles with different suffixes behave completely independently. This is one way of making sure that the random oracle gates obfuscate point functions even in the presence of each other, the other being to explicitly choose fresh randomness for each invocation of a single random oracle. We will adopt this convention in the rest of the paper.

On input $(\mathbf{a}, \mathbf{b})$ the obfuscator picks $r_1, r_2, \ldots r_k$ of appropriate length[3], uniformly and outputs the circuit

$$\begin{array}{lll} \mathcal{O}(\mathbf{a}, \mathbf{b}) = \psi_0 & (& MUX(\psi_{\ell,1}(x_1) \oplus u_1, \psi_{r,1}(y_1) \oplus v_1, z_1), \\ & & MUX(\psi_{\ell,2}(x_2) \oplus u_2, \psi_{r,2}(y_2) \oplus v_2, z_2), \\ & & \vdots \\ & & MUX(\psi_{\ell,k}(x_k) \oplus u_k, \psi_{r,k}(y_k) \oplus v_k, z_k) \\ &) & = w \\ & & \bigvee_{i=1}^{k} (\psi_{\ell,i}(x_i) \oplus u_i = \psi_{r,i}(y_i) \oplus v_i) \end{array}$$

ψ_0 and $\psi_{\cdot,\cdot}$ are random oracle calls with appropriate input and output size[3]. Finally, $u_i = \psi_i(a_i) \oplus r_i, v_i = \psi_i(b_i) \oplus r_i$ and $w = \psi_0(r_1||r_2||\ldots||r_k)$. It is clear that the obfuscator satisfies the polynomial slowdown property. It is easy to see that it satisfies the approximate functionality property: when $F^{SEP}_{\mathbf{a},\mathbf{b}}(x) = 1$ then $\mathcal{O}(\mathbf{a}, \mathbf{b})(x) = 1$ with probability 1 because all the equality checks are satisfied. When $F^{SEP}_{\mathbf{a},\mathbf{b}}(x) = 0$, an equality check that is supposed to fail will instead succeed with a probability of at most 2^{-2k}, taken over the randomness of ψ. The chance that this happens for at least one input to a specific oracle gate is at most $2^k 2^{-2k} = 2^{-k}$, and by a union bound the chance that it will happen for at least one oracle gate is $poly(k)2^{-k}$.

For a proof of the virtual black box property, see Appendix B.

Proof of impossibility of point oracle obfuscation.

Theorem 3 *There exists no query bounded point oracle obfuscator for the function family $\mathcal{F}^{SEP}$ described above.*

We will assume to the contrary that such an obfuscator $\mathcal{O}$ exists, and arrive at a contradiction by displaying adversaries whose output on $\mathcal{O}(F^{SEP}_{\mathbf{a},\mathbf{b}})$ is a general function of $(\mathbf{a}, \mathbf{b})$, rather than a single bit, and show that there is no simulator for these adversaries. This is purely for convenience – we could instead have displayed a set of adversaries, one corresponding to each of the output bits of our more powerful adversary.

Our proof shows that no adversary simulator can exist, not even a non-black box one. Therefore the question of whether the random oracles are programmable or not becomes moot.

Notation. By $Q^\epsilon(X) = Q^\epsilon(C, X)$ where C is an oracle circuit we mean the set of all pairs (ψ, q) such that ψ gets the query q with a probability $\geq \epsilon$ in the internal evaluation of ψ by the

[3] It is sufficient for the output size of the random oracles to be $2k$ bits, which would give the same lengths for r_i and w, and make the input size of ψ_0 $4k^2$.

algorithm $\mathsf{shadow\text{-}eval}(C, x)$ starting with an empty history on an input x randomly drawn from X. The probability is taken over X as well as the coin tosses of Shadow. Recall that by $Q_{\mathcal{O}}$ we mean the set of all oracle queries made by the obfuscator.

Lemma 5 *There is a query bounded algorithm that computes $Q^{\epsilon}(C, X)$.*

Proof. Evaluating the probabilities in the definition above by exhaustive search involves no querying. □

The next lemma applies to any query bounded oracle obfuscators, whether point oracle or random oracle.

Lemma 6 (Probable queries lemma) *Let X and $Y \supset X$ be nonempty sets such that $F(x) = 1 \ \forall x \in X$ and $F(x) = 0 \ \forall y \in Y \setminus X$. Let $\epsilon = o(\frac{1}{|M(k)||Q_{\mathcal{O}}(k)|})$ where $M(k)$ is an upper bound on $|\mathcal{O}(F)|$ and $Q^{\epsilon}(\mathcal{O}(F), X) \cap Q_{\mathcal{O}} \subset T$ for some T such that $|T| \in poly(k)$. Then there is a query bounded (in k) algorithm $\mathsf{find\text{-}sat}(\mathcal{O}(F), T, Y)$ that outputs some $x \in X$ w.h.p.*

The probability is taken over the randomness of the oracle gates in $\mathcal{O}(F)$ as well coin tosses of the obfuscator and $\mathsf{find\text{-}sat}$.

Proof. $\mathsf{find\text{-}sat}$ simply programs the shadow oracles on the set T and executes $\mathsf{find\text{-}error}$ with the inputs $\mathcal{O}(F), Y, \hat{F}_Y(y) \equiv 0$ and the shadow oracles as described.

The proof hinges on the fact that if for some $x \in X$, every oracle query that is internally evaluated by $\mathsf{shadow\text{-}eval}(\mathcal{O}(F), x)$ is identically distributed with the actual output of the oracle on that query, then $\mathsf{shadow\text{-}eval}(\mathcal{O}(F), x) = \mathcal{O}(F)(x)$. To lower bound the probability of this happening, observe that $\mathsf{Shadow}(\psi)(q)$ can deviate from the distribution of $\psi(q)$ only if $q \notin T \rightarrow q \notin Q^{\epsilon}(X)$. But there are at most $|Q_{\mathcal{O}}(k)|$ queries of this form, and each can occur with a probability of at most ϵ for a random $x \in X$. Taking a union bound over all such queries and all oracles (which are at most $|\mathcal{O}(F)|$ in number), we find that $\mathsf{shadow\text{-}eval}(\mathcal{O}(F), x)$ can deviate from $\mathcal{O}(F)(x)$ with a probability of at most $|\mathcal{O}(F)||Q_{\mathcal{O}}(k)|\epsilon = o(1)$. Thus $\mathsf{find\text{-}error}$ will succeed with a probability at least $\geq 1 - o(1) - negl(k) \in 1 - o(1)$. □

The rest of this section continues with the proof of unobfuscatability of $\mathcal{F}^{SEP}$ in the query bounded point oracle model. See Appendix C.

6 Conclusions

We exposed fundamental limitations of the approach of building circuit obfuscators in the plain model by instantiating obfuscators of the same circuits in the random oracle model. Intuitively, a random oracle is more powerful than a "point oracle" because it hides information about the inputs on which the obfuscator queried it because it outputs random strings instead of a 0/1 value. Understanding this limitation of point oracles is the first step in constructing plain-model obfuscators for functions that are significantly more complex than point functions.

We propose that positive results for obfuscation in the random oracle model should in the future attempt to achieve (query bounded) statistical security. We reiterate that existing results do in fact turn out to achieve this stronger notion. No computational (time or space) bounds should be assumed on the adversary, at least for the reason that it makes it conceptually clear how obfuscation is achieved in terms of existing and understood constructions. The query boundedness property enables us to enforce a clean separation between the security of obfuscation deriving from underlying primitives and that deriving from other computational hardness assumptions. For impossibility results, it is better to consider computationally bounded adversaries, although proofs involving the structure of the obfuscated circuit might only be possible in the unbounded setting.

There are three areas for further research that our work opens up:

1. Impossibility results for random oracle obfuscation with statistical security. In particular, it appears likely that there is no random oracle obfuscator for AC^0 achieving statistical security.

2. We think that our definitional approach as well as proof techniques will find applications beyond those considered in this paper. For instance, it appears possible to formalize and prove the statement "the existence of one-way functions cannot be proved to follow from the existence of point function obfuscators via black box reduction".

3. Obfuscating the circuit of Section 5 might be possible in the plain model by making extra assumptions on the super-strong one-way permutations of [Wee05].

Acknowledgements. The authors would like to thank Vinod Vaikuntanathan, Raghuvardhan Meka and Adam Klivans for their useful insights.

References

[BBP04] M. Bellare, A. Boldyreva, and A. Palacio. An uninstantiable random-oracle-model scheme for a hybrid-encryption problem. In *Proc. Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 171–188. Springer, 2004.

[Bea92] D. Beaver. Foundations of secure interactive computing. In *Proc. Advances in Cryptology - CRYPTO 1991*, volume 576 of *LNCS*, pages 377–391. Springer, 1992.

[BG81] C. Bennett and J. Gill. Relative to a random oracle $A, P^A \neq NP^A \neq co-NP^A$ with probability 1. *SIAM journal of computing*, 10:96–113, 1981.

[BGI+01] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In *Proc. Advances in Cryptology - CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, 2001.

[BR93] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *Proc. 1st ACM Conference on Computer and Communications Security*, pages 62–73. ACM, 1993.

[Can97] R. Canetti. Towards realizing random oracles: hash functions that hide all partial information. In *Proc. Advances in Cryptology - CRYPTO 1997*, volume 1294 of *LNCS*, pages 455–469. Springer, 1997.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *IEEE Symposium on Foundations of Computer Science*, pages 136–145, 2001.

[CGH04a] R. Canetti, O. Goldreich, and S. Halevi. On the random-oracle methodology as applied to length-restricted signature schemes. In *Proc. 1st Theory of Cryptography Conference (TCC)*, volume 2951 of *LNCS*, pages 40–57. Springer, 2004.

[CGH04b] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[CMR98] R. Canetti, D. Micciancio, and O. Reingold. Perfectly one-way probabilistic hash functions. In *Proc. 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 131–140. ACM, 1998.

[DS05] Y. Dodis and A. Smith. Correcting errors without leaking partial information. In *Proc. 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 654–663. ACM, 2005.

[GK03] S. Goldwasser and Y. Tauman Kalai. On the (in)security of the Fiat-Shamir paradigm. In *Proc. 44th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 102–113. IEEE Computer Society, 2003.

[IR89] R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. In *Proc. 21st ACM symposium on Theory of computing (STOC)*, pages 44–61. ACM Press, 1989.

[LPS04] B. Lynn, M. Prabhakaran, and A. Sahai. Positive results and techniques for obfuscation. In *Proc. Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 20–39. Springer, 2004.

[NS05] A. Narayanan and V. Shmatikov. Obfuscated databases and group privacy. In *Proc. 12th ACM Conference on Computer and Communications Security*, pages 102–111. ACM, 2005.

[VV85] L. Valiant and V. Vazirani. NP is as easy as detecting unique solutions. In *Proc. 17th Annual ACM Symposium on Theory of Computing (STOC)*, pages 458–463. ACM, 1985.

[Wee05] H. Wee. On obfuscating point functions. In *Proc. 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 523–532. ACM, 2005.

A Proof of theorem 1

Assume such an obfuscator $\mathcal{O}^{\psi}(F)$ exists. Consider the three families of adversaries $\mathcal{A}_0$, $\{\mathcal{A}_1^{\alpha}\}_{\alpha\in\{0,1\}^k}$ and $\{\mathcal{A}_2^{\beta}\}_{\beta\in\{0,1\}^k}$. We assume that an upper bound on the running time of the obfuscator is known.[4]

$\mathcal{A}_0$ executes find-error on the inputs $X = \{0,1\}^{2k}$, the constant 0-function and a shadow oracle with an empty history and outputs whatever find-error outputs. Note that if $\mathcal{A}_0$ outputs anything, it will be a string (x, y) such that $F_{a,b}(x, y) = 1$.

$\mathcal{A}_1^{\alpha}$ has the value α hardwired into it and executes as follows:

[4] This assumption is made simply for convenience. If such a bound is not known, we consider a sequence of families of adversaries where the adversaries in the i^{th} family all assume that the obfuscator runs in time $O(k^i)$.

- $\mathcal{A}_1$ picks a random β and evaluates $\mathcal{O}(F)(\alpha, \beta)$. If it is not 1 then $\mathcal{A}_1$ aborts.
- $\mathcal{A}_1$ executes build-cache on the inputs $X = \{(\alpha, y)\}_{y \in \{0,1\}^k}$ and the procedure F_X where $F_X(x) \equiv 1$. Denote by T the final value of $\mathsf{History}(\psi)$.
- $\mathcal{A}_1$ computes the following: for each $z \in Q_T$, $Q^{-1}(z)$ is the set of all y for $\mathsf{shadow\text{-}eval}(\mathcal{O}(F), \alpha, y)$ internally evaluated ψ on the input z.
- For each z having $|Q^{-1}(z) = 1|$, $\mathcal{A}_1$ evaluates $\mathcal{O}(F)(\alpha', y)$ where α' is any value different from α and y is the unique member of $Q^{-1}(z)$. If $\mathcal{O}(F)(\alpha', y) = 1$ for some y, then $\mathcal{A}_1$ outputs y.

Claim 7 *Algorithm $\mathcal{A}_1$ is a query bounded oracle algorithm.*

Proof. The first step ensures that $\mathcal{A}_1$ aborts (w.h.p) unless the $\alpha = a$. Therefore on the set X, F is indeed identically 1. From lemma 3 the cache T is of polynomial size, and so is the set $\{z \in Q_T : |Q^{-1}(z) = 1|\}$. The claim follows. □

$\mathcal{A}_2^{\beta}$ has the value β hardwired and executes as follows:

- $\mathcal{A}_2$ picks a random α and evaluates $\mathcal{O}(F)(\alpha, \beta)$. If it is not 1 then $\mathcal{A}_2$ aborts.
- Executes $\mathsf{build\text{-}cache}$ with $X = \{(x, \beta)\}_{x \in \{0,1\}^k}$ and F_X being the constant 1 function. Denote by T the final value of $\mathsf{History}(\psi)$.
- Execute $\mathsf{find\text{-}error}$ with $X = \{(x, y) \in \{0,1\}^{2k} : y \neq \beta\}$, the constant 1-function, and the shadow oracle as programmed by $\mathsf{build\text{-}cache}$ above. If $\mathsf{find\text{-}error}$ outputs (x, y) then output x.

Claim 8 *Algorithm $\mathcal{A}_2$ is a query bounded oracle algorithm.*

Proof. The first step ensures that $\beta = b$ with high probability. The claim now follows from the query boundedness properties of $\mathsf{build\text{-}cache}$ and $\mathsf{find\text{-}error}$. □

Claim 9 *On the input $F_{a,b}$, with significant probability, either $\mathcal{A}_0$ outputs (x, y) such that $F_{\alpha,\beta}(x, y) = 1$ or $\mathcal{A}_1^a$ outputs b or $\mathcal{A}_2^b$ outputs a.*

Proof. We consider three cases:

1. $\mathsf{shadow\text{-}eval}(\mathcal{O}(F), a, b) = 1$ when $\mathsf{History}(\psi) = \phi$. Observe that this implies that $\mathsf{shadow\text{-}eval}(\mathcal{O}(F)\ , a, b) = 1$ w.h.p., regardless of $\mathsf{History}(\psi)$. Therefore $\mathcal{A}_0$ will succeed w.h.p.
2. $\mathsf{shadow\text{-}eval}(\mathcal{O}(F), a, b) = 0$ when $\mathsf{History}(\psi) = \phi$ and $\mathbf{P}[\exists \beta :\ Q_{a,b} = Q_{a,\beta}] < \frac{1}{2}$.

   The probability is taken over the randomness of the obfuscated circuit. The probability is independent of $\mathsf{History}(\phi)$ (since there is only a single oracle gate), and hence refers both to the execution of $\mathsf{build\text{-}cache}$ by $\mathcal{A}_1$ and $\mathsf{find\text{-}error}$ by $\mathcal{A}_2$.

   In this case, $\mathcal{A}_1$ finds that $|Q^{-1}(a, b)| = 1$ (with probability at least $\frac{1}{2}$) and evaluates $\mathcal{O}(F)(\alpha, b)$ for some α which evaluates to 1 w.h.p. and therefore $\mathcal{A}_1$ succeeds in outputting b.

3. $\mathsf{shadow\text{-}eval}(\mathcal{O}(F), a, b) = 0$ when $\mathsf{History}(\psi) = \phi$ and $\mathbf{P}[\exists \beta : \ Q_{a,b} = Q_{a,\beta}] \geq \frac{1}{2}$.

   Now after $\mathcal{A}_2$ executes build-cache, with probability at least $\frac{1}{2}$, $\mathsf{History}(\psi)$ contains $Q_{a,b} = Q_{a,\beta}$ (since $\mathsf{shadow\text{-}eval}(\mathcal{O}(F), a, b) = 0$ when $\mathsf{History}(\psi) = \phi$). Therefore find-error computes $\mathsf{shadow\text{-}eval}(\mathcal{O}(F), a, \beta) = 1$, and hence find-error outputs (a, β) w.h.p whereupon $\mathcal{A}_2$ outputs a.

To complete the proof of the theorem, we argue as follows: for $\mathcal{O}$ to satisfy the virtual black box property, there must be ideal world simulators S_0, $\{S_1^a\}_a$, and $\{S_2^b\}_b$ corresponding to the real world adversaries described above. Then it must be the case that if α and β are picked at random, then on the input $F_{a,b}$, with significant probability, either S_0 outputs x, y such that $F_{a,b}(x, y) = 1$ or S_1^a outputs b or S_2^b outputs a. Clearly none of these can happen with a non-negligible probability.□

B Proof of Theorem 2, virtual black-box property

We begin by describing the simulator.

Simulator. The simulator simulates the view of the adversary. Initially it outputs a fake obfuscated circuit by generating u_i, v_i and w uniformly. The simulator traps the random oracle queries and answers them as described below. Finally the simulator outputs whatever the adversary outputs. Initially the sets $Q_0, Q_{\ell,i}$ and $Q_{r,i}$ are set to $\{\}$ for $1 \leq i \leq k$. They denote the set of queries made by the adversary to the random oracles $\psi_0, \psi_{\ell,i}$ and $\psi_{r,i}$ respectively at any point in the simulation.

A query p to the random oracle $\psi_{\ell,1}$ is answered as follows. For each $q \in Q_{r,1}$, the simulator picks $x_2, y_2, \ldots x_k, y_k$ uniformly and queries the $F^{SEP}_{\mathbf{a},\mathbf{b}}$ oracle on $(p, q, 0, x_2, y_2, 0, x_3, y_3, 0, \ldots x_k, y_k, 0)$. If the answer is 1 for some $q = q_0$, it returns $\psi_{r,1}(q_0) \oplus u_1 \oplus v_1$. If the answer is 0 for all $q \in Q_{r,1}$ then it picks a value r uniformly. If $r \oplus u_1 \in Q_1$ then the simulator aborts. Otherwise it returns r. The simulation is symmetric in left and right and is similar for all the other random oracles $\psi_{\cdot,\cdot}$.

A query $\langle p_1, p_2, \ldots p_k \rangle$ to the random oracle ψ_0 is simulated by constructing a query to the $F^{SEP}_{\mathbf{a},\mathbf{b}}$ oracle as follows. First, $x_1, y_1, \ldots x_k, y_k$ are selected uniformly. For each i: If there is some $q_{\ell,i} \in Q_{\ell,i}$ such that $\psi_{\ell_i}(q_{\ell,i}) \oplus u_i = p_i$ then the simulator sets $x_i = q_{\ell,i}$ and $z_i = 0$. Else if there is some $q_{r,i} \in Q_{r,i}$ such that $\psi_{r_i}(q_{r,i}) \oplus v_i = p_i$ then the simulator sets $y_i = q_{r,i}$ and $z_i = 1$.

If there is some z_i that has not been set to either 1 or 0, then the simulator returns a uniform value. If not then the simulator queries the $F^{SEP}_{\mathbf{a},\mathbf{b}}$ oracle on $(x_1, y_1, z_1, x_2, y_2, z_2, \ldots x_k, y_k, z_k)$. If the answer is 1 then it returns w. Else it returns a uniform value.

Lemma 7 *The simulator above satisfies the virtual black box property.*

The following 4 claims are trivial and we omit their proofs.

Claim 10 *The probability that the simulator aborts is $poly(k)2^{-k}$.* □

Claim 11 *The joint distribution of u_i, v_i and w is uniform in both the real and the simulated world.* □

Claim 12 *The responses of $\psi_{\cdot,i}$ are uniform and independent except on the input a_i or b_i in both the real world and simulated world.* $\square$

Claim 13 *In the simulation of $\psi_{\cdot,i}$, the probability that $p = a_i \wedge q = b_i$ is false and the answer to the simulator's query is true is $O(2^{-k})$.* $\square$

In the real world, $(\psi_{\ell,i}(a_i), \psi_{r,i}(b_i))$ is uniform subject to the constraint that $\psi_{\ell,i}(a_i) \oplus u_i = \psi_{r,i}(b_i) \oplus v_i$. We need to show that this property holds in the simulated world as well. Wlog, the adversary queries a_i before b_i. Then it is clear from the simulator code that $\psi_{\ell,i}(a_i)$ is uniform, and $\psi_{r,i}(b_i)$ is precisely the value that will satisfy the above constraint. Finally, we consider responses to queries to ψ_0. It is easy to verify that in both worlds, whenever the query is different from $(\psi_{\ell,1}(a_1) \oplus u_1, \psi_{\ell,2}(a_2) \oplus u_2, \ldots, \psi_{\ell,k}(a_k) \oplus u_k)$ (which is the same as $(\psi_{r,1}(b_1) \oplus v_1, \psi_{r,2}(b_2) \oplus v_2, \ldots, \psi_{r,k}(b_k) \oplus v_k)$), then the answer is uniform and when it is not, the answer is w. This completes the proof of the lemma (virtual black box property), and of Theorem 2. $\square$

C Proof of Theorem 3, continuation

Notation. The notation $\mathsf{Inp}(W_1, W_2, \ldots W_k)$ represents the set of inputs $W_1 \times W_2 \times \ldots \times W_k$ to $F^{SEP}_{\mathbf{a},\mathbf{b}}$ where $W_i \subset \{0,1\}^{2k+1}$ for each i (i.e, (x_i, y_i, z_i)). We will use some shortcuts to represent some special types of sets W_i. First, if we omit to specify W_i, then $W_i = \{(x_i, y_i, z_i) : x_i = a_i, y_i$ is arbitrary and $z_i = 0$ or x_i is arbitrary, $y_i = b_i$ and $z_i = 1\}$. By $W_i = *$ we mean $W_i = \{0,1\}^{2k+1}$. By $\boxed{a_i}$ we mean $x_i = a_i, y_i$ is arbitrary and $z_i = 0$. $\boxed{b_i}$ is defined analogously. $\boxed{\alpha_i}$ and $\boxed{\beta_i}$ are similar to $\boxed{a_i}$ and $\boxed{b_i}$ except that they are used when $\alpha_i \neq a_i$ or $\beta_i \neq b_i$. Thus, for example, we will write $\mathsf{Inp}(W_1 = \boxed{a_1}, W_i = *)$ to denote the set of inputs that satisfy the following constraints: $x_1 = a_1$; $x_j = a_j \vee y_j = b_j$ for $j \notin \{1, i\}$.

Some observations to aid intuition: $\mathsf{Inp}()$ is exactly the set of satisfying inputs to $F^{SEP}_{\mathbf{a},\mathbf{b}}$; $\mathsf{Inp}(W_i = \boxed{a_i}) \cup \mathsf{Inp}(W_i = \boxed{b_i}) = \mathsf{Inp}()$; A set of inputs contains no satisfying inputs if and only if it contains a constraint of the form $W_i = \boxed{\alpha_i}$ or $W_i = \boxed{\beta_i}$.

Lemma 8 *There is some significant ϵ and negligible δ such that*

$$\mathbf{P}[Q^{\epsilon}(S(W_1 = \boxed{a_1})) \cap Q_{\mathcal{O}} \not\subset Q^{\delta}(S(W_1 = *)) \cup_{i=2}^{k} Q^{\delta}(S(W_1 = *, W_i = *))] \geq 1 - \frac{1}{k}$$

The probability is taken over the coin tosses of the obfuscator.

This lemma can be considered a formalization of the notion that there is an oracle query that is a probable when the evaluator "knows" a_1 but not otherwise.

Proof. To prove this, we set $\epsilon = o(\frac{1}{M(k)|Q_{\mathcal{O}}(k)|})$. For this ϵ, we consider the smallest δ for which the inequality holds. If it is significant, then there is some $\delta' < \delta$ (say $\delta' = \frac{\delta}{2}$) which is also significant but for which the inequality *does not* hold. Consider an adversary $\mathcal{A}_1$ that has $\mathbf{a} \setminus \{a_1\}$ and $\mathbf{b}$ hardwired. It computes $T = Q^{\delta'}(S(W_1 = *)) \cup_{i=2}^{k} (Q^{\delta'}(S(W_1 = *, W_i = *)))$. Observe that $|T| \leq \frac{k}{\delta'} \in poly(k)$. By the probable queries lemma, $\mathcal{A}_1$ outputs a_1 with a nonnegligible probability (by setting $X = S(W_1 = \boxed{a_1}), Y = S(W_1 = *)$ and observing that with a probability at least $\frac{1}{k}, Q^\epsilon(X) \subset T$). $\square$

We continue with the proof of the theorem. [The membership assertions in the following hold with high probability over the coin tosses of the obfuscator, rather than with certainty, but we ignore this for convenience of presentation.]

Let $(\psi, q) \in Q^\epsilon(S(W_1 = \boxed{a_1})) - T$, where T is as above. Note that for each index $i > 1$ we must have $(\psi, q) \in Q^\epsilon(S(W_1 = \boxed{a_1}, W_i = \boxed{a_i})) - T$ or $(\psi, q) \in Q^\epsilon(S(W_1 = \boxed{a_1}, W_i = \boxed{b_i})) - T$. Therefore we must have $(\psi, q) \in Q^{\epsilon/2}(S(W_1 = \boxed{a_1}, W_i = \boxed{a_i})) - T$ or $(\psi, q) \in Q^{\epsilon/2}(S(W_1 = \boxed{a_1}, W_i = \boxed{b_i})) - T$. Let us call an index i "bad" if only one of the two inclusions holds. It is simple to show that if there are t bad indices, then $\epsilon < \frac{\epsilon}{2} + 2^{-t}$. Since ϵ is significant, there can be only $O(\log k)$ bad indices. Therefore an index picked at random is good w.h.p. Wlog, let this good index be named 2, i.e, $(\psi, q) \in Q^{\epsilon/2}(S(W_1 = \boxed{a_1}, W_2 = \boxed{a_2}))$ and $(\psi, q) \in Q^{\epsilon/2}(S(W_1 = \boxed{a_1}, W_2 = \boxed{b_2}))$.

The next claim can be understood as stating that if q is a probable query to ψ regardless of whether the disjunction $x_2 = a_2 \vee y_2 = b_2$ is satisfied because $x_2 = a_2$ or because $y_2 = b_2$, then it is a probable query regardless of whether the disjunction is satisfied at all.

Claim. $(\psi, q) \in Q^{\epsilon_1}(S(W_1 = \boxed{a_1}, W_2 = *)) - T$ for some significant ϵ_1. **Proof.** It is enough to prove that $(\psi, q) \in Q^{\epsilon_1}(S(W_1 = \boxed{a_1}, W_2 = *))$. Assume the contrary. Consider the largest ϵ_1 for which the inclusion holds. Suppose it is insignificant. Consider $\epsilon' = 2\epsilon_1$ so that the inclusion *does not* hold for ϵ'. Consider the adversary $\mathcal{A}_2$ that has $\mathbf{a}$ and $\mathbf{b} \setminus b_2$ hardwired. Since $(\psi, q) \in Q^{\epsilon/2}(S(W_1 = \boxed{a_1}, W_2 = \boxed{a_2}))$, $\mathcal{A}_2$ can guess (ψ, q) with a significant probability. Observe that $\mathbf{P}_{b'_2}[(\psi, q) \in Q^{\epsilon/2}(S(W_1 = \boxed{a_1}, W_2 = \boxed{b'_2}))] \geq \theta \rightarrow (\psi, q) \in Q^{\theta\epsilon/2}(S(W_1 = \boxed{a_1}, W_2 = *))$. By assumption, $\frac{\theta\epsilon}{2} \leq \epsilon'$ or $\theta \leq \frac{2\epsilon'}{\epsilon} \in negl(k)$. But $(\psi, q) \in Q^{\epsilon/2}(S(W_1 = \boxed{a_1}, W_2 = \boxed{b_2}))$. This provides a distinguisher for b_2, which contradicts the lensing lemma. $\square$

Claim. $(\psi, q) \notin Q^\delta(S(W_1 = *, W_2 = *))$ where δ is as above. **Proof.** $(\psi, q) \notin Q^\delta(S(W_1 = *, W_2 = *)) - T$ (because that set is empty) and $(\psi, q) \notin T$.

We are finally ready to complete the proof. The cumulative effect of the previous two Claims is to identify a way for the adversary to glean information from the obfuscated circuit even when one component (i.e, both a_2 and b_2) of the inputs is completely missing. This leaves the simulator empty handed, because it needs some information about at least one of a_i and b_i for each i to submit anything meaningful to the ideal functionality. We formalize this below.

Consider an adversary $\mathcal{A}_3$ that has $\mathbf{a} \setminus \{a_1, a_2\}, \mathbf{b} \setminus \{b_1, b_2\}$ hardwired. Further, it has hardwired a random set A_1 of size $\omega(|Q_\mathcal{O}|^2)$. $\mathcal{A}_3$'s goal is to output a_1 assuming $a_1 \in A_1$.

- For each $x \in A_1$: $\mathcal{A}_3$ computes $Q^{\epsilon_1}(S(W_1 = \boxed{x}, W_2 = *)) \cap Q_\mathcal{O}$ and picks a random member (ψ, q) thereof. Call it Q_x.

  While $\mathcal{A}_3$ cannot compute $Q_\mathcal{O}$, it *can* compute the intersection of $Q_\mathcal{O}$ with any polynomial sized set. It is here that we leverage the difference between a point oracle and a random oracle – given a random oracle and a point z it is impossible to tell whether or not the oracle has seen z.

- $\mathcal{A}_3$ outputs a random x with the property $\pi(x)$ that $Q_y \neq Q_x$ for any $y \neq x$.

Observe that $Q_{a_1} \notin T$ with probability at least $\frac{1}{|Q_\mathcal{O}|}$. Whenever this happens, the following two assertions hold:

- a_1 satisfies π w.h.p. To see this, observe that if $(\psi, q) \in Q^{\epsilon}(S(W_1 = \boxed{a_1}, W_2 = *)) - T$, then $\mathbf{P}_{x \in \{0,1\}^k}[(\psi, q) \in Q^{\epsilon'}(S(W_1 = \boxed{x}, W_2 = *))] \geq \theta \to (\psi, q) \in Q^{\epsilon'\theta}(S(W_1 = *, W_2 = *))$, implying $\theta\epsilon' < \delta$ or $\theta = negl(k)$. By a union bound, we have $\mathbf{P}[\exists y \in A_1 \setminus \{a_1\} : \ (\psi, q) = Q_y] = negl(k)$.

- At most $|Q_{\mathcal{O}}|$ different x's can satisfy π.

Therefore $\mathcal{A}_3$ succeeds with a probability of at least $\frac{1}{|Q_{\mathcal{O}}|^2}$.

On the other hand, a simulator that is given $\mathbf{a} \setminus \{a_1, a_2\}, \mathbf{b} \setminus \{b_1, b_2\}$ and a set A_1 of size $\omega(|Q_{\mathcal{O}}|^2)$ has only a negligible probability of making any queries to the ideal functionality where its output is nonzero, and so cannot gain any additional information. The simulator's probability of outputting a_1 is $\frac{1}{|A_1|} + negl(k)$ which is significantly smaller than $\frac{1}{|Q_{\mathcal{O}}|^2}$.

This completes the proof of the impossibility of a query bounded point oracle obfuscator for $\mathcal{F}$ (Theorem 3). $\square$