The Complexity of Online Memory Checking *

Moni Naor† Guy N. Rothblum‡

Abstract

We consider the problem of storing a large file on a remote and unreliable server. To verify that the file has not been corrupted, a user could store a small private (randomized) "fingerprint" on his own computer. This is the setting for the well-studied authentication problem in cryptography, and the required fingerprint size is well understood. We study the problem of sub-linear authentication: suppose the user would like to encode and store the file in a way that allows him to verify that it has not been corrupted, but without reading the entire file. If the user only wants to read q bits of the file, how large does the size s of the private fingerprint need to be? We define this problem formally, and show a tight lower bound on the relationship between s and q when the adversary is not computationally bounded, namely: $s \times q = \Omega(n)$, where n is the file size. This is an easier case of the online memory checking problem, introduced by Blum *et al.* in 1991, and hence the same (tight) lower bound applies also to that problem.

It was previously shown that when the adversary is computationally bounded, under the assumption that one-way functions exist, it is possible to construct much better online memory checkers. T he same is also true for sub-linear authentication schemes. We show that the existence of one-way functions is also a necessary condition: even slightly breaking the $s \times q = \Omega(n)$ lower bound in a computational setting implies the existence of one-way functions.

1 Introduction

The problem of memory checking was introduced by Blum *et al.* [BEG+94] as a natural extension of the notion of program checking of Blum and Kannan [BK95]. A memory checker receives from its user a sequence of "store" and "retrieve" operations to a large unreliable public memory. The checker also receives responses to these requests and may itself make additional requests to the unreliable memory. It uses the additional requests together with a small private and reliable memory to ascertain that all requests were answered correctly. The checker's assertion should be correct with high probability (a small two-sided error is permitted). Blum *et al.* made the distinction between online and offline memory checking. An *offline* checker may run the entire sequence of requests before outputting whether they were all handled correctly. An *online* checker must detect errors immediately after receiving an errant response to a "retrieve" request. The main complexity measures of a memory checker are its *space complexity*, the size of the secret reliable

*An extended abstract appeared in *Proceedings of the 46th Annual Symposium on Foundations of Computer Science* (FOCS 2005). This is the full version.

†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 by a grant from the Israel Science Foundation.

‡CSAIL, MIT, Cambridge MA 02139, USA. E-mail: rothblum@csail.mit.edu. This work was done while the author was at the Weizmann Institute of Science, Rehovot, Israel.

memory, and its *query complexity*, the number of queries made into the unreliable memory per user request.

We want memory checkers to work for any possible sequence of user requests and for any behavior of the unreliable memory. Thus we think of the user and the unreliable memory as being controlled by a malicious adversary. In the information-theoretic setting, a memory checker is required to be secure versus a computationally unbounded adversary. In the computational setting we relax this requirement to security versus probabilistic polynomial time adversaries.

A related issue introduced in this work is the problem of *sub-linear* authentication: authenticating a message without reading all of its bits. An authenticator is a cryptographic object that receives a large file from a user, encodes it into a large but unreliable public memory, and secretly stores a small reliable fingerprint. The authenticator can then be repeatedly used to verify that the encoding it stored has not been corrupted beyond recovery, i.e. that the entire original file can be recovered by a decoding procedure. If the encoding is completely uncorrupted, then it should be accepted with high probability, but if it is so corrupted that decoding the original file is impossible, then it should be rejected with high probability. Thus it is guaranteed that whenever the authenticator accepts, with high probability the *entire* original file can be reconstructed. The point is that the authenticator does not read the entire encoding that is stored in the unreliable memory, but only a few bits of it. Recently, and following this work, there has been growing interest in sub-linear authentication and in related questions about proving retrievability. See Section 1.2 for more on these related works. Authenticators check the consistency of the entire encoding, but are not required to decode any single bit of the file. Memory checkers, on the other hand, are required to recover and verify small pieces (usually bits) of the file. The main complexity measures we define for authenticators (similarly to memory checkers) are *space complexity*, the size of the secret reliable fingerprint, and *query complexity*, the number of bits read from the unreliable memory per verification.

It may seem that recent advances in local testing and correction of codes (e.g. [KT00], [FS95], [GS06]) could enable the construction of better online memory checkers and authenticators. As we show in this work, this hope turns out to be false. The main difference between the setting of testable and correctable codes and the settings of memory checking and sub-linear authentication, is that there is no bound on the number of errors that may be introduced into the unreliable memory. In particular, in our setting we want to detect even a malicious adversary who can modify the *entire* unreliable memory, whereas in the coding setting usually there is some bound assumed on the number of errors an adversary can introduce. In particular, the modified unreliable memory may be far from the original encoding, but close to a legitimate encoding of a different file. As we show in this work, handling an unbounded number of errors incurs a significant overhead in terms of the space and query complexities. Another direction that may raise false hopes is using PCPs of proximity (see [BSGH$^+$04], [BSS05]), but this approach encounters similar difficulties.

1.1 Our Results

In this work we quantify the complexity of online memory checking and sub-linear authentication. We focus on the tradeoff between the space and query complexities of online memory checkers and authenticators, and show tight lower bounds for this tradeoff in the information theoretic setting. When the adversary is computationally bounded, it was previously shown [BEG$^+$94] that very good online memory checkers can be constructed under the assumption that one-way functions exist. The same holds also for authenticators (see Section 4.2). We show that the existence of one-

way functions is an *essential condition* for breaking the lower bound in a computational setting.

We first show a connection between memory checking and sub-linear authentication. We show that any online memory checker can be used to construct an authenticator with similar space and query complexities. The idea behind the reduction is simple. Loosely speaking, to construct an authenticator, encode a file using a "good" error-correcting code (resilient to a constant fraction of errors), and then store it using the memory checker. To corrupt the encoding, an adversary must corrupt a constant fraction of its locations (by the properties of the error correcting code). Thus, to verify that the data hasn't been corrupted too much, the authenticator chooses at random $O(1)$ locations in the public encoding, and verifies that they have not been corrupted using the memory checker. The authenticator's space and query complexities are directly inherited from the memory checker's, see Section 4.1 for the details.

Throughout this work, lower bounds are shown for authenticators. By the above reduction, they also apply to the (more difficult) task of online memory checking. Let s and q be the space and query complexities of any authenticator. When the adversary is computationally unbounded we prove that $\mathbf{s} \times \mathbf{q} = \mathbf{\Omega(n)}$, where n is the size of the stored file. The same lower bound also applies to online memory checkers, and it relies only on the fact that the online memory checker's query complexity *for retrieve requests* is at most q (the query complexity for *store requests* may be unbounded). Corresponding upper bounds can be achieved by splitting the file into chunks of size q and storing a global hash function and the hash value of each chunk in the private memory (see Section 3.1.1 for an upper bound for online memory checkers, and Section 4.2 for an authenticator upper bounds).

When all parties run in probabilistic polynomial time, we show that the existence of an authenticator (or online memory checker) that breaks the lower bound (i.e. has $s \times q \leq d \cdot n$ for some constant $d > 0$) implies the existence of one-way functions (that are hard to invert for infinitely many input lengths: *"almost one-way functions"*).

In both results we use the simultaneous messages (SM) model of communication complexity, suggested by Yao in his seminal paper on communication complexity [Yao79], and especially a lower bound of Babai and Kimmel [BK97] (who generalized the lower bound of Newman and Szegedy [NS96]) for the communication complexity of SM protocols for equality testing. To the best of our knowledge, the only prior application of the SM model outside the field of communication complexity was for a positive result (a PIR construction) in the work of Beimel, Ishai and Kushilevitz [BIK05]. We define a communication complexity setting inspired by the SM model: consecutive messages (CM) protocols. We generalize the lower bound of Babai and Kimmel to the CM model, and then present a reduction from any online memory checker (or authenticator) to a CM protocol for equality testing. In the computational setting, we show that if one-way functions do not exist then the CM lower bound cannot be broken and our reduction from online memory checkers (and authenticators) to a CM protocol can be made efficient. We also use results on learning adaptively changing distributions (see Naor and Rothblum [NR06], the companion paper to this work) to show that the adversary can learn the distribution of queries to the unreliable memory that an authenticator will make.

1.2 Related Work

Authentication. Authentication is one of the major issues in Cryptography. An *authentication protocol* is a procedure by which a transmitter (Alice) tries to convey an n-bit long input message to a receiver (Bob). A malicious adversary controls the parties' communication channel. The

requirement is that if the adversary changes the message sent by Alice, then Bob must detect the change with high probability. Alice and Bob are allowed to share a small secret random string. Much attention has been given to protocols that are unconditionally secure and make no cryptographic assumptions, with a focus on quantifying the number of random bits that Alice and Bob must share. The problem was first considered by Gilbert, MacWilliams and Sloane [GMS74], who presented a solution where the number of secret bits shared by Alice and Bob was linear in the message size. Wegman and Carter [WC81] were the first to suggest using *almost strongly universal*$_2$ hash functions to achieve much better parameters. It was shown by Gemmel and Naor [GN93] that if the adversary has at most probability ε of fooling Bob then $\Theta(\log n + \log \frac{1}{\varepsilon})$ secret shared random bits are both necessary and sufficient. We note that any memory checker can be used to construct an authentication protocol, where the number of secret bits Alice and Bob share is on the same order of magnitude as the size of the memory checker's private reliable memory.

Sublinear Authentication. In recent years and following this work, the problem of sub-linear authentication has gained even more importance for real-world applications. Large databases are increasingly being outsourced to untrusted storage providers, and this is happening even with medical or other databases where reliability is crucial. Another wide-spread and growing phenomenon are services that offer individual users huge and growing remote storage capacities (e.g. webmail providers, social networks, repositories of digital photographs etc.). In all of these applications it is important to guarantee the integrity of the remotely stored data. See for example the more recent works of Ateniese *et al.* [ABC$^+$07], Clarke *et al.* [CSG$^+$05], Juels and Kaliski [JK07], Shacham and Waters [SW08], and Bowers, Juels and Oprea [BJO08] on verifying the integrity of data and "proofs of retrievability" (notions that are very similar to the authenticators introduced in this work).

Memory Checking. Offline memory checkers are checkers with relaxed soundness: they are only required to detect the presence of bugs somewhere in a long request sequence. Offline checkers are relatively well understood. Blum *et al.* [BEG$^+$94] showed an offline memory checker with amortized constant query complexity (over long operation sequences), using $O(\log n + \log 1/\varepsilon)$ bits of private memory, where n is the size of the file the user wants to store, and ε is the probability of error. This was improved by Dwork, Naor, Rothblum and Vaikuntanathan who showed how to get the same amortized query complexity for *any* (even short) operation sequences.

Blum *et al.* [BEG$^+$94] also showed that an offline memory checker must use at least $\Omega(\log n)$ bits of private memory. A tight space complexity lower bound of $\Omega(\log n + \log 1/\varepsilon)$ was presented by Gemmell and Naor [GN93]. The lower bound of [GN93] was shown in the model of message authentication codes, which is closely related to memory checking. Blum *et al.* made the distinction between invasive and noninvasive memory checkers. Checkers that use the unreliable memory to store information other than that requested by the user are called *invasive*, an invasive checker may make its own store requests to the public unreliable memory (beyond those requests made by the user), and store information that is much larger than the user's file. For example, an invasive checker could store in the unreliable memory hash values of the bits of the user file, or even the time when each bit was last modified. *Noninvasive* memory checkers are more restricted, a noninvasive memory checker can only store the user file in the public memory, and is not allowed to make any store requests beyond those made by the user. Ajtai [Ajt02] studied the complexity of *noninvasive offline* memory checking, and showed that any noninvasive memory checker with

logarithmic memory must have polynomial query complexity. Furthermore, his lower bound holds even for non-invasive offline checkers that are only secure against polynomial-time adversaries.

Online memory checkers are not as well-understood as offline checkers. Blum *et al.* [BEG+94] presented two online invasive memory checkers secure against polynomial-time adversaries. These memory checkers assume the existence of one-way functions. Both schemes use secret memory of size κ, where κ is the size of a secret key that allows cryptography (say a small polynomial in n), with query complexity $O(\log n)$. One of these schemes, based on UOWHFs (Naor and Yung [NY89]), only requires that the checker's memory be reliable (and not necessarily private). Dwork *et al.* [DNRV08] revisited the query complexity of computationally secure non-adaptive online checkers. They show that the query complexity of such checkers must be $\Omega(\log n / \log \log n)$, and also show how to trade-off the query complexities of store and retrieve operations. For information-theoretically secure *non-invasive* online checkers, Blum *et al.* [BEG+94] showed a tradeoff between their space and query complexities: their product is $\Omega(n)$. No non-trivial lower bound was known for general (i.e. *invasive*) online memory checkers.

The problem of memory checking is related to other program checking problems. The question of checking and hiding a sequence of store and retrieve requests to a random access memory was addressed by the papers of Goldreich and Ostrovsky ([Gol87], [Ost90] and [GO96]). These papers address the more difficult problem of software protection (in the computational setting), and consequently the solutions incur a larger overhead.

Incremental Cryptography. Incremental Cryptography was introduced by Bellare, Goldreich and Goldwasser [BGG94], who explored the efficiency of repeatedly applying cryptographic transformations (e.g. authentication, encryption) to a document as it changes. The goal is that the cost of re-applying the cryptographic transformation be proportional to the number of changes made to the document. In a subsequent work [BGG95] they introduced the notion of tamper-proof security for incremental authentication, which is closely related to online memory checking. In the tamper-proof incremental authentication setting, a user stores and modifies a large file on an unreliable memory, using a small secret and reliable memory to detect changes made by an adversary to the unreliable memory. The main difference between their setting and the setting of online memory checking is a focus on more powerful text editing operations (e.g. insert, delete, cut paste etc.). Work on incremental cryptography has focused on the computational setting, and no lower bounds were known for information-theoretic tamper-proof incremental cryptography, nor was it previously known whether computational assumptions are essential for the construction of good tamper-proof incremental authentication schemes. Our results apply also to the (more difficult) problem of tamper-proof incremental authentication.

Relationships Between Cryptographic Primitives. An important 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 [IL89], who showed that implementing many central cryptographic tasks implies the existence of one-way functions.

Proving that one-way functions are sufficient 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 [Imp95]). Beyond the pioneering work

of [IL89], well-known results in this vein include Ostrovsky and Wigderson's result [OW93] that one-way functions are essential for non-trivial zero-knowledge, the result of Beimel *et al.* [BIKM99] that one-way functions are essential for non-trivial private information retrieveal (Di Crescenzo *et al.* [CMO00] showed that actually oblivious transfer is essential). In a subsequent paper to this work, Naor, Segev and Smith [NSS06] show that breaking a different information-theoretic authentication bound (even slightly), implies the existence of one-way functions. As mentioned above, [BEG$^+$94] showed that one-way functions imply the existence of good online memory checkers. It was not known, however, whether one-way functions are *essential* for efficient online checking.

We note that, in general, the existence of an information-theoretic lower bound does not imply that breaking the lower bound in a computational setting implies the existence of one-way functions. One such example is breaking information-theoretic bounds for error-correcting codes. Micali, Peikert, Sudan and Wilson [MPSW05] show how to break such bounds in a computational setting. It is not known, in general, whether breaking such bounds implies the existence of one-way functions. Another example is the existence of interactive *argument* systems, such as those of Kilian [Kil92] and Micali [Mic94], for langauges that have no interactive proofs (it is not known whether one-way functions are essential for such results). For the specific case of memory checking, we are able to obtain such a result in this work, and show that breaking the information-theoretic lower bound *does* imply the existence of one-way functions.

The Simultaneous Messages Model. The problem of online memory checking is related to the two-player simultaneous messages (SM) communication model, introduced by Yao [Yao79]. In the SM model two players, Alice and Bob, would like to compute the value of a function f on their inputs x and y. To do this, they each use private random coins to compute a message to a referee, Carol. The objective is for Carol to use Alice's and Bob's messages to output $f(x, y)$ with high probability (w.h.p.), using low communication complexity (the communication complexity is length of Alice's and Bob's messages). If the parties are all deterministic then it is straightforward to determine the communication complexity of any function. The model becomes more interesting if Alice, Bob and Carol are all allowed to use private random coins, but there is no shared randomness (for some functions, e.g. the equality function, if Alice and Bob are allowed to share even a logarithmic number of random bits, then the communication complexity can be made constant).

Yao was interested in the communication complexity of the *equality* function. This question was first addressed by Newman and Szegedy [NS96], who showed that the randomized SM complexity of the equality function on $\{0,1\}^n$ is $\Omega(\sqrt{n})$. Their result was generalized by Babai and Kimmel [BK97], who showed that if d is the deterministic SM complexity of a function for which there exists a randomized SM protocol in which Alice sends messages of length a and Bob sends messages of length b, then $a \times b = \Omega(d)$. A direct corollary is that the order of magnitude of the randomized SM complexity of *any* function is at least the square root of the function's deterministic SM complexity (a similar result was independently shown by Bourgain and Wigderson). Kremer, Nisan and Ron [KNR99] explored the complexity of various variants of SM protocols.

1.3 Organization

We begin with definitions and notation in **Section 2**. Memory checkers, consecutive messages protocols and adaptively changing distributions are defined in **Section 3**, where we also present an upper bound for information-theoretically secure online memory checking. In **Section 4** we introduce authenticators, cryptographic objects used for storing information that allow quick verification

that the information stored has not been corrupted beyond recovery. We show that online memory checkers can be used to construct authenticators. **Section 5** gives a lower bound for authenticators and online memory checkers. The proof is in the form of a reduction from online memory checkers and authenticators to consecutive messages protocols for equality. In **Section 6** we show that the existence of efficient memory checkers in a computational setting implies the existence of (almost) one-way functions.

2 Definitions and Notation

Notation. Let $[n]$ be the set $\{1, 2, \ldots n\}$. For a (discrete) distribution D over a domain X we denote by $x \sim D$ the experiment of selecting $x \in X$ by the distribution D. For $x \in X$, let $D[x]$ be x's probability by D. For a subset $A \subseteq X$ let $D[A]$ be the probability (or weight) of A by D (i.e. $D[A] = \sum_{x \in A} D[x]$). We use U_ℓ to denote the uniform distribution over $\{0,1\}^\ell$.

Distributions and Ensembles. An ensemble $D = \{D_n\}_{n \in N}$ is a sequence of distributions $D_1, D_2, \ldots$. It is polynomial time constructible if there exists a probabilistic polynomial time Turing machine (PPTM) $\mathcal{M}$, such that $\mathcal{M}(1^n)$ generates D_n.

Definition 2.1. The *statistical distance* (or L_1 distance) between two distributions D and F over a domain X, denoted by $\Delta(D, F)$, is defined as:

$$\Delta(D, F) = \frac{1}{2} \sum_{x \in X} |\Pr[D = x] - \Pr[F = x]|$$

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

Two ensembles D and F are $\varepsilon(n)$-statistically **far** if there exists some n_0 such that for all $n \geq n_0$, 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.

One-Way and Distributionally One-Way functions. A function f, with input length $k(n)$ and output length $\ell(n)$, specifies for each $n \in N$ 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 2.3. A function f is a *one-way function* if it is computable in polynomial time, but cannot be inverted by any polynomial-time machine. Namely, 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:

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

Definition 2.4. *Distributionally One-Way Function (Impagliazzo and Luby [IL89]):* A function f is *distributionally one-way* if it is computable in polynomial time, but no polynomial-time machine can *randomly* invert it. Namely, for some constant $c > 0$, for any PPTM $\mathcal{A}$, the two ensembles:

1. $x \circ f(x)$, where $x \sim U_{k(n)}$.
2. $\mathcal{A}(f(x)) \circ f(x)$, where $x \sim U_{k(n)}$.

are $\frac{1}{n^c}$-statistically far (if the ensembles *are not* $\frac{1}{n^c}$-statistically far we say $\mathcal{A}$ 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). Distributionally one-way functions are a weaker primitive than one-way functions in the sense that any one-way function is also distributionally one-way, but the converse may not be true. Despite this, the *existence* of both primitives is equivalent:

Lemma 2.1. *(Impagliazzo and Luby [IL89]) 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 2.5. 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:

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

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 function is equivalent to that of almost distributionally one-way functions (the proof follows the proof of Lemma 2.1, see [IL89]).

3 Memory Checkers, CM Protocols and ACDs

In this section we define and state properties of some of the central primitives used in this work. These include memory checkers, simultaneous and consecutive messages protocols, and adaptively changing distributions.

3.1 Memory Checkers

Definition 3.1 (Online Memory Checker [BEG+94])**.** An online memory checker is a probabilistic Turing machine $\mathcal{C}$ with five tapes: a read only input tape for reading requests from the user $\mathcal{U}$, a write only output tape for writing responses to the user's requests or that the unreliable memory is buggy, a read-write work tape (the secret reliable memory), a write only tape for writing requests to the unreliable memory $\mathcal{M}$ and a read only input tape for reading $\mathcal{M}$'s responses.

For an n bit file,[1] the user $\mathcal{U}$ writes "store" and "retrieve" requests to the checker's input tape. Each such request specifies an index in the n-bit file and, for store requests, a binary value to be stored. In response to each retrieve requests, the checker writes either a binary output bit or the special symbol $BUGGY$ on its output tape. To answer each such request, the checker sends "read"

[1]The definition is easily extended to memories over non-binary alphabets.

and "write" requests to the public memory $\mathcal{M}$. For each such request, the checker specifies an address in the public memory, and, for store requests, a binary value to be stored. In response to "read" requests, the public memory replies with a binary value. The checker can also modify its (small) work-tape arbitrarily every time it receives a new message.

The checker $\mathcal{C}$'s operation should be secure (both correct and complete) for all user request sequences, and for any public memory behavior. Thus, we consider security versus a malicious adversary $\mathcal{A}$ that controls all messages sent to the checker (namely, controls both $\mathcal{U}$ and $\mathcal{M}$). The adversary gets to see all messages sent (and received) by the checker, including which addresses and values it reads and writes from and to the unreliable memory, but *does not* see the checker's coin tosses or secret memory. For any such adversary $\mathcal{A}$, starting with the unreliable memory set to all 0's, we require:

- Completeness. As long as $\mathcal{A}$ answers all of $\mathcal{C}$'s "read" requests to $\mathcal{M}$ correctly (with the last value stored at that address), $\mathcal{C}$ also answers *each* "retrieve" request from $\mathcal{U}$ correctly, with probability at least 0.95 (for each request, over all of $\mathcal{C}$'s and A's coins).[2]

- Soundness. Even if $\mathcal{A}$ responds incorrectly to a read request $\mathcal{C}$ makes to $\mathcal{M}$, then the probability that $\mathcal{C}$ answers a request from $\mathcal{U}$ incorrectly is at most 0.05 (over $\mathcal{C}$'s and $\mathcal{A}$'s coins). The checker $\mathcal{C}$ may either recover the correct answer independently or output "BUGGY", but it may not answer a request incorrectly (beyond the small error probability).[3]

If security (completeness and soundness) hold, as above, for *any* adversary $\mathcal{A}$ (regardless of its computational power), then the checker is said to be *information-theoretically* or *unconditionally* secure. We also consider *computationally* secure checkers, where security holds only with respect to polynomial-time adversaries. Throughout this work, all memory checkers are unconditionally secure unless we explicitly note otherwise.

The *space complexity* of a checker is the size of its secret memory (its persistent work-tape). The *query complexity* is the the number of requests the checker makes per request made by the user. A checker is *polynomial-time* if $\mathcal{C}$ runs in polynomial (in n) time.

Discussion. We would prefer online memory checkers to have small space complexity and small query complexity. Unfortunately, our main result in this work shows that, in the case of online memory checkers, these two goals collide. The relaxation of *computational* security is a significant one, as one-way functions can be used to construct very good online memory checkers (see [BEG+94]). In this work we show that (almost) one-way functions are an *essential* condition for constructing good computationally secure memory checkers.

Offline Checkers. In this work we focus on *online* memory checkers. As previously noted, [BEG+94] also introduced the notion of *offline* memory checkers. The main difference between online and offline checkers is that an *offline* checker is notified before it receives the last "retrieve" request in a sequence of requests. For soundness, it is only required that if *at some point* in the request sequence $\mathcal{M}$'s answer to some request was incorrect, and $\mathcal{C}$ passed an incorrect answer to $\mathcal{U}$, then at *some*

[2] The success probabilities we use in this work are arbitrary: in an information-theoretic setting memory checkers, authenticators and CM protocols can be amplified by parallel repetition. It is not clear that this is true in a computational setting (see Bellare, Impagliazzo and Naor [BIN97]).

[3] This relaxes the definition of [BEG+94], allowing independent recovery from errors.

point during the sequence of requests $\mathcal{C}$ responds that $\mathcal{M}$'s operation was BUGGY. The offline checker is not required to "pin-point" where in the request-response sequence an error occurred. This is a significantly relaxed requirement, and indeed [BEG+94] (and later [DNRV08]) show how to construct very efficient (amortized) offline memory checkers.

3.1.1 An Unconditionally Secure Online Memory Checker

In this section we present a construction of an information-theoretically secure online memory checker (this construction was implicit in [BEG+94]). The construction gives an upper-bound on the tradeoff between an online memory checker's space complexity s and query complexity q, showing that $s \times q = O(n)$. For constructions of *computationally secure* online memory checkers see [BEG+94].

We begin with a discussion of ε-biased hash function. To build an online memory checker, we first need to construct a family $\mathcal{H}$ of hash functions from $\{0,1\}^q$ to $\{0,1\}^m$ (for some m), with the following parameters:

Collision Probability: For any two $x, y \in \{0,1\}^q$, where $x \neq y$, when a hash function h is selected uniformly and at random from $\mathcal{H}$, the probability that $h(x) = h(y)$ is at most ε.

Output Size: Each function $h \in \mathcal{H}$ is from $\{0,1\}^q$ to $\{0,1\}^m$, where $m = O(\log 1/\varepsilon)$.

Description Size: The description of any function $h \in \mathcal{H}$ is not too large (see below).

Families of ε-biased hash functions are useful in constructing unconditionally secure memory checkers because their collision probability is low for *any* $x, y \in \{0,1\}^q$. Thus even if a computationally unbounded adversary selects x and y, the probability that $h(x) = h(y)$ when h is selected uniformly and at random from $\mathcal{H}$ is still at most ε. We will use a construction of a family of ε-biased hash functions due to Naor and Naor [NN93].[4] They showed (for any q, ε) how to construct a small family $\mathcal{H}$ of functions from $\{0,1\}^q$ to $\{0,1\}^{O(\log 1/\varepsilon)}$, such that the description of any $h \in \mathcal{H}$ is of size $O(\log q + \log \frac{1}{\varepsilon})$ with collision probability ε.

Given such a family of hash functions for $\varepsilon = 0.05$, we construct an online memory checker that is unconditionally secure with query complexity q. The memory checker divides the user's n-bit file into $\frac{n}{q}$ chunks, of q bits each (assume w.l.o.g. that q divides n). The checker will store a global hash function and the hash value of each chunk in the private memory. The user file is saved in public memory, and the memory checker selects a random $h \in \mathcal{H}$, saving it in the secret memory. For every $i \in [\frac{n}{q}]$, let $c_i \in \{0,1\}^q$ be the vector of the i-th chunk's bit values. For every chunk the memory checker saves $h(c_i)$ in the secret memory.

Retrieve: Whenever the user issues a "retrieve" request for a particular index, the memory checker reads the entire chunk that contains that index from the public memory. Suppose the value read was c'_i, the memory checker checks whether $h(c'_i)$ equals the stored hash value $h(c_i)$. If the two hash values are different then the public memory has been modified and the checker replies "BUGGY", otherwise the checker returns the value it read from the public memory for the index the user requested.

[4] We use this family of hash functions because of their small description size, but note that for most desired space and query complexities simpler families (with larger descriptions) could also be used to construct memory checkers with comparable parameters.

Store: When the user issues a "store" request to some index, the memory checker reads the index's entire chunk from the public memory. Suppose the value read was c'_i, the checker checks whether $h(c'_i)$ equals the stored hash value $h(c_i)$. If the two hash values are different then the public memory has been modified and the checker replies "BUGGY", otherwise the checker modifies the value of the chunk in the public memory as per the user's store request, computes a new hash value for the new chunk content, and stores it in the secret memory.

Correctness. The memory checker is secure (both sound and complete). As long as the contents of the public memory are not altered, the memory checker will answer all user queries correctly with probability 1. If an adversary modifies a value stored in some location in the public memory, then the first time the user requests access to some index whose chunk includes that location, with probability at least 0.95 (over the selection of h) the memory checker will detect that the public memory has been altered and output "BUGGY".

Complexity Analysis. For each store and retrieve request made by the user, the memory checker reads the entire chunk of the index accessed, thus the query complexity is q. The space complexity is $O(\log q)$ bits for the hash function h and another $O(1)$ bits per chunk for the stored hash value, a total of $s = O(\log q + \frac{n}{q})$ bits of secret memory. This implies an upper bound tradeoff of $s \times q = O(n)$ (unless the query complexity is significantly higher than $\frac{n}{\log n}$). We will show that this tradeoff is optimal for all values of s and q.

3.2 Communication Complexity Models

Simultaneous Messages Protocols. An SM Protocol for computing a function $f : X \times Y \mapsto \{0,1\}$ (see [Yao79, NS96, BK97]) is a protocol between three players: Alice, Bob and Carol. Alice is given an input $x \in X$, Bob is given an input $y \in Y$. We will work only with simultaneous messages (SM) protocols which are randomized with two-sided error, where each player is allowed to use private random coins. Alice computes an output message m_A, using only x and her private random coins. Bob similarly computes an output message m_B, using only y and his private coins. Carol receives (m_A, m_B) and uses her private coins to compute an output in $\{0,1\}$. We require that for all $(x,y) \in X \times Y$ Carol's output is $f(x,y)$ with probability at least $1-\varepsilon$ for some constant $\varepsilon < \frac{1}{2}$. The probability is taken over Alice's, Bob's and Carol's private coin tosses. ε is the error probability of the protocol. The main complexity measure of an SM protocol is its *communication complexity*, the maximal length of the messages sent by Alice and Bob. SM protocols for equality testing compute the equality function (Carol's output is 1 if $x = y$, and 0 otherwise). Newman and Szegedy [NS96], followed by Babai and Kimmel [BK97], showed that the communication complexity of SM protocols for equality testing is $\Omega(n)$.

Consecutive Messages Protocols. In this work, we introduce the consecutive messages model of communication complexity. A consecutive messages (CM) protocol is also a protocol between 3 players: Alice, Bob and Carol, for computing a function $f : X \times Y \mapsto \{0,1\}$. However, in a CM protocol Alice and Bob do not receive their inputs simultaneously, and thus they do not send their messages simultaneously. The players still send their messages somewhat independently, but they are also allowed a limited amount of shared *public* information. We note that CM protocols can be viewed as a generalization of SM protocols that gives the players more power to interact.

Despite this, we show that the best CM protocol for equality testing requires almost as much communication as the best SM protocol. While the motivation for defining CM protocols may not be immediately self-evident, we will later present a reduction from any online memory checker to a CM protocol for equality testing, and in that setting the players' ability to interact will be both useful and natural.

We view a CM protocol as a game between an adversary and the players (Alice, Bob and Carol), where the adversary selects Alice's and Bob's inputs (x and y respectively). The players' goal is for Carol to output $f(x, y)$ with high probability. The adversary's goal is to select x and y such that the probability that Carol will output $f(x, y)$ is not high. It is important to note that the adversary *cannot* modify any of the messages sent by the players (in this sense, all communication channels are *reliable*). We make the distinction between *private messages*, observable only by some of the players, and *public messages* observable by all the players and by the adversary. The protocol also allows *public random coins*, which are observable by all players and by the adversary. As in SM protocols, the players can use private random coins (Bob and Carol share their coins). We describe the step-by-step execution of a CM protocol:

1. The adversary gives Alice an input $x \in X$.
2. Alice computes a pair of messages (m_A, m_p), functions of x, her private random coins r_A, and the shared randomness r_p. Alice sends m_A to Carol over their private channel and publishes the public message m_p.
3. The adversary sees (x, r_p, m_p) and then selects an input $y \in Y$ for Bob.
4. Bob uses (y, r_p, m_p, r_{BC}) to compute a message m_B to Carol. r_{BC} are Bob and Carol's shared coins.
5. Carol computes her output $b_C \in \{0, 1\}$ as a function of $(m_A, m_B, m_p, r_p, r_{BC})$.

The complexity measures of a CM protocol are Alice's and Bob's message sizes as well as the length of the public message m_p. If we allow $|m_p| \geq \log |X|$ we can construct trivial protocols (Alice publishes x, Bob sends Carol $f(x, y)$). A CM protocol is *polynomial time* if Alice and Bob are PPTMs (regardless of Carol's running time, which strengthens negative results). The security requirement for a CM protocol is that for any adversary that selects an Alice's input $x \in X$, then receives (r_p, m_p), and *only then* selects Bob's input $y \in Y$, Carol's output b_C equals $f(x, y)$ with probability at least $1 - \varepsilon$, for some constant $\varepsilon < \frac{1}{2}$.

Theorem 3.1. *If there exists a CM protocol for equality testing where $X, Y = \{0, 1\}^n$ with success probability 0.83 and public messages of length at most $0.01 \cdot n$, where Alice's and Bob's messages are of lengths at most ℓ_A, ℓ_B, then $\ell_A \times \ell_B \geq c \cdot n$ ($c > 0$ is a constant specified in the proof).*

We also consider a computational security definition: a CM protocol is secure in the computational setting if it is secure versus any PPTM adversary. Theorem 6.1 of Section 6 states that breaking the information-theoretic lower bound in a computational setting implies the existence of one-way functions. We do not prove Theorem 3.1 separately, for the sake of brevity we derive it as a corollary of Theorem 6.1.

3.3 Adaptively Changing Distributions

Adaptively changing distributions (ACDs) were introduced in [NR06], the companion paper to this work. Here we give an overview on ACDs and the main results used in this work. We refer the reader to [NR06] for an in-depth discussion of adaptively changing distributions and these results.

Adaptively Changing Distributions. An ACD instance consists of a pair of probabilistic algorithms for generation ($\mathcal{G}$) and for sampling ($\mathcal{D}$). The generation algorithm $\mathcal{G}$ receives a public input[5] $x \in \{0,1\}^n$ and outputs an initial secret state $s_0 \in \{0,1\}^{s(n)}$ and a public state p_0.

After running the generation algorithm, we can consecutively activate a sampling algorithm $\mathcal{D}$ to generate samples form the adaptively changing distribution. In each activation, $\mathcal{D}$ receives as its input a pair of secret and public states, and outputs new secret and public states. The set of possible public states is denoted by S_p and the set of secret states is denoted by S_s.

Each new state is always a function of the current state and some randomness, which we assume is taken from a set R. The states generated by an ACD are determined by the function $\mathcal{D} : S_p \times S_s \times R \mapsto S_p \times S_s$. When $\mathcal{D}$ is activated for the first time, it is run on the initial public and secret states p_0 and s_0 respectively, generated by $\mathcal{G}$. The public output of the process is a sequence of public states $(x, p_0, p_1, p_2, \ldots)$.

Learning ACDs. An algorithm $\mathcal{L}$ for learning an ACD $(\mathcal{G}, \mathcal{D})$ sees x and p_0 (generated, together with s_0, by running $\mathcal{G}$), and is then allowed to observe $\mathcal{D}$ in consecutive activations, seeing the public output in its entirety (but seeing *none* of the secret states!). The learning algorithm's goal is to output a hypothesis h on the initial secret state that is functionally equivalent to s_0 for the next activation of $\mathcal{D}$ (the next "round"). The requirement is that with probability at least $1 - \delta$ (over the random coins of $\mathcal{G}$, $\mathcal{D}$, and the learning algorithm), the distribution of the next public state, given the past public states and that h was the initial secret output of $\mathcal{G}$, is ε-close to the same distribution with s_0 as the initial secret state (the "real" distribution of $\mathcal{D}$'s next public state).

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** $D_i^s(x, p_0, \ldots, p_i)$. This is a distribution on the public state that will be generated by $\mathcal{D}$'s next $(i+1$-th) activation.

Definition 3.2. *The Distribution D_i^s [NR06]* The distribution $D_i^s(x, p_0, \ldots, p_i)$ 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 $(x, p_0, \ldots, p_i)$, and given also that **the initial secret state that $\mathcal{G}$ computed was s.**

For any initial secret state $s \in S_{init}$, past public states $(x, p_0, \ldots, p_i)$, and (new) public state $p_{i+1} \in S_p$: $D_i^s(x, p_0, \ldots, p_i)[p_{i+1}]$ is the probability, conditioned on the observed public information $(x, p_0, \ldots, p_i)$ 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}. For convenience, we sometimes denote this distribution by D_i^s, when $(x, p_0, \ldots, p_i)$ are clear from the context.

Note that (efficiency consideration aside) the distribution $D_i^s(x, p_0, \ldots, p_i)$ can be computed by enumerating over all random strings (for $\mathcal{G}$ and $i+1$ activations of $\mathcal{D}$), for which the algorithm $\mathcal{G}$

[5] This is a variant of the basic definition of an ACD instance, see [NR06]

on input x 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 of the next public state computed by $\mathcal{D}$. We emphasize again that $\mathcal{L}$ sees only $(x, p_0, p_1 \ldots p_k)$, 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. In [NR06] much of the focus is on bounding this number of "rounds". An ACD learning process is defined below.

Definition 3.3. *Learning Process [NR06]* 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 learning algorithm is given an initial input $x \in \{0,1\}^n$. The generating algorithm $\mathcal{G}$ is run on input $x \in \{0,1\}^n$, generating initial secret and public states s_0 and p_0. The learning algorithm $\mathcal{L}$ receives the initial public state p_0.

2. 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 $D^{s_0}_{k+1}$, or $D^{s_0}_{k+1}(x, p_0, \ldots, p_i)$). 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.

We are now ready to define what it means to (ε, δ)-learn an ACD (for one input length).

Definition 3.4. *Learning Algorithm [NR06]* 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 (for any input $x \in \{0,1\}^n$) 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).

We conclude with an asymptotic definition of learning ACDs.

Definition 3.5. *Learning ACDs [NR06]* 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.

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 size $s(n)$ of the initial secret state, the higher the sample complexity for learning the ACD will be.

Known Results. In [NR06] it is shown that for any ACD there exists a (not computationally efficient) learning algorithm that activates $\mathcal{D}$ at most $O(\frac{s(n)}{\delta^2 \cdot \varepsilon^2})$ times. Furthermore, if $\mathcal{G}$ and $\mathcal{D}$ are PPTMs, x is selected uniformly at random, and almost one-way functions do not exist, then there

exists a PPTM learning algorithm for the ACD that activates $\mathcal{D}$ at most $O(\frac{s(n)}{\delta^2 \cdot \varepsilon^2})$ times. We use these results to prove lower bounds and to show that breaking the lower bounds in a computational setting implies the existence of almost one-way functions.

Theorem 3.2 (Learning ACDs information theoretically [NR06]). *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 $O\left(\frac{s(n)}{\delta^2(n) \cdot \varepsilon^2(n)}\right)$ steps.*

In the computational setting, we say that an ACDs is *efficiently constructible* if its generating algorithm $\mathcal{G}$ and its sampling algorithm $\mathcal{D}$ are both PPTMs. We now define hard-to-learn ACDs.

Definition 3.6 (Hard-to-Learn ACD [NR06]). 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:

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

The following theorem asserts the equivalence of hard-to-learn ACDs and almost one-way functions.

Theorem 3.3 (Learning ACDs and One-Way Functions [NR06]). *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 $O\left(\frac{s(n)}{\delta^2(n) \cdot \varepsilon^4(n)}\right)$ samples*[6].

4 Sublinear Authenticators

In this section we formally define authenticators and consider some of their basic properties. Authenticators are closely related to online memory checkers, and may be of independent interest, since they model a relaxed notion of memory checking: the notion of "repeatedly reliable" local testability (in the same way that memory checkers model repeatedly reliable repeated local decodability).

An authenticator is a triplet of probabilistic Turing machines for encoding ($\mathcal{E}$), decoding ($\mathcal{D}$) and verification ($\mathcal{V}$). The task of the authenticator is to encode an input x (using $\mathcal{E}$) into public and secret encodings. An adversary may alter the public encoding arbitrarily. The consistency verifier $\mathcal{V}$ checks whether x can be recovered from the (possibly altered) public encoding. The point is that $\mathcal{V}$ makes few queries to the public encoding. This is possible using the secret encoding, which is both *reliable* (cannot be altered by the adversary) and *secret* (the adversary cannot see it). The decoding algorithm $\mathcal{D}$ is used to determine whether the original input x can be recovered from the (altered) public encoding. Essentially, $\mathcal{V}$ verifies whether $\mathcal{D}$'s output on the public encoding is the same x that was given as input to $\mathcal{E}$.

[6] 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.

Definition 4.1 (Authenticator)**.** An authenticator is a triplet $(\mathcal{E}, \mathcal{D}, \mathcal{V})$ of probabilistic Turing machines, as follows:

- Encoder $\mathcal{E}$: receives an input $x \in \{0,1\}^n$ and random coins r. $\mathcal{E}$ outputs a "public" encoding of x, $p_x \in \{0,1\}^{u(n)}$, and a small "secret" encoding, $s_x \in \{0,1\}^{s(n)}$. We denote this as $p_x = \mathcal{E}_p(x,r), s_x = \mathcal{E}_s(x,r)$ or $(p_x, s_x) = \mathcal{E}(x,r)$.
- Decoder $\mathcal{D}$: is deterministic,[7] receives a public encoding $p \in \{0,1\}^{u(n)}$ and outputs a string $\mathcal{D}(p) \in \{0,1\}^n$.
- Consistency Verifier $\mathcal{V}$: receives a "public" encoding $p \in \{0,1\}^{u(n)}$ a small "secret" encoding $s \in \{0,1\}^{s(n)}$ and random coins r. We think of $\mathcal{V}$ as having "oracle access" to p, namely it can repeatedly ask for (and receive) single bits of p. We will later be concerned with the number of times $\mathcal{V}$ accesses p (the number of bits it reads) each time it is activated. $\mathcal{V}$ has arbitrary and unrestricted access to s (in particular it can read all of s). $\mathcal{V}$ then (online) either accepts or rejects the pair (s,p), and when it accepts it also outputs a new pair of "public" and "secret" encodings $p' \in \{0,1\}^{u(n)}, s' \in \{0,1\}^{s(n)}$.

An *operation sequence* on an authenticator consists of activating $\mathcal{E}$ on an input $x \in \{0,1\}^n$, and then repeatedly activating $\mathcal{V}$ for verification and for deriving new "public" and "secret" encodings. Let (p_x^0, s_x^0) be $\mathcal{E}$'s outputs on x. For any integer $i \geq 0$, let (p_x^{i+1}, s_x^{i+1}) be $\mathcal{V}$'s outputs on (p_x^i, s_x^i). For security, we require both completeness and soundness from the authenticator. Namely, for any adversary that chooses the initial input x:

- Completeness. For any adversary that chooses $x \in \{0,1\}^n$ and a number of rounds ℓ that is polynomial in n. For each $i \in [\ell]$, the verifier $\mathcal{V}$ accepts (p_x^i, s_x^i) with probability at least 0.9[8]. $\mathcal{D}$ *always*[9] outputs x on p_x^i.
- Soundness. For any adversary that chooses x, sees $(x, p_x^0 \ldots p_x^i)$, until at some point it chooses $p' \neq p_x^i$ such that $\mathcal{D}$ does not output x on p', $\mathcal{V}$ rejects (p', s_x^i) with probability at least 0.9 (over the adversary's, $\mathcal{V}$'s and $\mathcal{E}$'s coin tosses). The adversary "sees" x and p_0, and in each activation of $\mathcal{V}$ it "sees" which locations in the public memory $\mathcal{V}$ reads and writes (and what values are written). The adversary down *not*, however, get any information about the secret encodings generated by $\mathcal{E}$ and $\mathcal{V}$.

  Note the order of events: the adversary first sees $(x, p_x^0 \ldots p_x^i)$, as well as the locations read and written by $\mathcal{V}$, for some polynomial (in n) number i of activations. The adversary then chooses p', and *only then* $\mathcal{V}$ runs on (p', s_x^i). In particular, the adversary *cannot* change the public encoding after the verifier begins running on it (unlike the setting of online memory checkers).

The probability that the verifier accepts unaltered public encodings and rejects significantly altered public encodings is its *success probability* (0.9 in the definition). Unless we explicitly note

[7] W.l.o.g. $\mathcal{D}$ is deterministic, since $\mathcal{D}$ has no secret output and for now we are not concerned with its computational efficiency. A probabilistic $\mathcal{D}$ can be derandomized by trying all possible random strings. More generally, to increase computational efficiency, one may consider probabilistic $\mathcal{D}$'s.

[8] Alternatively, a stronger guarantee could be that the probability that $\mathcal{V}$ rejects even one of the pairs is at most 0.1. We strengthen the lower bounds by working with the more relaxed definition

[9] All results hold even if $\mathcal{D}$ only outputs x w.h.p. over $\mathcal{E}$'s and $\mathcal{V}$'s coins.

otherwise, the security (completeness and soundness) of authenticators in this work are *information-theoretic*. Namely, they hold against *any* (even unbounded) adversary. An authenticator is secure in the *computational setting* if it is secure versus any PPTM adversary. An authenticator is *polynomial time* if $\mathcal{V}$ and $\mathcal{E}$ are PPTMs.

Similarly to online memory checkers, the important complexity measures for an authenticator on inputs of length n are its *space complexity* $s(n)$, the size of the secret encoding, and its *query complexity* $q(n)$, the number of bits $\mathcal{V}$ reads from the public encoding. In this work we allow the authenticator to make arbitrary changes to the public encoding each time it is activates (the write complexity is unbounded). This strengthens lower bounds, though we note that in our upper bounds the authenticator doesn't modify the public encoding at all.

Discussion. Note that an adversary for an authenticator is in a sense "weaker" than an adversary for an online memory checkers: an authenticator is guaranteed that the public encoding does not change while it is being verified, whereas an adversary for an online memory checker may change the public encoding adaptively as the checker makes queries. We cannot hope to achieve such a strong security guarantee for authenticators, since if the adversary could determine changes to the public encoding adaptively as $\mathcal{V}$ runs it could wait until $\mathcal{V}$ queried all $q(n)$ locations and then change all the unread bits!

$\mathcal{V}$'s Access Pattern. A verifier $\mathcal{V}$ (w.l.o.g.) accesses the public encoding bit-by-bit: choosing a location (or address) in the public encoding to read, reading the bit value stored there, writing a new value and proceeding to the next location. Note that $\mathcal{V}$ may be adaptive, and thus it is important that the locations it requests from public memory are accessed consecutively. The *access pattern* of an activation of the verifier consists of the addresses in the public encoding that the verifier accessed, and the values it read and wrote.

Definition 4.2. Access Pattern. The verifier $\mathcal{V}$'s access pattern in an activation on some public and secret encodings, is the ordered collection of locations in the public memory that it *read*, and the values it read from those locations. Recall that the verifier can read the value of at most $q(n)$ locations. We *do not* consider the locations and values *written* to the public memory as part of the access pattern.

Note that, as discussed in Definition 4.1, an authenticator adversary gets to see $\mathcal{V}$'s access pattern (in this sense the access pattern is public information). We will also refer to the verifier's *access pattern distribution* on a particular public encoding p. This is the distribution of the verifier's access pattern in its next activation on public encoding p, given $\mathcal{E}$'s secret encoding output and the access patterns in all previous activations of $\mathcal{V}$. The randomness is over $\mathcal{V}$'s coin flips in its next activation and in all past activations. For a formal definition see Definition 5.1.

4.1 Authenticators and Memory Checkers

The following theorem shows that any online memory checker can be used to construct an authenticator with similar secret encoding size and query complexity. We will show lower bounds for authenticators, by this theorem they will also apply to online memory checkers.

Theorem 4.1. *If there exists an online memory checker with space and query complexities $s(n)$ and $q(n)$, then there exists an authenticator with space and query complexities $O(s(n))$ and $O(q(n))$.*

The same holds for computationally secure memory checkers and authenticators. If the memory checker is polynomial time, then so is the authenticator.

Proof. The authenticator $(\mathcal{E}, \mathcal{D}, \mathcal{V})$ we construct uses $\mathcal{C}$ and a good error correcting code to encode strings in $\{0,1\}^n$. Our construction uses a binary error-correcting code. The properties we want from the code are constant relative distance (any two codewords differ in at least a constant fraction of their coordinates), and constant relative rate (encoding a word only increases its length by a constant multiplicative factor). We also require that the code has efficient encoding and error correcting procedures. The error correcting procedure should successfully recover the original codeword as long as the fraction of indices that has been corrupted is at most half the relative distance. To be concrete, in the construction we will use the Justesen code [rJ72] J, which has all of the above properties (with relative distance $\frac{1}{10}$ and relative rate $\frac{1}{2}$).

- To encode an input $x \in \{0,1\}^n$, $\mathcal{E}$ encodes x using the the code J into $J(x) \in \{0,1\}^{2n}$ and then simulates storing the bits of $J(x)$ into a public memory using $\mathcal{C}$. Let $(\mathcal{C}_p(J(x)), \mathcal{C}_s(J(x)))$ be the resulting public and secret outputs. $\mathcal{E}$'s public encoding is $J(x) \circ \mathcal{C}_p(J(x))$ and the secret encoding is simply $\mathcal{C}_s(J(x))$.
- To decode a public encoding $z \circ p$ (where $z \in \{0,1\}^{2n}$), $\mathcal{D}$ decodes z using J's error correcting procedure, and outputs the resulting vector in $\{0,1\}^n$.
- $\mathcal{V}$ receives a public encoding $z \circ p$ (where $z \in \{0,1\}^{2n}$) and a secret encoding s. It then picks ℓ indices in $[2n]$ uniformly and at random (where $\ell = 60$), and simulates $\mathcal{C}$ exactly ℓ times (with initial secret memory s and initial public memory p) to retrieve all ℓ chosen indices. $\mathcal{V}$ accepts if for each index i that it chose, the memory's checker retrieved z_i when $\mathcal{V}$ requested that it retrieve the i-th index. Otherwise $\mathcal{V}$ rejects. Let p' and s' be the public and secret memories reached after ℓ consecutive activations of $\mathcal{C}$. $\mathcal{V}$ outputs a new public encoding $z \circ p'$ and a new secret encoding s'.

Claim 4.1. *The triplet $(\mathcal{E}, \mathcal{D}, \mathcal{V})$ is an authenticator.*

Proof. If no changes are made to the public encoding, then since we are using a code $\mathcal{D}$ will always correctly decode and with high probability (at least 0.95) $\mathcal{V}$ accepts (since $\mathcal{C}$ is an online memory checker). If, however, at some point the public memory $z \circ p$ is changed into $z' \circ p'$ and D can no longer correctly decode, then the relative distance between z and z' must be at least $\frac{1}{20}$. In each of $\mathcal{V}$'s iterations on $z' \circ p'$ there is a probability of at least $\frac{1}{20}$ of selecting an index that has changed between z and z'. Recall that $\ell = 60$, and thus $(1 - \frac{1}{20})^\ell < 0.05$. The probability of $\mathcal{V}$ "checking" an index in z' that has changed is at least 0.95. When checking such an index i, the probability that $\mathcal{C}$ gives the incorrect answer (z'_i) is at most 0.05. By the union bound the error is detected with probability at least 0.9. □

Complexity analysis. $\mathcal{V}$ uses the same amount of secret memory as $\mathcal{C}$ would on an input of size $2n$, w.l.o.g. $s(2n) = O(s(n))$ and thus $\mathcal{V}$ uses at most $O(s(n))$ bits of secret memory. $\mathcal{V}$'s query complexity is ℓ times $\mathcal{C}$'s query complexity on an input of size $2n$, plus ℓ additional queries. Again, w.l.o.g. $q(2n) = O(q(n))$ and thus $\mathcal{V}$'s query complexity is at most $O(q(n))$. A similar argument holds in the computational setting, and since the reduction is efficient, if the original memory checker was efficient, then so is the resulting authenticator. □

4.2 Constructions of Authenticators

An Unconditionally Secure Authenticator. Suppose we wanted an unconditionally secure authenticator. We could use the online memory checker construction of Section 3.1.1. This construction gives an upper-bound on the tradeoff between the authenticator's space complexity s and query complexity q, showing that $s \times q = O(n)$. To construct such an authenticator we use the unconditionally secure online memory checker of Section 3.1.1, putting it through the transformation of any online memory checker into an authenticator given in the proof of Theorem 4.1. Note that it is possible to directly construct a better authenticator with smaller hidden constants in its complexities.

A Computationally Secure Authenticator. We present a construction of a computationally secure authenticator. The authenticator will have constant ($O(1)$) query complexity, and space complexity that is the size of a secret key that allows cryptography (typically a small polynomial).

We begin by (informally) describing families of pseudo-random functions, introduced by Goldreich, Goldwasser and Micali [GGM86]. A collection of functions is a pseudo-random family if a random function from the collection is computationally indistinguishable from a completely random function. This property is remarkable because (unlike truly random functions) pseudo-random function have a succinct (polynomial) representation and are efficiently computable. We will use a family of pseudo-random functions from $D(n) = \{0,1\}^{O(\log n)}$ to $R(n) = \{0,1\}^{O(1)}$. The family of pseudo-random functions consists of a PPTM $\mathcal{F}$, that receives as its input a seed of length $\kappa(n) = n^{\varepsilon}$ (for some small $\varepsilon > 0$), $seed \in \{0,1\}^{\kappa(n)}$, and an input in the domain, $x \in D(n)$. The PPTM then outputs some item in the family's range $R(n)$. We denote this output by $f_{seed}(x) \triangleq \mathcal{F}(seed, x)$. It should be infeasible to distinguish with non-negligible advantage between a truly random function and a pseudo-random function f_{seed}, where the (short) *seed* is selected uniformly and at random. Such pseudo-random functions can be constructed from any one-way function (see [GGM86] and Håstad *et al.* [HILL99], or Goldreich [Gol01] for more details). We note that the reason we need to make cryptographic assumptions, and cannot use a truly random function, is that the size of the description of such a function (which will dominate the space complexity) is at least linear in n, which is too large (we might as well store the entire file may in the secret encoding!).

Given a collection of pseudo-random functions, we construct a computationally secure authenticator. The authenticator $(\mathcal{E}, \mathcal{D}, \mathcal{V})$ uses a good error correcting code to encode strings in $\{0,1\}^n$. We can again use the Justesen code (a binary code with relative rate $\frac{1}{2}$ and relative distance $\frac{1}{10}$).

- The encoding algorithm $\mathcal{E}$ gets an input vector $x \in \{0,1\}^n$ and encodes it using the error correcting code into $J(x) \in \{0,1\}^{2n}$ and stores it as the public encoding. The encoder also generates a random seed for a pseudo-random function $seed \in \{0,1\}^{\kappa(n)}$ and saves it as the secret encoding. There are $2n$ indices in the codeword, and for each such index $i \in [2n]$, $\mathcal{E}$ computes an index tag $t_i = f_{seed}(i \circ J(x)_i)$ and stores it in the public encoding.

- The decoding algorithm $\mathcal{D}$ receives a (potentially corrupted) codeword and pseudo-random tags. $\mathcal{D}$ ignores the tags and decodes the codeword using the decoding procedure for J (alternatively, $\mathcal{D}$ could check the consistency of the tags and treat corrupted tags as erasures, which would give a construction with better constants).

- The consistency verifier $\mathcal{V}$ receives the secret encoding *seed*, the public word $y \in \{0,1\}^{2n}$, and public pseudo-random tags. $\mathcal{V}$ selects $O(1)$ random indices in the codeword. For each index

i that it selected, $\mathcal{V}$ reads from the public encoding the bit value y_i and the pseudo-random tag t_i saved for i. For each such i, $\mathcal{V}$ checks whether indeed $t_i = f_{seed}(i \circ y_i)$. If not, $\mathcal{V}$ immediately rejects, but if the equality holds for all $O(1)$ of the randomly selected indices then $\mathcal{V}$ accepts.

Security. The authenticator is secure (both sound and complete). As long as the public encoding is not altered, the decoder will decode correctly and the verifier will accept with probability 1. The key point is that any PPTM adversary that tries to change the value in some bit of the codeword cannot hope to compute the appropriate new pseudo-random tag with success probability that is significantly higher than $\frac{1}{\log|R(n)|} \leq \frac{1}{2}$. This is because the new pseudo-random tag should be the value of a pseudo-random function at a point whose value the adversary has not seen. Intuitively, since it is impossible to predict the value of a random function at a point whose value has not been seen (the value is random), it is also impossible to predict the same value for a pseudo-random function. Thus, if an adversary modifies enough bit of the encoding to make decoding the original x impossible, then many tags are inconsistent and the verifier will find an inconsistent tag and reject with high probability.

Complexity Analysis. The space complexity is the size of the seed of a pseudo-random function. Under standard cryptographic assumptions we can take $\kappa(n) = O(n^{\varepsilon})$ for some small $\varepsilon > 0$ (κ could be polylogarithmic under stronger cryptographic assumptions). The verifier reads a constant number of bits from the (modified) codeword and their pseudo-random tags. Each tag is $\log|R(n)| = O(1)$ bits long, for a total query complexity of $O(1)$ bits per verification.

5 Lower Bound for Authenticators

5.1 A Simple(r) Construction

Before presenting a lower bound for *any* authenticator, we first present a lower bound for a restricted subclass: authenticators where the collection of locations and values in public memory that $\mathcal{V}$ reads and writes (the *access pattern*, see Definition 4.2), is independent of the secret encoding. We note that the best authenticators known (see Section 4.2) are of this type. The lower bound for these (simpler) authenticators is presented to demonstrate more clearly fundamentals of the general reduction, especially the connection between CM protocols for equality testing and authenticators.

Theorem 5.1. *For any authenticator, if the locations of the public encoding read by $\mathcal{V}$ are chosen independently of the secret encoding, and the authenticator has space complexity $s(n) > 0$ and query complexity $q(n) > 0$, then $s(n) \times q(n) = \Omega(n)$. Note that this does not restrict the number of locations in public memory that the authenticator* writes to*, only the number of locations it* reads from*.*

Proof. We use the authenticator to construct a CM protocol for equality testing. The CM protocol emulates an operation sequence of length 1: an input x is given to the encoding algorithm $\mathcal{E}$ and encoded into public and secret encodings. The adversary then replaces the public encoding by an encoding of some y, and the verifier is run to verify whether the adversary changed the public encoding (since $(\mathcal{E}, \mathcal{D}, \mathcal{V})$ is secure against *any* operation sequence, it must be secure against this one). In the CM protocol, Alice simulates the encoding algorithm $\mathcal{E}$ on her input x and sends

the resulting secret encoding to Carol. Bob simulates $\mathcal{E}$ on his input y using the same public randomness that Alice used. Bob then accesses the resulting public encoding (p_y) by simulating $\mathcal{V}$, and sends the sampled bits to Carol. Carol completes $\mathcal{V}$'s simulation as if $\mathcal{E}$ was run on x and then an adversary changed the public encoding to p_y. If $x = y$ then the public encodings used by Alice and Bob will be equal and $\mathcal{V}$ (and Carol) will accept (w.h.p.). Otherwise $\mathcal{V}$ (and Carol) will reject (w.h.p.).

The protocol $\mathcal{P}$:

Alice. Alice receives an input $x \in \{0,1\}^n$ and a shared public random string r_p. Alice runs $\mathcal{E}$ on x using r_p to obtain public and secret outputs (p_x, s_x^{dummy}). The secret encoding s_x^{dummy} is obtained using the public random string r_p, and will thus be seen by the adversary before he selects Bob's input y. For this reason, Alice will not send s_x^{dummy} to Carol. Instead, Alice will use her private random coins to *re-randomize* the secret encoding, selecting a second random string r' uniformly at random from the set $\mathcal{E}_{(x)}^{-1}(p_x) \triangleq \{r | p_x = \mathcal{E}_p(x,r)\}$ of random strings r such that $\mathcal{E}_p(x,r)$ is p_x. Alice computes a *secret* encoding $s_x = \mathcal{E}_s(x,r')$, and sends s_x to Carol ($s(n)$ bits).

The re-randomization technique presented here is used throughout this work. The CM adversary sees r_p and x, whereas an authenticator adversary only sees x and p_x (never any random string used by $\mathcal{E}$!). It may seem that the CM protocol "leaks" more of the secret encoding s_x than an authenticator, but this is not the case. The public random string does not give any more information on s_x than p_x would, because the *real* randomness that Alice used to generate s_x was selected *uniformly and at random* from the set $\mathcal{E}_{(x)}^{-1}(p_x)$. The only information that the CM adversary gains from seeing r_p is that the public encoding is p_x, exactly the information available to it in the authenticator scenario.

Bob. Bob receives an input $y \in \{0,1\}^n$ and runs $\mathcal{E}$ on y using the shared public random string r_p to obtain public and secret outputs (p_y, s_y^{dummy}). He then uses his private random coins (shared with Carol) to run $\mathcal{V}$ on $(p_y, 0^{s(n)})$ (the secret encoding that Bob uses is *always* $0^{s(n)}$). Bob sends Carol the $q(n)$ bit values read by $\mathcal{V}$. Note that Bob can sample by $0^{s(n)}$, rather than by the "real" secret encoding s_x, because we assumed the access pattern, the collection of locations and values read and written by $\mathcal{V}$ (see Definition 4.2), is completely independent of the secret encoding!

Carol. Carol would like to simulate $\mathcal{V}$ with secret encoding s_x on the public encoding p_y, using the bits of the public encoding read by Bob. Bob only sent Carol the values of the bits that he read, and not the locations from where he read them. Carol, however, can see Bob's random coins (recall the definition of CM protocols) and can thus reconstruct the locations that were accessed by using the secret encoding $0^{s(n)}$, the values read and Bob's random coins. Let $\vec{a}$ be Bob's access pattern, and $\mathcal{V}_{(p_y,s_x)}^{-1}(\vec{a})$ the set of random strings r for which $\mathcal{V}(p_y, s_x, r)$'s access pattern is $\vec{a}$. Carol selects uniformly and at random r'' from $\mathcal{V}_{(p_y,s_x)}^{-1}(\vec{a})$, and simulates $\mathcal{V}(p_y, s_x, r'')$. Carol accepts if and only if $\mathcal{V}$ accepts.

Claim 5.1. *$\mathcal{P}$ is a CM protocol for equality testing, with success probability 0.9.*

Proof. If there exists an adversary that makes the CM protocol fail with high enough probability, then this adversary can be used to make the authenticator fail with a similar probability. If $x = y$ then the protocol simulates running $\mathcal{V}$ on $\mathcal{E}$'s output, and thus any CM adversary that makes

Carol reject with high enough probability can be used to make the authenticator fail (breaking completeness). If $x \neq y$ then the protocol simulates the following authenticator operation sequence: the adversary selects x and gives it to $\mathcal{E}$, resulting in the output (p_x, s_x). Now the CM adversary (without seeing s_x) chooses y ($y \neq x$), and replaces p_x with p_y. This public encoding p_y is generated by running $\mathcal{E}$ on y with a uniformly random string from the set of random strings for which, on input x, the encoder $\mathcal{E}$ outputs p_x. It is important to note that the CM adversary learns nothing from the public random string and the public message, beyond the fact that the public encoding for x is p_x. The access pattern distribution of $\mathcal{V}$ on p_y is identical regardless of whether the secret encoding is s_x or s_y^{dummy}, and thus any CM adversary can be used against the authenticator, and the protocol simulates running $\mathcal{V}$ on (p_y, s_x). If Carol accepts w.h.p. then $\mathcal{V}$ also accepts w.h.p. and thus $(\mathcal{E}, \mathcal{D}, \mathcal{V})$ is not an authenticator (soundness is broken).

□

We conclude that $\mathcal{P}$ is indeed a CM protocol for equality testing with message lengths $s(n)$, $q(n)$ for Alice's and Bob's messages, and no public message. By Theorem 3.1: $q(n) \times s(n) = \Omega(n)$.
□

We have seen that authenticators and CM protocols are closely related. A key difficulty arises when we try to generalize this result: if $\mathcal{V}$ chooses the locations in public memory that it reads according to some distribution that depends on the secret encoding, then Bob cannot access p_y according to the same access pattern distribution as $\mathcal{V}$ (since Bob does not know s_x), and Carol cannot simulate $\mathcal{V}$ correctly.

The remainder of this section deals with this difficulty. We observe that the lower bound of Theorem 5.1 holds even for "one-time" authenticators: authenticators that are only guaranteed to work for operation sequences of length 1. The verifiers of one-time authenticators only need to verify that $\mathcal{E}$'s public encoding has not changed too much. This is a much weaker requirement than standard authenticators, which are required to work for operation sequences where $\mathcal{V}$ may be activated many times by the adversary. Even with this weakened requirement, as long as a one-time verifier's access pattern distribution is chosen independently of its secret memory, the (tight) lower bound of Theorem 5.1 holds.

5.2 Overview of The Generalized Reduction

We now turn to presenting a reduction from *any* authenticator with query complexity $q(n)$ (the query complexity bounds the number of locations read from public memory, the number of locations written to is not bounded), and space complexity $s(n)$, to a CM protocol for equality testing with message lengths $O(q(n))$ and $O(s(n))$. By Theorem 4.1 this also implies a reduction from online memory checkers to CM protocols for equality testing. After proving the resulting CM protocol is correct and complete, we deduce that the time-space tradeoff $q(n) \times s(n) = \Omega(n)$ holds both for authenticators and for memory checkers (these are the results stated in Theorem 5.3 and Corollary 5.4).

Using the secret encoding. As noted in the previous subsection, if the verifier determines its access pattern independently of the secret encoding, then the time-space tradeoff holds even for one-time authenticators (that are only required to work for operation sequences where the verifier is activated at most once). However, if we allow the verifier to use its secret encoding to determine

its access pattern, we can construct very efficient one-time authenticators. We sketch a simple one-time authenticator:

The encoder encodes the input using a high-distance code as the public encoding. The secret encoding is the value of the codeword at a few randomly selected locations (together with the selected locations). The decoder decodes the public encoding according to the closest codeword. If an adversary changes enough locations of the the public encoding to make the decoder err, then many locations must have changed (since the encoder used a high-distance code). The verifier can detect the switch with high probability by checking whether any of the saved locations were changed. We get a verifier making a constant number of queries to the public encoding and using a logarithmic secret encoding.

The example of one-time authenticators may seem artificial, but it illustrates the power of a verifier that uses the secret encoding to determine its *access pattern*. The key difficulty for the adversary in this case is that the verifier's access pattern in its next activation is unknown (and hard to predict). One should not, however, lose hope in showing a strong lower bound. Taking a closer look at the one-time verifier given in the example, for instance, we may observe that if the verifier is used again then the adversary already knows the locations whose values are saved in the secret encoding, and can thus easily fool the verifier by corrupting every location in the public encoding except the saved ones. The toy example illustrates a pivotal point: as long as the adversary doesn't know the verifier's access pattern *distribution*, the verifier is "safe". However, the adversary can learn the verifier's access pattern distribution by repeatedly activating it on the public encoding.

What's next? We begin by outlining the reduction that appears in the remainder of this section. We will first show that an adversary can learn the verifier's access pattern distribution on the (unchanged) public encoding. We then present a new technique: "bait and switch". The "bait and switch" technique is used to show that the adversary can change the public encoding without worrying that the verifier's access pattern distribution will also change (in this sense, the verifier does not gain anything if it determines its access pattern adaptively by the public encoding). Finally, we present the reduction from *any* authenticator to a CM protocol for equality testing.

Learning the Access Pattern Distribution. In Section 5.3 we consider the task of learning the verifier's access pattern distribution. We discuss the specifics of the learning task, and show that the adversary can (with high probability) learn an access pattern distribution that is statistically close to the verifier's access pattern distribution *for its next activation* on the (unchanged) public encoding. To prove this, we phrase the task of learning the access pattern distribution as a problem of learning adaptively changing distributions (see [NR06]).

Bait and Switch. After showing that the adversary can learn the verifier's access pattern distribution we would like to conclude that using the secret encoding to determine the access pattern distribution does not help the verifier. However, this is not immediate. The difficulty is that the adversary can only learn $\mathcal{V}$'s access pattern distribution on p, the *correct and unmodified* public encoding. At first glance it seems that this does not help the adversary at all, since it would like to "fool" $\mathcal{V}$ into accepting a very different p'! The adversary knows nothing about $\mathcal{V}$'s access pattern distribution on p' (since $\mathcal{V}$ may be adaptive). This access pattern distribution cannot be learned since once $\mathcal{V}$ is activated with a modified public encoding p' it may immediately reject, "shut down"

and stop working (the authenticator's behavior after it detects deception is completely unspecified). It seems like we are back where we started.[10] This is similar to the scenario of tracing traitors (see Fiat and Tassa [FT01] and Naor, Naor and Lotspiech [NNL01]) in which as soon as a traitor "detects" that it is being hunted down it may decide to shut down and "play innocent". In their scenario, the solution to this problem was simply to define this as an acceptable outcome of the traitor tracing process. We, however, cannot let the verifier "shut down" since the learning process would stop and the adversary would not be able to learn the verifier's access pattern distribution!

In Section 5.4 we overcome this obstacle, using a technique we refer to as "bait and switch". The solution is to construct a new verifier $\mathcal{V}_{mod}$ (based on the original verifier $\mathcal{V}$), that will run as $\mathcal{V}$, except that after the adversary learns $\mathcal{V}$'s access pattern distribution, $\mathcal{V}_{mod}$ will use on p' the access pattern distribution that the adversary learned. Thus the adversary will indeed know the verifier's access pattern distribution in its "critical invocation". The new verifier $\mathcal{V}_{mod}$ may seem much weaker than the original $\mathcal{V}$, but we will show that (if it is given some additional information) $\mathcal{V}_{mod}$'s failure probability is not much greater than the failure probability of $\mathcal{V}$. To reach this conclusion, we show that $\mathcal{V}_{mod}$ can check whether its access pattern distribution on the public encoding is far from the distribution induced by its secret encoding: in this case $\mathcal{V}_{mod}$ concludes that the adversary modified the public encoding and immediately rejects.

Constructing a CM Protocol. In Section 5.5 we use $\mathcal{V}_{mod}$ to construct a CM protocol $\mathcal{P}$ for equality testing, in the spirit of the simpler protocol constructed in Section 5.1. Alice receives x and runs $\mathcal{V}_{mod}$ until the critical invocation, Bob receives y and Carol simulates $\mathcal{V}_{mod}$ to determine whether $x = y$. All that Carol needs to simulate $\mathcal{V}_{mod}$ is shared public information and two short messages from Alice and Bob. Alice sends Carol an initial secret encoding of size $s(n)$. Bob sends Carol c access patterns on the public encoding ($c \cdot q(n)$ bits), where c is a constant. Claim 5.5 states that if $(\mathcal{E}, \mathcal{D}, \mathcal{V})$ was an authenticator, then $\mathcal{P}$ is a CM protocol for equality testing.

5.3 Learning The Access Pattern Distribution

In this subsection we describe the learning task of an adversary who wants to learn the verifier $\mathcal{V}$'s access pattern distribution. For a fuller discussion of the learning task, the model of learning adaptively changing distributions, and the learning algorithm, we refer the reader to Section 3.3 and to [NR06].

An authenticator functions as follows: an input x is given to the encoding algorithm $\mathcal{E}$, which outputs a public encoding p_x^0 and a secret encoding s_x. The adversary does not know which of the possible secret encodings was $\mathcal{E}$'s output. Each secret encoding s_x induces an access pattern distribution on $\mathcal{V}$'s possible access patterns (see Definitions 4.1,4.2). The only information hidden from the adversary at the beginning of the learning process is s_x. To learn the access pattern distribution, the adversary can repeatedly activate $\mathcal{V}$. Before $\mathcal{V}$ is activated on public encoding

[10] For a common-sense understanding of this obstacle, connoisseurs of the movie "Terminator 2: Judgment Day" may recall the T-1000 robot's attempt to impersonate John's foster mother. The robot samples her behavior in a domestic setting and imitates her perfectly, until it is asked a question with the name of John's dog deliberately altered. At this moment the "public setting" changes, the T-1000 fails to observe that the dog's name is false, and its behavior becomes very different from the foster mother's.
For a more concrete example, consider a verifier that, as long as it only reads 0's, only makes uniformly random queries, but once it reads a 1 starts using the secret memory to determine which locations are read. The access pattern on the all-0 public encoding is easy to learn, but this gives no information on the access pattern distribution once the public encoding is modified to contain many 1's!

p, for every possible initial secret encoding s^0, given the access patterns of all past activations of $\mathcal{V}$ (denoted by $\vec{a}$), and given also that s^0 was the initial secret encoding, the access pattern distribution of $\mathcal{V}$'s next activation on public encoding p is well defined (and computable by the unbounded adversary).

Definition 5.1 (Access Pattern Distribution)**.** Consider the operation sequence in which $\mathcal{E}$ is run on an input $x \in \{0,1\}^n$, and $\mathcal{V}$ is activated k times. Let s^0 be some possible initial secret encoding, and $\vec{a} = (a_0, \ldots a_{k-1})$ the vector of locations and values read in $\mathcal{V}$'s k previous activations (the access patterns of $\mathcal{V}$'s previous activations, these do not include the locations or values *written to* public memory).

We define $D^p_{\vec{a}}(s^0)$ to be the access pattern distribution of $\mathcal{V}$'s $(k+1)$-th activation on an arbitrary public encoding p, given that $\mathcal{V}$'s access pattern in its previous invocations was $\vec{a} = (a_0, \ldots a_{k-1})$ and the initial secret encoding given by $\mathcal{E}$ was s^0. The randomness is over all $k+1$ activations of $\mathcal{V}$.

Note that given s^0 and $\vec{a}$, the access pattern distribution is independent of x (the input) and of the initial public encoding. Given s^0, the access pattern distribution only depends on the locations and values that $\mathcal{V}$ already read, and these are included in $\vec{a}$. To compute the probability of any particular access pattern a_k by the distribution $D^p_{\vec{a}}(s^0)$, enumerate over all possible random strings. Let $\mathcal{V}^{-1}_{s^0}(\vec{a})$ be the set of random strings for $k+1$ activations of $\mathcal{V}$, on which $\mathcal{V}$'s access pattern (i.e. the locations and values read) in its first k activations with initial secret encoding s^0 would be $\vec{a}$.[11] The probability of the access pattern a_k by $D^p_{\vec{a}}(s^0)$ is the fraction of these random strings for which $\mathcal{V}$'s access pattern in the $(k+1)$-th activation on public encoding p would be a_k. Note that computing this probability requires knowing p, in particular the probability of any access pattern in which the verifier reads values inconsistent with those stored in p is 0. Actually, the values in p of locations read by a_k are sufficient information for computing a_k's probability.

The Access Pattern Distribution as an ACD. We would like to show that an adversary can "learn" the authenticator's access pattern distribution. Towards this end, we phrase the problem of learning the access pattern distribution in the framework of learning adaptively changing distributions of [NR06]. Given an authenticator we define an adaptively changing distribution (ACD) with generating algorithm and sampling algorithms:

- The generating algorithm $\mathcal{G}$ runs the encoder $\mathcal{E}$ on its input x.[12] The initial public state is the public encoding, the initial secret state is the secret encoding (of size $s(n)$).
- The ACD sampling algorithm $\mathcal{D}$ runs the verifier $\mathcal{V}$ on the public and secret states, modifying them. The secret state generated is the new secret encoding, the public state (or public outcome) is the public encoding concatenated with the verifier's access pattern in all its past activations.

The results of [NR06] give a learning algorithm $\mathcal{L}$ for learning $\mathcal{V}$'s access pattern distribution. Activating $\mathcal{L}$ on the public output of the encoding algorithm $\mathcal{E}$, results in k calls to $\mathcal{V}$.

[11] Note we implicitly assume here that $\vec{a}$ is a "legally generated" sequence of access patterns for some legally generated initial public memory. I.e., no adversary changes the public memory during the first k invocations of $\mathcal{V}$.

[12] Note that we use a variation of the ACD definition, where the generating algorithm receives an initial input $x \in \{0,1\}^n$, and not just the input length parameter 1^n. The information-theoretic results of [NR06] all still hold

Corollary 5.2. *Let $(\mathcal{E},\mathcal{D},\mathcal{V})$ be any authenticator. Let x be some value given to $\mathcal{E}$ with public encoding p_x^0 and secret encoding s_x. The learning algorithm $\mathcal{L}$ receives $\varepsilon(n),\delta(n)>0$, sees x,p_x^0, but **does not see** s_x. The learner $\mathcal{L}$ invokes $\mathcal{V}$ at most $k=O\left(\frac{s(n)}{\delta^2(n)\cdot\varepsilon^2(n)}\right)$ times, with resulting access pattern $\vec{a}$ and public encoding p_x^k (the learning algorithm only sees $\mathcal{V}$'s public outcomes). $\mathcal{L}$ then outputs a secret encoding $s_x^{\mathcal{L}}$, such that with probability at least $1-\delta$, the statistical distance between $D_{\vec{a}}^{p_x^k}(s_x)$ and $D_{\vec{a}}^{p_x^k}(s_x^{\mathcal{L}})$ is at most ε. The probability is over $\mathcal{E},\mathcal{V}$ and $\mathcal{L}$'s coin tosses.*

Corollary 5.2 follows immediately from Theorem 3.2 of [NR06]. Treating the authenticator as an ACD, s_x is the initial secret output of $\mathcal{G}$. After activating $\mathcal{D}$ (and implicitly activating $\mathcal{V}$) for k steps, the past access patterns $\vec{a}$ and the final public p_x^k completely define the public states. The learning algorithm $\mathcal{L}$ of [NR06] activates $\mathcal{V}$ for at most $O\left(\frac{s(n)}{\delta^2(n)\cdot\varepsilon^2(n)}\right)$ consecutive steps, and learns a secret encoding $s_k^{\mathcal{L}}$ such that with probability at least $1-\delta(n)$, the statistical distance between $D_{\vec{a}}^{p_x^k}(s_x)$ and $D_{\vec{a}}^{p_x^k}(s_x^{\mathcal{L}})$ is at most $\varepsilon(n)$.

5.4 Bait and Switch

In this subsection we complete the description of the "bait and switch" technique, used to show lower bounds for a verifier $\mathcal{V}$ that uses the secret encoding to determine its access pattern distribution (see Section 5.2 for a discussion of this problematic case). We construct from $\mathcal{V}$ a modified verifier $\mathcal{V}_{mod}$, which is identical to $\mathcal{V}$ except in its activation after the adversary has learned the access pattern distribution. In this final activation, the modified verifier $\mathcal{V}_{mod}$ will run differently than the original $\mathcal{V}$ (in particular, it will use an access pattern distribution known to the adversary), but it will be shown in Lemma 5.1 that $\mathcal{V}_{mod}$'s success probability is not much lower than that of the original $\mathcal{V}$.

We refer to an *operation sequence* of the following type: an adversary $\mathcal{A}$ selects an input $x\in\{0,1\}^n$, and activates $\mathcal{E}$ on x to generate some pair of public and secret encodings (p_x^0,s_x) (the adversary does not see s_x!). $\mathcal{A}$ then runs the learning algorithm $\mathcal{L}$ on (p_x^0,s_x) with $\varepsilon=\frac{1}{100},\delta=\frac{1}{100}$ to learn $\mathcal{V}_{mod}$'s access pattern distribution. By Corollary 5.2 this requires k activations of $\mathcal{V}_{mod}$ (which up to this point runs exactly like $\mathcal{V}$), where $k=O(s(n))$ (since ε,δ are constant). Note that if $(\mathcal{E},\mathcal{D},\mathcal{V})$ is an authenticator, it is secure against *any* adversary, and specifically $\mathcal{V}$ and $\mathcal{V}_{mod}$ are secure against $\mathcal{A}$. Let p_x^k be the final public encoding (the output of $\mathcal{V}_{mod}$'s last invocation), and recall Definition 5.1 of the access pattern distribution $D_{\vec{a}}^{p}(s_x)$. After $\mathcal{L}$ completes its run, $\mathcal{A}$ learns a secret encoding $s_x^{\mathcal{L}}$ such that with probability $1-\delta$ the statistical distance between $D_{\vec{a}}^{p_x^k}(s_x)$ and $D_{\vec{a}}^{p_x^k}(s_x^{\mathcal{L}})$ is at most ε, where the probability is over $\mathcal{E},\mathcal{V}$ and $\mathcal{L}$'s.

We think of $\mathcal{A}$ as "baiting" the verifier by giving it legal public encodings, until the verifier surrenders its access pattern distribution. $\mathcal{A}$ then changes the public encoding p_x^k to some p' and activates $\mathcal{V}_{mod}$ for the $(k+1)$-th time. We call this last activation the "critical" invocation of $\mathcal{V}_{mod}$. The access pattern distribution of the original $\mathcal{V}$ at this point would be $D_{\vec{a}}^{p'}(s_x)$. The access pattern distribution of the new $\mathcal{V}_{mod}$, however, will be $D_{\vec{a}}^{p'}(s_x^{\mathcal{L}})$, which may be quite different (details follow). From here on we will focus on operation sequences of this type.

Unfortunately for $\mathcal{A}$, all it knows before the critical invocation is that switching $\mathcal{E}$'s secret output from s_x to $s_x^{\mathcal{L}}$ would not have drastically changed $\mathcal{V}$'s access pattern distribution *on the unmodified* p_x^k. Recall that $\mathcal{A}$'s goal is to exchange p_x^k with some other public encoding p', and $\mathcal{V}$'s access pattern distribution *on* p' *may change greatly depending on whether* s_x *or* $s_x^{\mathcal{L}}$ *was* $\mathcal{E}$*'s secret*

output! We denote $D_1 \triangleq D_{\vec{a}}^{p'}(s_x)$ and $D_2 \triangleq D_{\vec{a}}^{p'}(s_x^{\mathcal{L}})$: D_1 and D_2 may be statistically far[13]. It may seem that $\mathcal{A}$ has gained very little by activating the learning algorithm. We will, however, construct $\mathcal{V}_{mod}$ to determine its access pattern distribution in the critical invocation by D_2. $\mathcal{V}_{mod}$ uses D_2 only for determining its access pattern distribution in the critical invocation, and then simulates $\mathcal{V}$ using the generated access pattern and s_x, the original secret output of $\mathcal{E}$. The changing of the secret encodings used to determine the access pattern distribution is the "switch" part of the malicious adversary's behavior. It will be shown in Lemma 5.1 that if $(\mathcal{E}, \mathcal{D}, \mathcal{V})$ is an authenticator, then the failure probability of $\mathcal{V}_{mod}$ in the critical invocation is roughly the same as $\mathcal{V}$'s.

In the critical invocation, we slightly change the way the verifier is run, and the information available to it. The adversary gives $\mathcal{V}_{mod}$ the "learned" secret encoding $s_x^{\mathcal{L}}$ and $c = O(1)$ access patterns $a_k^1 \dots a_k^c$ all sampled by the same distribution $D_2 = D_{\vec{a}}^{p'}(s_x^{\mathcal{L}})$, where p' is the new (potentially modified) public encoding. Note that the adversary always gives $\mathcal{V}_{mod}$ the correct values, and is only allowed to modify the final public encoding from p_x^k to p'. $\mathcal{V}_{mod}$ is only given the access patterns $a_k^1 \dots a_k^c$, and cannot access the public encoding itself (it is given c samples, rather than 1, to decrease its error probability). The adversary is given a great deal of control over $\mathcal{V}_{mod}$, and to ensure that it cannot "fool" $\mathcal{V}_{mod}$ into accepting an incorrect public encoding we also give $\mathcal{V}_{mod}$ the initial secret encoding s_x ($\mathcal{E}$'s secret output), and the access pattern $\vec{a} = (a_0, \dots a_{k-1})$ of the past k invocations of $\mathcal{V}_{mod}$. Note that $\mathcal{V}_{mod}$ gets no information about the public encodings (beyond $\vec{a}$ and $a_k^1 \dots a_k^c$), and in particular it *does not* get p_x^0, p_x^k or p'. The requirement from $\mathcal{V}_{mod}$ is that for any x, if its inputs are generated by the above operation sequence then:

- If the public encoding is not changed by the adversary ($p' = p_x^k$), then $\mathcal{V}_{mod}$ accepts with high probability.
- If $\mathcal{D}$'s output on p' is not x then $\mathcal{V}_{mod}$ rejects with high probability.

We will show that even such a modified verifier $\mathcal{V}_{mod}$ that runs in the above modified authentication setting, suffices for constructing a CM protocol for equality, and thus we derive a lower bound.

Using Learned Distribution on *Modified* Public Memory. The main remaining issue, as discussed above, is that if the adversary changes the public encoding to p', then the access pattern distributions induced by the "real" and "learned" secret encodings may be very different. To resolve this issue we would like for the new verifier $\mathcal{V}_{mod}$ to check whether D_1 and D_2 are statistically very close or far on the (potentially *modified*) public memory p'. Naturally, if the distributions are close than $\mathcal{V}_{mod}$'s security is not significantly effected by using the learned distribution. By simply running the original verifier $\mathcal{V}$, the modified $\mathcal{V}_{mod}$ will accept w.h.p. when the public memory is not modified, and reject w.h.p. if it has been tampered with beyond recovery by $\mathcal{D}$. Indeed, as long as the adversary does not tamper with the public encoding, the learning algorithm guarantees that the distributions will be close, so completeness is inherited from $\mathcal{V}$. If, however, the adversary modifies the public encoding, and the distributions D_1 and D_2 become far, then "all bets are off". Indeed, whereas $\mathcal{V}$ may reject the modified public encoding always, it is possible that $\mathcal{V}_{mod}$, using

[13]For example, consider a verifier that on public encoding 0^n queries $q(n)$ random locations, but whenever it reads a "1" starts using its secret encoding to determine the next location read. Learning the access pattern distribution on public input 0^n may be easy, but gives no information about the access pattern when the public encoding is 1^n! In other words, $D_{\vec{a}}^{0^n}(s_x)$ and $D_{\vec{a}}^{0^n}(s_x^{\mathcal{L}})$ are always *identical*, but $D_{\vec{a}}^{1^n}(s_x)$ and $D_{\vec{a}}^{1^n}(s_x^{\mathcal{L}})$ may be very far or even disjoint.

very different access patterns, will accept w.h.p. This problematic case is immediately resolved, however, if $\mathcal{V}_{mod}$ can check whether D_1 and D_2 are close or far. When they are far, $\mathcal{V}_{mod}$ will immediately reject! This guarantees soundness, and completeness is maintained by the learning algorithm's guarantee that the distributions will be close (w.h.p.) as long as the public memory is unmodified.

Checking whether D_1 and D_2 are close, however, is non-trivial. In particular, $\mathcal{V}_{mod}$ does not receive the statistical distance between D_1 and D_2, and in fact these two distributions are not completely specified to it. The difficulty is that $\mathcal{V}_{mod}$ cannot compute the probabilities of access patterns that include unknown bits of p' (since $\mathcal{V}$ may be adaptive). The good news, however, is that $\mathcal{V}_{mod}$ *can* compute the probabilities by D_1 and D_2 of any access pattern that only accesses bits of p' that are available (note that D_1 and D_2 are both over the same public encoding p')! In particular, $\mathcal{V}_{mod}$ can compute the probabilities of $a_k^1 \dots a_k^c$ by D_1 and D_2 (see the discussion following Definition 5.1). The access patterns' probabilities by D_1 and D_2 (extremely "local" information) are sufficient for estimating the statistical distance between D_1 and D_2. If the estimate is that the distributions are far, then $\mathcal{V}_{mod}$ will immediately reject. If the input passes the "local test" then $D_{\vec{a}}^{p'}(s_x^{\mathcal{L}})$ and $D_{\vec{a}}^{p'}(s_x)$ are statistically close with high probability, and $\mathcal{V}_{mod}$ can safely simulate $\mathcal{V}$ on p' using an access pattern generated by $D_{\vec{a}}^{p'}(s_x^{\mathcal{L}})$, in particular $\mathcal{V}_{mod}$ could use the access pattern a_k^1. $\mathcal{V}_{mod}$'s correctness in this case follows from $\mathcal{V}$ being a verifier for an authenticator.

Algorithm $\mathcal{V}_{mod}(s_x, s_x^{\mathcal{L}}, \vec{a}, a_k^1 \dots a_k^c)$ in its critical invocation:

1. Let $q_i = D_1[a_k^i]$, $r_i = D_2[a_k^i]$. Take: $d_i = \begin{cases} 0 & \text{when } r_i \leq q_i \\ \frac{r_i - q_i}{r_i} & \text{when } r_i > q_i \end{cases}$.
   If $\frac{\sum_{i=1}^{c} d_i}{c} > \frac{3}{2}\varepsilon$ then reject.
2. Denote by $\mathcal{V}_{s_x}^{-1}(\vec{a}, a_k^1)$ the set of random strings for which $\mathcal{V}$, with initial secret encoding s_x gives access pattern $(\vec{a}, a_k^1)$ in its first $k+1$ invocations. Select uniformly and at random $r \in \mathcal{V}_{s_x}^{-1}(\vec{a}, a_k^1)$. Simulate $\mathcal{V}$'s first k invocations and its critical invocation using r. Accept if and only if $\mathcal{V}$ accepts.

Throughout the remainder of this section we refer to $\mathcal{V}$'s success probability with D_1. This is the probability, over $\mathcal{E}$ and $\mathcal{V}$'s coin tosses, that $\mathcal{V}$ succeeds given that the initial secret encoding was s_x and $\mathcal{V}$'s access pattern in past invocations was $\vec{a}$. In this case $\mathcal{V}$'s access pattern distribution in the critical invocation is D_1.

Claim 5.2 ($\mathcal{V}_{mod}$ Rejects Far Distributions)**.** *There exists a constant $c > 0$ such that for the ε used by $\mathcal{V}_{mod}$ in Step 1:*

- *If $\Delta(D_1, D_2) > 2\varepsilon$ then with probability at least $\frac{99}{100}$ the verifier $\mathcal{V}_{mod}$ rejects its input.*
- *If $\Delta(D_1, D_2) \leq \varepsilon$ then $\mathcal{V}_{mod}$ accepts its input with probability at least $\frac{99}{100}$*

Proof. First note that for any $i \in [c]$, the value of d_i is bounded: $0 \leq d_i \leq 1$. We will show that $\forall i \in [c], E_{a_k^i \sim D_2}[d_i] = \Delta(D_1, D_2)$, where the expectation is over a_k^i sampled by D_2.

$$E_{a_k^i \sim D_2}[d_i] = \sum_{a_k^i : r_i > q_i} \left[D_2[a_k^i] \cdot \frac{D_2[a_k^i] - D_1[a_k^i]}{D_2[a_k^i]} \right] = \sum_{a_k^i : r_i > q_i} \left[D_2[a_k^i] - D_1[a_k^i] \right] = \Delta(D_1, D_2)$$

By Hoeffding's inequality, if we take a large enough (constant) c, then:

- If $\Delta(D_1, D_2) > 2\varepsilon$ then $E[d_i] > 2\varepsilon$, and thus $\Pr\left[\frac{\sum_{i=1}^{c} d_i}{c} \leq \frac{3}{2}\varepsilon\right] \leq \frac{1}{100}$.
- If $\Delta(D_1, D_2) \leq \varepsilon$ then $E[d_i] \leq \varepsilon$, and thus $\Pr\left[\frac{\sum_{i=1}^{c} d_i}{c} > \frac{3}{2}\varepsilon\right] \leq \frac{1}{100}$.

□

Claim 5.3. *For any D_1, D_2, if $\mathcal{V}$'s success probability using D_1 is at least α and $\Delta(D_1, D_2) \leq \beta$, then $\mathcal{V}_{mod}$'s simulation of $\mathcal{V}$ (in Step 2) is successful with probability at least $\alpha - \beta$*

Proof. We begin by observing that if $D_2 = D_1$ then $\mathcal{V}_{mod}$'s success probability is exactly α, since activating $\mathcal{V}_{mod}$ in this case is activating $\mathcal{V}$ against the adversary $\mathcal{A}$ (this is the scenario described in the simpler example of Section 5.1). Let $\mathcal{V}_{s_x}^{-1}(\vec{a})$ be the set of (secret) random strings for which $\mathcal{V}$, with initial secret encoding s_x, gives access pattern $\vec{a}$ in its first k invocations and any access pattern in the critical invocation (note that given $\vec{a}$ this set does not depend on the public encoding). Let $\mathcal{V}_{s_x}^{-1}(\vec{a}, a_k)$ be the subset of random strings in $\mathcal{V}_{s_x}^{-1}(\vec{a})$ that give access pattern a_k in $\mathcal{V}$'s critical invocation. We now examine the distribution of the random string used by $\mathcal{V}$: $\mathcal{V}$'s success probability on the public encoding p' when its random string is selected uniformly and at random from $\mathcal{V}_{s_x}^{-1}(\vec{a})$ (the "normal" scenario) is α. Selecting the random string uniformly and at random from $\mathcal{V}_{s_x}^{-1}(\vec{a})$ is equivalent to sampling a_k^1 by D_1 and then selecting uniformly and at random a string from $\mathcal{V}_{s_x}^{-1}(\vec{a}, a_k^1)$. Let R_1 be this (uniform) distribution over the random strings in $\mathcal{V}_{s_x}^{-1}(\vec{a})$.

$\mathcal{V}_{mod}$ activates $\mathcal{V}$ with a random string selected by sampling a_k^1 by D_2 and then selecting uniformly and at random a string from $\mathcal{V}_{s_x}^{-1}(\vec{a}, a_k^1)$. Let this distribution be R_2 (R_2 is also over the random strings in $\mathcal{V}_{s_x}^{-1}(\vec{a})$, but it may not be the uniform distribution). The distributions D_1 and D_2 sample a_k^1, and the statistical distance between them is at most β. It will be shown that this implies $\Delta(R_1, R_2) \leq \beta$, and thus $\mathcal{V}$'s success probability when its random coins are selected by R_2 (as $\mathcal{V}_{mod}$ runs it) is smaller by at most β than its success probability when its random coins are selected by R_1.

We have gone from a pair of distributions on $\mathcal{V}_{mod}$'s access pattern, to a pair of distributions on $\mathcal{V}_{mod}$'s random string. We will now show that the statistical distance between the two access pattern distributions is equal to the distance between the two random string distributions ($\Delta(D_1, D_2) = \Delta(R_1, R_2)$). For $r \in \mathcal{V}_{s_x}^{-1}(\vec{a})$ denote by $a_k(s_x, r)$ the access pattern obtained in the critical invocation by running $\mathcal{V}$ for $k+1$ invocations using random string r and initial secret encoding s_x:

$$\begin{aligned}
\Delta(R_1, R_2) &= \frac{1}{2} \sum_{r \in \mathcal{V}_{s_x}^{-1}(\vec{a})} |R_1[r] - R_2[r]| \\
&= \frac{1}{2} \sum_{r \in \mathcal{V}_{s_x}^{-1}(\vec{a})} \left| \frac{D_1[a_k(s_x, r)]}{|\mathcal{V}_{s_x}^{-1}(\vec{a}, a_k(s_x, r))|} - \frac{D_2[a_k(s_x, r)]}{|\mathcal{V}_{s_x}^{-1}(\vec{a}, a_k(s_x, r)|} \right| \\
&= \frac{1}{2} \sum_{r \in \mathcal{V}_{s_x}^{-1}(\vec{a})} \frac{1}{|\mathcal{V}_{s_x}^{-1}(\vec{a}, a_k(s_x, r))|} \cdot |D_1[a_k(s_x, r)] - D_2[a_k(s_x, r)]| \\
&= \frac{1}{2} \sum_{a_k} |D_1[a_k] - D_2[a_k]| \\
&= \Delta(D_1, D_2)
\end{aligned}$$

□

Claim 5.4. *If $\mathcal{V}$'s success probability is at least γ, then $\mathcal{V}_{mod}$'s success probability (in the critical invocation) is at least $\gamma - \frac{3}{100}$*

Proof. The proof is a consequence of Claims 5.2 and 5.3. The probability that the learning algorithm fails is at most δ. The authenticator's success probability for an input x on the operation sequence defined for $\mathcal{V}_{mod}$ is at least γ. We take C to be the set of possible access pattern distributions in $\mathcal{V}$'s critical invocation (each access pattern distribution is determined by the initial secret encoding s_x, by $\vec{a}$ and by p'). We take Q to be the distribution (over C) of the critical invocation's access pattern distribution. Q is determined by x and p', where the randomness is over $\mathcal{E}$ and $\mathcal{V}$'s coin flips (that determine s_x and $\vec{a}$). For $D_1 \in C$ let α_{D_1} be $\mathcal{V}$'s success probability with D_1 as its access pattern distribution in the critical invocation. We get that:

$$\sum_{D_1 \in C} Q[D_1] \cdot \alpha_{D_1} = \Pr[\mathcal{V} \text{ succeeds}] \geq \gamma$$

If $p' = p_x^k$ then for any $D_1 \in C$, with probability $1 - \delta$, the distributions D_1 and D_2 (the distribution learned by the algorithm) are ε-close. The probability that $\mathcal{V}_{mod}$ rejects in its first step is at most $\frac{1}{100}$ (By Claim 5.2). The probability that $\mathcal{V}_{mod}$ rejects in its second step using D_2 is at most the error probability of $\mathcal{V}$ using D_1 plus ε (by Claim 5.3). Recall we took $\varepsilon, \delta = \frac{1}{100}$. We use the union bound to bound $\mathcal{V}_{mod}$'s probability of error in this case by $(1 - \gamma) + \delta + \varepsilon + \frac{1}{100}$.

If $\mathcal{D}$ does not return x on the public encoding p' then for any $D_1 \in C$, if the statistical distance between D_1 and D_2 is more than 2ε, then the probability that $\mathcal{V}_{mod}$ doesn't reject its input in its first step is at most $\frac{1}{100}$. If the statistical distance between D_1 and D_2 is at most 2ε then the probability that $\mathcal{V}_{mod}$ accepts in its first step using D_2 is at most $\mathcal{V}$'s error probability using D_1 plus 2ε. $\mathcal{V}_{mod}$'s total error probability is thus at most $(1 - \gamma) + 2\varepsilon + \frac{1}{100}$. □

Lemma 5.1. *If $(\mathcal{E}, \mathcal{D}, \mathcal{V})$ is an authenticator, then $(\mathcal{E}, \mathcal{D}, \mathcal{V}_{mod})$ is both correct and complete for the operation sequence outlined above, with success probability at least* 0.85.

Proof. The Lemma is a direct consequence of Claim 5.4. The success probability of $(\mathcal{E}, \mathcal{D}, \mathcal{V}_{mod})$ before the critical invocation is identical to that of $(\mathcal{E}, \mathcal{D}, \mathcal{V})$. By Claim 5.4, the success probability of $(\mathcal{E}, \mathcal{D}, \mathcal{V}_{mod})$ in the critical invocation is at least $0.9 - 0.03 > 0.85$ □

5.5 Authenticators and Equality Testing

Given an authenticator $(\mathcal{E}, \mathcal{D}, \mathcal{V})$ we would like to construct a CM protocol $\mathcal{P}$ for equality testing, where Alice and Bob use the authenticator's algorithms to send Carol information that will let her simulate a run of the (modified) verification algorithm $\mathcal{V}_{mod}$. The protocol description follows:

Alice. Alice receives an input $x \in \{0,1\}^n$ and a "public" random string r_p that becomes public to the adversary as soon as the adversary specifies x. Alice will proceed as follows:

1. **Encoding:** Encode x using $\mathcal{E}$ into a public output $p_x^0 \in \{0,1\}^{u(n)}$ and a secret output $s_x^{dummy} \in \{0,1\}^{s(n)}$. $\mathcal{E}$'s randomness is taken from the public random string r_p.

2. **Public Message:** Simulate the learning algorithm $\mathcal{L}$ as if an adversary was trying to learn s_x^{dummy} with error probability $\delta = \frac{1}{100}$ and statistical distance $\varepsilon = \frac{1}{100}$. This requires activating $\mathcal{V}_{mod}$ for k steps ($k = O(s(n)$ by Corollary 5.2), note that in these activations $\mathcal{V}_{mod}$ simply

runs $\mathcal{V}$). Both the activation of $\mathcal{L}$ and the implicit activations of $\mathcal{V}_{mod}$ are done using coins from the public random input r_p (different coins are used for each invocation of $\mathcal{V}_{mod}$ and for $\mathcal{L}$'s randomness, but all coins are taken from r_p). The learning algorithm outputs some hypothesis $s_x^{\mathcal{L}}$ as to $\mathcal{E}$'s secret output (this conjecture is ignored by Alice, but will later be used by Bob and Carol). $\mathcal{L}$'s run also determines a final public encoding p_x^k. During $\mathcal{L}$'s run $\mathcal{V}_{mod}$ is activated $O(s(n))$ times, accessing $q(n)$ locations in the public encoding in each invocation, let $\vec{a}$ be $\mathcal{V}_{mod}$'s access pattern.

After simulating the learning algorithm, publish a public message for Bob and Carol. The public message includes s_x^{dummy}, $s_x^{\mathcal{L}}$ and the vector v_a of $O(s(n) \times q(n))$ bits read during $\mathcal{L}$'s execution. Note v_a includes only the bits values read, and not their locations in the public encoding. Alice does not publish the entire access pattern, even though Bob and Carol will need to use it (and not just the bit values she read). This can be done since Bob and Carol can reconstruct the access pattern themselves from the bit values read (v_a), using the public random string r_p and the "dummy" secret encoding s_x^{dummy}.

3. **Private Message:** Select a string r' uniformly and at random from the set of random strings for which the encoding algorithm $\mathcal{E}$ outputs the public encoding p_x^0 on input x and $\mathcal{V}_{mod}$'s access pattern is $\vec{a}$. This choice is made using Alice's *private* random coins. Let $s_x \in \{0,1\}^{s(n)}$ be $\mathcal{E}$'s secret encoding on input x with random string r'. Send s_x to Carol.

The final step Alice takes requires some further explanation. The public random string r_p and the public message Alice posts for Bob and Carol must both be information that is available to the adversary in the authenticator scenario. The adversary in the authenticator scenario does not know $\mathcal{E}$ and $\mathcal{V}_{mod}$'s private coin tosses, and thus Alice cannot make this information public to the adversary without compromising the security of the reduction. The only information Alice gives the adversary is information which is known to it in the authenticator scenario. Namely, that one of the random strings which are consistent with the public encoding and access pattern was used, and the random string that was used is uniformly distributed among all such consistent random strings.

Bob. Bob receives an input $y \in \{0,1\}^n$, a "public" random string r_p, a "dummy" initial secret encoding s_x^{dummy}, a learned secret encoding $s_x^{\mathcal{L}}$ and the vector v_a of $O(s(n) \times q(n))$ bits read by Alice's invocations of $\mathcal{V}_{mod}$. Bob's ultimate goal is to generate a public encoding and access it as $\mathcal{V}$ would in its next run (the critical invocation).

1. Simulate $\mathcal{E}$ on y using r_p (as Alice did) to obtain a public encoding p_y^0. If s_x^{dummy} (as sent by Alice) is not equal to the resulting secret encoding of y send a message to Carol telling her that $x \neq y$. Note that if $x = y$ then $p_x^0 = p_y^0$ (since Alice and Bob both used the same random coins from r_p).

2. Simulate $\mathcal{L}$'s invocations of $\mathcal{V}_{mod}$ using the initial secret encoding s_x^{dummy}, the random string r_p (with query results as published by Alice in v_a). If the query results published by Alice are inconsistent with the public encoding p_y^0, send a message to Carol telling her that $x \neq y$.

   Bob does not simulate $\mathcal{L}$ to learn the initial secret encoding, but simply to reconstruct $\vec{a}$ from v_a and learn which locations of the public encodings were accessed by $\mathcal{V}_{mod}$ when it

was invoked by Alice (recall Bob does not know p_x^0). The access pattern $\vec{a}$ determines the changes made to p_y^0, these changes result in a new public encoding p_y^k. Note that if $x = y$ then $p_x^k = p_y^k$.

3. In the terms of Section 5.4, Bob sends Carol the bit values of c access patterns distributed by $D_{\vec{a}}^{p_y^k}(s_x^{\mathcal{L}})$. Let $\mathcal{V}_{s_x^{\mathcal{L}}}^{-1}(\vec{a})$ be the set of all random strings for which $\mathcal{V}$'s invocations using $s_x^{\mathcal{L}}$ (the learned secret encoding) with public encoding p_y^0 result in access pattern $\vec{a}$. Repeat the following step c times, in the i-th iteration:

   Choose uniformly and at random some random string $r_b^i \in \mathcal{V}_{s_x^{\mathcal{L}}}^{-1}(\vec{a})$. The uniform random choice is done using Bob and Carol's shared *private* coins. Simulate the k invocations of $\mathcal{V}_{mod}$ using $s_x^{\mathcal{L}}$, r_b^i and $\vec{a}$. This invocation results in a secret encoding s_y^i and the public encoding p_y^k. Invoke $\mathcal{V}$ one final time using Bob and Carol's shared private coins, s_y^i and p_y^k. This is the "critical" invocation, whose values depend on y. Send the vector v_i^b of bit values accessed to Carol (the vector is at most $q(n)$ bits long).

Carol. Carol receives a "public" random string r_p, an initial secret encoding s_x^{dummy}, a learned secret encoding $s_x^{\mathcal{L}}$ and the vector v_a of $O(s(n) \times q(n))$ bits read by Alice's invocations of $\mathcal{V}_{mod}$. Carol also receives from Alice her "true" initial secret encoding s_x and from Bob c vectors $v_1^b \dots v_c^b$ of up to $q(n)$ bits accessed by the final invocation of $\mathcal{V}_{mod}$ (or a message that $x \neq y$). Carol's goal is to simulate the critical invocation of $\mathcal{V}_{mod}$ and determine whether $x = y$.

1. If Bob sent a message saying $x \neq y$, reply that x and y are different.
2. Simulate $\mathcal{L}$'s invocations of $\mathcal{V}_{mod}$ using the initial secret encoding s_x^{dummy}, the random string r_p and with query results as published by Alice in v_a. This is identical to Bob's simulation and, similarly to Bob, when this step is complete Carol knows the access pattern $\vec{a}$.
3. Simulate Bob's "critical" runs of $\mathcal{V}$ (using the shared private random coins). The purpose of this simulation is to reconstruct from $v_1^b \dots v_c^b$ the access patterns $a_k^1 \dots a_k^c$ of Bob's critical invocations of $\mathcal{V}$.
4. Simulate algorithm $\mathcal{V}_{mod}$ with initial secret encoding s_x, access pattern $\vec{a}$ in $\mathcal{V}_{mod}$'s first k invocations, learned secret encoding $s_x^{\mathcal{L}}$ and access patterns $a_k^1 \dots a_k^c$ for $\mathcal{V}$'s critical runs. If $\mathcal{V}_{mod}$ rejects, reply that x and y are different. Otherwise, reply that x and y are equal.

Claim 5.5. *If $(\mathcal{E}, \mathcal{D}, \mathcal{V})$ is an authenticator with space complexity $s(n)$, and query complexity $q(n)$ then $\mathcal{P}$ is a consecutive messages protocol for equality with Alice message length $s(n)$, Bob message size at most $O(q(n))$, public message length $O(s(n) \cdot q(n))$, and error probability at most 0.15.*

Proof. The messages lengths are as specified above. The proof of security is based on Lemma 5.1 (the security of algorithm $\mathcal{V}_{mod}$). Assume for a contradiction that there exists an adversary $\mathcal{A}$ whose success probability when it operates against the protocol $\mathcal{P}$ is at least 0.15. We will use $\mathcal{A}$ to construct an adversary $\mathcal{A}'$ operating against $\mathcal{V}_{mod}$. $\mathcal{A}'$ will simulate $\mathcal{A}$ to generate x and then simulate Alice's activations of the authenticator (including the activation of $\mathcal{E}$ and the activations of $\mathcal{V}_{mod}$ by the learning algorithm $\mathcal{L}$) to generate $(p_x^0, \vec{a}, s_x^{\mathcal{L}})$ distributed exactly as if $\mathcal{A}$ gave x to

Alice. Let s_x be $\mathcal{E}$'s secret output on input x. The adversary now selects the "public" random bits of r_p used by Alice's activations of $\mathcal{E}$ and $\mathcal{V}_{mod}$. This is done by selecting a random string uniformly and at random from the set of random strings for which $\mathcal{E}$ and $\mathcal{V}_{mod}$'s public outcomes on input x are $(p_x^0, \vec{a})$. After selecting this public random string, the adversary can combine it with the random bits used by the learning algorithm $\mathcal{L}$ to generate the public random string r_p and the dummy secret encoding s_x^{dummy} ($\mathcal{E}$'s secret output on x using the randomness from r_p). The joint distribution of $(r_p, p_x^0, \vec{a}, s_x^{dummy}, s_x^{\mathcal{L}})$ as generated by $\mathcal{A}'$ with respect to s_x is identical to their joint distribution with respect to s_x when they are generated by $\mathcal{A}$ and Alice in $\mathcal{P}$. $\mathcal{A}'$ will continue $\mathcal{A}$'s simulation as if these were the contents public messages posted by Alice. $\mathcal{A}$ selects some y, $\mathcal{A}'$ will simulate Bob using $(y, r_p, \vec{a}, s_x^{\mathcal{L}}, s_x^{dummy})$ to generate the public encoding p_y^k of y (exactly as Bob would in $\mathcal{P}$).

Now $\mathcal{A}'$ will activate $\mathcal{V}_{mod}$ with $\vec{a}, s_x, s_x^{\mathcal{L}}$ and c samples from $D_{\vec{a}}^{p_y^k}(s_x^{\mathcal{L}})$. This activation of $\mathcal{V}_{mod}$ simulates Carol's run. $\mathcal{V}_{mod}$'s inputs are distributed exactly as required (unless Bob sends to Carol a message $x \neq y$, in which case $\mathcal{P}$ always outputs the correct answer and the adversary $\mathcal{A}$ fails). We conclude that $\mathcal{A}'$'s success probability when operating against $\mathcal{V}_{mod}$ is at least $\mathcal{A}$'s success probability when operating against $\mathcal{P}$. This is a contradiction, since Lemma 5.1 guarantees that $\mathcal{A}'$'s success probability cannot be higher than 0.15. □

Theorem 5.3. *If $(\mathcal{E}, \mathcal{D}, \mathcal{V})$ is an authenticator with space complexity $s(n) > 0$ and query complexity $q(n) > 0$ then $s(n) \times q(n) = \Omega(n)$.*

Proof. We construct from $(\mathcal{E}, \mathcal{D}, \mathcal{V})$ the CM protocol $\mathcal{P}$, with messages of length $O(s(n))$, $O(q(n))$ from Alice and Bob (respectively) to Carol and success probability 0.85 (by Claim 5.5). Let $c_{\mathcal{L}}$ be a constant such that the learning algorithm $\mathcal{L}$ invokes $\mathcal{V}$ at most $c_{\mathcal{L}} \cdot s(n)$ times. If $s(n) \times q(n) < \frac{n}{100 c_{\mathcal{L}}}$ then the public message posted by Alice is short enough (of length less than $\frac{n}{100}$), and by the CM lower bound (Theorem 3.1) this implies that $s(n) \times q(n) = \Omega(n)$. If $s(n) \times q(n) \geq \frac{n}{100 c_{\mathcal{L}}}$ then certainly $s(n) \times (q(n) + 1) = \Omega(n)$ □

Corollary 5.4. *If $\mathcal{C}$ is an online memory checker with space complexity $s(n) > 0$, and query complexity $q(n) > 0$, then $s(n) \times q(n) = \Omega(n)$.*

6 Efficient Authenticators Imply (Almost) One-Way Functions

In this section we prove that breaking the information-theoretic authenticator and online memory checker lower bounds of the previous section in a computational setting implies the existence of almost one-way functions. The proof proceeds in two stages:

1. If there exists a polynomial-time CM protocol for equality testing that breaks the information theoretic lower bound of Theorem 3.1 in a computational setting, then there exist one-way functions. This strict generalization of the best known lower bound for SM protocols for equality testing (the lower bound of Babai and Kimmel) is proved in Section 6.1.

2. If almost one-way functions do not exist, then there exists a *polynomial-time* reduction from any authenticator to a CM protocol for equality testing. This (more technical) stage, outlined in Section 6.2, uses the fact that if almost one-way functions do not exist then there exists an efficient algorithm for learning ACDs (see Section 3.3 and [NR06]).

From these two claims we derive our result: assume there exists a polynomial-time authenticator (or online memory checker) that breaks the lower bound of Theorem 5.3 in a computational setting. If almost one-way functions do not exist, then by Item 2 outlined above, there exist a polynomial-time CM protocol for equality that breaks the CM lower bound in a computational setting. By Item 1, this implies that (standard) one-way functions *do* exist (and in particular so do almost one-way functions), a contradiction! The result is stated in Theorem 6.2 (for authenticators) and Corollary 6.3 (for online memory checkers).

Discussion. It is instructive to note why we can only show that efficient authenticators imply *almost* one-way functions. If we only started out by assuming (for contradiction) that *standard* one-way functions do not exist, then we could only get an efficient reduction from authenticators to CM protocols (as in Item 2) for *infinitely many input lengths*. A low-communication CM protocol that is secure for infinitely many input lengths only implies the existence of *almost* one-way functions. Thus we would get that one-way functions do not exist, but almost one-way functions *do* exist, which is not a contradiction. From a higher-level point of view, we only obtain a result for almost one-way functions because we need to use the assumption (made for contradiction) that cryptographic hardness does not exist, in two places: for making the adversary efficient (which is more standard), but also for making the *construction* of the cryptographic primitive efficient (which is less standard).

6.1 Breaking The CM Lower Bound implies One-Way Functions

In this section we show that breaking a Babai and Kimmel [BK97]-like lower bound in a computational setting implies the existence of one-way functions. Note that, in general, the existence of an information-theoretic lower bound does not imply that breaking the lower bound in a computational setting implies the existence of one-way functions (see the discussion on the relationship between cryptographic primitives in Section 1.2). In this specific case we are able to obtain such a result:

Theorem 6.1. *One-way functions exist if and only if there exists a polynomial time CM protocol $\mathcal{P}$ for equality testing that is secure in the computational setting, such that $\mathcal{P}$ has success probability 0.83, message lengths $s(n)$ and $q(n)$ from Alice and Bob (respectively), public message length at most $0.01 \cdot n$, and where $s(n) \times q(n) \leq c \cdot n$ (c is specified in the proof).*

Proof. We begin by noting that if one-way functions exist, then there exist CM protocols for equality testing with low communication complexity. This is since one-way functions can be used to build UOWHFs (see [NY89], Rompel [Rom90], and Katz and Koo [KK05]), that can in turn be used to construct CM protocols for equality testing with high success probability, a short public messages and $s(n), q(n) \leq \kappa$ for a κ that allows (secret key) cryptography. Without defining UOWHFs or delving into the details: the public randomness r_p is a random function h in the UOWHF family, Alice sends Carol $h(x)$ and Bob sends Carol $h(y)$. The security is immediate from the definition of UOWHFs, which guarantees that if Alice's input x is specified first, and *only afterwards* the hash function h is revealed, then no efficient adversary can find $y \neq x$ s.t. $h(x) = h(y)$.

The converse is less immediate, and we divide the proof into several parts. We assume (from here through the end of the proof) that $\mathcal{P}$ is a polynomial time CM protocol for equality testing with success probability greater than 0.83 and message lengths as specified in the theorem. We begin with some notation: Recall that Alice and Bob's inputs in $\{0,1\}^n$ are x and y respectively, and Alice, Bob and Carol's private random inputs are r_A, r_B, r_C respectively (recall

also that Carol can see Bob's private random coins). The public random input is r_p. Denote the "private" messages sent by Alice and Bob and the "public" message by m_A,m_B and m_p respectively. We also refer to these messages as functions of the inputs and random strings: $M_A(x, r_A, r_p), M_B(y, r_B, M_p(x, r_A, r_p), r_p), M_p(x, r_A, r_p)$.

Intuition. We now turn to constructing a one-way function from an efficient polynomial-time CM protocol. Generate k uniformly random strings, $r_B^1, \ldots, r_B^k$, for Bob, where $k = 200s(n) + 2500$, and take:

$$m_p = M_p(x, r_A, r_p), m_B^i = M_B(x, r_B^i, m_p, r_p)$$

We will show that the following function is *distributionally* one-way:

$$f(x, r_A, r_p, r_B^1, \ldots, r_B^k) = (r_p, r_B^1, \ldots, r_B^k, m_p, m_B^1, \ldots, m_B^k)$$

We begin with some (*very* high-level) intuition. The function receives (x, r_A, r_p), simulates Alice on input x with randomness (r_A, r_p) to generate m_p, and then simulates Bob on input (x, r_p, m_p) with k random strings to generate k messages $(m_B^1, \ldots, m_B^k)$. The messages are each of length $q(n)$, and there are $k = O(s(n))$ messages. We follow [BK97] and show that with very high probability the set of messages $(m_B^1, \ldots, m_B^k)$ "characterizes" Bob's behavior on input (x, r_p, m_p) (for *any* message m_A that Alice sent). If $q(n) \times s(n) \leq c \cdot n$ then the function f "shrinks" its input significantly (the output is shorter than x), and if f is not distributionally one-way then by inverting it an adversary can find a random $x' \neq x$ such that $(m_B^1, \ldots, m_B^k)$ "characterize" Bob's behavior both on x and on x'. This will imply a contradiction, since if Bob's behavior on (x, r_p, m_p) and (x', r_p, m_p) is similar (for *any* m_A) then the protocol cannot have high success probability both when Bob's input is x and when it is x' (recall Carol's output should be 1 if Bob's input is x and 0 if it is x'). The adversary can choose Bob's input from $\{x, x'\}$ at random, and the protocol will not have high success probability.

Lemma 6.1. *If $\mathcal{P}$ is a polynomial time CM protocol for equality testing as specified in Theorem 6.1 then the following function is distributionally one-way:*

$$f(x, r_A, r_p, r_B^1, \ldots, r_B^k) = (r_p, r_B^1, \ldots, r_B^k, m_p, m_B^1, \ldots, m_B^k)$$

Proof. We begin with some more notation:

- **Message sets:** Let Ω and Φ be Alice and Bob's message sets (respectively).
- **Message distributions:** Let $\alpha_{(x,r_p,m_p)}$ be the distribution of Alice's message on Ω given that Alice's input was x, that the public randomness was r_p and that the public message that Alice sent was m_p.
- **The strength of messages:** Denote by $F(y, m_A, r_p, m_p)$ the probability that Carol outputs 1 when Bob's input was y, given that Alice sent private message m_A, with public randomness r_p and public message m_p (this probability is over r_B and r_C). We call $F(y, m_A, r_p, m_p)$ the *strength* of (m_A, r_p, m_p) for y.

Suppose the function is not distributionally one-way, then there exists some PPTM $\mathcal{M}$ that comes $\frac{1}{100}$-close to inverting f for infinitely many n's. We will design an adversary $\mathcal{A}$ operating

against $\mathcal{P}$, which will use $\mathcal{M}$. In our analysis we will treat $\mathcal{M}$ as finding a truly random inverse, which will not greatly increase $\mathcal{A}$'s success probability ($\mathcal{A}$ will only use $\mathcal{M}$ once).

$\mathcal{A}$ selects a random input x for Alice and then receives r_p, m_p. $\mathcal{A}$ will then flip k random strings $r_B^1 \dots r_B^k$ and simulate Bob on each of these random strings as if Bob's input were x and the public messages were (r_p, m_p). These simulations yield k messages $m_B^1 \dots m_B^k \in \Phi$:

$$m_B^i = M_B(x, r_B^i, m_p, r_p)$$

$\mathcal{A}$ will then use $\mathcal{M}$ to find a random inverse of $(r_p, r_B^1, \dots, r_B^k, m_p, m_B^1, \dots, m_B^k)$, let this inverse be $(x', r'_A, r_p, r_B^1, \dots, r_B^k)$. Note that $\mathcal{A}$ is indeed inverting f on a uniformly random input (which is the input distribution for which $\mathcal{A}$ is guaranteed to work), since x and the random strings were all selected uniformly and at random. We will show that with high probability $x \neq x'$, and that the success probability of $\mathcal{P}$ cannot be high (given x, r_p, m_p) both when $y = x$ and when $y = x'$. The adversary $\mathcal{A}$ will select uniformly at random an input y to Bob from $\{x, x'\}$, and thus $\mathcal{P}$'s success probability will not be high when it operates against $\mathcal{A}$.

We follow [BK97] and for any (y, r_p, m_p) and some message $m_A \in \Omega$ we examine the random variable that is Carol's output given that Alice sent the message m_A, the public randomness and message were (r_p, m_p) and the input to Bob was y. We examine k such random variables, and the randomness is over random strings r_B^i and r_C^i used by Bob and Carol (respectively). These k random variables are $\{\xi_i(m_A)\}_{i=1}^k$, where $\xi_i(m_A)$ is Carol's output on $(m_A, m_B^i, m_p, r_B^i, r_C^i, r_p)$. The following claim (a modified version of Lemma 2.3 of [BK97]) states that with very high probability the multiset of messages $\{m_B^1 \dots m_B^k\}$ can be used to estimate, for any m_A, the strength of (m_A, r_p, m_p) for y.

Claim 6.1. *For any (y, r_p, m_p), with probability at least $\frac{99}{100}$ over $r_B^1 \dots r_B^k$, for any $m_A \in \Omega$:*

$$\Pr_{r_C^1 \dots r_C^k}\left[\left|\frac{1}{k}\sum_{i=1}^{k}\xi_i(m_A) - F(y, m_A, r_p, m_p)\right| > 0.1\right] < \frac{1}{2}$$

Proof. For fixed y, r_p, m_p, m_A:

$$\forall i, E_{r_B^i, r_C^i}(\xi_i(m_A)) = F(y, m_A, r_p, m_p)$$

The ξ_i's are independent random variables, and as [BK97] note, by the Chernoff Bound:

$$\Pr_{r_B^1 \dots r_B^k, r_C^1 \dots r_C^k}\left[\left|\frac{1}{k}\sum_{i=1}^{k}\xi_i(m_A) - F(y, m_A, r_p, m_p)\right| > 0.1\right] < 2e^{-k/200} \leq \frac{1}{200|\Omega|}$$

This implies that with probability at least $\frac{99}{100}$ over $r_B^1 \dots r_B^k$:

$$\Pr_{r_C^1 \dots r_C^k}\left[\left|\frac{1}{k}\sum_{i=1}^{k}\xi_i(m_A) - F(y, m_A, r_p, m_p)\right| > 0.1\right] < \frac{1}{2|\Omega|}$$

And thus, by the union bound, with probability at least $\frac{99}{100}$ over $r_B^1 \dots r_B^k$, for *every* $m_A \in \Omega$:

$$\Pr_{r_C^1 \dots r_C^k}\left[\left|\frac{1}{k}\sum_{i=1}^{k}\xi_i(m_A) - F(y, m_A, r_p, m_p)\right| > 0.1\right] < \frac{1}{2}$$

□

We will say that $r_B^1 \dots r_B^k$ *approximate* F for (y, r_p, m_p) if they satisfy the conditions of Claim 6.1.

Claim 6.2. *With probability at least* $\frac{9}{10}$*, by inverting* f*,* $\mathcal{A}$ *finds* $x' \neq x$ *such that* $r_B^1 \dots r_B^k$ *approximate* F *for* (x, r_p, m_p) *and for* (x', r_p, m_p)*.*

Proof. Let $(x, r_A, r_p, r_B^1 \dots r_B^k)$ be the input to f as induced by $\mathcal{A}$'s selection of random inputs and Alice's random coin flips, and suppose $(r_p, r_B^1, \dots, r_B^k, m_p, m_B^1, \dots, m_B^k)$ is f's output. Let $(x', r'_A, r_p, r_B^1, \dots, r_B^k)$ be the result of activating $\mathcal{M}$ on f's output.

We assumed $s(n) \times q(n) \leq \frac{n}{4000}$, and assume also $q(n) \leq \frac{n}{250000}$ (we take $c = \frac{1}{4000}$ and assume $q(n)$ is not too large on its own). We know that $|x| = n$, whereas:

$$|m_p| \leq \frac{n}{100}$$

$$|m_B^1, \dots, m_B^k| \leq k \cdot q(n) = 200 s(n) \cdot q(n) + 2500 \cdot q(n) \leq \frac{n}{20} + \frac{n}{100} = \frac{6n}{100}$$

We conclude that (without $(r_p, r_B^1 \dots r_B^k)$, which are simply written into f's output) the length of f's output is less than $\frac{7}{100}$ times the length of x. This implies that when we use $\mathcal{M}$ to invert f on $(r_p, r_B^1, \dots, r_B^k, m_p, m_B^1, \dots, m_B^k)$, with probability at least $1 - \frac{7}{100}$ the PPTM $\mathcal{M}$ returns an output $(x', r'_A, r_p, r_B^1, \dots, r_B^k)$ such that $x \neq x'$ (this is true for any $(r_A, r_p, r_B^1 \dots r_B^k)$, where the probability is over the selection of x and over $\mathcal{M}$'s random coins).

By Claim 6.1, for any (y, r_p, m_p), with probability at least $1 - \frac{1}{100}$ the random strings $(r_B^1, \dots, r_B^k)$ approximate F for (y, r_p, m_p). Thus with probability at least $1 - \frac{1}{50}$, $(r_B^1, \dots, r_B^k)$ approximate F both for (x, r_p, m_p) and for (x', r_p, m_p). By taking a union bound, we conclude that with probability greater than $1 - \frac{1}{10}$ both $x \neq x'$ and $(r_B^1, \dots, r_B^k)$ approximate F both for (x, r_p, m_p) and for (x', r_p, m_p). □

Claim 6.3. *If* $r_B^1 \dots r_B^k$ *approximate* F *for* (x, r_p, m_p) *and for* (x', r_p, m_p)*, then, given that Alice's input was* x *and her public outputs were* r_p *and* m_p*,* $\mathcal{P}$*'s success probability when* y *is selected randomly from* $\{x, x'\}$ *is at most* 0.8

Proof. We define the sets of weak and strong outputs of Alice for an input y with public messages (r_p, m_p) to Bob:

$$W_{(y,r_p,m_p)} \triangleq \{m_A \in \Omega | F(y, r_p, m_p, m_A) < 0.4\}$$

$$S_{(y,r_p,m_p)} \triangleq \{m_A \in \Omega | F(y, r_p, m_p, m_A) > 0.6\}$$

We know that $f(x, x) = 1$ and $f(x, x') = 0$. Assume for a contradiction that $\mathcal{P}$'s success probability, given that Alice's input was x and her public outputs were r_p and m_p, when y is selected randomly from $\{x, x'\}$, is at least 0.8. Thus the sum of $\mathcal{P}$'s success probabilities on x and x' is at least 1.6. This implies that:

$$\alpha_{(x,r_p,m_p)}[S_{(x,r_p,m_p)}] + \alpha_{(x,r_p,m_p)}[W_{(x',r_p,m_p)}] > 1$$

Since the success probability for messages that are not in $S_{(x,r_p,m_p)}$ for x or in $W_{(x',r_p,m_p)}$ for x' is at most 0.4. We conclude that there must be some $m_A^0 \in S_{(x,r_p,m_p)} \bigcap W_{(x',r_p,m_p)}$. However, since $r_B^1 \dots r_B^k$ approximate F for both (x, r_p, m_p) and (x', r_p, m_p):

$$\Pr_{r_C^1 \dots r_C^k} \left[\left| \frac{1}{k} \sum_{i=1}^{k} \xi_i(m_A^0) - F(x, m_A^0, r_p, m_p) \right| > 0.1 \right] < \frac{1}{2}$$

$$\Pr_{r_C^1 \cdots r_C^k}\left[\left|\frac{1}{k}\sum_{i=1}^{k}\xi_i(m_A^0) - F(x', m_A^0, r_p, m_p)\right| > 0.1\right] < \frac{1}{2}$$

Since $F(x, m_A^0, r_p, m_p) > 0.6$ and $F(x', m_A^0, r_p, m_p) < 0.4$ we conclude that one of the above events must have probability at least $\frac{1}{2}$, a contradiction. □

Claim 6.4. *If f is not distributionally one-way then $\mathcal{P}$ fails versus $\mathcal{A}$ with probability at least 0.17*

Proof. We have seen that with probability at least 0.9 $\mathcal{A}$ finds inputs x, y to Alice and Bob such that $\mathcal{P}$ fails with probability at least 0.2. We conclude that $\mathcal{P}$'s success probability against an adversary that can truly find a random inverse of f is at most 0.82. Since $\mathcal{A}$ uses a distribution that is $\frac{1}{100}$-close to that of a truly random inverse (and samples only once from this distribution), we conclude that $\mathcal{P}$'s success probability when it operates against $\mathcal{A}$ is certainly no greater than 0.83. □

From Claim 6.4 we derive a clear contradiction to the fact that $\mathcal{P}$ succeeds with probability greater than 0.83 against any polynomial time adversary. Thus f must be distributionally one-way. □

We conclude that indeed one-way functions exist if and only if there exists a polynomial time CM protocol $\mathcal{P}$ for equality testing with success probability greater than 0.83, message lengths $s(n)$ and $q(n)$ from Alice and Bob (respectively), and public message length at most $\frac{n}{100}$, such that $s(n) \times q(n) \leq c \cdot n$. Note that the proof of Theorem 6.1 holds even if Carol has unbounded computational power: only Alice, Bob and the adversary are computationally bounded. □

6.2 Constructing a Polynomial-Time Protocol

In this section we will show that the existence of efficient polynomial-time memory checkers (and authenticators) in the computational setting implies the existence of almost one-way functions. This is done by showing a reduction from any polynomial-time authenticator to a *polynomial-time* CM protocol for equality testing. By the results of the previous subsection, this implies the existence of almost one-way functions.

Theorem 6.2. *If there exists a polynomial time authenticator $(\mathcal{E}, \mathcal{D}, \mathcal{V})$ that is secure in the computational setting, with space complexity $s(n)$ and query complexity $q(n)$, where $s(n) \times q(n) \leq d \cdot n$ (for some small constant d), then there exist almost one-way functions. The converse is true for standard one-way functions.*

Proof. If (standard) one-way functions exist then by the construction of Section 4.2 there exist authenticators with constant query complexity and small polynomial (sub-linear) space complexity. It remains to prove the other direction of the claim.

If there exists an efficient polynomial time authenticator, but almost one-way functions do not exist, then on one hand in this subsection we will show that the authenticator can be used to construct an efficient polynomial time CM protocol for equality testing, but on the other hand in the previous subsection we showed that there exists an adversary that makes the CM protocol fail. Thus we obtain a contradiction.

Let $(\mathcal{E}, \mathcal{D}, \mathcal{V})$ be the polynomial-time authenticator promised in the theorem statement. We will show that (unless almost one-way functions exist) the construction presented in Section 5.5

can be modified to be a *polynomial-time* CM protocol for equality testing (for infinitely many n's). We need to show that if almost one-way functions do not exist then Alice and Bob can run in polynomial-time. Note that it is not required that Carol run in polynomial time (indeed we do not know how to make Carol efficient). We construct $\mathcal{V}_{mod}$ by the "bait and switch" technique, as in Section 5.4 (note that $\mathcal{V}_{mod}$ may not be efficient in its critical invocation). We follow the players' descriptions in Section 5.5, the description of each of the players' actions is in *italic type*, followed by the polynomial-time implementation.

Alice. We begin by showing that Alice can be run in polynomial time.

1. *"Encode x using $\mathcal{E}$ into a public output $p_x^0 \in \{0,1\}^{u(n)}$ and a secret output $s_x^{dummy} \in \{0,1\}^{s(n)}$. $\mathcal{E}$'s randomness is taken from the public random string r_p".*

   In her first step Alice activates $\mathcal{E}$ on her input x to obtain public and secret encodings. If the authenticator is polynomial time then this step can be run efficiently.

2. *"Run the learning algorithm $\mathcal{L}$ using the public random string r_p to learn s_x^{dummy} with error probability $\delta = \frac{1}{100}$ and statistical distance $\varepsilon = \frac{1}{100}$. The learning algorithm runs $\mathcal{V}_{mod}$ for k steps, where $k = O(s(n))$, and outputs some conjecture $s_x^{\mathcal{L}}$ as to $\mathcal{E}$'s secret output. $\mathcal{L}$'s run also determines a final public encoding p_x^k. The public message includes s_x^{dummy}, $s_x^{\mathcal{L}}$ and the vector v_a of $O(s(n) \times q(n))$ bits accessed (read or written) during $\mathcal{L}$'s run".*

   Alice needs to run the learning algorithm $\mathcal{L}$. By the results of [NR06], if almost one-way functions do not exist and the adversary chooses x at random, then for all but finitely many n's, $\mathcal{V}_{mod}$'s access pattern distribution can be learned in polynomial time (with error probability $\delta = \frac{1}{100}$ and statistical distance $\varepsilon = \frac{1}{100}$).[14] This result is derived from [NR06] by formulating the problem of learning the access pattern as learning an adaptively changing distribution, as was done in the proof of Corollary 5.2. Note that before its critical invocation $\mathcal{V}_{mod}$ simply activates $\mathcal{V}$ and is thus efficient. Alice activates the efficient $\mathcal{V}_{mod}$ for k steps, where $k = O(s(n))$, and thus the public message is indeed of size $O(s(n) \times q(n))$.

3. *"Select a string r' uniformly and at random from the set of random strings for which on input x the encoding algorithm $\mathcal{E}$ outputs the public encoding p_x^0 and $\mathcal{V}_{mod}$'s access patter is $\vec{a}$. This choice is made using Alice's* private *random coins. Let $s_x \in \{0,1\}^{s(n)}$ be $\mathcal{E}$'s secret encoding on input x with random string r'. Send s_x to Carol".*

   Finally, Alice needs to find a random inverse of the public results she obtained by activating the efficient $\mathcal{E}$ and $\mathcal{V}_{mod}$ with the public randomness. If almost one-way functions do not exist, then since all of Alice's inputs were chosen uniformly and at random, for all but finitely many n's, Alice can certainly come $\frac{1}{200}$-close to finding a random inverse in polynomial time.

We conclude that if almost one-way functions do not exist then Alice can be made polynomial time, and for all but finitely many n's Alice's behavior will be very close to the behavior of the information-theoretic Alice described in Section 5.5.

[14] Note that the CM adversary constructed in the proof of Theorem 6.1 indeed picks x uniformly at random.

Bob. We now show that Bob can also be made efficient. One tricky point is that a reasonable adversary would probably not select Bob's input y uniformly and at random, thus we do not use y in the efficiently computable functions that Bob inverts.

1. *"Simulate $\mathcal{E}$ on y using r_p to obtain public encoding p_y^0. If s_x^{dummy} (as sent by Alice) is not the resulting secret encoding of y send a message to Carol telling her that $x \neq y$".*

   Bob begins by running $\mathcal{E}$, an efficient computation.

2. *"Simulate $\mathcal{L}$'s invocations of $\mathcal{V}_{mod}$ using the initial secret encoding s_x^{dummy}, the random string r_p and with query results as published by Alice in v_a. If the query results published by Alice are inconsistent with the public encoding p_y^0, send a message to Carol telling her that $x \neq y$".*

   Bob runs the learning algorithm $\mathcal{L}$. We have already seen in the analysis of Alice that this can be done efficiently.

3. *"Repeat the following step c times, in the i-th iteration:*

   Choose uniformly and at random some random string $r_b^i \in \mathcal{V}^{-1}_{s_x^{\mathcal{L}}}(\vec{a})$. The uniform random choice is done using Bob and Carol's shared private *coins. Simulate the k invocations of $\mathcal{V}_{mod}$ using $s_x^{\mathcal{L}}$, r_b^i and $\vec{a}$. This invocation results in a secret encoding s_y^i and the public encoding p_y^k. Invoke $\mathcal{V}$ one final time using Bob and Carol's shared private coins, s_y^i and p_y^k. Send the vector v_i^b of bit values accessed to Carol (the vector is at most $q(n)$ bits long)".*

   Bob needs to randomly invert $\mathcal{V}_{mod}$'s activations by the learning algorithm a constant number (c) of times. We examine the following function:

   $$f_B(x, r_{\mathcal{E}}, k, r_{\mathcal{V}}^1, \ldots, r_{\mathcal{V}}^{q(n)}) = (s_x^{\mathcal{L}}, k, \vec{a}, 0 \ldots 0)$$

   Where in the input $q(n) = O(s(n))$ is the maximal number of times the learning algorithm $\mathcal{L}$ may activate $\mathcal{V}_{mod}$, and k is the actual number of times $\mathcal{L}$ activated $\mathcal{V}_{mod}$ when run by Alice. In the output, $s_x^{\mathcal{L}}$ is the secret encoding obtained by running $\mathcal{E}$ on x using the randomness $r_{\mathcal{E}}$, and $\vec{a}$ is the access pattern in $\mathcal{V}_{mod}$'s k activations on the public encoding given as output by $\mathcal{E}$. If $k < q(n)$, then the output is padded with 0's (thus the output length is identical for any choice of k).

   If almost one-way functions do not exit then f_B is not almost distributionally one-way and for all but finitely many n's, for any k,[15] Bob can come $\frac{1}{200 \cdot c}$-close to randomly inverting f_B when the input is selected uniformly and at random and k is fixed. Thus Bob can indeed compute the random strings r_B^i and from them the access pattern bits v_B^i.

We conclude that if almost one-way functions do not exist then Bob can also be made polynomial time, and for all but finitely many n's Bob's behavior will be very close to the behavior of the information-theoretic Bob described in Section 5.5.

[15] This is true for any k, even if it is not chosen at random. The statement is true because there is only a polynomial number of possible k's. See Lemma 4.1 of [NR06] for a full discussion

Conclusion. We have seen that Alice and Bob can be made efficient (assuming almost one-way functions don't exist), and for all but finitely many n's their behavior will not be significantly altered. We now analyze the success probability of the protocol we have constructed in this section. First, by the results of the "bait and switch" technique outlined in Section 5.4, if Alice and Bob could find random inverses in their final step then, since $(\mathcal{E}, \mathcal{D}, \mathcal{V})$'s success probability is at least 0.9 and the bait and switch loses at most a 0.03 success probability (recall the proof of Lemma 5.1), the protocol's success probability would be at least 0.87. We lose another 0.01 probability of success because Alice and Bob only come statistically close to finding random inverses, and the protocol's success probability against an adversary that selects x at random is (for all but finitely many input lengths) greater than 0.85.

To use the results of Theorem 6.1, we note that the adversary outlined in the theorem chooses x uniformly at random. The protocol we outlined succeeds for all but finitely many n's. If almost one way functions do not exist, then neither do standard one-way functions, and the adversary of Theorem 6.1 succeeds for infinitely many n's. Thus there are infinitely many n's for which both the protocol and the adversary succeed, a contradiction! □

Corollary 6.3. *If there exists a polynomial time online memory checker that is secure in the computational setting, with space complexity $s(n)$ and query complexity $q(n)$, where $s(n) \times q(n) \leq d \cdot n$ (for some small constant d), then there exist almost one-way functions. The converse is true for standard one-way functions.*

Proof. By the results of [BEG+94], if (standard) one-way functions exist then there exist very efficient polynomial-time online memory checkers.

As noted in Theorem 4.1, if there exists a polynomial time online memory checker that is secure in the computational setting, with space complexity $s(n)$ and query complexity $q(n)$, then there exists a polynomial time authenticator that is secure in the computational setting with space complexity $O(s(n))$ and query complexity $O(q(n))$. By Theorem 6.2 this implies the existence of almost one-way functions. □

7 Conclusions and Open Problems

We have shown that online memory checking (and authenticators) may be prohibitively expensive for many applications. On the other hand, for *offline* memory checking there exist very good information-theoretic schemes. This implies that applications requiring memory checking (with good parameters) should make cryptographic assumptions, or use an offline version of the problem.

In a computational setting with cryptographic assumptions (i.e. the existence of one-way functions), there is an interesting gap between the performance of authenticators and online memory checkers. While we know how to build authenticators with $O(1)$ query complexity, the best constructions of online memory checkers require $O(\log n)$ query complexity. This is due to the fact that memory checkers need to handle *replay attacks*, but it is natural to ask whether replay attacks can be handled without incurring a logarithmic overhead. Recently, [DNRV08] have shown that this cannot be achieved by *non-adaptive* memory checkers. It remains open to either show an *adaptive* checker that avoids the logarithmic overhead, or to show a general lower bound.

Another interesting question is the connection between combinatorial lower bounds (or lower bounds in communication complexity) and cryptographic assumptions that are required for breaking them in a computational setting. There is no clear way to break many information-theoretic

lower bounds in a computational setting. Moreover, even if an information-theoretic lower bound can be broken in a computational setting, showing that breaking it is equivalent to a widely used cryptographic primitive (e.g. one-way functions) may be difficult. The problem of private information retrieval with a single server is a good example: while a tight information theoretic lower bound was quickly found ([CKGS98]), it required significant work before this lower bound was broken in a computational setting ([KO97], [KO00]) and the cryptographic assumptions necessary for breaking it were well understood ([BIKM99] and [CMO00]).

8 Acknowledgements

We thank Ronen Gradwohl and Amir Yehudayoff for helpful discussions and for reading a preliminary version of this document. We also thank the anonymous referees for their thorough and insightful comments.

References

[ABC+07] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. Cryptology ePrint Archive, Report 2007/202, 2007.

[Ajt02] Miklós Ajtai. The invasiveness of off-line memory checking. In *STOC*, pages 504–513, 2002.

[BEG+94] Manuel Blum, William S. Evans, Peter Gemmell, Sampath Kannan, and Moni Naor. Checking the correctness of memories. *Algorithmica*, 12(2/3):225–244, 1994.

[BGG94] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Incremental cryptography: The case of hashing and signing. In *CRYPTO*, pages 216–233, 1994.

[BGG95] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Incremental cryptography and application to virus protection. In *STOC*, pages 45–56, 1995.

[BIK05] Amos Beimel, Yuval Ishai, and Eyal Kushilevitz. General constructions for information-theoretic private information retrieval. *J. Comput. Syst. Sci.*, 71(2):213–247, 2005.

[BIKM99] Amos Beimel, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. One-way functions are essential for single-server private information retrieval. In *STOC*, pages 89–98, 1999.

[BIN97] Mihir Bellare, Russell Impagliazzo, and Moni Naor. Does parallel repetition lower the error in computationally sound protocols? In *FOCS*, pages 374–383, 1997.

[BJO08] Kevin D. Bowers, Ari Juels, and Alina Oprea. Proofs of retrievability: Theory and implementation. Cryptology ePrint Archive, Report 2008/175, 2008.

[BK95] Manuel Blum and Sampath Kannan. Designing programs that check their work. *Journal of the ACM*, 42(1):269–291, 1995.

[BK97] László Babai and Peter G. Kimmel. Randomized simultaneous messages: Solution of a problem of yao in communication complexity. In *IEEE Conference on Computational Complexity*, pages 239–246, 1997.

[BSGH+04] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Robust pcps of proximity, shorter pcps and applications to coding. In *STOC*, pages 1–10, 2004.

[BSS05] Eli Ben-Sasson and Madhu Sudan. Simple pcps with poly-log rate and query complexity. In *STOC*, pages 266–275, 2005.

[CKGS98] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *Journal of the ACM*, 45(6):965–981, 1998.

[CMO00] Giovanni Di Crescenzo, Tal Malkin, and Rafail Ostrovsky. Single database private information retrieval implies oblivious transfer. In *EUROCRYPT*, pages 122–138, 2000.

[CSG+05] Dwaine E. Clarke, G. Edward Suh, Blaise Gassend, Ajay Sudan, Marten van Dijk, and Srinivas Devadas. Towards constant bandwidth overhead integrity checking of untrusted data. In *IEEE Symposium on Security and Privacy*, pages 139–153, 2005.

[DNRV08] Cynthia Dwork, Moni Naor, Guy N. Rothblum, and Vinod Vaikuntanathan. How efficient can memory checking be? Manuscript, 2008.

[FS95] Katalin Friedl and Madhu Sudan. Some improvements to total degree tests. In *ISTCS*, pages 190–198, 1995.

[FT01] Amos Fiat and Tamir Tassa. Dynamic traitor tracing. *J. Cryptology*, 14(3):211–223, 2001.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct pseudorandom functions. *Journal of the ACM*, 33(2):792–807, 1986.

[GMS74] Edgar N. Gilbert, Florence J. MacWilliams, and Neil J. A. Sloane. Codes which detect deception. *Bell System Technical Journal*, 53(3):405–424, 1974.

[GN93] Peter Gemmell and Moni Naor. Codes for interactive authentication. In *CRYPTO*, pages 355–367, 1993.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the ACM*, 43(3):431–473, 1996.

[Gol87] Oded Goldreich. Towards a theory of software protection and simulation by oblivious rams. In *STOC*, pages 182–194, 1987.

[Gol01] Oded Goldreich. *The Foundations of Cryptography - Volume 1.* Cambridge University Press, 2001.

[GS06] Oded Goldreich and Madhu Sudan. Locally testable codes and pcps of almost-linear length. *J. ACM*, 53(4):558–655, 2006.

[HILL99] J. Håstad, R. Impagliazzo, L. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[IL89] Russel Impagliazzo and Michael Luby. One-way functions are essential for complexity based cryptography. In *FOCS*, pages 230–235, 1989.

[Imp95] Russel Impagliazzo. A personal view of average-case complexity. In *Proceedings of the 10th Annual Conference on Structure in Complexity Theory*, pages 134–147, 1995.

[JK07] Ari Juels and Burton Kaliski. Pors: proofs of retrievability for large files. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, pages 584–597, New York, NY, USA, 2007. ACM.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *STOC*, pages 723–732, 1992.

[KK05] Jonathan Katz and Chiu-Yuen Koo. On constructing universal one-way hash functions from arbitrary one-way functions. Cryptology ePrint Archive, Report 2005/328, 2005.

[KNR99] Ilan Kremer, Noam Nisan, and Dana Ron. On randomized one-round communication complexity. *Computational Complexity*, 8(1):21–49, 1999.

[KO97] Eyal Kushilevitz and Rafail Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *FOCS*, pages 364–373, 1997.

[KO00] Eyal Kushilevitz and Rafail Ostrovsky. One-way trapdoor permutations are sufficient for non-trivial single-server private information retrieval. In *EUROCRYPT*, pages 104–121, 2000.

[KT00] Jonathan Katz and Luca Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *STOC*, pages 80–86, 2000.

[Mic94] Silvio Micali. Cs proofs (extended abstract). In *FOCS*, pages 436–453, 1994.

[MPSW05] Silvio Micali, Chris Peikert, Madhu Sudan, and David A. Wilson. Optimal error correction against computationally bounded noise. In *TCC*, pages 1–16, 2005.

[NN93] Joseph Naor and Moni Naor. Small–bias probability spaces: Efficient constructions and applications. *SIAM Journal on Computing*, 22(4):838–856, 1993.

[NNL01] Dalit Naor, Moni Naor, and Jeffery Lotspiech. Revocation and tracing schemes for stateless receivers. In *CRYPTO*, pages 41–62, 2001.

[NR06] Moni Naor and Guy N. Rothblum. Learning to impersonate. In *ICML*, pages 649–656, 2006.

[NS96] Ilan Newman and Mario Szegedy. Public vs. private coin flips in one round communication games (extended abstract). In *STOC*, pages 561–570, 1996.

[NSS06] Moni Naor, Gil Segev, and Adam Smith. Tight bounds for unconditional authentication protocols in the manual channel and shared key models. In *CRYPTO*, pages 214–231, 2006.

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *STOC*, pages 33–43, 1989.

[Ost90] Rafail Ostrovsky. Efficient computation on oblivious rams. In *STOC*, pages 514–523, 1990.

[OW93] Rafail Ostrovsky and Avi Wigderson. One-way fuctions are essential for non-trivial zero-knowledge. In *ISTCS*, pages 3–17, 1993.

[rJ72] Jørn Justesen. Class of constructive asymptotically good algebraic codes. *IEEE Transactions on Information Theory*, 18(5):652–656, 1972.

[Rom90] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 387–394, 1990.

[SW08] Hovav Shacham and Brent Waters. Compact proofs of retrievability. Cryptology ePrint Archive, Report 2008/073, 2008.

[WC81] Mark N. Wegman and Larry Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22:265–279, 1981.

[Yao79] Andrew Chi-Chih Yao. Some complexity questions related to distributive computing (preliminary report). In *STOC*, pages 209–213, 1979.