Separating Quantum and Classical Learning

Rocco A. Servedio[1]

Division of Engineering and Applied Sciences, Harvard University
Cambridge, MA 02138
rocco@deas.harvard.edu

Abstract. We consider a model of learning Boolean functions from *quantum membership queries*. This model was studied in [26], where it was shown that any class of Boolean functions which is information-theoretically learnable from polynomially many quantum membership queries is also information-theoretically learnable from polynomially many classical membership queries.
In this paper we establish a strong *computational* separation between quantum and classical learning. We prove that if any cryptographic one-way function exists, then there is a class of Boolean functions which is polynomial-time learnable from quantum membership queries but not polynomial-time learnable from classical membership queries. A novel consequence of our result is a quantum algorithm that breaks a general cryptographic construction which is secure in the classical setting.

1 Introduction

Over the past decade the study of quantum computation has generated much excitement and attracted intense research attention. One of the most interesting aspects of this new computing paradigm is the possibility that polynomial-time quantum computation may be strictly more powerful than polynomial-time classical computation, i.e. that the quantum class BQP may strictly contain BPP. Evidence for this possibility has been provided by Shor [27], who gave polynomial-time quantum algorithms for factoring and discrete logarithms, two problems which are not known to have polynomial-time classical algorithms.

Since many important learning problems are not known to be solvable in polynomial time, from a learning theory perspective the prospect of gaining computing power via quantum computation is quite intriguing. It is natural to ask whether efficient quantum algorithms can be designed for learning problems (such as the problem of learning DNF formulae) which have thus far resisted efforts to construct polynomial-time algorithms. The ultimate goal of research along these lines would be to construct quantum algorithms which learn using traditional (classical) example oracles, but such algorithms are not yet known.

1.1 Previous Work

As a first step in this direction, several researchers have studied quantum learning algorithms which have access to *quantum oracles*; so in this framework the source of examples as well as the learning algorithm itself is assumed to operate in a quantum fashion. The first work along these lines is due to Bshouty and Jackson [10] who defined a quantum PAC oracle and gave an efficient algorithm for learning DNF from a uniform-distribution quantum PAC oracle.

In a different community, many complexity theory researchers have studied the power of quantum computation with a black-box quantum oracle [3–8, 11, 13, 16, 28, 31]. This research has focused on understanding the relationship between the number of quantum versus classical black-box oracle queries required to determine whether or not a black-box function has some particular property such as ever taking a nonzero value [4, 6, 11, 16, 31] or being evenly balanced between the outputs zero and one [13].

Since a classical black-box oracle query to a Boolean function is the same thing as a membership query in learning theory, in light of the work described above it is natural to consider a model of *learning from quantum membership queries*. In contrast with the work described above, the goal here is to *exactly identify* the black-box function rather than to determine whether or not it has some property. Servedio and Gortler [26] defined such a model and proved that any concept class C which is information-theoretically learnable from polynomially many quantum membership queries is also information-theoretically learnable from polynomially many classical membership queries (they also gave a similar result for the quantum PAC learning model of [10]).

This result from [26] deals only with query complexity and does not address the *computational* complexity of quantum versus classical learning. Indeed, [26] also showed that polynomial-time quantum learning is more powerful than polynomial-time classical learning under the assumption that factoring is computationally hard for classical computers. This follows directly from the observation that Shor's quantum factoring algorithm enables quantum algorithms to efficiently learn concept classes whose classical learnability is directly related to the hardness of factoring [2, 20].

1.2 Our Results

We give strong evidence that efficient quantum learning algorithms are more powerful than efficient classical learning algorithms. Our main result is the following theorem:

Theorem 1. *If any one-way function exists, then there is a concept class C which is polynomial-time learnable from quantum membership queries but is not polynomial-time learnable from classical membership queries.*

This separation between quantum and classical learning is far more robust than previous work [26]. Even if a polynomial-time classical factoring algorithm were to be discovered, our separation would hold as long as *any* one-way function exists (a universally held belief in public-key cryptography).

The main cryptographic tool underlying our results is a new construction of pseudorandom functions which are invariant under an XOR mask (see Section 4). As described in Section 5.1, each concept $c \in C$ combines these new pseudorandom functions with pseudorandom permutations in a particular way. Roughly speaking, the XOR mask invariance of the new pseudorandom functions ensures that a quantum algorithm due to Simon [28] can be used to extract some information about the structure of the target concept and thus make progress towards learning. On the other hand, the pseudorandomness ensures that no probabilistic polynomial-time learning algorithm can extract any useful information, and thus no such algorithm can learn successfully.

As discussed in Section 6, our results prove the existence of a quantum oracle algorithm which defeats a general cryptographic construction secure in the classical setting. More precisely, given any one-way function we construct a family of pseudorandom functions which are classically secure but can be distinguished from truly random functions (and in fact exactly identified) by an algorithm which makes quantum oracle queries. To our knowledge, this is the first break of a general cryptographic protocol (not based on a specific assumption such as factoring) in a quantum setting.

2 Preliminaries

A *concept* is a Boolean function $c : \{0,1\}^n \rightarrow \{0,1\}$. A *concept class* $C = \cup_{n \geq 1} C_n$ is a collection of concepts with $C_n = \{c \in C : \ c \text{ is a concept over } \{0,1\}^n\}$.

For $\alpha, \beta \in \{0,1\}$ we write $\alpha \oplus \beta$ to denote the exclusive-or $\alpha + \beta \pmod 2$. Similarly for $x, y \in \{0,1\}^n$ we write $x \oplus y$ to denote the n-bit string which is the bitwise XOR of x and y. We write $x \cdot y$ to denote the inner product $x_1 y_1 + \cdots + x_n y_n \pmod 2$, and we write $|x|$ to denote the length of string x.

We use script capital letters to denote probability distributions over sets of functions; in particular $\mathcal{F}_n$ denotes the uniform distribution over all 2^{n2^n} functions from $\{0,1\}^n$ to $\{0,1\}^n$. If S is a finite set we write $\Pr_{s \in S}$ to denote a uniform choice of s from S.

We write $M(s)$ to indicate that algorithm M is given string s as input and M^g to indicate that M has access to an oracle for the function g. If M is a probabilistic polynomial-time (henceforth abbreviated p.p.t.) algorithm which has access to an oracle $g : \{0,1\}^{\ell_1} \rightarrow \{0,1\}^{\ell_2}$, then the running time of M^g is bounded by $p(\ell_1 + \ell_2)$ for some polynomial p.

In the model of *exact learning from membership queries* [1, 9, 14, 18, 19], the learning algorithm is given access to an oracle for an unknown *target concept* $c \in C_n$. When invoked on input $x \in \{0,1\}^n$, the oracle returns $c(x)$ (such an oracle call is known as a *membership query*). The goal of the learning algorithm is to construct a hypothesis $h : \{0,1\}^n \rightarrow \{0,1\}$ which is logically equivalent to c, i.e. $h(x) = c(x)$ for all $x \in \{0,1\}^n$. We say that a concept class C is *polynomial-time learnable from membership queries* if there is a p.p.t. oracle algorithm A with the following property: for all $n \geq 1$, for all $c \in C_n$, with probability at least $9/10$ A^c outputs a Boolean circuit h which is equivalent to c.

Detailed descriptions of quantum Turing machine and circuit models and of quantum oracle computation are given in [5, 12, 25, 30]. Most of the details are irrelevant for our needs, so we describe here only the essential aspects.

A quantum computation takes place over an m-bit *quantum register*. At any stage in the execution of a quantum computation the state of this register is some superposition $\sum_{z \in \{0,1\}^m} \alpha_z |z\rangle$ of basis states $|z\rangle$, where the α_z are complex numbers which satisfy the constraint $\sum_{z \in \{0,1\}^m} ||\alpha_z||^2 = 1$. We may thus view the state of a quantum register as being a 2^m-dimensional complex vector of unit norm. A single step of a quantum computation corresponds to a unitary transformation of this vector, i.e. a $2^m \times 2^m$

unitary matrix, and a quantum algorithm is defined by a sequence of such matrices. A single quantum computation step is analogous to a single gate in a classical circuit; in order to ensure that each step of quantum computation performs only a bounded amount of work, the model stipulates that each unitary matrix must be simple and local in a well-defined sense. At the end of a quantum computation a *measurement* is performed on the register and an m-bit string is obtained as output. If the register's final state is $\sum_{z\in\{0,1\}^m} \alpha_z|z\rangle$ then with probability $||\alpha_z||^2$ the string $z \in \{0,1\}^m$ is the output.

Quantum computation with access to an oracle $c : \{0,1\}^n \rightarrow \{0,1\}^\ell$ proceeds as follows. If the oracle c is invoked when the current state of the quantum register is $\sum_{x\in\{0,1\}^n, y\in\{0,1\}^\ell, w\in\{0,1\}^{m-n-\ell}} \alpha_{x,y,w}|x, y, w\rangle$, then at the next step the state of the register will be $\sum_{x\in\{0,1\}^n, y\in\{0,1\}^\ell, w\in\{0,1\}^{m-n-\ell}} \alpha_{x,y,w}|x, y\oplus c(x), w\rangle$. (It can be verified that this change of state is a unitary transformation for any oracle c.) Thus a quantum algorithm can invoke its oracle on a superposition of inputs and receives a corresponding superposition of responses. This model of quantum computation with an oracle has been studied by many researchers in complexity theory [3–8, 11, 13, 16, 28, 31] and has also recently been studied from a learning theory perspective [26].

We say that a concept class C is *polynomial-time learnable from quantum membership queries* if there is a quantum oracle algorithm A with the following property: for all $n \geq 1$, for all $c \in C_n$, A^c runs for at most poly(n) steps and with probability at least $9/10$ outputs a representation of a Boolean circuit h which is logically equivalent to c. It is well known that any classical oracle algorithm can be efficiently simulated by a quantum oracle algorithm, and thus any concept class which is polynomial-time learnable from classical membership queries is polynomial-time learnable from quantum membership queries.

3 Simon's Algorithm

Let $f : \{0,1\}^n \rightarrow \{0,1\}^n$ be a function and let $0^n \neq s \in \{0,1\}^n$. We say that f is *two-to-one with XOR mask* s if for all $y \neq x$, $f(x) = f(y) \iff y = x\oplus s$. More generally, f is *invariant under XOR mask with* s if $f(x) = f(x\oplus s)$ for all $x \in \{0,1\}^n$ (note that such a function need not be two-to-one).

Simon [28] has given a simple quantum algorithm which takes oracle access to a function $f : \{0,1\}^n \rightarrow \{0,1\}^n$, runs in poly$(n)$ time, and behaves as follows:

1. If f is a permutation on $\{0,1\}^n$, the algorithm outputs an n-bit string y which is uniformly distributed over $\{0,1\}^n$.
2. If f is two-to-one with XOR mask s, the algorithm outputs an n-bit string y which is uniformly distributed over the 2^{n-1} strings such that $y \cdot s = 0$.
3. If f is invariant under XOR mask with s, the algorithm outputs some n-bit string y which satisfies $y \cdot s = 0$.

Simon showed that by running this procedure $O(n)$ times a quantum algorithm can distinguish between Case 1 (f is a permutation) and Case 3 (f is invariant under some XOR mask) with high probability. In Case 1 after $O(n)$ repetitions the strings obtained will with probability $1 - 2^{-O(n)}$ contain a basis for the vector space $(Z_2)^n$ (here we are viewing n-bit strings as vectors over Z_2), while in Case 3 the strings obtained cannot contain such a basis since each string must lie in the subspace $\{y : y \cdot s = 0\}$. Simon also observed that in Case 2 (f is two-to-one with XOR mask s) the algorithm can be used to efficiently identify s with high probability. This is because after $O(n)$ repetitions, with high probability s will be the unique nonzero vector whose dot product with each y is 0; this vector can be found by solving the linear system defined by the y's.

Simon also analyzed the success probability of classical oracle algorithms for this problem. His analysis establishes the following theorem:

Theorem 2. *Let $0^n \neq s \in \{0,1\}^n$ be chosen uniformly and let $f : \{0,1\}^n \rightarrow \{0,1\}^n$ be an oracle chosen uniformly from the set of all functions which are two-to-one with XOR mask s. Then (i) there is a polynomial-time quantum oracle algorithm which identifies s with high probability; (ii) any p.p.t. classical oracle algorithm identifies s with probability $1/2^{\Omega(n)}$.*

This surprising ability of quantum oracle algorithms to efficiently find s is highly suggestive in the context of our quest for a learning problem which separates polynomial-time classical and quantum computation. Indeed, Simon's algorithm will play a crucial role in establishing that the concept class which we construct in Section 5 is learnable in poly(n) time by a quantum algorithm. Recall that in our learning scenario, though, the goal is to *exactly identify* the unknown target function, not just to identify the string s. Since $2^{\Omega(n)}$ bits are required to specify a randomly chosen function f which is two-to-one with XOR mask s, no algorithm (classical or quantum) can output a description of f in poly(n) time, much less learn f in poly(n) time. Thus it will not do to use truly random functions for our learning problem; instead we use *pseudorandom* functions as described in the next section.

4 Pseudorandomness

A *pseudorandom function family* [15] is a collection of functions $\{f_s : \{0,1\}^{|s|} \rightarrow \{0,1\}^{|s|}\}_{s \in \{0,1\}^*}$ with the following two properties: (i) *(efficient evaluation)* there is a deterministic algorithm which, given an n-bit seed s and an n-bit input x, runs in time poly(n) and outputs $f_s(x)$; (ii) *(pseudorandomness)* for all polynomials Q, all p.p.t. oracle algorithms M, and all sufficiently large n, we have that $\left|\Pr_{F \in \mathcal{F}_n}[M^F \text{ outputs } 1] - \Pr_{s \in \{0,1\}^n}[M^{f_s} \text{ outputs } 1]\right| < \frac{1}{Q(n)}$. Intuitively, the pseudorandomess property ensures that in any p.p.t. computation which uses a truly random function, a randomly chosen pseudorandom function may be used instead without affecting the outcome in a noticeable way. Well known results [15, 17] imply that pseudorandom function families exist if and only if any one-way function exists.

A *pseudorandom permutation family* is a pseudorandom function family with the added property that each function $f_s : \{0,1\}^{|s|} \rightarrow \{0,1\}^{|s|}$ is a permutation. Luby and Rackoff [21] gave the first construction of a pseudorandom permutation family from any pseudorandom function family. In their construction each permutation $f_s : \{0,1\}^n \rightarrow \{0,1\}^n$ has a seed s of length $|s| = 3n/2$ rather than n as in our definition above. Subsequent constructions [22–24] of pseudorandom permutation families $\{f_s : \{0,1\}^n \rightarrow \{0,1\}^n\}$ use n-bit seeds and hence match our definition exactly. (Our definition of pseudorandomness could easily be extended to allow seed lengths other than n. For our construction in Section 5 it will be convenient to have n-bit seeds.)

4.1 Pseudorandom Functions Invariant under XOR Mask

Our main cryptographic result, stated below, is proved in Appendix A:

Theorem 3. *If any one-way function exists, then there is a pseudorandom function family* $\{g_s : \{0,1\}^{|s|} \to \{0,1\}^{|s|}\}$ *such that* $g_s(x) = g_s(x \oplus s)$ *for all* $|x| = |s|$.

A first approach to constructing such a family is as follows: given any pseudorandom function family $\{f_s\}$, let $\{g_s\}$ be defined by

$$g_s(x) \stackrel{\text{def}}{=} f_s(x) \oplus f_s(x \oplus s). \tag{1}$$

This simple construction ensures that each function g_s is invariant under XOR mask with s, but the family $\{g_s\}$ need not be pseudorandom just because $\{f_s\}$ is pseudorandom. Indeed, if $\{h_s\}$ is similarly defined by $h_s(x) \stackrel{\text{def}}{=} g_s(x) \oplus g_s(x \oplus s)$, then $\{h_s\}$ is not pseudorandom since $h_s(x) = (f_s(x) \oplus f_s(x \oplus s)) \oplus (f_s(x \oplus s) \oplus f_s(x \oplus s \oplus s) = 0^n$.

While this example shows that (1) does not always preserve pseudorandomness, it leaves open the possibility that (1) may preserve pseudorandomness for certain function families $\{f_s\}$. In Appendix A we show that if $\{f_s\}$ is a pseudorandom function family which is constructed from any one-way function in a particular way, then the family $\{g_s\}$ defined by (1) is indeed pseudorandom.

It may at first appear that the pseudorandom function family $\{g_s\}$ given by Theorem 3 immediately yields a concept class which separates efficient quantum learning from efficient classical learning. The pseudorandomness of $\{g_s\}$ ensures that no p.p.t. algorithm can learn successfully; on the other hand, if Simon's quantum algorithm is given oracle access to a function which is two-to-one with XOR mask s, then it can efficiently find s with high probability. Hence it may seem that given access to g_s Simon's quantum algorithm can efficiently identify the seed s and thus learn the target concept.

The flaw in this argument is that each function g_s from Theorem 3, while invariant under XOR mask with s, need not be two-to-one. Indeed g_s could conceivably be invariant under XOR mask with, say, $\sqrt{n}$ linearly independent strings $s = s^1, s^2, \ldots, s^{\sqrt{n}}$. Such a set of strings spans a $2^{\sqrt{n}}$-element subspace of $\{0,1\}^n$; even if Simon's algorithm could identify this subspace, it would not indicate which element of the subspace is the true seed s. Hence a more sophisticated construction is required.

5 Proof of Theorem 1

5.1 The Concept Class C

We describe concepts over $\{0,1\}^m$ where $m = n + 2\log n + 1$. Each concept in C_m is defined by an $(n+1)$-tuple $(y, s^1, \ldots, s^n)$ where $y = y_1 \ldots y_n \in \{0,1\}^n$ and each $s^i \in \{0,1\}^n \setminus \{0^n\}$, so C_m contains $2^n(2^n - 1)^n$ distinct concepts. For brevity we write $\overline{s}$ to stand for $s^1, \ldots, s^n$ below.

Roughly speaking, each concept in C_m comprises n pseudorandom functions; as explained below the string y acts as a "password" and the strings $s^1, \dots, s^n$ are the seeds to the pseudorandom functions. Each concept $c \in C_m$ takes m-bit strings as inputs; we view such an m-bit input as a 4-tuple (b, x, i, j) where $b \in \{0,1\}$, $x \in \{0,1\}^n$, and $i, j \in \{0,1\}^{\log n}$ each represent a number in the range $\{1, 2, \dots, n\}$.

Let $\{h^0_s : \{0,1\}^{|s|} \to \{0,1\}^{|s|}\}_{s \in \{0,1\}^*}$ be a pseudorandom permutation family and let $\{h^1_s : \{0,1\}^{|s|} \to \{0,1\}^{|s|}\}_{s \in \{0,1\}^*}$ be the pseudorandom function family from Theorem 3, so $h^1_s(x) = h^1_s(x \oplus s)$. The concept $c_{y,\overline{s}}$ is defined as follows on input (b, x, i, j) :

- **If $b = 0$**: A query $(0, x, i, j)$ is called a *function query.* The value of $c_{y,\overline{s}}(0, x, i, j)$ is $h^{y_i}_{s^i}(x)_j$, i.e. the j-th bit of the n-bit string $h^{y_i}_{s^i}(x)$. Thus the bit y_i determines whether the i-th pseudorandom function used is a permutation or is invariant under XOR mask with s^i.
- **If $b = 1$**: A query $(1, x, i, j)$ is called a *seed query.* The value of $c_{y,\overline{s}}(1, x, i, j)$ is 0 if $x \neq y$ and is s^i_j (the j-th bit of the i-th seed s^i) if $x = y$.

The intuition behind our construction is simple: in order to learn the target concept successfully a learning algorithm must identify each seed string $s^1, \dots, s^n$. These strings can be identified by making seed queries $(1, y, i, j)$, but in order to make the correct seed queries the learning algorithm must know y. Since each bit y_i corresponds to whether an oracle is a permutation or is XOR-mask invariant, a quantum algorithm can determine each y_i and thus can learn successfully. However, no p.p.t. algorithm can distinguish between these two types of oracles (since in either case the oracle is pseudorandom and hence is indistinguishable from a truly random function), so no p.p.t. algorithm can learn y.

5.2 A Quantum Algorithm Which Learns C in Polynomial Time

Theorem 4. *C is polynomial-time learnable from quantum membership queries.*

Proof sketch: Let $c_{y,\overline{s}} \in C_m$ be the target concept. Each function $h^{y_i}_{s^i}$ is a permutation iff $y_i = 0$ and is XOR-mask invariant iff $y_i = 1$ (this is why we do not allow $s^i = 0^n$ in the definition of the concept class). Using quantum membership queries, a poly(n)-time quantum algorithm can run Simon's procedure n times, once for each function $h^{y_i}_{s^i}$, and thus determine each bit y_i with high probability. (One detail which arises here is that Simon's algorithm uses an oracle $\{0,1\}^n \to \{0,1\}^n$ whereas in our learning setting the oracle outputs one bit at a time. This is not a problem since it is possible to simulate any call to Simon's oracle by making n sequential calls, bit by bit, to our oracle.) Given the string $y = y_1 \dots y_n$, the algorithm can then make n^2 queries on inputs $(1, y, i, j)$ for $1 \leq i, j \leq n$ to learn each of the n strings $s^1, \dots, s^n$. Once y and $s^1, \dots, s^n$ are known it is straightforward to output a circuit for $c_{y,\overline{s}}$. $\square$

5.3 No Classical Algorithm Learns C in Polynomial Time

Theorem 5. *C is not polynomial-time learnable from classical membership queries.*

Let $C'_m \supset C_m$, $|C'_m| = 2^{n^2+n}$ be the concept class $C'_m = \{c_{y,\overline{s}} : y, s^1, \dots, s^n \in \{0,1\}^n\}$; thus C'_m includes concepts in which s^i may be 0^n. The following lemma states that it is hard to learn a target concept chosen uniformly from C'_m :

Lemma 1. *For all polynomials Q, all p.p.t. learning algorithms A, and all sufficiently large n, $\Pr_{c_{y,\overline{s}} \in C'_m}[A^{c_{y,\overline{s}}}$ outputs a hypothesis $h \equiv c_{y,\overline{s}}] < \frac{1}{Q(n)}$.*

To see that Lemma 1 implies Theorem 5, we note that the uniform distribution over C'_m and the uniform distribution over C_m are nearly identical (the two distributions have total variation distance $O(n/2^n)$). Lemma 1 thus has the following analogue for C_m which clearly implies Theorem 5:

Lemma 2. *For all polynomials Q, all p.p.t. learning algorithms A, and all sufficiently large n, $\Pr_{c_{y,\overline{s}} \in C_m}[A^{c_{y,\overline{s}}}$ outputs a hypothesis $h \equiv c_{y,\overline{s}}] < \frac{1}{Q(n)}$.*

The proof of Lemma 1 proceeds as follows: we say that a learning algorithm A *hits* y if at some point during its execution A makes a seed query $(1, y, i, j)$, and we say that A *misses* y if A does not hit y. We have that

$$\begin{aligned}
\Pr_{c_{y,\overline{s}} \in C'_m}[A^{c_{y,\overline{s}}} \text{ outputs } h \equiv c_{y,\overline{s}}] &= \Pr[A^{c_{y,\overline{s}}} \text{ outputs } h \equiv c_{y,\overline{s}} \ \ \& \ \ A^{c_{y,\overline{s}}} \text{ hits } y] + \\
&\quad \Pr[A^{c_{y,\overline{s}}} \text{ outputs } h \equiv c_{y,\overline{s}} \ \ \& \ \ A^{c_{y,\overline{s}}} \text{ misses } y] \\
&\leq \Pr[A^{c_{y,\overline{s}}} \text{ hits } y] + \\
&\quad \Pr[A^{c_{y,\overline{s}}} \text{ outputs } h \equiv c_{y,\overline{s}} \ \mid \ A^{c_{y,\overline{s}}} \text{ misses } y].
\end{aligned}$$

Lemma 1 thus follows from the following two lemmas:

Lemma 3. *For all polynomials Q, all p.p.t. learning algorithms A, and all sufficiently large n, $\Pr_{c_{y,\overline{s}} \in C'_m}[A^{c_{y,\overline{s}}}$ hits $y] < \frac{1}{Q(n)}$.*

Lemma 4. *For all polynomials Q, all p.p.t. learning algorithms A, and all sufficiently large n, $\Pr_{c_{y,\overline{s}} \in C'_m}[A^{c_{y,\overline{s}}}$ outputs $h \equiv c_{y,\overline{s}} \mid A^{c_{y,\overline{s}}}$ misses $y] < \frac{1}{Q(n)}$.*

Proof of Lemma 3. The idea of the proof is as follows: before hitting y for the first time algorithm A gets 0 as the answer to each seed query, so A might as well be querying a modified oracle which answers 0 to *every* seed query. We show that no p.p.t. algorithm which has access to such an oracle can output y with inverse polynomial success probability (intuitively this is because such an oracle consists entirely of pseudorandom functions and hence can provide no information to any p.p.t. algorithm), and thus A's probability of hitting y must be less than $1/\text{poly}(n)$ as well.

More formally, let A be any p.p.t. learning algorithm. Without loss of generality we may suppose that A always makes exactly $q(n)$ seed queries during its execution for some polynomial q. Let $X^1, \dots, X^{q(n)}$ be the sequence of strings in $\{0,1\}^n$ on which $A^{c_{y,\overline{s}}}$ makes its seed queries, i.e. $A^{c_{y,\overline{s}}}$ uses $(1, X^t, i_t, j_t)$ as its t-th seed query. Each X^t is a random variable over the probability space defined by the uniform choice of $c_{y,\overline{s}} \in C'_m$ and any internal randomness of algorithm A.

For each $c_{y,\overline{s}} \in C'_m$ let $\tilde{c}_{y,\overline{s}} : \{0,1\}^m \to \{0,1\}$ be a modified version of $c_{y,\overline{s}}$ which answers 0 to all seed queries, i.e. $\tilde{c}_{y,\overline{s}}(b,x,i,j)$ is $c_{y,\overline{s}}(b,x,i,j)$ if $b=0$ and is 0 if $b=1$. Consider the following algorithm B which takes access to an oracle for $\tilde{c}_{y,\overline{s}}$ and outputs an n-bit string. B executes algorithm $A^{\tilde{c}_{y,\overline{s}}}$ (note that the oracle used is $\tilde{c}_{y,\overline{s}}$ rather than $c_{y,\overline{s}}$), then chooses a uniform random value $1 \le t \le q(n)$ and outputs $\tilde{X}^t$, the string on which $A^{\tilde{c}_{y,\overline{s}}}$ made its t-th seed query. Like the X^ts, each $\tilde{X}^t$ is a random variable over the probability space defined by a uniform choice of $c_{y,\overline{s}} \in C'_m$ and any internal randomness of A.

The following two lemmas together imply Lemma 3:

Lemma 5. $2q(n)^2 \cdot \Pr_{c_{y,\overline{s}} \in C'_m}[B^{\tilde{c}_{y,\overline{s}}}$ *outputs* $y] \ge \Pr_{c_{y,\overline{s}} \in C'_m}[A^{c_{y,\overline{s}}}$ *hits* $y]$.

Lemma 6. $\left|\Pr_{c_{y,\overline{s}} \in C'_m}[B^{\tilde{c}_{y,\overline{s}}} \text{ outputs } y] - \frac{1}{2^n}\right| < \frac{1}{Q(n)}$ *for all polynomials Q and all sufficiently large n.*

Proof of Lemma 5. We have that

$$\sum_{t=1}^{q(n)} \Pr[X^t = y \ \& \ X^\tau \ne y \text{ for } \tau < t] \le \sum_{t=1}^{q(n)} \Pr[X^t = y \mid X^\tau \ne y \text{ for } \tau < t].$$

Since the left side of this inequality is exactly $\Pr_{c_{y,\overline{s}} \in C'_m}[A^{c_{y,\overline{s}}} \text{ hits } y]$, for some value $1 \le t_0 \le q(n)$ we have

$$\Pr[X^{t_0} = y \mid X^\tau \ne y \text{ for } \tau < t_0] \ \ge \ \Pr[A^{c_{y,\overline{s}}} \text{ hits } y]/q(n). \qquad (2)$$

Since the distribution of responses to function queries which A makes prior to its first seed query is the same regardless of whether the oracle is $c_{y,\overline{s}}$ or $\tilde{c}_{y,\overline{s}}$, it is clear that the random variables X^1 and $\tilde{X}^1$ are identically distributed. An inductive argument shows that for all $t \ge 1$, the conditional random variables $X^t \mid (X^\tau \ne y \text{ for } \tau < t)$ and $\tilde{X}^t \mid (\tilde{X}^\tau \ne y \text{ for } \tau < t)$ are identically distributed (in each case the conditioning ensures that the distribution of responses to seed queries which A makes prior to its t-th seed query is the same, i.e. all 0).

We consider two possible cases. If $\Pr_{c_{y,\overline{s}} \in C'_m}[\tilde{X}^\tau \ne y \text{ for } \tau < t_0] > 1/2$, then

$$\begin{aligned}\Pr_{c_{y,\overline{s}} \in C'_m}[B^{\tilde{c}_{y,\overline{s}}} \text{ outputs } y] &\ge \Pr[B^{\tilde{c}_{y,\overline{s}}} \text{ chooses } t_0] \cdot \Pr[\tilde{X}^{t_0} = y \ \& \ \tilde{X}^\tau \ne y \text{ for } \tau < t_0] \\ &= \frac{\Pr[\tilde{X}^{t_0} = y \mid \tilde{X}^\tau \ne y \text{ for } \tau < t_0] \cdot \Pr[\tilde{X}^\tau \ne y \text{ for } \tau < t_0]}{q(n)} \\ &> \Pr[\tilde{X}^{t_0} = y \mid \tilde{X}^\tau \ne y \text{ for } \tau < t_0]/2q(n) \\ &= \Pr[X^{t_0} = y \mid X^\tau \ne y \text{ for } \tau < t_0]/2q(n) \\ &\ge \Pr[A^{c_{y,\overline{s}}} \text{ hits } y]/2q(n)^2. \qquad \text{by (2)}\end{aligned}$$

Otherwise if $\Pr_{c_{y,\overline{s}} \in C'_m}[\tilde{X}^\tau \ne y \text{ for } \tau < t_0] \le 1/2$, then $\sum_{t=0}^{t_0-1} \Pr[\tilde{X}^t = y] \ge 1/2$ and hence $\Pr_{c_{y,\overline{s}} \in C'_m}[B^{\tilde{c}_{y,\overline{s}}} \text{ outputs } y]$ is at least

$$\sum_{t=1}^{t_0-1} \Pr[B^{\tilde{c}_{y,\overline{s}}} \text{ chooses } t] \cdot \Pr[\tilde{X}^t = y] \ge \frac{1}{2q(n)} \ge \frac{\Pr[A^{c_{y,\overline{s}}} \text{ hits } y]}{2q(n)^2}. \qquad \square$$

Proof of Lemma 6. For $z, \zeta \in \{0,1\}^n$ let $p^z_\zeta = \Pr_{c_{y,\overline{s}} \in C'_m}[B^{\tilde{c}_{y,\overline{s}}} \text{ outputs } z \mid y = \zeta]$. For $\ell \in \{1, \ldots, n\}$ let $\zeta || \ell$ denote ζ with the ℓ-th bit flipped. Similarly, for $S \subseteq \{1, \ldots, n\}$ let $\zeta || S$ denote ζ with bits flipped in all positions corresponding to S.

Fix $z, \zeta \in \{0,1\}^n$ and $\ell \in \{1, \ldots, n\}$ and consider the following algorithm $D_{z,\zeta,\ell}$ which takes access to an oracle $f : \{0,1\}^n \to \{0,1\}^n$ and outputs a single bit: For all $i \neq \ell$ algorithm $D_{z,\zeta,\ell}$ first chooses a random n-bit string s^i. $D_{z,\zeta,\ell}$ then runs algorithm B, simulating the oracle for B as follows:

- queries $(0, x, \ell, j)$ are answered with the bit $f(x)_j$
- for $i \neq \ell$ queries $(0, x, i, j)$ are answered with the bit $h^{\zeta_i}_{s^i}(x)_j$
- all queries $(1, x, i, j)$ are answered with the bit 0.

Finally algorithm $D_{z,\zeta,\ell}$ outputs 1 if B's output is z and outputs 0 otherwise.

It is easy to verify that for all z, ζ, ℓ we have $p^z_\zeta = \Pr_{s \in \{0,1\}^n}[D^{h^{\zeta_\ell}_s}_{z,\zeta,\ell} \text{ outputs } 1]$ and $p^z_{\zeta||\ell} = \Pr_{s \in \{0,1\}^n}[D^{h^{1-\zeta_\ell}_s}_{z,\zeta,\ell} \text{ outputs } 1]$. From the definition of pseudorandomness and the triangle inequality it follows that $|p^z_\zeta - p^z_{\zeta||\ell}| < \frac{1}{nQ(n)}$. Making $|S| \leq n$ applications of this inequality and using the triangle inequality, we find that $|p^z_\zeta - p^z_{\zeta||S}| < \frac{1}{Q(n)}$. We thus have that $|p^z_\zeta - p^z_z| < \frac{1}{Q(n)}$ for all $z, \zeta \in \{0,1\}^n$. Since $\sum_{z \in \{0,1\}^n} p^z_\zeta = 1$, we have that $\left|\Pr_{c_{y,\overline{s}} \in C'_m}[B^{\tilde{c}_{y,\overline{s}}} \text{ outputs } y] - \frac{1}{2^n}\right| = \left|\frac{1}{2^n}\left(\sum_{z \in \{0,1\}^n} p^z_z\right) - \frac{1}{2^n}\right| = \frac{1}{2^n}\left|\sum_{z \in \{0,1\}^n}(p^z_z - p^z_\zeta)\right| < \frac{1}{Q(n)}$. □

Proof of Lemma 4. The idea here is that conditioning on the event that A misses y ensures that the only information which A has about y and $\overline{s}$ comes from querying oracles for the pseudorandom functions $h^{y_i}_{s^i}$. Since these pseudorandom functions are indistinguishable from truly random functions, no p.p.t. algorithm can learn successfully.

Formally, let A be any p.p.t. learning algorithm. Consider the following algorithm B which takes access to an oracle $h^{y_n}_{s^n} : \{0,1\}^n \to \{0,1\}^n$ and outputs a representation of a function $g : \{0,1\}^n \to \{0,1\}^n$. Algorithm B first chooses $\hat{y} = y_1 \ldots y_{n-1}$ uniformly from $\{0,1\}^{n-1}$ and chooses $n-1$ strings $s^1, \ldots, s^{n-1}$ each uniformly from $\{0,1\}^n$. B then runs algorithm $A^{\tilde{c}_{y,\overline{s}}}$ (observe that B can simulate the oracle $\tilde{c}_{y,\overline{s}}$ since it has access to an oracle for $h^{y_n}_{s^n}$ and knows y_i, s^i for $i \neq n$) which generates some hypothesis h. Finally B outputs the function $g : \{0,1\}^n \to \{0,1\}^n$ defined by $g(x) \stackrel{\text{def}}{=} h(0, x, n, 1)h(0, x, n, 2) \ldots h(0, x, n, n)$.

The following two lemmas together imply Lemma 4:

Lemma 7. *For all sufficiently large n,* $\Pr_{y_n \in \{0,1\}, s^n \in \{0,1\}^n}[B^{h^{y_n}_{s^n}} \text{ outputs } g \equiv h^{y_n}_{s^n}] > \Pr_{c_{y,\overline{s}} \in C'_m}[A^{c_{y,\overline{s}}} \text{ outputs } h \equiv c_{y,\overline{s}} \mid A^{c_{y,\overline{s}}} \text{ misses } y]/2$.

Lemma 8. $\Pr_{y_n \in \{0,1\}, s^n \in \{0,1\}^n}[B^{h^{y_n}_{s^n}} \text{ outputs } g \equiv h^{y_n}_{s^n}] < \frac{1}{Q(n)}$ *for all polynomials Q and all sufficiently large n.*

Proof of Lemma 7. It is easy to see that if $A^{\tilde{c}_{y,\overline{s}}}$ outputs a hypothesis which is equivalent to $c_{y,\overline{s}}$, then g will be equivalent to $h^{y_n}_{s^n}$. For sufficiently large n we thus have that

$\Pr_{y_n \in \{0,1\}, s^n \in \{0,1\}^n}[B^{h^{y_n}_{s^n}} \text{ outputs } g \equiv h^{y_n}_{s^n}]$ is at least

$$\begin{aligned}\Pr_{c_{y,\overline{s}} \in C'_m}[A^{\tilde{c}_{y,\overline{s}}} \text{ outputs } h \equiv c_{y,\overline{s}}] &\geq \Pr[A^{\tilde{c}_{y,\overline{s}}} \text{ outputs } h \equiv c_{y,\overline{s}} \;\&\; A^{\tilde{c}_{y,\overline{s}}} \text{ misses } y] \\ &= \Pr[A^{\tilde{c}_{y,\overline{s}}} \text{ outputs } h \equiv c_{y,\overline{s}} \mid A^{\tilde{c}_{y,\overline{s}}} \text{ misses } y] \cdot \\ &\quad \Pr[A^{\tilde{c}_{y,\overline{s}}} \text{ misses } y] \\ &> \Pr[A^{\tilde{c}_{y,\overline{s}}} \text{ outputs } h \equiv c_{y,\overline{s}} \mid A^{\tilde{c}_{y,\overline{s}}} \text{ misses } y]/2\end{aligned}$$

where the last inequality follows from Lemma 3.

Let $TRANS(A^{c_{y,\overline{s}}})$ ($TRANS(A^{\tilde{c}_{y,\overline{s}}})$ respectively) denote a complete transcript of algorithm A's execution on oracle $c_{y,\overline{s}}$ ($\tilde{c}_{y,\overline{s}}$ respectively). $TRANS(A^{c_{y,\overline{s}}})$ and $TRANS(A^{\tilde{c}_{y,\overline{s}}})$ are each random variables over the probability space defined by a uniform choice of $c_{y,\overline{s}} \in C'_m$ and any internal randomness of algorithm A. An easy induction shows that the two conditional random variables $TRANS(A^{\tilde{c}_{y,\overline{s}}}) \mid (A^{\tilde{c}_{y,\overline{s}}} \text{ misses } y)$ and $TRANS(A^{c_{y,\overline{s}}}) \mid (A^{c_{y,\overline{s}}} \text{ misses } y)$ are identically distributed. This implies that $\Pr_{c_{y,\overline{s}} \in C'_m}[A^{\tilde{c}_{y,\overline{s}}} \text{ outputs } h \equiv c_{y,\overline{s}} | A^{\tilde{c}_{y,\overline{s}}} \text{ misses } y] = \Pr_{c_{y,\overline{s}} \in C'_m}[A^{c_{y,\overline{s}}} \text{ outputs } h \equiv c_{y,\overline{s}} | A^{c_{y,\overline{s}}} \text{ misses } y]$, which combined with the inequality above proves the lemma. □

Proof of Lemma 8. The following fact, which follows easily from the pseudorandomness of $\{h^0\}$ and $\{h^1\}$, states that $\{h^b_s\}_{b \in \{0,1\}, s \in \{0,1\}^n}$ is a pseudorandom function family:

Fact 1 *For all polynomials Q, p.p.t. oracle algorithms A, and sufficiently large n, we have* $\left|\Pr_{b \in \{0,1\}, s \in \{0,1\}^n}[A^{h^b_s} \textit{ outputs } 1] - \Pr_{F \in \mathcal{F}_n}[A^F \textit{ outputs } 1]\right| < \frac{1}{Q(n)}$.

Intuitively the pseudorandomness of $\{h^b_s\}$ should make it hard for $B^{h^{y_n}_{s^n}}$ to output $h^{y_n}_{s^n}$ since clearly no p.p.t. algorithm, given oracle access to a truly random function F, could output a function equivalent to F. Formally, we consider an algorithm D which takes oracle access to a function $f : \{0,1\}^n \to \{0,1\}^n$ and outputs a single bit. D runs B^f to obtain a function g and then selects a string $z \in \{0,1\}^n$ which was not used as an oracle query in the computation of B^f. D calls the oracle to obtain $f(z)$, evaluates g to obtain $g(z)$, and ouputs 1 if the two values are equal and 0 otherwise.

Clearly $\Pr[D^f \text{ outputs } 1] \geq \Pr[B^f \text{ outputs } g \equiv f]$. Since $\Pr_{F \in \mathcal{F}_n}[D^F \text{ outputs } 1] = 1/2^n$, using Fact 1 we find that $\left|\Pr_{y_n \in \{0,1\}, s^n \in \{0,1\}^n}[D^{h^{y_n}_{s^n}} \text{ outputs } 1] - \frac{1}{2^n}\right| < \frac{1}{2Q(n)}$ and hence $\Pr_{y_n \in \{0,1\}, s^n \in \{0,1\}^n}[B^{h^{y_n}_{s^n}} \text{ outputs } g \equiv h^{y_n}_{s^n}] < \frac{1}{Q(n)}$. □

6 Breaking Classical Cryptography in a Quantum Setting

Our constructions highlight some interesting issues concerning the relation between quantum oracle computation and classical cryptography. It is clear that a quantum algorithm, given access to a quantum black-box oracle for an unknown function, can efficiently distinguish between truly random functions and pseudorandom functions drawn from the family $\{g_s\}$ of Theorem 3. Our construction of $\{g_s\}$ thus shows that cryptographic constructions which are provably secure in the classical model can fail in a quantum setting. We emphasize that this failure does *not* depend on the ability of polynomial-time quantum algorithms to invert particular one-way functions such as factoring; even if no quantum algorithm can efficiently invert the one-way function used to construct $\{g_s\}$, our results show that a polynomial-time quantum algorithm can be a successful distinguisher. It would be interesting to obtain stronger constructions of pseudorandom functions which are provably secure in the quantum oracle framework.

Acknowledgements. We thank S. Gortler and A. Klivans for stimulating discussions.

References

1. D. Angluin. Queries and concept learning. *Machine Learning* **2** (1988), 319-342.
2. D. Angluin and M. Kharitonov. When won't membership queries help? *J. Comp. Syst. Sci.* **50** (1995), 336-355.
3. R. Beals, H. Buhrman, R. Cleve, M. Mosca and R. de Wolf. Quantum lower bounds by polynomials, *in* "Proc. 39th Symp. on Found. of Comp. Sci.," (1998), 352-361.
4. C. Bennett, E. Bernstein, G. Brassard and U. Vazirani. Strengths and weaknesses of quantum computing, *SIAM J. Comp.* **26**(5) (1997), 1510-1523.
5. E. Bernstein & U. Vazirani. Quantum complexity theory, *SICOMP* **26**(5) (1997), 1411-1473.
6. M. Boyer, G. Brassard, P. Høyer, A. Tapp. Tight bounds on quantum searching, *Forschritte der Physik* **46**(4-5) (1998), 493-505.
7. G. Brassard, P. Høyer and A. Tapp. Quantum counting, *in* "Proc. 25th Int. Conf. on Automata, Languages and Programming" (1998), 820-831.
8. G. Brassard and P. Høyer. An exact quantum polynomial-time algorithm for Simon's problem, *in* "Proc. Fifth Israeli Symp. on Theory of Comp. and Systems" (1997), 12-23.
9. N. Bshouty, R. Cleve, R. Gavaldà, S. Kannan and C. Tamon. Oracles and queries that are sufficient for exact learning, *J. Comput. Syst. Sci.* **52**(3) (1996), 421-433.
10. N. Bshouty and J. Jackson. Learning DNF over the uniform distribution using a quantum example oracle, *SIAM J. Comp.* **28**(3) (1999), 1136-1153.
11. H. Buhrman, R. Cleve and A. Wigderson. Quantum vs. classical communication and computation, *in* "Proc. 30th Symp. on Theory of Comp." (1998), 63-68.
12. R. Cleve. An introduction to quantum complexity theory, *to appear in* "Collected Papers on Quantum Computation and Quantum Information Theory," ed. by C. Macchiavello, G.M. Palma and A. Zeilinger.
13. D. Deutsch and R. Jozsa. Rapid solution of problems by quantum computation, *Proc. Royal Society of London A*, **439** (1992), 553-558.
14. R. Gavaldà. The complexity of learning with queries, *in* "Proc. Ninth Structure in Complexity Theory Conference" (1994), 324-337.
15. O. Goldreich, S. Goldwasser and S. Micali. How to construct random functions, *J. ACM* **33**(4) (1986), 792-807.
16. L. K. Grover. A fast quantum mechanical algorithm for database search, *in* "Proc. 28th Symp. on Theory of Comp." (1996), 212-219.
17. J. Håstad, R. Impagliazzo, L. Levin and M. Luby. A pseudorandom generator from any one-way function, *SIAM J. Comp.* **28**(4) (1999), 1364-1396.
18. T. Hegedűs. Generalized teaching dimensions and the query complexity of learning, *in* "Proc. Eighth Conf. on Comp. Learning Theory," (195), 108-117.
19. L. Hellerstein, K. Pillaipakkamnatt, V. Raghavan and D. Wilkins. How many queries are needed to learn? *J. ACM* **43**(5) (1996), 840-862.
20. M. Kearns and L. Valiant. Cryptographic limitations on learning boolean formulae and finite automata, *J. ACM* **41**(1) (1994), 67-95.

21. M. Luby and C. Rackoff. How to construct pseudorandom permutations from pseudrandom functions, *SIAM J. Comp.* **17**(2) (1988), 373-386.
22. J. Patarin. How to construct pseudorandom and super pseudorandom permutations from one single pseudorandom function, *in* "Adv. in Crypt. – EUROCRYPT '92" (1992), 256-266.
23. J. Pierpzyk. How to construct pseudorandom permutations from single pseudorandom functions. *in* "Adv. in Crypt. – EUROCRYPT '90" (1990), 140-150.
24. J. Pierpzyk and B. Sadeghiyan. A construction for super pseudorandom permutations from a single pseudorandom function. *in* "Adv. in Crypt. – EUROCRYPT '92" (1992), 267-284.
25. J. Preskill. Lecture notes on quantum computation (1998). Available at `http://www.theory.caltech.edu/people/preskill/ph229/`
26. R. Servedio and S. Gortler. Quantum versus classical learnability, *to appear in* "Proc. 16th Conf. on Comput. Complex." (2001).
27. P. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, *SIAM J. Comp.* **26**(5) (1997), 1484-1509.
28. D. Simon. On the power of quantum computation, *SIAM J. Comp.* **26**(5) (1997), 1474-1483.
29. A. Yao. Theory and applications of trapdoor functions, *in* "Proc. 23rd FOCS" (1982), 80-91.
30. A. Yao. Quantum circuit complexity, *in* "Proc. 34th FOCS" (1993), 352-361.
31. C. Zalka. Grover's quantum searching algorithm is optimal. *Physical Review A* **60** (1999), 2746–2751.

A Proof of Theorem 3

We say that a polynomial-time deterministic algorithm $G : \{0,1\}^n \rightarrow \{0,1\}^{2n}$ is a *pseudorandom generator* if for all polynomials Q, all p.p.t. algorithms A, and all sufficiently large n, $\left|\Pr_{z\in\{0,1\}^n}[A(G(z)) \text{ outputs } 1] - \Pr_{z\in\{0,1\}^{2n}}[A(z) \text{ outputs } 1]\right| < \frac{1}{Q(n)}$. Thus a pseudorandom generator is an efficient algorithm which converts an n-bit random string into a $2n$-bit string which "looks random" to any polynomial-time algorithm. Håstad et al. [17] have shown that pseudorandom generators exist if any one-way function exists.

For G a pseudorandom generator and $s \in \{0,1\}^n$ we write $G_0(s)$ to denote the first n bits of $G(s)$ and $G_1(s)$ to denote the last n bits of $G(s)$. For $x, s \in \{0,1\}^n$ let $f_s : \{0,1\}^n \rightarrow \{0,1\}^n$ be defined as $f_s(x) \stackrel{\text{def}}{=} G_{x_n}(G_{x_{n-1}}(\cdots(G_{x_2}(G_{x_1}(s)))\cdots))$. In [15] it is shown that $\{f_s\}$ is a pseudorandom function family. We now show that the family $\{g_s\}$ defined by $g_s(x) \stackrel{\text{def}}{=} f_s(x)\oplus f_s(x\oplus s)$ is pseudorandom.

Let $\mathcal{F}'_n$ be the following probability distribution over functions from $\{0,1\}^n$ to $\{0,1\}^n$: a function F' is drawn from $\mathcal{F}'_n$ by drawing a random function F from $\mathcal{F}_n$, drawing a random string $s \in \{0,1\}^n$, and letting F' be the function defined as $F'(x) = F(x)\oplus F(x\oplus s)$. Theorem 3 follows from the following two lemmas:

Lemma 9. *For all polynomials Q, all p.p.t. oracle algorithms M, and all sufficiently large n,* $\left|\Pr_{F\in\mathcal{F}_n}[M^F \textit{ outputs } 1] - \Pr_{F'\in\mathcal{F}'_n}[M^{F'} \textit{ outputs } 1]\right| < \frac{1}{Q(n)}$.

Proof. Consider an execution of M with an oracle $F' \in \mathcal{F}'_n$ defined by $F'(x) = F(x)\oplus F(x\oplus s)$. Let $S = \{x^1, \ldots, x^t\} \subset \{0,1\}^n$ be the set of strings which M uses as queries to F'. We say that M *finds* s if $x^i = x^j\oplus s$ for some $x^i, x^j \in S$. If M does not find s, then the distribution of answers which M receives from F' is identical to the distribution which M would receive if it were querying a random function $F \in \mathcal{F}_n$, since in both cases each distinct query is answered with a uniformly distributed n-bit string. Thus the left side of the inequality above is at most $\Pr[M \text{ finds } s]$. A simple inductive argument given in the proof of Theorem 3.3 of [28] shows that this probability is at most $\sum_{k=1}^{t}(k/(2^n - (k-2)(k-1)/2))$. Since M is polynomial-time, t is at most $\text{poly}(n)$ and the lemma follows. ◻

Lemma 10. *For all polynomials Q, all p.p.t. oracle algorithms A, and all sufficiently large n, $\left|\Pr_{F' \in \mathcal{F}'_n}[M^{F'} \textit{ outputs } 1] - \Pr_{s \in \{0,1\}^n}[M^{g_s} \textit{ outputs } 1]\right| < \frac{1}{Q(n)}$.*

Proof. We require the following fact which is due to Yao [29]:

Fact 2 *Let G be a pseudorandom generator, let $q(n)$ and $Q(n)$ be polynomials, and let M_* be a p.p.t. algorithm which takes as input $q(n)$ strings each of length $2n$ bits. Then for all sufficiently large n we have $|p_n^G - p_n^U| < \frac{1}{Q(n)}$, where p_n^U is the probability that M_* outputs 1 on input $q(n)$ random strings in $\{0,1\}^{2n}$ and p_n^G is the probability that M_* outputs 1 on input $q(n)$ strings each of which is obtained by applying G to a random string from $\{0,1\}^n$.*

We prove Lemma 10 by contradiction; so suppose that there exists a p.p.t. oracle algorithm M and a polynomial Q such that for infinitely many values of n,

$$\left|\Pr_{F' \in \mathcal{F}'_n}[M^{F'} \text{ outputs } 1] - \Pr_{s \in \{0,1\}^n}[M^{g_s} \text{ outputs } 1]\right| \geq \frac{1}{Q(n)}. \quad (3)$$

We will show that there is a p.p.t. algorithm M_* which contradicts Fact 2.

As in the proof in [15] that $\{f_s\}$ is a pseudorandom function family, we use a so-called "hybrid" argument. Consider the following algorithms A_i ($i = 0, 1, \ldots, n$), each of which defines a mapping from $\{0,1\}^n$ to $\{0,1\}^n$ and hence could conceivably be used as an oracle to answer M's queries. Conceptually, each algorithm A_i contains a full binary tree of depth n in which the root (at depth 0) is labeled with a random n-bit string s; if $i > 0$ then each node at depth i is also labeled with an independently chosen random n-bit string. Each node at depth $j > i$ also has an n-bit label determined as follows: if node v has label z then the left child of v has label $G_0(z)$ and the right child of v has label $G_1(z)$. Each node in the tree has an *address* which is a binary string: the root's address is the empty string, and if a node has address $\alpha \in \{0,1\}^*$ then its left child has address $\alpha 0$ and its right child has address $\alpha 1$ (so each leaf has a different n-bit string as its address). Let $L(x)$ denote the label of the node whose address is x. Algorithm A_i answers a query $x \in \{0,1\}^n$ with the n-bit string $L(x) \oplus L(x \oplus s)$.

(Note that algorithm A_i need not precompute any leaf labels. Instead, A_i can run in $\text{poly}(n)$ time at each invocation by randomly choosing s once and for all the first time it is invoked and labeling the necessary portion of the tree "on the fly" at each invocation by choosing random strings for the depth-i nodes as required and computing descendents' labels as described above. A_i must store the random strings which it uses to label depth-i nodes so as to maintain consistency over successive invocations.)

For $i = 0, 1, \ldots, n$ let $p_n^i = \Pr[M^{A_i} \text{ outputs } 1]$, i.e. the probability that M outputs 1 if its oracle queries are answered by algorithm A_i. Let $p_n^g = \Pr_{s \in \{0,1\}^n}[M^{g_s} \text{ outputs } 1]$

and $p_n^{F'} = \Pr_{F' \in \mathcal{F}'_n}[M^{F'}$ outputs 1]. We have that $p_n^0 = p_n^g$ since algorithm A_0 behaves exactly like an oracle for g_s where s is a random n-bit string. We also have that $p_n^n = p_n^{F'}$ since algorithm A_n behaves exactly like an oracle for $F' \in \mathcal{F}'_n$. Inequality (3) thus implies that $|p_n^0 - p_n^n| \geq 1/Q(n)$ for infinitely many values of n.

Now we describe the algorithm M_* which distinguishes between sets of strings. Let $q(n)$ be a polynomial which bounds the running time of M on inputs of length n (so M makes at most $q(n)$ oracle queries given access to an oracle from $\{0,1\}^n$ to $\{0,1\}^n$). The algorithm M_* takes as input a set U_n of $2q(n)$ strings of length $2n$. M_* works by first selecting a uniform random value $0 \leq i \leq n-1$ and a uniform random string $s \in \{0,1\}^n$. M_* then runs algorithm M, answering M's oracle queries as follows (there are two cases depending on whether or not the prefix $s_1 \dots s_i$ is 0^i):

– **Case 1: $s_1 \dots s_i \neq 0^i$.** Let $x = x_1 \dots x_n$ be the query string. If no earlier query string had prefix $x_1 \dots x_i$ or $(x_1 \oplus s_1) \dots (x_i \oplus s_i)$, then M_* takes the next two $2n$-bit strings from U_n; call these strings $u^1 = u_0^1 u_1^1$ and $u^2 = u_0^2 u_1^2$ where $|u_j^i| = n$. M_* stores the four pairs $(x_1 \dots x_i 0, u_0^1)$, $(x_1 \dots x_i 1, u_1^1)$, $((x_1 \oplus s_1) \dots (x_i \oplus s_i)0, u_0^2)$, $((x_1 \oplus s_1) \dots (x_i \oplus s_i)1, u_1^2)$ and answers with the string

$$G_{x_n}(G_{x_{n-1}}(\dots G_{x_{i+2}}(u^1_{x_{i+1}})\dots)) \oplus$$
$$G_{x_n \oplus s_n}(G_{x_{n-1} \oplus s_{n-1}}(\dots G_{x_{i+2} \oplus s_{i+2}}(u^2_{x_{i+1} \oplus s_{i+1}})\dots)). \qquad (*)$$

Otherwise, if an earlier query string had prefix $x_1 \dots x_i$ or $(x_1 \oplus s_1) \dots (x_i \oplus s_i)$, then instead M_* retrieves the two previously stored pairs $(x_1 \dots x_i x_{i+1}, u^1_{x_{i+1}})$ and $((x_1 \oplus s_1) \dots (x_i \oplus s_i)(x_{i+1} \oplus s_{i+1}), u^2_{x_{i+1} \oplus s_{i+1}})$ and answers with $(*)$ as above.

– **Case 2: $s_1 \dots s_i = 0^i$.** Let $x = x_1 \dots x_n$ be the query string. If no earlier query string had prefix $x_1 \dots x_i$, then M_* takes the next $2n$-bit string from U_n; call this string $u = u_0 u_1$ where $|u_0| = |u_1| = n$. M_* stores the two pairs $(x_1 \dots x_i 0, u_0)$, $(x_1 \dots x_i 1, u_1)$ and answers with

$$G_{x_n}(G_{x_{n-1}}(\dots G_{x_{i+2}}(u_{x_{i+1}})\dots)) \oplus$$
$$G_{x_n \oplus s_n}(G_{x_{n-1} \oplus s_{n-1}}(\dots G_{x_{i+2} \oplus s_{i+2}}(u_{x_{i+1} \oplus s_{i+1}})\dots)). \qquad (**)$$

Otherwise, if an earlier query string had prefix $x_1 \dots x_i$, then M_* retrieves the two pairs $(x_1 \dots x_i x_{i+1}, u_{x_{i+1}})$ and $(x_1 \dots x_i (x_{i+1} \oplus s_{i+1}), u_{x_{i+1} \oplus s_{i+1}})$ (these two pairs are the same if $s_{i+1} = 0$) and answers with $(**)$ as above.

The crucial properties of algorithm M_*, which are straightforwardly verified, are the following: If each string in U_n is generated by applying G to a random n-bit string (scenario 1), then M_* simulates a computation of M with oracle A_i. On the other hand, if each string in U_n is chosen uniformly from $\{0,1\}^{2n}$ (scenario 2), then M_* simulates a computation of M with oracle A_{i+1}.

It is easy to see now that in scenario 1 we have $\Pr[M_*$ outputs 1$] = \sum_{i=0}^{n-1} p_n^i / n$ while in scenario 2 we have $\Pr[M_*$ outputs 1$] = \sum_{i=1}^{n} p_n^{i+1}/n$. These two probabilities differ by $(1/n) \cdot |p_n^0 - p_n^n|$, which is at least $1/nQ(n)$ for infinitely many values of n. Now by Fact 2 the existence of M_* contradicts the fact that G is a pseudorandom generator, and the lemma is proved. □