Zero-Knowledge Proofs of Proximity*

Itay Berman
MIT

Ron D. Rothblum
MIT

Vinod Vaikuntanathan
MIT

February 11, 2017

Abstract

Interactive proofs of proximity (Ergün, Kumar and Rubinfeld, Information & Computation, 2004 and Rothblum, Vadhan and Wigderson, STOC 2013), or IPPs, are interactive proofs in which the verifier runs in time *sub-linear* in the input's length. Since the verifier cannot even read the entire input, following the property testing literature, the requirement is that she accepts inputs that are *in* the language and rejects ones that are *far* from the language. However, these proofs could (and in many cases, do) betray considerable *global* information about the input to the verifier.

In this work, we initiate the study of *zero-knowledge proofs of proximity* (ZKPP). A ZKPP convinces a sub-linear time verifier while ensuring that she learns nothing more than a few locations of the input (and the fact that the input is "close" to the language).

Our main focus is the setting of *statistical* zero-knowledge where we show that the following hold *unconditionally* (where N denotes the input size):

- Statistical ZKPPs can be sub-exponentially more efficient than property testers (or even non-interactive IPPs): We show a natural property which has a statistical ZKPP with a $\mathsf{polylog}(N)$ time verifier, but requires $\Omega(\sqrt{N})$ queries (and hence also runtime) for every property tester.
- Statistical ZKPPs can be sub-exponentially less efficient than IPPs: We show a property which has an IPP with a $\mathsf{polylog}(N)$ time verifier, but cannot have a statistical ZKPP with even an $N^{o(1)}$ time verifier.
- Statistical ZKPPs for some graph-based properties such as promise versions of expansion and bipartiteness.

Lastly, we also consider the computational setting where we show that:

- Assuming the existence of one-way functions, every language computable either in (logspace uniform) NC or in SC, has a *computational* ZKPP with a (roughly) $\sqrt{N}$ time verifier.
- Assuming the existence of collision-resistant hash functions, every language in NP has a *statistical* zero-knowledge *argument* of proximity with a $\mathsf{polylog}(N)$ verifier.

*Email: {itayberm,ronr,vinodv}@mit.edu

Contents

1 Introduction

Interactive proofs, introduced by Goldwasser, Micali and Rackoff [GMR89] are protocols that allow for a polynomial-time verifier to check the correctness of a computational statement, typically formulated as membership of an input x in a language $\mathcal{L}$, using an interactive protocol.

Interactive proofs have had an incredible impact on theoretical computer science in general, and especially on cryptography and complexity theory. However, given the vast amounts of data that are available nowadays, in some applications even polynomial running time may be too much.

Ergün, Kumar and Rubinfeld [EKR04] asked whether we can obtain proof-systems in which the verifier runs in *sub-linear* time. In particular, this means that the verifier does not even have time to read the entire input. Since it is impossible to obtain sub-linear verification in general, to obtain a meaningful notion we settle for a suitable notion of approximation. Inspired by the property testing literature [RS96, GGR98] (see also [Gol16]) a recent line of works, initiated by Rothblum, Vadhan and Wigderson [RVW13], focuses on interactive proofs in which soundness is relaxed and the verifier is only required to reject inputs that are *far* (say, in Hamming distance) from being in the language. Thus, the verifier is only assured that the input x is *close* to the language $\mathcal{L}$ and so these proof-systems are called *interactive proofs of proximity*, or IPPs for short. Recent results ([RVW13, GR16, FGL14, KR15, GGR15, RRR16, GG16, GR17]) have demonstrated that many languages admit very efficient IPPs.

One of the main advantages of classical interactive proofs is that they allow for proving statements in *zero-knowledge* [GMR89, GMW91]: amazingly, it is possible to prove that $x \in \mathcal{L}$ without revealing anything other than that. Beyond being of intrinsic interest, zero-knowledge proofs have a multitude of applications.

In this work we initiate the study of *zero-knowledge* proofs of proximity, or ZKPP for short. Specifically we ask:

Is it possible to prove correctness of a computation to a sub-linear *time verifier, so that the verifier does not learn more than it could have learned by reading a few bits from the input?*

Loosely speaking, we say that an IPP with prover P and verifier V is a ZKPP, if for any possible cheating verifier $\widehat{\mathsf{V}}$ that runs in time $t = o(N)$, where here and below N denotes the input length, there exists a simulator $\mathsf{S}_{\widehat{\mathsf{V}}}$ that runs in time roughly t and outputs a view that is indistinguishable from the view of $\widehat{\mathsf{V}}$ when interacting with P. Note that the bound on the running times for the verifier and the simulator also bounds their query complexity (i.e., the number of bits read from the input).

Interestingly, the notion of ZKPP has already implicitly appeared in the cryptographic literature 20 years ago. Bellare and Yung [BY96] noticed that the soundness of the [FLS99] construction of non-interactive zero-knowledge proof-system (NIZK) from trapdoor permutations breaks, if the cheating prover sends a description of a function that is not a permutation. [BY96] observed that to regain soundness in the [FLS99] protocol, it suffices to verify that the given function is *close* to being a permutation.

Focusing on the case that the domain of the permutation[1] is $\{0,1\}^n$, [BY96] suggested the

[1] We remark that the general case (i.e., when the domain is not $\{0,1\}^n$) introduces significant difficulties. See [GR13] and [CL17] for details.

following natural non-interactive zero-knowledge proof for certifying that a function is *close* to a permutation: the many random elements $y_1, \ldots, y_k$ in $\{0,1\}^n$ are specified as part of a common random string[2] (CRS), and the prover is required to provide inverses $x_1, \ldots, x_k$ to all of these elements. Soundness follows from the fact that if the function is far from a permutation then, with high probability, one of these elements will simply not have an inverse. Zero-knowledge is demonstrated by having the simulator sample the x's at random and obtain the y's by evaluating the permutation.

Since the verifier in the [BY96] protocol is only assured that the function is close to a permutation, in our terminology, the [BY96] protocol is a non-interactive ZKPP. Notice that the verifier runs in time $\mathsf{poly}(n)$, which is *poly-logarithmic* in the input (i.e., the truth table of f).

Knowledge Tightness. When we consider ZKPP with a poly-logarithmic verifier (as in the foregoing example), it will suffice for us to allow the simulator to run in time that is polynomial in that of the verifier (i.e., also poly-logarithmic time). However, when considering protocols were the verifier runs in time, say $\sqrt{N}$, we cannot afford such a polynomial simulation overhead.[3] Thus, following Goldreich [Gol01, Section 4.4.4.2], we will sometimes want to more precisely quantify the simulator's overhead.

1.1 Our Results

As is the case for standard zero-knowledge, the results that we can obtain depend heavily on the specific notion of zero-knowledge. These notions depend on what exactly it means that the output of the simulator is indistinguishable from a real interaction.

The main notion that we consider in this work is that of *statistical zero knowledge proofs of proximity*. Here, the requirement is that the distribution of the output of the simulator is *statistically close* to that of the real interaction.

1.1.1 Statistical ZKPP

Clearly not every IPP must necessarily be zero-knowledge.[4] Thus, the first natural question to ask is whether this notion is meaningful - do there exist languages with non-trivial statistical ZKPPs? We answer this question affirmatively. Moreover, we show that same natural problem considered by [BY96] (i.e., verifying that a function is a permutation) has a very efficient zero-knowledge proof of proximity.[5]

Theorem 1.1 (ZKPP for permutations, Informally Stated)**.** *Let* $\mathsf{PERMUTATION} = \{f : \{0,1\}^n \to \{0,1\}^n$ *such that* f *is a permutation*$\}$*. Then:*

[2] Recall that NIZKs inherently require the use of a CRS.

[3] Note that in such a case, if the simulator has a quadratic overhead, than in particular the simulator can read the entire input whereas the verifier cannot.

[4] Consider the following example, due to Fischer *et-al.* [FGL14]. Suppose that we want to check whether a given input consists of two consecutive palindromes (of possibly different lengths) or is far from such. Alon *et-al.* [AKNS00] showed that every property tester must make $\Omega(\sqrt{N})$ queries. However, if a prover provides the index that separates the two palindromes, the property becomes extremely easy to verify. Needless to say, this proof-system is blatantly not zero-knowledge.

[5] Note that protocol of [BY96] immediately yields an *honest-verifier* zero-knowledge proof of proximity. In contrast, our protocol is zero-knowledge against arbitrary cheating verifiers.

- **Property Testing Lower Bound:** *Every tester for* PERMUTATION *must make at least* $\Omega(2^{n/2})$ *queries to the input (and in particular must run in time* $\Omega(2^{n/2})$*).*
- ZKPP **Upper Bound:** PERMUTATION *has a* 4*-round statistical* ZKPP *in which the verifier runs in* poly(n) *time.*

We remark that in this result, and similarly to other results in the literature on (constant-round) statistical zero-knowledge (SZK), we can only bound the expected running time of our simulator. We also remark that Gur and Rothblum [GR15] give a lower bound on the complexity of *non-interactive* IPPs (i.e., IPP in which the entire interaction consists of a single message from the prover to the verifier, also known as MAPs) for PERMUTATION, and using that result we obtain a sub-exponential separation between the power of statistical ZKPP vs. MAPs.

Beyond the property of permutation we also consider two additional problems, both of which are graph problems and show that they admit efficient *honest-verifier* ZKPP protocols.[6] Both problems that we consider are in the bounded degree graph model, which has been widely studied in the property testing literature [GGR98, GR02].

Theorem 1.2 (Honest Verifier ZKPP for Expansion and Bipartiteness, Informally Stated)**.** *There exist* honest-verifier *statistical* ZKPP *in which the verifier's running time is* polylog(N)*, for input graphs of size* N*, for the following two promise problems:*

1. **Promise Expansion:** *Distinguish graphs with (vertex) expansion* α *from graphs that are far from even having expansion roughly* $\beta = \alpha^2/\log(N)$*.*
2. **Promise Bipartiteness:** *Distinguish bipartite graphs from graphs that are rapidly mixing and far from being bipartite.*

A few remarks are in order. We first note that the property testing complexity of both promise problems is $\tilde{\Theta}(\sqrt{N})$ [GR02, GR11, CS10, NS10, KS11]. Second, the IPP for promise-biparititeness that we use to prove Theorem 1.2 is due to [RVW13] and we merely point out that it is honest-verifier ZKPP. In contrast, the promise-expansion property above was not previously known to admit an (efficient) IPP (let alone an honest-verifier zero-knowledge one). We also remark that both of the problems in Theorem 1.2 refer to *promise problems*. In particular, we leave open the possibility of a ZKPP for bipartiteness that also handles graphs that are not rapidly mixing, and a ZKPP for expansion that accepts graphs that are α-expanding and rejects graphs that are far from α-expanding (rather than just those that are far from being $\alpha^2/\log(N)$-expanding as in Theorem 1.2). Lastly, we also leave open the possibility of extending these protocols to be statistical ZKPP against arbitrary cheating verifiers (rather than just honest verifiers).[7]

Limitations of Statistical ZKPP**.** Given these feasibility results, one may wonder whether statistical ZKPP are as powerful as IPPs. That is, can every IPP be converted to be statistically zero-knowledge with small overhead? We show that this is not the case:

[6] In such protocols, the simulator needs only to output an interaction that is indistinguishable from the interaction of the honest (original) verifier and the prover.

[7] Since honest-verifier SZK protocols can be converted to be zero-knowledge against arbitrary malicious verifiers ([GSV98], see also [Vad99]), it is reasonable to wonder whether the same holds for statistical ZKPP. We conjecture that this is the case but leave the question of verifying this conjecture to future work.

Theorem 1.3 (IPP $\not\subseteq$ SZKPP, Informally Stated)**.** *There exists a property* Π *that has an* IPP *in which the verifier runs in* $\mathsf{polylog}(N)$ *time, where* N *is the input length, but* Π *does not have a statistical* ZKPP *in which the verifier runs even in time* $N^{o(1)}$*.*

We note that Theorem 1.3 is unconditional. Interestingly, if we do allow for assumptions, then we can even separate MAP from SZKPP:

Theorem 1.4 (MAP $\not\subseteq$ SZKPP, Informally Stated)**.** *Assuming certain circuit lower bounds, there exists a property* Π *that has an* MAP *in which the verifier runs in* $\mathsf{polylog}(N)$ *time, where* N *is the input length, but* Π *does not have a statistical* ZPP *in which the verifier runs even in time* $N^{o(1)}$*.*

The circuit lower bound that we use follows from the (highly plausible) assumption that the Arthur-Merlin communication complexity of disjointness is n^{ε}, where n is the input length and $\varepsilon > 0$ is some constant.

1.1.2 The Computational Setting

As is the case in the classical setting, we can obtain much stronger results if we either (1) only require that the simulated view be *computationally* indistinguishable from the real interaction (i.e., computational zero-knowledge), or (2) only require soundness against *efficient* cheating provers (i.e., computational soundness or argument).

The following results show that under these relaxations, and assuming reasonable cryptographic assumptions, we can transform many of the known results from the literature of IPPs to be zero-knowledge. Focusing on computational zero-knowledge, we can derive such protocols for any language computable in bounded-depth or in bounded-space, where the verifier runs in roughly $\sqrt{N}$ time.

Theorem 1.5 (Computational ZKPP for Bounded Depth, Informally Stated)**.** *Assume that there exist one-way functions. Then, every language in logspace-uniform* NC*, has a computational* ZKPP*, where the verifier (and the simulator) run in time* $N^{\frac{1}{2}+o(1)}$ *and the number of rounds is* $\mathsf{polylog}(N)$*.*

Theorem 1.6 (Computational ZKPP for Bounded Space, Informally Stated)**.** *Assume that there exist one-way functions. Then, every language computable in* $\mathsf{poly}(N)$*-time and* $O(N^{\sigma})$*-space, for some sufficiently small constant* $\sigma > 0$*, has a computational* ZKPP*, where the verifier (and the simulator) run in time* $N^{\frac{1}{2}+O(\sigma)}$*.*

Interestingly, if we only require *computational soundness*, we can do even better. The following result gives *statistical* zero-knowledge arguments of proximity for every language in NP, and with a verifier that runs in *poly-logarithmic* time.

Theorem 1.7 (Statistical Zero-Knowledge Arguments for NP, Informally Stated)**.** *Assume that there exist collision-resistant hash functions. Then, every language in* NP*, has a constant-round* statistical *zero-knowledge argument of proximity, where the verifier runs in time* $\mathsf{polylog}(N)$*.*

We note that Theorems 1.5 to 1.7 strongly rely on (1) results from the literature on IPPs [RVW13, RRR16] and interactive arguments of proximity [Kil92, BGH+06, DR06], (2) a method introduced by [BGG+88] for transforming interactive proofs (and arguments) into zero-knowledge ones (while taking some additional care that is not required in the classical setting), and (3) the observation that the verifiers in many of the underlying protocols all make queries that do not depend on messages sent by the prover. See Section 6 for details.

1.2 Additional Related Works

A related notion of *zero-knowledge* PCPs *of* proximity was recently considered by Ishai and Weiss [IW14]. These are PCP systems in which, the verifier gets oracle access to both the input and to an alleged proof. Similarly to our notion of ZKPP, the verifier runs in sublinear and is assured (with high probability) that the input is close to the language. Here, zero-knowledge means that the verifier learns nothing more than what it could simulate by making few queries to the input. We emphasize that the difference between our model and that of [IW14] is that we consider *interactice* proofs, whereas [IW14] focus on PCP-style proofs: namely soundness is guaranteed only if the PCP proof string is written in advance.

A recent work by Ben-Sasson *et-al.* [BCF+16] studies zero-knowledge interactive oracle proofs - in a model in which the verifier receives *oracle* access to the communication tape, but full access to the input.[8] Our model of ZKPP is reversed - the verifier has oracle access to the input but full access to the communication tape.

Organization. General notations and definitions used throughout the paper are given in Section 2. The model of zero-knowledge proofs of proximity (ZKPP) is defined in Section 3. Our statistical ZKPP protocols for Permutations, Expansion and Bipartiteness, are presented and analyzed in Section 4, while our lower bounds for statistical ZKPP are in Section 5. Finally, in Section 6 we present our results on computational ZK proofs of proximity and the statistical ZK arguments of proximity.

2 Preliminaries

We use calligraphic letters to denote sets, uppercase for random variables, lowercase for values and functions, boldface for vectors, and uppercase sans-serif (e.g., A) for algorithms (i.e., Turing Machines). All logarithms considered here are in base two. Given a random variable X, we write $x \leftarrow X$ to indicate that x is selected according to X. Similarly, given a finite set $\mathcal{S}$, we let $s \leftarrow \mathcal{S}$ denote that s is selected according to the uniform distribution on $\mathcal{S}$. For an interactive protocol (A, B), let $\mathrm{out}(\mathsf{A}, \mathsf{B})$ denote a random variable induced by B's output in a random execution of (A, B) (usually, A will be some prover and B will be an honest verifier).

The relative distance, over alphabet Σ, between two strings $x \in \Sigma^n$ and $y \in \Sigma^n$ is defined by $\Delta(x,y) := \frac{|\{x_i \neq y_i \colon i \in [n]\}|}{n}$. If $\Delta(x,y) \leq \varepsilon$, we say that x is ε-close to y, and otherwise we say that x is ε-far from y. Similarly, we define the relative distance of x from a non-empty set $S \subseteq \Sigma^n$ by $\Delta(x,S) := \min_{y \in S} \Delta(x,y)$. If $\Delta(x,y) \leq \varepsilon$, we say that x is ε-close to S, and otherwise we say that x is ε-far from S. The bitwise exclusive-or between two binary strings $x, y \in \{0,1\}^n$ is denoted by $x \oplus y$. The statistical distance between two distributions P and Q over a finite set $\mathcal{U}$, is defined as $\mathrm{SD}(P,Q) := \max_{\mathcal{S} \subseteq \mathcal{U}} |P(\mathcal{S}) - Q(\mathcal{S})| = \frac{1}{2} \sum_{u \in \mathcal{U}} |P(u) - Q(u)|$ and their product distribution is denoted by $P \otimes B$.

The image of a function $f \colon \mathcal{X} \to \mathcal{Y}$ is defined as $\mathrm{Im}(f) = \{y \in \mathcal{Y} \colon \exists x \in \mathcal{X}, f(x) = y\}$. An additional notation that we will use is that if $S = (S_k)_{k \in \mathbb{N}}$ and $T = (T_k)_{k \in \mathbb{N}}$ are ensembles of sets, we denote by $S \subseteq T$ the fact that $S_k \subseteq T_k$ for every $k \in \mathbb{N}$.

[8] Interactive proofs in which the verifier is not charged for reading the entire communication tape are called either *probabilistically checkable interactive proofs* [RRR16] or *interactive oracle proofs* [BCS16] in the literature.

2.1 Hashing and Entropy

2.1.1 Entropy

Definition 2.1 (Entropy)**.** *The* entropy *of a discrete random variable X is defined as*

$$\mathrm{H}(X) := \mathrm{E}_{x \leftarrow X}\left[\log\left(\frac{1}{\Pr[X=x]}\right)\right].$$

The binary entropy function *$h\colon [0,1] \to [0,1]$ is defined to be the entropy of $X \sim$ Bernoulli(p), that is, $h(p) = -p\log(p) - (1-p)\log(1-p)$, where we use the convention that $h(0) = h(1) = 0$.*

Another notion of entropy that we shall use is that that of (conditional) average min-entropy.

Definition 2.2 (average min-entropy [DORS08])**.** *Let X, Y be jointly distributed random variables. The* average min-entropy *of X given Y is defined by*

$$\tilde{\mathrm{H}}_\infty(X|Y) := -\log\Big(\mathrm{E}_{y \leftarrow Y}\Big[\max_x \Pr[X=x \mid Y=y]\Big]\Big).$$

The following fact follows immediately from the above definition.

Fact 2.3. *Let X^n, Y^n be n-tuples of independent copies of the random variables X and Y respectively. Then $\tilde{\mathrm{H}}_\infty(X^n|Y^n) = n \cdot \tilde{\mathrm{H}}_\infty(X|Y)$.*

Proof. We prove for the case that $n = 2$. The general case follows by induction.

$$\begin{aligned}
\tilde{\mathrm{H}}_\infty(X^2|Y^2) &= -\log\left(\mathrm{E}_{(y_1,y_2)\leftarrow Y^2}\left[\max_{x_1,x_2} \Pr\big[X^2=(x_1,x_2) \mid Y^2=(y_1,y_2)\big]\right]\right) \\
&= -\log\left(\mathrm{E}_{(y_1,y_2)\leftarrow Y^2}\left[\max_{x_1,x_2} \Pr[X=x_1 \mid Y=y_1]\cdot\Pr[X=x_2 \mid Y=y_2]\right]\right) \\
&= -\log\left(\mathrm{E}_{(y_1,y_2)\leftarrow Y^2}\left[\max_{x_1} \Pr[X=x_1 \mid Y=y_1]\cdot\max_{x_2}\Pr[X=x_2 \mid Y=y_2]\right]\right),
\end{aligned}$$

where the second inequality follows since the first sample from (X,Y) is independent from the second one, and thre third inequality follows since for non-negative functions f, g, it holds that $\max_{x_1,x_2} f(x_1)\cdot g(x_2) = \max_{x_1} f(x_1)\cdot\max_{x_2} g(x_2)$. Letting $h(y) = \max_x \Pr[X=x \mid Y=y]$, we write

$$\begin{aligned}
\tilde{\mathrm{H}}_\infty(X^2|Y^2) &= -\log\big(\mathrm{E}_{(y_1,y_2)\leftarrow Y^2}[h(y_1)\cdot h(y_2)]\big) \\
&= -\log(\mathrm{E}_{y_1\leftarrow Y}[h(y_1)]\cdot\mathrm{E}_{y_2\leftarrow Y}[h(y_2)]) \\
&= -\log(\mathrm{E}_{y_1\leftarrow Y}[h(y_1)]) - \log(\mathrm{E}_{y_2\leftarrow Y}[h(y_2)]) \\
&= \tilde{\mathrm{H}}_\infty(X|Y) + \tilde{\mathrm{H}}_\infty(X|Y),
\end{aligned}$$

where the second inequality follows since the first sample of Y is independent of the second one. □

2.1.2 Hashing

Definition 2.4 (pairwise independent hash functions)**.** *A family of functions $\mathcal{H} = \{h : [N] \to [M]\}$ is* pairwise independent *if for every $x_1 \neq x_2 \in [N]$ and every $y_1, y_2 \in [M]$, it holds that*

$$\Pr_{h \leftarrow \mathcal{H}}[h(x_1) = y_1 \wedge h(x_2) = y_2] = \frac{1}{M^2}.$$

The existence of efficient pairwise independent hash functions is well known.

Fact 2.5 (c.f. [Vad12, Theorem 3.26])**.** *For every $n, m \in \mathbb{N}$, there exists a family of pairwise independent hash functions $\mathcal{H}_{n,m} = \{h \colon \{0,1\}^n \to \{0,1\}^m\}$ where a random function from $\mathcal{H}_{n,m}$ can be selected using $\max(m,n)+m$ bits, and given a description of $h \in \mathcal{H}_{n.m}$ and $x \in \{0,1\}^n$, the value $h(x)$ can be evaluated in time* $\mathsf{poly}(n,m)$.

Dodis *et-al.* [DORS08], showed the following generalization of the leftover hash lemma, for sources having high *conditional* min-entropy.

Lemma 2.6 (generalized leftover hash lemma [DORS08, Lemma 2.4])**.** *Let $\mathcal{H} = \{h \colon \{0,1\}^n \to \{0,1\}^m\}$ be a family of pairwise independent hash functions. Then, for any random variables X and Y and the random variable $H \leftarrow \mathcal{H}$, it holds that*

$$\mathrm{SD}\Big(\big(H(X), H, Y\big), \big(U_m, H, Y\big)\Big) \leq \frac{1}{2} \cdot \sqrt{2^{-\tilde{\mathrm{H}}_\infty(X|Y)} \cdot 2^m}\ ,$$

where U_m is distributed uniformly over $\{0,1\}^m$.

2.2 Statistical Zero-Knowledge

We use standard definitions and results from the literature of statistical zero-knowledge proofs, based mainly on [Vad99].

2.2.1 Statistical Zero-Knowledge Interactive Proofs

In this section we give the (almost) standard definitions for interactive proofs and honest-verifier zero-knowledge proofs.

Definition 2.7 (interactive proofs)**.** *Let $\Pi = (\Pi_{\mathsf{YES}}, \Pi_{\mathsf{NO}})$ be a promise problem. An* interactive proof system *for Π is an interactive protocol (P, V) with* completness error *$c \colon \mathbb{N} \to [0,1]$ and* soundness error *$s \colon \mathbb{N} \to [0,1]$ if the following holds for every security parameter $k \in \mathbb{N}$:*

- **Completeness***: If $x \in \Pi_{\mathsf{YES}}$, then, when $\mathsf{V}(x,k)$ interacts with $\mathsf{P}(x,k)$, with probability $1 - c(k)$ it accepts.*
- **Soundness***: If $x \in \Pi_{\mathsf{NO}}$, then for every prover strategy $\widehat{\mathsf{P}}$, when $\mathsf{V}(x,k)$ interacts with $\widehat{\mathsf{P}}$, with probability $1 - s(k)$ it rejects.*

If $c(\cdot)$ and $s(\cdot)$ are negligible functions, we say that (P, V) is an interactive proof system.

The above definition does not deal with the efficiency of the verifier or the prover, unlike the standard definition that requires the verifier to run in time $\mathsf{poly}(|x|, k)$. Jumping ahead, this is because in our settings the verifier (and also the simulator) will have only oracle-access to their inputs. We will refer to the efficiency of those algorithms within theorem statements.

Definition 2.8 (view of interactive protocol)**.** *Let* (P,V) *be an* r*-message interactive protocol. The* view *of* V *on a common input* x *(given as standard input or by oracle access to either of the parties) is defined by* $\mathrm{view}_{\mathsf{P},\mathsf{V}}(x) := (m_1, m_2, \ldots, m_r; \rho)$*, where* $m_1, m_2, \ldots, m_r$ *are the messages sent by the parties in a random execution of the protocol, and* ρ *contains of all the random coins* V *used during this execution.*

We allow probabilistic algorithms to fail by outputting $\perp$. An algorithm A is useful if $\Pr[\mathsf{A}(x) = \perp] \leq 1/2$ for every x, and let $\widetilde{\mathsf{A}}(x)$ denote the output distribution of $\mathsf{A}(x)$, conditioning on $\mathsf{A}(x) \neq \perp$.

Definition 2.9 (honest-verifier zero-knowledge proofs)**.** *Let* $\Pi = (\Pi_{\mathsf{YES}}, \Pi_{\mathsf{NO}})$ *be a promise problem. An interactive proof* (P,V) *for* Π *is said to be* honest-verifier statistical zero-knowledge*, if there exists an algorithm* S*, and a negligible function* $\mu\colon \mathbb{N} \to [0,1]$ *such that for every* $k \in \mathbb{N}$ *and* $x \in \Pi_{\mathsf{YES}}$*,*

$$\mathrm{SD}\Big(\widetilde{\mathsf{S}}(x,k), \mathrm{view}_{\mathsf{P},\mathsf{V}}(x,k)\Big) \leq \mu(k).$$

2.2.2 Statistical Distance, Entropy Difference and Sample-Access Proofs

Central in the study of statistical zero-knowledge are problems dealing with properties of distributions encoded by circuits.

Definition 2.10 (distributions encoded by circuits)**.** *Let* X *be a Boolean circuit with* m *input gates and* n *output gates. The* distribution encoded by X *is the distribution induced on* $\{0,1\}^n$ *by evaluating* X *on a uniformly selected string from* $\{0,1\}^m$*. By abuse of notation, we also write* X *for the distribution defined by the circuit* X*.*

Two particularly interesting problems are *statistical distance* and *entropy difference*.

Definition 2.11 (Statistical Distance)**.** *For any constants* $0 \leq \beta \leq \alpha \leq 1$*, the promise problem* $\mathsf{SD}^{\alpha,\beta} = (\mathsf{SD}^{\alpha,\beta}_{\mathsf{YES}}, \mathsf{SD}^{\alpha,\beta}_{\mathsf{NO}})$ *is given by*

$$\begin{aligned}\mathsf{SD}^{\alpha,\beta}_{\mathsf{YES}} &= \{(X,Y)\colon\ \mathrm{SD}(X,Y) \geq \alpha\}\\ \mathsf{SD}^{\alpha,\beta}_{\mathsf{NO}} &= \{(X,Y)\colon\ \mathrm{SD}(X,Y) \leq \beta\}.\end{aligned}$$

Above, X, Y *are distributions encoded by circuits according to Definition 2.10.*

Definition 2.12 (Entropy Difference)**.** Entropy Difference *is the promise problem* $\mathsf{ED} = (\mathsf{ED}_{\mathsf{YES}}, \mathsf{ED}_{\mathsf{NO}})$*, where*

$$\begin{aligned}\mathsf{ED}_{\mathsf{YES}} &= \{(X,Y)\colon\ \mathrm{H}(X) \geq \mathrm{H}(Y) + 1\},\\ \mathsf{ED}_{\mathsf{NO}} &= \{(X,Y)\colon\ \mathrm{H}(Y) \geq \mathrm{H}(X) + 1\}.\end{aligned}$$

Above, X, Y *are distributions encoded by circuits according to Definition 2.10.*

Both SD and ED are known to be complete for the class of problems that have statistical zero-knowledge proofs (see [Vad99, Theorem 3.5.1]). A fact that we will rely on heavily, is that the zero-knowledge proof-systems for SD and ED only require *sample* access to the distributions induced by the input circuits. That is, neither the verifier not the simulator in these proof-systems need to actually look at the circuits themselves. Rather, all that they need is the ability to generate random samples from the circuits.

Definition 2.13 (sample-access honest-verifier zero-knowledge proof)**.** *Let Π be a promise problem whose instances are pairs of distributions encoded by circuits. An honest-verifier zero-knowledge proof system for Π is* sample-access *if both the verifier and the simulator only require oracle access to random samples from the distributions encoded by the input circuits (in addition to explicitly getting the security parameter k).*

We can now state the results regarding the zero-knowledge proof systems of SD and ED. In fact, we will not care about SD but rather about *statistical closeness*, the complement of SD in which the YES and NO instances are switched, namely $\overline{\mathsf{SD}^{\alpha,\beta}} := \left(\mathsf{SD}^{\alpha,\beta}_{\mathsf{NO}}, \mathsf{SD}^{\alpha,\beta}_{\mathsf{YES}}\right)$.

Lemma 2.14. *Let $0 \leq \beta \leq \alpha \leq 1$ be constants such that*[9] *$h((1+\alpha)/2) < 1 - \beta$. Then, there exists a 2-message* sample-access *honest-verifier statistical zero-knowledge proof for $\overline{\mathsf{SD}^{\alpha,\beta}}$. Moreover, the running times of the verifier and the simulator in the above protocol given sample access to (X, Y) and security parameter k are $\mathsf{poly}(m, n, k)$, where m is the number of random coins needed to sample from X or Y (i.e., their input size) and n is the output size of X and Y.*

The protocol establishing Lemma 2.14 reduces, in a black-box way, an instance of $\overline{\mathsf{SD}^{\alpha,\beta}}$ to ED (see [Vad99, Section 4.4]) and then uses the next lemma.

Lemma 2.15. *There exists 2-message* sample-access *honest-verifier statistical zero-knowledge proof for* ED*. Moreover, the running times of the verifier and the simulator in the above protocol given sample access to (X, Y) and security parameter k are $\mathsf{poly}(m, n, k)$, where m is the number of random coins needed to sample from X or Y (i.e., their input size) and n is the output size of X and Y.*

3 ZKPP — Model and Definitions

In this section we formally define the model of *statistical* zero-knowledge proofs of proximity. We follow definition choices from the literature of classical statistical zero-knowledge proofs, mainly based on [Vad99] (see Section 2.2.2). For a discussion about the computational setting, see Remark 3.7 below.

Properties will be identified as sets of functions. A propery is an ensemble $\Pi = (\Pi_n, \mathcal{D}_n, \mathcal{R}_n)_{n\in\mathbb{N}}$, where $\Pi_n \subseteq \mathcal{F}_{\mathcal{D}_n \to \mathcal{R}_n}$ for every $n \in \mathbb{N}$, letting $\mathcal{F}_{\mathcal{D}\to\mathcal{R}}$ denote the set of all functions from domain $\mathcal{D}$ to range $\mathcal{R}$. Equivalently, we sometimes view Π_n as a string of length $|\mathcal{D}_n|$ over the alphabet $\mathcal{R}_n$ and write $\Pi_n \subseteq \mathcal{R}_n^{|\mathcal{D}_n|}$. For a property Π and $n \in \mathbb{N}$, let $N_\Pi(n) := |\mathcal{D}_n| \cdot \log(|\mathcal{R}_n|)$ denote the input-size of Π_n. Throughout this paper we remove Π and n from the above notation, and simply let N denote the input-size of the relevant property (this will be convenient when defining complexity classes; see ahead). We note that, depending on the context, we will sometimes refer to properties as languages. Lastly, similar to [Vad99], we use a security parameter to control our soundness and zero-knowledge guarantees (see Remark 3.6 for additional details).

Definition 3.1 (interactive proofs of proximity (IPP))**.** *An r-message* interactive proof of proximity *(IPP), with respect to proximity parameter $\varepsilon > 0$, (in short, ε-IPP) for the property $\Pi = (\Pi_n, \mathcal{D}_n, \mathcal{R}_n)_{n\in\mathbb{N}}$ is an interactive protocol (P, V) between a prover P, which gets* free *access to an input $f : \mathcal{D}_n \to \mathcal{R}_n$ as well as to ε, $|\mathcal{D}_n|$, $|\mathcal{R}_n|$ and k, and a verifier V, which gets* oracle *access to f as well as free access to ε, n, $|\mathcal{D}_n|$, $|\mathcal{R}_n|$ and k. The following conditions are satisfied at the end of the protocol for every $k \in \mathbb{N}$ and large enough $n \in \mathbb{N}$:*

[9] Recall that we use h to denote the binary entropy function $h(p) = -p\log(p) - (1-p)\log(1-p)$.

- **Completeness**: *If $f \in \Pi_n$, then, when* V *interacts with* P*, with probability* $1 - \mathsf{negl}(k)$ *it accepts.*
- **Soundness**: *If f is ε-far from Π_n, then for every prover strategy* $\widehat{\mathsf{P}}$*, when* V *interacts with* $\widehat{\mathsf{P}}$*, with probability* $1 - \mathsf{negl}(k)$ *it rejects.*

For $t = t(n, |\mathcal{D}_n|, |\mathcal{R}_n|, k, \varepsilon)$, $\mathsf{IPP}[t]$ *denotes the class of properties possessing ε-*IPP *in which the verifier's running time is at most $O(t)$. Finally, for a class of functions $\mathcal{C}$, we denote by* $\mathsf{IPP}[\mathcal{C}(n, |\mathcal{D}_n|, |\mathcal{R}_n|, k, \varepsilon)]$ *the class of properties Π for which there exists $t \in \mathcal{C}$ such that $\Pi \in \mathsf{IPP}[t]$.*

The probabilities that the verifier rejects in the completeness condition and accepts in the soundness condition are called the completeness error and soundness error, respectively. If the completeness condition holds with probability 1, then we say that the IPP has perfect completeness. A public-coin IPP is an IPP in which every message from the verifier to the prover consists only of fresh random coin tosses.

An IPP is said to have query complexity $q\colon \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times (0,1] \to \mathbb{N}$ if for every $n, k \in \mathbb{N}$, $\varepsilon > 0$, $x \in \mathcal{F}_{\mathcal{D}_n \to \mathcal{R}_n}$, and any prover strategy $\widehat{\mathsf{P}}$, the verifier V makes at most $q(n, |\mathcal{D}_n|, |\mathcal{R}_n|, k, \varepsilon)$ queries to x when interacting with $\widehat{\mathsf{P}}$. The IPP is said to have communication complexity $c\colon \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times (0,1] \to \mathbb{N}$ if for every $n, k \in \mathbb{N}$, $\varepsilon > 0$, and $x \in \mathcal{F}_{\mathcal{D}_n \to \mathcal{R}_n}$ the communication between V and P consists of at most $c(n, |\mathcal{D}_n|, |\mathcal{R}_n|, k, \varepsilon)$ bits.

Our main (but not exclusive) focus in this work is on properties that have IPPs in which the verifier's running time (and thus also the communication and query complexities) is poly-logarithmic in the input size and polynomial in the security parameter k and in the reciprocal of the proximity parameter $1/\varepsilon$, that is, the class $\mathsf{IPP}[\mathsf{poly}(\log(N), k, 1/\varepsilon)]$.

An IPP that consists of a single message sent from the prover (Merlin) to the verifier (Arthur) is called Merlin-Arthur proof of proximity (MAP). We extend all the above notations to Merlin-Arthur proofs of proximity in the natural way.

Before defining general ZKPPs, we first consider zero-knowledge with respect to *honest verifiers*.

Definition 3.2 (honest-verifier zero-knowledge proof of proximity (HV-SZKPP, HV-PZKPP))**.** *Let (P, V) be an interactive proof of proximity for a property $\Pi = (\Pi_n, \mathcal{D}_n, \mathcal{R}_n)_{n \in \mathbb{N}}$. The protocol (P, V) is said to be* honest-verifier statistical zero-knowledge with simulation overhead *s, for some function $s\colon \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times (0,1] \to \mathbb{N}$ if there exists a useful[10] probabilistic algorithm S, which (like V) gets oracle access to $f\colon \mathcal{D}_n \to \mathcal{R}_n$ as well as free access to ε, n, $|\mathcal{D}_n|$, $|\mathcal{R}_n|$ and k, and whose running time is at most $O(s(t_{\mathsf{V}}, n, |\mathcal{D}_n|, |\mathcal{R}_n|, k, \varepsilon))$, where $t_{\mathsf{V}}(n, |\mathcal{D}_n|, |\mathcal{R}_n|, k, \varepsilon)$ is V's running time, such that for every $k \in \mathbb{N}$, every large enough $n \in \mathbb{N}$ and $f\colon \mathcal{D}_n \to \mathcal{R}_n$, if $f \in \Pi_n$, then*

$$\mathrm{SD}\Big(\widetilde{\mathsf{S}}^f(\varepsilon, n, |\mathcal{D}_n|, |\mathcal{R}_n|, k), \mathrm{view}_{\mathsf{P},\mathsf{V}}(\varepsilon, n, |\mathcal{D}_n|, |\mathcal{R}_n|, k, f)\Big) \leq \mathsf{negl}(k).$$

If the $\mathsf{negl}(k)$ can be replaced with 0 in the above equation, (P, V) is said to be honest-verifier perfect zero-knowledge with simulation overhead *s.*

For $t = t(n, |\mathcal{D}_n|, |\mathcal{R}_n|, k, \varepsilon)$, $\mathsf{HV\text{-}SZKPP}[t, s]$ *(resp.,* $\mathsf{HV\text{-}PZKPP}[t, s]$*) denotes the class of properties possessing honest-verifier statistical (resp., perfect) zero-knowledge proof of proximity with simulation overhead s in which the verifier's running time is at most $O(t)$.*

[10] Recall that an algorithm A is useful if $\Pr[\mathsf{A}(x) = \perp] \leq 1/2$ for every x, and that $\widetilde{\mathsf{A}}(x)$ denote the output distribution of $\mathsf{A}(x)$, conditioning on $\mathsf{A}(x) \neq \perp$.

We say that the query complexity of a simulator S is $q' \colon \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times (0,1] \to \mathbb{N}$ if for every $n, k \in \mathbb{N}$, $\varepsilon > 0$, $f \in \mathcal{F}_{\mathcal{D}_n \to \mathcal{R}_n}$, S_n^f makes at most $q'(n, |\mathcal{D}_n|, |\mathcal{R}_n|, k, \varepsilon)$ queries to f.

A typical setting (that we will focus on) is when the verifier's running time is $\mathsf{poly}(\log(N), k, 1/\varepsilon)$, namely poly-logarithmic in the input length N and polynomial in the security parameter k and in the proximity parameter $1/\varepsilon$. In this setting we often allow for *polynomial* simulation overhead, that is the simulator's running time is also $\mathsf{poly}(\log(N), k, 1/\varepsilon)$. Specifically, we denote by $\mathsf{HV\text{-}SZKPP}\big[\mathsf{poly}(\log(N), k, 1/\varepsilon)\big]$ the class of properties $\Pi \in \mathsf{HV\text{-}SZKPP}[t, s]$ for $t = \mathsf{poly}(\log(N), k, 1/\varepsilon)$ and $s = \mathsf{poly}(t, \log(N), k, 1/\varepsilon)$. The class $\mathsf{HV\text{-}PZKPP}\big[\mathsf{poly}(\log(N), k, 1/\varepsilon)\big]$ is similarly defined.

Another setting of interest is when the verifier's running time is $N^\delta \cdot \mathsf{poly}(k, 1/\varepsilon)$, for some constant $\delta \in (0,1)$. In this setting, unlike the previous one, allowing the simulation overhead to be polynomial will give the simulator much greater computational power than the verifier (e.g., if $\delta = 1/2$ and s is quadratic in the verifier's running time, then the simulator can run in time $O(N)$ and in particular may read the entire input). In this setting we aim for the simulation overhead to be *linear* in the verifier's running time (but it can be polynomial in k and $1/\varepsilon$).[11]

When the simulation overhead is clear from context (as it will almost always be the case) we omit it from the notation.

Cheating Verifier ZKPP**.** We will allow cheating verifiers to be non-uniform by giving them an auxiliary input. For an algorithm A and a string $z \in \{0,1\}^*$ (all auxiliary inputs will be binary strings, regardless of the properties' alphabet), let $\mathsf{A}_{[z]}$ be A when z was given as auxiliary input. Since we care about algorithms whose running time is insufficient to read the entire input, we would not want to allow the running time to depend on the auxiliary input (otherwise, we could artificially inflate z so that A would be able to read the entire input). Thus, following [Vad99], we adopt the convention that the running time of A is independent of z, so if z is too long, A will not be able to access it in its entirety.

Definition 3.3 (cheating-verifier zero-knowledge proof of proximity (SZKPP, PZKPP))**.** *Let (P, V) be an interactive proof of proximity for a property $\Pi = (\Pi_n, \mathcal{D}_n, \mathcal{R}_n)_{n \in \mathbb{N}}$. (P, V) is said to be* cheating-verifier statistical zero-knowledge with simulation overhead *s, for some function $s \colon \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times (0,1] \to \mathbb{N}$, if for every algorithm $\widehat{\mathsf{V}}$ whose running time is $O(t_{\widehat{\mathsf{V}}}(n, |\mathcal{D}_n|, |\mathcal{R}_n|, k, \varepsilon))$, there exists a useful probabilistic algorithm S, which (like $\widehat{\mathsf{V}}$) gets oracle access to $f \colon \mathcal{D}_n \to \mathcal{R}_n$ as well as free access to ε, n, $|\mathcal{D}_n|$, $|\mathcal{R}_n|$ and k, and whose running time is at most $O(s(t_{\widehat{\mathsf{V}}}, n, |\mathcal{D}_n|, |\mathcal{R}_n|, k, \varepsilon))$, such that for every $k \in \mathbb{N}$, large enough $n \in \mathbb{N}$, $z \in \{0,1\}^*$ and $f \colon \mathcal{D}_n \to \mathcal{R}_n$, if $f \in \Pi_n$, then*

$$\mathrm{SD}\Big(\widetilde{\mathsf{S}}^f_{[z]}(\varepsilon, n, |\mathcal{D}_n|, |\mathcal{R}_n|, k), \mathrm{view}_{\mathsf{P}, \widehat{\mathsf{V}}_{[z]}}(\varepsilon, n, |\mathcal{D}_n|, |\mathcal{R}_n|, k, f)\Big) \leq \mathsf{negl}(k).$$

If the $\mathsf{negl}(k)$ can be replaced with 0 in the above equation, (P, V) is said to be resp., cheating-verifier perfect zero-knowledge with simulation overhead *s.*

For $t = t(n, |\mathcal{D}_n|, |\mathcal{R}_n|, k, \varepsilon)$, $\mathsf{SZKPP}[t, s]$ (resp., $\mathsf{PZKPP}[t, s]$) denotes the class of properties possessing cheating-verifier statistical (resp., perfect) zero-knowledge proof of proximity with simulation overhead s in which the verifier's running time is at most $O(t)$.

Expected Simulation Overhead. Definition 3.3 requires that the running time of the simulator always be bounded. Similarly to many results in the ZK literature, in some cases we can only bound the simulator's *expected* running time.

[11] This requirement is in the spirit of *constant* knowledge tightness, see [Gol01, Section 4.4.4.2].

Definition 3.4 (cheating-verifier ZKPP with expected simulation (ESZKPP, EPZKPP)). *Let* (P,V) *be an interactive proof of proximity for a property* $\Pi = (\Pi_n, \mathcal{D}_n, \mathcal{R}_n)_{n\in\mathbb{N}}$. *The protocol* (P,V) *is said to be* cheating-verifier statistical zero-knowledge with expected simulation overhead s *if it satisfies the same requirement as in Definition 3.3 except that we only bound the* expected *running time of the simulator.*

The classes $\mathsf{ESZKPP}[t,s]$ *and* $\mathsf{EPZKPP}[t,s]$ *are defined analogous to* $\mathsf{SZKPP}[t,s]$ *and* $\mathsf{PZKPP}[t,s]$ *from Definition 3.3.*

Unless explicitly saying otherwise, all zero-knowledge protocols we discuss are cheating-verifier ones.

As in the honest-verifier case, a typical setting is that in which the verifier's running time is poly-logarithmic in the input size N and polynomial in the security parameter k and in $1/\varepsilon$, and the simulator's (possibly only expected and not strict) running time is polynomial in the running time of the cheating-verifier that it simulates, poly-logarithmic in N and polynomial in k and $1/\varepsilon$. Specifically, if we allow the cheating-verifier the same computational powers as the honest-verifier, then both the honest-verifier and every simulator run in time $\mathsf{poly}(\log(N),k,1/\varepsilon)$. We let $\mathsf{ESZKPP}\left[\mathsf{poly}(\log(N),k,1/\varepsilon)\right]$ be the class of properties $\Pi \in \mathsf{ESZKPP}[t,s]$ for $t = \mathsf{poly}(\log(N),k,1/\varepsilon)$ and $s = \mathsf{poly}(t_{\widehat{\mathsf{V}}},\log(N),k,1/\varepsilon)$. The class $\mathsf{PZKPP}\left[\mathsf{poly}(\log(N),k,1/\varepsilon),\mathsf{poly}\right]$ is similarly defined.

We conclude this section with a few remarks on the model and the above definitions.

Remark 3.5 (Promise Problems). *Some of the protocols that we construct do not refer to a property but rather to a* promise problem. *More specifically, rather than distinguishing between inputs that are in the property* Π *for those that are* ε*-far from* Π*, we will consider a promise problem* $(\Pi_{\mathsf{YES}},\Pi_{\mathsf{NO}})$ *and the requirement is that the verifier accepts inputs that are in* Π_{YES} *and rejects inputs that are both in* Π_{NO} and *are* ε*-far from* Π_{YES}. *We extend the definitions above to handle such promise problems in the natural way.*

Remark 3.6 (The Security Parameter). *One of the original motivations to include security parameter in the classical definitions of statistical zero-knowledge proofs was to control the error parameters (completeness, soundness and simulation deviation) independently from the input's length. Specifically, one may want to provide a high-quality proof (i.e., very small errors) for short inputs (see [Vad99, Section 2.4]). In our setting, the situation is somehow reversed. We think of very large inputs that the verifier and simulators cannot even entirely read. Hence, it is infeasible to ask them for errors that are negligible in the input's length. Instead, we control the quality of the proof with the security parameter, independent of the input's length.*

Remark 3.7 (Computational ZKPP). *Since our focus is on the statistical case, we do not provide explicit definitions of computational zero-knowledge proofs of proximity. Indeed, these definitions can be easily interpolated from the statistical ones in a standard way (see for example Vadhan's [Vad99, Section 2] definition of computational zero-knowledge). Specifically, in the computational definitions one simply replaces the requirement that the simulator's output and the protocol's view are statistically-close with one in which they are computationally indistinguishable.*

4 The Power of ZKPP: The Statistical Case

4.1 ZKPP for Permutations

In this section, we look at functions $f : \{0,1\}^n \to \{0,1\}^n$ and consider the property of being a permutation. That is, we would like to distinguish between functions that are a permutation from

those that are *far* from being a permutation.

Definition 4.1 (The permutation property)**.** *For every $n \in \mathbb{N}$ let*

$$\mathsf{PERMUTATION}_n = \left\{ f \colon \{0,1\}^n \to \{0,1\}^n \;\middle|\; f \text{ is a permutation} \right\}.$$

We define the permutation property as $\mathsf{PERMUTATION} = \left(\mathsf{PERMUTATION}_n, \{0,1\}^n, \{0,1\}^n\right)_{n\in\mathbb{N}}$.

It is not difficult to see that any *property tester* for PERMUTATION must make at least $\Omega(\sqrt{N})$ queries, where $N = 2^n$. To see this, consider the following two distributions: (1) a random permutation over $\{0,1\}^n$; and (2) a random function from $\{0,1\}^n$ to $\{0,1\}^n$. The first distribution is supported exclusively on YES instances whereas it can be shown that the second is, with high probability, far from a permutation. However, if a tester makes $q \ll \sqrt{N}$ queries, then in both cases, with high probability, its view will be the same: q distinct random elements. The property testing lower bound follows.

As a matter of fact, Gur and Rothblum [GR15] have shown that the verifier in every MAP (i.e., non-interactive proof of proximity, see [GR16]) for PERMUTATION must make either $\Omega(N^{1/4})$ queries or use a proof of length $\Omega(N^{1/4})$.

In this section, we show that the PERMUTATION property has a 4-message (statistical) zero-knowledge proof of proximity with respect to *cheating* verifiers. We note that we only bound the *expected* number of queries and running time of the simulator of our protocol. We leave it as an open problem to obtain a protocol (possibly with more rounds of interaction) in which one can show a *high probability bound* on the query complexity and running times of the simulator.

Before stating the theorem a word on notation. In Section 3 we gave the prover and the verifier, as explicit inputs, the domain and range sizes — both, in the case of the permutation property, are 2^n. In this section, for convenience, instead of giving 2^n as an explicit input, we will simply give n. Relevant complexity measures (e.g., running time, query and communication complexity) will similarly be functions of n.

Theorem 4.2 (EPZKPP for Permutation)**.** $\mathsf{PERMUTATION} \in \mathsf{EPZKPP}\left[\mathsf{poly}(\log(N, k, 1/\varepsilon))\right]$. *Specifically,* PERMUTATION *has a cheating-verifier perfect zero-knowledge proof of proximity* $\left(\mathsf{P}_{\mathrm{perm}}, \mathsf{V}_{\mathrm{perm}}\right)$ *with expected simulator* $\mathsf{S}_{\mathrm{perm}}$ *with respect to proximity parameter $\varepsilon > 0$ such that the following properties hold for every $n \in \mathbb{N}$, every input $f \colon \{0,1\}^n \to \{0,1\}^n$ and security parameter $k \in \mathbb{N}$:*

1. *The interaction consists of four messages and the total communication is $O\left(n^2 k/\varepsilon^2\right)$ bits.*
2. $\mathsf{V}_{\mathrm{perm}}$*'s running time is* $\mathsf{poly}(n, k, 1/\varepsilon)$ *and* $\mathsf{V}_{\mathrm{perm}}$*'s query complexity is $O(nk/\varepsilon^2)$.*
3. *If $f \in \mathsf{PERMUTATION}_n$, then for every auxiliary input z, $\mathsf{S}_{\mathrm{perm}_{[z]}}$'s expected running time and query complexity, given access to a (possibly cheating) verifier $\widehat{\mathsf{V}}$, are $O\left(t_{\widehat{\mathsf{V}}}(\varepsilon, n, k, z)\right) + \mathsf{poly}(n, k, 1/\varepsilon)$ and $O\left(q_{\widehat{\mathsf{V}}}(\varepsilon, n, k, z) + nk/\varepsilon^2\right)$ respectively, where $t_{\widehat{\mathsf{V}}}(\varepsilon, n, k, z)$ and $q_{\widehat{\mathsf{V}}}(\varepsilon, n, k, z)$ are the running time and query complexity of $\widehat{\mathsf{V}}^f_{[z]}(\varepsilon, n, k)$.*

(Note that the input of $\mathsf{PERMUTATION}_n$ has size $n \cdot 2^n$, so a polynomial dependence on n translates into a poly-logarithmic dependence on the input-size.)

Combined with the aforementioned MAP lower bound for PERMUTATION, we obtain that the complexity of ZKPP (with expected simulation bounds) can be sub-exponentially smaller than those of MAPs (and therefore also of property testers).

Remark 4.3. *We mention that in Item 3 in the theorem statement, when the simulator simulates the view of an interaction with the* honest *verifier* V_{perm}*, its* strict *(rather than expected) query complexity is exactly equal to the query complexity of the verifier.*

The rest of this section is dedicated to proving Theorem 4.2.

4.1.1 Proof of Theorem 4.2

Consider the following simple IPP for PERMUTATION (based on the [BY96] protocol). Given oracle access to a function $f : \{0,1\}^n \to \{0,1\}^n$, the verifier chooses a random $r \in \{0,1\}^n$ and sends r to the prover. The prover computes $z = f^{-1}(r)$ and sends it to the verifier. The verifier checks that indeed $f(z) = r$ and if so accepts.

Clearly if f is a permutation then the verifier in this protocol accepts with probability 1, whereas if f is *far* from a permutation, then with some non-negligible probability the verifier chooses r which does not have a pre-image under f. In such a case the prover cannot make the verifier accept and so the protocol is sound.

It is also not hard to see that this protocol is *honest-verifier* zero-knowledge.[12] However, it is not *cheating-verifier* zero-knowledge: a cheating verifier could learn the inverse of some arbitrary r of its choice.

In order to make the protocol zero-knowledge, intuitively, we would like to have a way for the prover and verifier to jointly sample the element r such that both are assured that it is uniform. For simplicity let us focus on the task of just sampling a single bit σ. The specific properties that we need are

1. If f is a permutation then the prover is assured that the σ is random.
2. If f is far from being a permutation then the verifier is assured that σ is random.

In fact, the transformation of general honest-verifier statistical zero-knowledge proofs to cheating-verifier ones (see [Vad99, Chapter 6]) implements a sub-routine achieving a generalization of the above task, assuming *full* access to the input. We give a simple solution for our specific case. That is, using only oracle access to a function that is either a permutation or far from any permutation.

We proceed to describe a simple procedure for sampling such a random bit σ. First, the verifier chooses at random $x \in \{0,1\}^n$ and a pairwise independent hash function $h : \{0,1\}^n \to \{0,1\}$ and sends $y = f(x)$ and h to the prover. The prover now chooses a random bit $r \in \{0,1\}$ and sends r to the verifier. The verifier now sends x to the prover who checks that indeed $f(x) = y$. The random bit that they agree on is $\sigma = r \oplus h(x)$.

From the prover's perspective, if f is a permutation then y fully determines x and so r (which is chosen uniformly at random after y is specified) is independent of $h(x)$. Hence, $\sigma = r \oplus h(x)$ is uniformly random bit. On the other hand, from the verifier's perspective, if f is far from being a permutation, then, intuitively, even conditioned on the value y there still remains some entropy in x (indeed, x is essentially uniform among all the pre-images of y).[13] Now, using a variant of the leftover hash lemma, we can argue that $h(x)$ is close to random. Actually, since the leftover

[12] As a matter of fact, this protocol can be viewed as a non-interactive statistical zero-knowledge protocol for PERMUTATION (and is used as such in [BY96]).

[13] Actually, the amount of entropy is fairly small (and depends on how far f is from being a permutation). To obtain a sufficient amount of entropy, in our actual protocol we generate many such y's.

The Permutation Protocol $(\mathsf{P}_{\text{perm}}, \mathsf{V}_{\text{perm}})$

P_{perm}'s Input: A function $f\colon \{0,1\}^n \to \{0,1\}^n$, proximity parameter $\varepsilon > 0$ and security parameter k.
V_{perm}'s Input: ε, n, k and oracle access to f.

1. Both parties set $t = \lceil (n+1)/\varepsilon \rceil$ and $s = \lceil k/\varepsilon \rceil$.
2. V_{perm} samples $\overline{\mathbf{x}} = (x_1, x_2, \ldots, x_{t\cdot s}) \leftarrow \left(\{0,1\}^n\right)^{t\cdot s}$ and $h \leftarrow \mathcal{H}_{n\cdot t\cdot s, n\cdot s}$.[a]
   V_{perm} computes $y_i = f(x_i)$ for every $i \in [t\cdot s]$ (by querying f), and sends $\overline{\mathbf{y}} = (y_1, y_2, \ldots, y_{t\cdot s})$ and h to P_{perm}.
3. P_{perm} samples $\overline{\mathbf{r}} = (r_1, r_2, \ldots, r_s) \leftarrow \left(\{0,1\}^n\right)^s$ and send them to V_{perm}.
4. V_{perm} sends $\overline{\mathbf{x}}$ to P_{perm}.
5. P_{perm} checks that $f(x_i) = y_i$, for every $i \in [t\cdot s]$. If any check fails then P_{perm} sends $\perp$ and aborts.
6. P_{perm} sends $\overline{\mathbf{z}} = (z_1, z_2, \ldots, z_s)$ to V_{perm}, where $z_i = f^{-1}\left(r_i \oplus h(\overline{\mathbf{x}})_i\right)$, for every $i \in [s]$.[b]
7. V_{perm} accepts if $f(z_i) = r_i \oplus h(\overline{\mathbf{x}})_i$ for every $i \in [s]$, and otherwise it rejects.

[a] Recall that $\mathcal{H}_{n,m} = \{h\colon \{0,1\}^n \to \{0,1\}^m\}$ is a family of pairwise independent hash functions. See Fact 2.5.
[b] Recall that $\oplus$ stands for the bitwise exclusive-or. Also, we view $h(\overline{\mathbf{x}}) \in \{0,1\}^{n\cdot s}$ as $h(\overline{\mathbf{x}}) = (h(\overline{\mathbf{x}})_1, h(\overline{\mathbf{x}})_2, \ldots, h(\overline{\mathbf{x}})_s)$ such that $h(\overline{\mathbf{x}})_i \in \{0,1\}^n$.

Figure 1: The Permutation Protocol

hash lemma implies that pairwise independent hash functions are *strong* extractors, we have that $h(x)$ is close to random even conditioned on h and therefore also conditioned on r (which is a randomized function of h). Thus, we obtain that $\sigma = r \oplus h(x)$ is close to uniformly random bit and so our procedure satisfies the desired properties.

We proceed to the description of our actual protocol, which is based on the foregoing ideas. The protocol $\left(\mathsf{P}_{\text{perm}}, \mathsf{V}_{\text{perm}}\right)$ is given in Fig. 1.

It is easy to verify that $\left(\mathsf{P}_{\text{perm}}, \mathsf{V}_{\text{perm}}\right)$ has the desired round complexity, query complexity and verifier's running time, where we use the fact that $O(n^2 k/\varepsilon)$ bits suffice for describing the pairwise independent hash function in the protocol (see Fact 2.5).

To see that completeness holds observe that if $f : \{0,1\}^n \to \{0,1\}^n$ is a permutation, and the two parties follow the protocol, then indeed $f(x_i) = y_i$ and $f(z_i) = f(f^{-1}(r_i \oplus h(\overline{\mathbf{x}})_i)) = r_i \oplus h(\overline{\mathbf{x}})_i$ for every $i \in [t\cdot s]$, and therefore the parties complete the interaction and the verifier accepts.

It remains to show that the soundness and zero-knowledge conditions hold. Soundness follows from the following lemma, which is proved in Section 4.1.2.

Lemma 4.4. *Let $n, k \in \mathbb{N}$, let $\varepsilon > 0$ and suppose that $f\colon \{0,1\}^n \to \{0,1\}^n$ is ε-far from* $\mathsf{PERMUTATION}_n$. *Then, for every prover strategy $\widehat{\mathsf{P}}$, when ${\mathsf{V}_{\text{perm}}}^f(\varepsilon, n, k)$ interacts with $\widehat{\mathsf{P}}$ it rejects with probability $1 - \mathsf{negl}(k)$.*

Finally, to show that this protocol is perfect zero-knowledge, consider the simulator S_{perm} given in Fig. 2. The following lemma, which we prove in Section 4.1.3, shows that this simulator perfectly samples from the view of any (possible cheating) verifier.

The Simulator S_{perm} for The Permutation Protocol $(\mathsf{P}_{\text{perm}}, \mathsf{V}_{\text{perm}})$

Simulator's Input: ε, n, k, auxiliary input z, oracle access to $f\colon \{0,1\}^n \to \{0,1\}^n$ and access to (possibly cheating) verifier $\widehat{\mathsf{V}}$.

1. Run $\widehat{\mathsf{V}}^f_{[z]}(\varepsilon, n, k)$ using random coins ρ to get $\overline{\mathbf{y}} = (y_1, y_2, \ldots, y_{ts})$ and h.
2. Sample $\overline{\mathbf{r}} = (r_1, r_2, \ldots, r_s) \leftarrow (\{0,1\}^n)^s$ and give them to $\widehat{\mathsf{V}}_{[z]}$ as the answers from P_{perm}.
3. Continue to run $\widehat{\mathsf{V}}^f_{[z]}(\varepsilon, n, k)$ to get $\overline{\mathbf{x}} = (x_1, x_2, \ldots, x_{ts})$, the values $\widehat{\mathsf{V}}^f_{[z]}(\varepsilon, n, k)$ sends to P_{perm} in the third message of the protocol.
4. If there exists $i \in [t \cdot s]$ such that $f(x_i) \neq y_i$, output $(\overline{\mathbf{y}}, h, \overline{\mathbf{r}}, \overline{\mathbf{x}}, \perp, \rho)$.
5. Otherwise, repeat the following:
   (a) Sample $\overline{\mathbf{r'}} = (r'_1, r'_2, \ldots, r'_s) \leftarrow (\{0,1\}^n)^s$ and for every $i \in [s]$, set $r''_i = f(r'_i) \oplus h(\overline{\mathbf{x}})_i$.
   (b) Rewind $\widehat{\mathsf{V}}^f_{[z]}(\varepsilon, n, k)$ to the point it is waiting for the second message of the protocol using ρ again as the random coins (i.e., step 3 of the protocol).
   Give $\overline{\mathbf{r''}} = (r''_1, r''_2, \ldots, r''_{ts})$ as the answers from P_{perm}.
   (c) Continue to run $\widehat{\mathsf{V}}^f_{[z]}(\varepsilon, n, k)$ to get $\overline{\mathbf{x'}} = (x'_1, x'_2, \ldots, x'_{ts})$, the values $\widehat{\mathsf{V}}^f_{[z]}(\varepsilon, n, k)$ sends to P_{perm} in the third message of the protocol.
   (d) If $f(x_i) = y_i$ for every $i \in [ts]$, output $(\overline{\mathbf{y}}, h, \overline{\mathbf{r''}}, \overline{\mathbf{x'}}, \overline{\mathbf{r'}}, \rho)$ and halt.
   Otherwise, go back to 5a.

Figure 2: The Simulator for The Permutation Protocol

Lemma 4.5. *Let $n, k \in \mathbb{N}$, let $z \in \{0,1\}^*$, let $f \in \mathsf{PERMUTATION}_n$, let $\widehat{\mathsf{V}}$ be some verifier strategy and let $\mathsf{S}^f_{[z]}(\varepsilon, n, k, \widehat{\mathsf{V}})$ be the output of S_{perm} when running on input ε, n, k, auxiliary input z, with oracle access to f and access to $\widehat{\mathsf{V}}$. Then:*

$$\mathrm{SD}\left(\mathsf{S}^f_{[z]}(\varepsilon, n, k, \widehat{\mathsf{V}}), \mathrm{view}_{\mathsf{P}_{\text{perm}}, \widehat{\mathsf{V}}_{[z]}}(\varepsilon, n, k, f)\right) = 0.$$

Moreover, the expected running time and query complexity of $\mathsf{S}^f_{[z]}(\varepsilon, n, k, \widehat{\mathsf{V}})$ are as in Item 3 of the theorem statement.

This concludes the proof of Theorem 4.2 (modulo the proofs of Lemma 4.4 and Lemma 4.5 which are proved next).

4.1.2 Analyzing Soundness — Proof of Lemma 4.4

Before proving Lemma 4.4 we show two basic, but useful, properties of functions that are ε-far from permutations. The first property is that the image size of such functions cannot be too large.

Fact 4.6. *If f is ε-far from $\mathsf{PERMUTATION}_n$, then $|\mathrm{Im}(f)| \leq (1-\varepsilon) \cdot 2^n$.*

Proof. We prove the contrapositive. Let $f : \{0,1\}^n \to \{0,1\}^n$ and suppose that $|\mathrm{Im}(f)| > (1-\varepsilon) \cdot 2^n$. We show that f is ε-close to a permutation $f' : \{0,1\}^n \to \{0,1\}^n$.

Start by setting $f'(x) = f(x)$ for every x. Repeat the following process until $\mathrm{Im}(f') = \{0,1\}^n$: take $y \notin \mathrm{Im}(f)$; find $x \neq x'$ such that $f'(x) = f'(x')$ (such x, x' must exist since at this point f' is not a permutation); set $f'(x) = y$.

In every iteration $|\mathrm{Im}(f)|$ increases by one. The above process started with $|\mathrm{Im}(f')| > (1-\varepsilon)\cdot 2^n$, and thus takes less than $\varepsilon \cdot 2^n$ iterations. It follows that f' and f disagree on at most $\varepsilon \cdot 2^n$ inputs, or in other words f' is ε-close to f. Moreover, f' is a permutation, since $\mathrm{Im}(f') = \{0,1\}^n$. □

Next we show that even after seeing a random element in the image of a function that is ε-far from permutation, its preimage still has some entropy. The specific notion of entropy we use is average min-entropy.[14]

Claim 4.7. *Suppose that $f\colon \{0,1\}^n \to \{0,1\}^n$ is ε-far from $\mathsf{PERMUTATION}_n$. Let X be a random variable uniformly distributed over $\{0,1\}^n$, and let $Y = f(X)$. Then, $\tilde{\mathrm{H}}_\infty(X|Y) \geq \log(1/(1-\varepsilon))$.*

Proof. For $y \in \{0,1\}^n$, let $f^{-1}(y) = \{x \in \{0,1\}^n\colon f(x) = y\}$. Fix $y \in \mathrm{Im}(f)$. For $x \in f^{-1}(y)$, it holds that $\Pr[X = x|Y = y] = 1/\left|f^{-1}(y)\right|$, while for $x \notin f^{-1}(y)$, it holds that $\Pr[X = x|Y = y] = 0$. Thus, $\max_x \Pr[X = x|Y = y] = 1/\left|f^{-1}(y)\right|$. Moreover, it holds that $\Pr[Y = y] = \left|f^{-1}(y)\right|/2^n$. Finally, for every $y \notin \mathrm{Im}(f)$, it holds that $\Pr[Y = y] = 0$. Hence,

$$\begin{aligned}
\tilde{\mathrm{H}}_\infty(X|Y) &= -\log\Big(\mathrm{E}_{y\leftarrow Y}\Big[\max_x \Pr[X = x|Y = y]\Big]\Big) \\
&= -\log\left(\sum_{y\in\mathrm{Im}(f)} \frac{\left|f^{-1}(y)\right|}{2^n} \cdot \frac{1}{\left|f^{-1}(y)\right|}\right) \\
&= \log\left(\frac{2^n}{|\mathrm{Im}(f)|}\right) \\
&\geq \log\left(\frac{1}{1-\varepsilon}\right),
\end{aligned}$$

where the inequality follows from Fact 4.6. □

We are now ready to prove Lemma 4.4.

Proof of Lemma 4.4. Let $\widehat{\mathsf{P}}$ be a (cheating) prover strategy. We assume without loss of generality that $\widehat{\mathsf{P}}$ is deterministic (by fixing the best choice of random coin tosses).

Let $\overline{\mathbf{X}}$, H, $\overline{\mathbf{Y}}$, $\overline{\mathbf{R}}$ and $\overline{\mathbf{Z}}$ be the (jointly distributed) random variables induced by the values of $\overline{\mathbf{x}}$, h, $\overline{\mathbf{y}}$, $\overline{\mathbf{r}}$ and $\overline{\mathbf{z}}$ respectively, in a random execution of $(\widehat{\mathsf{P}}, \mathsf{V}_{\mathrm{perm}})$ and let $\mathrm{out}(\widehat{\mathsf{P}}, \mathsf{V}_{\mathrm{perm}})$ be the random variable induced by $\mathsf{V}_{\mathrm{perm}}$'s output in the same random execution (i.e., $\mathrm{out}(\widehat{\mathsf{P}}, \mathsf{V}_{\mathrm{perm}}) \in \{accept, reject\}$).

By the definition of the permutation protocol, it holds that

$$\begin{aligned}
\Pr\Big[\mathrm{out}(\widehat{\mathsf{P}}, \mathsf{V}_{\mathrm{perm}}) = accept\Big] &\leq \Pr\big[\forall i \in [s]\colon f(X'_i) = R_i \oplus H(\overline{\mathbf{X}})_i\big] \qquad (1)\\
&\leq \Pr\big[\forall i \in [s]\colon R_i \oplus H(\overline{\mathbf{X}})_i \in \mathrm{Im}(f)\big].
\end{aligned}$$

[14] Recall that average min-entropy of X given Y is defined as $\tilde{\mathrm{H}}_\infty(X|Y) := -\log(\mathrm{E}_{y\leftarrow Y}[\max_x \Pr[X = x \mid Y = y]])$ (see Definition 2.2).

Note that $\overline{\mathbf{R}} = (R_1, \ldots, R_s)$ is a function of $\overline{\mathbf{Y}}$ and H, determined by $\widehat{\mathsf{P}}$. It follows that

$$\begin{aligned}\Pr\left[\forall i \in [s]\colon R_i(\overline{\mathbf{Y}}, H) \oplus H(\overline{\mathbf{X}})_i \in \mathrm{Im}(f)\right] \leq \Pr\left[\forall i \in [s]\colon R_i(\overline{\mathbf{Y}}, H) \oplus U_i \in \mathrm{Im}(f)\right] \\ + \mathrm{SD}\big((H(\overline{\mathbf{X}}), H, \overline{\mathbf{Y}}), (\overline{\mathbf{U}}, H, \overline{\mathbf{Y}})\big),\end{aligned} \tag{2}$$

where $\overline{\mathbf{U}} = (U_1, U_2, \ldots, U_s)$ is uniform over $(\{0,1\}^n)^s$. We bound both terms in the right-hand side of Equation (2). For the first term, note that U_i is independent of U_j for $i \neq j$, and thus

$$\begin{aligned}\Pr\left[\forall i \in [s]\colon R_i(\overline{\mathbf{Y}}, H) \oplus U_i \in \mathrm{Im}(f)\right] &= \prod_{i=1}^{s} \Pr\left[R_i(\overline{\mathbf{Y}}, H) \oplus U_i \in \mathrm{Im}(f)\right] \\ &= \prod_{i=1}^{s} \Pr[U_i \in \mathrm{Im}(f)] \\ &\leq (1-\varepsilon)^s,\end{aligned} \tag{3}$$

where the second equality follows from the fact that for every $r \in \{0,1\}^n$, $r \oplus U_i$ is uniform over $\{0,1\}^n$ and the last inequality follows from Fact 4.6.

As for the second term of Equation (2), by Fact 2.3 it holds that

$$\tilde{\mathrm{H}}_\infty(\overline{\mathbf{X}}|\overline{\mathbf{Y}}) = t \cdot s \cdot \tilde{\mathrm{H}}_\infty(X_1|Y_1) \geq t \cdot s \cdot \log(1/(1-\varepsilon)), \tag{4}$$

where the inequality follows from Claim 4.7. Applying the generalized leftover hash lemma (Lemma 2.6) we now obtain that:

$$\mathrm{SD}\big((H(\overline{\mathbf{X}}), H, \overline{\mathbf{Y}}), (U, H, \overline{\mathbf{Y}})\big) \leq \frac{1}{2} \cdot \sqrt{2^{ts\log(1-\varepsilon)} \cdot 2^{ns}} = \frac{1}{2} \cdot \left(2^n \cdot (1-\varepsilon)^t\right)^{s/2}. \tag{5}$$

Plugging Equations (2), (3) and (5) into Equation (1), we have

$$\begin{aligned}\Pr\left[\mathrm{out}(\widehat{\mathsf{P}}, \mathsf{V}_{\mathrm{perm}}) = \mathit{accept}\right] &\leq (1-\varepsilon)^s + \frac{1}{2} \cdot \left(2^n \cdot (1-\varepsilon)^t\right)^{s/2} \\ &\leq (1-\varepsilon)^{k/\varepsilon} + \frac{1}{2} \cdot \left(2^n \cdot (1-\varepsilon)^{(n+1)/\varepsilon}\right)^{k/(2\varepsilon)} \\ &\leq 2^{-k} + \frac{1}{2} \cdot \left(2^n \cdot 2^{-(n+1)}\right)^{k/(2\varepsilon)} \\ &= 2^{-k} + 2^{-k/(2\varepsilon)-1},\end{aligned} \tag{6}$$

where the second inequality follows from our setting of $t = \lceil (n+1)/\varepsilon \rceil$ and $s = \lceil k/\varepsilon \rceil$ and the third inequality follows from the fact that $1 - x \leq 2^{-x}$ for any $x \geq 0$. Thus, the verifier accepts with probability that is exponentially vanishing in k, and in particular negligible.

□

4.1.3 Analyzing Zero-Knowledge — Proof of Lemma 4.5

Let $\widehat{\mathsf{V}}$ be a cheating verifier strategy and fix an input $f \in$ PERMUTATION. For simplicity, and without loss of generality, we assume that $\widehat{\mathsf{V}}$ is deterministic.[15]

[15] Recall that if the cheating verifier is randomized, we can fix its random coins as part of the auxiliary input to both parties.

Throughout this proof we fix an auxiliary input z to S_{perm} and remove it from the notation of both the simulator and the (possibly cheating) verifier (since all S_{perm} does with its auxiliary input is to provide it to $\widehat{\mathsf{V}}$, both algorithms get the same auxiliary inputs).

Recall that we let $\mathsf{S}^f(\varepsilon, n, k, \widehat{\mathsf{V}})$ denote the algorithm defined by running S_{perm} on input ε, n, k, with oracle access to f and access to $\widehat{\mathsf{V}}$. Note that $\mathsf{S}^f(\varepsilon, n, k, \widehat{\mathsf{V}})$ halts *almost surely*, namely the probability that it never halts is zero.

The following claim shows show that the output distribution of S_{perm} is identical to the view of $\widehat{\mathsf{V}}$. Later, in Claim 4.9, we will bound the (expected) running time and query complexity of S_{perm}.

Claim 4.8. *The output distribution of* S_{perm} *is identical to the view of* $\widehat{\mathsf{V}}$.

Proof. Let $\overline{\mathbf{X}}$, H, $\overline{\mathbf{Y}}$, $\overline{\mathbf{R}}$ and $\overline{\mathbf{Z}}$ be the (jointly distributed) random variables induced by the values of $\overline{\mathbf{x}}$, h, $\overline{\mathbf{y}}$, $\overline{\mathbf{r}}$ and $\overline{\mathbf{z}}$ respectively, in a random execution of $(\mathsf{P}_{\text{perm}}, \widehat{\mathsf{V}})$.

First observe that since $\widehat{\mathsf{V}}$ is deterministic, its first message $(\overline{\mathbf{y}}, h)$ is fixed and so $\overline{\mathbf{Y}} = \overline{\mathbf{y}}$ and $H = h$. Also, since the verifier is deterministic, there exists a function v such that $\overline{\mathbf{X}} = v(\overline{\mathbf{R}})$. Lastly, observe that there also exists a function u such that $\overline{\mathbf{Z}} = u(\overline{\mathbf{R}})$.

The view of the verifier is therefore:

$$\text{view}_{\mathsf{P},\widehat{\mathsf{V}}}(\varepsilon, n, k, f) = \left(\overline{\mathbf{Y}}, H, \overline{\mathbf{R}}, \overline{\mathbf{X}}, \overline{\mathbf{Z}}\right) = \left(\overline{\mathbf{y}}, h, \overline{\mathbf{R}}, v(\overline{\mathbf{R}}), u(\overline{\mathbf{R}})\right) \tag{7}$$

Similarly, let $\overline{\mathbf{X}_{\mathsf{S}}}$, H_{S}, $\overline{\mathbf{Y}_{\mathsf{S}}}$, $\overline{\mathbf{R}_{\mathsf{S}}}$, $\overline{\mathbf{Z}_{\mathsf{S}}}$ be the (jointly distributed) random variables induced by the output of a random execution of $\mathsf{S}^f(\varepsilon, n, k, \widehat{\mathsf{V}})$. We need to show that:

$$(\overline{\mathbf{Y}}, H, \overline{\mathbf{R}}, \overline{\mathbf{X}}, \overline{\mathbf{Z}}) \equiv (\overline{\mathbf{Y}_{\mathsf{S}}}, H_{\mathsf{S}}, \overline{\mathbf{R}_{\mathsf{S}}}, \overline{\mathbf{X}_{\mathsf{S}}}, \overline{\mathbf{Z}_{\mathsf{S}}}). \tag{8}$$

First observe that by construction $\overline{\mathbf{Y}_{\mathsf{S}}} = \overline{\mathbf{y}}$ and $H_{\mathsf{S}} = h$. Also observe that no matter what value $\overline{\mathbf{R}_{\mathsf{S}}}$ obtains, in all steps in which the simulator might generate an output, it holds that $\overline{\mathbf{X}_{\mathsf{S}}} = v(R_{\mathsf{S}})$, where the function v is as defined above. Similarly it holds that $\overline{\mathbf{Z}_{\mathsf{S}}} = u(R_{\mathsf{S}})$, where u was defined above.

Thus, Equation (8) reduces to showing that $\overline{\mathbf{R}}$ and $\overline{\mathbf{R}_{\mathsf{S}}}$ are identically distributed. Since $\overline{\mathbf{R}}$ is uniform all we need to show is that $\overline{\mathbf{R}_{\mathsf{S}}}$ is also uniformly distributed.

Let

$$\mathcal{A} = \left\{\overline{\mathbf{r}} \in (\{0,1\}^n)^s \;:\; \exists i \in [t \cdot s] \text{ s.t. } y_i \neq f(x_i) \text{ where } \overline{\mathbf{x}} = v(\overline{\mathbf{r}})\right\}, \tag{9}$$

where $\overline{\mathbf{y}} = (y_1, \ldots, y_{ts})$ (and recall that $\overline{\mathbf{y}}$ was fixed). Namely, $\mathcal{A}$ contains those elements in $(\{0,1\}^n)^s$, that had they been sent by P_{perm} as its second message, the verifier $\widehat{\mathsf{V}}$ would have sent $\overline{\mathbf{x}}$ that are not the preimages of $\overline{\mathbf{y}}$. Finally, let $p = \Pr_{r \leftarrow (\{0,1\}^n)^s}[r \notin \mathcal{A}]$ and fix $r^* \in (\{0,1\}^n)^s$. We show that $\Pr[R_{\mathsf{S}} = r^*] = 1/(2^n)^s$. The proof now splits according to r^*.

$r^* \in \mathcal{A}$**:** The only way $\mathsf{S}^f(\varepsilon, n, k, \widehat{\mathsf{V}})$ would output r^* is by choosing it in Step 2. Since $\mathsf{S}^f(\varepsilon, n, k, \widehat{\mathsf{V}})$ chooses the values in this step uniformly at random from $(\{0,1\}^n)^s$, it follows that $\Pr[R_{\mathsf{S}} = r^*] = 1/(2^n)^s$.

$r^* \notin \mathcal{A}$**:** The only way $\mathsf{S}^f(\varepsilon, n, k, \widehat{\mathsf{V}})$ would output r^* is by choosing $r'' = r^*$ in Step 5a. The probability that $\mathsf{S}^f(\varepsilon, n, k, \widehat{\mathsf{V}})$ reaches Step 5 at all is p. Having reached Step 5, and since f is a permutation, every time that $\mathsf{S}^f(\varepsilon, n, k, \widehat{\mathsf{V}})$ runs Step 5a, it samples r'' uniformly at random from $(\{0,1\}^n)^s$, independent of all previous messages it received from $\widehat{\mathsf{V}}$. In Step 5 $\mathsf{S}^f(\varepsilon, n, k, \widehat{\mathsf{V}})$ is performing rejection sampling until it gets $r'' \notin \mathcal{A}$ and then sets $\overline{\mathbf{R}}_{\mathsf{S}} = r''$. All in all, it holds that

$$\begin{aligned}\Pr[\overline{\mathbf{R}}_{\mathsf{S}} = r^*] &= p \cdot \Pr_{r'' \leftarrow (\{0,1\}^n)^s}[r'' = r^* \mid r'' \notin \mathcal{A}] \\ &= \Pr_{r \leftarrow (\{0,1\}^n)^s}[r \notin \mathcal{A}] \cdot \frac{\Pr_{r'' \leftarrow (\{0,1\}^n)^s}[r'' = r^*]}{\Pr_{r'' \leftarrow (\{0,1\}^n)^s}[r'' \notin \mathcal{A}]} \\ &= \Pr_{r \leftarrow (\{0,1\}^n)^s}[r = r^*] = \frac{1}{(2^n)^s}.\end{aligned}$$

(Note that we can condition on the event $r'' \notin \mathcal{A}$ since $r^* \notin \mathcal{A}$ and so $\Pr[r'' \notin \mathcal{A}] > 0$.)

Hence, $\overline{\mathbf{R}}_{\mathsf{S}}$ is uniform over $(\{0,1\}^n)^s$. This completes the proof of Claim 4.8. □

Claim 4.9. *If the cheating verifier $\widehat{\mathsf{V}}$ runs in time $t_{\widehat{\mathsf{V}}}$ and makes $q_{\widehat{\mathsf{V}}}$ queries, then the* expected *running time of the simulator S_{perm} is $O(t_{\widehat{\mathsf{V}}}) + \mathsf{poly}(n, k, 1/\varepsilon)$ and its query complexity is $O(q_{\widehat{\mathsf{V}}} + nk/\varepsilon^2)$.*

Proof. We prove this part by first showing the expected number of calls it makes to $\widehat{\mathsf{V}}$ is constant.

Let T be the number of times $\mathsf{S}^f(\varepsilon, n, k, \widehat{\mathsf{V}})$ executes Steps 3 and 5c when $\widehat{\mathsf{V}}$ uses the coins ρ. Note that T is equal to the number of times the simulator invokes $\widehat{\mathsf{V}}$. Let $\mathcal{A}$ be as defined in Equation (9) (recall that $\mathcal{A}$ was defined as the set of vectors $\overline{\mathbf{r}}$ for which the verifier $\widehat{\mathsf{V}}$ responds with $\overline{\mathbf{x}}$ that do not all correspond to the respective preimages of $\overline{\mathbf{y}}$). Let $p = \Pr_{r \leftarrow (\{0,1\}^n)^s}[r \notin \mathcal{A}]$.

Let E denote the event that $\mathsf{S}^f(\varepsilon, n, k, \widehat{\mathsf{V}})$ reaches Step 5. By construction, $\Pr[T = 1] = \Pr[\neg E] = 1 - p$. Moreover, it holds that the random variable $T|E$ is drawn from a geometric distribution with parameter p. Since the latter has expectation $1/p$ we have that:

$$\begin{aligned}\mathrm{E}[T] &= \Pr[\neg E] \cdot \mathrm{E}[T \mid \neg E] + \Pr[E] \cdot \mathrm{E}[T \mid E] \\ &= (1-p) \cdot 1 + p \cdot \frac{1}{p} \\ &= 2 - p,\end{aligned} \tag{10}$$

Thus, in expectation, the simulator invokes $\widehat{\mathsf{V}}$ at most twice.

Every time $\mathsf{S}^f(\varepsilon, n, k, \widehat{\mathsf{V}})$ calls $\widehat{\mathsf{V}}$, it samples a random string in $\{0,1\}^{n \cdot s}$, evaluates some $h \in \mathcal{H}_{n \cdot t \cdot s, n \cdot s}$, and makes $O(t \cdot s)$ calls to f. Recall that $t_{\widehat{\mathsf{V}}}$ and $q_{\widehat{\mathsf{V}}}$ denote the running time and query complexity of $\widehat{\mathsf{V}}$, respectively. The expected running time of $\mathsf{S}^f(\varepsilon, n, k, \widehat{\mathsf{V}})$ is thus at most $O(t_{\widehat{\mathsf{V}}}) + \mathsf{poly}(t, s, n) = O(t_{\widehat{\mathsf{V}}}) + \mathsf{poly}(n, k, 1/\varepsilon)$ (note that by Fact 2.5, evaluation of h can be done in time $\mathsf{poly}(t, s, n)$). The expected query complexity of $\mathsf{S}^f(\varepsilon, n, k, \widehat{\mathsf{V}})$ is thus at most $O(q_{\widehat{\mathsf{V}}} + t \cdot s) = O(q_{\widehat{\mathsf{V}}} + nk/\varepsilon^2)$. □

4.2 Promise Expansion is in HV-SZKPP

In this section we consider the property of a graph being a "good" expander, in the bounded degree graph model (see [GGR98, GR02]). Recall that in bounded degree graph model, the input graph is represented by an adjacency list and so, using a single query, the verifier can find the i^{th} neighbor of a vertex v.

The property of being a good expander was first considered by Goldreich and Ron [GR11]. They showed that any tester for the (spectral) expansion of a graph on n vertices must make at least $\Omega(\sqrt{n})$ queries. [GR11] also suggested a testing algorithm that matches this bound and conjectured its correctness. Czumaj and Sohler [CS10] focused on vertex expansion and proved that the [GR11] tester accepts graphs that are good expanders and rejects graphs that are far from having even much worst expansion. More specifically, [CS10] showed that the [GR11] tester accepts graphs with (vertex) expansion α and rejects graphs that are *far* from having even (vertex) expansion $\Theta\left(\frac{\alpha^2}{\log(n)}\right)$. Lastly, Nachmias and Shapira [NS10] and Kale and Seshadhri[KS11] improved [CS10]'s result and showed that the tester in fact rejects graphs that are far from having expansion $\Theta(\alpha^2)$. We show how to apply [CS10]'s approach to get an honest-verifier statistical zero-knowledge proof of proximity with only a *poly-logarithmic* dependence on n, as long as we have a similar type of gap between YES and NO instances as in [CS10].

Formally, a vertex expander is defined as follows.

Definition 4.10 (Vertex expander)**.** *Let $\alpha > 0$. A graph $G = (\mathcal{V}, \mathcal{E})$ is an α-*expander *if for every subset $\mathcal{U} \subseteq \mathcal{V}$ of size $|\mathcal{U}| \leq |\mathcal{V}|/2$, it holds that $|N(\mathcal{U})| \geq \alpha \cdot |\mathcal{U}|$, where $N(\mathcal{U}) := \{v \in \mathcal{V} \setminus \mathcal{U} \colon \exists u \in \mathcal{U} \text{ such that } (v,u) \in \mathcal{E}\}$.*

Throughout this section we fix a bound d on the degree of all graphs (which we think of as constant). We identify graphs on n vertices as functions $G \colon [n] \times [d] \to [n] \cup \{\perp\}$ such that $G(u,i) = v$ if v is the i'th neighbor of a vertex u and $G(u,i) = \perp$ if u has less than i neighbors.

Definition 4.11. *Let $d \in \mathbb{N}$. For $n \in \mathbb{N}$ and $\alpha = \alpha(n) > 0$, let*

$$\mathsf{EXPANDER}_n^{d,\alpha} = \left\{ G \colon [n] \times [d] \to [n] \;\middle|\; G \text{ is a } \alpha(n)\text{-expander} \right\}.$$

Let $\beta = \beta(n) \in (0, \alpha(n)]$. We define the expander promise problem (see Remark 3.5) as

$$\mathsf{EXPANDER}^{d,\alpha,\beta} = \left(\mathsf{EXPANDER}^{d,\alpha,\beta}_{\mathsf{YES},n}, \mathsf{EXPANDER}^{d,\alpha,\beta}_{\mathsf{NO},n}, [n] \times [d], [n] \right),$$

where $\mathsf{EXPANDER}^{d,\alpha,\beta}_{\mathsf{YES},n} = \mathsf{EXPANDER}^{d,\alpha}_n$ and $\mathsf{EXPANDER}^{d,\alpha,\beta}_{\mathsf{NO},n} = \mathsf{EXPANDER}^{d,\beta}_n$.

That is, YES instances of the promise problem are graphs that are α-expanders and NO instances are graphs that are far from even being β-expanders, for $\beta \leq \alpha$.

Theorem 4.12 (SZKPP for Expansion)**.** *Let $d \in \mathbb{N}$ and $\alpha \in (0, 1/3]$ be constants. Then, $\mathsf{EXPANDER}^{d,\alpha,\beta} \in \mathsf{HV\text{-}SZKPP}[\mathsf{poly}(\log(n), k, 1/\varepsilon)]$ for $\beta = \Theta\left(\frac{\alpha^2}{d^2 \log(n)}\right)$, where n is the number of vertices in the graph, k is the security parameter and ε is the proximity parameter.*

Proof of Theorem 4.12. We prove Theorem 4.12 by reducing graph expansion to the problem of testing whether two distributions are statistically close. The reduction is such that we can sample from each distribution using few queries to our original input graph. Given this reduction, we

can now use Lemma 2.14 which gives an honest-verifier statistical zero-knowledge proof for verifying whether the distribution induced by two circuits on a random input is statistically close. A crucial observation is that neither the verifier nor the simulator in the latter protocol need to actually look at their input circuits. Rather, they only need to be able to draw relatively few random samples from the distribution induced by the circuit on a random input. Intuitively, by applying our reduction we therefore obtain an honest-verifier statistical zero-knowledge proof for $\mathsf{EXPANDER}^{d,\alpha,\beta}$.

We proceed to give an overview of our reduction from $\mathsf{EXPANDER}^{d,\alpha,\beta}$ to statistical closeness. The reduction, which is randomized, chooses uniformly at random a vertex u and considers two distributions: the first, denoted by P_u^ℓ, outputs the last vertex in a random walk of length $\ell = \Theta\left(\frac{d^2\log(n)}{\alpha^2}\right)$ starting at u; the second, denoted by $U_{[n]}$, is uniform over all vertices. Observe that both distributions can be sampled using relatively few queries to the input graph.

We observe that if the graph is an α-expander (i.e., a YES instance) then for *any* choice of u, it holds that $\mathrm{SD}\left(P_u^\ell, U_{[n]}\right) \approx 0$ (and so the two distributions are close). On the other hand, [CS10] showed that if the graph is far from being a $\Theta\left(\frac{\alpha^2}{d^2\log(n)}\right)$-expander (i.e., a NO instance), then there exists a set of vertices $\mathcal{U}$ with $|\mathcal{U}| = \Omega(n)$ such that for every $u \in \mathcal{U}$, it holds that $\mathrm{SD}\left(P_u^\ell, U_{[n]}\right) \gg 0$. Thus, in the NO case, with constant probability (over the choice of u), our reduction generates distributions that are statistically far. We proceed to the actual proof.

Our protocol uses the following lazy random walk on the graph G.

Definition 4.13 (Random walk)**.** *Let $G = (\mathcal{V}, \mathcal{E})$ be a (simple) bounded degree d graph. For $u \in \mathcal{V}$, define $p_u(v) = 1/2d$ if $(u,v) \in \mathcal{E}$ (i.e., (u,v) is an edge), and $p_u(u) = 1 - \deg(u)/2d$. For $\ell \in \mathbb{N}$, a* random walk of length ℓ starting at u *is a random process that chooses ℓ vertices $u_1, \ldots, u_\ell$ such that $u_1 := u$ and every vertex u_{i+1} is sampled from the distribution p_{u_i}. The distribution $P_u^\ell(v)$ is the probability that $u_\ell = v$.*

Assume without loss of generality that $\varepsilon \leq 0.1$, where ε is the proximity parameter.[16] Let $h : [0,1] \to [0,1]$ be the binary entropy function (recall that $h(p) := -p\log(p) - (1-p)\log(1-p)$). By a routine calculation, it holds that

$$h\left(\frac{1}{2}\cdot(1+0.2)\right) \leq h(0.6) < 0.98 < 1 - 0.01. \tag{11}$$

Thus, we can apply Lemma 2.14, with respect to the constants $\alpha = 0.2$ and $\beta = 0.01$ to obtain an honest verifier statistical zero knowledge protocol $\left(\mathsf{P}^{(\cdot,\cdot)}(k), \mathsf{V}^{(\cdot,\cdot)}(k)\right)$ for $\overline{\mathsf{SD}^{0.2,0.01}}$. Thus, $\left(\mathsf{P}^{(\cdot,\cdot)}(k), \mathsf{V}^{(\cdot,\cdot)}(k)\right)$ is a statistical zero-knowledge protocol for distinguishing between YES instances, which are pairs of circuits that have statistical distance at most 0.01, from NO instances, which are pairs of circuits whose statistical distance is at least 0.2.[17] Using the latter, we construct a protocol $(\mathsf{P}_{\mathrm{expan}}, \mathsf{V}_{\mathrm{expan}})$ for verifying expansion, which is given in Fig. 3.

The next two lemmas show the completeness and soundness of the expander protocol.

Lemma 4.14. *Let $n, k \in \mathbb{N}$, let $\varepsilon > 0$ and let $G\colon [n] \times [d] \to [n]$. Assume that G is in $\mathsf{EXPANDER}_n^{d,\alpha}$ (i.e., G is an α-expander), then $\mathsf{V}_{\mathrm{expan}}{}_{\alpha,d}^{G}(\varepsilon, n, k)$, when interacting with $\mathsf{P}_{\mathrm{expan}}{}_{\alpha,d}(\varepsilon, G, k)$ according to the expander protocol (Fig. 3), accepts with probability $1 - \mathsf{negl}(k)$.*

[16] Otherwise, we can just "reset" ε to 0.1.

[17] The constant 0.2 that we use here stems from the analysis of [CS10]. On the other hand, the constant 0.01 is arbitrary (but was chosen so that Equation (11) holds).

The Expander Protocol $(\mathsf{P}_{\text{expan}}, \mathsf{V}_{\text{expan}})$

Prover's Input: A graph $G\colon [n] \times [d] \to [n]$, expansion parameter $\alpha > 0$, proximity parameter $\varepsilon > 0$ and security parameter k.
Verifier's Input: α, ε, n, d, k and oracle access to G.

1. $\mathsf{V}_{\text{expan}}$ samples $u \leftarrow [n]$ and sends u to $\mathsf{P}_{\text{expan}}$.
2. The parties construct two oracle circuits: one encodes the distribution P_u^ℓ for $\ell = \left\lceil \frac{8d^2}{\alpha^2} \ln(\sqrt{n}/0.01) \right\rceil$; the other encodes $U_{[n]}$, the uniform distribution on the graph's vertices.
3. The parties run the protocol $\left(\mathsf{P}^{P_u^\ell, U_{[n]}}(k), \mathsf{V}^{P_u^\ell, U_{[n]}}(k)\right)$, where $\left(\mathsf{P}^{(\cdot,\cdot)}, \mathsf{V}^{(\cdot,\cdot)}\right)$ is the protocol for $\overline{\mathsf{SD}^{0.2,0.01}}$ from Lemma 2.14.

Figure 3: The Expander Protocol

Lemma 4.14 is proven in Section 4.2.1 via a standard analysis of random walks on expanders.

Lemma 4.15. *Let $n, k \in \mathbb{N}$, let $0 < \varepsilon \leq 0.1$ and let $G\colon [n] \times [d] \to [n]$. Assume that G is ε-far from $\mathsf{EXPANDER}_n^{d,\beta}$ for $\beta = \Theta\left(\frac{\alpha^2}{d^2 \log(n)}\right)$, then for every prover strategy $\widehat{\mathsf{P}}$, when $\mathsf{V}_{\text{expan}}{}_{\alpha,d}^{G}(\varepsilon, n, k)$ interacts with $\widehat{\mathsf{P}}$ according to the expander protocol (Fig. 3), with probability at least $\varepsilon/24 - \mathsf{negl}(k)$ it rejects.*

Lemma 4.15 is proven in Section 4.2.2 via a combinatorial property of graphs that are ε-far from β-expanders, shown by [CS10].

As for honest-verifier zero-knowledge, let $\mathsf{S}^{(\cdot,\cdot)}$ denote the simulator of the protocol for $\overline{\mathsf{SD}^{0.2,0.01}}$ from Lemma 2.14. Note that if G is an α-expander, the same mixing argument used to establish completeness implies that $\mathrm{SD}\left(P_u^\ell, U_{[n]}\right) \leq 0.01$, for every vertex u (see Section 4.2.1). Hence, for every vertex u it holds that $\mathsf{S}^{P_u^\ell, U_{[n]}}(k)$ simulates $\left(\mathsf{P}^{P_u^\ell, U_{[n]}}(k), \mathsf{V}^{P_u^\ell, U_{[n]}}(k)\right)$ with simulation deviation at most $\mu(k)$, for some negligible function μ. Our simulator for the expansion protocol, denoted by $\mathsf{S}_{\text{expan}}$ will choose a vertex u uniformly at random, and output $\left(u, \mathsf{S}^{P_u^\ell, U_{[n]}}(k)\right)$. Now observe that $\mathsf{S}_{\text{expan}}$'s deviation from $\mathsf{V}_{\text{expan}}$'s view in $(\mathsf{P}_{\text{expan}}, \mathsf{V}_{\text{expan}})$ is precisely equal to the expected deviation, over the choice of a random vertex u, of $\mathsf{S}^{P_u^\ell, U_{[n]}}(k)$ from the view of $\mathsf{V}^{P_u^\ell, U_{[n]}}$ in the protocol $\left(\mathsf{P}^{P_u^\ell, U_{[n]}}(k), \mathsf{V}^{P_u^\ell, U_{[n]}}(k)\right)$. Since the latter is bounded by μ for *every* choice of u, the expected deviation is bounded by $\mu(k)$ as well.

So far we have shown that the expander protocol (Fig. 3) has $\mathsf{negl}(k)$ completeness error, $1 - (\varepsilon/24 - \mathsf{negl}(k))$ soundness error and is honest-verifier statistical zero-knowledge. To reduce the soundness error the parties will repeat the above protocol in parallel for $\mathsf{poly}(k)/\varepsilon$ times. Since *honest-verifier* statistical zero-knowledge is preserved under parallel repetition, and parallel repetition reduces the soundness errors of IPPs at an exponential rate (see, e.g., [GGR15, Appendix A]), the resulting protocol is an honest-verifier statistical zero-knowledge proof of proximity.

Finally, we argue about the efficiency of $\mathsf{V}_{\text{expan}}$ (the analysis of the simulator's efficiency is similar). The verifier $\mathsf{V}_{\text{expan}}$ needs to provide $\mathsf{V}^{P_u^\ell, U_{[n]}}(k)$ samples from P_u^ℓ and $U_{[n]}$. Generating a random sample from $U_{[n]}$ is easy and requires $O(\log n)$ random coins. Generating a random sample from P_u^ℓ is standard as well, requires $\mathsf{poly}(\ell, d) = \mathsf{poly}(\log n)$ random coins and oracle calls

to the input graph. By Lemma 2.14, it follows the $\mathsf{V}_{\text{expan}}$'s running time (accounting for the parallel repetition as well) is at most $\text{poly}(\log(n), 1/\varepsilon, k)$. □

4.2.1 Analyzing Completeness — Proving Lemma 4.14

Lemma 4.14 is an easy implication of the following standard result regarding random walks on expanders.

Lemma 4.16 (Expanders are Rapidly Mixing (c.f. [NS10, proof of Theorem 2.1])). *Suppose that G is an α-expander graph on n vertices with bounded degree d. Then for every vertex u and $\ell \in \mathbb{N}$ it holds that* $\text{SD}\left(P_u^\ell, U_{[n]}\right) \leq \sqrt{n} \cdot e^{-\frac{\alpha^2}{8d^2} \cdot \ell}$.

Proof of Lemma 4.14. From the choice of ℓ and Lemma 4.16, it holds that

$$\text{SD}\left(P_u^\ell, U_{[n]}\right) \leq 0.01,$$

for every vertex u. Let W be the random variable induced by the values of u chosen in step 1 of a random execution of the protocol It follows that

$$\Pr\left[\mathsf{V}_{\text{expan}}{}^{G}_{\alpha,d}(\varepsilon, n, k) \text{ accepts}\right] = \Pr\left[\mathsf{V}^{P_W^\ell, U_{[n]}}(k) \text{ accepts}\right] \geq 1 - \mathsf{negl}(k),$$

where the last inequlity follows from the completeness property of (P, V)(Lemma 2.14). □

4.2.2 Analyzing Soundness — Proving Lemma 4.15

The soundness of the expander protocol follows from the following combinatoral property of graphs that are far from expanders.

Lemma 4.17 ([CS10, Corollary 4.6 and Lemma 4.7]). *Let G be a graph on n vertices with bounded degree d. There exists a constant $c = c(d) > 0$ such that the following holds. If G is ε-far from any β-expander with $\beta \leq 1/10$, then there exists $\mathcal{U} \subset [n]$ with $|\mathcal{U}| \geq \varepsilon \cdot n/24$ such that for every $u \in \mathcal{U}$, it holds that* $\text{SD}\left(P_u^\ell, U_{[n]}\right) \geq \frac{1-2\varepsilon}{4}$, *where $\ell \leq 1/(10c\beta)$.*

Recall that $\mathsf{V}_{\text{expan}}$'s first steps is to choose uniformly at random a vertex u. This u will belong to the set $\mathcal{U}$ from the Lemma 4.17 with probability at least $\varepsilon/24$. Conditioned on the latter, we claim that the input to $\left(\mathsf{P}^{P_u^\ell, U_{[n]}}(k), \mathsf{V}^{P_u^\ell, U_{[n]}}(k)\right)$ is a NO input and so soundness follows immediately from the soundness of the protocol for $\overline{\mathsf{SD}}$ (Lemma 2.14).

We argue soundness with respect to $\beta = \left\lfloor \frac{\alpha^2}{20 \cdot c \cdot d^2 \cdot \log(\sqrt{n}/0.01)} \right\rfloor$, where $c = c(d) > 0$ is the constant guaranteed to exist by Lemma 4.17.

Proof of Lemma 4.15. Let $\widehat{\mathsf{P}}$ be any prover strategy. Let W be the random variable induced by the values of u chosen in step 1 of a random execution of the protocol and let $\mathcal{U}$ be the set guaranteed to exist by Lemma 4.17. It holds that

$$\Pr\left[\mathsf{V}_{\text{expan}}{}^{G}_{\alpha,d}(\varepsilon, n, k) \text{ accepts}\right] \leq \Pr[W \notin \mathcal{U}] + \Pr\left[\mathsf{V}_{\text{expan}}{}^{G}_{\alpha,d}(\varepsilon, n, k) \text{ accepts} \;\middle|\; W \in \mathcal{U}\right]. \tag{12}$$

We bound both terms in the right-hand side of Equation (12). Lemma 4.17 yields that $\Pr[W \notin \mathcal{U}] \leq 1-\varepsilon/24$. As for the second term, Lemma 4.17 yields that $\mathrm{SD}\left(P^{\ell}_{w}, U_{[n]}\right) \geq \frac{1-2\varepsilon}{4} \geq 0.2$ for every $w \in \mathcal{U}$ (note that β was chosen so that $\ell \leq 1/(10c\beta)$ and that we assumed above that $\varepsilon < 0.1$). It follows that

$$\Pr\left[\mathsf{V}_{\mathrm{expan}}{}^{G}_{\alpha,d}(\varepsilon, n, k) \text{ accepts} \;\middle|\; W \in \mathcal{U}\right] \leq \mathrm{E}_{u \leftarrow \mathcal{U}}\left[\Pr\left[\mathsf{V}^{P^{\ell}_{u}, U_{[n]}}(k) \text{ accepts}\right]\right] \leq \mathsf{negl}(k), \tag{13}$$

where the last inequality follows from the soundness properties of V (Lemma 2.14). □

4.3 Promise Bipartiteness is in HV-SZKPP

In this section we consider the property of a graph being bipartite in the bounded degree model (we introduced this model in Section 4.2). The property of being a bipartite graph was first considered by Goldrich and Ron [GR02, GR99]. They showed a tester for bipartiteness of graphs with n vertices which makes at most $\tilde{O}(\sqrt{n})$ queries. They also showed a matching lower bound, namely that any such tester must make at least $\Omega(\sqrt{n})$ queries.

Rothblum et al. [RVW13] gave an interactive proof of proximity for the following promise version of this property in which the verifier's running time is $\Theta(\log n)$. In this version, YES instances are bipartite graphs. NO instances, in addition to being far from bipartite, are also well-mixing, namely that a random walk of $\Theta(\log n)$ steps ends at each vertex with probability at least $1/2n$ (e.g., expanders). We denote this property by BIPARTITE. In [RVW13]'s protocol, the verifier takes a random walk of length $\Theta(\log n)$ starting at a randomly chosen vertex (in each step it performs a self-loop with probability at least $1/2$; see Definition 4.13). The verifier then sends the prover the start and end vertices of the walk and asks the prover to tell him the parity of the number of non-self-loop steps it took during the walk.

If the graph is bipartite, then the parity of the number of non-self-loops is equal to the parity of the shortest simple path from the start vertex to the end vertex. [RVW13] showed that if the graph is far from being bipartite and well-mixing, then the chance of taking a path of even non-self-loop steps is close to that of taking a path of odd non-self-loop steps. Hence, any cheating prover will fail to convince the verifier.

In fact, it is easy to see that the above protocol is also an honest-verifier perfect zero-knowledge proof of proximity. The simulator will simply act as the verifier: take a random walk and output the parity of the non-self-loop steps in this walk (which the simulator knows since it performs the walk). Since this result follows immediately from [RVW13]'s protocol, we only state an informal version of the it here, and refer the reader to [RVW13] for formal definitions and description of the protocol.

Theorem 4.18 ([RVW13, Section 5.1], informal)**.** BIPARTITE $\in$ HV-PZKPP$[\mathsf{poly}(\log(N), k, 1/\varepsilon)]$.

It is an interesting open question to show a *cheating-verifier* zero-knowledge proof of proximity for BIPARTITE.

5 Limitations of SZKPP

In light of the positive results in Section 4 an important questions rises:

Does every property that has a sub-linear IPP *also have a sub-linear* statistical zero-knowledge IPP*?*

We give a negative answer to the above question.[18] Actually we show two incomparable lower bounds:

1. There exists a property Π that has an IPP in which the verifier runs in poly-logarithmic time, but the verifier in any zero-knowledge proof of proximity for Π cannot run in poly-logarithmic time. (Actually we can even show that such a verifier cannot run in time $N^{o(1)}$, see Remark 5.6).Thus, this lower bound can be viewed as a separation between the class $\mathsf{IPP}\big[\mathsf{poly}(\log(N), k, 1/\varepsilon)\big]$ and $\mathsf{HV\text{-}ESZKPP}\big[\mathsf{poly}(\log(N), k, 1/\varepsilon)\big]$.
2. We show an additional lower bound which separates $\mathsf{HV\text{-}ESZKPP}\big[\mathsf{poly}(\log(N), k, 1/\varepsilon)$ even from a weaker class - namely the class of languages admitting *non-interactive* proofs of proximity, also known as Merlin-Arthur proofs of proximity or MAPs [GR16]. However, in contrast to the previous separation from IPPs, this result is conditional: we can only prove it assuming a (very plausible) circuit lower bound. Specifically, we assume that (randomized) $\mathsf{DNF}_{\oplus}$, namely DNF formulas composed with one layer of parity gates (see [CS16, ABG+14] and references therein), cannot compute the disjointness function. This circuit lower bound is implied by the assumption that the Arthur-Merlin communication complexity of disjointness is n^{ε}, for inputs of length n and some constant $\varepsilon > 0$.

5.1 $\mathsf{IPP} \not\subseteq \mathsf{ESZKPP}$

We show that there exists a property $\Pi \in \mathsf{IPP}[\mathsf{poly}(\log(N), k, 1/\varepsilon)]$ but $\Pi \notin \mathsf{HV\text{-}ESZKPP}[\mathsf{poly}(\log(N), k, 1/\varepsilon)]$. Namely, Π that has an efficient IPP, which *unconditionally* cannot have such a statistical zero-knowledge IPP.[19]

Theorem 5.1. $\mathsf{IPP}\big[\mathsf{poly}(\log(N), k, 1/\varepsilon)\big] \not\subseteq \mathsf{HV\text{-}ESZKPP}\big[\mathsf{poly}(\log(N), k, 1/\varepsilon)\big]$.

The proof of Theorem 5.1 is done in two steps. The first step is to argue the existence of a property Π which has an interactive proof of proximity with a large number of rounds and $\mathsf{polylog}(N)$-time verifier , but such that in every 2-*message* interactive proof of proximity for Π, the verifier's running time must be N^{δ}, for some constant $\delta > 0$. Actually, such a result was recently established by Gur and Rothblum [GR17]:

Lemma 5.2 ([GR17, Theorem 1])**.** *The exists* $\Pi \in \mathsf{IPP}[\mathsf{poly}(\log(N), k, 1/\varepsilon)]$ *such that the verifier in every* 2*-message* IPP *for* Π*, with respect to proximity parameter* $\varepsilon = 1/10$ *and completeness and soundness error* $1/3$*, must run in time* $\Omega\big(N^{\delta}\big)$*, for some universal constant* $\delta > 0$*.*

The second step in proving Theorem 5.1 is a general round reduction transformation for any honest-verifier statistical zero-knowledge proof of proximity. Namely, we would like a procedure that takes any *many-messages* honest-verifier zero-knowledge proof of proximity and turns it into a 2-*message* honest-verifier zero-knowledge proof of proximity while only slightly deteriorating the verifier's and simulator's running times. Specifically, we show the following lemma.

Lemma 5.3 (Efficient Round Reduction for SZKPP)**.** *Suppose that the property* Π *has an honest-verifier statistical zero-knowledge* ε-IPP *such that for every input length* $N \in \mathbb{N}$ *and security parameter* $k \in \mathbb{N}$ *the*

[18] We emphasize that here we refer to *statistical* zero-knowledge. Indeed, in Section 6 below we show that for *computational* zero-knowledge such a transformation is possible, for a large class of IPPs (see Theorem 6.2).

[19] Our actualy result refers to statistical zero-knowledge with *expected* simulation bounds, but this only makes our result stronger.

simulator's expected running time is bounded by $t_{\mathsf{S}}(\varepsilon, N, k) = t'_{\mathsf{S}}(\varepsilon, N) \cdot \mathsf{poly}(k)$ *and for every value of* ε*, the function* $t'_{\mathsf{S}}(\varepsilon, \cdot)$ *is monotone non-decreasing.*

Then, Π *has a 2-message honest verifier statistical zero-knowledge* ε-IPP *such that for every input length* N *and security parameter* k *the running time of the verifier is* $\mathsf{poly}(t_{\mathsf{S}}(\varepsilon, N, k'), k)$*, for* $k' = \mathsf{poly}(t'_{\mathsf{S}}(\varepsilon, N))$.

For the setting of poly-logarithmic zero-knowledge proof of proximity, Lemma 5.3 can be stated as follows.

Corollary 5.4. *Every* $\Pi \in \mathsf{HV\text{-}ESZKPP}[\mathsf{poly}(\log(N), k, 1/\varepsilon)]$ *has a 2-message honest-verifier statistical zero-knowledge* ε-IPP *with expected simulation, such that the verifier's running time is* $\mathsf{poly}(\log(N), k, 1/\varepsilon)$.

Remark 5.5 (Comparison with the Babai-Moran [BM88] Round)**.** *Lemma 5.3 and Corollary 5.4 should be contrasted with the classical round reduction of interactive proofs, due to Babai and Moran [BM88] (and shown in [RVW13] to hold also for* IPP*s). In contrast to Lemma 5.3, the Babai-Moran round reduction increases the complexity of the verifier exponentially in the round complexity of the original protocol. In contrast, the overhead in Lemma 5.3 is only polynomial, which is crucial for our lower bound.*

The proof of Lemma 5.3 is a direct application of the proof that the promise problem Entropy Difference (ED, see Definition 2.12) is complete for the class SZK (see [Vad99]). That proof takes an instance x of any promise problem $\Pi = (\Pi_{\mathsf{YES}}, \Pi_{\mathsf{NO}}) \in \mathsf{SZK}$ and efficiently constructs two distributions X and Y such that if $x \in \Pi_{\mathsf{YES}}$ then $\mathrm{H}(X) \geq \mathrm{H}(Y) + 1$, and if $x \in \Pi_{\mathsf{NO}}$ then $\mathrm{H}(Y) \geq \mathrm{H}(X) + 1$. That proof goes on to show a zero-knowledge protocol to distinguish between the case that $\mathrm{H}(X) \geq \mathrm{H}(Y) + 1$ and the case that $\mathrm{H}(Y) \geq \mathrm{H}(X) + 1$. Two important points regarding that proof: (1) sampling from X and Y can be done by running (many times) the simulator for the original problem Π; (2) the protocol for ED consists of only two messages and requires only sample access to X and Y (we stated this fact in Lemma 2.15).

In our settings, we can view a property Π as a promise problem where functions possessing the property are in Π_{YES} and functions that are ε-far from possessing the property are in Π_{NO}. Then, we can have the verifier "run" the reduction to ED and apply the sample-access protocol for ED. The unbounded prover will behave as in the protocol for ED. Recall that the original simulator (i.e., the one for the property's IPP) required only oracle access to the input function. Since sampling from the distributions only requires running the original simulator, the new verifier can implement this step with only oracle access to the input function and with only polynomial overhead to the running time of the original simulator. We defer the actual proof of Lemma 5.3 to Appendix B.1.

Using Lemmas 5.2 and 5.3 we can now prove Theorem 5.1.

Proof of Theorem 5.1. Let Π be the property guaranteed to exist by Lemma 5.2. Assume towards a contradiction that $\Pi \in \mathsf{HV\text{-}ESZKPP}[\mathsf{poly}(\log(N), k, 1/\varepsilon)]$. Namely, Π has an honest-verifier statistical zero-knowledge interactive proof of proximity with the simulator's expected running time being $(\log(N))^{\alpha} \cdot k^{\beta} \cdot (1/\varepsilon)^{\gamma}$ for constants $\alpha, \beta, \gamma > 0$. Applying Lemma 5.3 with respect to Π yields that Π has a 2-message ε-IPP (P, V), with V's running time being $(\log(N))^{\delta_1} \cdot k^{\delta_2} \cdot (1/\varepsilon)^{\delta_3}$ for constants $\delta_1 = \delta_1(\alpha, \beta), \delta_2, \delta_3 = \delta_3(\beta, \gamma) > 0$.

Set $\varepsilon = 1/10$ and k such that the soundness error of (P, V) is at most $1/3$. Note that in this setting, V's running time is $O(\log(N)^{\delta_1}) = \mathsf{poly}(\log(N))$. This is a contradiction to Lemma 5.2. □

Remark 5.6. *We remark that the proof of Theorem 5.1 actually establishes the stronger result that* Π *cannot even have an* HV-ESZKPP *protocol in which the verifier runs in time* $N^{o(1)} \cdot \mathsf{poly}(k, 1/\varepsilon)$*. Indeed, assuming*

a simulator with expected running time $N^{o(1)} \cdot^{\alpha} \cdot k^{\beta} \cdot (1/\varepsilon)^{\gamma}$, Lemma 5.3 yields that Π has a 2 ε-IPP in which the verifier runs in $N^{o(1)}$ time, in contradiction to Lemma 5.2.

5.2 MAP $\not\subseteq$ ESZKPP, assuming Circuit Lower Bounds

We show that there exists a property $\Pi \in \mathsf{MAP}\left[\mathsf{poly}(\log(N), k, 1/\varepsilon)\right]$ but, assuming certain circuit lower bounds, it holds that $\Pi \notin \mathsf{HV\text{-}ESZKPP}\left[\mathsf{poly}(\log(N), k, 1/\varepsilon)\right]$

Let t-$\mathsf{DNF}_{\oplus}$ refer to depth 3 circuits, whose output gate is an bounded fan-in OR gate, intermediate level are composed of fan-in t AND gates and third layer is composed of (unbounded fan-in) parity gates. The size of a t-$\mathsf{DNF}_{\oplus}$ gate is the fan-in of its top gate. A randomized t-$\mathsf{DNF}_{\oplus}$ simply refers to a distribution over t-$\mathsf{DNF}_{\oplus}$ circuits. We say that a randomized t-$\mathsf{DNF}_{\oplus}$ circuit $C : \{0,1\}^k \to \{0,1\}$ computes a function f if for every $x \in \{0,1\}^k$ it holds that $\Pr[C(x) = f(x)] \geq 2/3$.

For any $k \in \mathbb{N}$ and strings $x, y \in \{0,1\}^k$, we define $\mathsf{DISJ}_k(x,y) = 1$ if for every $i \in [k]$ it holds that either $x_i = 0$ or $y_i = 0$ and $\mathsf{DISJ}_k(x,y) = 0$ otherwise. The following conjecture states that small randomized $\mathsf{DNF}_{\oplus}$ circuits cannot compute DISJ.[20]

Conjecture 5.7. *There exists a constant $\delta > 0$ such that every randomized t-$\mathsf{DNF}_{\oplus}$ of size S that computes DISJ_k it holds that $\min(t, \log(S)) = \Omega(k^{\delta})$.*

We remark that a randomized t-$\mathsf{DNF}_{\oplus}$ circuit of size S yields an Arthur-Merlin communication complexity with complexity $\log(S) + t$.[21] To the best of our knowledge, it is believed that the Arthur-Merlin communication complexity of disjointness is believed to be $\Omega(k)$ (which would imply Conjecture 5.7 with $\delta = 1$). We mention that proving any non-trivial Arthur-Merlin communication complexity lower bound is a notorious open problem.

Theorem 5.8. *If Conjecture 5.7 holds, then* $\mathsf{MAP}\left[\mathsf{poly}(\log(N), k, 1/\varepsilon)\right] \not\subseteq \mathsf{HV\text{-}ESZKPP}\left[\mathsf{poly}(\log(N), k, 1/\varepsilon)\right]$.

We begin by an outline of the proof. Our main tool will be a binary linear error-correcting $C : \{0,1\}^k \to \{0,1\}^n$, with constant relative distance and almost-linear[22] blocklength, which is also *locally testable* and *locally decodable*. A code $C : \{0,1\}^k \to \{0,1\}^n$ is locally testable if there exists a procedure that makes only few queries to a word $w \in \{0,1\}^n$, and determines with high probability if it is a codeword (i.e., if $w = C(x)$ for some message $x \in \{0,1\}^k$) or far from the code (see Definition 5.9 for the formal definition). A code is locally decodable if there exists a procedure that takes as input an index $i \in [k]$ and a word $w \in \{0,1\}^n$ close to some codeword $C(x)$, makes only few queries to w, and outputs x_i with high probability (see Definition 5.10 for the formal definition).

The property that we consider is the *Code Intersection* (CI) property. This property consists of pairs of codewords $(C(x), C(y))$, coded under the foregoing code, such that $\mathsf{DISJ}(x,y) = 0$ (i.e., x and y intersect). This problem was previously considered by Gur and Rothblum [GR16] who showed that it has a very efficient MAP (we re-prove this fact since we use a slightly different code).

[20] In contrast, note that there is a very simple CNF formula for computing DISJ.

[21] First, Alice and Bob choose a $\mathsf{DNF}_{\oplus}$ circuit from the distribution and specify it to Merlin. Merlin then sends an index of which term in the circuit is satisfied. A single term is an fan-in t AND gate composed with parity gates. Alice and Bob can compute this term's value using $2t$ communication, by having them send to each other their respective contributions to each of the t parities.

[22] Note that we are using the term "linear" in two different ways. First, the code is a linear function of the message. Second, the length of the codeword is almost linear in the length of messages.

Indeed, it is easy to see that CI has a very efficient MAP. Merlin simply sends to Arthur the index i on which x and y intersect. Arthur, using the local testability, will verify that the input is close to a pair of codewords, and then locally decodes x_i and y_i. Arthur accepts iff $x_i = y_i = 1$. This proof of proximity, however, reveals a lot to Arthur (and in particular is not zero-knowledge). Specifically, Arthur learns the index of the intersection. As a matter of fact, this is not a coincidence. We show that, assuming that Conjecture 5.7 holds, the property CI does not have any honest-verifier zero-knowledge IPP with poly-logarithmic complexity.

To see how we prove the lower bound, consider the promise problem *Code Disjointness* (CD), in which the YES instances are pairs of codewords $(C(x), C(y))$ such that $\mathsf{DISJ}(x, y) = 1$, and NO instances are pairs of codewords $(C(x), C(y))$ such that $\mathsf{DISJ}(x, y) = 0$. Note that NO instances of CD are in the property CI. Moreover, YES instances of CD are $\delta(C)/2$-far from CI, where $\delta(C)$ is the relative distance of the code C.

Assume, toward a contradiction, that CI has an honest-verifier statistical zero-knowledge IPP with poly-logarithmic complexity. We argue that this implies that the complement promise problem of CD has a *constant-round* IPP. The latter fact basically follows from the fact that entropy difference (ED) is complete for the class of promise problems having a statistical zero-knowledge proof, and is itself closed under complement.

Thus, we have constructed an IPP which accepts inputs from CD and rejects inputs from CI. Using a result of Rothblum *et-al.* [RVW13], we can derive from this IPP a quasi-polynomial size randomized DNF for the same promise problem. We further observe that since the code C is a *linear* code, we have obtained a circuit that computes the disjointness function on input (x, y) by first applying a linear transformation and then the aforementioned randomized DNF. Or in other words, a quasi-polynomial sized $\mathsf{DNF}_{\oplus}$ circuit. This contradicts Conjecture 5.7.

We proceed to the formal proof of Theorem 5.8. We begin with definitions and notations. An error-correcting code is an injective function $C : \{0,1\}^k \to \{0,1\}^n$. The code C is said to have relative distance $\delta(C)$ if for any $x \neq x' \in \{0,1\}^k$ it holds that $\Delta(C(x), C(x')) \geq \delta(C)$.[23] Throughout this work we deal with (uniform) algorithms, and so we will need (families of) error-correcting codes. Formally, for a parameters $k = k(\ell) \geq 1$ and $= n(\ell) \geq k(\ell)$ we define an ensemble of error correcting code as an ensemble $C = \left(C_\ell : \{0,1\}^{k(\ell)} \to \{0,1\}^{n(\ell)}\right)_{\ell \in \mathbb{N}}$ of error-correcting codes. An ensemble of error correcting codes $C = (C_\ell)_{\ell \in \mathbb{N}}$ is said to have relative distance $\delta(C)$ if for all sufficiently large ℓ, each code C_ℓ in the ensemble has relative distance $\delta(C)$.

We next formally define locally testable and decodable codes.

Definition 5.9 ((strong) locally testable codes (c.f. [GS06])). *Let $t \colon \mathbb{N} \to \mathbb{N}$. A ensemble of error-correcting code $C = \left(C_\ell : \{0,1\}^{k(\ell)} \to \{0,1\}^{n(\ell)}\right)_{\ell \in \mathbb{N}}$ is t-*locally-testable *if there exists a probabilistic algorithm (tester) T that, given explicit input ℓ and oracle access to $w \in \{0,1\}^{n(\ell)}$, runs in time $t(\ell)$, and satisfies the following.*

- **Completeness:** *For every $x \in \{0,1\}^{k(\ell)}$ it holds that $\Pr\left[T^{C_\ell(x)}(\ell) = 1\right] = 1$.*
- **Soundness:** *For every $w \in \{0,1\}^{n(\ell)}$ it holds that $\Pr[T^w(\ell) = 0] \geq \Omega(\Delta(w, \mathrm{Im}(C_\ell)))$, where $\Delta(w, \mathrm{Im}(C_\ell))$ is the relative distance of w from the code.*[24]

[23] Recall that the relative distance between $y \in \{0,1\}^n$ and $y' \in \{0,1\}^n$ is defined as $\Delta(y, y') = \frac{\left|\{y_i \neq y'_i \colon i \in [n]\}\right|}{n}$.

[24] Recall that the relative distance of $x \in \{0,1\}^n$ from a non-empty set $\mathcal{S} \subseteq \{0,1\}$ is defined as $\Delta(x, \mathcal{S}) = \min_{y \in \mathcal{S}} \Delta(x, y)$.

Definition 5.10 (locally decodable codes (c.f. [KT00])). *Let $t\colon \mathbb{N} \to \mathbb{N}$. A ensemble of error-correcting code $C = \left(C_\ell : \{0,1\}^{k(\ell)} \to \{0,1\}^{n(\ell)}\right)_{\ell\in\mathbb{N}}$ is t-*locally-decodable *if there exists a constant $\delta_{\mathsf{radius}} \in (0, \delta(C)/2)$ and a probabilistic algorithm (decoder) D that, given oracle access to $w \in \{0,1\}^n$ and explicit inputs $i \in [k]$ and $\ell \in \mathbb{N}$, runs in time $t(\ell)$ and satisfies the following.*

- **Completeness:** *For every $i \in [k(\ell)]$ and $x \in \{0,1\}^{k(\ell)}$, it holds that $\Pr\left[D^{C(x)}(i) = x_i\right] = 1$.*
- **Soundness:** *For every $i \in [k(\ell)]$ and every $w \in \{0,1\}^{n(\ell)}$ with $\Delta(w, C(x)) \leq \delta_{\mathsf{radius}}$, it holds that $\Pr[D^w(i) = x_i] \geq 2/3$.*[25]

We use the following well-known fact.

Lemma 5.11. *There exists an ensemble of binary linear codes $C = \left(C_\ell : \{0,1\}^{k(\ell)} \to \{0,1\}^{n(\ell)}\right)_{\ell\in\mathbb{N}}$, for $k(\ell) = \tilde{O}(\ell)$ and $n(\ell) \leq k(\ell)^{1.01}$, whose relative distance is some constant $\delta > 0$ and that is $\mathsf{polylog}(\ell)$-locally-testable and $\mathsf{polylog}(\ell)$-locally-decodable.*

See Appendix B.3 for a sketch of the construction (which is basically the concatenation of the low degree extension code, over a field of poly-logarithmic size, with a good binary code).

Using Lemma 5.11, we can now define the property *Code Intersection*.

Definition 5.12 (Code Intersection). *Let $C = (C_\ell)_{\ell\in\mathbb{N}}$ be the code guaranteed to exist by Lemma 5.11. For $\ell \in \mathbb{N}$, let*

$$\mathsf{CI}_\ell = \left\{ \left(C_\ell(x), C_\ell(y)\right) : x, y \in \{0,1\}^{k(\ell)} \text{ such that } \mathsf{DISJ}_{k(\ell)}(x,y) = 0 \right\}.$$

We define the Code Intersection *property as $\mathsf{CI} = (\mathsf{CI}_\ell, [2n(\ell)], \{0,1\})_{\ell\in\mathbb{N}}$.*

The proof of Theorem 5.8 follows immediately from the next two lemmas, proven in Sections 5.2.1 and 5.2.2.

Lemma 5.13. $\mathsf{CI} \in \mathsf{MAP}\left[\mathsf{poly}(\log(N), k, 1/\varepsilon)\right]$.

Lemma 5.14. *If Conjecture 5.7 holds, then $\mathsf{CI} \notin \mathsf{HV\text{-}ESZKPP}\left[\mathsf{poly}(\log(N), k, 1/\varepsilon)\right]$.*

5.2.1 Proving Lemma 5.13

Consider the protocol $(\mathsf{P}_{\mathsf{CI}}, \mathsf{V}_{\mathsf{CI}})$ from Fig. 4. Perfect completeness follows from the perfect completeness in the local testing and decoding procedures. We proceed to argue that soundness holds.

Fix $\varepsilon > 0$, sufficiently large $\ell \in \mathbb{N}$ and $w = (w_1, w_2) \in \{0,1\}^{2\cdot n(\ell)}$ such that w is ε-far from CI_ℓ. We assume without loss of generality that $\varepsilon < \delta_{\mathsf{radius}}$ (otherwise "reset" ε to δ_{radius}). We consider two cases:

$\Delta(w_1, \mathrm{Im}(C_\ell)) \geq \varepsilon/2$ **or** $\Delta(w_2, \mathrm{Im}(C_\ell)) \geq \varepsilon/2$**:** Let $j \in \{1,2\}$ such that $\Delta(w_j, \mathrm{Im}(C_\ell)) \geq \epsilon/2$. By the soundness condition of the tester T, it holds that

$$\Pr[\mathsf{V}_{\mathsf{CI}}{}^w(\varepsilon, n(\ell)) \text{ rejects}] \geq \Pr[\mathsf{T}^{w_j}(\ell) = 0] \geq \Omega(\Delta(w_j, \mathrm{Im}(C_\ell))) \geq \Omega(\varepsilon/2).$$

[25] Since $\delta_{\mathsf{radius}} \leq \delta_C/2$ the message x is unique.

The Code Intersection Protocol $(\mathsf{P}_{\mathsf{CI}}, \mathsf{V}_{\mathsf{CI}})$

Prover's Input: A pair of strings $(w_1, w_2) \in \{0,1\}^{2 \cdot n(\ell)}$ and proximity parameter $\varepsilon > 0$.
Verifier's Input: ℓ, $n(\ell)$, ε and oracle access to (w_1, w_2).

Let C be the code ensemble from Lemma 5.11.
Let T be the tester from Definition 5.9 with respect to C.
Let $\delta_{\mathsf{radius}}(C)$ and D be the decoding radius and decoder, respectively, from Definition 5.10 with respect to C.

1. P_{CI} finds $i \in [k(\ell)]$ such that $w = (C_\ell(x), C_\ell(y))$ for some $x, y \in \{0,1\}^{k(\ell)}$ and $x_i = y_i$. Sends i to V_{CI}.
2. V_{CI} acts as follows:
   (a) Set $\varepsilon = \min\{\varepsilon, 2\delta_{\mathsf{radius}}(C)\}$.
   (b) Run $\mathsf{T}^{w_1}(\ell)$ and $\mathsf{T}^{w_2}(\ell)$ and reject if any of them rejects.
   (c) Accept if $\mathsf{D}^{w_1}(i, \ell) = \mathsf{D}^{w_2}(i, \ell) = 1$, and otherwise reject.

Figure 4: The Code Intersection Protocol

$\Delta(w_1, \mathrm{Im}(C_\ell)) \leq \varepsilon/2$ **and** $\Delta(w_2, \mathrm{Im}(C_\ell)) \leq \varepsilon/2$**:** Fix a cheating prover $\widehat{\mathsf{P}}$. Assume without loss of generality that $\widehat{\mathsf{P}}$ is deterministic and let i^* be the index it sends to V_{CI} in step 1. Let $x, y \in \{0,1\}^{k(\ell)}$ such that $\Delta(w_1, C_\ell(x)) \leq \varepsilon/2$ and $\Delta(w_2, C_\ell(y)) \leq \varepsilon/2$ (such x and y are unique since $\varepsilon \leq \delta_{\mathsf{radius}}(C)$). Moreover, as w is ε-far from CI_ℓ, it must be that either $x_{i^*} = 0$ or $y_{i^*} = 0$. Observe that if $x_{i^*} = 0$, then by the soundness of the decoding procedure, with probability $2/3$, the decoder will output 0 in which case our verifier rejects. The case that $y_{i^*} = 0$ is analyzed similarly.

Combining both conditions, it holds that $\Pr[\mathsf{V}_{\mathsf{CI}}{}^w(\varepsilon, n(\ell)) \text{ rejects}] \geq \min\{\Omega(\varepsilon/2), 1/3\} = \Omega(\varepsilon)$.

So far we have shown that the code intersection protocol (Fig. 4) has prefect completeness and soundness error $1 - O(\varepsilon)$. To reduce the soundness error it suffices to have the verifier repeat its check $\mathsf{poly}(k)/\varepsilon$ times.[26] As shown in [GR16] this reduces the soundness error to 2^{-k} and so the resulting protocol is an ε-MAP.

Finally, it is easy to verify that the ultimate verifier run in time $\mathsf{poly}(\log(\ell), k, 1/\varepsilon)$ which, since the input length (i.e., $2 \cdot n(\ell)$) is $\mathsf{poly}(\ell)$, is $\mathsf{poly}(\log(N), k, 1/\varepsilon)$.

5.2.2 Proof of Lemma 5.14

We prove the contrapositive. Assume that $\mathsf{CI} \in \mathsf{HV\text{-}ESZKPP}\big[\mathsf{poly}(\log(N), k, 1/\varepsilon)\big]$ and consider the promise problem of *Code Disjointness*.

Definition 5.15 (Code Disjointness)**.** *For $\ell \in \mathbb{N}$, let*

$$\mathsf{CD}_{\mathsf{YES},\ell} = \Big\{ \big(C_\ell(x), C_\ell(y)\big) : x, y \in \{0,1\}^{k(\ell)} \textit{ such that } \mathsf{DISJ}_{k(\ell)}(x, y) = 1 \Big\}$$
$$\mathsf{CD}_{\mathsf{NO},\ell} = \Big\{ \big(C_\ell(x), C_\ell(y)\big) : x, y \in \{0,1\}^{k(\ell)} \textit{ such that } \mathsf{DISJ}_{k(\ell)}(x, y) = 0 \Big\}.$$

[26] Note that here k refers to the security parameter and not the message length $k(\ell)$.

Let $\mathsf{CD} = \left(\mathsf{CD}_{\mathsf{YES},\ell}, \mathsf{CD}_{\mathsf{NO},\ell}\right)_{\ell\in\mathbb{N}}$.

Note that the input length here is $N = 2 \cdot n(\ell) = \tilde{O}(\ell^{1.01})$. Hence, the query complexity and communication complexity of the IPP are $\mathsf{poly}(\log(\ell), k, 1/\varepsilon)$.

We will use the zero-knowledge proof of proximity for CI to design a randomized $\mathsf{DNF}_{\oplus}$ circuit that solves disjointness. Note that by definition $\mathsf{CD}_{\mathsf{NO},\ell} = \mathsf{CI}_\ell$.

Observe that every string $w \in \mathsf{CD}_{\mathsf{YES},\ell}$ is $\delta(C)/2$-far from CI_ℓ. Thus, an (honest-verifier) statistical zero-knowledge IPP for CI immediately yields an (honest-verifier) statistical zero-knowledge proof for the complement of CD. Recall that in Section 5.1 we used that entropy difference (ED) is complete for the class SZK. Here, we will use this fact again, plus that there is an easy reduction from ED to its complement, to show the following claim, proven in Appendix B.2.

Claim 5.16. *The promise problem* CD *has an interactive proof system with the following properties.*

1. *The verifier gets ℓ as explicit input and oracle access to $w \in \mathsf{CD}_{\mathsf{YES},\ell} \cup \mathsf{CD}_{\mathsf{NO},\ell}$.*
2. *The completeness and soundness errors are both $1/3$.*
3. *The verifier's running time is $\mathsf{poly}(\log(\ell))$.*
4. *The parties exchange a constant number of messages.*

Using the Goldwasser-Sipser [GS89] transformation from private-coin to public-coin interactive proofs and the Babai-Moran [BM88] round reduction (see [RVW13, Section 4] for more details). We obtain a 2-message Arthur Merlin interactive proof, where the verifier runs in time $\mathsf{polylog}(\ell)$. Applying an additional transformation from such proof-systems to randomized DNFs due to [RVW13] (see also [GR17]), we can obtain the following:

Claim 5.17 (Based on [RVW13, Section 4])**.** *There exists a randomized $\mathsf{polylog}(\ell)$-DNF of size $2^{\mathsf{polylog}(\ell)}$ that computes (the promise problem)* CD *for inputs of size $2 \cdot n(\ell)$.*

By observing that all $w \in \mathsf{CD}_{YES,\ell} \cup \mathsf{CD}_{\mathsf{NO},\ell}$ are composed of two codewords, and the the code is a binary *linear* error correcting code, Claim 5.17 implies that there exists a randomized $\mathsf{polylog}(\ell)$-$\mathsf{DNF}_{\oplus}$ circuit of size $2^{\mathsf{polylog}(\ell)}$ that computes $\mathsf{DISJ}_{k(\ell)}$. This contradicts Conjecture 5.7. This concludes the proof of Section 5.2.1.

Remark 5.18 (Relaxed Local Decoders and the [GGK15] Code)**.** *We remark that for our result, as in [GR16] it suffices for us to use* relaxed *local decoders (as put forth in [BGH+06]). Loosely speaking, relaxed local decoding allows the decoder to refuse to decode if it notices that the word is corrupt.*

Given that, it is tempting to ask why we did not use the locally testable and (relaxed) decodable codes of Golderich et-al. [GGK15]. Indeed, their codes have constant-query *whereas the code that we used requires poly-logarithmic query complexity. The only reason that we do not use the [GGK15] code is that the* computational *complexity of this code was not analyzed in [GGK15].*

6 Computational ZK Proofs and Statistical ZK Arguments of Proximity

In this section we show that, assuming reasonable cryptographic assumptions (specifically, the existence of one-way or collision-resistant hash functions), a large class of IPPs and arguments

of proximity[27] can be transformed to be zero-knowledge. As a consequence, using the results of [RVW13, RRR16, Kil92, BGH+06, DR06] we obtain computational ZK proofs of proximity for small-depth and for small-space computations, and statistical ZK arguments of proximity for all of NP.

Our transformation should be contrasted with an analogous transformation of Ben-Or *et-al.* [BGG+88] for classical public-coin interactive proofs (and arguments). Indeed, our transformation is based on the main idea of [BGG+88]. However, in contrast to their result, our transformation does not apply to *arbitrary* public-coin IPPs. Rather, it only applies to such IPPs in which the queries that the verifier makes do not depend on messages sent by the prover. We say that such IPPs make *prover-oblivious queries*.

Definition 6.1. *We say that an* IPP *makes* prover-oblivious queries *if the input locations that the verifier queries are fully determined by its random coin tosses and the answers to previous queries that it made. That is, the queries do not depend on messages sent by the prover.*

Thus, an IPP with prover-oblivious queries can be thought of as a two steps process. In the first step the verifier can make queries to its input but it is not allowed to interact with the prover. In the second step, the parties are allowed to interact but the verifier is no longer allowed to query the input.[28]

Interestingly (and crucially for our purpose), the *general purpose* IPPs and arguments of proximity in the literature are indeed public-coin and make only prover-oblivious queries. Using this fact, together with our transformation, we obtain general purpose ZK proofs of proximity.

Our main transformation is summarized in the following two theorems.

Theorem 6.2 (IPPs $\rightarrow$ Computational ZK)**.** *Assume that one-way functions exist. Suppose that the language $\mathcal{L}$ has an ℓ-message* public-coin IPP *with* prover oblivious queries *where the verifier runs in time $t_{\mathsf{V}} = t_{\mathsf{V}}(N,k,\varepsilon)$ and the (honest) prover runs in time $t_{\mathsf{P}} = t_{\mathsf{P}}(N,k,\varepsilon)$. Then, $\mathcal{L}$ has an $(\ell + \mathsf{poly}(k))$-message* computational ZKPP *in which the prover runs in time $t'_{\mathsf{P}}(N,k,\varepsilon) := (t_{\mathsf{P}}(N,k,\varepsilon) + \mathsf{poly}(t_{\mathsf{V}}(n,k,\varepsilon))) \cdot \mathsf{poly}(k)$ and the verifier runs in time $t'_{\mathsf{V}}(N,k,\varepsilon) := t_{\mathsf{V}}(N,k,\varepsilon) \cdot \mathsf{poly}(k)$. The simulation overhead (see discussion in Section 3) is $s(t_{\widehat{\mathsf{V}}}, N, k, \varepsilon) = t_{\widehat{\mathsf{V}}} \cdot \mathsf{poly}(k)$, for cheating verifiers that run in time $t_{\widehat{\mathsf{V}}} = t_{\widehat{\mathsf{V}}}(N,k,\varepsilon)$.*

Theorem 6.3 (Arguments of Proximity $\rightarrow$ Statistical ZK Arguments)**.** *Assume that one-way functions exist. Suppose that the language $\mathcal{L}$ has an ℓ-message* public-coin *interactive argument of proximity with* prover oblivious queries *where the verifier runs in time $t_{\mathsf{V}} = t_{\mathsf{V}}(N,k,\varepsilon)$ and the (honest) prover runs in time $t_{\mathsf{P}} = (N,k,\varepsilon)$. Then, $\mathcal{L}$ has an $(\ell \cdot \mathsf{poly}(k))$-message* statistical *zero-knowledge argument of proximity in which the prover runs in time $t'_{\mathsf{P}}(N,k,\varepsilon) := (t_{\mathsf{P}}(N,k,\varepsilon) + \mathsf{poly}(t_{\mathsf{V}}(N,k,\varepsilon))) \cdot \mathsf{poly}(k)$ and the verifier runs in time $t'_{\mathsf{V}}(N,k,\varepsilon) := t_{\mathsf{V}}(N,k,\varepsilon) \cdot \mathsf{poly}(k)$.*

Furthermore, if there exist collision-resistant hash functions, then the round complexity of the foregoing argument-system can be reduced to $(\ell + O(1))$.

Our proof of Theorems 6.2 and 6.3 is based on the idea, which originates in the work of Ben-Or *et-al.* [BGG+88], of having the prover commit to its messages rather than sending them in the clear. Since the protocol is *public-coin* the verifier can continue the interaction even though it does

[27] An argument of proximity is similar to an IPP except that the soundness condition is further relaxed and required to hold only for polynomial-time cheating provers. See [KR15] for details and a formal definition.

[28] Our notion of *prover-oblivious queries* extends the notion of *proof-oblivious queries* studied by Gur and Rothblum [GR16] in the context of MAPs (i.e., non-interactive proofs of proximity).

not see the actual contents of the prover's messages. After all commitments have been sent, the verifier only needs to check that there exist suitable decommitments that would have made the underlying IPP verifier accept. Since the commitment hides the contents of the messages, it cannot do so by itself and we would like to use the prover. At this point, one could try to naively argue that the residual statement is an NP statement, and so we can invoke a general purpose zero-knowledge protocol for NP (e.g., the classical [GMW91] protocol or the more efficient [IKOS09] protocol).[29]

Herein arises the main difficulty with this approach. While the statement that the verifier needs to check at the end of the interaction does consist of an existential quantifier applied to a polynomial-time computable predicate, the latter predicate makes oracle access to the input x and so we do not know how to express it as an NP statement. To resolve this difficulty, we restrict our attention to verifiers that make *prover-oblivious queries*. Thus, our verifier can actually make its queries before and we can construct a NP statement that refers to the actual values that it reads from the input. At this point we can indeed invoke a general purpose zero-knowledge protocol for NP and conclude the proof.

Lastly, we remark that the specific flavor of soundness and zero-knowledge that we obtain depends on the commitment scheme we use. Specifically, instantiating the above approach with a computationally hiding and statistically binding commitment scheme yields a *computational* zero-knowledge proof of proximity, whereas a statistically hiding and computationally binding one yields a statistical zero-knowledge *argument* of proximity.

As noted above, we need the following result from [IKOS09]:

Lemma 6.4 ([IKOS09])**.** *Let $\mathcal{L} \in \mathsf{NP}$ with witness relation $R(\cdot,\cdot)$ that is computable in time t. If there exist one-way functions, then $\mathcal{L}$ has a computational zero-knowledge proof in which the verifier runs in time $\tilde{O}(t) \cdot \mathsf{poly}(k)$ and the prover runs in time $\mathsf{poly}(N,k)$. For every (malicious) verifier running in time T, the simulator runs in time $(T + \tilde{O}(t)) \cdot \mathsf{poly}(k)$. The number of rounds is $\mathsf{poly}(k)$.*

Actually, since the running times are not specified in [IKOS09], we give an overview of the construction in Appendix B.4.

We proceed to give a proof sketch of Theorem 6.2 and note that Theorem 6.3 is proved similarly (using statistically hiding commitments).[30]

Proof Sketch of Theorem 6.2. The existence of one-way functions implies the existence of the following cryptographic protocols that we will use:

- A computationally hiding and statistically binding commitment scheme [Nao91, HILL99]. Moreover, after one initial set-up message from the receiver to the sender (where this setup can be re-used for a polynomial number of commitments), the commitment scheme is non-interactive: the sender only needs to send a single message to the receiver. (This commitment scheme will be used to derive the first part of Theorem 6.2.)

[29] The verifier in the [IKOS09] protocol runs in time that is *linear* in the complexity t of the NP verification process. In contrast, the [GMW91] verifier runs in time $\mathsf{poly}(t)$. The distinction is important for us since in Corollaries 6.8 and 6.9 below, we will apply Theorem 6.2 on a statement that can be verified in roughly $\sqrt{n}$ time, and so we cannot afford a polynomial overhead.

[30] Note that statistically hiding commitments can be based on any one-way function [HNO+09]. For the furthermore part, we need to use *constant-round* statistically hiding commitments and *constant-round* statistical zero-knowledge arguments for NP. Both are known to exist assuming collision resistant hash functions [NY89, BCY91].

- Computational zero-knowledge proofs for any language in NP in which the verifier runs in time that is almost linear in the complexity of the witness relation (see Lemma 6.4).

Let (P, V) be an ℓ-round *public-coin* IPP for $\mathcal{L}$ with *prover oblivious queries*. We describe the construction of a *computational* zero-knowledge proof of proximity for $\mathcal{L}$. As alluded to above, the construction of a statistical zero-knowledge argument of proximity is similar, except that we replace the computationally hiding and statistically binding commitment with one that is statistically hiding and computationally binding, and replace the computatational zero-knowledge proof for NP with a statistical zero-knowledge argument.

We proceed to describe the computational ZKPP $(\mathsf{P}', \mathsf{V}')$ for $\mathcal{L}$, on input x of length N, security parameter k and proximity parameter ε. First, $(\mathsf{P}', \mathsf{V}')$ run the setup for the commitment scheme. After this initial step, the interaction consists of two main parts. In the first part, P' and V' emulate the interaction between P and V, where P' only *commits* to the messages that P would have sent. Since the protocol (P, V) is a *public-coin* protocol, the verifier V' can continue the interaction without actually knowing the contents of the messages that it receives (since V' only needs to sample and send random coin tosses).

Then, in the second part, V' has already obtained commitments $c_1, \ldots, c_\ell$ to some messages $\alpha_1, \ldots, \alpha_\ell$ that P would have sent. At this point we would like P' to prove the statement:

$$\exists d_1, \ldots, d_\ell, \alpha_1, \ldots, \alpha_\ell \text{ such that } \begin{array}{c} d_i \text{ is a decommitment of } c_i \text{ with respect to message } \alpha_i, \text{ for every } i \in [r] \\ \text{and} \\ \mathsf{V}^x(N, \varepsilon, k, (\alpha_1, \beta_1, \ldots, \alpha_\ell, \beta_\ell)) = 1. \end{array} \tag{14}$$

The statement in Equation (14) is almost, but not quite, an NP statement. The reason that we would like to phrase it as an NP statement is that by Lemma 6.4 (and using our assumption that there exist one-way functions), there exist very efficient (computational) zero knowledge proofs for any language in NP. Thus, we would like for P' to prove Equation (14) to V' using such a general purpose zero-knowledge proof-system.

The problem that we encounter is that Equation (14) is not precisely an NP statement since it refers to oracle access to a given string x.[31] To overcome this problem, we use our assumption that V makes *prover oblivious queries*. Hence, the queries that V makes depend only on its own random coin tosses (and answers to previous queries that it has made), but not on the messages sent by P. Denote by $Q(x; \rho)$ the sequence of (possibly adaptive) queries that V makes on input x and random string ρ. Since $Q(x, \rho)$ depends only on the randomness (and, possibly, on answers to previous queries to x), the verifier V' can sample ρ at random and generate this set. We can now re-state Equation (14) as:

$$\exists d_1, \ldots, d_\ell, \alpha_1, \ldots, \alpha_\ell \text{ such that } \begin{array}{c} d_i \text{ is a decommitment of } c_i \text{ with respect to message } \alpha_i, \text{ for every } i \in [r] \\ \text{and} \\ \mathsf{V}(x_{Q(x;\rho)}, (N, \varepsilon, k), (\alpha_1, \beta_1, \ldots, \alpha_\ell, \beta_\ell)) = 1, \end{array} \tag{15}$$

which is in fact an NP relation, for which P' has a witness. Therefore, using our assumption that one-way functions exist, there exists a computational zero-knowledge proof for Equation (15). P'

[31] As a matter of fact Equation (14) can be expressed as an "non-interactive proof of proximity" or MAP [GR16].

and V' engage in this proof-system and V' accepts or rejects accordingly. (To actually run this proof-system V' first shares ρ with P' - but it does so only after they have completed the emulation of (P, V).)

Completeness of $(\mathsf{P}', \mathsf{V}')$ follows from the completeness of (P, V) and the (perfect) completeness of the [IKOS09] zero-knowledge proof. The analysis of soundness and zero-knowledge is standard and we omit them from this preliminary version.

We proceed to analyze the efficiency of the proof-system. We consider the three phases of interactions separately:

- **Setup Phase:** First, the two parties set up the commitment scheme this step is done using a single round of communication and with complexity $\mathsf{poly}(k)$ for both parties.
- **Commitment Phase:** Each bit that P sends to V in the original protocol is emulated by a (non-interactive) commitment (with a $\mathsf{poly}(k)$ overhead). Messages sent from V to P are unchanged (recall that these refer to random coin tosses). Thus, the round complexity of this part is ℓ and there is a $\mathsf{poly}(k)$ overhead to the running time of both parties.
- **Final Phase:** V' first sends the random string used by the underlying V. This introduces a $t_{\mathsf{V}}(N, k, \varepsilon)$ overhead to both parties. Then, both parties run the [IKOS09] protocol on an NP statement that can be verified in time $t = t_{\mathsf{V}}(N, \varepsilon, k) \cdot \mathsf{poly}(k)$. The [IKOS09] verifier runs in time that is $O(t) = t_{\mathsf{V}}(N, \varepsilon, k) \cdot \mathsf{poly}(k)$ whereas the [IKOS09] prover runs in time $\mathsf{poly}(t) = \mathsf{poly}(t_{\mathsf{V}}(N, \varepsilon, k), k)$. The number of rounds is $\mathsf{poly}(k)$.

□

To obtain our ZKPP results, we will combine Theorem 6.2 with known results from the literature. Specifically, we will use the following results: (where throughout N denotes the input length, k the security parameter, and ε the security parameter).

Theorem 6.5 ([RVW13])**.** *Every language in logspace-uniform* NC*, has a* $\mathsf{polylog}(N)$*-round public-coin* ε*-*IPP*, for* $\varepsilon = N^{-1/2}$*, with perfect completeness and* $1/2$ *soundness error. The verifier runs in time* $N^{\frac{1}{2}+o(1)}$ *and the (honest) prover runs in time* $\mathsf{poly}(N)$*. Furthermore, the verifier makes prover oblivious queries.*

Theorem 6.6 ([RRR16])**.** *Let* $\mathcal{L}$ *be a language that is computable in* $\mathsf{poly}(N)$*-time and* $O(N^{\sigma})$*-space, for some sufficiently small constant* $\sigma > 0$*. Then* $\mathcal{L}$ *has a constant-round public-coin* ε*-*IPP *for* $\varepsilon = N^{-1/2}$*, with perfect completeness and* $1/2$ *soundness error. The verifier runs in time* $N^{1/2+O(\sigma)}$ *and the (honest) prover runs in time* $\mathsf{poly}(N)$*. Furthermore, the verifier makes prover oblivious queries.*

Theorem 6.7 ([Kil92, BGH+06, DR06])**.** *Assume that there exist collision-resistant hash functions. Then, every language in* NP *has a* 4*-message public-coin argument of* ε*-proximity with perfect completeness and* $1/2$ *soundness error (for any value of* $\varepsilon > 0$*). The verifier runs in time* $\mathsf{poly}(\log(N), k, 1/\varepsilon)$ *and the prover runs in time* $\mathsf{poly}(N, k)$*. Furthermore, the verifier makes prover oblivious queries.*

We remark that the fact that the verifier makes prover oblivious queries is not stated explicitly in the above works but can be verified by inspection.[32] Combining Theorems 6.5 and 6.6 with Theorem 6.2, and Theorem 6.7 with Theorem 6.3 we derive the following corollaries:

[32] Theorem 6.7 is obtained by applying Kilian's [Kil92] protocol to a PCP of proximity (c.f., [BGH+06, DR06]). See further discussions in [RVW13, KR15]. We remark that for the resulting argument of proximity to have proof oblivious queries, we need to use a PCP of proximity whose queries are non-adaptive in the proof. Such general purpose PCPs of proximity were constructed in [BGH+06, DR06].

Corollary 6.8 (Computational ZKPP for Bounded Depth)**.** *Assume that there exist one-way functions. Then, every language in logspace-uniform* NC*, has a* $(\mathsf{polylog}(N) + \mathsf{poly}(k))$*-round computational zero-knowledge proof of* ε*-proximity, for* $\varepsilon = n^{-1/2}$*. The verifier runs in time* $N^{\frac{1}{2}+o(1)} \cdot \mathsf{poly}(k)$ *and the (honest) prover runs in time* $\mathsf{poly}(N, k)$*. The simulation overhead is* $s(t_{\widehat{\mathsf{V}}}, N, k, \varepsilon) = t_{\widehat{\mathsf{V}}} \cdot \mathsf{poly}(k)$*, for (malicious) verifiers running in time* $t_{\widehat{\mathsf{V}}} = t_{\widehat{\mathsf{V}}}(N, k, \varepsilon)$*.*

Corollary 6.9 (Computational ZKPP for Bounded Space)**.** *Assume that there exist one-way functions. Let* $\mathcal{L}$ *be a language that is computable in* $\mathsf{poly}(N)$*-time and* $O(N^{\sigma})$*-space, for some sufficiently small constant* $\sigma > 0$*. Then,* $\mathcal{L}$ *has a* $\mathsf{poly}(k)$*-message computational zero-knowledge proof of* ε*-proximity, for* $\varepsilon = N^{-1/2}$*. The verifier runs in time* $N^{1/2+O(\sigma)} \cdot \mathsf{poly}(k)$ *and the (honest) prover runs in time* $\mathsf{poly}(N, k)$*. The simulation overhead is* $s(t_{\widehat{\mathsf{V}}}, N, k, \varepsilon) = t_{\widehat{\mathsf{V}}} \cdot \mathsf{poly}(k)$*, for (malicious) verifiers running in time* $t_{\widehat{\mathsf{V}}} = t_{\widehat{\mathsf{V}}}(N, k, \varepsilon)$*.*

Corollary 6.10 (Statistical Zero-Knowledge Arguments)**.** *Assume that there exist collision resistant hash functions. Then, every language in* NP*, has a constant-round statistical zero-knowledge argument of* ε*-proximity, for every value of* $\varepsilon > 0$*. The verifier runs in time* $\mathsf{poly}(\log(N), k, 1/\varepsilon)$ *and the (honest) prover runs in time* $\mathsf{poly}(N, k)$*.*

Acknowledgments

We thank Oded Goldreich and Omer Paneth for useful discussions.

The first and third author were supported in part by NSF Grants CNS-1350619 and CNS-1414119, Alfred P. Sloan Research Fellowship, Microsoft Faculty Fellowship, the NEC Corporation, a Steven and Renee Finn Career Development Chair from MIT. This work was also sponsored in part by the Defense Advanced Research Projects Agency (DARPA) and the U.S. Army Research Office under contracts W911NF-15-C-0226.

The second author was partially supported by NSF MACS - CNS-1413920, DARPA IBM - W911NF-15-C-0236 and SIMONS Investigator award Agreement Dated 6-5-12.

References

[ABG+14] Adi Akavia, Andrej Bogdanov, Siyao Guo, Akshay Kamath, and Alon Rosen. Candidate weak pseudorandom functions in $AC^0 \circ MOD_2$. In *Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014*, pages 251–260, 2014. 2

[AKNS00] Noga Alon, Michael Krivelevich, Ilan Newman, and Mario Szegedy. Regular languages are testable with a constant number of queries. *SIAM J. Comput.*, 30(6):1842–1862, 2000. 4

[BCF+16] Eli Ben-Sasson, Alessandro Chiesa, Michael A. Forbes, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner. On probabilistic checking in perfect zero knowledge. *IACR Cryptology ePrint Archive*, 2016:988, 2016. 1.2

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, pages 31–60, 2016. 8

[BCY91] Gilles Brassard, Claude Crépeau, and Moti Yung. Constant-round perfect zero-knowledge computationally convincing protocols. *Theor. Comput. Sci.*, 84(1):23–52, 1991. 30

[BGG+88] Michael Ben-Or, Oded Goldreich, Shafi Goldwasser, Johan Håstad, Joe Kilian, Silvio Micali, and Phillip Rogaway. Everything provable is provable in zero-knowledge. In *Advances in Cryptology - CRYPTO '88, 8th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1988, Proceedings*, pages 37–56, 1988. 1.1.2, 6, 6

[BGH+06] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Robust PCPs of proximity, shorter PCPs, and applications to coding. *SIAM J. Comput.*, 36(4):889–974, 2006. 1.1.2, 5.18, 6, 6.7, 32

[BM88] László Babai and Shlomo Moran. Arthur-Merlin games: A randomized proof system and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, pages 254–276, 1988. 5.5, 5.2.2

[BY96] Mihir Bellare and Moti Yung. Certifying permutations: Noninteractive zero-knowledge based on any trapdoor permutation. *J. Cryptology*, 9(3):149–166, 1996. 1, 1.1.1, 5, 4.1.1, 12

[CL17] Ran Canetti and Amit Lichtenberg, 2017. Unpublished manuscript. 1

[CS10] Artur Czumaj and Christian Sohler. Testing expansion in bounded-degree graphs. *Combinatorics, Probability & Computing*, 19(5-6):693–709, 2010. 1.1.1, 4.2, 4.2, 17, 4.2, 4.17

[CS16] Gil Cohen and Igor Shinkar. The complexity of DNF of parities. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 47–58, 2016. 2

[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. 2.2, 2.1.2, 2.6

[DR06] Irit Dinur and Omer Reingold. Assignment testers: Towards a combinatorial proof of the PCP theorem. *SIAM J. Comput.*, 36(4):975–1024, 2006. 1.1.2, 6, 6.7, 32

[EKR04] Funda Ergün, Ravi Kumar, and Ronitt Rubinfeld. Fast approximate probabilistically checkable proofs. *Inf. Comput.*, 189(2):135–159, 2004. 1

[FGL14] Eldar Fischer, Yonatan Goldhirsh, and Oded Lachish. Partial tests, universal tests and decomposability. In *Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014*, pages 483–500, 2014. 1, 4

[FLS99] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs under general assumptions. *SIAM Journal on Computing*, 1999. Preliminary version in *FOCS'90*. 1

[GG16] Oded Goldreich and Tom Gur. Universal locally testable codes. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:42, 2016. 1

[GGK15] Oded Goldreich, Tom Gur, and Ilan Komargodski. Strong locally testable codes with relaxed local decoders. In *30th Conference on Computational Complexity, CCC 2015, June 17-19, 2015, Portland, Oregon, USA*, pages 1–41, 2015. 5.18

[GGR98] Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, 1998. 1, 1.1.1, 4.2

[GGR15] Oded Goldreich, Tom Gur, and Ron D. Rothblum. Proofs of proximity for context-free languages and read-once branching programs - (extended abstract). In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, pages 666–677, 2015. 1, 4.2

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, pages 186–208, 1989. Preliminary version in *STOC'85*. 1

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987. B.4, 1

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, pages 691–729, 1991. Preliminary version in *FOCS'86*. 1, 6, 29

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, 2001. 1, 11

[Gol16] Oded Goldreich. *Introduction to Property Testing*. forthcoming (`http://www.wisdom.weizmann.ac.il/~oded/pt-intro.html`), 2016. 1

[GR99] Oded Goldreich and Dana Ron. A sublinear bipartiteness tester for bounded degree graphs. *Combinatorica*, 19(3):335–373, 1999. 4.3

[GR02] Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002. 1.1.1, 1.1.1, 4.2, 4.3

[GR11] Oded Goldreich and Dana Ron. On testing expansion in bounded-degree graphs. In Oded Goldreich, editor, *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation - In Collaboration with Lidor Avigad, Mihir Bellare, Zvika Brakerski, Shafi Goldwasser, Shai Halevi, Tali Kaufman, Leonid Levin, Noam Nisan, Dana Ron, Madhu Sudan, Luca Trevisan, Salil Vadhan, Avi Wigderson, David Zuckerman*, volume 6650 of *Lecture Notes in Computer Science*, pages 68–75. Springer, 2011. 1.1.1, 4.2

[GR13] Oded Goldreich and Ron D. Rothblum. Enhancements of trapdoor permutations. *J. Cryptology*, 26(3):484–512, 2013. 1

[GR15] Tom Gur and Ron D. Rothblum, 2015. Unpublished observation. 1.1.1, 4.1

[GR16] Tom Gur and Ron D. Rothblum. Non-interactive proofs of proximity. *Computational Complexity*, pages 1–109, 2016. 1, 4.1, 2, 5.2, 5.2.1, 5.18, 28, 31

[GR17] Tom Gur and Ron D. Rothblum. A hierarchy theorem for interactive proofs of proximity. In *Proceedings of the 2017 ACM Conference on Innovations in Theoretical Computer Science, Berkeley, CA, USA, January 9-11, 2016*, 2017. 1, 5.1, 5.2, 5.2.2

[GS89] Shafi Goldwasser and Michael Sipser. Private coins versus public coins in interactive proof systems. *Advances in Computing Research: Randomness and Computation*, pages 73–90, 1989. 5.2.2

[GS92] Peter Gemmell and Madhu Sudan. Highly resilient correctors for polynomials. *Inf. Process. Lett.*, 43(4):169–174, 1992. B.3

[GS06] Oded Goldreich and Madhu Sudan. Locally testable codes and pcps of almost-linear length. *J. ACM*, 53(4):558–655, 2006. 5.9

[GSV98] Oded Goldreich, Amit Sahai, and Salil P. Vadhan. Honest-verifier statistical zero-knowledge equals general statistical zero-knowledge. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 399–408, 1998. 7

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999. 6, 36

[HNO+09] Iftach Haitner, Minh Nguyen, Shien Jin Ong, Omer Reingold, and Salil Vadhan. Statistically hiding commitments and statistical zero-knowledge arguments from any one-way function. *SIAM Journal on Computing*, pages 1153–1218, 2009. Preliminary versions in *FOCS '06* and *STOC '07*. 30

[IKOS09] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge proofs from secure multiparty computation. *SIAM J. Comput.*, 39(3):1121–1152, 2009. 6, 6.4, 6, 29, 6, B.4, 35, B.4

[IW14] Yuval Ishai and Mor Weiss. Probabilistically checkable proofs of proximity with zero-knowledge. In *Theory of Cryptography - 11th Theory of Cryptography Conference, TCC 2014, San Diego, CA, USA, February 24-26, 2014. Proceedings*, pages 121–145, 2014. 1.2

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC)*, pages 723–732, 1992. 1.1.2, 6, 6.7, 32

[KR15] Yael Tauman Kalai and Ron D. Rothblum. Arguments of proximity - [extended abstract]. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, pages 422–442, 2015. 1, 27, 32

[KS11] Satyen Kale and C. Seshadhri. An expansion tester for bounded degree graphs. *SIAM J. Comput.*, 40(3):709–720, 2011. 1.1.1, 4.2

[KT00] Jonathan Katz and Luca Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 80–86, 2000. 5.10

[Nao91] Moni Naor. Bit commitment using pseudorandomness. *J. Cryptology*, 4(2):151–158, 1991. 6, 36

[NS10] Asaf Nachmias and Asaf Shapira. Testing the expansion of a graph. *Inf. Comput.*, 208(4):309–314, 2010. 1.1.1, 4.2, 4.16

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC)*, pages 33–43, 1989. 30

[RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 49–62, 2016. 1, 1.1.2, 8, 6, 6.6

[RS96] Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM J. Comput.*, 25(2):252–271, 1996. 1, B.3

[RVW13] Guy N. Rothblum, Salil P. Vadhan, and Avi Wigderson. Interactive proofs of proximity: delegating computation in sublinear time. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 793–802, 2013. 1, 1.1.1, 1.1.2, 4.3, 4.18, 5.5, 5.2, 5.2.2, 5.17, 6, 6.5, 32

[Sud95] Madhu Sudan. *Efficient Checking of Polynomials and Proofs anf the Hardness of Approximation Problems*, volume 1001 of *Lecture Notes in Computer Science*. Springer, 1995. B.3

[Vad99] Salil P. Vadhan. *A Study of Statistical Zero-Knowledge Proofs*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1999. 7, 2.2, 2.2.2, 2.2.2, 3, 3, 3.6, 3.7, 4.1.1, 5.1, A, A

[Vad12] Salil P. Vadhan. *Pseudorandomness*. Now Publishers Inc., Hanover, MA, USA, 2012. 2.5

A Reducing HV-SZKPP to Entropy Difference

In this section we show how to reduce any property with honest-verifier zero-knowledge to an instance of *Entropy Difference*.

Lemma A.1. *Suppose that a property* Π *has a honest-verifier statistical zero-knowledge* ε-IPP *such that for every input length* N *and security parameter* $k \in \mathbb{N}$ *the simulator's expected running time is bounded by* $t_{\mathsf{S}}(\varepsilon, N, k) = t'_{\mathsf{S}}(\varepsilon, N) \cdot \mathsf{poly}(k)$ *and for every* ε *the function* $t'_{\mathsf{S}}(\varepsilon, \cdot)$ *is monotone non-decreasing.*

Then, there is a reduction from Π *to* ED. *Specifically, the reduction is given* ε *and an input length* N *and outputs two oracle aided circuits* $C_0, C_1 \colon \{0,1\}^m \to \{0,1\}^n$ *such that the following holds.*

1. *(C_0, C_1) is an instance of* ED*:*

$$f \in \Pi \implies \mathrm{H}(C_0^f) \geq \mathrm{H}(C_1^f) + 1,$$
$$f \text{ is } \varepsilon\text{-far from } \Pi \implies \mathrm{H}(C_1^f) \geq \mathrm{H}(C_0^f) + 1.$$

2. *The reduction's running time is* $\mathsf{poly}(t_{\mathsf{S}}(\varepsilon, N, \mathsf{poly}(t'_{\mathsf{S}}(\varepsilon, N))))$.

Note that the last item implies that for every $x \in \{0,1\}^m$ and $b \in \{0,1\}$, computing $C_b(x)$ requires only $\mathsf{poly}(t_{\mathsf{S}}(\varepsilon, N, \mathsf{poly}(t'_{\mathsf{S}}(\varepsilon, N))))$ many oracle calls. The proof of the above lemma follows from the proof that entropy difference is SZK-hard [Vad99]. We only give sufficient details to demonstrate how to apply that proof to our setting.

Proof sketch. Assume that (P, V) is the ε-IPP for Π and let S be the honest-verifier simulator for (P, V) whose simulation deviation is $\mu(k) = \mathsf{negl}(k)$. We assume for simplicity that $t_{\mathsf{S}}(N, k, \varepsilon)$ is a strict bound (and not only expected) on the running of S. The proof can be extended to handle expected bounds as well (in fact, the proof in [Vad99] handles even weaker simulators). Assume without loss of generality that P and V send their messages in turns, P sends the odd messages and V the even ones. Let $v(\varepsilon, N, k)$ be a bound on the number of messages sent by V to P for every f. In addition, let $c(\varepsilon, N, k)$ and $r(\varepsilon, N, k)$ be bounds on the total communication (measures in bits) between P and V and the number of random bits accessed by the verifier, respectively. We now modify the proof system so that V sends its random coins to P in an additional message just before the end of the protocol. The total communication and number of messages sent from V to P now increases to $c'(\varepsilon, N, k) = c(\varepsilon, N, k) + r(\varepsilon, N, k)$ and $v'(\varepsilon, N, k) = v(\varepsilon, N, k) + 1$, respectively. S is modified to simulate the additional last message as well, without increasing its simulation deviation (this is possible since S was supposed to simulate V's random coins anyway).

Fix ε and N and let $k' \in \mathbb{N}$ such that $\mu(k') \leq \min\{1/v'(\varepsilon, N, k') \cdot c'(\varepsilon, N, k'), 1/4 - 2^{-40}\}$ and the completeness and soundness errors of (P, V) are at most 2^{-40}. Note that it suffices to take $k' = \mathsf{poly}(t'_{\mathsf{S}}(\varepsilon, N))$ (i.e., a fixed polynomial for all ε and N): It holds that $v'(\varepsilon, N, k') \cdot c'(\varepsilon, N, k') \leq t^2_{\mathsf{S}}(\varepsilon, N, k') = t'^2_{\mathsf{S}}(\varepsilon, N) \cdot \mathsf{poly}(k')$. Thus, we can take k' such that $\mu'(k') \leq 1/t'^2_{\mathsf{S}}(\varepsilon, N)$, for some negligible function $\mu'(k') = \mu(k') \cdot \mathsf{poly}(k')$. Since $t'_{\mathsf{S}}(\varepsilon, N)$ is monotone non-decreasing in N, taking $k' = \mathsf{poly}(t'_{\mathsf{S}}(\varepsilon, N))$ guarantee the required condition for large enough (depending on μ) N (for simplicity, we ignore shorter inputs that can be solved via brute-force by the verifier).

Finally, let S_i^f be the random variable distributed according to the first i messages in the output of a random execution of $\mathsf{S}^f(\varepsilon, N, k')$. In the following we remove ε, N and k' from the notation.

Constructing C_0 and C_1. Define $X = S_2^f \otimes S_4^f \otimes \cdots \otimes S_{2v'}^f$.[33] Similarly, define Y_1 to be $Y_1 = S_1^f \otimes S_3^f \otimes \cdots \otimes S_{2v'-1}^f$ and define Y_2 to be the uniform distribution on $r - 7$ bits. Furthermore, define Y_3 as follows: run S^f $8\ln(c'v' + 2)$ times independently; if the verifier rejects in the majority of the transcripts obtained, output $c'v' + 2$ random bits; otherwise, output the empty string. Define $Y = Y_1 \otimes Y_2 \otimes Y_3$. Finally, the circuits C_0 and C_1 take as input random coins to sample and output $x \leftarrow X$ and $y \leftarrow Y$, respectively. Since we require that the input (resp., output) lengths of C_0 and C_1 will be equal, we pad the shorter input (resp., output) with redundant random coins (resp., zeros).[34]

[33] Recall that $P \otimes Q$ stands for the product distribution of P and Q.

[34] Let m_X and n_Y denote the input and output lengths of X, respectively. Let m_Y, n_Y be similarly defined. For example, if $m_X < m_Y$ and $n_X < n_Y$ we can modify X as follows: sample $x \leftarrow X$ using part of the given m_Y random

Analysis. That (C_0, C_1) is an instance of Entropy Difference (Item 1) follows from [Vad99, Claims 3.3.14 and 3.3.15]. The reduction's running time (Item 2) follows from the constructions of C_0 and C_1. □

B Missing Proofs

B.1 Proving Lemma 5.3

The proof of Lemma 5.3, sketch of which is given below, immediately follows from Lemmas 2.15 and A.1.

Proof sketch of Lemma 5.3. Both the verifier and the prover will run the reduction form Lemma A.1 to get two distributions encoded by oracle-aided circuits (C_0, C_1). They will then run the protocol from Lemma 2.15 with respect to these distributions. Since the latter is a sample-access protocol the verifier can indeed run it using only oracle access to its input f.

The running time of the reduction implies that the input and outputs sizes of C_0 and C_1 are $\mathsf{poly}(t_{\mathsf{S}}(\varepsilon, N, \mathsf{poly}(t'_{\mathsf{S}}(\varepsilon, N))))$. By Lemma 2.15 the running time of the verifier is thus $\mathsf{poly}(t_{\mathsf{S}}(\varepsilon, N, \mathsf{poly}(t'_{\mathsf{S}}(\varepsilon, N))), k)$, as required.

Zero-knowledge follows from similar arguments to the ones made above. □

B.2 Proving Claim 5.16

The proof of Claim 5.16 follows similar lines to that of Lemma 5.3.

Proof sketch of Claim 5.16. Both the verifier and the prover will run the reduction form Lemma A.1 with respect to the property CI and proximity parameter $\delta(C)/2$ to get two distributions encoded by oracle-aided circuits (C_0, C_1). If $w \in \mathsf{CD}_{\mathsf{YES},\ell}$ then w is $\delta(C)/2$-far from CI and thus $\mathrm{H}(C_1^w) \geq \mathrm{H}(C_0^w) + 1$. However, if $w \in \mathsf{CD}_{\mathsf{NO},\ell}$ then $w \in \mathsf{CI}$ and thus $\mathrm{H}(C_0^w) \geq \mathrm{H}(C_1^w) + 1$.

The verifier and the prover will then run the protocol from Lemma 2.15 with respect to the instance (C_1^w, C_0^w) (note that the order of the circuits has changed) and security parameter k chosen such that the completeness and soundness errors are both $1/3$ for large enough ℓ. Since the latter is a sample-access protocol the verifier can indeed run it using only oracle access to its input w.

Recall that we assumed that $\mathsf{CI} \in \mathsf{HV\text{-}ESZKPP}\left[\mathsf{poly}(\log(N), k, 1/\varepsilon)\right]$. Hence, the simulator for CI runs in time $\mathsf{poly}(\log(\ell), k, 1/\varepsilon)$, which by the choice of parameters is simply $\mathsf{poly}(\log \ell)$ (recall the the proximity parameter $\delta(C)$ and the security parameter k are constant). The running time of the reduction implies that the input (i.e., the number of bits need to sample from the distribution) and outputs sizes of C_0 and C_1 are $\mathsf{poly}(\log \ell)$ as well, and by Lemma 2.15 the running time of the verifier is thus $\mathsf{poly}(\log(\ell))$, as required. □

B.3 Proof Sketch of Lemma 5.11

We start with a low degree extension code, over a finite field $\mathbb{F}$, which view messages $x \in \mathbb{F}^\ell$ as functions $x : H^m \to \mathbb{F}$, where $H \subseteq \mathbb{F}$ is a subset and m is a dimension, such that $|H^m| = \ell$. The code maps x to its low degree extension: namely, the unique individual degree $|H| - 1$ polynomial that agrees with x on H^m. By the Shwartz-Zippel lemma this code has relative distance $1 - \frac{(|H|-1)\cdot m}{|\mathbb{F}|}$.

bits and output $x0^{n_y - n_x}$.

Furthermore, this code is known to be locally testable [RS96] and decodable [GS92, Sud95] using $O(|H| \cdot m)$ queries. We set $|H| = (\log(\ell))^c$, $m = \frac{\log(\ell)}{c \cdot \log\log(\ell)}$ and $|\mathbb{F}| = O(m \cdot |H|)$ for a sufficiently large constant $c \geq 1$. Furthermore, we use a field $\mathbb{F}$ which is an extension field of the binary field $\mathbb{F}_2$.

We then concatenate the above low degree extension code with a good binary linear code. The overall resulting code has message length $k(\ell) = |H|^m \cdot \log(F) = \tilde{O}(\ell)$, blocklength $n(\ell) = O(\mathbb{F}^m \cdot \mathbb{F}) = \tilde{O}(\ell^{1+1/c})$, constant relative distance and locally testable and decodable with $O(|H| \cdot m) = \mathsf{polylog}(\ell)$ queries, which meets our desired parameters by setting c to be sufficiently large. Furthermore, since the low degree extension is *linear* over the large field $\mathbb{F}$, which is an extension field of $\mathbb{F}_2$, it is also linear over $\mathbb{F}_2$ and therefore the resulting code is also $\mathbb{F}_2$-linear.

B.4 Proof Sketch of Lemma 6.4

We use the [IKOS09] "MPC in the head" construction. More specifically, we will use their simplest variant, which is based on the [GMW87] 3-party protocol, in the OT-hybrid, with *semi-honest* security against 2 (semi-honest) players.[35] Below we refer to this as the IKOS protocol.

We first recall that the [GMW87] protocol, with 2-out-of-3 semi-honest can be implemented so that the parties, and the (semi-honest) simulator, run in time $O(t')$ (where we count OT calls at unit cost), where t' is the circuit complexity of the function.

The IKOS protocol works in k sequential phases (in order to obtain 2^{-k} soundness), where each phase works as follows. The prover first runs the [GMW87] protocol with respect to to the function $f(x, w_1, w_2, w_3) = R(x, w_1 \oplus w_2 \oplus w_3)$, where w_1, w_2, w_3 are an additive secret sharing of the witness W. Observe that f is computable by a size $t' = tildeO(t)$ circuit, where t is the complexity of R of the NP relation (an extra log factor comes from emulating Turing machines by circuits). Thus, the parties and the MPC simulator run in time $\tilde{O}(t)$.

After running the MPC protocol (in its "head"), the IKOS prover commits to the view of all the players.[36] Then, the verifier chooses two (distinct) players $i, j \in \{1, 2, 3\}$ at random, and sends i and j to the prover. The prover decommits to these players views. The verifier rejects if the decommitments are invalid, the views are inconsistent, or if the result of the computation is not 1. Otherwise it accepts.

For the analysis of soundness and zero-knowledge of the IKOS protocol see [IKOS09]. Here we focus on the running times of the verifier and the simulator.

Observe that all that the verifier's running time in each phase is $\tilde{O}(t) * \mathsf{poly}(secp)$ as required.

We proceed to describe the IKOS simulator. Fix a malicious verifier $\widehat{\mathsf{V}}$. The simulator also runs for k phases. In each phase it repeats the following procedure at most $\mathsf{poly}(secp)$ times (and aborts if all fail):

1. Select at random a pair of (distinct) players $i, j \in \{1, 2, 3\}$, and runs the [GMW87] simulation on them (with respect to random strings w_i and w_j).

2. The simulator generates commitments to the simulated views for these players as well as a fake simulation (e.g., all zero) for the third player. The simulator "sends" these commitments to the verifier $\widehat{\mathsf{V}}$.

[35] Indeed, note that the [IKOS09] approach transforms *semi-honest* secure MPC protocols into proof-systems that are zero-knowledge with respect to *malicious* verifiers.

[36] Here we use a satistically binding commitment scheme, which follows from the existence of one-way functions [HILL99, Nao91].

3. $\widehat{\mathsf{V}}$ responds with a pair of distinct indices $i', j' \in \{1, 2, 3\}$ (otherwise, since the IKOS prover would abort, the simulator can output its generated view so far followed by a $\perp$ symbol).

4. If i', j' are not the same as i, j, then continue the loop.

5. Otherwise, the simulator can send decommitments to $\widehat{\mathsf{V}}$ and they continue to the next phase.

Overall the simulation of a single phase takes $(\tilde{O}(t) + T) \cdot \mathsf{poly}(k)$ time, where T is the running time of $\widehat{\mathsf{V}}$. See [IKOS09] for additional details.