Learning to Impersonate

Moni Naor* Guy N. Rothblum†

Abstract

Consider Alice, who is interacting with Bob. Alice and Bob have some shared secret which helps Alice identify Bob-impersonators. Now consider Eve, who knows Alice and Bob, but does not know their shared secret. Eve would like to impersonate Bob and "fool" Alice without knowing the secret. If Eve is computationally unbounded, how long does she need to observe Alice and Bob interacting before she can successfully impersonate Bob? What is a good strategy for Eve in this setting? If Eve runs in polynomial time, and if there exists a one-way function, then it is not hard to see that Alice and Bob may be "safe" from impersonators, but is the existence of one-way functions an essential condition? Namely, if one-way functions do not exist, can an efficient Eve always impersonate Bob?

In this work we consider these natural questions from the point of view of Ever, who is trying to observe Bob and learn to impersonate him. We formalize this setting in a new computational learning model of *learning adaptively changing distributions* (ACDs), which we believe captures a wide variety of natural learning tasks and is of interest from both cryptographic and computational learning points of view. We present a learning algorithm that Eve can use to successfully learn to impersonate Bob in the information-theoretic setting. We also show that in the computational setting an efficient Eve can learn to impersonate any efficient Bob if and only if one-way function do not exist.

1 Introduction

Consider the T-1000 robot in the movie "Terminator 2: Judgement Day". The robot can "imitate anything it touches... anything it samples," which, if it were possible, would be an astonishing computational learning achievement. The goal of the robot is to perfectly impersonate objects, but the objects it is learning to impersonate (namely people) are highly complex, with probabilistic behavior that changes adaptively in different settings. In this work we consider this type of computational learning: we formalize a model for learning adaptively changing distributions (ACDs), which is relevant both to the field of computational learning theory and to the field of cryptography.

The challenges of learning adaptively changing distributions are illustrated by the impersonation example outlined in the abstract: Alice and Bob generate a random shared secret key. This secret key is used (repeatedly) to protect Alice from *Bob-impersonators*: it should be impossible for any third party that does not have the secret key to imitate Bob's behavior (in Alice's eyes). Unfortunately for Alice and Bob, Eve is eavesdropping and observing Alice and Bob's interaction.

*Incumbent of the Judith Kleeman Professorial Chair, Dept. of Computer Science and Applied Math, Weizmann Institute of Science, Rehovot 76100, Israel. E-mail: `moni.naor@weizmann.ac.il`. Research supported in part by a grant from the Israel Science Foundation.

†CSAIL, MIT, Cambridge MA, USA. E-mail: `rothblum@csail.mit.edu`. This work was done while the author was at the Weizmann Institute of Science, Rehovot, Israel.

Eve does not know the secret key, but would like to *learn* to fool Alice by impersonating Bob. How long would it take for Eve to learn to generate outputs that are indistinguishable (to Alice) from Bob's? What is Eve's best strategy in this setting? These are fundamental questions in cryptography, since if Eve can perfectly imitate Bob any meaningful notion of authentication or identification between Alice and Bob is lost. Note that we make no restrictions on Bob's behavior, which may be affected by interaction, private random coins, his past outputs and even by the environment.

Essentially, Eve wants to learn to imitate the behavior of a stochastic object whose behavior changes adaptively. We measure learning by the ability to predict or emulate the *distribution* of the object's next output. In this work we explore this new model of learning *adaptively changing distributions* (ACDs): distributions that can change adaptively and arbitrarily. Our model for learning is inspired by the work of Kearns *et al.* [20] on learning *static* distribution, and by the PAC-Learning model of Valiant [32]. We believe this is a very natural computational learning model, as was evidenced by its recent use in Naor and Rothblum [25], the companion work to this paper. The goal in [25] was to show that if a cryptographic object (an authenticator or online memory checker) uses a cryptographic secret to generate public behavior, then after observing the cryptographic object (not for too long), an adversary can predict the distribution of its behavior. This is an essential ingredient for showing the cryptographic lower bound of [25].

In this paper we initiate the study of learning adaptively changing distributions. We define the new model, and show polynomially tight bounds for the sample complexity of learning in the information theoretic setting. We show that information-theoretically Eve can always learn to impersonate Bob after observing him for a reasonable number of interactions with Alice. We note that the information-theoretic task of learning ACDs is far from trivial, and we believe the analysis of the learning algorithm is technically interesting, especially in its incorporation of the mean value theorem in a setting of computational learning. From a cryptographic point of view, our study of the interplay between the amount of secret entropy shared by the parties and the ability of an unbounded adversary to impersonate one of the parties is a continuation of a similar study initiated for encryption systems by Shannon [30] in his seminal paper on the theory of secrecy systems. Complexity-based cryptography achieves the information-theoretically impossible by assuming that adversaries are computationally bounded, and that some tasks (e.g. inverting a one-way function) are hard for the adversary. In the impersonation setting, it is not hard to see that if one-way functions exist then a computationally bounded adversary cannot learn to impersonate all Bobs. In this work we show that an efficient adversary's ability to impersonate Bob is tightly tied to the existence of one-way functions: if one-way functions do not exist then Eve can *efficiently* learn to impersonate any (efficient) Bob. Showing such tight equivalence between cryptographic tasks is important because it gives a good characterization of the task's inherent cryptographic difficulty. Furthermore, this result is used in [25] to show that one-way functions are essential for good online memory checking and sublinear authentication. We proceed by formally introducing the model of learning adaptively changing distributions and formally stating our new results. A comparison with previous work follows.

1.1 The Model

Adaptively Changing Distributions We are interested in learning distributions that change adaptively over time. An adaptively changing distribution (ACD) is a process that defines a (randomized) mapping between states that are composed of two parts: a *public* state p and a *secret* state s. In the impersonation example, the ACD is Bob (or the algorithm that Bob runs), the public state is the public interaction between Alice and Bob, and the secret state is initially Alice and Bob's shared secret. An ACD is activated on such a pair of states, and each activation generates a new pair of states, thus when Bob runs on the current public and secret states he outputs a new public state (some new communication for Alice, appended to the history of communication), and a new secret state (which would usually include the initial shared secret). We consider the new public state to be the output of the process, and for simplicity restrict ourselves to a setting in which both states do not change between activations of the ACD (i.e. only Bob generates messages on the public channel). The set of possible public states is denoted by S_p and the set of secret states is denoted by S_s. We do not restrict the sizes of the sets of states.

The new state is always a function of the current state and some randomness, which we assume is taken from a set R. Thus the states generated by an ACD are determined by a function $\mathcal{D}$:

$$\mathcal{D} : S_p \times S_s \times R \mapsto S_p \times S_s$$

When $\mathcal{D}$ (in the example, $\mathcal{D}$ is the algorithm that Bob runs) is activated for the first time, it is run on some initial public and secret states p_0 and s_0 respectively (the process by which these initial states are generated is of great importance in our learning model, and will be described and motivated below). The public output of the process is a sequence of public states $(p_1, p_2, \ldots)$.

We would like to examine the task of "learning" the distribution of the next public state, with a learning algorithm that knows $\mathcal{D}$ but only has access to the sequence $(p_0, p_1, p_2, \ldots)$ of public states. In the impersonation example the learning algorithm corresponds to Eve's role (Eve sees only the public communication between Alice and Bob). We compare the performance of such a learning algorithm to that of an observer (Alice) who knows not only the public outputs, but also the *initial* secret state s_0. Note that the observer to which we compare the learning algorithm does not know any other secret states, nor does it know anything about the randomness used by $\mathcal{D}$ in each step (in the example, Alice only sees the initial shared secret and messages sent by Bob, she never sees the random coins Bob flips). Thus the learning algorithm's goal is learning to approximate the distribution of the next public output of $\mathcal{D}$, where the randomness for the distribution is over all past activations of $\mathcal{D}$, given the past public states and the initial secret state. This correspond to Eve's goal in the example: Eve would like to approximate the distribution of Bob's next output in a way that will "fool" Alice. Note that this distribution is indeed adaptive, and may change as $\mathcal{D}$ is activated.

The Initial State Recall that when $\mathcal{D}$ is activated for the first time, it is activated on *initial* secret and public states s_0 and p_0. The learning algorithm activates $\mathcal{D}$ consecutively, trying to learn the distribution of the public state after $\mathcal{D}$'s next activation (recall the distribution is taken given all the past public information and the initial secret state), even though s_0 isn't known to it.

The initial state is selected by a distribution $\mathcal{G}$, in the example this would be the secret-key-generating algorithm that Alice and Bob use. We refer to the set of possible initial secret states as S_{init} (this is the "concept class"). The generating distribution $\mathcal{G}$ generates the initial secret state, outputting an initial public state $p_0 \in S_p$ and an initial secret state $s_0 \in S_{init}$. We can view $\mathcal{G}$ as a function of its randomness:

$$\mathcal{G} : R \mapsto S_p \times S_{init}$$

Definition 1.1. *Adaptively Changing Distribution*

An adaptively changing distribution (ACD) is a pair of probabilistic algorithms for generation ($\mathcal{G}$) and for sampling ($\mathcal{D}$). The generation algorithm $\mathcal{G}$ outputs an initial public state and an initial secret state in S_{init}. The sampling algorithm $\mathcal{D}$ receives public and private states, and generates new (public and private) states.

Learning ACDs Given algorithms $\mathcal{G}$ and $\mathcal{D}$ for generation and sampling, a learning algorithm $\mathcal{L}$ for the ACD $(\mathcal{G}, \mathcal{D})$ is a probabilistic algorithm that observes $\mathcal{G}$ and $\mathcal{D}$'s public outputs as they run. We always assume that the learning algorithm "knows" a description of $\mathcal{G}$ and $\mathcal{D}$ (for any possible input to the algorithms, $\mathcal{L}$ can compute their output). A learning process begins when $\mathcal{G}$ is activated, generating the initial public and secret states (p_0 and s_0). The learning algorithm is then activated, and receives the initial public state p_0 (the learning algorithm does not know s_0!).

Throughout the learning process, the sampling algorithm $\mathcal{D}$ is run consecutively, changing the public (and secret) state. Let p_i and s_i be the public secret states after $\mathcal{D}$'s i-th activation. We often refer to **the distribution** $D_i^s(p_0, \ldots, p_i)$. This is a (conditional) distribution over the public state that will be generated by $\mathcal{D}$'s next ($i + 1$-th) activation.

Definition 1.2. *The Distribution* D_i^s

The distribution $D_i^s(p_0, \ldots, p_i)$ is the distribution of the next public state that $\mathcal{D}$ will output (in the $i + 1$-th time it runs), given that past public outputs were $(p_0, \ldots, p_i)$, and given also that **the initial secret state that $\mathcal{G}$ computed was s**. In the impersonation example, this is the distribution Alice expects of Bob's in round $i + 1$, given that their shared secret was s and given the previous messages sent by Bob were $(p_1, \ldots, p_{i+1})$.

For any initial secret state $s \in S_{init}$, past public states $(p_0, \ldots, p_i)$, and (new) public state $p_{i+1} \in S_p$: $D_i^s(p_0, \ldots, p_i)[p_{i+1}]$ is the probability, conditioned on the public information $(p_0, \ldots, p_i)$ and on the fact that the initial secret state $\mathcal{G}$ computed was s, that in $\mathcal{D}$'s $(i+1)$-th activation, the public output will be p_{i+1}.

Note that (computational issues aside) the distribution $D_i^s(p_0, \ldots, p_i)$ can be computed by enumerating over all random strings (for $\mathcal{G}$ and $i + 1$ activations of $\mathcal{D}$), for which the algorithm $\mathcal{G}$ outputs (p_0, s) and then $\mathcal{D}$ outputs $(p_1, \ldots, p_i)$. The probability that $\mathcal{D}$ generates the new public state p_{i+1} is the fraction of these random strings for which $\mathcal{D}$'s $i + 1$-th output is p_{i+1}.

After letting $\mathcal{D}$ run for k steps, $\mathcal{L}$ should stop and output some hypothesis h as to the distribution $D_{k+1}^{s_0}(p_0, \ldots, p_k)$: the distribution of the next public state computed by $\mathcal{D}$. It is important to note that $\mathcal{L}$ sees only $(p_0, p_1 \ldots p_k)$, the secret states $(s_0, s_1 \ldots s_k)$ and the random coins used by $\mathcal{D}$ are kept hidden from it. The number of times $\mathcal{D}$ is allowed to run (k) is determined by the learning algorithm. We will be interested in bounding this number of "rounds".

Definition 1.3. *Learning Process*

A learning process with learning algorithm $\mathcal{L}$ for learning an ACD $(\mathcal{G}, \mathcal{D})$ on input length n, is run in the following manner:

1. The generating algorithm $\mathcal{G}$ is run on input 1^n, generating initial secret and public states s_0 and p_0.
2. The learning algorithm $\mathcal{L}$ receives the initial public state p_0.
3. The learning process proceeds in rounds. In each round i, $\mathcal{L}$ is given the new public state and may choose whether to proceed to the next round (and get the next public state) or

output a hypothesis h as to the distribution of the public state after $\mathcal{D}$'s next activation (the distribution $D^{s_0}_{k+1}(p_0, \ldots, p_i)$). The learning process halts when $\mathcal{L}$ outputs this hypothesis. If $\mathcal{L}$ chooses not to output a hypothesis, then $\mathcal{D}$ is activated on the current public and secret states, generating a new state and proceeding to the next round.

Definition 1.4. *Learning Algorithm*
Let $\mathcal{L}$ be an algorithm for learning an ACD $(\mathcal{G}, \mathcal{D})$ on input length n. We say that $\mathcal{L}$ is an (ε, δ)-learning algorithm for $(\mathcal{G}, \mathcal{D})$, if when run in a learning process for $(\mathcal{G}, \mathcal{D})$, $\mathcal{L}$ always halts and outputs some hypothesis h that specifies a hypothesis distribution D_h of the public state after $\mathcal{D}$'s next activation. We require that with high probability the hypothesis distribution is *statistically close* to the real distribution of the next public state.

Namely, with probability at least $1 - \delta$ (over the coin tosses of $\mathcal{D}, \mathcal{G}$, and $\mathcal{L}$), $\Delta(D^{s_0}_{k+1}, D_h) \leq \varepsilon$, where Δ is the statistical distance between the distributions (see Section 2).

The hypothesis h can be used to generate a distribution that is close to the distribution of $\mathcal{D}$'s next public output. In the terminology of [20] we focus on learning by generation, though in the information-theoretic setting learning by generation and evaluation are equivalent.

The main complexity measure for an ACD learning algorithm is the maximal number of rounds it lets $\mathcal{D}$ run before outputting the hypothesis h (the *sample complexity*). Naturally, we expect that the higher ε and δ are, and the larger the concept class S_{init}, the higher the sample complexity for learning the ACD will be.

Our goal in this work is to study under what conditions ACDs can be learned, and to quantify the number of rounds required for learning ACDs. We present an algorithm for learning any ACD. Unfortunately, this "generic" algorithm may need exponential time even if the generation and sampling algorithms are efficient (i.e. polynomial time). We initiate a study of the computational complexity of learning ACDs, especially the conditions under which learning ACDs efficiently (in polynomial time) is always possible.

To study the complexity of learning ACDs (sample complexity or computational complexity), we consider the task of learning ACDs for many input lengths. Towards this end we consider an ACD as an *ensemble*, or a family of ACDs, one for each input length. We will assume the input length parameter 1^n is given as input to the generating algorithm $\mathcal{G}$[1]. The input length parameter is always known to the learning algorithm $\mathcal{L}$, and thus (w.l.o.g) we assume it is part of the initial public state p_0 that $\mathcal{G}$ outputs. We require that the generation and sampling algorithms, as well as the learning algorithm itself, work for different input lengths. Note that the generation and sampling algorithms may be either uniform or non-uniform. In the latter case, the learning algorithm will also be non-uniform. Throughout this work, when we refer to ACDs or learning ACDs, we always implicitly refer to ensembles of ACDs.

Definition 1.5. *Learning ACDs*
Let $\mathcal{L}$ be an algorithm for learning an ACD $(\mathcal{G}, \mathcal{D})$. We say that $\mathcal{L}$ is an $(\varepsilon(n), \delta(n))$-learning algorithm for $(\mathcal{G}, \mathcal{D})$, if there exists some n_0 such that for any input length $n \geq n_0$, the algorithm $\mathcal{L}$ is a $(\varepsilon(n), \delta(n))$-learning algorithm for $(\mathcal{G}, \mathcal{D})$ on input length n.

[1]Throughout this work will refer to $\mathcal{G}$'s input as always being 1^n, though our results hold even if $\mathcal{G}$'s input is some auxiliary input string in $\{0,1\}^n$ (assuming this auxiliary input to $\mathcal{G}$ is known to the learning algorithm).

1.2 Our Results

In Section 3 we present an algorithm for learning ACDs and analyze its sample complexity (without limiting its running time). The learning algorithm checks whether almost all of the initial secret states induce statistically close distributions. If they do, then the algorithm has a good idea of the distribution of the next public output. If not, then we show that the algorithm can expect to learn something about the initial secret state. This is a very natural observation: *"If I can't simulate what is about to happen, then I can expect to learn something by observing it"* (this is the converse to the observation *"if I can simulate what is going to happen, then I learn nothing from observing it"*, which is the basis of the definition of zero knowledge protocols in cryptography, see Goldwasser, Micali and Rackoff [13]).

More formally, we examine the initial secret state as a random variable, and show that unless almost all initial secret states induce close distributions, after $\mathcal{D}$ generates a new public state there is a large *expected* drop in the Shannon Entropy of the initial secret state. The learning algorithm is guaranteed to learn an ε-close distribution with probability at least $1-\delta$ after at most $O(\frac{\log|S_{init}|}{\varepsilon^2 \cdot \delta^2})$ rounds ("samples"). In terms of its running time the learning algorithm may require exponential time even if the ACD's algorithms $\mathcal{G}$ and $\mathcal{D}$ run in probabilistic polynomial time. In Appendix A we present a lower bound, proving that learning ACDs requires sample complexity that is at least linear in $\log|S_{init}|$, in $\frac{1}{\varepsilon}$ and in $\frac{1}{\delta}$. Specifically, we prove that learning ACDs requires activating $\mathcal{D}$ at least $\Omega(\frac{1}{\delta} + \frac{\log|S_{init}|}{\varepsilon})$ times. In particular, this implies that algorithms for learning ACDs cannot be amplified efficiently (as opposed to, for example, PAC learning algorithms). We note that the algorithm we present is *nearly optimal* in the sense that no other algorithm for learning ACDs can achieve better (up to constant factors) performance (more on this in Section 3), the polynomial gap between our upper and lower bound only indicates that perhaps our analysis is not optimal.

In fact in all our upper bounds the $\log|S_{init}|$ factor can be improved to $H(s_0)$, where $H(s_0)$ is the Shannon entropy of the initial secret state s_0 that $\mathcal{G}$ outputs, given p_0. Additionally, an alternative (stronger) statement of the result would be that the algorithm can generate an ε-close distribution in all but at most $O(\frac{\log|S_{init}|}{\varepsilon^2 \cdot \delta^2})$ rounds. We note that all the results hold even if the public state is not guaranteed to be *persistent*, and may change between activations of $\mathcal{D}$. For example, this could be the case in the impersonation scenario, if Alice issued "challenges" to Bob: Alice's "challenges" would be added to the public state between activations of Bob.

In Section 4 we investigate the computational power needed to learn ACDs that are constructible in probabilistic polynomial time. We first observe that if one-way functions (see Section 2 for standard cryptographic definitions) exist, then there exist ACDs constructible in polynomial time that cannot be learned in polynomial time. We show that one-way functions are also an *essential* condition. One-way functions (that are hard to invert for infinitely many input lengths) exist if and only if there exists a polynomial time ACD that cannot be learned by any polynomial time algorithm that activates $\mathcal{D}$ at most $O(\frac{\log|S_{init}|}{\delta^2 \cdot \varepsilon^2})$ times. One example for an ACD that is constructible in polynomial time is any probabilistic automata with a polynomial number of states.

To show this result we improve a fundamental result of Goldreich [9]: he showed that one-way functions exist if and only if there exist a pair of polynomial-time constructible distributions (ensembles) that are statistically far but computationally indistinguishable. This result does not, however, specify how large a gap between the statistical and computational distinguishabilities implies the existence of one-way functions (Goldreich only shows that some polynomial gap suffices). In our setting the question becomes cardinal, and we prove an essentially tight result: one-way functions exist if and only if there exist a pair of polynomial-time constructible distributions (ensembles) with statistical distance ε, such that no polynomial-time algorithm can distinguish between them with advantage that is close to ε.

1.3 A Discussion of The Model

We turn to discussing the model of learning adaptively changing distributions, and especially the motivation for our definition of learnability in this model. We would like to argue that several (seemingly innocuous) modifications to the definition of learning ACDs would make the learning task too hard or impossible.

In this work, we present an algorithm for learning any ACD. The algorithm will work for any ACD $(\mathcal{G}, \mathcal{D})$, as long as it knows $\mathcal{G}$ and $\mathcal{D}$. Note that unlike other models of learning (e.g. the PAC-learning model), even in the information-theoretic setting, we can never hope to guarantee that after letting the algorithm run for a set number i of rounds, it always outputs a good hypothesis. This is since (for example), for any fixed i, the ACD may output some fixed distribution up to round i, and only then begin to use its secret state! An algorithm for learning ACDs must use its knowledge of the ACD $(\mathcal{G}, \mathcal{D})$ in order to decide, on-the-fly, at what round it outputs its hypothesis.

In our model for learning ACDs we compare the performance of the learning algorithm to that of an observer who knows the *initial* secret state. It is natural to ask whether one could design a learning algorithm that gains as good an idea about the distribution of the next public state as an observer who knows *all* the secret states, however this turns out to be impossible. No algorithm for learning ACDs can hope to learn the distribution of the public state generated by the ACD's next activation given all the the secret states. To see this, consider for example that the distribution generated in the next activation may be determined by a random string that was flipped in the ACD's last activation and stored in the secret state. In this scenario, no learning algorithm (that does not see the last secret state) can hope to learn the distribution of the ACD's next public output. This is the reason that we settle for learning the distribution where the randomness is taken over *all previous activations* of $\mathcal{D}$, given all past public outcomes. Also, we cannot hope for *exact* learning, since two different initial secret states may behave extremely similarly: in this scenario no learning algorithm could exactly learn the distribution of the next public state with high probability. Thus, we settle for approximate learning.

1.4 Related Work and its Relationship

Computational Learning Kearns *et al.* [20] were the first to explore efficient approximate learning of *static* distributions, inspired by the PAC-learning model of Valiant [32]. They showed that if one-way functions exist, then there exist sets of distributions that are easy to construct but hard to learn. Thus they focused on presenting efficient learning algorithms for restricted classes of static distributions. Our work diverges in its focus on *adaptively changing* distributions.

Much attention has been given in the computational learning literature to learning finite-state automata, see Kearns and Vazirani [22]. Other work has considered learning probabilistic automata (PFAs) and especially hidden Markov models (HMMs). The focus of this research is learning the object generating samples (e.g. learning the transition probabilities of the HMM), and computing maximum likelihood estimates on unknown data. A tutorial on the theory of HMMs and applications to speech recognition is given by Rabiner [27]. Abe and Warmuth [1], and Gillman and Sipser [8] showed hardness of learning HMMs. Kearns *et al.* [20] presented hardness results for learning general PFAs. On the other hand, the Baum-Welch method [2, 3] is often used in practise for learning HMMs from a sample. Ron *et al.* [28] presented an algorithm for learning a restricted class of PFAs. While ACDs may bear a superficial resemblance to HMMs (both can be viewed as chains of secret and public states), the size of an ACDs secret state is not bounded, and thus each new public and secret sample generated by an ACD may in fact be a function of all previous secret and public states. In HMMs the size of the secret state is bounded and small, furthermore each secret states is only a function of the previous secret state.

Other work that is relevant in the field of computational learning theory includes the study of *drifting concepts* (see [15]). In the model of drifting concepts, a learning algorithm is trying to approximate a (standard) concept that is *slowly* changing. This is very different from the model of ACDs. ACDs are probability distributions (and not concepts), and no restriction is made on the amount of change in the distribution between samples. Kearns and Schapire [21] began an investigation of learning probabilistic concepts, where examples are not simply labelled as 0 or 1, but have a probability of being 0 or 1. Despite the fact that they are probabilistic, learning these (static) concepts is different from the task of learning distributions. Probabilistic concepts induce a probability distribution on the label of each example, whereas when dealing with distributions the learning algorithm only sees samples, and no “labelled examples” come into play.

Cryptographic Primitives An central question in cryptography is studying which assumptions are required to implement a cryptographic task or primitive. For most cryptographic tasks the answer is well understood: either one-way functions are essential and sufficient, or stronger assumptions are used. This study was initiated by Impagliazzo and Luby [18], who showed that implementing many central cryptographic tasks implies the existence of one-way functions. Proving that one-way functions are essential and necessary for implementing a cryptographic task provides a good characterization of the task’s inherent difficulty, and the possibility of actually implementing a solution (for a discussion see Impagliazzo [16]).

Impagliazzo and Levin [17] showed that efficient universal learning (in an “average case” model) is possible if and only if one-way functions do not exist. They show that universal learning becomes possible exactly when cryptography becomes impossible (and vice versa). In this work we show a similar result for learning adaptively changing distributions.

Identification and Authentication In the setting of secret-key cryptography, Shannon [30] defined perfect secrecy as statistical independence of the plaintext and ciphertext, and then showed that perfect secrecy requires that the entropy of the secret key be at least as large as the entropy of the plaintext. In the context of authentication and identification there is no clear definition of perfect security, since an adversary always has some chance of forging a message or impersonating one of the parties. A study on information-theoretic lower bounds for authentication theory was initiated by Simmons [31]. See Maurer [24] for a more generalized lower bound.

In the setting of identification, a corollary of our results on learning ACDs is that if Alice and Bob share a secret key with entropy $H(K)$, then even in a very general model where they can each maintain a very large state, interact and use the environment[2], a computationally unbounded Eve can (almost) perfectly impersonate Bob in all but $O(H(K))$ rounds of identification. Furthermore,

[2] This assumes, of course, that Eve knows everything that Alice does about the environment. This isn’t the case, for example, in the bounded storage model of cryptography.

we show that in a computational setting the existence of one-way functions is an *essential* (and sufficient) condition for breaking the $O(H(K))$ lower bound.

2 Definitions and Notations

2.1 Preliminaries

Notation We denote by $[n]$ the set $\{1, 2, \dots n\}$. For a vector $\vec{v}$, we refer to $\vec{v}$'s i-th coordinate as $\vec{v}_i$. For a (discrete) distribution D over a set X we denote by $x \sim D$ the experiment of selecting $x \in X$ by the distribution D. For $x \in X$, we denote by $D[x]$ the probability of x being selected by the distribution D. For a subset $A \subseteq X$ we denote by $D[A]$ the probability (or weight) of the set A by the distribution D (i.e. $D[A] = \sum_{x \in A} D[x]$).

Distributions, Ensembles and Entropy An ensemble $D = \{D_n\}_{n \in \mathbb{N}}$ is a sequence of random variables, each ranging over $\{0,1\}^{\ell(n)}$, we consider only ensembles where $\ell(n)$ is polynomial in n (we occasionally abuse notation and use D in place of D_n). An ensemble D is polynomial time constructible if there exists a probabilistic polynomial time Turing Machine (PPTM) $\mathcal{M}$ such that $D_n = \mathcal{M}(1^n)$.

We denote by $H(X)$ the (Shannon) entropy of a distribution X over $\{0,1\}^{\ell}$, as defined by Shannon in [29] (see also Cover and Thomas [6]).

$$H(X) = - \sum_{\alpha \in \{0,1\}^{\ell}} Pr[X = \alpha] \cdot \log(Pr[X = \alpha])$$

Definition 2.1. The *statistical distance* between two distributions X and Y over $\{0,1\}^{\ell}$, which we denote by $\Delta(X, Y)$, is defined as:

$$\Delta(X, Y) = \frac{1}{2} \sum_{\alpha \in \{0,1\}^{\ell}} |Pr[X = \alpha] - Pr[Y = \alpha]|$$

Definition 2.2. Two distributions X and Y are ε-statistically far if $\Delta(X, Y) \geq \varepsilon$. Otherwise X and Y are ε-statistically close.

Similarly, two ensembles D and F are $\varepsilon(n)$-statistically far if there exists some n_0 such that for all $n \geq n_0$ the distributions D_n and F_n are $\varepsilon(n)$-statistically far.

D and F are $\varepsilon(n)$-statistically close if there exists some n_0 such that for all $n \geq n_0$ the distributions D_n and F_n are $\varepsilon(n)$-statistically close. Note that ensembles that are not $\varepsilon(n)$-statistically far are not necessarily $\varepsilon(n)$-statistically close.

We say that two ensembles X and Y are statistically far if there exists some polynomial p such that X and Y are $\frac{1}{p(n)}$-statistically far.

Definition 2.3. *Computational Indistinguishability (Goldwasser Micali [12], Yao [33])* Two probability ensembles D and F are $\varepsilon(n)$-computationally indistinguishable if for any PPTM $\mathcal{M}$, that receives 1^n and one sample s from D_n or F_n and outputs 0 or 1, there exists some n_0 such that:

$$\forall n \geq n_0 : |Pr_{s \sim D_n}[\mathcal{M}(s) = 1] - Pr_{s \sim F_n}[\mathcal{M}(s) = 1]| < \varepsilon(n)$$

We say that two ensembles D and F are computationally indistinguishable if for any polynomial p, D and F are $\frac{1}{p(n)}$-computationally indistinguishable.

Definition 2.4. *False Entropy Ensemble (Håstad, Impagliazzo, Levin, Luby [14])* A polynomial-time constructible ensemble F is a false entropy ensemble if there exists a polynomial-time constructible ensemble D such that F and D are computationally indistinuguishable but D has higher entropy than F: there exists a polynomial p and $n_0 \in \mathbb{N}$ such that:

$$\forall n \geq n_0 : H(D_n) \geq H(F_n) + \frac{1}{p(n)}$$

One-way and distributionally one-way functions A function f, with input length $k(n)$ and output length $\ell(n)$, specifies for each $n \in \mathbb{N}$ a function $f_n : \{0,1\}^{k(n)} \mapsto \{0,1\}^{\ell(n)}$. We only consider functions with polynomial input lengths (in n), and occasionally abuse notation and use $f(x)$ rather than $f_n(x)$ for simplicity. The function f is computable in polynomial time (efficiently computable) if there exists a Turing Machine that for any $x \in \{0,1\}^{k(n)}$ outputs $f_n(x)$ and runs in time polynomial in n.

Definition 2.5. A function f is a *one-way function* if it is computable in polynomial time, and for any PPTM $\mathcal{A}$ the probability that $\mathcal{A}$ inverts f on a random input is negligible. Namely, for any polynomial p, there exists some n_0 such that:

$$\forall n \geq n_0 : Pr_{x \sim U_{k(n)}, \mathcal{M}}[\mathcal{M}(f(x)) \in f^{-1}(f(x))] < \frac{1}{p(n)}$$

Definition 2.6. *Distributionally One-Way Function (Impagliazzo Luby [18])* A function f is distributionally one-way if it is computable in polynomial time, and for some constant $c > 0$, for any PPTM $\mathcal{M}$, the two ensembles:

1. $x \circ f(x)$, where x is selected by $U_{k(n)}$
2. $\mathcal{M}(f(x)) \circ f(x)$, where the randomness is over $x \sim U_{k(n)}$ and $\mathcal{M}$'s coin flips

are $\frac{1}{n^c}$-statistically far (if the ensembles *are not* $\frac{1}{n^c}$-statistically far we say that $\mathcal{M}$ comes $\frac{1}{n^c}$-close to inverting f). The intuition is that it is hard to find a random inverse of a distributionally one-way function (even though finding *some* inverse may be easy). This is a weaker primitive than a one-way function, in the sense that any one-way function is also distributionally one-way, but the converse may not be true. Despite this, the following lemma states that the existence of both primitives is equivalent.

Lemma 2.1. *(Impagliazzo Luby [18]) Distributionally one-way functions exist if and only if one-way functions exist.*

We now define almost one-way functions, functions that are only hard to invert for infinitely many input lengths (compared with standard one-way functions that are hard to invert for all but finitely many input lengths).

Definition 2.7. A function f is an *almost one-way function* if it is computable in polynomial time, and for infinitely many input lengths, for any PPTM $\mathcal{A}$, the probability that $\mathcal{A}$ inverts f on a random input is negligible. Namely, for any polynomial p, there exist infinitely many n's such that:

$$Pr_{x \sim U_{k(n)}, \mathcal{M}}[\mathcal{M}(f(x)) \in f^{-1}(x)] < \frac{1}{p(n)}$$

Similarly, we define almost distributionally one-way functions as functions that are hard to randomly invert for infinitely many input lengths, the existence of almost one-way and almost distributionally one-way functions are equivalent, by the methods [18] used to prove Lemma 2.1.

3 An Information Theoretic Learning Algorithm

In this section we present an algorithm for learning ACDs. Recall the algorithm can only observe $\mathcal{D}$'s public outputs as it runs. After observing $\mathcal{D}$'s public outputs for i steps, we refer to **the distribution $\mathbf{G_i}$:** A (conditional) distribution over S_{init} (recall S_{init} is the concept class of possible initial secret states). For a secret state $s \in S_{init}$, $G_i[s]$ is the conditional probability that s was the initial secret state computed by $\mathcal{G}$ ($s = s_0$), given all public information (the initial public state $\mathcal{G}$ computed was p_0 and the public outputs of the past activations of $\mathcal{D}$ were $(p_1 \ldots p_i)$). The randomness is over $\mathcal{G}$ and $\mathcal{D}$'s (secret) random coins.

Recall also that D_i^s (defined in Section 1) is the distribution of the public state that will be generated by $\mathcal{D}$'s next activation ($\mathcal{D}$'s public output), given that the initial secret state was s, and given all the public information.

We will present an information-theoretic learning algorithm $\mathcal{L}$ for learning any ACD $(\mathcal{G}, \mathcal{D})$. We do not analyze $\mathcal{L}$'s size or running-time, but note that it uses $\mathcal{G}$ and $\mathcal{D}$, and if $\mathcal{G}$ and $\mathcal{D}$ are uniform then so is $\mathcal{L}$. We remark that in general $\mathcal{L}$ is not polynomial-time, and even if $\mathcal{G}$ and $\mathcal{D}$ are PPTMs, $\mathcal{L}$ as we present it below may require exponential time (we will see in the next section that if there is no efficient learning algorithm for a polynomial-time ACD, then almost one-way functions exist).

Theorem 3.1. *For any ACD $(\mathcal{G}, \mathcal{D})$, there exists a learning algorithm $\mathcal{L}$ that $(\delta(n), \varepsilon(n))$-learns the ACD, and activates $\mathcal{D}$ for at most $O\left(\frac{\log |S_{init}|}{\delta^2(n) \cdot \varepsilon^2(n)}\right)$ steps.*[3]

Proof. The hypothesis that the learning algorithm will output will be some secret state $h \in S_{init}$, such that with high probability the hypothesis secret state and the real secret state (s_0) induce statistically close distributions after $\mathcal{D}$'s next activation.

We refer to a fixed input length n, and take $\delta = \delta(n)$ and $\varepsilon = \varepsilon(n)$. As the algorithm runs, we examine the random variable X, the initial secret state that $\mathcal{G}$ computed (s_0), and its Shannon Entropy $H(X)$ (see Section 2). Intuitively, as $\mathcal{L}$ sees samples distributed by $D_i^{s_0}$, this entropy should decrease. We will show that in each round either most of $\mathcal{G}$'s possible outputs generate close distributions, in which case the algorithm can stop, or the algorithm is given a new sample and the expected entropy of X given the new sample decreases significantly.

The learning algorithm $\mathcal{L}$

1. For $i \leftarrow 1 \ldots \frac{64 \cdot \log |S_{init}|}{\delta^2(n) \cdot \varepsilon^2(n)}$ do:

   (a) If there exists some very high weight (by G_i) subset of initial secret states, whose distributions (D_i^s) are all "close" then we can terminate: if there exists some $A \subseteq S_{init}$ such that $Pr_{G_i}[A] \geq 1 - \frac{\delta}{2}$ and $\forall a_1, a_2 \in A, \Delta(D_i^{a_1}, D_i^{a_2}) \leq \varepsilon$, then choose some $h \in A$, output h and terminate.

   (b) Activate $\mathcal{D}$ to sample p_{i+1} and proceed to round $i + 1$.

2. If the condition of Step 1a was not met in any loop iteration, output arbitrarily some $h \in S_{init}$ and terminate.

Claim 3.1. *If $\mathcal{L}$ terminates within the loop in round i, then with probability at least $1 - \frac{\delta}{2}$, the statistical distance $\Delta(D_i^{s_0}, D_i^h)$ is at most ε*

[3] As mentioned previously, this theorem holds even if $\mathcal{G}$ is given an "auxiliary" input in $\{0,1\}^n$, instead of just an input length parameter. In this setting, the learning algorithm also sees the auxiliary input.

Proof. For any two initial secret states in A, the statistical distance between their induced distributions is at most ε. If $\mathcal{L}$ terminates in the loop, then $Pr_{G_i}[A] \geq 1-\frac{\delta}{2}$ and thus $Pr_{G_i}[s_0 \in A] \geq 1-\frac{\delta}{2}$. If $s_0 \in A$ then (since $h \in A$) we conclude that $\Delta(D_i^{s_0}, D_i^h) \leq \varepsilon$. □

It remains to bound the probability that $\mathcal{L}$ terminates in its second step. This will be done by showing that as long as the condition of the first step in the loop does not hold, the entropy $H(X)$ decreases significantly with high probability. Towards this end we examine each round separately, and look at two random variables:

- The random variable X_i over S_{init}, distributed by G_i. This is the conditional distribution of the secret state generated by $\mathcal{G}$, given all of the past public outcomes.
- The random variable Y_i over S_p, the public output of $\mathcal{D}$'s next activation, given all of the past public output. This distribution can be generated by sampling an initial secret state s by G_i and then sampling from D_i^s.

We would like to show that the expected entropy of X_i given Y_i is significantly smaller than the entropy of X_i.

Lemma 3.1. *For any distribution G_i on the initial secret state, if in the i-th round the condition of Step 1a of $\mathcal{L}$ does not hold (i.e. there exists no $A \subseteq S_{init}$ such that $Pr_{G_i}[A] \geq 1-\frac{\delta}{2}$ and $\forall a_1, a_2 \in A, \Delta(D_i^{a_1}, D_i^{a_2}) \leq \varepsilon$), then the expected entropy drop is:*

$$E_{y \sim Y_i}[H(X_i) - H(X_i|Y_i = y)] \geq \frac{\varepsilon^2 \cdot \delta}{32}$$

Proof. The expected entropy drop is exactly:

$$E_{y \sim Y_i}[H(X_i) - H(X_i|Y_i = y)] = H(X_i) - H(X_i|Y_i) = I(X_i;Y_i) = H(Y_i) - H(Y_i|X_i)$$

Where $I(X_i;Y_i)$ is the mutual information between X_i and Y_i (see [6]). Consider the distributions on the next public state $\{D_i^s\}_{s \in S_{init}}$ over S_p, where $t = |S_p|$. We view these distributions as vectors in $\mathbb{R}^t$ (the j-th coordinate of the vector is the probability of public state j by the distribution), and denote $\vec{d_s} = D_i^s$ ($\forall s \in S_{init}, \|\vec{d_s}\|_1 = 1$). In round i, since $\mathcal{L}$ did not terminate inside the loop, $\forall A \subseteq S_{init}$ such that $Pr_{G_i}[A] \geq 1-\frac{\delta}{2}$ we know that $\exists a_1, a_2 \in A$ such that $\Delta(\vec{d_{a_1}}, \vec{d_{a_2}}) > \varepsilon$, where Δ is the statistical distance ($\frac{1}{2}\|\vec{d_{a_1}} - \vec{d_{a_2}}\|_1$).
For a vector $\vec{v} \in \mathbb{R}^t$, define the set of vectors that are γ-close to $\vec{v}$:

$$B_{\vec{v}}^{\gamma} \triangleq \{s : \Delta(\vec{v}, \vec{d_s}) \leq \gamma\}$$

Since for all $\vec{v} \in \mathbb{R}^t$ every two distributions in $B_{\vec{v}}^{\frac{\varepsilon}{2}}$ are ε-close:

$$\forall \vec{v} \in \mathbb{R}^t, Pr_{G_i}[B_{\vec{v}}^{\frac{\varepsilon}{2}}] \leq 1 - \frac{\delta}{2}$$

We now turn to using the *mean value theorem* (see e.g. [24]): consider the Entropy function as $\eta : \mathbb{R}^t \mapsto \mathbb{R}$, where $\eta(\vec{v}) = -\sum_{j=1}^{t} \vec{v}_j \cdot \log \vec{v}_j$ (we take $0 \log 0 = 0$). The random variable Y_i is distributed by the weighted mean of the distributions $\vec{y} = \sum_{s \in S_{init}} G_i[s] \cdot \vec{d_s}$, so $H(Y_i) = \eta(\vec{y})$ and we also note that $H(Y_i|X_i) = \sum_{s \in S_{init}} G_i[s] \cdot \eta(\vec{d_s})$. W.l.o.g. we assume $\forall j \in [t], y_j > 0$ (if $y_j = 0$

then no distribution ever samples coordinate j, and thus we can ignore the coordinate altogether). By the mean value theorem for every $s \in S_{init}$ there exists some $\vec{\ell_s}$ on the line between $\vec{y}$ and $\vec{d_s}$ such that:

$$\eta(\vec{d_s}) = \eta(\vec{y}) + \nabla\eta(\vec{y}) \times (\vec{d_s} - \vec{y}) + \frac{1}{2}(\vec{d_s} - \vec{y})^T \times Hes_\eta(\vec{\ell_s}) \times (\vec{d_s} - \vec{y})$$

Where $Hes_\eta(\ell_s)$ is the Hessian of the function η at ℓ_s, the matrix of second derivatives of η at ℓ_s. For the entropy functions we get that

$$Hes_\eta(\vec{\ell_s}) = \begin{pmatrix} -\frac{1}{(\vec{\ell_s})_1} & 0 & \dots & 0 \\ 0 & \dots & \dots & 0 \\ 0 & \dots & 0 & -\frac{1}{(\vec{\ell_s})_t} \end{pmatrix}$$

Therefore, by taking:

$$\alpha_s \triangleq \frac{1}{2} \cdot (\vec{d_s} - \vec{y})^T \times Hes_\eta(\vec{\ell_s}) \times (\vec{d_s} - \vec{y}) = -\frac{1}{2} \cdot \sum_{j=1}^{t} \frac{((\vec{d_s})_j - (\vec{y})_j)^2}{(\vec{\ell_s})_j}$$

We get by the mean value theorem that:

$$\begin{aligned}
H(Y_i) - H(Y_i|X_i) &= \eta(\vec{y}) - \sum_{s\in S_{init}} G_i[s] \cdot \eta(\vec{d_s}) \\
&= \eta(\vec{y}) - \sum_{s\in S_{init}} \left[G_i[s] \cdot \Big(\eta(\vec{y}) + \nabla\eta(\vec{y}) \times (\vec{d_s} - \vec{y}) + \alpha_s\Big)\right] \\
&= \eta(\vec{y}) - \left[\sum_{s\in S_{init}} (G_i[s] \cdot \eta(\vec{y}))\right] - \nabla\eta(\vec{y}) \times \left[\sum_{s\in S_{init}} (G_i[s] \times (\vec{d_s} - \vec{y}))\right] - \left[\sum_{s\in S_{init}} (G_i[s] \cdot \alpha_s)\right] \\
&= \eta(\vec{y}) - [\eta(\vec{y})] - \nabla\eta(\vec{y}) \times [\vec{y} - \vec{y}] - \left[\sum_{s\in S_{init}} (G_i[s] \cdot \alpha_s)\right] \\
&= -\sum_{s\in S_{init}} G_i[s] \cdot \alpha_s
\end{aligned}$$

We now bound α_s for s such that $\Delta(\vec{y}, \vec{d_s}) \geq \frac{\varepsilon}{2}$. We concentrate on j's in T such that $\frac{|(\vec{d_s})_j - (\vec{y})_j|}{(\vec{\ell_s})_j} \geq \frac{\varepsilon}{4}$. For such "good" j's we make the following claim:

Claim 3.2. *If $\Delta(\vec{y}, \vec{d_s}) \geq \frac{\varepsilon}{2}$, then $\sum_{good\ j's} |(\vec{d_s})_j - (\vec{y})_j| \geq \frac{\varepsilon}{2}$*

Proof. For "bad j's" (j's that aren't "good") we know that $\frac{\varepsilon}{4} > \frac{|(\vec{d_s})_j - (\vec{y})_j|}{(\vec{\ell_s})_j} \geq \frac{|(\vec{d_s})_j - (\vec{y})_j|}{\max\{(\vec{d_s})_j, (\vec{y})_j\}}$, and thus:

$$|(\vec{d_s})_j - (\vec{y})_j| < \frac{\varepsilon}{4} \cdot \max\{(\vec{d_s})_j, (\vec{y})_j\}$$

We conclude that:

$$\sum_{bad\ j's} |(\vec{d_s})_j - (\vec{y})_j| < \sum_{bad\ j's} \frac{\varepsilon}{4} \cdot \max\{(\vec{d_s})_j, (\vec{y})_j\} \leq \frac{\varepsilon}{2}$$

The statistical difference $\Delta(\vec{y}, \vec{d_s})$ is at least $\frac{\varepsilon}{2}$, so the sum of differences on *all* indices is at least ε. Thus the sum of differences on "good" indices must be at least $\frac{\varepsilon}{2}$. □

We conclude that:

$$-\alpha_s = \frac{1}{2} \cdot \sum_{j=1}^{t} \frac{|(\vec{d_s})_j - (\vec{y})_j|^2}{(\vec{\ell_s})_j} \geq \frac{1}{2} \sum_{good\ j's} \frac{|(\vec{d_s})_j - (\vec{y})_j|^2}{(\vec{\ell_s})_j} \geq \frac{1}{2} \sum_{good\ j's} \frac{\varepsilon}{4} |(\vec{d_s})_j - (\vec{y})_j| \geq \frac{\varepsilon^2}{16}$$

Now recall that $Pr_{G_i}[B_{\vec{y}}^{\frac{\varepsilon}{2}}] < 1 - \frac{\delta}{2}$, or in other words the total weight of vectors $\vec{d_s}$ that are at distance at least $\frac{\varepsilon}{2}$ from $\vec{y}$ ("good" vectors) is at least $\frac{\delta}{2}$. We conclude that:

$$H(Y_i) - H(Y_i|X_i) = \sum_{s \in S_{init}} G_i[s] \cdot -\alpha_s \geq \sum_{good\ s's} G_i[s] \cdot -\alpha_s \geq \frac{\varepsilon^2}{16} \sum_{good\ s's} G_i[s] \geq \frac{\varepsilon^2}{32} \cdot \delta$$

□

We have shown that the *expected* entropy drop of X_i in each round is at least $\frac{\delta \cdot \varepsilon^2}{32}$ (regardless of past outcomes). The initial entropy of X is at most $\log |S_{init}|$, and thus (using Theorem B.1 of Appendix B), within the first $\frac{64 \cdot \log |S_{init}|}{\delta^2 \cdot \varepsilon^2}$ rounds, with probability at least $1 - \frac{\delta}{2}$, the entropy of X conditioned on past samples becomes 0. This is true even though the entropy losses, as random variables, are not at all independent. For a full discussion of this result, see appendix B which deals with probabilistic recurrence relations. Once the entropy of X given past samples becomes 0, s_0 is known and the algorithm will terminate within the loop.

To conclude, with probability at least $1 - \frac{\delta}{2}$ the algorithm's termination is in the loop. When the algorithm terminates in the loop, with probability at least $1 - \frac{\delta}{2}$ its hypothesis distribution is ε-close to the target distribution. Thus, the success probability of the algorithm is at least $1 - \delta$. □

4 Hard to Learn ACDs Imply Almost One-Way Functions

4.1 Two Useful Results

Combining almost one-way functions In this paper we use several efficiently computable functions, and show that if there exist algorithms that for all but finitely many n's can find almost-random inverses for *all* the functions then there exist no ACDs that are hard to learn. We would like to use this fact to show that the existence of hard to learn ACDs implies the existence of one-way functions. The difficultly is that the sets of functions may be of polynomial size, if almost one-way functions do not exist then each of the functions is invertible for all but finitely many n's, but this does not directly imply that together they are *all* invertible for all but finitely many n's. The following Lemma deals with this technicality, and the proof can also be applied to collections of standard one-way functions:

Lemma 4.1. *For $n \in \mathbb{N}$, let $f_1 \dots f_{q(n)}$ be some collection of efficiently computable functions, where*

$$f_{i_n} : \{0,1\}^{k_i(n)} \mapsto \{0,1\}^{\ell_i(n)}$$

If almost one-way functions do not exist, then for any polynomial p there exists a PPTM $\mathcal{M}$ such that for all but finitely many n's, for all $i \in [q(n)]$, $\mathcal{M}$ with input i comes $\frac{1}{p(n)}$-close to inverting f_i (the distributions $x \circ f_{i_n}(x)$ and $\mathcal{M}(i, f_{i_n}(x)) \circ f_{i_n}(x)$, where $x \sim U_{k_i(n)}$, are $\frac{1}{p(n)}$-close).

Proof. We define a new function g, that receives an input beginning with an index $i \in [q(n)]$ and activates f_i on the rest of it's input. For any n, g_n's domain size is $k_g(n) = \log q(n) + max_{j\in[q(n)]}\{k_j(n)\}$, and g_n's range size is $\ell_g(n) = \log q(n) + max_{j\in[q(n)]}\{\ell_j(n)\}$.

$$g_n(i, x, z) = (i, f_i(x), 0^{\ell_g(n)-\ell_i(n)-\log q(n)})$$

Where $i \in \{0,1\}^{\log q(n)}$, $x \in \{0,1\}^{k_i(n)}$, $z \in \{0,1\}^{k_g(n)-k_i(n)-\log q(n)}$ (z pads the input so we can use inputs of a similar length for all f_i's, the output is similarly padded).

We assumed that almost one-way functions do not exist, and thus g is not almost distributionally one-way and there exists an algorithm $\mathcal{M}$ that for all but finitely many n's comes $\frac{1}{q(n)\cdot n^c}$-close to inverting g on randomly selected inputs. This implies that for all but finitely many n's, for any fixed $i \in [q(n)]$, $\mathcal{M}$ comes $\frac{1}{n^c}$-close to inverting g on inputs of the form (i, x, z), where x and z are selected uniformly and at random (and i is fixed).

We construct a PPTM $\mathcal{M}'$ for inverting f_{i_n}'s: on input (i, y) the PPTM $\mathcal{M}'$ activates $\mathcal{M}$ on $(i, y, 0^{\ell_g(n)-\ell_i(n)})$ to get (i, x, z) and returns x. $\mathcal{M}'$'s activation of $\mathcal{M}$ is on the distribution generated by selecting x and z uniformly and at random, and then activating g on (i, x, z) (i is fixed) . We have seen that for all but finitely many n's $\mathcal{M}$ comes $\frac{1}{n^c}$-close to inverting g on inputs of this form. Thus, for all but finitely many n's, for any $i \in [q(n)]$, when $\mathcal{M}'$ is activated with i it also comes $\frac{1}{n^c}$-close to inverting f_{i_n} on a randomly selected input. $\square$

Statistical versus computational distance Goldreich [9] showed that one-way functions exist if and only if there exist ensembles that are statistically far but computationally indistinguishable. This result is important in showing the connection between widely used cryptographic primitives, but it leaves an interesting question open: how much larger than the computational distinguishability can the statistical distance be before one-way functions are implied? Goldreich shows that some polynomial gap implies one-way functions, but it seems natural to try and improve this result for smaller gaps. For example, does the existence of a pair of ensembles that are at statistical distance $\frac{1}{2}$, but cannot be distinguished with advantage $\frac{1}{100}$ by any PPTM, imply the existence of one-way functions? Goldreich's result, and the methods he uses (using false entropy generators and the results of Håstad *et al.* [14]), do not provide an answer to this question. In the next Theorem we provide a simpler and self-contained (up to the equivalence of one-way and distributionally one-way functions) proof of a tighter result. As a corollary we derive a simpler proof that the existence of false-entropy generators implies the existence of one-way functions.

Theorem 4.1. *One-way functions exist if and only if there exists a pair of polynomial time constructible ensembles D^0 and D^1 and polynomials $\varepsilon(n), p(n)$ such that:*

- *The ensembles D^0 and D^1 are $\varepsilon(n)$-statistically far.*
- *The ensembles D^0 and D^1 are $(\varepsilon(n) - p(n))$-computationally indistinguishable.*

Proof. If one-way functions exist then there exist pseudo-random generators (see [14]), and thus there exist ensembles that are very far statistically but computationally indistinguishable. We will focus on the second direction of the theorem. Assume (w.l.o.g.) the two sequences are over the same domain T. We look at D^0 and D^1 as functions, whose input is the random coins flipped by the algorithms that generate the ensembles. Suppose the polynomial $q(n)$ bounds from above the number of coins flipped by both D_n^0 and D_n^1. We construct an efficiently computable function $g_n : \{0,1\}^{q(n)+1} \mapsto T$

$$\forall r \in \{0,1\}^{q(n)} : g(b,x) = \begin{cases} D_n^0(r) & \text{when } b = 0 \\ D_n^1(r) & \text{when } b = 1 \end{cases}$$

If one-way functions do not exist then g is not distributionally one-way and there exists a PPTM $\mathcal{M}$ that, for infinitely many n's, comes $O(p^3(n) \cdot \frac{1}{\log \frac{1}{p(n)}})$-close to finding a random inverse for g.

We will use $\mathcal{M}$ to construct an *efficient* algorithm $\mathcal{A}$ for distinguishing between D^0 and D^1. $\mathcal{A}$ receives a sample s in the domain T of the distributions. The sample s is generated by activating the function g on a uniformly random input (both the bit b and the random string r are selected uniformly and at random). The goal of the algorithm $\mathcal{A}$ will be to "guess" whether the bit b used to generate the sample was 0 or 1 ($\mathcal{A}$'s out put is a single bit). We will show that if b and r are selected uniformly and at random, then $\mathcal{A}$ guesses b correctly with probability $\frac{1}{2} + \frac{\varepsilon}{2} - \frac{p(n)}{2}$, which implies that D_n^0 and D_n^1 are *not* $(\varepsilon(n) - p(n))$-computationally indistinguishable.

The Algorithm $\mathcal{A}$ on input s:

1. for $i \leftarrow 1 \ldots O(\frac{1}{p^2(n)} \cdot \log \frac{1}{p(n)})$:
   activate $\mathcal{M}$ to get $(b_i, r_i) \leftarrow \mathcal{M}(s)$
2. If at least half of the b_i's are 0's, answer 0, otherwise answer 1.

The following lemma shows that $\mathcal{A}$ is a good predictor for b on a uniformly generated sample:

Lemma 4.2. *If $\Delta(D_n^0, D_n^1) \geq \varepsilon(n)$, then the probability that $\mathcal{A}$ successfully predicts b is bounded away from $\frac{1}{2}$:*

$$Pr[\mathcal{A}(g(b,r)) = b] \geq \frac{1}{2} + \frac{\varepsilon}{2} - \frac{p(n)}{2}$$

Where the probability is over the random selection of (b,r) and $\mathcal{A}$'s coin tosses.

Proof. Denote the statistical distance between D_n^0 and D_n^1 by ε. We begin by assuming that $\mathcal{M}$ randomly inverts g (in truth $\mathcal{M}$ only comes statistically close to randomly inverting g, which will suffice). We divide the samples in the distributions' domain T into three disjoint sets, T_0, $T_\approx$ and T_1, according to which of the two distributions gives them a higher probability. The set T_0 is the set of items whose probability by D^0 is significantly larger than their probability by D^1, T_1 is the set of items whose probability by D^0 is significantly smaller than by D^1, and $T_\approx$ includes all items not in T_0 or T_1 (items whose probability by both distributions is approximately equal). Formally:

$$T_0 = \{s \in T : (1 + \frac{p(n)}{4})D_n^1[s] < D_n^0[s]\}$$

$$T_1 = \{s \in T : (1 + \frac{p(n)}{4})D_n^0[s] < D_n^1[s]\}$$

$$T_\approx = \{s \in T : \frac{D_n^0[s]}{(1 + \frac{p(n)}{4})} \leq D_n^1[s] \leq (1 + \frac{p(n)}{4})D_n^0[s]\}$$

The algorithm $\mathcal{A}$ answers 0 on items in T_0 and 1 on items in T_1 (with high probability):

Claim 4.1. *If $s \in T_0$, then $Pr[\mathcal{A}(s) = 0] \geq 1 - \frac{p(n)}{100}$. If $s \in T_1$, then $Pr[\mathcal{A}(s) = 1] \geq 1 - \frac{p(n)}{100}$.*

We now analyze $\mathcal{A}$'s success probability. Suppose that the sample s is generated by D_n^0 (the case $b = 0$). To bound from below the probability that $\mathcal{A}$ outputs 0, we consider the 3 possible scenarios $s \in T_0, s \in T_{\approx}, s \in T_1$. If $s \in T_0$, then $\mathcal{A}$ outputs 0 with probability at least $1 - \frac{p(n)}{100}$ (see Claim 4.1). We get that:

$$Pr[\mathcal{A}(s) = 0 | b = 0] \geq D_n^0[T_0] \cdot (1 - \frac{p(n)}{100}) + \sum_{s \in T_{\approx}} D_n^0[s] \cdot Pr[\mathcal{A}(s) = 0]$$

And similarly:

$$Pr[\mathcal{A}(s) = 1 | b = 1] \geq D_n^1[T_1] \cdot (1 - \frac{p(n)}{100}) + \sum_{s \in T_{\approx}} D_n^1[s] \cdot Pr[\mathcal{A}(s) = 1]$$

Since b is selected uniformly and at random we conclude that:

$$\begin{aligned} 2 \cdot Pr[\text{A(s) = b}] \quad \geq \quad & (1 - \frac{p(n)}{100}) \cdot (D_n^0[T_0] + D_n^1[T_1]) + \\ & \sum_{s \in T_{\approx}} D_n^0[s] \cdot Pr[\mathcal{A}(s) = 0] + D_n^1[s] \cdot Pr[\mathcal{A}(s) = 1] \end{aligned}$$

Focusing on the second part of the equation, observe that:

$$\begin{aligned} \sum_{s \in T_{\approx}} D_n^1[s] \cdot Pr[\mathcal{A}(s) = 1] \quad & \geq \quad \sum_{s \in T_{\approx}} \frac{D_n^0[s]}{1 + \frac{p(n)}{4}} \cdot Pr[\mathcal{A}(s) = 1] \\ & = \quad \sum_{s \in T_{\approx}} (D_n^0[s] - \frac{\frac{p(n)}{4} \cdot D_n^0[s]}{1 + \frac{p(n)}{4}}) \cdot Pr[\mathcal{A}(s) = 1] \\ & \geq \quad (\sum_{s \in T_{\approx}} D_n^0[s] \cdot Pr[\mathcal{A}(s) = 1]) - (\sum_{s \in T_{\approx}} \frac{p(n)}{4} \cdot D_n^0[s]) \\ & \geq \quad (\sum_{s \in T_{\approx}} D_n^0[s] \cdot Pr[\mathcal{A}(s) = 1]) - \frac{p(n)}{4} \end{aligned}$$

Thus:

$$\begin{aligned} 2 \cdot Pr[\text{A is right on s}] \quad \geq \quad & (1 - \frac{p(n)}{100}) \cdot (D_n^0[T_0] + D_n^1[T_1]) + \\ & (\sum_{s \in T_{\approx}} D_n^0[s] \cdot (Pr[\mathcal{A}(s) = 0] + Pr[\mathcal{A}(s) = 1])) - \frac{p(n)}{4} \\ = \quad & (1 - \frac{p(n)}{100}) \cdot (D_n^0[T_0] + D_n^1[T_1]) + (\sum_{s \in T_{\approx}} D_n^0[s]) - \frac{p(n)}{4} \\ \geq \quad & D_n^0[T_0] + D_n^1[T_1] + D_n^0[T_{\approx}] - \frac{p(n)}{2} \end{aligned}$$

We would now like to show that T_1's probability is significantly higher by D_n^1 than by D_n^0. This seems to be true, since the two distributions have statistical distance ε. The (small) problem is that T_1 is not the set of *all* samples which D_n^1 assigns higher probability than D_n^0, but only the set of such items with *significantly* higher probability. The following claim states that, despite this, T_1 probability by D_n^1 is indeed significantly higher than by D_n^0:

Claim 4.2. $D_n^0[T_1] \leq D_n^1[T_1] + \varepsilon - \frac{p(n)}{4}$.

We conclude that:

$$2 \cdot Pr[\text{A is right on s}] \geq D_n^0[T_0] + D_n^0[T_1] + D_n^0[T_{\approx}] + \varepsilon - \frac{3p(n)}{4} = 1 + \varepsilon - \frac{3p(n)}{4}$$

This result was achieved assuming that $\mathcal{M}$ was a *perfect* algorithm for randomly inverting g. In truth, however, $\mathcal{M}$ only comes $O(p^3(n) \cdot \frac{1}{\log \frac{1}{p(n)}})$-close to inverting g. This is sufficient, since we activated $\mathcal{M}$ at most $O(\frac{1}{p^2(n)} \cdot \log \frac{1}{p(n)})$ times on a randomly selected sample, "accumulating" an error of at most $\frac{p(n)}{8}$:

$$Pr[\text{A is right on s}] \geq \frac{1}{2} + \frac{\varepsilon}{2} - \frac{p(n)}{2}$$

$\square$

By lemma 4.2, $\mathcal{A}$ is a good predictor for infinitely many input lengths. On any input length n for which $\mathcal{A}$ is a good predictor, it can be used to distinguish between D_n^0 and D_n^1 with advantage $\varepsilon(n) - p(n)$. Thus the two ensembles are not $\varepsilon(n) - p(n)$-computationally indistinguishable, a contradiction to the assumption that g is not distributionally one-way.

$\square$

4.2 ACDs and One-Way Functions

In this section we deal with efficiently constructible ACDs. We say an ACDs is *efficiently constructible* if its generating algorithm $\mathcal{G}$ and its sampling algorithm $\mathcal{D}$ are both PPTMs. We will show that if there exists an efficiently constructible ACD that is hard to learn by a PPTM learning algorithm making few samples, then there exist almost one-way functions. This will also imply that if there exists a polynomial time authenticator, where $\mathcal{V}$'s access pattern is hard to learn, then almost one-way functions exist.

Definition 4.1. An ACD $(\mathcal{G}, \mathcal{D})$ is *hard to $(\delta(n), \varepsilon(n))$-learn with $k(n)$ samples* if:

1. It is efficiently constructible
2. For any PPTM that tries to learn the ACD and lets the learning process run for at most $k(n)$ steps, outputting some hypothesis h, for infinitely many n's:

$$Pr\left[\Delta(D_{k+1}^{s_0}, D_{k+1}^{h}) > \varepsilon(n)\right] > \delta(n)$$

Theorem 4.2. *Almost one-way functions exist if and only if there exists an adaptively changing distribution $(\mathcal{G}, \mathcal{D})$ and polynomials $\varepsilon(n), \delta(n)$, such that it is hard to $(\delta(n), \varepsilon(n))$-learn the ACD $(\mathcal{G}, \mathcal{D})$ with $O\left(\frac{\log |S_{init}|}{\delta^2(n) \cdot \varepsilon^4(n)}\right)$ samples*[4].

[4]Similarly to the information-theoretic case, this theorem also holds if $\mathcal{G}$ is given an auxiliary input instead of just the input length parameter 1^n, as long as the auxiliary input is selected by an efficiently computable distribution that is known to the learning algorithm.

Proof. If almost one-way functions exist then there exist pseudo-random generators with arbitrary polynomial stretch for the corresponding (infinitely many) input lengths (see [14]). To construct an ACD, flip an n-bit seed for the generator as the initial secret input. The sampling algorithm $\mathcal{D}$ outputs n pseudo-random bits in each activation. Note that $\mathcal{D}$ doesn't even need to use random coins! If this ACD could be learned (with non-negligible $(\delta(n), \varepsilon(n))$) then the learning algorithm could be used to distinguish the generator's output from uniform random bits (a contradiction). Alternatively, from the results of Kearns *et al.* [20], if almost one-way functions exist then they can be used to construct pseudo-random functions for the corresponding input lengths (see [10]), and there exists an ACD that is hard to learn. In fact, the distributions that they construct are static and do not change!

We are primarily concerned with the other direction of the theorem, and follow the notation of the previous subsection. Recall $\vec{y}$ is the weighted mean of the sampling distributions (by G_i):

$$\vec{y} = \sum_{s \in S_{init}} G_i[s] \cdot \vec{d_s}$$

Assume that almost one-way functions do not exist, and let $q(n) = c \cdot \left(\frac{\log |S_{init}|}{\delta^2 \cdot \varepsilon^4} \right)$ (the constant c will be specified later), $q(n)$ will be an upper bound on the number of rounds the algorithm runs. We construct an efficient algorithm $\mathcal{L}$ for learning $(\mathcal{G}, \mathcal{D})$ for infinitely many n's, beginning with a high-level description of $\mathcal{L}$'s operation. Note that the high-level description of this new (polynomial time) learning algorithm is (intentionally) very similar to the information-theoretic algorithm described in the previous subsection. In fact, the algorithms differ mainly in the termination condition within their loops, where the termination condition of the new algorithm is a polynomial-time equivalent of the information-theoretic algorithm's termination condition.

The efficient learning algorithm $\mathcal{L}$ (in round i):

1. For $i \leftarrow 1 \ldots q(n)$ do:

   (a) Recall $\vec{y}$ is the distribution of the next public output, p_{i+1}, where the probability is over $\mathcal{G}$ and $\mathcal{D}$'s random coins given all of the previous samples and public states.
   Estimate whether the weight of distributions $\vec{d_s}$ that are "close" to $\vec{y}$ is high. If the weight estimate is high then approximate sampling some $h \in S_{init}$ by G_i, output h and terminate (the exact specification of this step will be given later).

   (b) Activate $\mathcal{D}$ to sample p_{i+1} (by $D_i^{s_0}$) and proceed to round $i + 1$.

2. Output arbitrarily some $h \in S_{init}$ and terminate.

It remains to specify more precisely Step 1a of the algorithm, and to prove that if almost one-way functions do not exist then it can be executed efficiently. After showing this we will prove the algorithm's correctness. We begin with an intuitive overview of Step 1a.

The Big Picture: The algorithm's goal in Step 1a is to test whether there is a subset $A \subseteq S_{init}$ of high weight (by G_i) such that the distributions $\{D_i^s\}_{s \in A}$ are all close to $\vec{y}$. More formally, in this step we run the *concentrated − around − mean* procedure to distinguish between two scenarios:

1. The weight of distributions at distance $\frac{\varepsilon}{2}$ from $\vec{y}$ is not very large (less than $1 - \frac{\delta}{2}$):

   In this case $concentrated - around - mean$ should reject with high probability (probability at least $1 - \frac{\delta}{4q(n)}$). This implies that when $concentrated - around - mean$ accepts, by selecting a random h from S_{init} according to the distribution G_i, with high probability we select h such that D_i^h is ε-close to $D_i^{s_0}$.

2. The weight of distributions at distance $\frac{\varepsilon^2}{36}$ from $\vec{y}$ is very large (at least $1 - \frac{\delta}{10}$):

   In this case $concentrated - around - mean$ should with high probability (probability at least $1 - \frac{\delta}{4q(n)}$). This implies, by similar arguments to those used in the proof of Theorem 3.1, that whenever $concentrated - around - mean$ rejects the entropy of X decreases significantly, and thus with high probability within $q(n)$ rounds $concentrated - around - mean$ will accept with high probability.

The main difficulty is showing that if almost one-way functions do not exist then we can implement $concentrated - around - mean$ efficiently. The procedure $concentrated - around - mean$ requires that we implement the following tasks efficiently:

Generating G_i: Generating samples (items in S_{init}) by the distribution G_i

Generating Y_i: Generating samples from the distribution Y_i

Testing $\Delta(Y_i, D_i^s)$: For an initial secret state $s \in S_{init}$, generated by sampling from the distribution G_i, testing whether Y_i and D_i^s are $\frac{\varepsilon}{2}$-far or $\frac{\varepsilon^2}{36}$-close (by taking many samples from Y_i and D_i^s).

We implement these tasks using two (efficiently computable) functions, f and g.
Take $\ell = 200 \frac{\log 4q(n)}{\delta^2} \log \frac{1}{\delta}$, $m = \frac{200}{\varepsilon^4}(\log \frac{1}{\delta} + 4)$.

The function f: f receives the input length parameter 1^n, a random string for the generating algorithm, a number $i \in [q(n)]$ (the number of sampling rounds) and random strings for $q(n)$ invocations of the sampling algorithm. f outputs i and all the public states and samples generated by running $\mathcal{G}$ and then running i activations of $\mathcal{D}$ using the given random strings. If $i < q(n)$, f pads its output with 0's. For input $(1^n, i, r_{\mathcal{G}}, r_0, \ldots, r_{q(n)-1})$, let p_j be the j-th public state that was generated:

$$f(1^n, i, r_{\mathcal{G}}, r_{\mathcal{D}}^1, \ldots, r_{\mathcal{D}}^{q(n)}) = (1^n, i, p_0, p_1, \ldots, p_i, 0 \ldots 0)$$

If the ACD is efficiently constructible then f is efficiently computable. We assumed that almost one-way functions do not exist, thus by Lemma 4.1 f is not almost distributionally one-way, and there exists a PPTM $\mathcal{M}_f$ that for all but finitely many n's, for any $i \in [q(n)]$, comes $\frac{\delta}{8 \cdot 4q(n) \cdot \ell \cdot m}$-close to inverting f when i is fixed and the rest of the input is selected uniformly and at random.

The function g: The function g is very similar to f, and receives the same input as f, except for an additional bit $b \in \{0, 1\}$ and a random string r_f for $\mathcal{M}_f$:

$$g(1^n, i, b, r_{\mathcal{G}}, r_f, r_{\mathcal{D}}^1, \ldots, r_{\mathcal{D}}^{q(n)}) = (1^n, i, s_0', p_0, p_1, \ldots, p_i, 0 \ldots 0)$$

Where $p_0, \ldots, p_{i-1}$ are generated as on f, s'_0 is generated by G_{i-1}, and b determines whether p_i is generated by sampling from the distribution $\vec{y}$ (when $b = 0$), or by sampling from $\vec{d_{s'_0}}$ (when $b = 1$).

We now specify g's computational more precisely: the function g acts like f in generating the first $i-1$ samples and public states, but it then uses $\mathcal{M}_f$ to sample the distribution G_{i-1} of the initial secret state given the input length parameter 1^n, the previous samples and public states, generating a secret initial state s'_0[5]. Take:

$$(1^n, i, r'_{\mathcal{G}}, r'^1_{\mathcal{D}}, \ldots, r'^{q(n)}_{\mathcal{D}}) = \mathcal{M}_f(f(1^n, i, r_{\mathcal{G}}, r^1_{\mathcal{D}}, \ldots, r^{q(n)}_{\mathcal{D}}))$$

And g computes s'_0, the secret state generated by $\mathcal{G}$ with input 1^n and random string $r'_{\mathcal{G}}$. After computing s'_0, the function g outputs it and then generates a new public output p_i. If $b = 1$, then p_i is generated exactly as in the function f (using (p_{i-1}, s_{i-1}) and $r^i_{\mathcal{D}}$). If $b = 0$ then $\mathcal{G}$ uses s'_0 and the random strings $r'^1_{\mathcal{D}}, \ldots, r'^{i-1}_{\mathcal{D}}$ to generate secret states $s'_1 \ldots s'_{i-1}$ (the random strings were generated by inverting f and thus they compute the same public states). The new public output p_i is then taken to be $\mathcal{D}$'s output on state (s'_{i-1}, p_{i-1}) using the randomness $r^i_{\mathcal{D}}$ (note that g uses the same random string $r^i_{\mathcal{D}}$ in both cases). We conclude that:

$$p_i = \begin{cases} b = 0 & \mathcal{D}(s'_{i-1}, p_{i-1}, r^i_{\mathcal{D}}) \\ b = 1 & \mathcal{D}(s_{i-1}, p_{i-1}, r^i_{\mathcal{D}}) \end{cases}$$

The intuition is that if $b = 1$, g gives exactly the same information as f. However, if $b = 0$, g also exposes the secret output of $\mathcal{G}$. The 0's at the end of the functions are padding to ensure that outputs are of the same length regardless of i.

If the ACD is efficiently constructible and almost one-way functions do not exist then g is also efficiently computable. By Lemma 4.1 we conclude that if almost one-way functions do not exist then for all but finitely many input lengths the function g inverts f successfully, and there exists a PPTM $\mathcal{M}_g$ that for the same (all but finitely many) input lengths, for any i, comes $\frac{\delta}{8 \cdot 4q(n) \cdot \ell \cdot m}$-close to randomly inverting g when i is fixed and the rest of the input is selected uniformly at random.

Using f and g: Unless we explicitly state otherwise we assume that $\mathcal{M}_f$ and $\mathcal{M}_g$ randomly invert f and g (respectively), we will later count the number of times that f and g are activated by $\mathcal{L}$ and modify the error probability accordingly.

The key point is that by inverting f we can (almost) sample both from G_i and from Y_i (the distribution of the next public output). To sample from G_i use $\mathcal{M}_f$ to invert f on $(1^n, i, p_0, p_1, \ldots, p_i, 0 \ldots 0)$ to get $(1^n, i, r_{\mathcal{G}}, r^1_{\mathcal{D}}, \ldots, r^i_{\mathcal{D}})$. Since the samples $y_1 \ldots y_i$ were indeed generated using uniform random inputs, the distribution of $r_{\mathcal{G}}$ is the distribution of random strings that $\mathcal{G}$ used, given the public information. Sample $s \in S_{init}$ by computing the secret state generated by $\mathcal{G}(1^n, r_G)$. Similarly, by computing $\mathcal{G}$ and $\mathcal{D}$'s run over i rounds with the random strings $(r_{\mathcal{G}}, r^1_{\mathcal{D}}, \ldots, r^i_{\mathcal{D}})$, we can compute secret states $(s'_1, \ldots s'_i)$. We then sample the distribution Y_i by computing the public state generated by activating $\mathcal{D}$ on state (s'_i, p_i) with a uniform random string.

We use g to test the statistical distance between Y_i and D^s_i, where $s \in S_{init}$ was generated by sampling the distribution G_i. To do this, observe that by sampling a public state p_i from the distribution Y_i and then inverting g on $(1^n, i, s, p_0, p_1, \ldots, p_i, 0 \ldots 0)$ we get $(1^n, i, b, r_{\mathcal{G}}, r_f, r^1_{\mathcal{D}}, \ldots, r^{q(n)}_{\mathcal{D}})$.

[5] M_f must be used in the computation of g because the initial secret state s'_0 (sampled by the distribution G_i) cannot be generated in probabilistic polynomial time unless f can be inverted! It is worth noting that this technicality could be avoided by using the (stronger) assumption that auxiliary-input one-way function do not exist (in which s'_0 could be used as the auxiliary input)

Now, from the same argument used in the proof of Theorem 4.1, we observe that if $\Delta(Y_i, D_s^i) \geq \frac{\varepsilon}{2}$ then $b = 1$ with probability at least $\frac{1}{2} + \frac{\varepsilon^2}{16}$. On the other hand, if $\Delta(Y_i, D_s^i) \leq \frac{\varepsilon^2}{36}$ then clearly the probability that $b = 0$ is nearly equal to the probability that $b = 1$ (more concretely $Pr[b = 1] \leq \frac{1}{2} + \frac{\varepsilon^2}{36}$). By repeating this test many times (generating many p_i's) we can distinguish between the two cases! We use a sub-procedure $far - from - mean(s)$ to run this distance test.

Finally, to estimate the weight (by G_i) of initial secret states $s \in S_{init}$ such that D_s^i is close to Y_i, we draw samples $s \sim G_i$ and then test whether $\vec{d_s}$ is statistically close to $\vec{y}$.

The *concentrated-around-mean* procedure:

1. For $j \leftarrow 1 \ldots \ell$, where $\ell = 200\frac{\log 4q(n)}{\delta^2} \log \frac{1}{\delta}$:
   (a) Use $\mathcal{M}_f$ to sample $s \sim G_i$
   (b) Run *far-from-mean(s).* If *far-from-mean(s)* accepts then $a_j = 0$, otherwise $a_j = 1$.
2. If $\frac{1}{\ell}\sum_{j=1}^{\ell} a_j \geq \frac{\delta}{4}$ then reject, otherwise accept.

The *far-from-mean(s)* procedure:

1. For $k \leftarrow 1 \ldots m$, where $m = \frac{200}{\varepsilon^4}(\log \frac{1}{\delta} + 4)$:
   (a) Use $\mathcal{M}_f$ to sample $p_{i+1}^k \sim Y_i$.
   (b) Use $\mathcal{M}_g$ to obtain:
   $(1^n, i, b^k, r_{\mathcal{G}}, r_f, r_{\mathcal{D}}^1, \ldots, r_{\mathcal{D}}^{q(n)}) = \mathcal{M}_g(1^n, i+1, s, p_0, p_1, \ldots, p_i, p_{i+1}^k, 0 \ldots 0)$
2. If $\frac{1}{m}\sum_{k=1}^{m} b^k \geq \frac{1}{2} + \frac{\varepsilon^2}{22}$ then accept, otherwise reject.

Claim 4.3. *The* far-from-mean(s) *procedure has the following properties:*

1. *If $\Delta(\vec{d_s}, \vec{y}) \geq \frac{\varepsilon}{2}$ then the probability that* far-from-mean(s) *accepts is at most $\frac{\delta}{10}$.*
2. *If $\Delta(\vec{d_s}, \vec{y}) \leq \frac{\varepsilon^2}{36}$ then the probability that* estimate-dist(s) *rejects is at most $\frac{\delta}{10}$.*

Proof. We consider two distributions on the input to $\mathcal{M}_g$, which is:

$$(1^n, i+1, s, p_0, p_1, \ldots, p_i, p_{i+1}^k)$$

In the first distribution D, p_{i+1}^k is distributed by $\vec{y}$ (Y_i), and in the second distribution F, p_{i+1}^k is drawn by $\vec{d_s}$ (D_s^i). Following the proof of Theorem 4.1, we look at these distributions as functions on random coins. The function g (with the given public outcomes) computes the distribution F when its input b is 1, and the distribution F when its input b is 0.

The inputs on which we invert g with i (by using $\mathcal{M}_g$) are selected uniformly and at random from the inputs for which $b = 1$. If $\Delta(\vec{d_s}, \vec{y}) \geq \frac{\varepsilon}{2}$ then by Claim 4.2 of Theorem 4.1, for any iteration k the probability that $b^k = 1$ is at least $\frac{1}{2} + \frac{\varepsilon^2}{16}$. We run this experiment $m = \frac{200}{\varepsilon^4}(\log \frac{1}{\delta} + 4)$ times, and thus the probability that $\frac{1}{2} + \frac{\varepsilon^2}{22}$ or more of the times $b_k = 1$ is (by the Chernoff Bound) at least $1 - \frac{\delta}{10}$.

If $\Delta(\vec{d_s}, \vec{y}) \leq \frac{\varepsilon^2}{36}$, then in all iterations no $\mathcal{M}_g$, that chooses b_k by taking a random inverse of g on randomly selected inputs with $b_k = 1$, can have a greater than $\frac{\varepsilon^2}{36}$-advantage (over $\frac{1}{2}$) when guessing b_k. We run this experiment $m = \frac{200}{\varepsilon^4}(\log \frac{1}{\delta} + 4)$ times, the probability that $\frac{1}{2} + \frac{\varepsilon^2}{22}$ or more of the times $b_k = 1$ is (by the Chernoff Bound) at most $\frac{\delta}{10}$. $\square$

Claim 4.4. *The* concentrated-around-mean *procedure has the following properties:*

1. *If* $Pr_{G_i}[B_{\vec{y}}^{\frac{\varepsilon}{2}}] < 1 - \frac{\delta}{2}$ *then* concentrated-around-mean *rejects with probability at least* $1 - \frac{\delta}{4q(n)}$.
2. *If* $Pr_{G_i}[B_{\vec{y}}^{\frac{\varepsilon^2}{36}}] > 1 - \frac{\delta}{10}$ *then* concentrated-around-mean *accepts with probability at least* $1 - \frac{\delta}{4q(n)}$.

Proof. If $Pr_{G_i}[B_{\vec{y}}^{\frac{\varepsilon}{2}}] < 1 - \frac{\delta}{2}$ then in every iteration with probability at least $\frac{\delta}{2}$ we get that $\Delta(\vec{d_s}, \vec{y}) \geq \frac{\varepsilon}{2}$ and with probability at least $1 - \frac{\delta}{10}$ *far-from-mean* rejects such s's. We conclude that in every iteration $a_j = 1$ with probability at least $\frac{\delta}{2} - \frac{\delta}{10} > \frac{\delta}{3}$. By the Chernoff Bound, taking $\ell = 200\frac{\log 4q(n)}{\delta^2} \log \frac{1}{\delta}$, the probability that $\frac{1}{\ell}\sum_{j=1}^{\ell} a_j < \frac{\delta}{4}$ is at most $\frac{\delta}{4q(n)}$.

If $Pr_{G_i}[B_{\vec{y}}^{\frac{\varepsilon^2}{36}}] > 1 - \frac{\delta}{10}$ then in every iteration with probability at most $\frac{\delta}{10}$ we get $\Delta(\vec{d_s}, \vec{y}) > \frac{\varepsilon^2}{36}$, otherwise $\Delta(\vec{d_s}, \vec{y}) \leq \frac{\varepsilon^2}{36}$ and *far-from-mean* rejects with probability at most $\frac{\delta}{10}$. We get (by the Union Bound) that the total probability that $a_j = 1$ in any iteration is at most $\frac{2\delta}{10}$. By the Chernoff Bound, taking $\ell = 200\frac{\log 4q(n)}{\delta^2} \log \frac{1}{\delta}$, the probability that $\frac{1}{\ell}\sum_{j=1}^{\ell} a_j \geq \frac{\delta}{4}$ is at most $\frac{\delta}{4q(n)}$. $\square$

The *concentrated-around-mean* procedure is run at most $q(n)$ times, so by the union bound the probability that there exists a round in which *concentrated-around-mean* "fails" is at most $\frac{\delta}{4}$ (we say *concentrated-around-mean* fails if it accepts when $Pr_{G_i}[B_{\vec{y}}^{\frac{\varepsilon}{2}}] < 1 - \frac{\delta}{2}$ or rejects when $Pr_{G_i}[B_{\vec{y}}^{\frac{\varepsilon^2}{36}}] > 1 - \frac{\delta}{10}$).

Correctness when *concentrated-around-mean* accepts: The algorithm $\mathcal{L}$ runs the *concentrated-around-mean* procedure in its second step. If *concentrated-around-mean* accepts in round i, then the algorithm samples $h \sim G_i$ and returns h. If *concentrated-around-mean* never failed, then when it accepts we know that $Pr_{G_i}[B_{\vec{y}}^{\frac{\varepsilon}{2}}] \geq 1 - \frac{\delta}{2}$. Following the arguments used in the proof of Theorem 3.1, there exists a set $A \subseteq S_{init}$ such that $G_i[A]$ is at least $1 - \frac{\delta}{2}$ and when the algorithm selects $h \sim G_i$, with probability at least $1 - \frac{\delta}{2}$ we know that $\Delta(D_{s_0}^i, D_h^i) \leq \varepsilon$.

The *concentrated-around-mean* procedure eventually accepts with high probability: We would now like to show that with high probability, $\mathcal{L}$ does not terminate in its third step. To see this, note that if *concentrated-around-mean* never failed then whenever it rejected it is known that $Pr_{G_i}[B_{\vec{y}}^{\frac{\varepsilon^2}{36}}] \leq 1 - \frac{\delta}{10}$. Thus there exists a set of initial secret states of weight at least $\frac{\delta}{10}$, and for any s in this set, the distribution D_s^i is $\frac{\varepsilon^2}{36}$-far from $\vec{y}$. By the argument used in the proof of Lemma 3.1, this implies an expected drop of $\frac{\delta}{80} \cdot \frac{\varepsilon^4}{36^2}$ in the entropy of X. By Theorem B.1, within $q(n) = \frac{8 \cdot 80 \cdot 36^2 \cdot \log |S_{init}|}{\delta^2 \cdot \varepsilon^4}$ rounds, with probability at least $1 - \frac{\delta}{8}$, the entropy becomes 0 and then in the next *concentrated-around-mean* will accept (unless it fails).

The algorithm's success probability: We conclude that the probability that $\mathcal{L}$ fails is at most $\frac{\delta}{4}+\frac{\delta}{2}+\frac{\delta}{8}=\frac{7\delta}{8}$. This is the error probability assuming that $\mathcal{M}_f$ and $\mathcal{M}_g$ invert f and g perfectly, but $\mathcal{M}_f$ and $\mathcal{M}_g$ only come $\frac{\delta}{8\cdot 4\cdot q(n)\cdot \ell\cdot m}$-close to inverting their respective functions. This is sufficient, since $\mathcal{L}$ only calls $\mathcal{M}_f$ and M_g at most $1+q(n)\cdot\ell\cdot(1+3\cdot m)\leq 4\cdot q(n)\cdot\ell\cdot m$ times (one call when *concentrated-around-mean* succeeds, $q(n)$ calls to *concentrated-around-mean*, each of them making ℓ calls to $\mathcal{M}_f$ and ℓ calls to *far-from-mean*, each call to *far-from-mean* makes m calls ro $\mathcal{M}_f$ and m calls to $\mathcal{M}_g$, each call to $\mathcal{M}_g$ makes a single call to $\mathcal{M}_f$). Thus the total accumulated error from using $\mathcal{M}_f$ and $\mathcal{M}_g$ instead of "perfect" inverting algorithms is at most $\frac{\delta}{8}$ and we conclude that for infinitely many n's, with probability at least $1-\delta$, $\mathcal{L}$ learns h such that $\Delta(D^i_{s_0}, D^i_h)\leq\varepsilon$. $\square$

5 Acknowledgements

We thank Oded Goldreich, Sofya Raskhodnikova, Ran Raz, Omer Reingold, Dana Ron, Uri Rothblum and Adam Smith for helpful discussions.

References

[1] Naoki Abe and Manfred K. Warmuth. "On the Computational Complexity of Approximating Distributions by Probabilistic Automata". *Machine Learning* 9: 205-260 (1992)

[2] L. E. Baum. "An inequality and associated maximization technique in statistical estimation of probabilistic functions of a Markov process". *Inequalities* 627(3):1-8, 1972.

[3] L. E. Baum, T. Petrie, G. Soules and N. Weiss. "A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains". *Annals of Mathematical Statistics* 41, 164-171. (1970)

[4] Amos Beimel, Yuval Ishai, Eyal Kushilevitz, Tal Malkin. "One-Way Functions Are Essential for Single-Server Private Information Retrieval" *STOC 1999*. 89-98

[5] Manuel Blum and Silvio Micali. "How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits". *SIAM Journal on Computing* 13(4): 850-864 (1984)

[6] Thomas M. Cover and Joy Thomas. "Elements of Information Theory". *John Wiley & Sons, New York, NY.* (1991)

[7] Giovanni Di Crescenzo, Tal Malkin, Rafail Ostrovsky. "Single Database Private Information Retrieval Implies Oblivious Transfer". *EUROCRYPT 2000* 122-138

[8] David Gillman and Michael Sipser: "Inference and Minimization of Hidden Markov Chains". *COLT 1994* 147-158

[9] Oded Goldreich. "A Note on Computational Indistinguishability". *Information Processing Letters* 34(6): 277-281 (1990)

[10] Oded Goldreich, Shafi Goldwasser and Silvio Micali. "How to construct random functions". *Journal of the ACM* 33(4): 792-807 (1986)

[11] Oded Goldreich and Salil P. Vadhan. "Comparing Entropies in Statistical Zero Knowledge with Applications to the Structure of SZK". *IEEE Conference on Computational Complexity 1999* 54-

[12] Shafi Goldwasser and Silvio Micali. "Probabilistic Encryption and How to Play Mental Poker Keeping Secret All Partial Information" *STOC 1982*. 365-377

[13] Shafi Goldwasser, Silvio Micali and Charles Rackoff. "The Knowledge Complexity of Interactive Proof Systems". *SIAM Journal on Computing* 18(1): 186-208 (1989)

[14] Johan Håstad, Russell Impagliazzo, Leonid A. Levin and Michael Luby. "A Pseudorandom Generator from any One-way Function". *SIAM Journal on Computing* 28(4): 1364-1396 (1999)

[15] David P. Helmbold and Philip M. Long. "Tracking Drifting Concepts By Minimizing Disagreements". *Machine Learning* 14(1): 27-45 (1994)

[16] Russell Impagliazzo: "A Personal View of Average-Case Complexity". *Structure in Complexity Theory Conference 1995* 134-147

[17] Russell Impagliazzo and Leonid A. Levin: "No Better Ways to Generate Hard NP Instances than Picking Uniformly at Random". *FOCS 1990* 812-821

[18] Russell Impagliazzo and Michael Luby. "One-way Functions are Essential for Complexity Based Cryptography". *FOCS 1989* 230-235

[19] Richard M. Karp. "Probabilistic Recurrence Relations". *Journal of the ACM* 41(6): 1136-1150 (1994)

[20] Michael J. Kearns, Yishay Mansour, Dana Ron, Ronitt Rubinfeld, Robert E. Schapire and Linda Sellie. "On the learnability of discrete distributions". *STOC 1994* 273-282

[21] Michael J. Kearns, Robert E. Schapire. "Efficient Distribution-Free Learning of Probabilistic Concepts". *Journal of Computer and System Sciences* 48(3): 464-497 (1994)

[22] Michael J. Kearns and Umesh V. Vazirani. "An Introduction to Computational Learning Theory". MIT Press. (1994)

[23] Ueli M. Maurer. "Authentication theory and hypothesis testing". *IEEE Transactions on Information Theory* 46(4): 1350-1356. (2000)

[24] John F. Monahan. "Numerical Methods of Statistics". *Cambridge University Press.* (2001)

[25] Moni Naor and Guy Rothblum. "The Complexity of Online Memory Checking". *Manuscript* (2005) available from authors' webpages

[26] Rafail Ostrovsky, Avi Wigderson. "One-Way Fuctions are Essential for Non-Trivial Zero-Knowledge". *ISTCS 1993* 3-17

[27] L. R. Rabiner. "A tutorial on hidden Markov models and selected applications in speech recognition". *Proceedings of the IEEE* 77(2): 257-286. (1989)

[28] Dana Ron, Yoram Singer, Naftali Tishby. "The Power of Amnesia: Learning Probabilistic Automata with Variable Memory Length". *Machine Learning* 25(2-3): 117-149 (1996)

[29] Claude E. Shannon. "A mathematical theory of communication". *Bell System Technical Journal* 27: 379-423 and 623-656 (1948)

[30] Claude E. Shannon. "Communication Theory of Secrecy Systems", *Bell System Technical Journal* 28: 656-715. (1949)

[31] Gustavus J. Simmons. "Authentication Theory/Coding Theory". *CRYPTO 1984* 411-431

[32] Leslie G. Valiant. "A Theory of the Learnable". *Communications of the ACM* 27(11): 1134-1142 (1984)

[33] Andrew Chi-Chih Yao. "Theory and Applications of Trapdoor Functions (Extended Abstract)". *FOCS 1982 80-91*

A A Lower Bound

In this section we sketch a lower bound for the sample complexity of algorithms for learning ACDs. The lower bound shows that any algorithm for learning ACDs (regardless of its computational strength) must have sample complexity that is at least linear in n, in $\frac{1}{\varepsilon}$ and (more surprisingly) in $\frac{1}{\delta}$. In particular, this implies that there is no efficient amplification for the confidence of algorithms for learning ACDs.

Theorem A.1. *There exists an ACD $(\mathcal{G}, \mathcal{D})$ such that any algorithm that $(\delta(n), \varepsilon(n))$-learns $(\mathcal{G}, \mathcal{D})$ must activate $\mathcal{D}$ at least $\Omega(\frac{1}{\delta(n)} + \frac{\log |S_{init}|}{\varepsilon(n)})$ times.*

Proof Sketch. For any $\varepsilon(n)$, there exists an ACD such that any algorithm that $(\frac{1}{2}, \varepsilon(n))$-learns it must activate $\mathcal{D}$ at least $\Omega(\frac{\log |S_{init}|}{\varepsilon})$ times (this lower bound holds even for static distributions).

The more interesting part of this theorem is showing a lower bound of $\Omega(\frac{1}{\delta(n)})$ activations of $\mathcal{D}$. Let $\varepsilon = \varepsilon(n)$, and $\delta = \delta(n)$, and take the concept class S_{init} to be the set $\{0,1\}^n$ ($\log |S_{init}| = n$). The generating algorithm generates an initial secret state selected uniformly and at random from $\{0,1\}^n$ (the initial public state is always 0). The sampling algorithm $\mathcal{D}$ is deterministic, and it never alters the secret state. Also, the number of times $\mathcal{D}$ has been activated is always part of the public state.

In each round some of the possible initial secret states will become unviable given the public state. We take S^i_{init} to be the set of initial secret states that are still viable given all the public information before round i (thus $S^1_{init} = C$). In round i, if the secret state is unviable then $\mathcal{D}$ outputs a special symbol (say #). Otherwise, if the secret state is any of the $2\delta \cdot |S^i_{init}|$ (lexicographically) first viable initial secret states in S^i_{init}, then $\mathcal{D}$ always outputs 1 and the secret state, otherwise $\mathcal{D}$ outputs 0. If $\mathcal{D}$'s output was 0 then the secret state must be one of the $(1-2\delta) \cdot |S^i_{init}|$ lexicographically last secret states in S^i_{init}, the set of these secret states is S^{i+1}_{init}.

The intuition is that as long as the learning algorithm only sees 0 outputs it cannot have a high probability of successfully imitating the distribution of the next output. With high probability the learning algorithm will only see 0's for at least $\Omega(\frac{1}{\delta})$ rounds, and thus it cannot successfully learn with high probability in less than $\Omega(\frac{1}{\delta})$ rounds. This is captured in the following two claims:

Claim A.1. *As long as $\mathcal{D}$ has only given 0's as output, for any $\varepsilon < 1$, no learning algorithm can come ε-close to generating the distribution of $\mathcal{D}$'s next output with probability greater than $1-2\delta$.*

Claim A.2. *For $\delta < \frac{1}{3}$, with probability at least $\frac{2}{3}$, $\mathcal{D}$ will output 0 in the first $\Omega(\frac{1}{\delta})$ rounds.*

We conclude that any algorithm for learning the ACD with confidence parameter $\delta < \frac{1}{3}$ must activate $\mathcal{D}$ at least $\Omega(\frac{1}{\delta})$ times.

$\square$

B Probabilistic Recurrence

In this appendix we are concerned with analyzing probabilistic algorithms that use a divide-and-conquer strategy to reduce the size of a problem they are trying to solve. This setting arises throughout the computer science literature, as was noted by Karp [19], who presented a generalized methodology for analyzing such algorithms. Karp, however, assumed that the algorithms never increase the size of the problem they work on. Our learning algorithms, presented in Section 3, treat the entropy of an unknown random variable as the "problem size" and try to reduce it. We show that after every step these algorithms make there is a high *expected* entropy-drop, but there is no guarantee that the entropy cannot increase! Thus, Karp's results cannot be immediately used in our setting. We will prove a theorem that shows that even if the problem size may increase, as long as the expected drop in the problem size is high, it will not take too long for the algorithm to reduce the problem to size 0 (with high probability). We treat the case where the expected drop in the size of the problem is constant, and follow [19] in using induction to show a tail inequality.

Theorem B.1. *Let s be a size function over problem instances and $T(x) = 1 + T(h(x))$ a probabilistic recurrence relation, where $T(0) = 0$ and $h(x)$ is a random variable with $E[s(h(y))] \leq s(y) - d$ (for any problem instance y). Then for any positive integer t:*

$$Pr[T(x) \geq t] \leq \frac{s(x)}{t \cdot d}$$

Proof. The proof will be by induction on t, for $t = 1$: either $s(x) \leq d$, in which case $T(x) \leq 1$, or $s(x) > d$ and the claim is trivial.

We assume correctness for t, and prove it for $t+1$: if $s(x) \geq (t+1) \cdot d$ then the claim is trivially true. For $s(x) < (t+1) \cdot d$ we get:

$$\begin{aligned}
Pr[T(x) \geq t+1] &= E_{h(x)}[Pr[T(h(x)) \geq t]] \\
&\leq E_{h(x)}\left[\frac{s(h(x))}{t \cdot d}\right] \\
&= \frac{E_{h(x)}\left[s(h(x))\right]}{t \cdot d} \\
&\leq \frac{s(x) - d}{t \cdot d} \\
&< \frac{s(x)}{(t+1) \cdot d}
\end{aligned}$$

□