Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 146 (2021)

Sample-Based Proofs of Proximity

Guy Goldberg [*]
Weizmann Institute

Guy N. Rothblum [†]
Weizmann Institute

Abstract

Suppose we have random sampling access to a huge object, such as a graph or a database. Namely, we can observe the values of *random* locations in the object, say random records in the database or random edges in the graph. We cannot, however, query locations of our choice. Can we verify complex properties of the object using only this restricted sampling access?

In this work, we initiate the study of *sample-based* proof systems, where the verifier is extremely constrained; Given an input, the verifier can only obtain samples of uniformly random and i.i.d. locations in the input string, together with the values at those locations. The goal is verifying complex properties in sublinear time, using only this restricted access. Following the literature on Property Testing and on Interactive Proofs of Proximity (IPPs), we seek proof systems where the verifier accepts every input that has the property, and with high probability rejects every input that is *far* from the property.

We study both interactive and non-interactive sample-based proof systems, showing:

- On the positive side, our main result is that rich families of properties / languages have sub-linear sample-based interactive proofs of proximity (SIPPs). We show that every language in $\mathcal{NC}$ has a SIPP, where the sample and communication complexities, as well as the verifier's running time, are $\widetilde{O}(\sqrt{n})$, and with polylog(n) communication rounds. We also show that every language that can be computed in polynomial-time and bounded-polynomial space has a SIPP, where the sample and communication complexities of the protocol, as well as the verifier's running time are roughly $\sqrt{n}$, and with a constant number of rounds.

  This is achieved by constructing a reduction protocol from SIPPs to IPPs. With the aid of an untrusted prover, this reduction enables a restricted, sample-based verifier to simulate an execution of a (query-based) IPP, even though it cannot query the input. Applying the reduction to known query-based IPPs yields SIPPs for the families described above.
- We show that every language with an adequate (query-based) property tester has a 1-round SIPP with *constant* sample complexity and logarithmic communication complexity. One such language is equality testing, for which we give an explicit and simple SIPP.
- On the negative side, we show that *interaction* can be essential: we prove that there is no *non*-interactive sample-based proof of proximity for equality testing.
- Finally, we prove that *private coins* can dramatically increase the power of SIPPs. We show a strong separation between the power of public-coin SIPPs and private-coin SIPPs for Equality Testing.

[*]Email: guy.goldberg@weizmann.ac.il. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819702) and from the Israel Science Foundation (grant number 5219/17).

[†]Email: rothblum@alum.mit.edu. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819702), from the Israel Science Foundation (grant number 5219/17), and from the Simons Foundation Collaboration on the Theory of Algorithmic Fairness.

ISSN 1433-8092

Table of Contents

1 Introduction

A fundamental question in the theory of computing is understanding the power of efficiently verifiable proof systems. This question was studied in various computational models, and with various restrictions on the verifier. In this work we initiate a study of proof systems with an extremely restricted *sample-based* verifier. Such a verifier can only obtain samples of uniformly random and i.i.d. locations in the input string, together with the values at those locations. It cannot query locations of its choice. The goal is verifying complex properties in sublinear time, using only this restricted access. We begin this introduction with background on interactive proof systems and proof systems with sublinear verifiers, before presenting more formally the model we study in this work.

An interactive proof system [GMR89, BM88] is an interactive protocol between a prover and a weaker verifier, in which the prover tries to convince the verifier of the validity of some computational statement. The computational statement is most commonly considered to be $x \in L$, where x is an input (that is usually known to both the prover and the verifier), and L is some language. We require that if $x \in L$, then there exists a prover strategy that makes the verifier accept ("completeness"). A prover that follows this strategy is named an "honest" prover. The second requirement ("soundness"), is that if $x \notin L$, then *no* prover can make the verifier accept, except for a small probability of error. The famous $\mathcal{IP} = \mathcal{PSPACE}$ Theorem [LFKN92, Sha92] shows that interactive proof systems with polynomial-time verifiers are remarkably powerful. Such proof systems can be used to prove any statement computable in polynomial space.

There are various resources one can consider when restricting the verifier in a proof system. One challenging frontier is sublinear-time verification [EKR04, RVW13]. In such a proof system, the verifier cannot even read the input in its entirely. Following the literature on sublinear-time algorithms and property testing [RS96, GGR98], soundness is relaxed. In an *Interactive Proof of Proximity* (IPP), the verifier has to only make an approximate verification. Instead of rejecting any inputs $x \notin L$, the verifier is required to reject (with high probability) only inputs that are *far* from L, where we say that x is ϵ-far from L if the (fractional) Hamming distance of x from every input in L is at least $\epsilon \in (0, 1)$. The verifier should still accept any input that is in the language. IPPs are known for general families of languages [RVW13, RRR16]. This stands in contrast to the study of property testing, where testers rely on the specific structure (e.g combinatorial or algebraic) of the problem at hand.

In a standard IPP, the verifier has query access to the input. The input $x \in \{0, 1\}^n$ is treated as an oracle. For each query, the verifier chooses an index $i \in [n]$, and "reads" the i-th bit of the input x_i. The main goal of an efficient sublinear proof system is for the verifier to have sublinear running time. In addition, we also consider the *query complexity* of the verifier, which is the number of queries the verifier makes (the number of bits it reads from the input). Other measures of complexity are *communication complexity* (total number of bits exchanged between the prover and the verifier), *round complexity* (where in each round, each party sends a single message to the other party) and the running time of the honest prover. We stress that the prover has full (explicit) access to the input. Whereas we are interested in the running time of the *honest* prover, a cheating prover is unreliable and untrusted, and is computationally unbounded.

We can also consider non-interactive proof systems, which is a more limited type of proofs. In such a proof system, the prover is restricted to send a single message to the verifier (a proof); the verifier cannot send any messages to the prover. In a non-interactive proof of *proximity* [GR18] the verifier has query access to the input, and should have sublinear running time and query complexity. Soundness is relaxed in a similar manner to IPPs, and the verifier needs to reject only inputs that are far from the language. Such proofs are named *Merlin-Arthur Proofs of Proximity*, or *MAPs*. See Section 2 for formal definitions of (query-based) IPPs and MAPs.

1.1 This Work

The queries of a sublinear verifier in a proof system can be restricted in various ways. In this work, we take the restriction on the verifier input access to the extreme. We consider proof systems where the verifier has only *sample-based access* to the input. In this model, the verifier is only provided with uniformly distributed labeled samples from the input. Namely, it draws samples from the input $x \in \{0,1\}^n$, each one of the form (i, x_i). The index i of each sample is distributed uniformly and i.i.d from the set of all possible indices $[n]$, and $x_i \in \{0,1\}$ is the value of the input x at index i. Our work is inspired by the property-testing literature: the sample-based access model was introduced by Goldreich, Goldwasser and Ron [GGR98]. A comprehensive systematic study of sample-based property testers was initiated by Goldreich and Ron [GR16].

SIPPs: The main objects we study in this work are interactive proof systems of proximity, where the verifier has only sample-based access to the input. We call such proof systems *Sample-based Interactive Proofs of Proximity*, or *SIPPs*. Such proof systems are essentially IPPs, where the verifier is restricted to be sample-based, as described above. We emphasize that while the verifier is restricted in its access to the input, the prover still has full access to it. In such proof systems we replace the term *query complexity* with *sample complexity*, as the verifier now draws samples, instead of making queries. Other measures of complexity are similar to the measures of complexity of IPPs. For a formal definition of SIPPs see Definition 2.11.

SMAPs: We are also interested in *non*-interactive proof systems of proximity, where again the verifier is restricted to be sample-based. We call such proof systems *Sample-based Merlin-Arthur proofs of Proximity*, or *SMAPs*. We emphasize that in SMAPs (as in MAPs) the verifier has full access to the *proof*. Its access to the input x is the only additional restriction. As in SIPPs, we use the term *sample complexity* to denote the number of samples the verifier draws from the input. The other important measure of complexity of a SMAP is the *proof length*. For a formal definition of SMAPs see Definition 2.12.

Research Question: We now present the main research question of this work

What are the power and the limitations of interactive and non-interactive proofs of proximity when the verifier has only sample-based access to the input?

A Concrete Motivation: Delegation of Sample-based Computations We find the sample-based access model to be very natural, and worthy of study in its own right. It captures access to objects for which obtaining labeled samples is easy, whereas querying them in arbitrary locations is infeasible. In addition, verification with a sample-based verifier is also motivated by real-world applications. Consider the scenario in which a passive observer (a client) wants to study some complex phenomena, even though it only sees a small random part of the world. For examples, consider a statistician who wants to compute some statistical quantity over a large population (by performing a survey on a small group), a website that tries to learn the preferences of its potential clients (and only sees a random sample comprised of users that currently use it), or a weather forecaster who wishes to understand weather patterns, based on data it receives from a few sensors deployed at random locations. An untrusted company might claim to have already computed a model of the phenomena, based on data that is not available to the observer. The company offers to sell the model to the client. How could the client ascertain that the model indeed represent the reality, when it only sees a small random part of the relevant data?

A solution we propose in this work is to have the company provide a proof for the quality (approximate correctness) of its output. Such a proof can be a "written" proof, in which the company explains succinctly the validity of the claimed model. A second option is to have an interactive proof, in which one asks the company questions regarding the model in order to be convinced of its validity. A written, non-interactive proof is the easier option. Therefore, it is of interest to distinguish between tasks that must have interaction for their verification, and tasks that can be verified by non-interactive proofs. In both cases, the observer would need to consider the resource gap between learning and verifying, where the main resources for verification are the "proof complexity" (length of written proof, or the cost of communicating with the company in interactive proofs), along with the data it needs to gather on its own (the samples). If verifying is easier than learning, it can purchase the model from the company and cost-effectively verify the proof, instead of learning the model on its own.

We note that delegation of computation was considered in previous works, starting with [GKR08], who studied how a a powerful server (but not unbounded) can run a computation for a weaker client, and provide an interactive proof of the output's correctness. It was extended to the case when the verifier can only run in sublinear time in [RVW13], and to the area of machine learning in [GRSY21]. In this work we further extend this line of study, to delegation of computational problems where the access of the client is sample-based.

1.2 Our Results

In this work we show that sample-based interactive proofs of proximity can be very powerful. In some cases, sample-based IPPs are almost as powerful as query-based IPPs. This is perhaps surprising, as sample-based testers are much more limited compared to query-based testers.

Our main result is that rich families of languages have sub-linear SIPPs. In addition, we show that even a single round of communication increases the power of the verifier. On the other hand, interaction is necessary to utilize the power of a prover. We show limits on the power of *non*-interactive proofs of proximity. We next describe our results in more detail.

Interactive Sample-Based Verification for Rich Families: The first family we consider is log-space uniform $\mathcal{NC}$.

Theorem 1. *(informal; see Theorem 5.4) Let L be a language in log-space uniform $\mathcal{NC}$, and let $\epsilon \in (0,1)$ be a fixed constant. Then there is a SIPP with proximity parameter ϵ for L. The sample and communication complexities of the SIPP, as well as the verifier's running time, are $\widetilde{O}(\sqrt{n})$. In addition, the SIPP has poly$(\log n)$ rounds, and the (honest) prover runs in time poly(n).*

The second family is languages that can be computed in polynomial-time and bounded-polynomial space.

Theorem 2. *(informal; see Theorem 5.5) Fix a constant $\sigma \in (0,1)$, and let L be a language that is computable in poly(n)-time and $O(n^\sigma)$-space. Let $\epsilon \in (0,1)$ be a fixed constant. Then there is a SIPP with proximity parameter ϵ for L. The sample and communication complexities of the SIPP, as well as the verifier's running time, are $n^{1/2+O(\sigma)}$. In addition, the SIPP has a constant number of rounds, and the (honest) prover runs in time poly(n).*

In order to prove these theorems, we construct a generic reduction from SIPPs to IPPs. This reduction is the most technically-involved part of this work. We then apply the reduction to the IPPs of [RVW13, RRR16, RR20], and prove Theorem 1 and Theorem 2.

We make several remarks on the SIPPs of Theorem 1 and Theorem 2. First, Kalai and Ruthblum [KR15] showed that, under cryptographic assumptions, the sample and communication complexities cannot both be $o(\sqrt{n})$, so both results are nearly optimal. Both SIPPs use private coins, and we note that this is in contrast to known query-based IPPs for general computations, which use public coins [RVW13, RRR16, RR20]. The power of SIPPs with public coins is an intriguing open question. Finally, the communication and sample complexities can be traded off: For desired sample complexity $s \leq \sqrt{n}$, we can obtain similar results with communication roughly $\frac{n}{s}$. Note, however, that we do not know how to obtain communication complexity $o(\sqrt{n})$ (unlike the case of IPPs). See Section 1.4 for further discussion on the above points.

1-round SIPPs: We show that even with a single round of communication, SIPPs are *exponentially* more powerful than sample-based testers. Specifically, we consider the problem of Equality Testing. In this problem, an algorithm gets as input two binary strings x, y, and needs to accept if $x = y$ and reject if x is far from y. Formally, we define $\mathrm{EQU} = \{(x, x) \mid x \in \{0,1\}^n\} \subseteq \{0,1\}^{2n}$.

Theorem 3. *(informal; see section 3.1) For any fixed $\epsilon > 0$:*

1. *Any* sample-based tester *with proximity parameter ϵ for EQU has sample complexity $s = \Omega(\sqrt{n})$.*
2. *There exists a 1-round, private coin, sample-based interactive proof of proximity for EQU, with proximity parameter ϵ, communication complexity $c = O(\log n/\epsilon)$ and sample complexity $s = O(1/\epsilon)$.*

That is, we prove an exponential separation between the power of sample-based testers and the power of 1-round sample-based interactive proofs of proximity.[1] The first item of Theorem 3 follows from first principles, and for the second item we construct and analyze a simple SIPP. Note that the lower bound in Theorem 3 is tight; for every $\epsilon > 0$ there exists a sample-based tester for EQU with proximity parameter ϵ and sample complexity $O(\sqrt{n}/\epsilon)$.

We then extend and generalize the SIPP for equality to every property that has a non-adaptive and fair (query-based) tester.[2]

Theorem 4. *(informal; see Theorem 3.1) Let $\epsilon > 0$, and let L be a property with non-adaptive (query-based) fair tester, with proximity parameter ϵ, and query complexity $q = q(n, \epsilon)$. Then L has a 1-round SIPP, with proximity parameter ϵ, communication complexity $O(q^2 \cdot \log n)$ and sample complexity $s = O(q)$.*

In particular, if the query complexity q of the query-based tester is constant (i.e., does not depend on n), then the sample complexity of the resulted SIPP is also constant, and its communication complexity is logarithmic. For languages that have POTs (proximity oblivious testers) we give SIPPs with slightly better communication complexity. See Theorem 3.2 for the exact statement and parameters.

Lower Bounds: We showed that sample-based *interactive* proofs of proximity are very powerful, when compared to sample-based testers. Our next result is on the limitation of sample-based *non*-interactive proofs, which we call SMAPs. We show that for the Equality Testing Problem, such proofs are very limited in their power. Namely, we prove the following result:

Theorem 5. *(informal; see Theorem 4.4) Let $\epsilon > 0$. If a SMAP for EQU with proximity parameter ϵ has proof length $p = o(n)$, then its sample complexity is $s = \Omega(\frac{\sqrt{n}}{\log n})$.*

[1] We compare the sample complexity + communication complexity of the SIPP with the sample complexity of the tester.

[2] A tester is said to be *non-adaptive* if it determines all its queries based on its internal coin tosses, independently of the specific (implicit) input it gets. A testers is said to be *fair* if each of its queries to an input $x \in \{0,1\}^n$ is uniformly distributed in $[n]$.

As observed by [RVW13], if we allow a linear proof length, a prover can simply send the entire input as the proof. The verifier can read the proof, and verify it is equal to the real input by drawing a constant number of samples from it. Hence, Equality Testing has a SMAP with linear proof length and constant sample complexity (i.e., $s = O(1/\epsilon)$). In addition, Equality Testing has a sample-based tester with sample complexity of $O(\sqrt{n})$ (without a proof). Theorem 5 shows that equality testing has no SMAP that is "non-trivial". A SMAP for this problem must have linear proof length, or sample complexity as large as the sample complexity of a tester that is not aided by a proof (up to a logarithmic factor).

The proof of Theorem 5 is by a reduction to a communication complexity protocol. We show that a SMAP for equality with short proof length and low sample complexity, implies a communication protocol for equality with parameters that are impossible to achieve.

Finally, our last result is regarding the limitation of *public-coins* SIPPs. We consider again the Equality Testing Problem, and show that public-coins SIPPs for this problem are very limited.

Theorem 6. *(informal; see Theorem 4.7) If a public-coins SIPP for EQU with proximity parameter $\epsilon = 0.1$ has communication complexity c and sample complexity $s = o(\sqrt{n})$, then $c \cdot s = \Omega(n)$.*

We note that the lower bound of Theorem 6 is weaker than the one of Theorem 5, but it is tight (up to a logarithmic factor). For every $s \leq \sqrt{n}$ and $\epsilon > 0$ there exists a 1-round public-coins SIPP for equality testing with proximity parameter ϵ, sample complexity s and communication complexity $c = \widetilde{O}(\frac{n}{\epsilon \cdot s})$ (see Section 3.3). Also note that Theorem 6 implies an exponential separation between the power of private-coins SIPPs and public-coins SIPPs. This result stands in contrast to the model of query-based IPPs, in which [RVW13] showed that the expressive power of private-coin IPPS is essentially equivalent to that of public-coin IPPs.

1.3 Technical Overview

Main Result: Our main technical contribution is the reduction from SIPPs to IPPs. We next give a high-level sketch of this reduction, which we describe formally and prove in Section 5.

Consider a language L that has a sublinear IPP (i.e., an IPP with query complexity, communication complexity and verifier running time that are all sublinear). Also assume the IPP uses only public coins, and so w.l.o.g the verifier queries the input only after its interaction with the prover ends.[3] Furthermore, assume the indices of the queries depend solely on the public coins. That is, they do not depend on the messages sent from the prover, nor on the input itself. Note that this assumption holds for many useful sublinear IPPs, see e.g. [RVW13, RRR16, RR20].

Our goal is to show that L has a sublinear *sample-based* IPP. Consider the following strategy: The verifier of the SIPP privately tosses the coins that a verifier of the query-based IPP would have tossed publicly. It then computes the indices $Q \subseteq [n]$ it needs to query for these coin tosses, before it starts the interaction with the prover. But now, the verifier faces an issue. These queries have structure; they are dependent on each other. The SIPP verifier can draw a set of i.i.d samples from the input, but it cannot query the indices Q.

The first key idea of the reduction is to execute a (query-based) IPP on a *permuted* version of the input. That is, rearrange the bits of the input string x according some permutation, such that the indices of the queries the IPP verifier needs to make to the permuted input are mapped to the indices of samples the SIPP verifier drew. I.e, the SIPP verifier draws a set of samples, let S be their indices. The verifier then chooses a permutation π mapping Q to S, sends π to the prover, and executes the

[3]In a public coins protocol the coin tosses are the only messages the verifier sends to the prover. Hence, the verifier's messages do not depend on its queries, and we can assume the verifier queries the input only after the interaction is over.

query-based IPP on the input $y = \pi^{-1}(x)$ with respect to the claim $\pi(y) \in L$[4] where in the IPP the verifier uses the coins chosen above (leading to queries Q). The completeness of this protocol is immediate. But what about its soundness?

The issue is that sending the permutation to the prover might *reveal* Q. The prover can use the knowledge on Q to deduce what coin tosses the verifier is going to use for the execution of the query-based IPP. This is a big problem, as the soundness of an interactive proof relies heavily on the prover not knowing the verifier's coins in advance.

A possible solution is to use a *random* permutation. We show that, if the indices in S are distributed uniformly and i.i.d, then a random permutation that maps Q to S does not leak information regarding Q. One can show that this implies that the soundness of the IPP is maintained. An issue with this solution is that the *representation* of a random permutation is large: it requires $\Theta(n \log n)$ bits. As the verifier needs to send the prover a representation of the permutation, this results in a protocol with super-linear communication complexity.

Instead, use can use a family of *k-wise independent* permutations, where k is the query complexity of the IPP (i.e., the size of Q). A permutation sampled from such a family is indistinguishable from a permutation chosen at random from the set of all permutations, for any process that receives the value of either of the permutations at any k points of its choice. The verifier can choose a (random) permutation from the family, that maps Q to S. We show that sending this permutation to the prover does not reveal Q, as long as it represented in a canonical way. In addition, such a permutation can be represented by a small number of bits. Known construction of approximate k-wise independent permutations can be represented by only $\widetilde{O}(k)$ bits, where again k is the query complexity of the original IPPs, and hence sublinear.[5]

Say that we have such a family of permutations. How can we choose a random permutation from it that maps the indices Q to the indices S? A possible process is for the verifier to draw permutations again and again, until it is "lucky" and finds a suitable permutation. This process may take exponential time, which results in a verifier with runtime that is very much not sublinear. Unfortunately, we do not know of a family that allows for performing such a sampling process in significantly more efficient manner.

Thus, rather than k-wise permutations, we use k-wise independent *hash functions*. This solves the issue of efficiently sampling a function that maps the indices Q to the indices S.[6] But what about the other indices? In hash functions, unlike permutations, there are collisions. A function might map two indices to the same one, or have an index in the range without a source. Hence, it might be impossible to "recover" the original input from its transformed version.

Our solution to this uses two ideas. First, observe that a k-wise independent hash function indeed can have collisions, but not too many. With high probability, at least a constant fraction of the indices are mapped one-to-one, and hence the bits at these "good" locations can be recovered. For a single function we can ignore all the other indices, and consider only the "good" ones.

Next, rather than a single one, we use *many* hash functions. We prove that using $m = O(\log n)$ functions ensures that *every* bit can be recovered by at least a constant fraction of the functions, with high probability. As its first message, the verifier sends to the prover all of the m hash functions. See Figure 4 for the exact construction. The protocol is well-defined, and it is complete (since the honest prover can recover the entire input).

[4]Note that in order to execute an SIPP for one language, we need to perform an IPP for another language. This is why the actual reduction is on a family of languages, that should be closed under the appropriate operation.

[5]For our purposes it is enough to consider a relaxed notion of *almost* k-wise independence, where the advantage of a distinguisher is limited by some δ. For efficient constructions of such families see the work of Kaplan, Naor and Reingold [KNR06].

[6]Namely, we use polynomials of degree at most $k-1$ as the hash functions. The verifier can find a suitable function by performing polynomial interpolation, which can be done in sublinear time by using the Fast Fourier Transform.

We turn back to soundness. One concern (that was already discussed), is that the prover might learn something about the coins that are going to be used for the IPP, from the functions sent by the verifier. We prove in Claim 5.15 that this is not the case. From the prover's perspective, the coins used for the IPP are totally random, even conditioned on the hash functions it gets from the verifier. Another concern is that during the transformation process we might damage the distance of the tested input from the language. In Claim 5.16 we prove that this transformation is *distance preserving*, and can only decrease the distance by a constant factor.

This concludes the high-level description of the reduction. For the full details (including how we untangle some of the circular definitions that hide in the description above) see Section 5.

1-round SIPPs: In Theorem 3 we show an exponential separation between the power of sample-based testers and the power of 1-round SIPPs. In Claim 3.4 we prove that every sample-based tester for the Equality Testing problem requires $\Omega(\sqrt{n})$ samples. The proof is based on the fact that with high probability, a sample-based tester that draws $o(\sqrt{n})$ samples from an input $z = (x, y)$ does not see both x_j and y_j for any value of j (a "collision"). This implies the tester cannot distinguish between accepting inputs, and uniformly random inputs (which it must reject with high probability).

In Claim 3.5 we present a 1-round SIPPs for verifying equality. At a high level, the protocol is as follows. The verifier draws a sample (i, z_i) from the input $z = (x, y) \in \{0,1\}^{2n}$. If $i \leq n$ it sends to the prover $j = i$. Otherwise, $i > n$, and it sends $j = i - n$. The prover sends to the verifier a bit b, and the verifier accepts iff $z_i = b$.

In the completeness case, when $x = y$, the honest prover sends back to the verifier the value of $x_j = y_j$ and the verifier accepts. In the soundness case, when x is ϵ-far from y, with probability ϵ $x_j \neq y_j$. In this case, the best a prover can do is to send a random bit as b, and the verifier rejects with probability 1/2. The soundness of the protocol can be amplified to 2/3 by $O(1/\epsilon)$ parallel repetition of this basic protocol.

More generally, we show that every property with a non-adaptive (query-based, fair) tester has a 1-round SIPP. See Section 3.2 for details.

Lower Bounds: In Theorem 5 we show a lower bound on the power of non-interactive proofs of proximity (SMAPs) for Equality Testing. The proof is by a reduction to a communication complexity problem in the consecutive messages (CM) model setting [NR05]. In such a protocol, Alice and Bob get inputs x and y (respectively) from an adversary, and their shared goal is to make Carol compute a function f of x and y, by sending her short messages. In addition, after Alice gets her input (and before Bob gets his), she sends a public message that all parties can see (adversary included).

Naor and Rothblum [NR05] showed a lower bound on the length of the (private) messages required in a CM protocol for Equality, when the public message length is sublinear. We show that a SMAP for Equality with low sample complexity and sublinear proof length implies a CM protocol for Equality with short private messages and sublinear public message (that are impossible to achieve, by the above lower bound). The key observation for the reduction is that the samples of a verifier in a SMAP are drawn i.i.d. Hence, Carol can "ask" Alice to sample the x part of the input, Bob to sample y, and "simulate" the execution of the SMAP verifier by using these samples, and using the public message as the proof string.

In Theorem 6 we show a lower bound on the power of *public-coins* SIPPs for Equailty Testing. We prove this theorem by using tools from the area of Information Theory. At a high level, we show that a verifier cannot distinguish between an input it should accept (sampled uniformly from the set of all accepting inputs) and an input sampled uniformly from the set of all possible inputs (which it should reject with high probability). We consider the view of the verifier in an execution of the protocol with a (random) input it should accept. We show that if the communication complexity of the protocol is low, then the input (conditioned on the transcript of the protocol) still has a lot of entropy. We then use Shearer's Lemma to show that since the samples of the verifier are distributed uniformly and i.i.d, the entropy of its view (i.e., the part of the input the verifier sees in the samples) is also large. This implies, by a corollary of Pinsker's inequality, that the view of the verifier is close to uniform (even with the help of the prover), and hence it cannot distinguish the accepting input from a uniformly random input that it should reject.

1.4 Related Work

In the previous sections we discussed query-based and sample-based property testers, as well as query-based interactive and non-interactive proofs of proximity. We elaborate on these models, as well as the literature on testing and verifying properties of distributions, and the relationship to our work.

Query-access vs. Sample-based Access in Property Testing: In the standard definition of property testers, the tester has the ability to make arbitrary queries to the input. This definition was presented by Rubinfeld and Sudan [RS96], and it is the main definition studied by Goldreich, Goldwasser and Ron [GGR98]. Sample-based testers were defined by Goldreich, Goldwasser and Ron [GGR98], and a comprehensive systematic study of their power and their limitations was initiated by Goldreich and Ron [GR16]. The sample-based access model is syntactically weaker than the query-access model. A randomized algorithm with query access can simulate a sample-based algorithm by simply choosing indices uniformly and i.i.d, query these indices, and "feed" the sample-based algorithm with these queries.

Furthermore, sample-based testers are *strictly* weaker than query-based testers. One example is Linearity Testing, where the input is a Boolean function $f : \{0,1\}^n \to \{0,1\}$, and one needs to decide if f is a linear function or ϵ-far from any linear function. In their seminal work, Blum, Luby and Rubinfeld [BLR90] showed that there exists a query-based tester for this problem, with query complexity $O(1/\epsilon)$ (independent of the input length). On the other hand, [GR16] showed that any *sample-based* tester for linearity requires $s = \Omega(1/\epsilon + n)$ samples (provided $\epsilon \geq 1/2^n$).

A second example of such a separation is in testing Bipartiteness in the Dense Graph model (we do not define formally the model or this property in this work). There exists a query based tester for this property with query complexity of $O(1/\epsilon^2)$ ([Gol17], section 8.3.1), whereas [GR16] showed that any sample-based tester for it requires $s = \Omega(\sqrt{n}/\epsilon)$ samples (provided $\epsilon > 1/\sqrt{n}$).

Another example is of Monotonicity Testing. The input is again a Boolean function $f : \{0,1\}^n \to \{0,1\}$. Golderich et al. [GGL$^+$00] showed a query-based tester for this property with query complexity $O(n/\epsilon)$. The authors also showed (Theorem 5 in [GGL$^+$00]) that any sample-based tester for this property requires $\Omega(\sqrt{2^n/\epsilon})$ samples (provided $\epsilon = O(n^{-3/2})$).

Note that all of the 3 properties above have query-based testers with low query complexity, and those testers are fair and non-adaptive. Hence, we can directly apply Theorem 4 to deduce that each one of this properties has a 1-round SIPP with low sample and communication complexities.[7]

Distribution Testing and Verification: In *distribution testing*, a tester is given samples of an unknown distribution over a finite domain $\Sigma = [N]$. It is asked to determine whether the distribution has some property or is far from any distribution that has this property (i.e., the variation distance is large).[8] The main complexity measure of interest is the tester's sample complexity, as a function of the size of the distribution's domain. See Canonne [Can15] for a survey on this area.

[7]In fact, Linearity Testing and Monotonicity Testing have POTs, so we can apply Theorem 3.2 to get SIPPs for these properties with better communication complexity.

[8]The variation distance between D and D' is $\frac{1}{2} \cdot \sum_v |D(v) - D'(v)|$ where $D(v)$ (resp., $D'(v)$) denotes the probability that an element distributed according to D (resp., D') equals v.

Chiesa and Gur [CG18] initiated the study of proofs of proximity for distribution testing, and explored both interactive and non-interactive proof systems. Note that the prover in such a proof system has full knowledge of the distribution, even if it cannot be represented in a finite manner. They showed that such proof systems can be powerful. For example, every property of distributions has a non-interactive proof system with sample complexity $O(\sqrt{N})$ and proof length $O(N \log N)$.

In the sample-based setting we can also consider a large input alphabet Σ. Such an input implicitly defines a distribution over Σ, where the probability of every $\sigma \in \Sigma$ is in proportion to the number of its appearances in the input string. However, a sample-based tester gets more information than just samples from this distribution; It gets the *index* of each sample. This allows for expressing much richer properties in the sample-based settings than in distribution testing. Other properties, that are invariant under permutations of the indices, are called *symmetric* properties.[9] Goldreich and Ron [GR16] showed that without a prover, testing such properties is equivalent to distribution testing. That is, the advantage of seeing the indices of the samples does not increase the power of the tester significantly.

This is *not* the case when the tester is aided by a prover. Namely, sample-based interactive proofs (of symmetric properties) are *stronger* than interactive proofs for distributions. With the aid of a prover, a verifier can take use of the indices it sees. For example, consider the problem where the input is a string $x \in \Sigma^n$, and one needs to test if the distribution of x_i, for a uniformly random choice of i, is itself uniform over Σ. On the one hand, [CG18] proved that in the distribution testing setting, when the verifier does not see the indices of the samples, any proof system for this problem requires $\Omega(\sqrt{N})$ samples. On the other hand, a result of [EKR04] implies a SIPP with *constant* sample complexity (and logarithmic communication complexity) for verifying this property.[10]

We note that one result of [CG18] is similar in spirit to a result in this work. Theorem 1.4 in [CG18] shows an exponential separation between the power of distribution testers and the power of 1-round interactive proofs of proximity for distributions. In Theorem 3 in this work we show an exponential separation between the power of sample-based testers and the power of 1-round SIPPs.

Also note that [CG18] showed a strong separation between the power of interactive proofs and the power of public-coins interactive proofs for distributions. Namely, [CG18] shows that if a distributions property has a public-coins interactive proof system with communication complexity c and sample complexity s, then this property has a *tester* with sample complexity $O(c \cdot s)$.

In Theorem 6 we also show a separation between the power of private-coins protocols and the power of public-coins protocols, in the model of sample-based IPPs. The lower bound in our result is of different nature than the one of [CG18]. First, we give a lower bound for a specific problem (equality testing), wheres [CG18] showed a general separation result. Second, equality testing has a tester with sample complexity $O(\sqrt{n})$. We give a lower bound which is *not* of the form $c \cdot s = \Omega(\sqrt{n})$. In Theorem 6 we prove that if a public-coins SIPPs for equality has sample complexity $s = o(\sqrt{n})$ and communication complexity c, then $c \cdot s = \Omega(n)$. We have that (assuming $s = o(\sqrt{n})$), the product $c \cdot s$ is (much) larger than the sample complexity of testing equality (without a prover). Also note that the proof uses different tools than the ones used by [CG18].

Machine Learning: Goldwasser et al. [GRSY21] studied verification of machine learning. In this setting, a verifier has sampling access to an unknown distribution over labeled examples, and wants to verify that a given hypothesis, which has error ϵ, is the best one in some fixed hypothesis class. From a property testing perspective, the prover wants to convince the verifier that the input distribution D has the property "every hypothesis in the class has error larger than ϵ on D". The verifier should reject

[9]Formally, a property $L \subseteq \Sigma^n$ is said to be symmetric if for any permutation $\pi : [n] \to [n]$, if it holds that $x \in L$ if and only if $\pi(x) \in L$, where $\pi(x) \in \Sigma^n$ is defined by $[\pi(x)]_i = x_{\pi(i)}$ for every $i \in [n]$.

[10]The result of [EKR04] is stated for verifying that the input is a "permutation", which is equivalent to the uniformity testing problem defined above.

distributions that are far from this property. The work of [GRSY21] shows that for some hypothesis classes, verification can be easier than learning, where the measure of complexity is sample complexity.

An important case studied in machine learning is when the distribution of the samples (without the labels) is uniform. This scenario is similar to the one we study in this work, in the sense that the view of the verifier is the same: In both cases, it receives labeled samples from a large "world", where the distribution over the domain set / indices is uniform. Even for the case of a uniform distribution, one difference between [GRSY21] and this work, is that they focus on verification of specific types of properties (namely, machine learning properties as above), with sample complexity smaller than the *VC dimension* of the hypothesis class in question. We, on the other hand, study verification of general properties, and in some of our results the sample complexity is as large as the square root of the domain size (which is still sublinear).

Related Work on Interactive Proofs: The area of interactive proofs has a long and rich history. Interactive proofs were introduced by Goldwasser, Micali and Rackoff [GMR89], and independently by Babai and Moran [BM88]. The surprising expressive power of $\mathcal{IP}$ was determined in a sequence of works by Lund, Fortnow, Karloff and Nissan [LFKN92] and Shamir [Sha92], who showed that every language in $\mathcal{PSPACE}$ has an interactive proof.

The notion of interactive proofs was considered under various restrictions on the verifier. The power of finite state verifiers was studied by Dwork and Stockmeyer [DS92a, DS92b]. Condon and Ladner [CL88], Condon and Lipton [CL89], and Condon [Con91] studied space (and time) bounded verifiers. Goldwasser et al. [GGH$^+$07] considered the parallel running time of the verifier, and showed results for $\mathcal{NC}^0$ verifiers. Goldwasser, Kalai and Rothblum [GKR08] explored the power of doubly-efficient interactive proofs, where the honest prover runs in polynomial-time, and the verifier runs in *linear*-time. A main result of [GKR08] is that every language in logspace-uniform $\mathcal{NC}$ has such a proof system, where the protocol has $\text{poly}(\log n)$ rounds.

Interactive proofs of *proximity* were first considered by Ergun, Kumar and Rubinfeld [EKR04] (where they were called approximate interactive proofs). Rothblum, Vadhan and Wigderson [RVW13] conducted a systematic study of the power of such proofs, which they named Interactive Proofs of Proximity (IPPs). Interactive and non-interactive proofs of proximity have drawn considerable attention recently [FGL14, KR15, GGR15, RRR16, GR17, BRV18, CG18, GLR18, GR18, CG18, RR19, RR20, GRSY21]. We next compare our work to some of these results.

Comparison With IPPs for $\mathcal{NC}$ languages: One of the main result of [RVW13] is that every language in logspace-uniform $\mathcal{NC}$ has a (query-based) IPP with $O(n^{1/2+o(1)})$ query and communication complexities (and similar verifier running time), and $\text{polylog}(n)$ communication rounds. This result was later improved by Rothblum and Rothblum [RR20], which showed that these languages have IPPs with $\widetilde{O}(\sqrt{n})$ query complexity, communication complexity and verifier running time. The protocols in both results use the GKR protocol [GKR08] as a sub-routine.

One of our results (Theorem 1) builds upon the result of [RVW13, RR20], and extends it to the case where the verifier is *extremely* restricted - it only sees sampled labels from the input, and cannot perform queries. That limitation on the verifier power comes with (almost) no degradation in the protocol complexities, but it does have some costs. As discussed above, one of the downsides of our construction is that the protocol of Theorem 1 uses *private-coins*, whereas the protocol of [RVW13, RR20] is a *public-coin* protocol. An open question is whether this limitation in *inherent* to sample-based interactive protocols, or just an artifact of the specific construction of this work. Note that for query-based IPPs [RVW13] showed that the power of public-coins protocols is almost equivalent to the power of private-coins protocols.

A second difference between the protocols, is that the protocol of [RVW13, RR20] allows a "full" trade-off between the query complexity and the communication complexity. It allows the protocol to have any query complexity q and communication complexity c such that $q \cdot c = \widetilde{\Theta}(n)$. In our construction there is a limited trade-off; the communication complexity must be at least as large as the sample complexity.

Note that trade-off of the communication and query complexities in the result of [RR20] is optimal, up to poly-logarithmic factors. Kalai and Rothblum [KR15] showed that, under a reasonable cryptographic assumption, there exists a language L in $\mathcal{NC}^1$ such that the product of the query and communication complexities of any IPP for L cannot be sublinear. As SIPPs are a restriction of IPPs, this lower bounds also holds also for the sample-based setting. Namely, the result of Theorem 1 is tight, up to poly-logarithmic factors.

Comparison With IPPs for Languages Computable in Bounded Space: Reingold, Rothblum and Rothblum [RRR16] studied the power of interactive proof systems with *constant* number of rounds. Their main result is that any language computable in bounded polynomial space and polynomial time has a public-coin interactive proof system with constant number of rounds, low communication complexity and verifier running time, and efficient honest prover. By replacing the [GKR08] step in [RVW13] with their result, they show that every such language has a public-coin, constant-round *IPP*, with query and communication complexities, as well as verifier's running time, roughly $\sqrt{n}$. In addition, the running time of the honest prover in the IPP is $\text{poly}(n)$ (Theorem 3 in [RRR16]. See Theorem 5.27 for the exact formulation of the theorem). In this work, we apply our new SIPP to IPP reduction on this result of [RRR16], to show that every such language has *sample-based* IPP with similar parameters (Theorem 2). As above, one difference between our SIPP and the IPP of [RRR16] is that the SIPP uses private coins whereas the IPP is public-coin. Again, it as an open question whether this limitation in inherent to SIPPs or not.

Non-interactive Proofs of Proximity: Gur and Rothblum [GR18] initiated a study of non-interactive proofs of proximity. Such proof systems can be viewed as the property testing analogue of an $\mathcal{NP}$ proof system (whereas IPPs are the property testing analogue of $\mathcal{IP}$). In contrast to polynomial-time algorithms, sublinear-time algorithms inherently rely on randomization. Since an $\mathcal{NP}$ proof system with a randomized verifier is known as a Merlin-Arthur (MA) proof system, [GR18] named sublinear non-interactive proof systems Merlin-Arthur proofs of Proximity (MAPs). In this work we study non-interactive proofs of proximity, with sample-based verifiers. Following the naming of [GR18], we name such proof systems *Sample-based Merlin-Arthur proofs of Proximity* (SMAPs).

Gur and Rothblum [GR18] show that MAPs can be exponentially stronger than property testers, but exponentially weaker than IPPs. In this work, the focus of the study of non-interactive proofs is on lower bounds. Namely, we prove that SMAPs can be *extremely* limited in their power (see Theorem 5). We note that Theorem 5 of [GR18] implies a generic lower bound on the power of SMAPs. For the problem of equality testing, Theorem 5 (of this work) gives an improved lower bound, in terms of the possible trade-offs between sample complexity and proof length. See Section 4.1 for various results from [GR18] that are relevant to the setting of this work.

Fischer, Goldhirsh and Lachish [FGL14] introduced the notion of *partial testing*, which is closely related to MAPs. A property L is a said to be L'-partially testable, for $L' \subseteq L$, if inputs in L' can be distinguished from inputs that are far from L by a tester that makes only few queries. This notion can be naturally extended to the sample-based setting.

1.5 Organization

In Section 2 we cover some basic notations, and make formal definitions regarding property testers, IPPs and SIPPs. We introduce in this section the new definitions of SIPPs (Definition 2.11) and

SMAPs (Definition 2.12). In Section 3 we examine the Equality Testing Problem, show an SIPP for it, and prove that it demonstrates an exponential separation between the power of sample-based testers and SIPPs. We then extend the SIPP for equality to every language that has a non-adaptive tester with constant query complexity. In Section 4 we continue to consider the Equality Testing problem, and show a lower bound for SMAPs for it. In Section 5 we show a reduction from SIPPs to IPPs. We use this reduction to prove that every language in log-space uniform $\mathcal{NC}$ has a sub-linear SIPP, and that every language computable in polynomial-time and bounded-polynomial space also has a sub-linear SIPP.

2 Definitions and Preliminaries

We begin with some basic notations:

- Denote by $[n]$ the set $[n] = \{1, 2, \ldots, n\}$.
- Denote by $[n]^k$ the set of all k-tuples of elements of $[n]$.
- Denote by $[n]^{\{k\}}$ the set of all k-tuples of *distinct* elements of $[n]$.
- For a set Ω, denote by U_Ω the uniform distribution on the elements of Ω.
- For a set A, denote by $a \sim A$ a random variable a that is chosen uniformly at random from the set A.
- For a set A, denote by $a_1, a_2, \ldots, a_k \sim A$ k random variables $a_1, \ldots, a_k$ that are chosen uniformly at random and i.i.d from the set A.
- For a set $A \subseteq [n]$ and a function $f : [n] \to [n]$, let $f(A) = \{f(x) \mid x \in A\}$.
- For $B = (x_1, \ldots, x_k) \in [n]^k$ and a function $f : [n] \to [n]$, let $f(B) = (f(x_1), \ldots, f(x_k))$, and note that if $B \in [n]^{\{k\}}$ and $f|_B$ is injective then $f(B) \in [n]^{\{k\}}$.
- For a set $A \subseteq [n]$ and a function $f : [n] \to [n]$, let $f^{-1}(A) = \{x \mid f(x) \in A\}$.
- For a function $f : [n] \to [n]$ and $i \in [n]$, let $f^{-1}(i) = f^{-1}(\{i\})$.

A *language* L is a set of strings of arbitrary lengths, $L \subseteq \{0,1\}^*$. A *pair language* L is a set of pairs of strings, $L \subseteq \{0,1\}^* \times \{0,1\}^*$. Note that every pair language L can be interpreted as a standard language. We identify a language $L \subseteq \{0,1\}^*$ (which might be a pair language) with the boolean function $L : \{0,1\}^* \to \{0,1\}$ that outputs 1 iff the input is in the set L. For a string $x \in \{0,1\}^n$ and for $i \in [n]$ we denote the i-th element of x by x_i.

We define the *relative* distance between strings over some alphabet Σ as the fraction of locations on which they differ. That is, for $u, v \in \Sigma^\ell$ we denote $\Delta(u, v) = |\{i \mid u_i \neq v_i\}|/l$. We say that u is *δ-close* to v (resp., *δ-far* from v) if $\Delta(u, v) \leq \delta$ (resp., $\Delta(u, v) > \delta$). The relative distance of a string to a set of strings is defined in the natural manner; that is $\Delta(u, S) = \min_{v \in S} \Delta(u, v)$.

2.1 Entropy and Information Theory

Definition 2.1. *For a random variable X over finite support S, we use $H(X)$ to denote the* Shannon Entropy *of X. That is, $H(X) = -\sum_{x \in S} \Pr[X = x] \cdot \log(\Pr[X = x])$.*

Definition 2.2. *For a random variable X over finite support S, we use $H_\infty(X)$ to denote the* min-entropy *of X. That is, $H_\infty(X) = -\log(\max_{x \in S} \Pr[X = x])$.*

Definition 2.3. *For random variables X, Y over finite support S, we use $H(X \mid Y = y)$ to denote the* conditional entropy *of X given $Y = y$. That is:*

$$H(X \mid Y = y) = -\sum_{x \in S} \Pr[X = x \mid Y = y] \cdot \log(\Pr[X = x \mid Y = y])$$

We use $H(X \mid Y)$ to denote the conditional entropy *of X given Y. That is:*

$$H(X \mid Y) = -\sum_{y \in S} \Pr[Y = y] \cdot H(X \mid Y = y)$$

Definition 2.4. *For random variables X, Y over finite support S, we use $\widetilde{H}_\infty(X \mid Y)$ to denote the* average min-entropy *of X given Y. That is:*

$$\widetilde{H}_\infty(X \mid Y) = -\log(\mathbb{E}_{y \leftarrow Y}[\max_{x \in S} \Pr[X = x \mid Y = y]])$$

Definition 2.5. *For probability distributions P and Q over finite support S, the* Kullback-Leibler divergence *from P to Q is:*

$$D_{KL}(P||Q) = \sum_{x \in S} P(x) \cdot \log(\frac{P(x)}{Q(x)})$$

2.2 Property Testing, IPPs and SIPPs

We say that an algorithm $\mathcal{A}$ has *oracle access* (or query access) to a string $x \in \{0,1\}^n$ when $\mathcal{A}$ is an oracle Turing Machine, where the oracle is for the function $f : [n] \to \{0,1\}$ that is defined by $f(i) = x_i$ for every $i \in [n]$. We say that x is the *implicit* input of $\mathcal{A}$. Each time $\mathcal{A}$ uses this oracle for some $i \in [n]$ we say that $\mathcal{A}$ *queries* the input x at index i.

The algorithm $\mathcal{A}$ might also have *explicit* input. We denote the output of $\mathcal{A}$ with oracle access to $x \in \{0,1\}^n$ and with explicit input z by $\mathcal{A}^x(z) \in \{0,1\}$. We shall associate the output 1 (resp., 0) with the decision to accept (resp., reject) the input. When $\mathcal{A}$ is a probabilistic algorithm (as it usually is), $\mathcal{A}^x(z)$ is a Boolean-valued random variable.

For a string $x \in \{0,1\}^n$, define the distribution D_x to be the uniform distribution over the set $\{(i, x_i)\}_{i=1}^n$. We say that an algorithm $\mathcal{A}$ has *sample-based access* to a string $x \in \{0,1\}^n$ when $\mathcal{A}$ is a Turing Machine that can draw samples from D_x. That is, $\mathcal{A}$ is an algorithm that can draw samples of the form (i, x_i), where for each sample i is distributed uniformly and i.i.d from the set of all possible indices $[n]$. We denote the output of $\mathcal{A}$ with sample-based access to $x \in \{0,1\}^n$ and explicit input z by $\mathcal{A}^{\sim x}(z)$. $\mathcal{A}^{\sim x}(z)$ is a Boolean-valued random variable, and its probability is over both the coins of $\mathcal{A}$ and over the samples from D_x.

We now give the definitions of *proximity testers* and of *Proximity Oblivious Testers* (POTs). We also define a standard variation of testers, where the tester is limited to be *sample-based*. All of these definitions are based on definitions from Goldreich's book [Gol17]. In this work we focus on one-sided error testers (that is, testers with full completeness). For the definition of two-sided error testers (which is more standard) and other variations see [Gol17].

Definition 2.6. *(Proximity Tester) Let L be a language. A (query-based)* tester *for L is a probabilistic algorithm, denoted T, that, on (explicit) input parameters n and ϵ and oracle access to (implicit) input $x \in \{0,1\}^n$, outputs a binary verdict that satisfies the following two conditions:*

- ***Completeness:*** *For every $n \in \mathbb{N}$ and $\epsilon > 0$, and for every $x \in L$, it holds that:*

$$\Pr[T^x(n, \epsilon) = 1] \geq 2/3.$$

- ***Soundness:*** *For every $n \in \mathbb{N}$ and $\epsilon > 0$, and for every x that is ϵ-far from L, it holds that:*

$$\Pr[T^x(n, \epsilon) = 1] \leq 1/3.$$

where the probabilities are over the coins of T.

We say that the tester has query complexity $q = q(n, \epsilon)$ if, on input n, ϵ, and oracle access to any $x \in \{0,1\}^n$, the tester makes at most $q(n, \epsilon)$ queries. We say that a tester is *non-adaptive* if it determines all its queries based on its explicit input and internal coin tosses, independently of the specific x to which it is given oracle access. In contrast, an *adaptive* tester may determine its $(i+1)$-th query based on the answers it has received to the prior i queries. If the query complexity of a tester does not depend on n, we say that the tester has *constant* query complexity. We say that a tester with query complexity q is *fair* if, when given oracle access to an input $x \in \{0,1\}^n$, each of its (possibly adaptive) q queries is uniformly distributed in $[n]$.

If the completeness condition holds with probability 1 (i.e, $T^x(n, \epsilon) = 1$ for every $x \in L$) we say that the tester has *perfect completeness.* In this work, we assume a tester always has perfect completeness, unless stated otherwise.

Definition 2.7. *(Proximity Oblivious Tester) Let L be a language, and let $\varrho : (0,1] \to (0,1]$ be a monotonically non-decreasing function. A* Proximity Oblivious Tester *(POT) T for L with detection probability ϱ is a tester as in Definition 2.6, with the following modifications:*

- *T does not get ϵ as an input*
- *The soundness requirement from Definition 2.6 is replaced with the following condition:* ***Soundness:*** *For every $n \in \mathbb{N}$,* every *$\epsilon > 0$, and every x that is ϵ-far from L, it holds that:*

$$\Pr[T^x(n) = 0] \geq \varrho(\epsilon).$$

where the probabilities are over the coins of T.

We emphasize the difference between a "standard" tester and a POT for some language L. A standard tester receives ϵ as input, and needs to reject every input x that is ϵ-far from L. A POT does *not* receive ϵ as input, but the probability with which it accepts $x \notin L$ is lower bounded by some function of the distance of x from L. The query complexity of a POT is defined similarly to the query complexity of a standard tester. The definition of a *fair* POT is similar to the definition of a fair tester.

Definition 2.8. *(Sample-Based Proximity Tester) Let L be a language. A* sample-based *tester for L is a probabilistic algorithm, denoted T, that, on input parameters n and ϵ and* sample-based access *to $x \in \{0,1\}^n$, outputs a binary verdict that satisfies the following two conditions:*

- ***Completeness:*** *For every $n \in \mathbb{N}$ and $\epsilon > 0$, and for every $x \in L$, it holds that:*

$$\Pr[T^{\sim x}(n, \epsilon) = 1] \geq 2/3.$$

- ***Soundness:*** *For every $n \in \mathbb{N}$ and $\epsilon > 0$, and for every x that is ϵ-far from L, it holds that:*

$$\Pr[T^{\sim x}(n, \epsilon) = 1] \leq 1/3.$$

where the probabilities are over the coins of T and the samples drawn by it.

We say that the tester has *sample* complexity $s = s(n, \epsilon)$ if, on input n, ϵ, and sample-based access to any $x \in \{0,1\}^n$, the tester makes at most $s(n, \epsilon)$ queries. If the sample complexity of a tester does not depend on n, we say that the tester has *constant* sample complexity. Perfect completeness of a sample-based testers is defined as for query-based testers.

We stress that the difference between standard (query-based) testers and sample-based testers is the type of access they have to their (implicit) input. A query-based tester can choose which indices it queries in the input, whereas a sample-based tester cannot. In practice, the indices that a query-based tester queries are often chosen at random, but they might depend on each other. That is, the index of the $(i+1)$-th query of the tester might depend on the indices of the prior i queries. The indices of the samples that a sample-based tester draws are always chosen *independently* at random. Note that this distinction is different than the distinction between adaptive and nonadaptive testers. The index of the $(i+1)$-th query of a (query-based) nonadaptive tester might depend on the *indices* of the prior i queries, but it does not depend on the value of the input x at these indices.

We now present the definition of an *interactive proof of proximity* (IPP). This notion generalizes the notion of a tester (which is now called a "verifier"), and allows it to interact with an all-powerful untrusted prover who has full explicit access to the input x. Such proof-systems were first introduced by Ergun, Kumar and Rubinfeld [EKR04] and were more recently further studied by Rothblum, Vadhan and Wigderson [RVW13]. The verifier $\mathcal{V}$ has *oracle access* to x, but the prover $\mathcal{P}$ has explicit (full) access to x, and both $\mathcal{V}, \mathcal{P}$ have explicit access to z. We denote the output of $\mathcal{V}$ after interacting with $\mathcal{P}$ by (the Boolean-valued random variable) $(\mathcal{P}(x), \mathcal{V}^x)(z)$. Similarly, when $\mathcal{V}$ has *sample-access* to x, $\mathcal{P}$ has explicit access to x, and both $\mathcal{V}, \mathcal{P}$ have explicit access to z, we denote the output of the $\mathcal{V}$ after interacting with $\mathcal{P}$ by (the Boolean-valued random variable) $(\mathcal{P}(x), \mathcal{V}^{\sim x})(z)$.

Definition 2.9. *(Interactive Proof of Proximity (IPP) [EKR04, RVW13]) Let $\epsilon = \epsilon(n) \in (0,1)$. A (query-based)* Interactive Proof of Proximity (IPP) *with proximity parameter ϵ for the pair language L is an interactive protocol with two parties: a (more powerful) prover $\mathcal{P}$ and a (more restricted) verifier $\mathcal{V}$. The verifier gets oracle access to $x \in \{0,1\}^n$ whereas the prover has full access to x. In addition, both parties get as (explicit) input $z \in \{0,1\}^m$ where $m = m(n)$.*

At the end of the interaction, the following two conditions are satisfied:

- ***Completeness:*** *For every pair $(z,x) \in L$ it holds that*

$$\Pr[(\mathcal{P}(x), \mathcal{V}^x)(z, |x|) = 1] \geq 2/3.$$

- ***Soundness:*** *For every $z \in \{0,1\}^m$ and x that is ϵ-far from the set $\{x' \mid (z,x') \in L\}$, and for every computationally unbounded (cheating) prover $\mathcal{P}^*$ it holds that*

$$\Pr[(\mathcal{P}^*(x), \mathcal{V}^x)(z, |x|) = 1] \leq 1/3.$$

where the probabilities are over the coins of $\mathcal{V}$.

An IPP for L is said to have query complexity $q = q(n, \epsilon)$ if, for every $(z,x) \in L$, the verifier $\mathcal{V}$ makes at most $q(|x|, \epsilon)$ queries to x when interacting with $\mathcal{P}$. The IPP is said to have communication complexity $c = c(n, \epsilon)$ if, for every pair $(z,x) \in L$, the communication between $\mathcal{V}$ and $\mathcal{P}$ consists of at most $c(|x|, \epsilon)$ bits. We say that the verifier time complexity of an IPP is $v = v(n, \epsilon)$ if, for every pair $(z,x) \in L$, the running time of $\mathcal{V}$ is bounded by $v(|x|, \epsilon)$. We say that the prover time complexity of an IPP is $t = t(n, \epsilon)$ if, for every pair $(z,x) \in L$, the running time of the *honest* prover is bounded by $t(|x|, \epsilon)$. The number of *rounds* of an IPP (which we also call *round complexity*) is the number of rounds of communication in the protocol, where in each round each party sends a single message.

We say that an IPP is *public coin* if the messages of $\mathcal{V}$ are restricted to be random bits, and $\mathcal{V}$ does not use any other random bits that are not contained in these messages. Perfect completeness of an IPP is defined similarly to the definition of perfect completeness of testers. Namely, we require that the honest prover can *always* convince the verifier to accept. In this work we assume that all IPPs have perfect completeness, unless explicitly stated otherwise.

We also consider the definition of a *non*-interactive proof of proximity, a notion of proof which was defined and first studied by Gur and Rothblum [GR16].

Definition 2.10. *A* Merlin-Arthur proof of Proximity *(MAP) is an IPP where the interaction is restricted to a single message from the prover to the verifier.*

In this case we call the prover's message a *proof*, and use the term *proof length* instead of communication complexity.

We next introduce a *new* definition, of sample-based interactive proofs of proximity. These proofs are the main objects that we study in this work.

Definition 2.11. *(Sample-Based Interactive Proof of Proximity) Let $\epsilon = \epsilon(n) \in (0,1)$. A* Sample-Based *Interactive Proof of Proximity (SIPP) with proximity parameter ϵ for the pair language L is an interactive protocol with two parties: a (more powerful) prover $\mathcal{P}$ and a (more restricted) verifier $\mathcal{V}$. The verifier gets* sample-based access $x \in \{0,1\}^n$ *whereas the prover has full access to x. In addition, both parties get as (explicit) input $z \in \{0,1\}^m$ where $m = m(n)$. At the end of the interaction, the following two conditions are satisfied:*

- ***Completeness:*** *For every pair $(z,x) \in L$ it holds that*

$$\Pr[(\mathcal{P}(x), \mathcal{V}^{\sim x})(z, |x|) = 1]] \geq 2/3.$$

- ***Soundness:*** *For every $z \in \{0,1\}^m$ and x that is ϵ-far from the set $\{x' \mid (z,x') \in L\}$, and for every computationally unbounded (cheating) prover $\mathcal{P}^*$ it holds that*

$$\Pr[(\mathcal{P}^*(x), \mathcal{V}^{\sim x})(z, |x|) = 1] \leq 1/3.$$

where the probabilities are over the coins of $\mathcal{V}$ and the samples drawn by it.

An SIPP for L is said to have *sample* complexity $s = s(n,\epsilon)$ if, for every $(z,x) \in L$, the verifier $\mathcal{V}$ draws at most $s(|x|,\epsilon)$ samples from x when interacting with $\mathcal{P}$.

The communication complexity, verifier time complexity, prover time complexity and round complexity for an SIPP are defined similarly to the complexities for an IPP. We say that the *total* complexity of an IPP is the sum of its communication and sample complexities $s(n,\epsilon) + c(n,\epsilon)$. If the sample complexity of an SIPP does not depend on n, we say that the SIPP has *constant* sample complexity. Similarly to IPPs, we say that an SIPP is *public coin* if the messages of $\mathcal{V}$ are restricted to be random bits, and $\mathcal{V}$ does not use any other random bits that are not contained in these messages. Perfect completeness of an SIPP is defined as it is defined for IPPs.

We also introduce the following new definition, of *non*-interactive sample-based proofs of proximity.

Definition 2.12. *A* Sample-based Merlin-Arthur proof of Proximity *(SMAP) is an SIPP where the interaction is restricted to a single message from the prover to the verifier.*

In this case, as in MAPs, we call the prover's message a *proof*, and use the term *proof length* instead of communication complexity.

2.3 Error Correcting Codes

Definition 2.13. *An* Error Correcting Code *is a function $ECC : \{0,1\}^k \to \{0,1\}^n$. The* codewords *of ECC are the elements in the range of ECC. The* distance *of the code ECC is the minimal relative distance between two codewords of ECC. The* rate *of the code ECC is $\frac{k}{n}$.*

Definition 2.14. *An* asymptotically good *code with rate t and (relative) distance ϵ is a family of codes $ECC_n : \{0,1\}^k \to \{0,1\}^n$ where each code ECC_n has distance at least ϵ and rate at least t.*

2.4 The Equality Testing Problem

We next present the classical *Equality Testing* problem. In this problem, the input is a binary string of length $2n$, $z \in \{0,1\}^{2n}$. This string is interpreted as $z = (x_1, x_2, ..., x_n, y_1, ...y_n)$, and the tester needs to accept if $x = y$ and reject if x is ϵ-far from y.

Formally, for $n \in \mathbb{N}$ we define the following property:

Definition 2.15. $EQU = \{(x,x) \mid x \in \{0,1\}^n\} \subseteq \{0,1\}^{2n}$.

3 1-round SIPPs

In this section we show an exponential separation between the power of sample-based testers and the power of 1-round sample-based interactive proofs of proximity.[11]

In particular, we show that every property that has a non-adaptive *query-based* tester with constant query complexity, has a 1-round SIPP with constant sample complexity and logarithmic communication complexity (even if the property does not have a sample-based tester with constant sample complexity).

We begin by studying the "Equality Testing" problem, and show for it:

1. A query-based tester with query complexity $q = O(1/\epsilon)$.
2. A lower bound for *sample-based* testers, of $s = \Omega(\sqrt{n})$.
3. A 1-round, private-coins, sample-based interactive proof of proximity, with communication complexity $c = O(\log n/\epsilon)$ and sample complexity $s = O(1/\epsilon)$.

Note that this already shows an exponential gap between the power of sample-based testers and the power of SIPPs.

We then extend the equality SIPP to any property that has a non-adaptive and *fair*[12] (query based) tester.

Theorem 3.1. *Let $\epsilon > 0$, and let $L \subseteq \{0,1\}^n$ be a language that has a non-adaptive query-based fair tester with proximity parameter ϵ and query complexity $q = q(n,\epsilon)$. Then L has a 1-round SIPP, with proximity parameter ϵ, communication complexity $O(q^2 \cdot \log n)$ and sample complexity $s = O(q)$.*

In the case when the query-based tester for L has constant query complexity, the resulted SIPP for L has indeed constant sample complexity and logarithmic communication complexity. For properties that have POTs (proximity oblivious testers) we give SIPPs with slightly better communication complexity:

Theorem 3.2. *Let $\varrho : (0,1] \to (0,1]$ be a monotonically non-increasing function, let $\epsilon > 0$, and let $L \subseteq \{0,1\}^n$ be a language that has a non-adaptive query-based fair POT, with query complexity q and detection probability ϱ. Then L has a 1-round SIPP, with proximity parameter ϵ, communication complexity $c = O(\frac{q^2}{\varrho(\epsilon)} \cdot \log n)$ and sample complexity $s = O(q/\varrho(\epsilon))$.*

Note that for any $\epsilon > 0$, any language with (one-sided error) POT with detection probability ϱ and query complexity q has a (standard) tester with proximity parameter ϵ and query complexity $q/\varrho(\epsilon)$ (which is achieved by simply executing the POT for $1/\varrho(\epsilon)$ times). Therefore, applying Theorem 3.1 on the (standard) tester that is induced by a POT yields a SIPP with communication complexity $c = O(\frac{q^2}{\varrho(\epsilon)^2} \cdot \log n)$, whereas applying Theorem 3.2 yields an SIPP with better communication complexity, $c = O(\frac{q^2}{\varrho(\epsilon)} \cdot \log n)$, and with the same sample complexity.

[11]For the separation we compare the sample complexity of the tester with the *total* complexity of the SIPP (communication complexity + sample complexity).

[12]Recall that a tester is *fair* if each of its queries is uniformly distributed in $[n]$.

3.1 Equality Testing

We begin by presenting a simple and well-known query-based tester for equality, that uses $O(1/\epsilon)$ queries:

Claim 3.3. *Let EQU be as in Definition 2.15, and let $\epsilon > 0$. There exists a (query-based) tester for EQU with proximity parameter ϵ and query complexity of $q = O(1/\epsilon)$.*

Proof. We present a POT for equality in Figure 1.

Figure 1 Query Based POT for Equality, T
Input: Implicit (query) access to $z = (x_1, x_2, ..., x_n, y_1, ...y_n) \in \{0,1\}^{2n}$, explicit access to ϵ
Output: Accept / Reject

1. Choose $i \sim [n]$.
2. Accept iff $x_i = y_i$.

If $x = y$ then T always accepts. It is also easy to see that if $\Delta(x, y) > \epsilon$, then T rejects with probability ϵ (since in this case, $\Pr_{i \sim [n]}[x_i \neq y_i] > \epsilon$). To get a tester with constant soundness (say, 2/3), repeat the execution of the tester T for $O(1/\epsilon)$ times, with independent randomness in each execution, and accept iff T accepted in all of the executions. □

Lower Bound for Sample-Based tester: The tester T described in Figure 1 relies on its ability to query the implicit input. It needs to query x and y at the same index i. This cannot be performed by a sample-based tester, as the indices of the samples it draws are distributed uniformly and independently over $[2n]$. In other words, the tester cannot "coordinate" the indices of the samples it draws from the "x side" of the input, and the indices of the samples from the "y side".

Indeed, any sample-based tester for equality must draw many more samples than the number of queries the query-based tester makes.

Claim 3.4. *Let EQU be as in Definition 2.15, and let $\epsilon > 0$. Any sample-based tester for EQU with proximity parameter ϵ must draw $s = \Omega(\sqrt{n})$ samples.*

Proof sketch. Consider the following two distributions over $\{0,1\}^{2n}$. D_{YES} is the uniform distribution over $\{(x, x) \mid x \in \{0,1\}^n\} \subseteq \{0,1\}^{2n}$, the inputs that tester for EQU must accept. D_{NO} is defined to be the uniform distribution over all of $\{0,1\}^{2n}$.

A tester for EQU must accept an input sampled from D_{YES}. It must reject with high probability an input sampled from D_{NO}, since such an input is very far from EQU (with high probability). We argue that a sample-based tester that draws $o(\sqrt{n})$ samples cannot distinguish between these two distributions. Assume that the tester does not see any "collision" (i.e, it did not sample both x_i and y_i for any value of i). Then its view of the input is the same for an input sampled from D_{YES} as the view it has of an input sampled from D_{NO}. With high probability, no such collision occur for $o(\sqrt{n})$ i.i.d samples, and therefore the tester cannot distinguish between these distributions. Hence, the tester cannot satisfy both completeness and soundness. □

SIPP for equality: We next present a sample-based interactive proof of proximity for equality.

Claim 3.5. *Let EQU be as in Definition 2.15, and let $\epsilon > 0$. There exists an 1-round SIPP for EQU with proximity parameter ϵ, sample complexity $s = O(1/\epsilon)$ and communication complexity $O(\log n/\epsilon)$.*

Proof. We describe the SIPP for equality in Figure 2.

Figure 2 SIPP for Equality Between a verifier $\mathcal{V}$ and a prover $\mathcal{P}$
Verifier Input: Sample access to $z = (x_1, x_2, ..., x_n, y_1, ...y_n) \in \{0,1\}^{2n}$, explicit access to ϵ.
Prover Input: Full (explicit) access to z.
Output: Accept / Reject

1. $\mathcal{V}$ draws (i, z_i), where $i \sim [2n]$.
2. If $1 \le i \le n$: $\mathcal{V}$ sends $j = i$ to $\mathcal{P}$.
3. If $n+1 \le i \le 2n$: $\mathcal{V}$ sends $j = i - n$ to $\mathcal{P}$.
4. Honest prover $\mathcal{P}$ receives $j \in [n]$ and sends $b = x_j = y_j$ to $\mathcal{V}$.
5. $\mathcal{V}$ receives b^* from $\mathcal{P}$ and accepts iff $b^* = z_i$.

Correctness: It is immediate to check that if $x = y$ then the honest prover who follows the protocol will always make $\mathcal{V}$ accept.

We now claim that in the case $\Delta(x, y) > \epsilon$, any (malicious) prover $\mathcal{P}^*$ can make $\mathcal{V}$ accept with probability at most $1 - \frac{\epsilon}{2}$. Indeed, in this case $\Pr[x_j \neq y_j] = \epsilon$, where the probability is over the choice of j. That is, with probability of at least ϵ, the verifier sampled (i, z_i) such that $x_j \neq y_j$ (where $j = i$ if $i \le n$ and $j = i - n$ otherwise).

The crucial point is that in this case, the prover, who receives only $j \in [n]$ from $\mathcal{V}$, *does not know* if $\mathcal{V}$ sampled (j, x_j) or $(j + n, y_j)$, as i is kept secret. Now, if $\mathcal{P}^*$ sends $b^* = x_j$ to $\mathcal{V}$ and $\mathcal{V}$ sampled $(i, z_i) = (j + n, y_j)$, $\mathcal{V}$ rejects, since $b^* = x_j \neq y_j = z_j$. Similarly, if $\mathcal{P}^*$ sends $b^* = y_j$ to $\mathcal{V}$ and $\mathcal{V}$ sampled (j, x_j), $\mathcal{V}$ also rejects. Recall that the verifier draws its samples uniformly at random, and hence the probability of each of the events $i = j$, $i = j + n$ conditioned on one of them happening is $1/2$. Therefore, no matter what is the message $b^* \in \{0,1\}$ that $\mathcal{P}^*$ sends to $\mathcal{V}$, with probability of $1/2$ $\mathcal{V}$ "catches" the prover and rejects. Now, since $\Pr[x_j \neq y_j] \ge \epsilon$, we get that in total the prover can only make $\mathcal{V}$ accept with probability at most $1 - \frac{\epsilon}{2}$.

To get a protocol with constant soundness (say, $2/3$), repeat the execution of the protocol described in Figure 2 for $O(1/\epsilon)$ times, *in parallel*. See Goldreich's book [Gol98] (Apdx. C.1) for a proof that the soundness error in parallel repetition of interactive proofs decreases exponentially with the number of repetitions.

Complexity Analysis: The SIPP has 1 round, and its sample complexity is $s = O(1/\epsilon)$. Its communication complexity is $c = O(\log n/\epsilon)$, since in each (parallel) execution the verifier sends a single element of $[n]$, the prover sends a single bit, and there are $O(1/\epsilon)$ iterations. □

3.2 SIPP for any property with non-adaptive tester

In this section we prove Theorem 3.1 and Theorem 3.2. We prove these theorems in a single proof, as the SIPPs in both cases are very similar.

Proof. (of Theorem 3.1 and Theorem 3.2) Let L be a property with a non-adaptive tester T, with query complexity $k = q(n, \epsilon)$, or a non-adaptive POT T with query complexity k and detection probability ϱ. Assume that T tosses during its execution $r(n)$ coins. Since the tester is non-adaptive, for every $1 \le j \le k$ there exists a function $q_j : \{0,1\}^{r(n)} \to [n]$ such that if the coin tosses of T were $\rho \in \{0,1\}^{r(n)}$, the j-th query of it is on index $q_j(\rho)$. We denote by $T(y_1, \ldots, y_k; \rho)$ the output of T on implicit input x, when its coin tosses were ρ and for each j, the query on index $q_j(\rho)$ returned the value $y_j \in \{0,1\}$. Under this terminology, the completeness requirement from T is that for any $\rho \in \{0,1\}^{r(n)}$ and any $x \in L$:

$$T(x_{q_1(\rho)}, \ldots, x_{q_k(\rho)}; \rho) = 1$$

If T is a standard tester with proximity parameter ϵ, the soundness requirement from it is that for any x that is ϵ-far from L:

$$\Pr_{\rho \sim \{0,1\}^r}[T(x_{q_1(\rho)}, \ldots, x_{q_k(\rho)}; \rho) = 1] \leq 1/3$$

If T is a POT with detection probability ϱ, the soundness requirement from it is that for any $\epsilon > 0$ and any x that is ϵ-far from L:

$$\Pr_{\rho \sim \{0,1\}^r}[T(x_{q_1(\rho)}, \ldots, x_{q_k(\rho)}; \rho) = 0] \geq \varrho(\epsilon)$$

Since the tester is fair, for every $j \in [k]$ there exists a procedure that is the "inverse" of the operation of the function q_j. Namely, there exists a (possibly random) procedure $R_j : [n] \to \{0,1\}^{r(n)}$ such that $q_j(R_j(i)) = i$ (for every possible output of $R_j(i)$), and for every $\rho \in \{0,1\}^{r(n)}$:

$$\Pr_{i \sim [n]}[R_j(i) = \rho] = \frac{1}{2^{r(n)}}$$

where the probability is also over the coins of R_j.

We next describe the basic SIPP for the statement $x \in L$. The basic SIPP is used both in the case where T is a standard tester, and in the case where T is a POT. The differences between the cases is in the analysis, and in the number of times that the basic SIPP is repeated in order to achieve the required soundness error.

The high level idea of the basic SIPP is to extend the SIPP from Claim 3.5, as we describe next. The verifier starts by drawing a single sample from x. The verifier then asks the prover to "simulate" some queries the tester T would have performed to x. In this list of queries, the verifier "hides" a single query which it already knows the value of (the sample it drew). After the prover sends (alleged) values for the queries, the verifier checks that T would have accept if it got these values from the queries, and that the value the prover returned for the "planted" query is consistent with the sample it drew. To get the desired soundness, the verifier and the prover use parallel repetition of this basic SIPP.

We describe this protocol more formally in Figure 3.

Completeness: Let $x \in L$. Since the tester T is fair, for every $i \in [n]$ and $j \in [k]$ there exists $\rho \in \{0,1\}^{r(n)}$ such that $q_j(\rho) = i$. That is, there exists a randomness that would have caused T to make its j-th query to the index i (that the verifier sampled). Furthermore, ρ is appropriately distributed. Hence, the verifier can indeed compute ρ in step 3, and send the prover $q_1(\rho), \ldots, q_k(\rho)$ in step 4. Now, if the prover is honest, it sends back to the verifier $x_{q_1(\rho)}, \ldots, x_{q_k(\rho)}$. In this case, the first condition for the verifier to accept holds, since $x^*_{q_j(\rho)} = x_i$. The second condition also hold, since from the completeness of T, the value verifier computes in step 6 is $T(x^*_{q_1(\rho)}, \ldots, x^*_{q_k(\rho)}; \rho) = (x_{q_1(\rho)}, \ldots, x_{q_k(\rho)}; \rho) = 1$.

Soundness: We first consider the case when T is a standard tester. Let $\epsilon > 0$ and let $x \in \{0,1\}^n$ be ϵ-far from L. We claim that the verifier in Figure 3 rejects the statement $x \in L$ with probability of at least $\frac{2}{3k}$.

Figure 3 SIPP for L with tester T, between a verifier $\mathcal{V}$ and a prover $\mathcal{P}$
Verifier Input: Sample access to $x \in \{0,1\}^n$.
Prover Input: Full (explicit) access to x.
Output: Accept / Reject

1. $\mathcal{V}$ draws (i, x_i), where $i \sim [n]$.
2. $\mathcal{V}$ chooses uniformly at random $j \sim [k]$.
3. $\mathcal{V}$ computes $\rho = R_j(i) \in \{0,1\}^{r(n)}$ such that $q_j(\rho) = i$.
4. $\mathcal{V}$ sends $q_1(\rho), \ldots, q_k(\rho)$ to $\mathcal{P}$.
5. Honest prover $\mathcal{P}$ sends $x_{q_1(\rho)}, \ldots, x_{q_k(\rho)}$ to $\mathcal{V}$.
6. $\mathcal{V}$ receives $x^*_{q_1(\rho)}, \ldots, x^*_{q_k(\rho)}$ and accepts iff $x^*_{q_j(\rho)} = x_i$ and $T(x^*_{q_1(\rho)}, \ldots, x^*_{q_k(\rho)}; \rho) = 1$.

Since the tester T is fair, as in the completeness case, the coin tosses ρ computed by $\mathcal{V}$ at step 3 are properly distributed (i.e., uniform and i.i.d). Therefore, from the soundness of T, we have:

$$\Pr[T(x_{q_1(\rho)}, \ldots, x_{q_k(\rho)}; \rho) = 1] \leq 1/3$$

where the probability is over the randomness of steps 1-3.

In the case that $T(x_{q_1(\rho)}, \ldots, x_{q_k(\rho)}; \rho) = 1$, the (malicious) prover can send the correct $x_{q_1(\rho)}, \ldots, x_{q_k(\rho)}$, and the verifier would (incorrectly) accept. But this case happens only with a probability of at most 1/3.

In the case that $T(x_{q_1(\rho)}, \ldots, x_{q_k(\rho)}; \rho) = 0$, the malicious prover has two options. The first option is to send the correct $x_{q_1(\rho)}, \ldots, x_{q_k(\rho)}$. In this option, the verifier will reject in step 6, as the second condition does not hold.

The second option for the prover is to send "incorrect" values $x^*_{q_1(\rho)}, \ldots, x^*_{q_k(\rho)} \neq x_{q_1(\rho)}, \ldots, x_{q_k(\rho)}$. In this option, there exists (at least) one query j^* on which the prover "cheats" on. That it, $x^*_{q_{j^*}(\rho)} \neq x_{q_{j^*}(\rho)}$. We now argue that the distribution of j conditioned on ρ is uniform over $[k]$. This is because $\Pr_{i \sim [n]}[R_j(i) = \rho] = \frac{1}{2^{r(n)}}$ for *every* fixed j. Therefore, with probability of $\frac{1}{k}$ we have $j = j^*$. In this case, as $q_j(\rho) = i$, the verifier also rejects in step 6, as the first condition $x^*_{q_j(\rho)} = x_i$ does not hold.

All in all, when $x \in \{0,1\}^n$ is ϵ-far from L, the verifier in Figure 3 rejects with probability of at least $\frac{2}{3k}$. The soundness error of the protocol can be reduced to a constant (say, 1/3) by repeating the basic protocol $O(k)$ times, in *parallel* (see [Gol98], Apdx C.1).

We next claim that if T is a POT with detection probability ϱ, and if x is ϵ-far from L, the verifier in Figure 3 rejects with probability of at least $\frac{\varrho(\epsilon)}{k}$. The analysis is similar to the case when T is a standard tester. Now, we have:

$$\Pr[T(x_{q_1(\rho)}, \ldots, x_{q_k(\rho)}; \rho) = 1] \geq \varrho(\epsilon)$$

Again, when $T(x_{q_1(\rho)}, \ldots, x_{q_k(\rho)}; \rho) = 0$, the prover has two options. If it sends the the correct values, the verifier will always rejects. If it sends incorrect values, then with probability of at least $\frac{1}{k}$ the verifier would catch it. In total, since $T(x_{q_1(\rho)}, \ldots, x_{q_k(\rho)}; \rho) = 0$ with probability of at least $\varrho(\epsilon)$, the verifier rejects the statement $x \in L$ with probability of at least $\frac{\varrho(\epsilon)}{k}$.

Reducing the soundness error to 1/3 can be done as before, by parallel repetition of the basic SIPP. Here the required number of repetitions is $O(\frac{k}{\varrho(\epsilon)})$.

Complexity Analysis: The SIPP of Figure 3 has 1 round. Each repetition of the basic protocol requires the verifier to draw a single sample. In addition, the communication complexity of each repetition is $O(k \log n)$. The verifier sends a list of length k of indices (each requires $\log n$ bits to be represented), and the prover sends a list of length k of bits.

Therefore, in the case T is a standard tester and the basic SIPP is repeated $O(k)$ times, the sample complexity is $s = O(k) = O(q(\epsilon))$, and the communication complexity is $c = O(k \log n \cdot k) = O(q(\epsilon)^2 \cdot \log n)$.

In the case T is a POT, the basic SIPP is repeated $O(k/\varrho(\epsilon))$ times, and sample complexity is $s = O(k/\varrho(\epsilon)) = O(q/\varrho(\epsilon))$, and the communication complexity is $c = O(k \log n \cdot \frac{k}{\varrho(\epsilon)})) = O(\frac{q^2}{\varrho(\epsilon)} \cdot \log n)$. $\square$

Application to Equality Testing: Observe that now, Claim 3.5 can be realized as application of Theorem 3.2 on the POT T described in Figure 1. But there is an issue with this POT - it is not fair. Its first query is always on the first part of the input, and the second query is on the second part. This can be resolved by "renaming" its queries as follows. Let $i \sim [2n]$, and take the indices of the POT's queries to be $q_1 = i$, and $q_2 = \begin{cases} i - n & \text{if } i > n \\ i + n & \text{otherwise} \end{cases}$. This way, both q_1 and q_2 are distributed uniformly over $[2n]$, and the POT T is fair.

The detection probability of T is $\varrho(\epsilon) = \epsilon$ (as was shown in Claim 3.3), and its query complexity is 2. Therefore, for the SIPP achieved by applying Theorem 3.2 on this POT has sample complexity $s = O(1/\epsilon)$ communication complexity $c = O(\log n/\epsilon)$, which are the same as in the SIPP from Claim 3.5.

3.3 Public-Coins Protocol

The protocol described in Claim 3.5 relies on the verifier using private coins. Namely, it keeps the index $i \in [n]$ it chooses secret; the soundness of the protocol does not hold if the prover knows i. In this section we consider a *public-coins* protocol for equality testing.

Theorem 3.6. *Let EQU be as in Definition 2.15, let $s : \mathbb{N} \to \mathbb{N}$ be a non-decreasing function, and let $\epsilon > 0$. There exists an 1-round* public-coins *SIPP for EQU with proximity parameter ϵ, sample complexity s and communication complexity $c = \widetilde{O}(\frac{n}{\epsilon \cdot s})$.*

Proof sketch. The protocol is as follows. Recall that the input is $z = (x, y) \in \{0,1\}^{2n}$, and that if $x = y$ the verifier should accept, and if x is ϵ-far from y it should reject with high probability. The verifier sends $c = \Theta(\log n \cdot \frac{n}{\epsilon \cdot s})$ random bits to the prover. In the case that $x = y$, the (honest) prover interprets the bits as $(c/\log n)$ indices, $I \subseteq [n]$, and sends the value of $x = y$ at the indices I. Note that for each $i \in I$, the prover sends a single bit b_i, but "commits" to the value of z at two indices, i and $i + n$. That is, the prover "claims" that $z_i = z_{i+n} = b_i$ for every $i \in I$.

Now, the verifier receives $\{b_i\}_{i \in I}$ from the prover. Next, it draws s samples, at indices $S \subseteq [2n]$, denoted $\{z_i\}_{i \in S}$. The verifier accepts iff the values of the samples are consistence with the message the prover sent. That is, it checks that for every $i \in S$ such that $i \leq n$ and $i \in I$ it holds that $b_i = z_i$, and that for every $i \in S$ such that $i > n$ and $i - n \in I$ it holds that $b_{i-n} = z_i$.

The completeness of the protocol is immediate. We next prove that the soundness of the protocol holds.

If x is ϵ-far from y, then there exists a set of indices $B \subseteq [n]$ of size at least $\epsilon \cdot n$ such that $x_i \neq y_i$ for every $i \in B$. Let $B' = I \cap B$. Since I is a set of $\Theta(\frac{n}{\epsilon \cdot s})$ indices from $[n]$ chosen uniformly at random and i.i.d, with high probability $|B'| = \Omega(\frac{n}{s})$. Now, for every $i \in B'$ a cheating prover must send a value b_i, but $x_i \neq y_i$. Thus, the prover must 'lie" in its message regarding the values at those indices. That is, for every $i \in B'$, it must send $b_i \neq x_i$ or $b_i \neq y_i$.

Let $B'' \subseteq [2n]$ be the set of indices in which the value of the input $z = (x, y)$ does not match value from the prover's message. That is, $i \in B''$ if $i \leq n$ and $b_i \neq z_i$, or if $i > n$ and $b_{i-n} \neq z_i$. Note that $|B''| = |B'|$. But now, since the verifier draws s samples and $|B''| = |B'| = \Omega(\frac{n}{s})$ (with high probability), we get that with high probability $S \cap B'' \neq \varnothing$. That is, with high probability, the verifier sampled some index $i \in [2n]$ such that $b_i \neq z_i$ (and $i \leq n$) or $b_{i-n} \neq z_i$ (and $i > n$). In both cases, from the definition of the protocol, the verifier rejects. That is, we showed that if x is ϵ-far from y, then with high probability the verifier rejects. □

Note that for constant ϵ, this protocol is optimal, up to a $\log n$ factor. Moreover, a $\log n$ factor can be shaved off using standard derandomization techniques (e.g. expander graph random walks). See Theorem 4.7.

4 Lower Bounds

In the previous section we showed a sample-based interactive proof of proximity for equality, which used private coins. In this section, we consider two variations of the model. Namely, we study *non-interactive* sample-based proofs of proximity, and *public coin* sample-based interactive proofs of proximity. In both of these settings we prove lower bounds for the Equality Testing Problem.

4.1 Non-Interactive Sample-Based Proofs of Proximity (SMAPs):

Formally, we define *non-interactive sample-based proof of proximity* to be a SIPP, where the communication is restricted to be a single message from the prover to the verifier. Namely, in a non-interactive sample-based proof of proximity for a property L, the verifier is given *sample-based* access an input x and *explicit* access to a proof string π. We require that for inputs $x \in L$ there exists a proof π that the verifier accepts, and for inputs that are ϵ-far from L, no proof will make the verifier accept, except with some small probability of error. We are mainly interested in two measures of complexity for such proof systems. The first is the sample complexity of the verifier (the number of samples it draws from the input x), which we denote $s = s(n, \epsilon)$. The second measure of complexity is the length of the proof, which we denote $p = p(n, \epsilon)$. This measure is the equivalent of the communication complexity in the interactive setting.[13] We are also interested in the time complexity of the verifier, and consider it to be efficient if it is linear in $s(n, \epsilon) + c(n, \epsilon)$.

We stress that the verifier has *full* (explicit) access to the proof string π. That is, it is not "charged" for reading bits from the proof. The study of a "PCP-like" model, where the verifier might read only (small) part of the proof, is left for future work.

The notion of non-interactive proofs of proximity where the verifier has *query-access* to the input was first studied by Gur and Rothblum [GR18], which named them *Merlin-Arthur proofs of proximity* (MAPs). Accordingly, we call the new type of non-interactive proofs studied here *Sample-Based Merlin-Arthur proofs of proximity* (SMAPs).

Known Results: Though the notion of SMAPs is new and introduced in this work, a couple of results from [GR18] apply to the SMAPs setting as well. We cite them here, interpreted as results regarding SMAP, for context.

The first result is the observation that any property has a SMAP with *linear* proof length and *constant* sample complexity. Namely, the proof π for the statement $x \in L$ is simply x itself. The verifier can check that π is in L without even looking at the input x, and it can check that $\pi = x$

[13]Indeed, when we consider a non-interactive sample-based proof of proximity as a SIPP restricted to a single message from the prover, $p(n, \epsilon) = c(n, \epsilon)$.

(w.h.p) by drawing a constant number of samples from x, and verifying that x and π agree on the sampled indices.[14]

In particular, for the Equality Testing problem, we get the following result:

Claim 4.1. *Let EQU be as in Definition 2.15, and let $\epsilon > 0$. Then there exists a SMAP for L, with proof length $p(n) = O(n)$ and sample complexity $s(n) = O(1/\epsilon)$.*

Another result which applies to the SMAP setting is that a SMAP can provide at most quadratic improvement over a standard property tester (Theorem 4.2 in [GR18]).[15]

Claim 4.2. *Let L be a language with SMAP with proximity parameter ϵ, proof length p and sample complexity s. Then L has a sample-based property tester with proximity parameter ϵ and sample complexity $O(p \cdot s)$.*

Using the known lower bound for the sample complexity of sample-based testers for equality, $s = \Omega(\sqrt{n})$ (see Claim 3.4), we deduce from Claim 4.2 the following result:

Corollary 4.3. *Let EQU be as in Definition 2.15, and fix $\epsilon = 0.1$. If a SMAP for L with proximity parameter ϵ has proof length p and sample complexity s, then $p \cdot s = \Omega(\sqrt{n})$.*

We next prove the following result:

Theorem 4.4. *Let EQU be as in Definition 2.15, and fix $\epsilon = 0.1$. If a SMAP for EQU with proximity parameter ϵ has proof length $p = o(n)$, then its sample complexity is $s = \Omega(\frac{\sqrt{n}}{\log n})$.*

Recall that there is a sample-based tester for equality with sample complexity $s = O(\sqrt{n})$. In addition, from Proposition 4.1, there exists a SMAP with linear proof length and constant sample complexity. One might wonder if the length of the proof might be shorter than linear, perhaps by increasing the sample complexity of the verifier to be super-constant. Theorem 4.4 shows that "nothing" non-trivial can be done. As soon as the proof length is sub-linear, the sample complexity of the verifier must be as large as the sample complexity of a tester that is not aided by a proof (up to a logarithmic factor).

We prove Theorem 4.4 by a reduction to a Communication Complexity problem. We next describe the model of *Consecutive Messages Protocols*, introduced by Naor and Rothblum [NR05], and a Theorem in this model which is used to prove Theorem 4.4.

Consecutive Messages Protocols: A consecutive messages (CM) protocol for a function $f : X \times Y \to \{0, 1\}$ is a protocol between 3 players: Alice, Bob and Carol. We view a CM protocol as a game between an adversary and the players, where the adversary selects Alice's and Bob's inputs ($x \in X$ and $y \in Y$ 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

[14]Another result is that any *sparse* property has a SMAP with short proof length and constant sample complexity. Namely, a property L with $t = |L|$ elements has a SMAP with $p(n) = \log_2 t$, where the proof π for $x \in L$ is the "index" of x in some agreed enumeration of the elements of L. By drawing a constant number of samples from x, the verifier can distinguish between the case of π being the index of x, and x being far from the element in L which is at index π. We note that the verifier might not be time-efficient in this SMAP construction, since computing what is the element of L at index π might be computationally difficult. Also note that the Equality property is not sparse, as $\log|\text{EQU}| = \log(2^{n/2}) = \Theta(n)$.

[15]In [GR18] the authors study MAPs with *proof oblivious queries*. These are MAPs in which the verifer's queries are independent of the provided proof. It is easy to see that any SMAP can be interpreted as MAP with proof oblivious queries, and therefore their results applies to SMAPs as well. In the statement of Theorem 4.2 in [GR18], the resulting tester is *query-based* tester with $O(p \cdot s)$ query complexity. It is implicit in the proof that if one starts with sample-based verifier, then the resulting tester is also *sample-based*.

not high. In this model all communication channels are reliable - the adversary cannot modify any of the messages sent by the players. 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 players can use private random coins (Bob and Carol share their coins).[16] 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 and her private random coins ρ_A. Alice sends m_A to Carol over their private channel and publishes the public message m_p.
3. The adversary sees m_p and then selects an input $y \in Y$ for Bob.
4. Bob uses (y, m_p, ρ_{BC}) to compute a message m_B to Carol. ρ_{BC} are Bob and Carol's shared coins.
5. Carol computes her output $b_C \in \{0,1\}$ as a function of $(m_A, m_B, m_p, \rho_{BC})$

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. The requirement for a CM protocol is that for any adversary that selects an Alice's input $x \in X$, then receives 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 α for some constant $\alpha > \frac{1}{2}$.

Lower Bound for CM protocol for Equality: We now consider the Equality Problem, where $X = Y = \{0,1\}^n$, and for $x, y \in \{0,1\}^n$, $f(x,y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases}$.

[NR05] proved the following lower bound for a CM protocol for this problem:

Theorem 4.5. *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 $\ell_A \cdot \ell_B \geq C \cdot n$, where $C > 0$ is some constant.*

We note that the *simultaneous messages* (SM) communication model, where Alice and Bob get their input simultaneously, and no public message is allowed, was well studied prior to the work of [NR05]. The model was introduced by Yao in his seminal paper on communication complexity [Yao79]. In this model, a similar lower bound for equality was shown by Newman and Szegedy [NS96], and it was generalized later by Babai and Kimmel [BK97]. The contribution of [NR05] was extending the lower bound for the CM case, where a public message is allowed.

For Theorem 4.4, we view the public message as a (non-interactive) proof. This allows us to make a reduction from SMAP for equality to CM protocol for equality, as we do next.

In the reduction we use an asymptotically good error correcting code:

Fact 4.6. *(Existence of Good Codes) Let $\epsilon = 0.1$. There exists a constant t and an asymptotically good error correcting code with rate t and relative distance ϵ.*

A construction of such a code can be obtained by the Gilbert-Varshamov Bound [Gil52, Var57].

In the proof of Theorem 4.4 we also use the following notations. Observe that from the definition of SMAP with proof length p for equality testing, for every $x \in \{0,1\}^n$ there exists a proof $\pi \in \{0,1\}^p$ such that a verifier with sample access to $(x,x) \in \{0,1\}^{2n}$ and explicit access to π, always accepts.

[16]Note that in the original description of CM protocols, the players could also use public randomness. We omit this element of the protocol since it is not required in this work, and since omitting it only makes the impossibility result from [NR05] stronger (since the players in the protocol are only more restricted).

For every $x \in \{0,1\}^n$, we denote that "proof for x" by $\pi_x \in \{0,1\}^p$.[17] We also make the following definition. Let $\mathcal{V}$ be a verifier of an equality SMAP with sample complexity s, and let $S \in [2n]^s$ be indices of s samples. We denote by $\mathcal{V}(z\big|_S; \pi)$ the output of $\mathcal{V}$ which receives $z \in \{0,1\}^{2n}$ as implicit input, π as explicit input (proof), and conditioned on the event that the samples $\mathcal{V}$ draws are exactly at indices S.

Note that under this terminology, the completeness requirement for the SMAP can be phrased as:

$$\forall x \in \{0,1\}^n \exists \pi \in \{0,1\}^p \text{ s.t } \Pr_{S \sim \mathbf{S}}[\mathcal{V}((x \circ x)\big|_S; \pi) = 1] = 1$$

Where $\mathbf{S}$ is the uniform distribution on $[2n]^s$ (that is, s-tuples of elements from $[2n]$), and the probability is also over the internal coin tosses of $\mathcal{V}$.

The soundness requirement is:

$$\forall x, y \in \{0,1\}^n \text{ s.t } \Delta(x,y) > 0.1, \forall \pi^* \in \{0,1\}^p \Pr_{S \sim \mathbf{S}}[\mathcal{V}((x \circ y)\big|_S; \pi^*) = 0] \geq 1 - \delta$$

where $1 - \delta$ for $\delta < \frac{1}{2}$ is the soundness error of the protocol.

We are now ready to prove Theorem 4.4.

Proof. Let t be the rate of the code from Fact 4.6, and let $k = t \cdot n$ (for simplicity of notation, assume that k is an integer).

Assume that there exists an SMAP for the Equality Testing Problem, for inputs of length $2n$, with proof length $p(2n)$, sample complexity $s(2n)$, proximity parameter $\epsilon = 0.1$, and soundness error δ. We construct a CM protocol for equality, for $X = Y = \{0,1\}^k$, with messages length $\ell_A = \ell_B = \Theta(\log k \cdot s(k))$, public message length $\Theta(p(k))$ and success probability $1 - \delta$.

The Reduction: The CM protocol for equality works as follows. First, Alice receives input $x \in \{0,1\}^k$, and encodes it using $\mathrm{ECC} : \{0,1\}^k \to \{0,1\}^n$, the asymptotically good error correcting code from Fact 4.6, to get $\mathrm{ECC}(x)$. She then draws uniformly at random and i.i.d s samples from $\mathrm{ECC}(x)$, denote their indices by $S_A \in [n]^s$. Alice sends her samples $\{(i, x_i) \mid i \in S_A\}$ to Carol as her private message m_A. Now, consider the SMAP for Equality Testing from the assumption, for inputs of length $2n$. From its completeness, there exists a proof $\pi \in \{0,1\}^{p(2n)}$ such that the verifier, on explicit input π and implicit input $\mathrm{ECC}(x) \circ \mathrm{ECC}(x) \in \{0,1\}^{2n}$ always accepts. That is, $\Pr_{S \sim \mathbf{S}}[\mathcal{V}((\mathrm{ECC}(x) \circ \mathrm{ECC}(x))\big|_S; \pi) = 1] = 1$. Let $\pi_{\mathrm{ECC}(x)} = \pi$ be this proof. Now, as her public message m_p, Alice publishes $\pi_{\mathrm{ECC}(x)}$.

Next, Bob receives input $y \in \{0,1\}^k$. He encodes it by the same error correcting code Alice used, to get $\mathrm{ECC}(y) \in \{0,1\}^n$. Bob then draws uniformly a random and i.i.d s samples from $\mathrm{ECC}(y)$, denote their indices by $S_B \in [n]^s$. Bob sends his samples $\{(i, y_i) \mid i \in S_B\}$ to Carol as his private message m_B. Bob ignores the public message sent by Alice.

Now, Carol sees the samples drawn by Alice from $\mathrm{ECC}(x)$, the samples drawn by Bob from $\mathrm{ECC}(y)$, and $\pi_{\mathrm{ECC}(x)}$ the public message sent by Alice. Carol computes her output $b_C \in \{0,1\}$ by the following procedure. First, she chooses uniformly at random and i.i.d s samples from the $2s$ samples she received from Alice and Bob. Denote the indices of the samples she chooses from the samples sent by Alice (resp., Bob) by S'_A (resp., S'_B). Denote $S_C = S'_A \cup (n + S'_B) \in [2n]^s$. Next, Carol simulates the execution of the verifier of the Equality SMAP for the statement $\mathrm{ECC}(x) = \mathrm{ECC}(y)$, as if it received samples at indices S_C, and $\pi_{\mathrm{ECC}(x)}$ as the proof.

That is, she computes $b_C = \mathcal{V}((\mathrm{ECC}(x) \circ \mathrm{ECC}(y))\big|_{S_C}, \pi_{\mathrm{ECC}(x)})$, which is her final output.

[17]Note that every x is mapped by the SMAP into single proof π_x, but it is possible that many inputs will be mapped to the same proof.

Complexity Analysis: Alice and Bob send $s(2n)$ samples each, where the samples are of the form $(i, z_i) \in [n] \times \{0,1\}$. Therefore, the length of their private messages is $\ell_A = \ell_B = \log n \cdot s(2n) = \Theta(\log k \cdot s(k))$.

The public message sent by Alice is $\pi_{\mathrm{ECC}(x)}$, which is of length $p(2n) = \Theta(p(k))$.

Correctness: We now show that the success probability of the described CM protocol is $1 - \delta$ (where δ is the soundness error of the SMAP). First, observe that when S_A and S_B are s-tuples of elements drawn uniformly at random and i.i.d from $[n]$, then the distribution of the indices S_C is as they were drawn uniformly at random and i.i.d from $[2n]$. That is, the distribution of S_C is $\mathbf{S}$, the uniform distribution on $[2n]^s$. This observation simply follows from the procedure that Carol uses to choose S_C. Also observe that Carol receives from Alice and Bob the value of $\mathrm{ECC}(x) \circ \mathrm{ECC}(y)$ at the s indices S_C (along with values of s other bits, which she does not use). Therefore, Carol can indeed simulate the verifier and compute the value $\mathcal{V}((\mathrm{ECC}(x) \circ \mathrm{ECC}(y)\big|_{S_C}; \pi_{\mathrm{ECC}(x)})$. In addition, the probability of Carol to accept equals to the probability of the verifier to accept when given (sample-based) access to $\mathrm{ECC}(x) \circ \mathrm{ECC}(y)$ and $\pi_{\mathrm{ECC}(x)}$ as a proof. This is since the distribution of S_C is $\mathbf{S}$ and because, by definition, for any x, y and any purported proof π, it holds that:

$$\Pr_{\mathcal{V}\text{'s samples}}[\mathcal{V}^{\sim(\mathrm{ECC}(x)\circ\mathrm{ECC}(y))}(\pi) = 1] = \Pr_{S\sim\mathbf{S}}[\mathcal{V}(\mathrm{ECC}(x) \circ \mathrm{ECC}(y)\big|_{S}; \pi) = 1]$$

where the probabilities are also over the coins of $\mathcal{V}$.

Now, consider some possible strategy for the adversary. We argue that the adversary cannot make Carol reject while giving to Alice and Bob inputs $x = y$, and cannot make Carol accept while giving to Alice and Bob inputs $x \neq y$ (except for small probability). For the first case, assume the adversary gave to Alice and Bob inputs $x = y$. In this case, the public message sent by Alice is $\pi_{\mathrm{ECC}(x)}$. Recall that $\pi_{\mathrm{ECC}(x)}$ is the proof such that $\mathcal{V}$, when given sample-based access to $\mathrm{ECC}(x) \circ \mathrm{ECC}(x)$ (which equals $\mathrm{ECC}(x) \circ \mathrm{ECC}(y)$, as $x = y$) and the proof $\pi_{\mathrm{ECC}(x)}$ would always accept. Since Carol simulates the execution of $\mathcal{V}$ on implicit input $\mathrm{ECC}(x) \circ \mathrm{ECC}(y) = \mathrm{ECC}(x) \circ \mathrm{ECC}(x)$ and proof $\pi_{\mathrm{ECC}(x)}$, we conclude that Carol accepts. In other words, the completeness of the SMAP implies the completeness of the CM protocol.

For the second case, assume the adversary gave to Alice and Bob inputs $x \neq y$. In this case, the public message sent by Alice is also $\pi_{\mathrm{ECC}(x)}$ (and the adversary might choose y according to this public message). Assume (towards contradiction) that the adversary makes Carol accept with high probability (larger than δ). We show that such an adversary "breaks" the soundness of the SMAP protocol. Indeed, in this case, as $x \neq y$ and ECC is an error correcting code with distance 0.1, we have $\Delta(\mathrm{ECC}(x), \mathrm{ECC}(y)) \geq 0.1$. That is, $\mathrm{ECC}(y)$ is 0.1-far from $\mathrm{ECC}(x)$. In addition, Carol simulates the execution of $\mathcal{V}$ on implicit input $\mathrm{ECC}(x) \circ \mathrm{ECC}(y)$ (and proof $\pi_{\mathrm{ECC}(x)}$) and accepts with high probability. We deduct that there exists an input ($z = \mathrm{ECC}(x) \circ \mathrm{ECC}(y)$) that is 0.1-far from the equality property, and a proof ($\pi = \pi_{\mathrm{ECC}(x)}$), such that $\mathcal{V}$, when given sample-based access to the input z and the proof π, accepts with high probability (larger than δ). This is a contradiction to the soundness of the SMAP. In other words, an adversary that "breaks" the soundness of the CM implies an adversary that "breaks" the soundness of the SMAP. We conclude that the CM protocol is complete and sound, and that its success probability is at least $1 - \delta$.

Lower Bound: Assume towards contradiction that there exists an SMAP for Equality Testing, with proximity parameter $\epsilon = 0.1$, soundness error $1/3$, proof length $p = o(n)$ and sample complexity $s = o(\frac{\sqrt{n}}{\log n})$. The soundness error of the SMAP can be reduced to $1 - 0.83$ by executing the verifier twice, each time with the fresh samples, and with the same proof. This hurts the sample complexity by only a constant factor. Now, from the reduction, there exists a CM protocol for equality of inputs of size $\Theta(n)$, with messages length $\ell_A = \ell_B = o(\log n \cdot \frac{\sqrt{n}}{\log n}) = o(\sqrt{n})$ (and hence $\ell_A \cdot \ell_B = o(n)$), public message length $p = o(n)$ and success probability 0.83. This is a contradiction to Theorem 4.5. □

4.2 Public-Coin SIPPs

We say that an interaction uses *public-coins* if the verifier reveals the outcome of its coins immediately after tossing them. In such a protocol, the messages the verifier sends to the prover are simply the results of its coin flips, and he is not allowed to send any other messages. We call other interactive protocols *private-coins* protocols.

Note that the equality testing protocol of Claim 3.5, and the more general protocol of Theorem 3.1 are private-coins protocols.[18] It is not clear if it is possible to convert them into public-coins protocols, or how.

A classic result in the theory of standard interactive protocols is that the expressive power of private-coin interactive proofs is essentially equivalent to that of public-coin interactive proofs [GS86]. This fact was extended to the IPP setting by [RVW13], when the verifier has query-access to the input.

In this section, we show that this fact does not extend to the case where the verifier has *sample-access* to the input. We prove that public-coins SIPP are *much weaker* than private-coins SIPP. Specifically, we show that Equality Testing does not have a public-coins SIPP with communication and sample complexities that are as good as the private-coins SIPP showed in Claim 3.5.

Theorem 4.7. *Let EQU be as in Definition 2.15. If a public-coins SIPP for EQU with proximity parameter $\epsilon = 0.1$ has communication complexity c and sample complexity $s \le \frac{\sqrt{n}}{1000}$, then $c \cdot s > \frac{n}{10000}$.*

Note that this lower bound is tight, up to a logarithmic factor. There exists a sample-based tester for equality testing (without a prover) with sample complexity $O(\sqrt{n})$. In addition, for any desired sample complexity $s \le \sqrt{n}$ there is a (1-round) public-coins SIPP for equality testing, with sample complexity s and communication complexity $c = \widetilde{O}(\frac{n}{s})$ (and thus $c \cdot s = \widetilde{O}(n)$). See Theorem 3.6.

The proof of Theorem 4.7 uses tools from information theory. In the proof of the Theorem we use the following technical Lemma, which states that if a random variable has almost full Shannon entropy, then its distribution is close to uniform.

Lemma 4.8. *Let X be a random variable with a finite support S. Assume that $H(X) \ge \log|S| - \epsilon$. Then X is $\sqrt{\epsilon/2}$-close to the uniform distribution over S.*

Proof. The Lemma follows from Pinkser's inequality, as we show next. Let P be the probability distribution of X. That is, for every $x \in S$ we have $P(x) = \Pr[X = x]$. Let Q be the uniform probability distribution on S. That is, for every $x \in S, Q(x) = \frac{1}{|S|}$. From the definitions of Kullback-Leibler divergence and Entropy, we have the following:

$$\begin{aligned} D_{KL}(P||Q) &= \sum_{x\in S} P(x)\cdot\log\left(\frac{P(x)}{Q(x)}\right) = \sum_{x\in S} P(x)\cdot\log(P(x)\cdot|S|) = \sum_{x\in S} P(x)\cdot(\log P(x) + \log|S|) = \\ &= \sum_{x\in S} P(x)\cdot\log P(x) + \sum_{x\in S} P(x)\cdot\log|S| = -H(X) + \log|S| \end{aligned} \tag{1}$$

From the hypothesis $H(X) \ge \log|S| - \epsilon$, and hence:

$$D_{KL}(P||Q) = -H(X) + \log|S| \le -\log|S| + \epsilon + \log|S| = \epsilon$$

[18] Also note that the reduction protocol used for Theorem 5.2 in the next section is also a private-coins protocol.

Now, from Pinkser's Inequality we get that:

$$\Delta(P,Q) \leq \sqrt{\frac{1}{2} D_{KL}(P||Q)} \leq \sqrt{\frac{\epsilon}{2}}$$

where $\Delta(P,Q)$ is the statistical distance between P and Q. $\square$

For the proof of Theorem 4.7 we also need the following Lemmas, which we cite without proof.

Lemma 4.9. *(Shearer's Lemma) Let $X = X_1, \ldots, X_n$ be a random variable. If $\mathbf{S}$ is any distribution on subsets of the coordinates $[n]$, such for every i, $\Pr_{S \sim \mathbf{S}}[i \in S] \geq \mu$, then $\mathbb{E}_{S \sim \mathbf{S}}[H(X_S)] \geq \mu \cdot H(X)$.*

Note that the entropy $H(X_S)$ is with respect to a fixed S. That is, the "uncertainty" measured in $H(X_S)$ holds after S is fixed. We use conditional entropy to reflect this fact, and write below $\mathbb{E}_{S \sim \mathbf{S}}[H(X_S \mid S)] \geq \mu \cdot H(X)$ as the conclusion of Shearer's Lemma.

Lemma 4.10. *Let A, B be random variables, and let $\delta > 0$. With probability at least $1-\delta$ over $b \sim B$, the conditional entropy of A given $B = b$ is at least $\widetilde{H}_\infty(A \mid B) - \log(1/\delta)$.*

See Definition 2.4 for the definition of the average min-entropy $\widetilde{H}_\infty(A \mid B)$, and Definition 2.3 for the definition of conditional entropy. For a proof of Lemma 4.10 see Lemma 2.2 (a) in the work of Dodis et al. [DORS08].

Lemma 4.11. *Let A, B be random variables, and assume B has at most 2^λ possible values. Then $\widetilde{H}_\infty(A \mid B) \geq H_\infty(A) - \lambda$.*

For a proof of Lemma 4.11 see Lemma 2.2 (b) in [DORS08].

We are now ready to prove Theorem 4.7.

Proof. Consider a public-coin SIPP of equality between a verifier $\mathcal{V}$ and a prover $\mathcal{P}$ with communication complexity c and sample complexity $s \leq \frac{\sqrt{n}}{1000}$. Note that because the protocol is public-coin, we can assume w.l.o.g that the verifier draws its samples after the interaction ends.

We argue that the verifier, even with the help of the prover, cannot distinguish between an input it should accept (sampled uniformly from the set of all accepting inputs) and an input sampled uniformly from the set of all possible inputs (which it should reject with high probability, as we prove below). Namely, let D_{YES} be the uniform distribution over $\{(x,x) \mid \in \{0,1\}^n\} \subseteq \{0,1\}^{2n}$ (the accepting inputs), and let D_{NO} be the uniform distribution over all of $\{0,1\}^{2n}$. We claim that:

1. The verifier must accept every input from the distribution D_{YES}.
2. With high probability, the verifier rejects inputs sampled from the distribution D_{NO}.
3. If $c \cdot s \leq \frac{n}{10000}$ then the (sample-based) verifier cannot distinguish between an input sampled from D_{YES} and input sampled from D_{NO}, even with help from the prover.

From the definition of D_{YES}, $\mathcal{V}$ must accept every input sampled from this distribution. In addition, it must reject with probability at least $1/2$ an input z sampled from D_{NO}, where the probability is over the choice of z, the public coins and the verifier's samples. Observe that for every $i \in [n]$ we have $\Pr_{z=(x,y) \sim D_{\text{NO}}}[x_i = y_i] = \frac{1}{2}$. Hence, from Chernoff's bound, $\Pr_{z=(x,y) \sim D_{\text{NO}}}[\Delta(x,y) > 0.1] > 0.99$ (for sufficiently large n), and in this case z is 0.1-far from EQU. Therefore, since $\mathcal{V}$ must reject any input that is 0.1-far from EQU with probability at least $2/3$, it must reject $z \sim D_{\text{NO}}$ with probability at least $2/3 - 0.01 > 1/2$.

To show that the verifier cannot distinguish between inputs sampled from D_{YES} and from D_{NO}, we consider the *view* of the verifier. For some distribution D, consider the execution of the protocol for $z \sim D$ as the implicit input. Let d be the number of rounds in the protocol, and for $j \in [d]$ denote by α_j the message the verifier sends at round j. Denote by β_j the message that the prover sends at round j. We say that $(\alpha_1, \dots, \alpha_d, \beta_1, \dots, \beta_d)$ is the *transcript* of the execution of the protocol. Let $r(n)$ be the number of (public) coins the verifier tossed, and let $\rho = \alpha_1 \circ \dots \circ \alpha_d \in \{0,1\}^{r(n)}$ be their values. Let $\pi = \beta_1 \circ \dots \circ \beta_d$ be all of the messages that the prover sent to the verifier. For a set of indices $S \subseteq [2n]$, denote by $z_S \in \{0,1\}^{|S|}$ the projection of z to the indices S.

Now, the *view* of the verifier is the random variable (ρ, π, S, z_S), where $\rho \sim \{0,1\}^{r(n)}$, $S \sim \mathbf{S_{2n}}$ and $z \sim D$, where $\mathbf{S_{2n}}$ is the uniform distribution of subsets of size s over $[2n]$. Note that at the end of the protocol, all of the information available to the verifier is contained in its view. Thus, its verdict on the claim $x = y$ is a function of this view.

Consider the case when $z \sim D_{\text{YES}}$. That is, $z = (x, x)$ for $x \in \{0,1\}^n$. There is some prover strategy that makes the verifier accept. That is, for every ρ there exists $\pi = \pi(x, \rho)$ such that for every $S \subseteq [2n]$ the verifier accepts when its view is $(\rho, \pi(x,\rho), S, (x,x)_S)$.[19]

We next consider the view of the verifier when $z \sim D_{\text{NO}}$. That is, $z = (x, y)$ for $x, y \sim \{0,1\}^n$. Suppose that the (cheating) prover's strategy is as follows. It draws $w \sim \{0,1\}^n$, and then sends its messages during the protocol as if the input was (w, w). For public coins ρ the concatenation of the prover's messages is $\pi(w, \rho)$. In this case, the view of the verifier is $(\rho, \pi(w,\rho), S, (x,y)_S)$, where $\rho \sim \{0,1\}^{r(n)}, x, y, w \sim \{0,1\}^n$ and S is a set of s indices sampled uniformly at random and i.i.d from $[2n]$. Note that now $\pi(w, \rho)$ is independent of $(x,y)_S$ (and of S). Since all of the bits of x and y are independent of each other (and of everything else in the verifier's view), the distribution of the input part of the view (i.e. $(x,y)_S$)) conditioned on the other parts of the view, is of s uniform and i.i.d random bits. Recall that from the soundness of the SIPP, the verifier must reject with high probability when this is its view.

We now claim that the verifier cannot distinguish between the view $(\rho, \pi(x,\rho), S, (x,x)_S)$ and the view $(\rho, \pi(w,\rho), S, (x,y)_S)$, except with a small advantage. Since, as stated above, $(x,y)_S$ are simply s random bits, distributed uniformly at random and i.i.d, this claim reduces to showing that the distribution of $(x,x)_S$, conditioned on $\pi(x,\rho), \rho$ and S, is, with high probability, very close to uniform.

At a high level, the proof of this goes as follows. First, since $x \sim \{0,1\}^n$, the random variable x has full entropy (namely, $H(x) = n$). Since the communication complexity of the SIPP is low, $\pi(x,\rho)$ is short, and hence x still has almost full entropy even conditioned on $\pi(x,\rho)$. Since x and ρ are independent random variables, x still has almost full entropy conditioned on $\pi(x,\rho), \rho$. Next, since $|S| = s \leq \frac{\sqrt{n}}{1000}$, with high probability the verifier does not see any "collisions" in its samples. That is, for the input $z = (x,x) \in \{0,1\}^{2n}$, it does not sample both z_i and z_{n+i} for any $i \in [n]$. Thus, its samples can be considered as they are all in the "first part" of the input (i.e., S can be considered as a random subset of $[n]$, instead as of $[2n]$). Next, since S is a uniformly random subset, x_S conditioned on $\pi(x,\rho), \rho, S$ still has almost full entropy (this argument follows from Shearer's Lemma). Finally, from the Lemma we proved above, since x_S conditioned on $\pi(x,\rho), \rho, S$ has almost full entropy, its distribution is close to uniform.

We next prove these arguments in more detail. First, from the definition of x we have $H_\infty(x) = H(x) = n$. Since x and ρ are independent, we have $H_\infty(x \mid \rho) = H_\infty(x) = n$.

Since the communication complexity of the SIPP is c, the length of $\pi(x,\rho)$ is at most c. That is, $\pi(x,\rho)$ is a random variable that has at most 2^c possible values. Hence, from Lemma 4.11 we have:

$$\widetilde{H}_\infty(x \mid \pi(x,\rho), \rho) \geq H_\infty(x \mid \rho) - c = n - c$$

From Lemma 4.10, we get that with probability of at least 0.99 over the choice of x and ρ, $H(x \mid \pi(x,\rho) = \pi, \rho) \geq n - c - \log 100 \geq n - c - 7$. Let B_1 be the event (over $x \sim \{0,1\}^n$ and

[19] We assume w.l.o.g that the honest prover is deterministic.

$\rho \sim \{0,1\}^{r(n)}$) that the inequality above does not hold. Then, if B_1 does not happen, we have:

$$H(x \mid \pi(x,\rho),\rho) \geq n - c - 7$$

That is, from the verifier's perspective, after its interaction with the prover ends, the entropy of x conditioned on the transcript of the protocol is larger than $n-c-7$ (unless B_1 happens).

We next show that the entropy of the part of x that the verifier "sees" is also large. Recall that the full implicit input is $z=(x,x)$ for $x \sim \{0,1\}^n$. Let B_2 be the event that the verifier sampled both z_i and z_{n+i} for some $i \in [n]$ (i.e., a "collision" happened). Since $s \leq \frac{\sqrt{n}}{1000}$, the event B_2 happens with probability at most 0.01 (over the verifier's samples).

Consider the view of the verifier $(\rho, \pi(x,\rho), S, (x,x)_S)$ when B_2 does not happen. We argue that it is equivalent to the view $(\rho, \pi(x,\rho), S', x_{S'})$ where $S' \subseteq [n]$, is constructed by the following procedure. For every $i \in S$, if $i \leq n$ then i is added to S', and if $i > n$ then $i-n$ is added to S'. That is, we identify the sample (i, z_i) with the sample $(n+i, z_{n+i})$, where we have $z_i = z_{n+i} = x_i$ since $z=(x,x)$. Since B_2 does not happen, the resulting S' is indeed a set (i.e., it has no repetitions) and $|S'| = |S| = s$. Moreover, in this case the views are indeed equivalent to each other, as $(x,x)_S = x_{S'} \in \{0,1\}^s$ for every $x \in \{0,1\}^n$ and $S \subseteq [2n]$. If the verifier accepts the view $(\rho,\pi(x,\rho),S,(x,x)_S)$, and the event B_2 does not happen, it must also accept the view $(\rho,\pi(x,\rho),S',x_{S'})$.

Letting $\mathbf{S_n}$ be the uniform distribution of subsets of size s over $[n]$, the indices of the verifier's samples can now be considered as a random subset $S' \sim \mathbf{S_n}$. For this distribution we have

$$\Pr_{S' \sim \mathbf{S_n}}[i \in S'] = \frac{s}{n}$$

for every $i \in [n]$. Thus, we can apply Lemma 4.9 (Shearer's Lemma) with $\mu = \frac{s}{n}$ to get:[20]

$$\mathbb{E}_{S' \sim \mathbf{S_n}}[H(x_{S'} \mid \pi(x,\rho),\rho,S')] \geq \frac{s}{n} \cdot H(x \mid \pi(x,\rho),\rho)$$

Assuming that B_1 also does not happen, we have $H(x \mid \pi(x,\rho),\rho) \geq n-c-7$ and get:

$$\mathbb{E}_{S' \sim \mathbf{S_n}}[H(x_{S'} \mid \pi(x,\rho),\rho,S')] \geq \frac{s}{n} \cdot (n-c-7) = s - \frac{s \cdot (c+7)}{n}$$

By the contradiction assumption we have $c \cdot s \leq \frac{n}{10000}$ and $s \leq \frac{\sqrt{n}}{1000}$, and get that (for sufficiently large n):

$$\mathbb{E}_{S' \sim \mathbf{S_n}}[H(x_{S'} \mid \pi(x,\rho),\rho,S')] \geq s - \frac{s \cdot c}{n} - \frac{s \cdot 7}{n} \geq s - \frac{1}{10000} - \frac{1}{100} = s - \frac{1}{9900}$$

That is, the expected entropy of the part of x that the verifier "sees" is almost full (conditioned on the other parts of its view, and assuming B_1 and B_2 do not happen).

Since $s \geq H(x_{S'} \mid \pi(x,\rho),\rho,S')$, by Markov's inequality we get that, conditioned on B_1, B_2 not happening, with probability at least 0.99 over $S' \sim \mathbf{S_n}$:

$$H(x_{S'} \mid \pi(x,\rho),\rho,S') \geq s - \frac{1}{100}$$

Let B_3 be the event (over $S' \sim \mathbf{S_n}$) that this inequality does not hold. Let B be the event that either of B_1, B_2, B_3 happens (i.e., $B = B_1 \cup B_2 \cup B_3$), and let $\bar{B}$ be the complement of the event B.

We get that conditioned on $\bar{B}$, $H(x_{S'} \mid \pi(x,\rho),\rho,S') \geq s - \frac{1}{100}$. From Lemma 4.8 we get that, conditioned on $\bar{B}$, the (statistical) distance of the distribution of $(x,x)_S = x_{S'}$ conditioned on $\pi(x,\rho),\rho,S'$

[20]Note that Shearer's Lemma can be applied to conditional random variables.

from the uniform distribution over $\{0,1\}^s$ is $\sqrt{1/200} < 0.08$. Since each of B_1, B_2, B_3 happens with probability of at most 0.01, from the union bound B happens with probability at most 0.03.

From the triangle inequality, since if B does not happen the distribution of x_S is 0.08-close to uniform, and B happens with probability at most 0.03, we have that the distribution of $(\rho, \pi(x,\rho), S, (x,x)_S)$ is 0.11-close to the distribution $(\rho, \pi(w,\rho), S, (x,y)_S)$.

To complete the proof, recall that if the verifier rejects every input that is 0.1 far from EQU with probability at least 2/3 (where the probability is over the verifier's randomness, i.e. its samples and public coins), it must reject an input sampled from D_{NO} with probability at least 1/2 (where the probability is over the choice of the input and the verifier's randomness). But now, since the distribution of the view of the verifier in the D_{NO} case is 0.11-close to its view in the D_{YES} case, it must reject an input sampled from D_{YES} with probability at least $1/2 - 0.11 > 0.3$, which is a contradiction to the completeness of the SIPP. $\square$

5 A Reduction from SIPPs to IPPs

Consider a family of languages, and assume all of the languages in it have (query-based) IPPs with sub-linear query and communication complexities. Do all of the languages in it also have *sample-based* IPPs with sub-linear sample and communication complexity? In this section we answer the question in the affirmative (under some assumptions on the family, and on the IPPs). Towards this end, we construct a generic reduction from sample-based IPPs to query-based IPPs.

Before stating the main theorem of this section, we introduce the following definition.

Definition 5.1. *A family $\mathcal{L}$ of languages is* closed under composition with log-space uniform $\mathcal{NC}^1$ circuits*, if the following condition holds: For every language $L : \{0,1\}^* \to \{0,1\}$ in $\mathcal{L}$ (which might be a pair language or not) and every function $g : \{0,1\}^* \to \{0,1\}^*$ that can be computed by an ensemble of log-space uniform $\mathcal{NC}^1$ circuits, the composition $L(g(\cdot)) : \{0,1\}^* \to \{0,1\}$ is in $\mathcal{L}$.*

Note that, for example, the family of languages that can be computed by a log-space Turing Machine, and log-space uniform $\mathcal{NC}^i$ (for $i \geq 1$) are families of languages that are closed under composition with log-space uniform $\mathcal{NC}^1$ circuits.

We now state the main theorem of this section:

Theorem 5.2. *Let $\mathcal{L}$ be a family of pair languages that is closed under composition with log-space uniform $\mathcal{NC}^1$ circuits. Assume that there are functions $q, c, v, d : \mathbb{N} \to \mathbb{N}$ and a real value $\epsilon > 0$ such that every $L \in \mathcal{L}$ has a (query-based) IPP with proximity parameter ϵ, with the following properties:*

1. *The protocol uses only public-coins.*
2. *The query complexity of the protocol is $O(q(n))$, its communication complexity is $O(c(n))$, its randomness complexity (the number of public coins the verifier tosses) is $r = r(n)$, the verifier running time is $O(v(n))$, and its round complexity is poly(d). Also assume (w.l.o.g) that the verifier tosses exactly $r(n)$ coins.*
3. *The indices of the queries depend only on the public coins of the verifier. That is, for each $L \in \mathcal{L}$, there is a deterministic function that maps the public coin tosses of the verifier to the indices it queries in the input. Also assume (w.l.o.g) that the verifier queries the input only at the end of the IPP, after the interaction with the prover ends.*
4. *The (honest) prover runs in time poly(n).*

Then every $L \in \mathcal{L}$ has a sample-based *IPP with proximity parameter* 120ϵ*. The sample complexity of the protocol is $\widetilde{O}(q(n))$, its communication complexity is $\widetilde{O}(c(n)+q(n))$, the verifier running time is $\widetilde{O}(v(n)+q(n))$, its round complexity is poly(d),*[21]*, it is a private coin protocol, and it has imperfect completeness. In addition, the (honest) prover runs in time poly(n).*

Remark 5.3 (Imperfect completeness of the sample-based protocol)**.** *The sample-based IPP achieved in Theorem 5.2 does not have perfect completeness, even if the query-based IPP we start with does have prefect completeness. It is possible to slightly change the reduction in order to get a sample-based IPP with perfect completeness. The main cost is increasing the round complexity. See Remark 5.25.*

We use this reduction to prove that rich families of languages have SIPPs with sub-linear sample and communication complexities. The first family we consider is log-space uniform $\mathcal{NC}$. Rothblum, Vadhan and Wigderson [RVW13] and Rothblum and Rothblum [RR20] showed that any language in this family has a sub-linear (query-based) IPP. By applying the reduction of Theorem 5.2 on their result we prove the following:

Theorem 5.4. *Let L be a language in log-space uniform $\mathcal{NC}$, and let $\epsilon \in (0,1)$ be a fixed constant. Then there is a SIPP with proximity parameter ϵ for L. The sample and communication complexities of the SIPP, as well as the verifier's running time, are $\widetilde{O}(\sqrt{n})$. In addition, the SIPP has poly$(\log n)$ rounds, and the (honest) prover runs in time poly(n).*

Reingold, Rothblum and Rothblum [RRR16] proved that every language that can be computed in polynomial-time and bounded-polynomial space has a sub-linear constant-round IPP. We apply our reduction on their result, and show that every such language also has a sub-linear constant-round *sample-based* IPP:

Theorem 5.5. *Fix a constant $\sigma \in (0,1)$, and let L be a language that is computable in poly(n)-time and $O(n^{\sigma})$-space. Let $\epsilon \in (0,1)$ be a fixed constant. Then there is a SIPP with proximity parameter ϵ for L. The sample and communication complexities of the SIPP, as well as the verifier's running time, are $n^{1/2+O(\sigma)}$. In addition, the SIPP has a constant number of rounds, and the (honest) prover runs in time poly(n).*

5.1 The Reduction

General Structure of the Reduction: Given some language L in $\mathcal{L}$, the hypothesis of Theorem 5.2 ensures that there exists an interactive proof of proximity for testing if x is in L or ϵ-far from L. One would want to use this IPP directly as a SIPP for testing the same property in the sample-based access model, but there is a clear problem with this idea. At the end of the IPP, the verifier queries x at some indices, which depend on the run of the IPP (namely, on the coins that the verifier tossed during the execution of the protocol). But the verifier in a SIPP cannot query x at these indices, since it has only sample-based access to the input. It can only sample x at some uniformly random (and i.i.d) indices, which do not depend on the execution of the protocol.

To overcome this obstacle, one might suggest the following solution. First, the verifier draws samples from the input, at uniformly random and i.i.d indices S. The verifier then computes ("simulates") which coin tosses for the IPP *would have* caused it to query the input at *exactly* the indices S (note that from the hypothesis, the indices queried by the verifier in the IPP depend solely on the verifier public coins). The verifier then executes the protocol using the coin tosses it computed instead of using fresh randomness. At the end of the protocol, when the verifier needs to query the input at

[21]We note that the reduction increases the number of rounds by 1. Namely, the number of rounds of the sample-based IPP for some $L \in \mathcal{L}$ is $d'+1$, where d' is the number of rounds of the query-based IPP of a language in $\mathcal{L}$.

some indices, these indices are exactly S - and therefore the verifier already has the value of the input at these indices.

This idea seems promising, but it has a major flaw. First, from the hypothesis, there is a mapping between the verifier's public coins to query indices, but this mapping might not be onto. There might be a set of indices such that no coin tosses would have caused the verifier to query these indices. As the indices in S are chosen uniformly at random and i.i.d, S might be such a set of indices, and the verifier will not be able to "simulate" the required coin tosses, since none exist.

The issue is actually more acute; even if the mapping is onto, the *distribution* of the query indices needed for the IPP might be different from the distribution of S. In particular, the query indices for an IPP are typically *not* independent - they usually have some structure that is based on the IPP.[22] Therefore, if the verifier uses that mapping to simulate coin tosses, the distribution of these coin tosses will not be uniformly random. Using such coin tosses for an IPP might destroy its soundness.

To solve this obstacle we use the following key idea. Instead of running the IPP directly on the given input x, the verifier executes an IPP on a *transformed* version of x, denoted by $f(x)$ and with respect to a "transformed language" $f(L)$. The transformation function f is chosen in way that ensures it has some important properties: f is distance preserving (in a way that will be formalized later), and random samples on x are "mapped to" structured queries in $f(x)$. We continue with more details on the protocol and the transformation.

The Protocol: The transformation of the input is represented as a function $f : \{0,1\}^n \to \{0,1\}^{m\cdot n}$, where $m = O(\log n)$. Denote by $\mathcal{F}$ the family of all the transformation functions that can be used for the protocol. We index the functions in $\mathcal{F}$ by keys from a set $KEYS$ (which will be defined formally later). Each key is a short binary string that describes / encodes the function $f_{key} \in \mathcal{F}$, and $\mathcal{F} = \{f_{key}\}_{key \in KEYS}$. The key of a function has some additional properties that will be described later in this section.

For a language[23] L define the pair language $L' = \{(key, f_{key}(x)) \mid key \in KEYS, x \in L\}$, where key is the explicit part of the input, and $f_{key}(x)$ is the implicit part of the input. $KEYS$ and f_{key} are chosen in a way that ensures that if $L \in \mathcal{L}$ then $L' \in \mathcal{L}$, and therefore there exists a (query access) IPP for L'.

The protocol between a verifier and a prover for testing if $x \in \{0,1\}^n$ is in L or 120ϵ-far from L is described formally in Figure 5. We next give a high-level description of the protocol.

First, the verifier draws $m \cdot k = \widetilde{O}(q(n))$ samples from x, where $k = \log n \cdot q(n)$.[24] Denote their indices by $S \in [n]^{m\cdot k}$. Next, it "simulates" privately the coin tosses that are required for the execution of the query-based IPP to prove a claim of the form $(key, f_{key}(x)) \in L'$, for some $key \in KEYS$. Note that key and the transformation function f_{key} were not chosen yet. However, by the hypothesis of Theorem 5.2 (Property 2) the verifier knows in advance the number of coins $r(n)$ that are needed for the execution of the IPP for L'. Based on the coin tosses, the verifier computes the indices it will need to query at the end of that IPP. Denote those indices by $Q \in [m \cdot n]^{\{k\}}$.[25] Again, from the hypothesis of the Theorem (Property 3), the indices Q depend only on the coin tosses (and independent of the explicit and implicit inputs), and therefore they can be computed even though f_{key} has not been chosen yet.

Next, (and only now), the verifier chooses the transformation function $f_{key} \in \mathcal{F}$, based on S and

[22]In particular, in the IPPs from [RVW13, RR20] that are used to prove Theorem 5.4, the query indices are not independent of each other.

[23]For the intuitive description of the protocol we assume that L is a standard (not a pair) language. In the actual application of the protocol, L itself would be a pair language.

[24]Recall that $q(n)$ is a function such that all languages in $\mathcal{L}$ have (query-based) IPP with query complexity $O(q(n))$.

[25]Note we can assume w.l.o.g that the query indices are distinct (that is, the protocol does not query the input on the same index twice). We use tuples of distinct elements here instead of sets for technical reasons.

Q. The key of the chosen function has some important properties: First, having *key* allows efficient computation of $f_{key}(x)$ for any $x \in \{0,1\}^n$. In other words, the key of the function f_{key} describes the operation of the function. Second, the key is computed as a function of Q and S, but (when S is chosen i.i.d and uniformly at random) the key does not reveal any information about Q. Third, f_{key} is constructed in a way that allows recovering the value of $f_{key}(x)$ at the indices Q from the value of x at the indices S. More specifically, each one of the bits of $f_{key}(x)$ at the indices Q equals to some bit of x at an index from the set S. The exact construction of f_{key} and the formulation of the properties are explained below.

Now, as its first message, the verifier sends the key of the function f_{key} to the prover. Next, the verifier and the prover interact and execute the IPP for the statement $(key, f_{key}(x)) \in L'$, where $key \in KEYS$ is an explicit input (both the verifier and the prover have direct access to it), and $f_{key}(x)$ is the implicit part of the input. While executing the protocol, instead of sending fresh public coins to the prover, the verifier uses the same coins it tossed for the computation of Q.

After the interaction ends, the verifier has to query the transformed input $f_{key}(x)$ at indices Q. But now, from a property of f_{key} that was described above, the verifier can recover the value of $f_{key}(x)$ at the indices Q from the value of x at the indices S. As the verifier already has the value of x at the indices S (from the samples it drew), it can use these values instead of querying $f_{key}(x)$ directly. To finish the protocol, the verifier now accepts or rejects x according to the instructions induced by the IPP it executed for $(key, f_{key}(x)) \in L'$.

Note that, in contrast to the "flawed solution" described above, here the IPP is executed with i.i.d random coins - the coins that the verifier tossed privately at the beginning of the protocol. In addition, since the key sent to the prover does not reveal any information regarding Q, it also does not reveal any information regarding these coin tosses. Therefore, from the prover's perspective, these coins seem like they were generated by fresh randomness, and the soundness of the protocol is maintained.

5.2 Construction of the Transformation Function

We encapsulate the requirements from the transformation function f in the following Lemma:

Lemma 5.6. *For all integers $k \le n$ and t, such that n is a power of 2, taking $m = t \cdot \log n$, there exist:*

1. *A procedure $K : [n]^{m\cdot k} \times [m \cdot n]^{\{k\}} \to KEYS$, where $KEYS = \{0,1\}^{m\cdot k\cdot \log n}$, that maps a pair S, Q to a key $K(S,Q)$. The runtime of K is $k \cdot poly(\log k, \log n, t)$.*
2. *A procedure $F : KEYS \times \{0,1\}^n \to \{0,1\}^{m\cdot n}$ that maps a key and an input $x \in \{0,1\}^n$ to an output in $\{0,1\}^{m\cdot n}$. The procedure F can be computed in $poly(n)$ time.*

   Let $f_{key} : \{0,1\}^n \to \{0,1\}^{m\cdot n}$ be the function $F(key, \cdot)$ (that is, the application of F with a fixed key). Let $\mathcal{F} = \{f_{key}\}_{key \in KEYS}$.
3. *A recovery procedure $R : KEYS \times [m \cdot n] \to [n]$.*

   Let $R_{key} : [m \cdot n] \to [n]$ be the function $R(key, \cdot)$ (that is, the application of R with a fixed key).

with the following properties:

1. *(Mapping x and $f(x)$) For every $x \in \{0,1\}^n$, index $q \in [m \cdot n]$ and $key \in KEYS$, the value of $[f_{key}(x)]_q$ depends on a single bit of x. Moreover, $R_{key}(q)$ describes which bit of x it depends on. Namely:*

$$[f_{key}(x)]_q = x_{R_{key}(q)}$$

   Note that R_{key} is a function of the index q, and does not depend on the value of x.

2. *(Simulating Queries) For every $S \in [n]^{m \cdot k}, Q \in [m \cdot n]^{\{k\}}$, and every $q \in Q$:*

$$R_{K(S,Q)}(q) \in S$$

(That is, the value of $f_{K(S,Q)}(x)$ at indices Q can be recovered from the value of x at indices S, for every $x \in \{0,1\}^n$).

3. *(Distance Preserving) For every $Q \in [m \cdot n]^{\{k\}}$ and every $\epsilon > 0$, if x is 60ϵ-far from L then:* [26]

$$\Pr_{S \sim \mathbf{S}, key \leftarrow K(S,Q)}[f_{key}(x) \text{ is } \epsilon\text{-far from } \{f_{key}(x') \mid x' \in L\}] \geq 1 - n^{-\Omega(t)}$$

where $\mathbf{S}$ is the uniform distribution on $(m \cdot k)$-tuples of elements from $[n]$.

4. *(Q-Hiding) The key $K(S,Q)$ does not reveal Q. That is, for every $Q, Q' \in [m \cdot n]^{\{k\}}$ and every $key \in KEYS$:*

$$\Pr_{S \sim \mathbf{S}}[K(S,Q) = key] = \Pr_{S \sim \mathbf{S}}[K(S,Q') = key]$$

(Where $\mathbf{S}$ is the uniform distribution on $(m \cdot k)$-tuples of elements from $[n]$, as in Property 3).

5. *(Circuit Reversing F) With high probability over the choice of key, the function f_{key} is injective:*

$$\Pr_{S \sim \mathbf{S}, key \leftarrow K(S,Q)}[\exists x, x' \in \{0,1\}^n \text{ such that } x \neq x' \text{ and } f_{key}(x) = f_{key}(x')] \leq n^{-\Omega(t)}$$

(Where $\mathbf{S}$ is the uniform distribution on $(m \cdot k)$-tuples of elements from $[n]$, as in Property 3).

Moreover, there exists a logspace-uniform $\mathcal{NC}^1$ circuit $\mathcal{C}$ that gets as input $key \in KEYS$ and $y = F(key, x) \in \{0,1\}^{m \cdot n}$, and if f_{key} is injective, outputs x.

6. *(Efficient Recovery) There exists a TM that gets as input $key \in KEYS$ and $q_1, \ldots, q_k \in [m \cdot n]$ and outputs $R(key, q_1), \ldots, R(key, q_k) \in [n]$, and its runtime is $k \cdot poly(\log k, \log n, t)$.*

We now describe the construction of the K, F and R functions described in Lemma 5.6. The construction uses a family of k-wise independent hash functions.

k-wise Independent Hash Functions: We begin by defining the notion of k-wise independent hash functions.

Definition 5.7. *(k-wise independent hash functions) For $n, m, k \in \mathbb{N}$ such that $k \leq n$, a family of functions $\mathcal{H} = \{h : [n] \to [m]\}$ is* k-wise independent *if for all distinct $x_1, x_2, \ldots, x_k \in [n]$, and for $h \sim \mathcal{H}$, the random variables $h(x_1), \ldots, h(x_k)$ are independent and uniformly distributed in $[m]$.*

Note that our construction only uses the case where $m = n$.

We now present a known construction for a family of functions with this property.

Theorem 5.8. *(k-wise independent hash functions from polynomials) Let $\mathbb{F}$ be a finite field. For $a_0, a_1, \ldots, a_{k-1} \in \mathbb{F}$ define $h_{a_0,a_1,\ldots,a_{k-1}} : \mathbb{F} \to \mathbb{F}$ by $h_{a_0,a_1,\ldots,a_{k-1}}(x) = a_0 + a_1 x + a_2 x^2_+ \cdots + a_{k-1} x^{k-1}$.*

Let $\mathcal{H} = \{h_{a_0,a_1,\ldots,a_{k-1}} \mid a_0, a_1, \ldots, a_{k-1} \in \mathbb{F}\}$. Then $\mathcal{H}$, which is the set of all polynomial of degree at most $k-1$ over $\mathbb{F}$, is a family of k-wise independent hash functions from $\mathbb{F}$ to $\mathbb{F}$.

[26]Note that here x is assumed to be 60ϵ-far from L, though in Theorem 5.2 the proximity parameter is 120ϵ. The reason is that here we assume n is a power of 2, whereas in the main Theorem the implicit input length n might not be so. In order to apply the Lemma during the proof of the main Theorem we use padding, which might damage the proximity by a factor of at most 2.

For Lemma 5.6 we assume that n is a power of 2. We take $\mathcal{H}$ to be the family of hash functions from Theorem 5.8, where $\mathbb{F}$ is taken to be the finite field of size n. For this family of hash functions we have the following Fact:

Fact 5.9. *There is a Turing Machine that takes as input an encoding $\langle h \rangle \in \{0,1\}^{k \cdot \log n}$ of a hash function $h \in \mathcal{H}$ and $i \in [n]$, and outputs $h(i)$. (Namely, for $h = h_{a_0,\ldots,a_{k-1}}$ the encoding $\langle h \rangle$ is the concatenation of the canonical encodings of $a_0, \ldots, a_{k-1} \in [n]$).*

The runtime of this TM is $k \cdot poly(\log n)$. Moreover, there is a logspace-uniform ensemble of circuits of size $n^{O(1)}$ and depth $O(\log n)$ that takes as input $\langle h \rangle \in \{0,1\}^{k \cdot \log n}$ and $i \in [n]$ and outputs $h(i)$.

Proof. The TM that gets as inputs $\langle h \rangle = \langle a_0 \rangle, \ldots, \langle a_{k-1} \rangle$ and $i \in [n]$ and outputs $h(i)$ simply evaluates $h(i)$ in a straight-forward manner. That it, it computes i^j for any $0 \leq j < k$, and then computes the sum $\sum_{j=0}^{k-1} a_j \cdot i^j$, where all of the arithmetic operations are over the field $\mathbb{F}$. These computations requires $O(k)$ arithmetic operations over a field of size n, and therefore the runtime of the TM is $k \cdot \text{poly}(\log n)$.

The circuit that gets the same inputs, and has the same output, works similarly to the TM. Namely, its inputs are $\langle h \rangle = \langle a_0 \rangle, \ldots, \langle a_{k-1} \rangle \in \{0,1\}^{k \cdot \log n}$ and $i \in [n]$, and its output is $h(i) \in [n]$. The circuit is constructed as follows.

It first computes in parallel i^j for any $0 \leq j < k$, in k sub-circuits. This can be done by pre-computing all of the values α^j for every $\alpha \in [n]$ and $0 \leq j < k$, and keeping them in a lookup-table. The circuits for the look-up are of depth $\log k \cdot \text{poly}(\log \log n)$ and size $k \cdot \text{poly}(\log n) = n^{O(1)}$.

It then combines the outputs of the sub-circuits with the inputs $\langle a_0 \rangle, \ldots, \langle a_{k-1} \rangle$ to compute $\sum_{j=0}^{k-1} a_j \cdot i^j$. This computation can be done in a circuit of depth $O(\log n)$ and size $n^{O(1)}$. Multiplication of two $\log n$ bits number can be done in such a circuit in a straight-forward fashion (indeed, it can be done in a circuit of depth $O(\log \log n)$.) Summation of n numbers, each of $\log n$ bits, can also be done in such a circuit. See Theorem 1.20 in the book of Vollmer on Circuit Complexity [Vol99].

In total, the depth of the circuit that computes $h(i)$ is $O(\log n)$, and its size is $n^{O(1)}$. $\square$

In addition, we have the following facts regarding $\mathcal{H}$.

Fact 5.10. *Let $\mathbb{F}, \mathcal{H}$ be as in Theorem 5.8 where $\mathbb{F}$ is taken to be the finite field of size n. Then for any two k-tuples of elements from $\mathbb{F}$, denoted A, B, such that the elements in A are distinct, there exists a unique $h \in \mathcal{H}$ such that $h(A) = B$. Moreover, there exists TM that gets as input A, B and outputs the (canonical) encoding of h, $\langle h \rangle$, such that $h(A) = B$ in time $k \cdot poly(\log k, \log n)$, by using the (inverse) Fast Fourier Transform (FFT).*

Fact 5.11. *Let $\mathbb{F}, \mathcal{H}$ be as in Theorem 5.8 where $\mathbb{F}$ is taken to be the finite field of size n. There exists a TM that gets as input a k-tuple A and an encoding of $h \in \mathcal{H}$, $\langle h \rangle$, and outputs the k-tuple $B = h(A)$ in time $k \cdot poly(\log k, \log n)$, by using the Fast Fourier Transform (FFT).*

For a detailed proof of facts 5.10 and 5.11, and efficient implementation of FFT see chapter 30 in the classic book of Coremen, Leiserson, Rivest and Stein [CLRS09]. (In particular, see Theorem 30.1 for the existence proof, and sections 30.2 and 30.3 for discussion and implementation of the Discrete Fourier Transform (DFT)).

The Construction: We now show the construction of K, F and R for Lemma 5.6.

For $S \in [n]^{m \cdot k}, Q \in [m \cdot n]^{\{k\}}$, we need to specify the key $K(S,Q)$, the function $f = f_{K(S,Q)} \in \mathcal{F}$ and the function $R_{K(S,Q)} : [m \cdot n] \to [n]$ that satisfy the list of properties in the Lemma statement. In Figure 4 we give a formal description of the construction, and continue with more details and intuition.

Figure 4 Construction of K, F, R
Input: $S \in [n]^{m \cdot k}$ (samples indices) and $Q \in [m \cdot n]^{\{k\}}$ (query indices)
Output: A key $K(S,Q) \in \mathit{KEYS}$, a function $f_{K(S,Q)} \in \mathcal{F}$ and a function $R_{K(S,Q)} : [m \cdot n] \to [n]$.

```
1. Let $S_1, \ldots, S_m \in [n]^k$ be the $k$-tuples such that $S = (S_1, \ldots, S_m) \in [n]^{m \cdot k}$.
2. Let $Q_1, \ldots, Q_m \in [n]^{\{k\}}$ be $k$-tuples of distinct elements from $[n]$, such that for every $1 \leq j \leq m$,
   $Q_j$ contains all of the elements from $Q$ that are in the range $I_j = \{(j-1) \cdot n + 1, \ldots, j \cdot n\}$
   (adding dummy elements as needed).
3. For $j$ in $1 \ldots m$:
   (a) Choose $h_j$ to be the unique function in $\mathcal{H}$ such that $h_j(Q_j) = S_j$.
4. Let $K(S,Q) \leftarrow \langle h_1 \rangle \circ \cdots \circ \langle h_m \rangle$.
5. For $j$ in $1 \ldots m$:
   (a) Define $f_j : \{0,1\}^n \to \{0,1\}^n$ by $[f_j(x)]_i = x_{h_j(i)}$ for any $i \in [n]$.
6. Let $F(K(S,Q), x) = f_{K(S,Q)}(x) = f_1(x) \circ f_2(x) \circ \cdots \circ f_m(x)$.
7. For $q \in [m \cdot n]$, define $R_{K(S,Q)}(q)$ as follows:
   (a) Let $j \in [m]$ be the index such that $q \in I_j$.
   (b) Let $i = q - (j-1) \cdot n \in [n]$.
   (c) Set $R_{K(S,Q)}(q) = h_j(i)$.
```

Construction of K: The first steps are to create from the tuples $S \in [n]^{m \cdot k}$ and $Q \in [m \cdot n]^{\{k\}}$ smaller tuples. From each of the two tuples S, Q, we create m smaller tuples, where each smaller tuple is of length k, and contains only elements from $[n]$. First, S is a tuple of length $m \cdot k$, and it is divided into m tuples, $S_1, \dots, S_m$, each of length k, in the natural way. That is, S_1 is the tuple that contains the first k elements of S (according to the order in S), S_2 is the tuple that contains the k next elements of S, etc. Formally, if $S = (s_1, s_2, \dots, s_k, s_{k+1}, \dots, s_{km})$ then for each $1 \leq j \leq m$ let $S_j = (s_{k(j-1)+1}, \dots, s_{k \cdot j})$.

The process to create $Q_1, \dots, Q_m \in [n]^{\{k\}}$ from $Q \in [m \cdot n]^{\{k\}}$ is different. Instead of separating the elements of the tuple according to their location (as done with S), the separation is done according to their values. Recall that Q is a tuple of length k of (distinct) elements from the set $[m \cdot n]$. The set $[m \cdot n]$ is considered as m disjoint ranges, each of length n, in the natural way. That is, let $I_j = \{(j-1) \cdot n + 1, \dots, j \cdot n\}$, and observe that indeed $\bigcup_{j=1}^{m} I_j = [m \cdot n]$ and $I_j, I_{j'}$ are disjoint for $j \neq j'$ (and $|I_j| = n$). Now, let Q'_j be the restriction of Q to the range I_j. That is, Q'_j is the tuple which is created by taking Q and removing from it all of the elements that are not in I_j, while respecting the order from Q. Note that the lengths of the tuples might be different from each other, but because the length of Q is k, each tuple is of length at most k. To make all of the tuples have the same length k, complete each Q'_j by adding "dummy" elements to it. Namely, if the length of Q'_j is $l < k$, take the first $k - l$ elements of I_j which are not in Q'_j, and add them to Q'_j. After adding the dummy elements, every Q'_j is a tuple of length k of distinct elements from the range I_j. The process is still not done, as each small tuple should contain only elements from $[n]$ (this is needed for the next step of the construction). In order to do that, define $Q_j \in [n]^{\{k\}}$ by $[Q_j]_u = [Q'_j]_u - (j-1) \cdot n$ for every $1 \leq u \leq k$. That is, for $j = 1$ we have $Q_1 = Q'_1 \in [n]^{\{k\}}$ (as $I_1 = [n]$), and for $2 \leq j \leq m$ we decrease the value of each element of Q'_j by the value of the last element of I_{j-1}. Note that since the elements of Q are distinct, for every $1 \leq j \leq m$ the elements of Q_j are distinct. Also note that it even though the elements of Q_j are from $[n]$, it is "easy" to take an index $q \in [m \cdot n]$ and compute to which tuple Q_j it would have gone, and what would be its "new value" there (i.e, this can be computed by a logspace-uniform $\mathcal{NC}^1$ ensemble of circuits). We say that the smaller tuples $Q_1, \dots, Q_m$ are *induced* by Q, and note that the process to create $Q_1, \dots, Q_m$ from Q is deterministic, and can be computed in a straight-forward fashion in time $k \cdot m \cdot \text{poly}(\log n) = k \cdot \text{poly}(\log n, t)$ (as it involves at most $O(k \cdot m)$ elementary operations on elements from $[n]$).

Next, for each $1 \leq j \leq m$, choose h_j to be the unique function in $\mathcal{H}$ such that $h_j(Q_j) = S_j$. Note that such a function exists and it is unique from Fact 5.10.

Take the key $K(S, Q)$ to be the concatenation $\langle h_1 \rangle \circ \dots \circ \langle h_m \rangle \in \{0,1\}^{m \cdot k \cdot \log n} = \mathit{KEYS}$, where $\langle h \rangle \in \{0,1\}^{k \cdot \log n}$ is the encoding of h from Fact 5.9. Note that from Fact 5.10, each encoding $\langle h_j \rangle$ can be computed in time $k \cdot \text{poly}(\log n)$ from S_j, Q_j. Together with the run-time of computing all of the tuples S_j, Q_j from S, Q, we get that computing $K(S, Q)$ can be done in time $k \cdot \text{poly}(\log n, t)$. In particular, the verifier can compute (privately) the key $K(S, Q)$ in time that is sub-linear in n.

Construction of F and R: We next describe the construction of F and R from $K(S, Q) = \langle h_1 \rangle \circ \dots \circ \langle h_m \rangle$. Note that this process is "public"; it can be done based on information that is known to both the verifier and the prover.[27]

First, construct m functions, $f_1, \dots, f_m : \{0,1\}^n \to \{0,1\}^n$ from $h_1, \dots, h_m$ as follows.

For $x \in \{0,1\}^n, i \in [n], 1 \leq j \leq m$ define $f_j(x) \in \{0,1\}^n$ by:

$$[f_j(x)]_i = x_{h_j(i)}$$

Finally, let $\mathit{key} = K(S, Q)$ and take $F(\mathit{key}, x) \in \{0,1\}^{m \cdot n}$ to be $F(\mathit{key}, x) = f_{\mathit{key}}(x) = f_1(x) \circ f_2(x) \circ \dots \circ f_m(x)$. The function $R(\mathit{key}, \cdot) = R_{\mathit{key}}$ is implied directly from the construction of f.

[27]In contrast, computing $K(S, Q)$ is done privately by the verifier, as it keeps S and Q secret.

Namely, for $q \in [m \cdot n]$ let $j \in [m]$ be the index such that $q \in I_j$. Let $i = q - (j-1) \cdot n$ (that is, i is the "relative index" of q in I_j). Note that $i \in [n]$. Now, observe that $[f_{key}(x)]_q = [f_j(x)]_i = x_{h_j(i)}$ and define $R_{key}(q) = h_j(i)$.

Proofs for Lemma 5.6: We now prove that K, F and R as described above indeed satisfy the requirements of Lemma 5.6. We first observe that Property 1 of Lemma 5.6 holds directly from the construction of R. Namely, for every $key \in KEYS$, the function R_{key} was constructed in a way such that for every $q \in [m \cdot n]$ and every $x \in \{0,1\}^n$:

$$[f_{key}(x)]_q = x_{R_{key}(q)}$$

We next observe that the value of $F(key, x)$ can be computed efficiently.

Claim 5.12. *There exists a TM that gets as input $key \in KEYS, x \in \{0,1\}^n$ and outputs $F(key, x)$, and runs in time poly(n).*

Proof. We show that every bit of $F(key, x)$ can be computed in time poly(n). This is enough, since $F(key, x)$ has poly(n) bits.

For $key \in KEYS$, let $key = \langle h_1 \rangle \circ \cdots \circ \langle h_m \rangle$, and for every $1 \leq m$ let f_j be the function constructed at step 5 of Figure 4 from h_j. For every $i \in [n]$ and $1 \leq j \leq m$, we have by definition $[f_j(x)]_i = x_{h_j(i)}$. From Fact 5.9, $h_j(i)$ can be computed in poly(n) time, and hence $x_{h_j(i)}$ can be computed in poly(n) time.

As $F(key, x) = f_1(x) \circ f_2(x) \circ \cdots \circ f_m(x)$, we are done. $\square$

We next show that Property 2 of Lemma 5.6 holds.

Claim 5.13. *(Property 2 - Simulating Queries) For every $S \in [n]^{m \cdot k}, Q \in [m \cdot n]^{\{k\}}$, and every $q \in Q$:*

$$R_{K(S,Q)}(q) \in S$$

(That is, the value of $f_{K(S,Q)}(x)$ at indices Q can be recovered from the value of x at indices S, for every $x \in \{0,1\}^n$).

Proof. Let $S \in [n]^{m \cdot k}, Q \in [m \cdot n]^{\{k\}}$, and $q \in Q$. The proof follows from the construction of R and from the construction of the tuples Q_j from Q.

Namely, let $S \in [n]^{m \cdot k}, Q \in [m \cdot n]^{\{k\}}, q \in Q$ and let $key = K(S, Q)$. Let $j \in [m]$ be such that $q \in I_j$, and let $i = q - (j-1) \cdot n$. From the choice of R_{key} we have $R_{key}(q) = h_j(i)$. From the choice of h_j, we have $h_j(Q_j) = S_j \subseteq S$. From the construction of Q_j and since $q \in Q$, we have $i \in Q_j$. Therefore, $R_{key}(q) = h_j(i) \in S$ as required. $\square$

We now make a claim that describes the distributions of the hash functions $h_1, \ldots, h_m$ which are chosen during the construction of $K(Q, S)$. This claim will be used to prove a few of the properties of Lemma 5.6

Claim 5.14. *Fix $Q \in [m \cdot n]^{\{k\}}$, and let $S \sim \mathbf{S}$ where $\mathbf{S}$ is the uniform distribution on $(m \cdot k)$-tuples of elements from $[n]$. Let $h_1, \ldots, h_m \in \mathcal{H}$ be the hash functions that are chosen by the construction of $K(Q, S)$. Then for every $1 \leq j \leq m$ the distribution of h_j is uniform over $\mathcal{H}$. Moreover, $h_1, \ldots, h_m$ are independent of each other. That is, the distribution of $h_1, \ldots, h_m$ is as if they were chosen uniformly at random and i.i.d from $\mathcal{H}$.*

Proof. Let $Q_1, \dots, Q_m \in [n]^{\{k\}}$ be the smaller tuples induced by Q in the construction, and let $S_1, \dots, S_m \in [n]^k$ be the smaller tuples induced by S in the construction. Fix $1 \le j \le m$.

To show that the distribution of h_j is uniform over $\mathcal{H}$ we need to show that for every fixed $h \in \mathcal{H}$:

$$\Pr_{S \sim \mathbf{S}}[h_j = h] = \frac{1}{|\mathcal{H}|} = \frac{1}{n^k}$$

We used the fact that $|\mathcal{H}| = n^k$, which is immediate from the choice of $\mathcal{H}$.

Indeed, fix $h \in \mathcal{H}$. Observe that since the k elements in Q_j are distinct, and from Fact 5.10, $h_j = h$ iff $h_j(Q_j) = h(Q_j)$. (One direction is trivial, and the second direction holds since two polynomials of degree at most $k-1$ that agree on k distinct points are equal to each other). But now, observe that from the construction, $h_j(Q_j) = S_j$ (this is how h_j was chosen). Therefore:

$$\Pr_{S \sim \mathbf{S}}[h_j = h] = \Pr_{S \sim \mathbf{S}}[h_j(Q_j) = h(Q_j)] = \Pr_{S \sim \mathbf{S}}[S_j = h(Q_j)].$$

Now, observe that from the construction, when $S \sim \mathbf{S}$ the induced S_j is distributed uniformly from $[n]^k$. (Namely, each of the k elements of S_j was sampled uniformly at random and independently from $[n]$). That is, for any fixed k-tuple $S' \in [n]^k$, we have $\Pr_{S \sim \mathbf{S}}[S_j = S'] = \frac{1}{|[n]^k|} = \frac{1}{n^k}$. Note that $h(Q_j)$ is some fixed tuple in $[n]^k$, hence $\Pr_{S \sim \mathbf{S}}[S_j = h_j(Q_j)] = \frac{1}{n^k}$, and therefore also $\Pr_{S \sim \mathbf{S}}[h_j = h] = \frac{1}{n^k}$ as required.

For the "moreover" part, observe that when $S \sim \mathbf{S}$ the induced smaller tuples $S_1, \dots, S_m$ are independent of each other (since each S_j contains different samples, and the samples were drawn i.i.d), and therefore $h_1, \dots, h_m$ are independent of each other. □

We now show that Property 4 of Lemma 5.6 holds.

Claim 5.15. *(Property 4 - Q-Hiding) The key $K(S,Q)$ does not reveal Q.*
That is, for every $Q, Q' \in [m \cdot n]^{\{k\}}$ and every key $\in$ KEYS:

$$\Pr_{S \sim \mathbf{S}}[K(S,Q) = \textit{key}] = \Pr_{S \sim \mathbf{S}}[K(S,Q') = \textit{key}]$$

where $\mathbf{S}$ is the uniform distribution on $(m \cdot k)$-tuples from $[n]$.

Proof. Let $Q, Q' \in [m \cdot n]^{\{k\}}$ and let $\textit{key} \in \textit{KEYS}$. Assume that key is a valid key. That is,

$$\textit{key} = \langle h_1^* \rangle \circ \cdots \circ \langle h_m^* \rangle$$

for some $h_1^*, \dots, h_m^* \in \mathcal{H}$. (otherwise, $\Pr_{S \sim \mathbf{S}}[K(S,Q) = \textit{key}] = \Pr_{S \sim \mathbf{S}}[K(S,Q') = \textit{key}] = 0$ and we are done).

For $S \sim \mathbf{S}$, denote by $h_1, \dots, h_m \in \mathcal{H}$ the hash functions such that:

$$K(S,Q) = \langle h_1 \rangle \circ \cdots \circ \langle h_m \rangle$$

Similarly, for $S \sim \mathbf{S}$, denote by $h'_1, \dots, h'_m \in \mathcal{H}$ the hash functions such that:

$$K(S,Q') = \langle h'_1 \rangle \circ \cdots \circ \langle h'_m \rangle$$

Now, since the encoding $h \to \langle h \rangle$ is an injective function, $K(S,Q) = \textit{key}$ iff for every $1 \le j \le m$ it holds that $h_j = h_j^*$. Similarly, $K(S,Q') = \textit{key}$ iff for every $1 \le j \le m$ it holds that $h'_j = h_j^*$.

Therefore,

$$\Pr[K(S,Q) = \textit{key}] = \Pr[h_1 = h_1^*] \cdot \ldots \cdot \Pr[h_1 = h_m^*]$$

and

$$\Pr[K(S,Q') = \textit{key}] = \Pr[h'_1 = h_1^*] \cdot \ldots \cdot \Pr[h'_1 = h_m^*]$$

where the probabilities are over $S \sim \mathbf{S}$.

But now, from Claim 5.14, for every $1 \leq j \leq m$ we have:

$$\Pr_{S \sim \mathbf{S}}[h_j = h_j^*] = \Pr_{S \sim \mathbf{S}}[h_j' = h_j^*] = \frac{1}{n^k}$$

since h_j^* is some fixed element in $\mathcal{H}$.

Therefore, $\Pr[K(S,Q) = key] = \Pr[K(S,Q') = key]$, which is what we needed to show. $\square$

We now show that Property 3 of Lemma 5.6 holds.

Claim 5.16. *(Property 3 - Distance Preserving) For every $Q \in [m \cdot n]^{\{k\}}$ and every $\epsilon > 0$, if x is 60ϵ-far from L then:*

$$\Pr_{S \sim \mathbf{S}, key \leftarrow K(S,Q)}[f_{key}(x) \text{ is } \epsilon\text{-far from } \{f_{key}(x') \mid x' \in L\}] \geq 1 - n^{-\Omega(t)}$$

where $\mathbf{S}$ is the uniform distribution on $(m \cdot k)$-tuples of elements from $[n]$.

Proof. Intuitively, we want to show that f_{key} preserves distances, with high probability (over the samples the verifier draws). Recall that $f_{key}(x) \in \{0,1\}^{m \cdot n}$ is a concatenation of $f_1(x) \circ \cdots \circ f_m(x)$, and each f_j is constructed from $h_j \in \mathcal{H}$.

For $h \in \mathcal{H}$ and $i \in [n]$ we say that i is *covered* by h if i is in the range of h. That is, h covers i if $i \in h([n])$, or, equivalently, if $h^{-1}(i)$ is not empty. Observe that from the construction of f_j, when h_j covers i there is (at least) one bit of $f_j(x)$ which is equal to x_i. Namely, the value of $f_j(x)$ at each of the indices from the set $h_j^{-1}(i)$ (which contains at least one index when h_j covers i) equals to x_i.

We first claim that for a fixed $i \in [n]$, when drawing $h \sim \mathcal{H}$, i is covered by h with high probability (where the probability is over the choice of h).

Claim 5.17. *Let $\mathcal{H}$ be as in Fact 5.10 and let $i \in [n]$. Then:*

$$\Pr_{h \sim \mathcal{H}}[i \in h([n])] \geq 1/30$$

We prove Claim 5.17 by proving two lemmas. We first show that for any family of k-wise independent hash functions, the range of a hash function chosen uniformly at random from this family is large, with high probability. We then show that the range of a random polynomial of degree at most $k-1$ has a "symmetry" property. Namely, when drawing a function h from the family of polynomials of degree at most $k-1$ (uniformly at random), and for $i, i' \in [n]$, the probability of i to be in the range of h is the same as the probability of i' to be in the range of h.

Lemma 5.18. *(The range of a random hash function is large) For $k \geq 2$ let $\mathcal{H}$ be a family of k-wise independent hash functions from $[n]$ to $[n]$. Then:*

$$\Pr_{h \sim \mathcal{H}}[|h([n])| \leq \frac{n}{20}] \leq 1/3$$

Proof. For a function $h \in \mathcal{H}$, we say that the number of *collisions* in h is the number of (unordered) pairs $\{i,j\}$ such that $h(i) = h(j)$, for $i, j \in [n], i \neq j$. We denote this number by $c(h)$, and note that for any $h \in \mathcal{H}$, $0 \leq c(h) \leq \binom{n}{2}$.

We first show that $|h([n])| \leq \frac{n}{20} \implies c(h) \geq 6n$. That is, if h has small range, it has many collisions. Indeed, denote $m = |h([n])|$ and assume $m \leq \frac{n}{20}$. Choose some arbitrary order for the elements of the range of h, $h([n]) = \{a_1, a_2, \ldots, a_m\}$. Denote by x_i the size of the pre-image of a_i. That is, $x_i = |h^{-1}(a_i)|$. From the definitions we get that $\sum_{i=1}^m x_i = n$, since each "source" appears in exactly one of the pre-images $h^{-1}(a_i)$.

We now count the number of collisions in h, by counting the number of collisions at each a_i (where a collision between i, j is *at* k if $h(i) = h(j) = k$). Observe that i, j collide if and only if there exists $0 \leq i \leq m$ such that $i, j \in h^{-1}(a_i)$. Hence, the number of collisions at a_i is exactly $\binom{x_i}{2}$, and each collision in h is counted exactly once by this method. Therefore, the total number of collisions in h is exactly $c(h) = \sum_{i=1}^{m} \binom{x_i}{2}$.

Consider the function $\varphi : x \to \frac{x(x-1)}{2}$ (and note that $\varphi(x) = \binom{x}{2}$ for an integer x). Evidently, φ is a convex function. Therefore, by Jensen's inequality, we get:

$$\frac{c(h)}{m} = \frac{\sum_{i=1}^{m} \binom{x_i}{2}}{m} = \frac{\sum_{i=1}^{m} \varphi(x_i)}{m} \geq \varphi\left(\frac{\sum_{i=1}^{m} x_i}{m}\right) = \varphi\left(\frac{n}{m}\right) = \frac{1}{2} \cdot (\left(\frac{n}{m}\right)^2 - \frac{n}{m})$$

We now use the assumption that $m \leq \frac{n}{20} \implies \frac{n}{m} \geq 20$ to derive:

$$c(h) \geq \frac{1}{2} \cdot \left(\frac{n^2}{m} - n\right) = \frac{n}{2} \cdot \left(\frac{n}{m} - 1\right) \geq \frac{n}{2} \cdot 19 \geq 6n$$

which is what we wanted to show.

We now claim that $\mathbb{E}_{h\sim\mathcal{H}}[c(h)] \leq \frac{n}{2}$. Indeed, for a fixed pair $\{i, j\}$, we have $\Pr_{h\sim\mathcal{H}}[h(i) = h(j)] = \frac{1}{n}$, since $\mathcal{H}$ is a family of k-wise independent hash functions for $k \geq 2$. Therefore, by the linearity of the expectation, $\mathbb{E}_{h\sim\mathcal{H}}[c(h)] = \frac{\binom{n}{2}}{n} \leq \frac{n}{2}$ as claimed.

Hence, from Markov's inequality:

$$\Pr_{h\sim\mathcal{H}}[|c(h)| \geq 6n] \leq \frac{\mathbb{E}[c(h)]}{6n} \leq \frac{n/2}{6n} = 1/3$$

We showed above that $|r(h)| \leq \frac{n}{20} \implies c(h) \geq 6n$, and therefore:

$$\Pr_{h\sim\mathcal{H}}[|r(h)| \leq \frac{n}{20}] \leq \Pr_{h\sim\mathcal{H}}[|c(h)| \geq 6n]$$

and the Lemma follows. □

We now state and prove the second Lemma which we need to prove Claim 5.17.

Lemma 5.19. *(The probability of being in the range of a random polynomial) Let $n \in \mathbb{N}$ and $k \geq 2$, and let $\mathcal{H}$ be as in Fact 5.10. That is, $\mathcal{H}$ is the family of all polynomials of degree at most $k - 1$ over the field of n elements. Let $i, i' \in [n]$. Then:*

$$\Pr_{h\sim\mathcal{H}}[i \in h([n])] = \Pr_{h\sim\mathcal{H}}[i' \in h([n])]$$

Proof. Recall that $\mathcal{H} = \{h_{a_0,a_1,\ldots,a_k} \mid a_0, a_1, \ldots, a_{k-1} \in \mathbb{F}\}$, where $h_{a_0,a_1,\ldots,a_{k-1}} : \mathbb{F} \to \mathbb{F}$ is defined by $h_{a_0,a_1,\ldots,a_{k-1}}(x) = a_0 + a_1x + a_2x^2_+ \cdots + a_{k-1}x^{k-1}$, and $\mathbb{F}$ is a field with n elements. Let $i, i' \in \mathbb{F}$. We describe a bijection $\phi : \mathcal{H} \to \mathcal{H}$ such that i is in the range of $\phi(h)$ iff i' is in the range of h. The existence of such a bijection yields the required result of the Lemma, as the probabilities in the Lemma statement are over choosing h from $\mathcal{H}$ uniformly at random.

The bijection is defined by:

$$\phi(h_{a_0,a_1,\ldots,a_k}) = h_{a_0-i'+i,a_1,\ldots,a_k}$$

It is immediate to check that ϕ is indeed a bijection (since $\mathbb{F}$ is a field). Now, i is in the range of $\phi(h)$ iff there exists $x \in \mathbb{F}$ such that $(a_0 - i' + i) + a_1x + a_2x^2_+ \cdots + a_{k-1}x^{k-1} = i$, which happens iff $a_0 + a_1x + a_2x^2_+ \cdots + a_{k-1}x^{k-1} = i'$, which happens iff i' is in the range of h. □

We are now ready to prove Claim 5.17.

Proof. Let $i \in [n]$. We need to show that:

$$\Pr_{h\sim\mathcal{H}}[i \in h([n])] \geq 1/30$$

Where $\mathcal{H}$ is the family of all polynomials of degree at most $k-1$ over the field of n elements, which is a family of k-wise independent hash functions. From Lemma 5.19 there exists $0 \leq p \leq 1$ such that for every $i \in [n]$, $\Pr_{h\sim\mathcal{H}}[i \in h([n])] = p$. Therefore, and from the linearity of the expectation:

$$\mathbb{E}_{h\sim H}[h([n])] = n \cdot p \tag{$\star$}$$

On the other hand, by Markov's inequality, for any non-negative random variable X and $a > 0$:

$$\Pr[X \geq a] \leq \frac{\mathbb{E}[X]}{a} \implies \mathbb{E}[X] \geq a \cdot \Pr[X \geq a]$$

By applying this inequality on $X = |h([n])|$ and $a = \frac{n}{20}$ we get:

$$\mathbb{E}[|h([n])|] \geq \frac{n}{20} \cdot \Pr[|h([n])| \geq \frac{n}{20}]$$

But from Lemma 5.18 we have:

$$\Pr_{h\sim\mathcal{H}}[|h([n])| \geq \frac{n}{20}] \geq 2/3$$

and hence:

$$\mathbb{E}[|h([n])|] \geq \frac{n}{20} \cdot \Pr[|h([n])| \geq \frac{n}{20}] \geq \frac{n}{20} \cdot \frac{2}{3} = \frac{n}{30}$$

Together with ($\star$) we get that:

$$n \cdot p = \mathbb{E}_{h\sim H}[h([n])] \geq \frac{n}{30}$$

and hence:

$$\Pr_{h\sim\mathcal{H}}[i \in h([n])] = p \geq 1/30$$

□

We showed that for every fixed $i \in [n]$, and $h \sim \mathcal{H}$, i is covered by h with high probability. We now claim that when drawing m hash functions from $\mathcal{H}$ uniformly at random and i.i.d, $h_1, \ldots, h_m \sim \mathcal{H}$ then for *all* $i \in [n]$ (simultaneously), i is covered by *many* of the hash functions (where the probability is over the choice of the hash functions). Recall that number of hash functions is $m = t \log n$.

Claim 5.20. *Let $\mathcal{H}$ be as in Fact 5.10. For $h_1, \ldots, h_m$ drawn uniformly at random and i.i.d from $\mathcal{H}$, and for every $1 \leq j \leq m$ and $i \in [n]$ denote by X_i^j the indicator random variable that is defined by*

$$X_i^j = \begin{cases} 1 & \textit{if } i \textit{ is covered by } h_j \\ 0 & \textit{otherwise} \end{cases}$$

and for every $i \in [n]$ let $X_i = \sum_{j=1}^m X_i^j$ be the number of values of j such that i is covered by h_j. Then

$$\Pr_{h_1,\ldots,h_m\sim\mathcal{H}}[\exists i \in [n] \textit{ such that } X_i < \frac{m}{60}] \leq n^{-\Omega(t)}$$

Proof. The proof of the above claim is by a standard application of Chernoff's bound and the union bound. First, fix $i \in [n]$. From Claim 5.17, for every $1 \leq j \leq m$, $\Pr[X_i^j = 1] \geq 1/30$. Therefore, letting $\mu = \mathbb{E}[X_i]$ we have $\mu \geq m/30$, and from Chernoff's bound we get:

$$\Pr_{h_1,\ldots,h_m \sim \mathcal{H}}[X_i < \frac{m}{60}] = \Pr[X_i < \frac{1}{2} \cdot \mu] \leq e^{-\frac{(\frac{1}{2})^2 \cdot \mu}{2}} = e^{-\frac{m}{240}} = e^{-\frac{t \log n}{240}} = n^{-t/240}$$

Now, from the union bound:

$$\Pr_{h_1,\ldots,h_m \sim \mathcal{H}}[\exists i \in [n] \text{ such that } X_i < \frac{m}{60}] \leq n \cdot n^{-t/240} = n^{-\Omega(t)}$$

□

Finally, we next use Claim 5.20 to prove Property 3 of Lemma 5.6. Take $Q \in [m \cdot n]^{\{k\}}$, $\epsilon > 0$, and take $x \in \{0,1\}^n$ such that x is 60ϵ-far from L. We need to show that

$$\Pr_{S \sim \mathbf{S}, key \leftarrow K(S,Q)}[f_{key}(x) \text{ is } \epsilon\text{-far from } \{f_{key}(x') \mid x' \in L\}] \geq 1 - n^{-\Omega(t)}$$

For $S \sim \mathbf{S}$ let $key = K(S,Q)$, and let $h_1, \ldots, h_m \in \mathcal{H}$ be the hash functions such that $key = \langle h_1 \rangle \circ \cdots \circ \langle h_m \rangle$. Recall that from Claim 5.14, the distribution of $h_1, \ldots, h_m$ is as if each h_j was drawn uniformly at random and i.i.d from $\mathcal{H}$.

For every $1 \leq j \leq m$ let $f_j : \{0,1\}^n \to \{0,1\}^n$ be the function that is obtained by the construction from h_j. Since $f_{key} = f_1(x) \circ \cdots \circ f_m(x)$, (and since the distribution of $h_1, \ldots, h_m$ is uniform and i.i.d over $\mathcal{H}$ when $S \sim \mathbf{S}$), we get that it is enough to show that:

$$\Pr_{h_1,\ldots,h_j \sim \mathcal{H}}[f_1(x) \circ \cdots \circ f_m(x) \text{ is } \epsilon\text{-far from } \{f_1(x') \circ \cdots \circ f_m(x') \mid x' \in L\}] \geq 1 - n^{-\Omega(t)}$$

Now, for every $i \in [n]$ let X_i be as in Claim 5.20, and denote by B the "bad" event that:

$$\exists i \in [n] \text{ such that } X_i < \tfrac{m}{60}$$

And note that by Claim 5.20 the probability of B happening is very small. Conditioned on B not happening, we get that for every $i \in [n]$ there are at least $\frac{m}{60}$ hash functions that cover i. We now make the following key claim:

Claim 5.21. *Let $0 \leq \alpha \leq 1$. Conditioned on B not happening, for every $x, x' \in \{0,1\}^n$:*

$$\Delta(x, x') > \alpha \implies \Delta(f_{key}(x), f_{key}(x')) > \alpha/60$$

Proof. (of the claim) Assume that the event B does not happen, and let x, x' be of distance at least α. Then there is a set of indices $I \subseteq [n]$ such that $x_i \neq x'_i$ for every $i \in I$, and $|I| > n \cdot \alpha$. Since the event B does not happen, every $i \in I$ is covered by at least $\frac{m}{60}$ hash functions. But for each h_j that covers i, the value of $f_j(x)$ at the indices $h_j^{-1}(i)$ equals x_i, and the value of $f_j(x')$ at these indices is $x'_i \neq x_i$. Therefore, we get that $f_j(x)$ and $f_j(x')$ disagree on all of the indices $h_j^{-1}(i)$, which is a non-empty set when h_j covers i.

Now, since for each $i \in I$ there are at least $\frac{m}{60}$ hash functions that cover i, and thus i "contributes" at least $\frac{m}{60}$ indices on which $f_{key}(x) = f_1(x) \circ \cdots \circ f_m(x)$ and $f_{key}(x') = f_1(x') \circ \cdots \circ f_m(x')$ disagree. This holds for all of the $n \cdot \alpha$ indices in I, and therefore in total we get that $f_{key}(x)$ and $f_{key}(x')$ disagree on at least $n \cdot \alpha \cdot \frac{m}{60}$ indices. Note that there is no "double counting", since for each $1 \leq j \leq m$ and $i \neq i'$ the sets $h_j^{-1}(i)$ and $h_j^{-1}(i')$ are disjoint, and therefore each "disagreement" was counted at most once.

Finally, the length of $f_{key}(x)$ and $f_{key}(x')$ is $n \cdot m$, and they disagree on (at least) $n \cdot \alpha \cdot \frac{m}{60}$ indices, therefore $\Delta(f_{key}(x), f_{key}(x')) > \alpha/60$. □

Now, from the hypothesis, x is 60ϵ-far from every $x' \in L$. That is, $\Delta(x, x') > 60\epsilon$ for every $x' \in L$. Conditioned on the event B not happening, Claim 5.21 holds, and $\Delta(f_{key}(x), f_{key}(x')) > \epsilon$ for every $x' \in L$ simultaneously. Since from Claim 5.20 the event B happens with probability of at most $n^{-\Omega(t)}$, we are done. □

Next, we show that Property 5 of Lemma 5.6 holds.

Claim 5.22. *(Property 5 - Circuit Reversing F) With high probability over the choice of key, the function f_{key} is injective:*

$$\Pr_{S \sim \mathbf{S}, key \leftarrow K(S,Q)}[\exists x, x' \in \{0,1\}^n \text{ such that } x \neq x' \text{ and } f_{key}(x) = f_{key}(x')] \leq n^{-\Omega(t)}$$

Where $\mathbf{S}$ is the uniform distribution on $(m \cdot k)$-tuples of elements from $[n]$.

Moreover, there exists a logspace-uniform $\mathcal{NC}^1$ circuit $\mathcal{C}$ that gets as input $key \in KEYS$ and $y = F(key, x) \in \{0,1\}^{m \cdot n}$, and if f_{key} is injective, outputs x.

Proof. We first show that with high probability f_{key} is injective. As proved in Claim 5.14, when $S \sim \mathbf{S}$ and $key = \langle h_1 \rangle \circ \cdots \circ \langle h_m \rangle \leftarrow K(S, Q)$, the distribution of each h_j is uniform, and $h_1, \ldots, h_m$ are independent of each other.

For $i \in n$, let X_i be as in Claim 5.20. Fix *key*, and assume there are $x, x' \in \{0,1\}^n$ such that $x \neq x'$ and $f_{key}(x) = f_{key}(x')$. Let $i \in [n]$ be an index such that $x_i \neq x'_i$. Then i is an index that is not covered by any hash function h_j. That is, $X_i = 0$. But from Claim 5.20, the event that there exists i such that $X_i = 0$ happens with probability at most $n^{-\Omega(t)}$. Hence, the event that there exists $x, x' \in \{0,1\}^n$ such that $x \neq x'$ and $f_{key}(x) = f_{key}(x')$ also happens with probability at most $n^{-\Omega(t)}$.

Construction of $\mathcal{C}$: We next describe the $\mathcal{NC}^1$ circuit $\mathcal{C}$. The input of the circuit $\mathcal{C}$ is $key = \langle h_1 \rangle \circ \cdots \circ \langle h_m \rangle$ and $y = y_1 \circ \cdots \circ y_m \in \{0,1\}^{m \cdot n}$. Its output is $x \in \{0,1,\perp\}^n$ such that if f_{key} is injective and there exists $x' \in \{0,1\}^n$ such that $f_{key}(x) = y$, then $x = x'$.

The high level idea of the construction is as follows. Recall that from the construction of F, if $f_{key}(x) = y$ then $[y_j]_i = x_{h_j(i)}$ for every j. Accordingly, to compute the ℓ-th bit of the output x, we need to find an index i (of some y_j) such that $h_j(i) = \ell$. Towards this end, we first evaluate *in parallel* $h_j(i)$ for every j and i. Next, for every j and output index ℓ we use these evaluations to search for an index i such that $h_j(i) = \ell$. From the assumption that f_{key} is injective, for every index ℓ there exists some h_j for which the search succeeds. Finally, as the l-th bit of the output of $\mathcal{C}$ we return the value of y_j at index i.

We next give the exact construction of the circuit. The circuit $\mathcal{C}$ is constructed in layers, from sub-circuits of 4 types.

1. $\mathcal{H}$-circuits ("hash function evaluation" circuits). Each such circuit gets as input a description of a hash function $\langle h \rangle$ and an index $i \in [n]$ (that matches an index of some y_j), and outputs $h(i)$.

2. $\mathcal{R}$-circuits ("search" circuits). Each such circuit gets as input $\ell^*, \ell_1, \ldots, \ell_n \in [n]$, and outputs $i \in [n]$ such that $\ell_i = \ell^*$ (if such i exists), or outputs $\perp$ if no such index exists.

3. $\mathcal{P}$-circuits ("projection" circuits). Each such circuit gets as input $a \in \{0,1\}^n$ and $i \in [n] \cup \{\perp\}$, and outputs $a_i \in \{0,1\}$ if $i \neq \perp$ and $\perp$ if $i = \perp$.

4. $\mathcal{M}$-circuits ("combining" circuits). Each such circuit gets as input $a_1, \ldots, a_m \in \{0,1,\perp\}$. If $a_1, \ldots, a_m = \perp$, the output of the circuit is $\perp$. Otherwise, its output is a_j for some $a_j \neq \perp$ (for concreteness, it outputs such a_j with the smallest value of j such that $a_j \neq \perp$)

We next describe how these sub-circuits are combined to construct $\mathcal{C}$. The first layer of $\mathcal{C}$ evaluates $h_j(i)$ for every $1 \leq j \leq m$ and $i \in [n]$. It contains $n \cdot m$ $\mathcal{H}$-circuits, $\mathcal{H}_{1,1}, \mathcal{H}_{1,2}, \ldots, \mathcal{H}_{1,n}, \mathcal{H}_{2,1}, \ldots, \mathcal{H}_{n,m}$. For $1 \leq j \leq m$ and $i \in [n]$, the input of the circuit $\mathcal{H}_{i,j}$ is $\langle h_j \rangle$ (which is an input of the full circuit $\mathcal{C}$), and i is an hard-wired input of it. The output of the circuit $\mathcal{H}_{i,j}$ is $h_j(i)$.

The second layer of $\mathcal{C}$ operates independently for each h_j. For each index ℓ (of the output of $\mathcal{C}$) and each hash function h_j, it "searches" some index i (of y_j) such that $h_j(i) = \ell$. That is, it finds an element of $h_j^{-1}(\ell)$ (if such an element exists). The layer is constructed as follows. It contains $n \cdot m$ $\mathcal{R}$-circuits, $\mathcal{R}_{1,1}, \mathcal{R}_{1,2}, \ldots, \mathcal{R}_{1,n}, \mathcal{R}_{2,1}, \ldots, \mathcal{R}_{n,m}$. For $1 \leq j \leq m$ and $l \in [n]$, the inputs of the circuit $\mathcal{R}_{l,j}$ are the outputs of the circuits $\mathcal{H}_{1,j}, \mathcal{H}_{2,j}, \ldots, \mathcal{H}_{n,j}$, and ℓ is an hard-wired input of it. From the definition of $\mathcal{R}$, the output of $\mathcal{R}_{\ell,j}$ is some index in $h_j^{-1}(\ell)$ if one exists, and $\perp$ if $h_j^{-1}(\ell)$ is empty.

The third layer of $\mathcal{C}$ "projects" the input $y = y_1 \circ \cdots \circ y_m \in \{0,1\}^{m \cdot n}$ on the indices computed at the previous layer. It contains $n \cdot m$ $\mathcal{P}$-circuits, $\mathcal{P}_{1,1}, \mathcal{P}_{1,2}, \ldots, \mathcal{P}_{1,n}, \mathcal{P}_{2,1}, \ldots, \mathcal{P}_{n,m}$. For $1 \leq j \leq m$ and $\ell \in [n]$, the inputs of $\mathcal{P}_{\ell,j}$ are $y_j \in \{0,1\}^n$ and the output of $\mathcal{R}_{\ell,j}$. From the definition of $\mathcal{R}$, the output of $\mathcal{P}_{\ell,j}$ is $[y_j]_i$ for some index i such that $h_j(i) = \ell$ (if $h_j^{-1}(\ell)$ is not empty) and $\perp$ if $h_j^{-1}(\ell)$ is empty. Note that from the construction, if $y = F(key, x)$, then $[y_j]_i$ has the same value for every i such that $h_j(i) = \ell$. Namely, its value is $[y_j]_i = x_\ell$.

Lastly, the forth layer of $\mathcal{C}$ combines the outputs of the sub-circuits of the previous layer, to output the final output of $\mathcal{C}$. This layer contains n $\mathcal{M}$-circuits, $\mathcal{M}_1, \ldots, \mathcal{M}_n$ one for each output bit of $\mathcal{C}$. For $\ell \in [n]$, the inputs of $\mathcal{M}_\ell$ are the outputs of the circuits $\mathcal{P}_{\ell,1}, \ldots, \mathcal{P}_{\ell,m}$. The output of $\mathcal{M}_\ell$ is returned as the ℓ-th bit of the output of $\mathcal{C}$.

Correctness: Fix $key = \langle h_1 \rangle \circ \cdots \circ \langle h_m \rangle$, and assume that f_{key} is injective. Let $x \in \{0,1\}^n$, let $f_1, \ldots, f_m$ be the functions defined in the construction of F (namely, $[f_j(x)]_i = x_{h_j(i)}$ for every $i \in [n]$). Let $y = y_1 \circ \cdots \circ y_j = f_1(x) \circ \cdots \circ f_j(x) \in \{0,1\}^{m \cdot n}$.

We need to show that the output of $\mathcal{C}$ on the inputs key, y is x. Consider the ℓ-th bit of x. From the construction, the value $\mathcal{C}$ outputs for this bit is the output of $\mathcal{M}_\ell$. Again from the construction, if $h_j^{-1}(\ell)$ is not empty for some $1 \leq j \leq m$, the output of $\mathcal{M}_\ell$ is the value $[y_j]_i$ for some i and j such that $h_j(i) = \ell$. But now, from the hypothesis $f_j(x) = y_j$ and from the definition of f, we get that $\mathcal{C}$ correctly outputs the value $[y_j]_i = [f_j(x)]_i = x_{h_j(i)} = x_\ell$.

We are left to show that there exists $1 \leq j \leq m$ such that $h_j^{-1}(\ell)$ is not empty (otherwise, the output of $\mathcal{C}$ for the bit x_ℓ is $\perp$). Assume towards contradiction that $h_j^{-1}(\ell)$ is empty for every $1 \leq j \leq m$. Let $x' \in \{0,1\}^n$ be the same as x, but with the ℓ-th bit flipped. We have $x' \neq x$, but since $h_j^{-1}(\ell)$ is empty, and from the definition of F, we have $f_{key}(x) = f_{key}(x')$, which is a contradiction to the assumption that f_{key} is injective.

Complexity: From Fact 5.9, the circuits $\mathcal{H}$ can be constructed with size $n^{O(1)}$ and depth $O(\log n)$. The other types of sub-circuits can be constructed with $n^{O(1)}$ and depth $O(\log n)$ in a straight-forward fashion. From the layered structure of $\mathcal{C}$, we get that $\mathcal{C}$ itself has size $n^{O(1)}$ and depth $O(\log n)$. Logspace uniformity follows by construction.

□

Next, we claim that a similar circuit can be used to check if an input f_{key} is injective or not.

Claim 5.23. *(Circuit checking if f_{key} is injective) There exists a logspace-uniform $\mathcal{NC}^1$ circuit that gets as input key $\in$ KEYS and outputs "yes" if and only if f_{key} is injective.*

Proof. The construction of the circuit is in layers, as follows. The first two layers are the same as in the circuit from Claim 5.22. Namely, they contain $n \cdot m$ $\mathcal{H}$-circuits (for the first layer), and $n \cdot m$

$\mathcal{R}$-circuits (for the second layer). For every $l \in [n]$ and $1 \leq j \leq m$, the output of $\mathcal{R}_{\ell,j}$ is an index in $h_j^{-1}(\ell)$ if one exists, and $\perp$ if $h_j^{-1}(\ell)$ is empty.

The next layer of the circuit contains m combining circuits, which operate similarly to the $\mathcal{M}$-circuits. The difference is that now, the input of each such circuit is $a_1, \ldots, a_n \in [n] \cup \{\perp\}$ (instead of input in $\{0, 1, \perp\}$).

For $\ell \in [n]$, the inputs of $\mathcal{M}_\ell$ are the outputs of $\mathcal{R}_{\ell,1}, \ldots, \mathcal{R}_{\ell,m}$. By this construction and by the operation of $\mathcal{M}$, we have that the output of $\mathcal{M}_\ell$ is $\perp$ if and only if the index ℓ is covered by h_j for some $1 \leq j \leq m$.

The last layer of the circuit is a single sub-circuit, that gets as input the outputs of all of the $\mathcal{M}_\ell$ circuits, and output "yes" if and only if none of its inputs is $\perp$.

We get that the output of the full circuit is "yes" if and only if each index $\ell \in [n]$ is covered by some hash function h_j. The correctness of the claim follows, since f_{key} is injective if and only if each index $\ell \in [n]$ is covered by some hash function h_j.

The complexity and uniformity analysis is similar to the one in Claim 5.22. □

Lastly, we show that Property 6 of Lemma 5.6 holds.

Claim 5.24. *(Property 6 - Efficient Recovery) There exists a TM that gets as input $key \in KEYS$ and $q_1, \ldots, q_k \in [m \cdot n]$ and outputs $R(key, q_1), \ldots, R(key, q_k) \in [n]$, and its runtime is $k \cdot poly(\log k, \log n, t)$.*

Proof. For input $key = \langle h_1 \rangle \circ \cdots \circ \langle h_m \rangle \in KEYS$ and $q_1, \ldots, q_k \in [m \cdot n]$, the TM works as follows. First, it executes the Sub-steps (a) and (b) of Step (7) in Figure 4 for all of $q_1, \ldots, q_k$ one by one. That is, for every $q \in \{q_1, \ldots, q_k\}$ it computes j_q such that $q \in I_{j_q}$, and then computes $i_q = q - (j-1) \cdot n$ (again, i_q is the "relative" index of q in I_{j_q}). These steps involve only basic arithmetic operations that can be done in time $\text{poly}(\log n, \log k, t)$. Therefore, executing them sequentially for all of $q_1, \ldots, q_k$ takes $k \cdot \text{poly}(\log n, \log k, t)$ time.

Now, for Sub-step (c), evaluating $h_{j_q}(i_q)$ for every $q \in \{q_1, \ldots, q_k\}$ in a sequential fashion would take $k^2 \cdot \text{poly}(\log n, \log k, t)$ time, which is too much. Instead, the TM uses the Fast Fourier Transform in the following way. For every $1 \leq j \leq m = t \cdot \log n$, it computes $h_j(i_q)$ for *all* $q \in \{q_1, \ldots, q_k\}$. Some of the values computed this way are not needed. But by using FFT as a black-box the TM is able to, for every fixed h_j, evaluate all of the values $h_j(i_q)$ *simultaneously* in time $k \cdot \text{poly}(\log n, \log k)$ (see Fact 5.11). Accordingly, the TM performs m applications of FFT, one for each polynomial h_j. In each application it evaluates the value of $h_j(i_q)$ for all of $q \in \{q_1, \ldots, q_k\}$. Each application is done in time $k \cdot \text{poly}(\log n, \log k)$, and there are m applications, hence the total run-time of this sub-step is $m \cdot k \cdot \text{poly}(\log n, \log k) = k \cdot \text{poly}(\log n, \log k, t)$.

As part of its computations, the TM computed $h_{j_q}(i_q)$ for every $q \in \{q_1, \ldots, q_k\}$, so it can output it as the value for $R(key, q)$. (It ignores the values it evaluated of the form $h_j(i_q)$ for $j \neq j_q$.) By the definition of R in Figure 4, the TM returns the correct output. In addition, as we showed, its runtime is $k \cdot \text{poly}(\log n, \log k, t)$. □

5.3 Proof of the Reduction Protocol

In this section we describe the reduction protocol formally, use the properties proved in Lemma 5.6 to analyze it, and prove Theorem 5.2.

Proof. (of Theorem 5.2)

Let $\mathcal{L}$ be a family of pair languages that is closed under composition with log-space uniform $\mathcal{NC}^1$ circuits. Let $\epsilon > 0$ be a real value and let $q, c, v, d : \mathbb{N} \to \mathbb{N}$ be functions such that every $L \in \mathcal{L}$ has a (query-based) IPP with proximity parameter ϵ with the following properties:

1. The protocol uses only public-coins.

2. The query complexity of the protocol is $O(q(n))$, its communication complexity is $O(c(n))$, its randomness complexity (the number of public coins the verifier tosses) is $r = r(n)$, the verifier running time is $O(v(n))$, and its round complexity is $\mathrm{poly}(d)$. Also assume (w.l.o.g) that the verifier tosses exactly $r(n)$ coins.

3. The indices of the queries depend only on the public coins of the verifier. That is, for each $L \in \mathcal{L}$, there is a deterministic function that maps the public coin tosses of the verifier to the indices it queries in the input. Also assume (w.l.o.g) that the verifier queries the input only at the end of the IPP, after the interaction with the prover ends.

4. The (honest) prover runs in time $\mathrm{poly}(n)$.

Let $L \in \mathcal{L}$. We need to prove that L has a *sample-based* IPP with proximity parameter 120ϵ. In addition, we need to prove that the sample complexity of this SIPP is $\widetilde{O}(q(n))$, that its communication complexity is $\widetilde{O}(c(n) + q(n))$, that its round complexity is $\mathrm{poly}(d)$, that the verifier running time is $\widetilde{O}(v(n) + q(n))$, and that the honest prover runs in time $\mathrm{poly}(n)$.

First, the verifier has explicit access to n, the length of the implicit input. We can assume w.l.o.g that n is a power of 2. If not, padding can be used in a way that damages the proximity of the implicit input from the language by a factor of at most 2.[28]

Let $k = \log n \cdot q(n)$, and $t = 480$ (and $m = t \log n$), and take K, F, R to be as in Lemma 5.6. That is, $K : [n]^{m \cdot k} \times [m \cdot n]^{\{k\}}$, $F : \mathit{KEYS} \times \{0,1\}^n \to \{0,1\}^{m \cdot n}$, $R : \mathit{KEYS} \times [m \cdot n] \to [n]$, where $\mathit{KEYS} = \{0,1\}^{m \cdot k \cdot \log n}$.

Now, we want to consider the pair language $\{(\mathit{key}, F(\mathit{key}, x)) \mid \mathit{key} \in \mathit{KEYS}, x \in L\}$, where key is the explicit part of the input, and $F(\mathit{key}, x)$ is the implicit part of the input. The issue is that this language may not be in $\mathcal{L}$. For a fixed key, to check if (key, y) is in this language, one needs to find an x such that $f_{\mathit{key}}(x) = y$. This might take exponential time, and if f_{key} is not injective there might be multiple valid values for such x. Therefore, we define a slightly different language, which is in $\mathcal{L}$.

Let $\mathcal{C}$ be the $\mathcal{NC}^1$ circuit from property 5 of Lemma 5.6. That is, $\mathcal{C}$ is a circuit that gets as input key and y, and if f_{key} is injective, outputs x such that $f_{\mathit{key}}(x) = y$. Consider the language L' that is constructed by composition of $\mathcal{C}$ and L.[29] That is, L' is a pair language, where key is the explicit part of the input and y is the implicit part of the input. A pair (key, y) is in L' iff for input (key, y), the output of the circuit $\mathcal{C}$ is in L.

Since $\mathcal{L}$ is closed under composition with log-space uniform $\mathcal{NC}^1$ circuits, $L' \in \mathcal{L}$. Therefore, L' has a (query-based) IPP with proximity parameter ϵ, with the properties described in the hypothesis.

Denote this IPP by $\mathcal{IP}'$, and denote its query complexity by $k' = O(q(n))$. Observe that k' is (asymptotically) smaller than k. Let $r(n)$ be the number of coins the verifier needs to toss in the execution of $\mathcal{IP}'$.

Now, let $L'' = \{(\mathit{key}, F(\mathit{key}, x)) \mid \mathit{key} \in \mathit{KEYS}, x \in L\}$, and note that $L' \neq L''$. But from Lemma 5.6, for every key such that $f_{\mathit{key}}(x)$ is injective, $(\mathit{key}, y) \in L'$ if and only if $(\mathit{key}, y) \in L''$. In addition, with high probability over the choice of key, $f_{\mathit{key}}(x)$ is injective. Hence, with high probability we also have that $(\mathit{key}, y) \in L'$ if and only if $(\mathit{key}, y) \in L''$.

The SIPP for L is as presented in Section 5.2, which we describe formally in Figure 5.

We next prove the correctness of this protocol.

[28] That it, let n' be the smallest power of 2 larger than n. For $x \in \{0,1\}^n$ define $\mathrm{padded}(x) = (x, 0, \dots, 0) \in \{0,1\}^{n'}$, where $n' - n$ zeros are used for the padding. Define $L_n = L \cap \{0,1\}^n$ and define $L'_n = \{\mathrm{padded}(x) \mid x \in L_n\}$. Replace the statement $x \in L_n$ (which is equivalent to $x \in L$, since the length of x is n) with the statement $(x, 0, \dots, 0) \in L'_n$. It is easy to see that $x \in L_n \iff \mathrm{padded}(x) \in L'_n$, and that if $\Delta(x, L_n) > 2\alpha$ then $\Delta(\mathrm{padded}(x), L'_n) > \alpha$ (since $n' < 2n$).

[29] We identify a circuit with the language it computes, and identify a language $L \subseteq \{0,1\}^*$ with the boolean function $L : \{0,1\}^* \to \{0,1\}$ that outputs 1 iff the input is in the set L.

Figure 5 Reduction SIPP for L
Verifier Input: Sample access to $x \in \{0,1\}^n$, explicit access to ϵ.
Prover Input: Full (explicit) access to x.
Output: Accept / Reject

1. $\mathcal{V}$ draws $m \cdot k$ samples from x, denote their indices $S \in [n]^{m \cdot k}$.
2. $\mathcal{V}$ tosses $r(n)$ coins (privately), denote the results by $\rho \in \{0,1\}^{r(n)}$.
3. $\mathcal{V}$ computes Q', the k' indices a verifier should have queried the implicit input after executing $\mathcal{IP}'$, assuming the ρ were the coins tosses during the execution of the protocol.
   (a) $\mathcal{V}$ adds $k - k'$ arbitrary indices in $[m \cdot n]$ to Q', to produce the tuple $Q \in [m \cdot n]^{\{k\}}$.
4. $\mathcal{V}$ computes $\mathit{key} = K(S, Q)$.
5. $\mathcal{V}$ sends key to $\mathcal{P}$.
6. $\mathcal{V}$ and $\mathcal{P}$ execute $\mathcal{IP}'$ on the statement $(\mathit{key}, f_{\mathit{key}}(x)) \in L'$, where key is the explicit input and $f_{\mathit{key}}(x) \in \{0,1\}^{m \cdot n}$ is the implicit input, and with the following modifications:
   (a) $\mathcal{V}$ uses ρ as its random coins (instead of fresh randomness).
   (b) Each time $\mathcal{V}$ needs to query the implicit input $f_{\mathit{key}}(x)$ at index q, it uses the value $x_{R_{\mathit{key}}(q)}$ instead. Note that this happens only at the end of the execution of $\mathcal{IP}'$.
7. After the execution of $\mathcal{IP}'$, $\mathcal{V}$ accepts or rejects according to verdict of $\mathcal{IP}'$.

Correctness and Verifier Running Time: We first show that the protocol is well defined, and can be performed by an efficient verifier. We show that each step of the protocol can be performed. Step 1 can be performed since $\mathcal{V}$ has sample-access to x, and in time $O(m \cdot k) = \widetilde{O}(q(n))$. Step 2 can be performed since by property 2 of the hypothesis, during the execution of $\mathcal{IP}'$ the verifier tosses exactly $r(n)$ coins. The running time of the verifier in this step is at most $O(v(n))$.

Step 3 can be performed since by property 3 of the hypothesis, the indices of the queries which the verifier must perform at the end of $\mathcal{IP}'$ depend only on its public coin tosses. Since $k' \leq k$ (for large enough n), the size of Q' is not larger than k, and therefore $\mathcal{V}$ can compute the tuple Q of size k, which contains all of the indices a verifier should have queried after executing $\mathcal{IP}'$. For concreteness, the addition of $k' - k$ indices in step 3 (a) can be done by adding to Q' the first $k - k'$ that are not already in Q'. The computation of the query indices from the coin tosses must be done by the verifier of $\mathcal{IP}'$, and therefore the runtime of step 3 is at most $O(v(n))$.

Step 4 can be performed by the verifier, and at time $k \cdot \text{poly}(\log k, \log n) = \widetilde{O}(q(n))$, by the first section of Lemma 5.6. In step 5, $\mathcal{V}$ sends to $\mathcal{P}$ a single message (the key), which is an element of $KEYS = \{0,1\}^{m \cdot k \cdot \log n}$. The running time of the verifier in this step is $m \cdot k \cdot \log n = \widetilde{O}(q(n))$.

In step 6, the modification of using ρ instead of tossing coins with fresh randomness can be done, again, by the assumption that the verifier tosses exactly $r(n)$ coins (we show later that using ρ instead of fresh randomness does not hurt the soundness of $\mathcal{IP}'$).

Additionally in step 6, for every $q \in Q$, the verifier can use the value $x_{R_{key}(q)}$ instead of $f_{key}(x)$ at index q since from Lemma 5.6, property 1, we have $[f_{key}(x)]_q = x_{R_{key}(q)}$ for every $q \in [m \cdot n]$. In addition, for every $q \in Q$ the verifier has the value of $x_{R_{key}(q)}$; from property 2 of Lemma 5.6, $R_{key}(q) \in S$, and the verifier has the value of x_s for every $s \in S$ from the samples it draws at step 1. In other words, the verifier has the value of $f_{key}(x)$ at all of the indices $q \in Q$. From step 3, Q indeed contains all of the indices the verifier needs to query $f_{key}(x)$ for the execution of $\mathcal{IP}'$.

The runtime of the verifier for simulating $\mathcal{IP}'$ is by definition at most $O(v(n))$. In addition, the verifier can compute $R_{key}(q)$ for every $q \in Q$ in time $k \cdot \text{poly}(\log k, \log n, t) = \widetilde{O}(q(n))$ by Lemma 5.6, property 5 (Efficient Recovery).

In conclusion, we proved that the protocol described in Figure 5 can be performed, and the run time of the verifier is $\widetilde{O}(v(n) + q(n))$. We next prove the completeness and soundness of the protocol.

f_{key} is injective with high probability: We first observe that with probability of 0.99 over the samples of the verifier, the function f_{key} is injective. Denote by $\mathbf{S}$ is the uniform distribution on $(m \cdot k)$-tuples of elements from $[n]$. Observe that in step 1 of the protocol, $\mathcal{V}$ draws $(m \cdot k)$ samples uniformly at random from x, and therefore the distribution of their indices is $\mathbf{S}$. From property 5 of Lemma 5.6 (Circuit Reversing F) we have:

$$\Pr_{S \sim \mathbf{S}, key \leftarrow K(S,Q)}[\exists x, x' \in \{0,1\}^n \text{ such that } x \neq x' \text{ and } f_{key}(x) = f_{key}(x')] \leq n^{-\Omega(t)}$$

Hence, from the choice of t, and for large enough n, with probability of at least 0.99 over the samples, f_{key} is injective.

Observe that in this case, $(key, y) \in L'$ if and only if $(key, y) \in \{(key, F(key, x)) \mid key \in KEYS, x \in L\}$ (simultaneously for all y). Hence, $\Delta((key, f_{key}(x)), L') = \Delta(f_{key}(x), \{f_{key}(x') \mid x' \in L\})$. Specifically, $f_{key}(x)$ is ϵ-far from $\{f_{key}(x') \mid x' \in L\}$ if and only if $(key, f_{key}(x))$ is ϵ-far from L'.[30]

Completeness: Let $x \in L$. We need to show that there exists a prover $\mathcal{P}$ that makes $\mathcal{V}$ accept. Since $x \in L$ and with probability 0.99 (over the verifier's samples) f_{key} is injective, we have $(key, f_{key}(x)) \in L'$ with probability 0.99. In this case, from the (full) completeness of $\mathcal{IP}'$, there exists

[30]We denote the distance of a pair (a, b) from a pair language L by $\Delta((a, b), L) = \min_{b'} \Delta(b, b')$, where a is the explicit input of L, b is the implicit input, and the minimum is taken over all b' such that $(a, b') \in L$.

a prover strategy $\mathcal{P}'$ for $\mathcal{IP}'$ that makes the verifier accept $(key, f_{key}(x))$ when executing $\mathcal{IP}$. The prover $\mathcal{P}$ which makes $\mathcal{V}$ accept x is the prover that simply simulates $\mathcal{P}'$ during step 6 of the protocol. Note that indeed $\mathcal{P}'$ can be simulated, since $\mathcal{P}$ gets *key* at step 5 of the protocol, and $F(key, x)$ can be computed by the prover.

We get that if $x \in L$, then with probability 0.99 the verifier accepts after the execution of the protocol described in Figure 5.

Remark 5.25 (Obtaining perfect completeness). *Our protocol as presented above obtains* imperfect *completeness: there is a small probability that the function f_{key} is not injective, and in this case the verifier might reject even if it is interacting with the honest prover. To get perfect completeness, the verifier must check if f_{key} is injective. If it is not injective, then the verifier can halt the execution of the protocol and accept (without executing $\mathcal{IP}'$). This hurts the soundness of the protocol, since now if f_{key} is not injective and x is far from L, the verifier wrongly accepts. But since f_{key} is injective with probability* 0.99*, this hurt is minor.*

The issue is that checking if f_{key} is injective can be time-expensive for the verifier. A solution is to use a doubly-efficient interactive proof (DEIP) for this claim (e.g., using the protocol of [GKR08]). This can be done, since from Claim 5.23 there exists an $\mathcal{NC}^1$ circuit that gets key as input and outputs if f_{key} is injective or not. Note that the input length of this protocol is the length of key, which is $\widetilde{O}(q(n))$. Hence, the verifier's running time for executing this sub-protocol is at most $\widetilde{O}(q(n))$, and so is the contribution of this protocol to the communication complexity of the entire reduction protocol.

The main added cost for obtaining perfect completeness is in the added round complexity needed to run the DEIP.

Soundness: Let $x \in \{0,1\}^n$ and assume x is 120ϵ-far from L. We first claim that with high probability over $\mathcal{V}$'s choice of key, $f_{key}(x)$ is ϵ-far from $\{f_{key}(x') \mid x' \in L\}$.

As above, the distribution of the verifier's samples is $\mathbf{S}$. Hence, as the padding hurts the proximity by a factor of at most 2, and from property 3 of Lemma 5.6 (distance preserving), we have that for x that is 60ϵ-far from L:

$$\Pr_{S \sim \mathbf{S}, key \leftarrow K(S,Q)}[f_{key}(x) \text{ is } \epsilon\text{-far from } \{f_{key}(x') \mid x' \in L\}] \geq 1 - n^{-\Omega(t)}$$

Therefore, at step 6 of the protocol, from the choice of t, and for large enough n, with probability of at least 0.99, $f_{key}(x)$ is ϵ-far from $\{f_{key}(x') \mid x' \in L\}$. In addition, with probability of at least 0.99, f_{key} is injective.

We now claim that if $f_{key}(x)$ is ϵ-far from $\{f_{key}(x') \mid x' \in L\}$ and f_{key} is injective, then by the soundness of the $\mathcal{IP}'$ protocol $\mathcal{V}$ rejects the statement $(key, f_{key}(x)) \in L'$ at the simulation of $\mathcal{IP}'$ at step 6. First, from the observation above, if $f_{key}(x)$ is ϵ-far from $\{f_{key}(x') \mid x' \in L\}$ then $(key, f_{key}(x))$ is ϵ-far from L'. Hence, the hypothesis condition for the soundness of the $\mathcal{IP}'$ protocol holds.

Next, we need to show that the soundness of $\mathcal{IP}'$ is maintained after the two modifications made to it. The second modification (step 6 (b)) is that the verifier, for each query index q, uses the value $x_{R_{key}(q)}$ (obtained by its samples) instead of querying $f_{key}(x)$ at index q. This change does not hurt the soundness of the protocol, as it is done privately by the verifier; we assume the samples the verifier draws are reliable (cannot be altered by the prover) and private (cannot be seen by the prover).

The possible issue is with the first modification (step 6 (a)), of using ρ as the coin tosses for executing $\mathcal{IP}'$. Recall that the soundness of a (public-coins) protocol is promised only when the randomness of the random coin tosses is "fresh". Here this is clearly not the case. ρ, the random coins used during $\mathcal{IP}'$, are pre-computed by the verifier before the protocol $\mathcal{IP}'$ begins. If that was the only modification, there was no issue - the prover has no way to know if the verifier tossed all of the coins privately and then used them one-by-one, or tosses then "on the go", when he needs to send them.

But here, the verifier computes a key $K(S, Q)$ that *depends on* ρ, and sends it to the prover before they execute the protocol $\mathcal{IP}'$. This key might leak "information" on ρ, and hence the prover might have prior knowledge on the randomness used for $\mathcal{IP}'$. If this happens, the soundness of $\mathcal{IP}'$ hurts, and the soundness of the reduction protocol might not hold.

We claim that this is not the case. That is, from the construction of K, the key $K(S, Q)$ that is sent to the prover does not leak any information on ρ.

Denote the key the prover receives by key^*. It is enough to show that for any $\rho \in \{0,1\}^{r(n)}$, and for any $key \in KEYS$, the probability (from the prover perspective) of the verifier's coins to be ρ (before seeing key^*) is the same as the probability of the verifier's coins to be ρ conditioned on $key^* = key$. We prove something stronger. We claim that from the prover perspective, key^* seems as it was sampled from a universal distribution that does not depend on the verifier's coins. Namely, for any $\rho, \rho' \in \{0,1\}^{r(n)}$, we want to show that the distribution of key^* when the verifier's coins are ρ is the same as the distribution of key^* when the verifier's coins are ρ'. This is a strong notion of "not leaking information" regarding ρ, since from the prover perspective, all it sees at step 5 is a sample from a distribution that does not depend on ρ.

Indeed, fix $\rho, \rho' \in \{0,1\}^{r(n)}$. Denote by $Q(\rho) \in [m \cdot n]^k$ (resp., $Q(\rho')$) the query indices computed by the verifier at step 3 for random coins ρ (resp., ρ'). From Property 4 of Lemma 5.6 (Q-hiding), we have that for any $key \in KEYS$:

$$\Pr_{S' \sim \mathbf{S}}[K(S', Q(\rho)) = key] = \Pr_{S' \sim \mathbf{S}}[K(S', Q(\rho')) = key]$$

where $\mathbf{S}$ is again the uniform distribution on $(m \cdot k)$-tuples of elements from $[n]$.

Since the verifier keeps its samples secret, and the distribution of their indices is $\mathbf{S}$, we get that from the prover point of view:

$$\Pr[key^* = key \mid \text{Verifier's coins } = \rho] = \Pr[key^* = key \mid \text{ Verifier's coins } = \rho']$$

where the probability is over the samples of the verifier.

Hence, the distribution of key^* when the verifier's coins are ρ is indeed the same as the distribution of key^* when the verifier's coins are ρ'. Therefore, $K(S, Q)$ does not leak any information on the verifier's coins; from the prover perspective, for every $\rho \in \{0,1\}^{r(n)}$ and $key \in KEYS$:

$$\Pr[\text{Verifier's coins } = \rho \mid key^* = key] = \Pr[\text{Verifier's coins } = \rho]$$

where the probability is again over the samples of the verifier. Hence, the soundness of $\mathcal{IP}'$ is maintained, even though ρ is used instead of fresh randomness during its execution.

In conclusion, we showed that when $x \in \{0,1\}^n$ is 120ϵ-far from L then with probability of at least 0.99, $f_{key}(x)$ is ϵ-far from $\{f_{key}(x') \mid x' \in L\}$ (by the distance preserving property), and with probability of at least 0.99, $f_{key}(x)$ is injective. We also showed that in this case, the soundness of $\mathcal{IP}'$ is maintained (by the Q-hiding property). That is, there is no prover that can convince the verifier in $\mathcal{IP}'$ to accept the claim $(key, f_{key}(x)) \in L'$ with probability larger than 1/3. Therefore there is no prover for the reduction protocol that can convince the verifier to accept the claim $x \in L$ with probability larger than $\frac{1}{0.99} \cdot \frac{1}{0.99} \cdot \frac{1}{3}$. By repeating the the entire reduction protocol twice (which only increases its communication and sample complexities by a factor of 2), we get a protocol with soundness error of 1/3, as required.

Sample Complexity: The verifier draws samples only at step 1 of the protocol. At this step it draws $m \cdot k$ samples, where $m = 480 \log n$, and $k = q(n) \cdot \log n$. Therefore, the sample complexity of the protocol is $\widetilde{O}(q(n))$, as required.

Communication Complexity: The only communication between the verifier and the prover happens at steps 5 and 6 of the protocol. At step 5, the verifier sends a message which is an element of $\mathit{KEYS} = \{0,1\}^{m \cdot k \cdot \log n}$, where $m = 480 \log n$, and $k = q(n) \cdot \log n$. Therefore, the communication complexity of this step is $\widetilde{O}(q(n))$.

The communication complexity at step 6 is as the communication complexity of the protocol $\mathcal{IP}'$. From the hypothesis, and since the length of the implicit input in $\mathcal{IP}'$ is $m \cdot n = \widetilde{O}(n)$, the communication complexity of $\mathcal{IP}'$ is $\widetilde{O}(c(n))$.

Therefore, the total communication complexity of the protocol is $\widetilde{O}(c(n) + q(n))$.

Rounds Complexity: Again, the communication between the verifier and the prover in the protocol described in Figure 5 happens at steps 5 and 6. At step 5 a single message is sent. The number of communication rounds at step 6 is as the number of rounds in $\mathcal{IP}'$. From the hypothesis, the round complexity of $\mathcal{IP}'$ is $\mathrm{poly}(d)$. Therefore, in total, the round complexity of the protocol is $\mathrm{poly}(d)$.

Honest Prover Running Time: The (honest) prover has full access to x, and it gets key from the verifier at step 5 of the protocol. From Lemma 5.6, the prover can compute $F(\mathit{key}, x) = f_{\mathit{key}}(x)$ in $\mathrm{poly}(n)$ time.

After computing $f_{\mathit{key}}(x)$, the prover can execute the protocol $\mathcal{IP}'$ in $\mathrm{poly}(n)$ time, from the hypothesis. In total, we get that the honest prover running time is $\mathrm{poly}(n)$.

□

5.4 Applications of the Reduction to Concrete Families

We present next known families of languages, for which the hypothesis of Theorem 5.2 holds. By applying Theorem 5.2 on these families, we deduce that each language in them has a sample-based IPP with sub-linear query and communication complexities.

First, a main theorem from [RVW13, RR20] asserts that any language in $\mathcal{NC}$ has an IPP with sub-linear query and communication complexities:

Theorem 5.26. *Let L be a language in $\mathcal{NC}$, then there is a (query-based) IPP for L. The query and communication complexities, as well as the verifier's running time, are $\widetilde{O}(\sqrt{n})$. The IPP has poly$(\log n)$ rounds, and the (honest) prover runs in time poly(n). Moreover, this protocol uses only public-coins. The verifier needs to query the input only after the last round of the interaction, and the indices of the queries depend only on the public coins of the verifier.*

Note that it is not explicitly stated in [RVW13, RR20] that the queries the verifier makes depend only on the public coins. One can see that this indeed holds by examining the protocols in [RVW13, RR20], and [GKR08].

We can now finish the prove of Theorem 5.4:

Proof. From Theorem 5.26 we deduce that the conditions from Theorem 5.2 hold for $\mathcal{L} = \mathcal{NC}$, with query complexity, communication complexity and verifier running time $\widetilde{O}(\sqrt{n})$, and $\mathrm{poly}(\log n)$ round complexity (note that any language can be considered as a pair language, where the explicit part of the input is empty). Hence, by a direct application of Theorem 5.2 on $\mathcal{L} = \mathcal{NC}$, we get the result. □

The second family of languages we consider is of languages computable by polynomial-time and bounded-polynomial space. For this family, [RRR16] proved the following result (Theorem 3 in [RRR16]):

Theorem 5.27. *Fix any sufficiently small constant $\sigma \in (0,1)$. Let L be a language that is computable in poly(n)-time and $O(n^\sigma)$-space. Then L has a constant-round IPP with proximity parameter ϵ for $\epsilon = n^{-1/2}$ with perfect completeness and soundness $1/2$. The query and communication complexities, as well as the verifier's running time, are $n^{1/2+O(\sigma)}$. The (honest) prover runs in time poly(n). Moreover, this protocol uses only public-coins. The verifier needs to query the input only after the last round of the interaction, and the indices of the queries depend only on the public coins of the verifier.*

Note that again, it is not explicitly stated in [RRR16] that the queries the verifier makes depend only on the public coins. This can be asserted by examining the protocol. Also observe that for a fixed $\sigma \in (0,1)$, the family of languages computable in poly(n)-time and $O(n^\sigma)$-space is closed under composition with log-space uniform $\mathcal{NC}^1$ circuits.

We can now finish the prove of Theorem 5.5:

Proof. Fix $\sigma \in (0,1)$. From Theorem 5.27 we deduce that the conditions from Theorem 5.2 hold for $\mathcal{L}$, the family of languages computable in poly(n)-time and $O(n^\sigma)$-space, with query complexity, communication complexity and verifier running time $n^{1/2+O(\sigma)}$, and constant round complexity. Hence, by a direct application of Theorem 5.2 on $\mathcal{L}$, we get the result. □

Acknowledgments

We would like to thank Irit Dinur, Oded Goldreich, Shafi Goldwasser and Ron Rothblum for helpful and illuminating conversations and comments. We would also like to thank the anonymous ITCS '22 reviewers for their helpful comments.

References

[BK97] László Babai and Peter G. Kimmel. Randomized simultaneous messages: Solution of a problem of yao in communication complexity. In *Proceedings of the Twelfth Annual IEEE Conference on Computational Complexity, Ulm, Germany, June 24-27, 1997*, pages 239–246. IEEE Computer Society, 1997.

[BLR90] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 73–83. ACM, 1990.

[BM88] László Babai and Shlomo Moran. Arthur-merlin games: A randomized proof system, and a hierarchy of complexity classes. *J. Comput. Syst. Sci.*, 36(2):254–276, 1988.

[BRV18] Itay Berman, Ron D. Rothblum, and Vinod Vaikuntanathan. Zero-knowledge proofs of proximity. In Anna R. Karlin, editor, *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, volume 94 of *LIPIcs*, pages 19:1–19:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[Can15] Clément L. Canonne. A survey on distribution testing: Your data is big. but is it blue? *Electron. Colloquium Comput. Complex.*, 22:63, 2015.

[CG18] Alessandro Chiesa and Tom Gur. Proofs of proximity for distribution testing. In Anna R. Karlin, editor, *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, volume 94 of *LIPIcs*, pages 53:1–53:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[CL88] Anne Condon and Richard E. Ladner. Probabilistic game automata. *J. Comput. Syst. Sci.*, 36(3):452–489, 1988.

[CL89] Anne Condon and Richard J. Lipton. On the complexity of space bounded interactive proofs (extended abstract). In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*, pages 462–467. IEEE Computer Society, 1989.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition.* MIT Press, 2009.

[Con91] Anne Condon. Space-bounded probabilistic game automata. *J. ACM*, 38(2):472–494, 1991.

[DORS08] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam D. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 38(1):97–139, 2008.

[DS92a] Cynthia Dwork and Larry J. Stockmeyer. Finite state verifiers I: the power of interaction. *J. ACM*, 39(4):800–828, 1992.

[DS92b] Cynthia Dwork and Larry J. Stockmeyer. Finite state verifiers II: zero knowledge. *J. ACM*, 39(4):829–858, 1992.

[EKR04] Funda Ergün, Ravi Kumar, and Ronitt Rubinfeld. Fast approximate probabilistically checkable proofs. *Inf. Comput.*, 189(2):135–159, 2004.

[FGL14] Eldar Fischer, Yonatan Goldhirsh, and Oded Lachish. Partial tests, universal tests and decomposability. In Moni Naor, editor, *Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014*, pages 483–500. ACM, 2014.

[GGH$^+$07] Shafi Goldwasser, Dan Gutfreund, Alexander Healy, Tali Kaufman, and Guy N. Rothblum. Verifying and decoding in constant depth. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 440–449. ACM, 2007.

[GGL$^+$00] Oded Goldreich, Shafi Goldwasser, Eric Lehman, Dana Ron, and Alex Samorodnitsky. Testing monotonicity. *Comb.*, 20(3):301–337, 2000.

[GGR98] Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, 1998.

[GGR15] Oded Goldreich, Tom Gur, and Ron D. Rothblum. Proofs of proximity for context-free languages and read-once branching programs - (extended abstract). In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 666–677. Springer, 2015.

[Gil52] E. Gilbert. A comparison of signalling alphabets. *Bell System Technical Journal*, 31:504–522, 1952.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In Cynthia Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 113–122. ACM, 2008.

[GLR18] Tom Gur, Yang P. Liu, and Ron D. Rothblum. An exponential separation between MA and AM proofs of proximity. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 73:1–73:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[Gol98] Oded Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*, volume 17 of *Algorithms and Combinatorics*. Springer, 1998.

[Gol17] Oded Goldreich. *Introduction to Property Testing*. Cambridge University Press, 2017.

[GR16] Oded Goldreich and Dana Ron. On sample-based testers. *ACM Trans. Comput. Theory*, 8(2):7:1–7:54, 2016.

[GR17] Tom Gur and Ron D. Rothblum. A hierarchy theorem for interactive proofs of proximity. In Christos H. Papadimitriou, editor, *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*, volume 67 of *LIPIcs*, pages 39:1–39:43. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[GR18] Tom Gur and Ron D. Rothblum. Non-interactive proofs of proximity. *Comput. Complex.*, 27(1):99–207, 2018.

[GRSY21] Shafi Goldwasser, Guy N. Rothblum, Jonathan Shafer, and Amir Yehudayoff. Interactive proofs for verifying machine learning. In James R. Lee, editor, *12th Innovations in Theoretical Computer Science Conference, ITCS 2021, January 6-8, 2021, Virtual Conference*, volume 185 of *LIPIcs*, pages 41:1–41:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[GS86] Shafi Goldwasser and Michael Sipser. Private coins versus public coins in interactive proof systems. In Juris Hartmanis, editor, *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 59–68. ACM, 1986.

[KNR06] Eyal Kaplan, Moni Naor, and Omer Reingold. Derandomized constructions of k-wise (almost) independent permutations. *Electron. Colloquium Comput. Complex.*, (002), 2006.

[KR15] Yael Tauman Kalai and Ron D. Rothblum. Arguments of proximity - [extended abstract]. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 422–442. Springer, 2015.

[LFKN92] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992.

[NR05] Moni Naor and Guy N. Rothblum. The complexity of online memory checking. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*, pages 573–584. IEEE Computer Society, 2005.

[NS96] Ilan Newman and Mario Szegedy. Public vs. private coin flips in one round communication games (extended abstract). In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 561–570. ACM, 1996.

[RR19] Noga Ron-Zewi and Ron Rothblum. Local proofs approaching the witness length. *Electron. Colloquium Comput. Complex.*, page 127, 2019.

[RR20] Guy N. Rothblum and Ron D. Rothblum. Batch verification and proofs of proximity with polylog overhead. In Rafael Pass and Krzysztof Pietrzak, editors, *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part II*, volume 12551 of *Lecture Notes in Computer Science*, pages 108–138. Springer, 2020.

[RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 49–62. ACM, 2016.

[RS96] Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM J. Comput.*, 25(2):252–271, 1996.

[RVW13] Guy N. Rothblum, Salil P. Vadhan, and Avi Wigderson. Interactive proofs of proximity: delegating computation in sublinear time. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 793–802. ACM, 2013.

[Sha92] Adi Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, 1992.

[Var57] R. R. Varshamov. Estimate of the number of signals in error correcting codes. *Dokl. Akad. Nauk SSSR*, 117:739–741, 1957.

[Vol99] Heribert Vollmer. *Introduction to Circuit Complexity - A Uniform Approach.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 1999.

[Yao79] Andrew Chi-Chih Yao. Some complexity questions related to distributive computing (preliminary report). In Michael J. Fischer, Richard A. DeMillo, Nancy A. Lynch, Walter A. Burkhard, and Alfred V. Aho, editors, *Proceedings of the 11h Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1979, Atlanta, Georgia, USA*, pages 209–213. ACM, 1979.