Electronic Colloquium on Computational Complexity, Revision 3 of Report No. 138 (2022)

Robustness for Space-Bounded Statistical Zero Knowledge*

Eric Allender
Rutgers University, NJ, USA

Jacob Gray
University of Toronto, Canada

Saachi Mutreja
Columbia University, NY, USA

Harsha Tirumala
Rutgers University, NJ, USA

Pengxiang Wang
EPFL, Swiss Federal Institute of Technology, Lausanne, Switzerland

Abstract

We show that the space-bounded Statistical Zero Knowledge classes $\mathsf{SZK_L}$ and $\mathsf{NISZK_L}$ are surprisingly robust, in that the power of the verifier and simulator can be strengthened or weakened without affecting the resulting class. Coupled with other recent characterizations of these classes [5], this can be viewed as lending support to the conjecture that these classes may coincide with the non-space-bounded classes SZK and NISZK, respectively.

2012 ACM Subject Classification Complexity Classes

Keywords and phrases Interactive Proofs

Funding *Eric Allender*: Supported in part by NSF Grants CCF-1909216 and CCF-1909683.
Jacob Gray: Supported in part by NSF grants CNS-215018 and CCF-1852215
Saachi Mutreja: Supported in part by NSF grants CNS-215018 and CCF-1852215
Harsha Tirumala: Supported in part by NSF Grants CCF-1909216 and CCF-1909683.
Pengxiang Wang: Supported in part by NSF grants CNS-215018 and CCF-1852215

* An abbreviated version of this work, with some proofs omitted, appeared previously as [3].

ISSN 1433-8092

1 Introduction

The complexity class SZK (Statistical Zero Knowledge) and its "non-interactive" subclass NISZK have been studied intensively by the research communities in cryptography and computational complexity theory. In [15], a space-bounded version of SZK, denoted $\mathsf{SZK_L}$ was introduced, primarily as a tool for understanding the complexity of estimating the entropy of distributions represented by very simple computational models (such as low-degree polynomials, and NC^0 circuits). There, it was shown that $\mathsf{SZK_L}$ contains many important problems previously known to lie in SZK, such as Graph Isomorphism, Discrete Log, and Decisional Diffie-Hellman. The corresponding "non-interactive" subclass of $\mathsf{SZK_L}$, denoted $\mathsf{NISZK_L}$, was subsequently introduced in [2], primarily as a tool for clarifying the complexity of computing time-bounded Kolmogorov complexity under very restrictive reducibilities (such as projections). Just as every problem in SZK $\leq_{tt}^{\mathsf{AC}^0}$ reduces to problems in NISZK [17], so also every problem in $\mathsf{SZK_L} \leq_{tt}^{\mathsf{AC}^0}$ reduces to problems in $\mathsf{NISZK_L}$, and thus $\mathsf{NISZK_L}$ contains intractable problems if and only if $\mathsf{SZK_L}$ does.

Very recently, all of these classes were given surprising new characterizations, in terms of efficient reducibility to the Kolmogorov random strings. Let $\widetilde{R}_K$ be the (undecidable) promise problem $(Y_{\widetilde{R}_K}, N_{\widetilde{R}_K})$ where $Y_{\widetilde{R}_K}$ contains all strings y such that $K(y) \geq |y|/2$ and the NO instances $N_{\widetilde{R}_K}$ consists of those strings y where $K(y) \leq |y|/2 - e(|y|)$ for some approximation error term $e(n)$, where $e(n) = \omega(\log n)$ and $e(n) = n^{o(1)}$.

▶ **Theorem 1.** *[5] Let A be a decidable promise problem. Then*
- *$A \in$ NISZK if and only if A is reducible to $\widetilde{R}_K$ by randomized polynomial time reductions.*
- *$A \in \mathsf{NISZK}_L$ if and only if A is reducible to $\widetilde{R}_K$ by randomized AC^0 or logspace reductions.*
- *$A \in$ SZK if and only if A is reducible to $\widetilde{R}_K$ by randomized polynomial time "Boolean formula" reductions.*
- *$A \in \mathsf{SZK}_L$ if and only if A is reducible to $\widetilde{R}_K$ by randomized logspace "Boolean formula" reductions.*

In all cases, the randomized reductions are restricted to be "honest", so that on inputs of length n all queries are of length $\geq n^\epsilon$.

There are very few natural examples of computational problems A where the class of problems reducible to A via polynomial-time reductions differs (or is conjectured to differ) from the class or problems reducible to A via AC^0 reductions. For example the natural complete problems for NISZK under $\leq_m^{\mathsf{P}}$ reductions remain complete under AC^0 reductions. Thus Theorem 1 gives rise to speculation that NISZK and $\mathsf{NISZK_L}$ might be equal. (This would also imply that SZK $= \mathsf{SZK_L}$.)

This motivates a closer examination of $\mathsf{SZK_L}$ and $\mathsf{NISZK_L}$, to answer questions that have not been addressed by earlier work on these classes.

Our main results are:

1. **The verifier and simulator may be very weak.** $\mathsf{NISZK_L}$ and $\mathsf{SZK_L}$ are defined in terms of three algorithms: (1) A logspace-bounded *verifier*, who interacts with (2) a computationally-unbounded *prover*, following the usual rules of an interactive proof, and (3) a logspace-bounded *simulator*, who ensures the zero-knowledge aspects of the protocol. (More formal definitions are to be found in Section 2.) We show that the verifier and simulator can be restricted to lie in AC^0. Let us explain why this is surprising.
   The proof presented in [2], showing that $\mathsf{EA_{NC^0}}$ is complete for $\mathsf{NISZK_L}$, makes it clear that the verifier and simulator can be restricted to lie in $\mathsf{AC}^0[\oplus]$ (as was observed in [27]). But the proof in [2] (and a similar argument in [17]) relies heavily on hashing, and it is

known that, although there are families of universal hash functions in $\mathsf{AC}^0[\oplus]$, no such families lie in AC^0 [22]. We provide an alternative construction, which avoids hashing, and allows the verifier and simulator to be very weak indeed.

2. **The verifier and simulator may be somewhat stronger.** The proof presented in [2], showing that $\mathsf{EA}_{\mathsf{NC}^0}$ is complete for $\mathsf{NISZK}_\mathsf{L}$, also makes it clear that the verifier and simulator can be as powerful as $\oplus\mathsf{L}$, without leaving $\mathsf{NISZK}_\mathsf{L}$. This is because the proof relies on the fact that logspace computation lies in the complexity class PREN of functions that have *perfect randomized encodings* [9], and $\oplus\mathsf{L}$ also lies in PREN. Applebaum, Ishai, and Kushilevitz defined PREN and the somewhat larger class SREN (for *statistical randomized encodings*), in proving that there are one-way functions in SREN if and only if there are one-way functions in NC^0. They also showed that other important classes of functions, such as NL and GapL, are contained in SREN.[1] We initially suspected that $\mathsf{NISZK}_\mathsf{L}$ could be characterized using verifiers and simulators computable in GapL (or even in the slightly larger class DET, consisting of problems that are $\leq_T^{\mathsf{NC}^1}$ reducible to GapL), since DET is known to be contained in $\mathsf{NISZK}_\mathsf{L}$ [2].[2] However, we were unable to reach that goal.
   We were, however, able to show that the simulator and verifier can be as powerful as NL, without making use of the properties of SREN. In fact, we go further in that direction. We define the class PM, consisting of those problems that are $\leq_T^{\mathsf{L}}$-reducible to the Perfect Matching problem. PM contains NL [21], and is not known to lie in (uniform) NC (and it is not known to be contained in SREN). We show that statistical zero knowledge protocols defined using simulators and verifiers that are computable in PM yield only problems in $\mathsf{NISZK}_\mathsf{L}$.
3. **The complexity of the simulator is key.** As part of our attempt to characterize $\mathsf{NISZK}_\mathsf{L}$ using simulators and verifiers computable in DET, we considered varying the complexity of the simulator and the verifier separately. Among other things, we show that the verifier can be as complex as DET if the simulator is logspace-computable. In most cases of interest, the NISZK class defined with verifier and simulator lying in some complexity class remains unchanged if the rules are changed so that the verifier is significantly stronger or weaker.

We also establish some additional closure properties of $\mathsf{NISZK}_\mathsf{L}$ and SZK_L, some of which are required for the characterizations given in [5]. The rest of the paper is organized as follows;

In Section 3, we show how $\mathsf{NISZK}_\mathsf{L}$ can be defined equivalently using an AC^0 verifier and simulator. Formally, we prove that $\mathsf{NISZK}_\mathsf{L} = \mathsf{NISZK}_{\mathsf{AC}^0}$. Our proof involves defining a modification of the complete problem for $\mathsf{NISZK}_\mathsf{L}$, which remains complete for the class under a suitably weak form of reduction. The proof that this problem is in $\mathsf{NISZK}_\mathsf{L}$ involves hashing with a logspace verifier, which we cannot perform in AC^0. To get around this problem, we use a randomized encoding of a logspace machine computing this hashing. The randomized encoding is both computable by an AC^0 verifier and preserves several important properties of the original post-hashing distribution, which allows the modified complete problem to be solved in $\mathsf{NISZK}_{\mathsf{AC}^0}$ and establish the stated result.

Section 4 involves showing that increasing the power of the verifier and simulator to lie in PM does not increase the size of $\mathsf{NISZK}_\mathsf{L}$ (where PM is the class of problems (containing NL) that are logspace Turing reducible to Perfect Matching). We show that $\mathsf{NISZK}_\mathsf{L} = \mathsf{NISZK}_{\mathsf{PM}}$

[1] This is not stated explicitly for GapL, but it follows from [20, Theorem 1]. See also [13, Section 4.2].

[2] More precisely, as observed in [4], the Rigid Graph (non-) Isomorphism problem is hard for DET [29], and the Rigid Graph Non-Isomorphism problem is in $\mathsf{NISZK}_\mathsf{L}$ [2, Corollary 23].

in two steps: first, we begin by showing that $\mathsf{NISZK_L} = \mathsf{NISZK_{\oplus L}}$, using that problems in $\oplus\mathsf{L}$ have easily computable (AC^0) randomized encodings that retain some important statistical properties of the original distribution. The second step is to prove that $\mathsf{NISZK_{PM}} \subseteq \mathsf{NISZK_{\oplus L}}$. To do this, we utilize ideas from [8] to show how strings chosen uniformly at random can help in reducing instances of problems in PM to instances of a language in $\oplus\mathsf{L}$. This allows us to prove that $\mathsf{NISZK_{PM}} \subseteq \mathsf{NISZK_{\oplus L}}$ and completes the proof.

Section 5 expands the list of problems known to lie in $\mathsf{NISZK_L}$. McKenzie and Cook [23] studied different formulations of the problem of solving linear congruences. These problems are not known to lie in DET, which is the largest well-studied subclass of P known to be contained in $\mathsf{NISZK_L}$. However, these problems are randomly logspace-reducible to DET [10]. We show that $\mathsf{NISZK_L}$ is closed under randomized logspace reductions, and hence show that these problems also reside in $\mathsf{NISZK_L}$.

Section 6 shows that the complexity of the simulator is more important than the complexity of the verifier in non-interactive zero-knowledge protocols. In particular, the verifier can be as powerful as DET, while still defining only problems in $\mathsf{NISZK_L}$. In general, we show that if classes A, B satisfy $A \subseteq B \subseteq \mathsf{NISZK}_A$, then the verifier of the class NISZK_A can be boosted to class B without increasing the power of the class. Since the proof system can compute what the stronger B verifier can compute, the idea is to use the proof system as a replacement for the stronger verifier. We then obtain some concrete equalities by substituting in different choices of A and B.

Finally, Section 7 will show that $\mathsf{SZK_L}$ is closed under logspace Boolean formula truth-table reductions. The proof is an adaptation of [28] and primarily involves making circuit constructions into branching program constructions while also ensuring that they can be computed in logspace as opposed to polynomial time. The complete problem for $\mathsf{SZK_L}$ is to compute the statistical distance of a pair of branching programs, so the proof details how to combine pairs of branching programs to compute the "AND" or "OR" of pairs of branching programs.Using these constructions, given a desired Boolean formula, a final pair of branching programs can be created which are statistically distant iff the statistical distance of each of the original pairs satisfies the formula. Since this can be done in logspace, this establishes that the closure property holds.

2 Preliminaries

We assume familiarity with the basic complexity classes $\mathsf{L}, \mathsf{NL}, \oplus\mathsf{L}$ and P, and the circuit complexity classes NC^0 and AC^0. We assume knowledge of m-reducibility (many-one-reducibility) and Turing-reducibility. We also will need to refer to *projection* reducibility ($\leq_m^{\mathsf{proj}}$). A projection is a function f that is computed by a circuit that has no gates (other than NOT gates). Thus each output gate is either a constant, or it is connected via a wire to an input bit or a negated input bit. The $\leq_m^{\mathsf{proj}}$ reductions that we consider in this paper are all special cases of uniform AC^0 reductions. $\#\mathsf{L}$ is the class of functions that count the number of accepting paths of NL machines, and $\mathsf{GapL} = \{f - g : f, g \in \#\mathsf{L}\}$. The determinant is complete for GapL under $\leq_m^{\mathsf{AC}^0}$ reductions[3], and the complexity class DET is the class of languages NC^1-Turing reducible to functions in GapL.[4]

[3] See, for instance [7, Theorem 1] for a discussion of the history of this result.
[4] It is an interesting question, whether one needs to consider NC^1-Turing reductions in order to define the class DET. We refer the reader to [1, Open Question 6] for a discussion of this point.

We use the notation $q \sim S$ to denote that element q is chosen uniformly at random from the finite set S.

Many of the problems we consider deal with entropy (also known as Shannon entropy). The *entropy* of a distribution X (denoted $H(X)$) is the expected value of $\log(1/\Pr[X = x])$. Given two distributions X and Y, the *statistical difference* between the two is denoted $\Delta(X, Y)$ and is equal to $\sum_\alpha \left|\Pr[X = \alpha] - \Pr[Y = \alpha]\right|/2$. Equivalently, for finite domains D, $\Delta(X, Y) = \max_{S \subseteq D}\{\left|\Pr_X[S] - \Pr_Y[S]\right|\}$. This quantity is also known as the *total variation distance* between X and Y. The *support* of X, denoted $\text{supp}(X)$, is $\{x : \Pr[X = x] > 0\}$.

▶ **Definition 2.** *Promise Problem: a promise problem Π is a pair of disjoint sets (Π_Y, Π_N) (the "YES" and "NO" instances, respectively). A* solution *for Π is any set S such that $\Pi_Y \subseteq S$, and $S \cap \Pi_N = \emptyset$.*

▶ **Definition 3.** *A* branching program *is a directed acyclic graph with a single source and two sinks labeled 1 and 0, respectively. Each non-sink node in the graph is labeled with a variable in $\{x_1, \ldots, x_n\}$ and has two edges leading out of it: one labeled 1 and one labeled 0. A branching program computes a Boolean function f on input $x = x_1 \ldots x_n$ by first placing a pebble on the source node. At any time when the pebble is on a node v labeled x_i, the pebble is moved to the (unique) vertex u that is reached by the edge labeled 1 if $x_i = 1$ (or by the edge labeled 0 if $x_i = 0$). If the pebble eventually reaches the sink labeled b, then $f(x) = b$. Branching programs can also be used to compute functions $f : \{0,1\}^m \to \{0,1\}^n$, by concatenating n branching programs $p_1, \ldots, p_n$, where p_i computes the function $f_i(x) =$ the i-th bit of $f(x)$. For more information on the definitions, backgrounds, and nuances of these complexity classes, circuits, and branching programs, see the text by Vollmer [30].*

▶ **Definition 4.** *Non-interactive zero-knowledge proof (*NISZK*) [Adapted from [2, 17]]: A non-interactive statistical zero-knowledge proof system for a promise problem Π is defined by a pair of deterministic polynomial time machines[5] (V, S) (the* verifier *and* simulator*, respectively) and a probabilistic routine P (the* prover*) that is computationally unbounded, together with a polynomial $r(n)$ (which will give the size of the random reference string σ), such that:*

1. *(Completeness): For all $x \in \Pi_Y$, the probability (over random σ, and over the random choices of P) that $V(x, \sigma, P(x, \sigma))$ accepts is at least $1 - 2^{-O(|x|)}$.*
2. *(Soundness): For all $x \in \Pi_N$, and for every possible prover P', the probability that $V(x, \sigma, P'(x, \sigma))$ accepts is at most $2^{-O(|x|)}$. (Note P' here can be malicious, meaning it can try to fool the verifier)*
3. *(Zero Knowledge): For all $x \in \Pi_Y$, the statistical distance between the following two distributions is bounded by $2^{-|x|}$:*
   a. *Choose $\sigma \leftarrow \{0,1\}^{r(|x|)}$ uniformly random, $p \leftarrow P(x, \sigma)$, and output (p, σ).*
   b. *$S(x, r)$ (where the coins r for S are chosen uniformly at random).*

It is known that changing the definition, to have the error probability in the soundness and completeness conditions and in the simulator's deviation be $\frac{1}{n^{\omega(1)}}$ results in an equivalent definition [2, 17]. (See the comments after [2, Claim 39].) We will occasionally make use of this equivalent formulation, when it is convenient.

NISZK *is the class of promise problems for which there is a non-interactive statistical zero knowledge proof system.*

---

[5] In prior work on NISZK [17, 2], the verifier and simulator were said to be probabilistic machines. We prefer to be explicit about the random input sequences provided to each machine, and thus the machines can be viewed as deterministic machines taking a sequence of random bits as input.

$\mathsf{NISZK}_{\mathcal{C}}$ *denotes the class of problems in* NISZK *where the verifier* V *and simulator* S *lie in complexity class* $\mathcal{C}$.

▶ **Definition 5.** *[2, 17] (*EA *and* $\mathsf{EA}_{\mathsf{NC}^0}$*). Consider Boolean circuits* $C_X : \{0,1\}^m \to \{0,1\}^n$ *representing distribution* X. *(That is,* $\Pr[X = x] = \Pr[C(y) = x]$ *where* y *is chosen uniformly at random.) The promise problem* EA *is given by:*

$$\mathsf{EA}_Y := \{(C_X, k) : H(X) > k + 1\}$$

$$\mathsf{EA}_N := \{(C_X, k) : H(X) < k - 1\}$$

$\mathsf{EA}_{\mathsf{NC}^0}$ *is the variant of* EA *where the distribution* C_X *is an* NC^0 *circuit with each output bit depending on at most 4 input bits.*

▶ **Definition 6** (SDU and $\mathsf{SDU}_{\mathsf{NC}^0}$). *Consider Boolean circuits* $C_X : \{0,1\}^m \to \{0,1\}^n$ *representing distributions* X. *The promise problem* $\mathsf{SDU} = (\mathsf{SDU}_Y, \mathsf{SDU}_N)$ *is given by:*

$$\mathsf{SDU}_Y := \{C_X : \Delta(X, U_n) < 1/n\}$$

$$\mathsf{SDU}_N := \{C_X : \Delta(X, U_n) > 1 - 1/n\}.$$

$\mathsf{SDU}_{\mathsf{NC}^0}$ *is the analogous problem, where the distributions* X *are represented by* NC^0 *circuits where no output bit depends on more than* four *input bits.*

▶ **Theorem 7.** *[2, 5]:* $\mathsf{EA}_{\mathsf{NC}^0}$ *and* $\mathsf{SDU}_{\mathsf{NC}^0}$ *are complete for* $\mathsf{NISZK}_{\mathsf{L}}$ *under* $\leq_{\mathrm{m}}^{\mathsf{proj}}$. $\mathsf{EA}_{\mathsf{NC}^0}$ *remains complete, even if* k *is fixed to* $k = n - 3$.

▶ **Definition 8.** *[15, 28] (*SD *and* $\mathsf{SD}_{\mathsf{BP}}$*). Consider a pair of Boolean circuits* $C_1, C_2 : \{0,1\}^m \to \{0,1\}^n$ *representing distributions* X_1, X_2. *The promise problem* SD *is given by:*

$$\mathsf{SD}_Y := \{(C_1, C_2) : \Delta(X_1, X_2) > 2/3\}$$

$$\mathsf{SD}_N := \{(C_1, C_2) : \Delta(X_1, X_2) < 1/3\}.$$

$\mathsf{SD}_{\mathsf{BP}}$ *is the variant of* SD *where the distributions* X_1, X_2 *are represented by branching programs.*

2.1 Perfect Randomized Encodings

We will make use of the machinery of *perfect randomized encodings* [9].

▶ **Definition 9.** *Let* $f : \{0,1\}^n \to \{0,1\}^\ell$ *be a function. We say that* $\hat{f} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^s$ *is a perfect randomized encoding of* f *with blowup* b *if it is:*

- ***Input independent:*** *for every* $x, x' \in \{0,1\}^n$ *such that* $f(x) = f(x')$, *the random variables* $\hat{f}(x, U_m)$ *and* $\hat{f}(x', U_m)$ *are identically distributed.*
- ***Output Disjoint:*** *for every* $x, x' \in \{0,1\}^n$ *such that* $f(x) \neq f(x')$, $\mathrm{supp}(\hat{f}(x, U_m)) \cap \mathrm{supp}(\hat{f}(x', U_m)) = \emptyset$.
- ***Uniform:*** *for every* $x \in \{0,1\}^n$ *the random variable* $\hat{f}(x, U_m)$ *is uniform over the set* $\mathrm{supp}(\hat{f}(x, U_m))$.
- ***Balanced:*** *for every* $x, x' \in \{0,1\}^n$ $|\mathrm{supp}(\hat{f}(x, U_m))| = |\mathrm{supp}(\hat{f}(x', U_m))| = b$.

The following property of perfect randomized encodings is established in [15].

▶ **Lemma 10.** *Let* $f : \{0,1\}^n \to \{0,1\}^\ell$ *be a function and let* $\hat{f} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^s$ *be a perfect randomized encoding of* f *with blowup* b. *Then* $H(\hat{f}(U_n, U_m)) = H(f(U_n)) + \log b$.

3 Simulators and Verifiers in AC^0

In this section, we show that $\mathsf{NISZK_L}$ can be defined equivalently using verifiers and simulators that are computable in AC^0. The standard complete problems for NISZK and $\mathsf{NISZK_L}$ take a circuit C as input, where the circuit is viewed as representing a probability distribution X; the goal is to approximate the entropy of X, or to estimate how far X is from the uniform distribution. Earlier work [18, 2, 27] that had presented non-interactive zero-knowledge protocols for these problems had made use of the fact that the verifier could compute hash functions, and thereby convert low-entropy distributions to distributions with small support. But an AC^0 verifier cannot compute hash functions [22].

Our approach is to "delegate" the problem of computing hash functions to a logspace verifier, and then to make use of the uniform encoding of this verifier to obtain the desired distributions via an AC^0 reduction.[6] To this end, we begin by defining a suitably restricted version of $\mathsf{SDU_{NC^0}}$ and show (in Section 3.1) that this restricted version remains complete for $\mathsf{NISZK_L}$ under AC^0 reductions (and even under projections).[7]

With this new complete problem in hand, we provide (in Section 3.2) a $\mathsf{NISZK_{AC^0}}$ protocol for the complete problem, proving its correctness in Section 3.3, to conclude with the main result of this section:

▶ **Theorem 11.** $\mathsf{NISZK_L} = \mathsf{NISZK_{AC^0}}$.

▶ **Definition 12.** *Consider an NC^0 circuit $C : \{0,1\}^m \to \{0,1\}^n$ and the probability distribution X on $\{0,1\}^n$ defined as $C(U_m)$ - where U_m denotes m uniformly random bits. For some fixed $\epsilon > 0$ (chosen later in Remark 17), we define:*

$$\mathsf{SDU'}_{\mathsf{NC^0},Y} = \{X : \Delta(C, U_n) < \frac{1}{2^{n^\epsilon}}\}$$

$$\mathsf{SDU'}_{\mathsf{NC^0},N} = \{X : |\operatorname{supp}(X)| \le 2^{n-n^\epsilon}\}$$

We will show that $\mathsf{SDU'_{NC^0}}$ is complete for $\mathsf{NISZK_L}$ under uniform $\le_m^{\mathsf{proj}}$ reductions. In order to do so, we first show that $\mathsf{SDU'_{NC^0}}$ is in $\mathsf{NISZK_L}$ by providing a reduction to $\mathsf{SDU_{NC^0}}$.

▷ Claim 13. $\mathsf{SDU'_{NC^0}} \le_m^{\mathsf{proj}} \mathsf{SDU_{NC^0}}$, and thus $\mathsf{SDU'_{NC^0}} \in \mathsf{NISZK_L}$.

Proof. On a given probability distribution X defined on $\{0,1\}^n$ for $\mathsf{SDU'_{NC^0}}$, we claim that the identity function $f(X) = X$ is a reduction of $\mathsf{SDU'_{NC^0}}$ to $\mathsf{SDU_{NC^0}}$. If X is a YES instance for $\mathsf{SDU'_{NC^0}}$, then $\Delta(X, U_n) < \frac{1}{2^{n^\epsilon}}$, which clearly is a YES instance of $\mathsf{SDU_{NC^0}}$. If X is a NO instance for $\mathsf{SDU'_{NC^0}}$, then $|\operatorname{supp}(X)| \le 2^{n-n^\epsilon}$. Thus, if we let T be the complement of $\operatorname{supp}(X)$, we have that, under the uniform distribution, a string α is in T with probability $\ge 1 - \frac{1}{2^{n^\epsilon}}$, whereas this event has probability zero under X. Thus $\Delta(X, U_n) \ge 1 - \frac{1}{2^{n^\epsilon}}$, easily making it a NO instance of $\mathsf{SDU_{NC^0}}$. ◀

3.1 Hardness for $\mathsf{SDU'_{NC^0}}$

▶ **Theorem 14.** $\mathsf{SDU'_{NC^0}}$ *is hard for* $\mathsf{NISZK_L}$ *under* $\le_m^{\mathsf{proj}}$ *reductions.*

[6] In retrospect, the proof of the one-sided-error part of [5, Theorem 32] implicitly requires that this restriction be complete for $\mathsf{NISZK_L}$. Hence we are now providing a missing part of that proof.

[7] This restricted version of $\mathsf{SDU_{NC^0}}$ can be seen as a version of the "image density" problem that was defined and studied in [14].

Proof. In order to show that $\mathsf{SDU'}_{\mathsf{NC}^0}$ is hard for $\mathsf{NISZK_L}$, we will show that the reduction given in [2] proving the hardness of $\mathsf{SDU}_{\mathsf{NC}^0}$ for $\mathsf{NISZK_L}$ actually produces an instance of $\mathsf{SDU'}_{\mathsf{NC}^0}$.

Let Π be an arbitrary promise problem in $\mathsf{NISZK_L}$ with proof system (P, V) and simulator S. Let x be an instance of Π. Let $M_x(r)$ denote a machine that simulates $S(x)$ with randomness r to obtain a transcript (σ, p) - if $V(x, \sigma, p)$ accepts then $M_x(r)$ outputs σ; else it outputs $0^{|\sigma|}$. We will assume without loss of generality that $|\sigma| = n^k$ for some constant k.

It was shown in [18, Lemma 3.1] that for the promise problem EA, there is an NISZK protocol with completeness error, soundness error and simulator deviation all bounded from above by 2^{-m} for inputs of length m. Furthermore, as noted in the paragraph before Claim 38 in [2], the proof carries over to show that $\mathsf{EA_{BP}}$ has an $\mathsf{NISZK_L}$ protocol with the same parameters. Thus, any problem in $\mathsf{NISZK_L}$ can be recognized with exponentially small error parameters by reducing the problem to $\mathsf{EA_{BP}}$ and then running the above protocol for $\mathsf{EA_{BP}}$ on that instance. In particular, this holds for $\mathsf{EA}_{\mathsf{NC}^0}$. In what follows, let M_x be the distribution described in the preceding paragraph, assuming that the simulator S and verifier V yield a protocol with these exponentially small error parameters.

▷ Claim 15. If $x \in \Pi_{YES}$ then $\Delta(M_x(r), U_{n^k}) \leq 1/2^{n-1}$. And if $x \in \Pi_{NO}$ then $|\operatorname{supp}(M_x(r))| \leq 2^{n^k - n^{\epsilon k}}$ for $\epsilon < \frac{1}{k}$.

Proof. For $x \in \Pi_{YES}$, claim 38 of [2] shows that $\Delta(M_x(r), U_{n^k}) \leq 1/2^{n-1}$, establishing the first part of the claim.

For $x \in \Pi_{NO}$, from the soundness guarantee of the $\mathsf{NISZK_L}$ protocol for $\mathsf{EA}_{\mathsf{NC}^0}$, we know that, for at least a $1 - \frac{1}{2^n}$ fraction of the shared reference strings $\sigma \in \{0,1\}^{n^k}$, there is no message p that the prover can send that will cause V to accept. Thus there are at most $2^{n^k - n}$ outputs of $M_x(r)$ other than 0^{n^k}. For $\epsilon < \frac{1}{k}$, we have $|\operatorname{supp}(M_x(r))| \leq 2^{n^k - n^{\epsilon k}}$. ◀

The above claim talks about the distribution $M_x(r)$ where M is a logspace machine. We will instead consider an NC^0 distribution with similar properties that can be constructed using projections. This distribution (denoted by C_x) is a perfect randomized encoding of $M_x(r)$. We make use of the following construction:

▶ **Lemma 16.** *[2, Lemma 35]. There is a function computable in AC^0 (in fact, it can be a projection) that takes as input a branching program*[8] *Q of size l computing a function f and produces as output a list p_i of NC^0 circuits, where p_i computes the i-th bit of a function $\hat{f}$ that is a perfect randomized encoding of f that has blowup $b = 2^{(\binom{l}{2}-1)2((l-1)^2-1)}$ (and thus the length of $\hat{f}(r) = \log b + |f(r)|$). Each p_i depends on at most four input bits from (x, r) (where r is the sequence of random bits in the randomized encoding).*

The properties of perfect randomized encodings (see Definition 9) imply that the range of $\hat{f}$ (and thus also the range of C_x) can be partitioned into equal sized pieces corresponding to each value of $f(r)$. Thus, let $\alpha_1, \alpha_2, .., \alpha_z$ be the range of $f(r)$, and let $[\alpha] = \{\hat{f}(r, s) : f(r) = \alpha\}$. It follows that $|[\alpha]| = b$. For a given α, and for a given β of length $\log b$ we denote by $\alpha\beta$ the β-th element of $[\alpha]$. Since the simulator S runs in logspace, each bit of $M_x(r)$ can be simulated with a branching program Q_x. Furthermore, it is straightforward to see that there

[8] The reviewers have requested additional detail, regarding the format in which a branching program is presented. In the context of [2, Lemma 35], the branching program can be presented as a matrix A, where $A_{i,j}$ is (b, k) if there is a transition from node i to node j if bit position x_k is equal to b, and $A_{i,j}$ is equal to 1 (0) if there is unconditionally (not) a transition from node i to node j.

is an AC^0-computable function that takes x as input and produces an encoding of Q_x as output, and it can even be seen that this function can be a projection. Let the list of NC^0 circuits produced from Q_x by the construction of Lemma 16 be denoted C_x.

We show that this distribution C_x is an instance of $\mathsf{SDU'}_{\mathsf{NC}^0}$ if $x \in \Pi$. For $x \in \Pi_{YES}$, we have $\Delta(M_x(r), U_{n^k}) \leq 1/2^{n-1}$, and we want to show $\Delta(C_x(r), U_{\log b + n^k}) \leq 1/2^{n-1}$. Thus it will suffice to observe that $\Delta(M_x(r), U_{n^k}) = \Delta(C_x(r), U_{\log b + n^k}) \leq 1/2^{n-1}$.

To see this, note that

$$\Delta(C_x(r), U_{\log b + n^k}) = \sum_{\alpha\beta} \big| \Pr[C_x = \alpha\beta] - \frac{1}{2^{n^k+b}} \big|/2 = \sum_{\beta}\sum_{\alpha} \big| \Pr[M_x = \alpha]\frac{1}{2^b} - \frac{1}{2^b}\frac{1}{2^{n^k}} \big|/2$$

$$= \sum_{\alpha} \big| \Pr[M_x = \alpha] - \frac{1}{2^{n^k}} \big|/2 = \Delta(M_x(r), \mathcal{U}_{n^k}).$$

Thus, for $x \in \Pi_{YES}$, C_x is a YES instance for $\mathsf{SDU'}_{\mathsf{NC}^0}$.

For $x \in \Pi_{NO}$, Claim 15 shows that $|\operatorname{supp}(M_x(r))| \leq 2^{n^k - n}$. Since the NC^0 circuit C_x is a perfect randomized encoding of $M_x(r)$, we have that the size of the support of C_x for $x \in \Pi_{NO}$ is bounded from above by $b \times 2^{n^k-n}$. Note that $\log b$ is polynomial in n; let $q(n) = \log b$. Let $r(n)$ denote the length of the output of C; $r(n) = q(n) + n^k$. Thus the size of $\operatorname{supp}(C_x) \leq 2^{n^k - n + q(n)} = 2^{r(n)-n} < 2^{r(n)-r(n)^\epsilon}$ (if $1/\epsilon$ is chosen to be greater than the degree of $r(n)$), and hence C_x is a NO instance for $\mathsf{SDU'}_{\mathsf{NC}^0}$. ◀

▶ Remark 17. Here is how we pick ϵ in the definition of $\mathsf{SDU'}_{\mathsf{NC}^0}$. $\mathsf{SDU}_{\mathsf{NC}^0}$ is in $\mathsf{NISZK}_\mathsf{L}$ via some simulator and verifier, where the error parameters are exponentially small, and the shared reference strings σ have length n^k on inputs of length n. Now we pick $\epsilon > 0$ so that $\epsilon < 1/k$ (as in Claim 15) and also $1/\epsilon$ is greater than the degree of $r(n)$ (as in the last sentence of the proof of Theorem 14).

3.2 $\mathsf{NISZK}_{\mathsf{AC}^0}$ protocol for $\mathsf{SDU'}_{\mathsf{NC}^0}$

In this section, we provide an $\mathsf{NISZK}_{\mathsf{AC}^0}$ protocol for $\mathsf{SDU'}_{\mathsf{NC}^0}$ to conclude the proof of Theorem 11. We then prove the correctness of this protocol in Section 3.3. As above, we will consider the input distribution X on $\{0,1\}^n$ defined by some NC^0 circuit $C : \{0,1\}^m \to \{0,1\}^n$.

▶ **Theorem 18.** $\mathsf{SDU'}_{\mathsf{NC}^0} \in \mathsf{NISZK}_{\mathsf{AC}^0}$.

Proof. We first provide an $\mathsf{NISZK}_{\mathsf{AC}^0}$ protocol for $\mathsf{SDU'}_{\mathsf{NC}^0}$ by specifying the behavior of the Prover, Verifier and Simulator machines. The proofs of zero knowledge, completeness and soundness follow in section 3.3.

3.2.1 Non Interactive proof system for $\mathsf{SDU'}_{\mathsf{NC}^0}$

1. Let C take inputs of length m and produce outputs of length n, and let σ be the reference string of length n.
2. If there is no r such that $C(r) = \sigma$, then the prover sends $\perp$. Otherwise, the prover picks an element r uniformly at random from the set $\{r|C(r) = \sigma\}$ and sends it to the verifier.
3. V accepts iff $C(r) = \sigma$. (Since C is an NC^0 circuit, this can be accomplished in AC^0 – this step can not be accomplished in NC^0 since it depends on all of the bits of σ.)

3.2.2 Simulator for $\mathsf{SDU'}_{\mathsf{NC}^0}$ proof system

1. Pick a random s of length m and compute $\gamma = C(s)$.
2. Output (s, γ).

3.3 Proofs of Zero Knowledge, Completeness and Soundness

3.3.1 Completeness

▷ Claim 19. If $X \in \mathsf{SDU'}_{\mathsf{NC}^0,Y}$, then the verifier accepts with probability $\geq 1 - \frac{1}{2^{n^\epsilon}}$.

Proof. If X is a YES instance, then $\Delta(X, U_n) < \frac{1}{2^{n^\epsilon}}$. This implies $|\operatorname{supp}(X)| > 2^n(1 - \frac{1}{2^{n^\epsilon}})$, which immediately implies the stated lower bound on the verifier's probability of acceptance. ◀

3.3.2 Soundness

▷ Claim 20. If $X \in \mathsf{SDU'}_{\mathsf{NC}^0,N}$, then for every prover, the probability that the verifier accepts is at most $\frac{1}{2^{n^\epsilon}}$.

Proof. For every $\sigma \notin \operatorname{supp}(X)$, no prover can make the verifier accept. If $X \in \mathsf{SDU'}_{\mathsf{NC}^0,N}$, the probability that $\sigma \notin \operatorname{supp}(X)$ is greater than $1 - \frac{1}{2^{n^\epsilon}}$. ◀

3.3.3 Statistical Zero-Knowledge

▷ Claim 21. For $X \in \mathsf{SDU'}_{\mathsf{NC}^0,Y}$, $\Delta((p, \sigma), (s, \gamma)) = O(\frac{1}{2^{n^\epsilon}})$.

Proof. Since we are considering only YES instances $X \in \mathsf{SDU'}_{\mathsf{NC}^0,Y}$, we have that $\Pr[\sigma \notin \operatorname{range}(C)] \leq \frac{1}{2^{n^\epsilon}}$. Thus $\Pr[(\perp, \sigma)] \leq \frac{1}{2^{n^\epsilon}}$. Thus, in the subsequent analysis, we consider only the case where the prover's message is not equal to $\perp$.

Recall that $\sigma \sim \{0,1\}^n$, $s \sim \{0,1\}^m$, $p \sim \{r : C(r) = \sigma\}$ and $\gamma = C(s)$. In order to provide an upper bound on $\Delta((p, \sigma), (s, \gamma))$, we consider the element wise probability of each distribution and show that for $X \in \mathsf{SDU'}_{\mathsf{NC}^0,Y}$ the claim holds. For $a \in \{0,1\}^m$ and $b \in \{0,1\}^n$ we have :

$$\Delta((p, \sigma), (s, \gamma)) = \sum_{(a,b)} \frac{1}{2} \left|\Pr[(p, \sigma) = (a, b)] - \Pr[(s, \gamma) = (a, b)]\right|$$

Let us consider an element $b \in \{0,1\}^n$. Let $A_b = \{a_1, a_2, .., a_{k_b}\}$ be the pre-images of b under C; that is, for $1 \leq i \leq k_b$ it holds that $C(a_i) = b$. Let $\beta_b = \Pr_{y \sim U_m}[C(y) = b]$. Then $k_b 2^{-m} = \beta_b$ (since exactly k_b elements of $\{0,1\}^m$ are mapped to b under C). Let $B = \{b | \neg\exists y : C(y) = b\}$. Since $\Delta(C(U_m), U_n) \leq \frac{1}{2^{n^\epsilon}}$, it follows that $\frac{|B|}{2^m} \leq \frac{1}{2^{n^\epsilon}}$. We have :

$$\begin{aligned}
\Delta((p, \sigma), (s, \gamma)) &= \sum_{(a,b)} \frac{1}{2}(|\Pr[(p, \sigma) = (a, b)] - \Pr[(s, \gamma) = (a, b)]|) \\
&= \frac{1}{2} \sum_{(a,b):b \in B} |\Pr[(p, \sigma) = (a, b)] - \Pr[(s, \gamma) = (a, b)]| \\
&\quad + \frac{1}{2} \sum_{(a,b):b \notin B} |\Pr[(p, \sigma) = (a, b)] - \Pr[(s, \gamma) = (a, b)]|
\end{aligned}$$

For (a, b) satisfying $b \in B$, we have $\Pr[(s, \gamma) = (a, b)] = \Pr[(p, \sigma) = (a, b)] = 0$. For $b \notin B$ and a satisfying $C(a) \neq b$ we again have $\Pr[(s, \gamma) = (a, b)] = \Pr[(p, \sigma) = (a, b)] = 0$. For (a, b) satisfying $C(a) = b$ we have $\Pr[(s, \gamma) = (a, b)] = 2^{-m}$ since $s \sim U_m$ and picking s fixes b. We also have $\Pr[(p, \sigma) = (a, b)] = \frac{2^{-n}}{k_b}$ since $\sigma \sim U_n$ and then the prover picks p uniformly from

A_b. This gives us

$$\Delta((p,\sigma),(s,\gamma)) = \frac{1}{2}\sum_{(a,b):C(a)=b}\left|2^{-m} - \frac{2^{-n}}{k_b}\right|$$
$$= \frac{1}{2}\sum_{(a,b):C(a)=b}\left|2^{-m} - \frac{2^{-m-n}}{\beta_b}\right|$$
$$= \frac{1}{2}\sum_{(a,b):C(a)=b}\frac{2^{-m}}{\beta_b}\left|\beta_b - 2^{-n}\right|$$
$$\leq \frac{1}{2}\sum_{(a,b):C(a)=b}\left|\beta_b - 2^{-n}\right| = \Delta(C(U_m),U_n) \leq \frac{1}{2^{n^\epsilon}}$$

where the first inequality holds since $\beta_b \geq 2^{-m}$ whenever $\beta_b \neq 0$. Thus we have :

$$\Delta((p,\sigma),(s,\gamma)) = O(\frac{1}{2^{n^\epsilon}}).$$

◀

This concludes the proof of Theorem 18 - $\mathsf{SDU'}_{\mathsf{NC}^0} \in \mathsf{NISZK}_{\mathsf{AC}^0}$. Combining this with Theorem 14, we conclude the proof of Theorem 11 - $\mathsf{NISZK}_\mathsf{L} = \mathsf{NISZK}_{\mathsf{AC}^0}$. ◀

4 Simulator and Verifier in PM

In this section, we show that $\mathsf{NISZK}_\mathsf{L}$ can be defined equivalently using verifiers and simulators that lie in the class PM of problems that logspace-Turing reduce to Perfect Matching. (PM is not known to lie in (uniform) NC.) That is, we can increase the computational power of the simulator and the verifier from L to PM without affecting the power of noninteractive statistical zero knowledge protocols.

The Perfect Matching problem is the well-known problem of deciding, given an undirected graph G with $2n$ vertices, if there is a set of n edges covering all of the vertices. We define a corresponding complexity class PM as follows:

$$\mathsf{PM} := \{A : A \leq_T^L \text{Perfect Matching}\}$$

It is known that $\mathsf{NL} \subseteq \mathsf{PM}$ [21].

Our argument proceeds by first observing[9] that $\mathsf{NISZK}_\mathsf{L} = \mathsf{NISZK}_{\oplus\mathsf{L}}$, and then making use of the details of the argument that Perfect Matching is in $\oplus\mathsf{L/poly}$ [8].

▶ **Proposition 22.** $\mathsf{NISZK}_{\oplus\mathsf{L}} = \mathsf{NISZK}_\mathsf{L}$

Proof. It suffices to show $\mathsf{NISZK}_{\oplus\mathsf{L}} \subseteq \mathsf{NISZK}_\mathsf{L}$. We do this by showing that the problem $\mathsf{EA}_{\mathsf{NC}^0}$ is hard for $\mathsf{NISZK}_{\oplus\mathsf{L}}$; this suffices since $\mathsf{EA}_{\mathsf{NC}^0}$ is complete for $\mathsf{NISZK}_\mathsf{L}$. The proof of [2, Theorem 26] (showing that $\mathsf{EA}_{\mathsf{NC}^0}$ is complete for $\mathsf{NISZK}_\mathsf{L}$ involves (a) building a branching program to simulate a logspace computation called M_x that is constructed from a logspace-computable simulator and verifier, and (b) constructing an NC^0-computable perfect randomized encoding of M_x, using the fact that $\mathsf{L} \subset \mathcal{PREN}$, where $\mathcal{PREN}$ is the class defined in [9], consisting of all problems with perfect randomized encodings. But Theorem

[9] This equality was previously observed in [27].

4.18 in [9] shows the stronger result that $\oplus\mathsf{L}$ lies in $\mathcal{PREN}$, and hence the argument of [2, Theorem 26] carries over immediately, to reduce any problem in $\mathsf{NISZK}_{\oplus\mathsf{L}}$ to $\mathsf{EA}_{\mathsf{NC}^0}$ (by modifying step (a), to build a *parity* branching program for M_x that is constructed from a $\oplus\mathsf{L}$ simulator and verifier). ◀

We also rely on the following lemma:

▶ **Lemma 23.** *Adapted from [8, Section 3] and [24, Section 4]: Let $W = (w_1, w_2, \cdots, w_{n^{k+3}})$ be a sequence of n^{k+3} weight functions, where each $w_i : [\binom{n}{2}] \to [4n^2]$ is a distinct weight assignment to edges in n-vertex graphs. Let (G, w_i) denote the result of weighting the edges of G using weight assignment w_i. Then there is a function f in* GapL*, such that, if (G, w_i) has a unique perfect matching of weight j, then $f(G, W, i, j) \in \{1, -1\}$, and if G has no perfect matching, then for* every *(W, i, j), it holds that $f(G, W, i, j) = 0$. Furthermore, if W is chosen uniformly at random, then with probability $\geq 1 - 2^{-n^k}$, for <u>each</u> n-vertex graph G:*

- *If G has no perfect matching then $\forall i \forall j\ f(G, W, i, j) = 0$.*
- *If G has a perfect matching then $\exists i$ such that (G, w_i) has a unique minimum-weight matching, and hence $\exists i \exists j\ f(G, W, i, j) \in \{1, -1\}$.*

Thus if we define $g(G, W)$ to be $1 - \Pi_{i,j}(1 - f(G, W, i, j)^2)$, we have that $g \in$ GapL *(by the closure properties of* GapL *established in [7, Section 4]) and with probability $\geq 1 - 2^{-n^k}$ (for randomly-chosen W), $g(G, W) = 1$ if G has a perfect matching, and $g(G, W) = 0$ otherwise.*

Note that this lemma is saying that most W constitute a good "advice string", in the sense that $g(G, W)$ provides the correct answer to the question "Does G have a perfect matching?" for every graph G with n vertices.

▶ **Corollary 24.** *For every language $A \in$* PM *there is a language $B \in \oplus\mathsf{L}$ such that, if $x \in A$, then $\Pr_{W \leftarrow [4n^2]^{n^5}}[(x, W) \in B] \geq 1 - 2^{-n^2}$, and if $x \notin A$, then $\Pr_{W \leftarrow [4n^2]^{n^5}}[(x, W) \in B] \leq 2^{-n^2}$.*

Proof. Let A be in PM, where there is a logspace oracle machine M accepting A with an oracle P for Perfect Matching. We may assume without loss of generality that all queries made by M on inputs of length n have the same number of vertices $p(n)$. This is because G has a perfect matching iff $G \cup \{x_1 - y_1, x_2 - y_2, ..., x_k - y_k\}$ has a perfect matching. (I.e., we can "pad" the queries, to make them all the same length.)

Let $C = \{(G, W) : g(G, W) \equiv 1 \bmod 2\}$, where g is the function from Lemma 23. Clearly, $C \in \oplus\mathsf{L}$. Now, a logspace oracle machine with input (x, W) and oracle C can simulate the computation of M^P on x; each time M poses the query "Is $G \in P$", instead we ask if $(G, W) \in C$. Then with high probability (over the random choice of W) all of the queries will be answered correctly and hence this routine will accept if and only if $x \in A$, by Lemma 23. Let B be the language accepted by this logspace oracle machine. We see that $B \in \mathsf{L}^C \subseteq \mathsf{L}^{\oplus\mathsf{L}} = \oplus\mathsf{L}$, where the last equality is from [19]. ◀

▶ **Theorem 25.** $\mathsf{NISZK}_\mathsf{L} = \mathsf{NISZK}_\mathsf{PM}$

Proof. We show that $\mathsf{NISZK}_\mathsf{PM} \subseteq \mathsf{NISZK}_{\oplus\mathsf{L}}$, and then appeal to Proposition 22.

Let Π be an arbitrary problem in $\mathsf{NISZK}_\mathsf{PM}$, and let (S, P, V) be the PM simulator, prover, and verifier for Π, respectively. Let S' and V' be the $\oplus\mathsf{L}$ languages that are probabilistic realizations of S, V, respectively, guaranteed by Corollary 24. We now define a $\mathsf{NISZK}_\mathsf{L}$ protocol (S'', P'', V'') for Π.

On input x with shared randomness σW, the prover P'' sends the same message $p = P(x, \sigma)$ as the original prover sends. The verifier V'', returns the value of $V'((x, \sigma, p), W)$, which with high probability is equal to $V(x, \sigma, p)$. The simulator S'', given as input x and random sequence rW, executes $S'((x, r, i), W)$ for each bit position i to obtain a bit that (with high probability) is equal to the i^{th} bit of $S(x, r)$, which is a string of the form (σ, p), and outputs $(\sigma W, p)$.

Now we will analyze the properties of (S'', P'', V''):

- Completeness: Suppose $x \in \Pi_Y$, then $\Pr_\sigma[V(x, \sigma, P(x, \sigma)) = 1] \geq 1 - 2^{-O(n)}$. Since $\forall y \in \{0,1\}^n : \Pr_W[V(y) = V'(y, W)] \geq 1 - 2^{-n^k}$ we have:

$$\Pr_{\sigma W}[V'((x, \sigma, P''(x, \sigma)), W) = 1] \geq [1 - 2^{-O(n)}][1 - 2^{-n^k}] = 1 - 2^{-O(n)}$$

- Soundness: Suppose $x \in \Pi_N$, then $\Pr_\sigma[\forall p : V(x, \sigma, p) = 0] \geq 1 - 2^{-O(n)}$. Since $\forall y \in \{0,1\}^n : \Pr_W[V(y) = V'(y, W)] \geq 1 - 2^{-n^k}$, we have:

$$\Pr_{\sigma W}[\forall p : V'((x, \sigma, p), W) = 0] \geq [1 - 2^{-O(n)}][1 - 2^{-n^k}] = 1 - 2^{-O(n)}$$

- Statistical Zero-Knowledge: Suppose $x \in \Pi_Y$. Let S^* denote the distribution on strings of the form (σ, p) that $S(x, r)$ produces, where r is uniformly generated, and let P^* denote the distribution on strings given by $(\sigma, P(x, \sigma))$ where σ is chosen uniformly at random. Similarly, let S''^* denote the distribution on strings of the form $(\sigma W, p)$ that $S''(x, rW)$ produces, where r and W are chosen uniformly, and let P''^* be the distribution given by $(\sigma W, P''(x, \sigma W))$. Let $A = \{(\sigma W, p) : \exists i \exists r\ S(x, r)_i \neq S'((x, r, i), W)\}$. Since $\Pr_W[\forall i \forall r : S(x, r)_i = S'((x, r, i), W)] \geq 1 - 2^{-O(n)}$ we have:

$$\begin{aligned}
\Delta(S''^*, P''^*) &= \frac{1}{2} \sum_{(\sigma W, p)} \big| \Pr[S''^* = (\sigma W, p)] - \Pr[P''^* = (\sigma W, p)] \big| \\
&\leq \frac{1}{2}(2^{-O(n)} + \sum_{(\sigma W, p) \in \overline{A}} \big| \Pr[S''^* = (\sigma W, p)] - \Pr[P''^* = (\sigma W, p)]) \big| \\
&= \frac{1}{2}(2^{-O(n)} + \sum_{(\sigma W, p) \in \overline{A}} \big| \Pr[S^* = (\sigma, p)] - \Pr[P^* = (\sigma, p)] \big| \Pr[W]) \\
&\leq 2^{-O(n)} + \sum_W \Pr[W] \frac{1}{2} \sum_{(\sigma, p)} \big| \Pr[S^* = (\sigma, p)] - \Pr[P^* = (\sigma, p)] \big| \\
&= 2^{-O(n)} + \Delta(S^*, P^*) = 2^{-O(n)}
\end{aligned}$$

Therefore (S'', P'', V'') is a $\mathsf{NISZK}_{\oplus \mathsf{L}}$ protocol deciding Π. ◀

5 Additional problems in $\mathsf{NISZK}_\mathsf{L}$

In this section, we give additional examples of problems in P that lie in $\mathsf{NISZK}_\mathsf{L}$. These problems are not known to lie in (uniform) NC. Our main tool is to show that $\mathsf{NISZK}_\mathsf{L}$ is closed under a class of randomized reductions.

The following definition is from [5]:

▶ **Definition 26.** *A promise problem $A = (Y, N)$ is $\leq_{\mathrm{m}}^{\mathsf{BPL}}$-reducible to $B = (Y', N')$ with threshold θ if there is a logspace-computable function f and there is a polynomial p such that*

- *$x \in Y$ implies $\Pr_{r \in \{0,1\}^{p(|x|)}}[f(x, r) \in Y'] \geq \theta$.*
- *$x \in N$ implies $\Pr_{r \in \{0,1\}^{p(|x|)}}[f(x, r) \in N'] \geq \theta$.*

Note, in particular, that the logspace machine computing the reduction has two-way access to the random bits r; this is consistent with the model of probabilistic logspace that is used in defining $\mathsf{NISZK}_\mathsf{L}$.

▶ **Theorem 27.** $\mathsf{NISZK_L}$ *is closed under* $\leq_m^{\mathsf{BPL}}$ *reductions with threshold* $1 - \frac{1}{n^{\omega(1)}}$.

Proof. Let $\Pi \leq_{\mathrm{m}}^{\mathsf{BPL}} \mathsf{EA}_{\mathsf{NC}^0}$, via logspace-computable function f. Let (S_1, V_1, P_1) be the $\mathsf{NISZK_L}$ proof system for $\mathsf{EA}_{\mathsf{NC}^0}$.

Algorithm 1 Simulator $S(x, r\sigma')$

$(\sigma, p) \leftarrow S_1(f(x, \sigma'), r)$;
return $((\sigma, \sigma'), p)$;

Algorithm 2 Verifier $V(x, (\sigma, \sigma'), p)$

return $V_1((f(x, \sigma'), \sigma, p))$

Algorithm 3 Prover $P(x, (\sigma, \sigma'))$

return $P_1((f(x, \sigma'), \sigma))$;

We now claim that (S, P, V) is a $\mathsf{NISZK_L}$ protocol for Π.

It is apparent that S and V are computable in logspace. We just need to go through completeness, soundness, and statistical zero-knowledge of this protocol.

- Completeness: Suppose x is YES instance of Π. Then with probability $1 - \frac{1}{n^{\omega(1)}}$ (over randomness of σ'), we have that $f(x, \sigma')$ is a YES instance of $\mathsf{EA}_{\mathsf{NC}^0}$. Thus for a randomly chosen σ:

$$\Pr[V_1(f(x, \sigma'), \sigma, P_1(f(x, \sigma'), \sigma)) = 1] \geq 1 - \frac{1}{n^{\omega(1)}}$$

- Soundness: Suppose x is NO instance of Π. Then with probability $1 - \frac{1}{n^{\omega(1)}}$ (over randomness of σ'), we have that $f(x, \sigma')$ is a NO instance of $\mathsf{EA}_{\mathsf{NC}^0}$. Thus for a randomly chosen σ:

$$\Pr[V_1(f(x, \sigma'), \sigma, P_1(f(x, \sigma'), \sigma)) = 0] \geq 1 - \frac{1}{n^{\omega(1)}}$$

- Statistical Zero-Knowledge: If x is a YES instance, $f(x, \sigma')$ is a YES instance of $\mathsf{EA}_{\mathsf{NC}^0}$ with probability close to 1. For any YES instance y of $\mathsf{EA}_{\mathsf{NC}^0}$, the distribution given by S_1 on input y is exponentially close the the distribution on transcripts (σ, p) induced by (V_1, P_1) on input y. Thus the distribution on $(\sigma\sigma', p)$ induced by (V, P) has distance at most $\frac{1}{n^{\omega(1)}}$ from the distribution produced by S on input x. The claim now follows by the comments regarding error probabilities in Definition 4.

◀

McKenzie and Cook [23] defined and studied the problems LCON, LCONX and $\mathsf{LCONNULL}$. LCON is the problem of determining if a system of linear congruences over the integers mod q has a solution. LCONX is the problem of finding a solution, if one exists, and $\mathsf{LCONNULL}$ is the problem of computing a spanning set for the null space of the system.

These problems are known to lie in uniform NC^3 [23], but are not known to lie in uniform NC^2, although Arvind and Vijayaraghavan showed that there is a set B in $\mathsf{L}^{\mathsf{GapL}} \subseteq \mathsf{DET} \subseteq \mathsf{NC}^2$ such that $x \in \mathsf{LCON}$ if and only if $(x, W) \in B$, where W is a randomly-chosen weight function [10]. (The probability of error is exponentially small.) The mapping $x \mapsto (x, W)$ is clearly a $\leq_{\mathrm{m}}^{\mathsf{BPL}}$ reduction. Since $\mathsf{DET} \subseteq \mathsf{NISZK_L}$ [2], it follows that

$$\mathsf{LCON} \in \mathsf{NISZK_L}$$

The arguments in [10] carry over to LCONX and $\mathsf{LCONNULL}$ as well.

▶ **Corollary 28.** $\mathsf{LCON} \in \mathsf{NISZK_L}$. $\mathsf{LCONX} \in \mathsf{NISZK_L}$. $\mathsf{LCONNULL} \in \mathsf{NISZK_L}$.

6 Varying the Power of the Verifier

In this section, we show that the computational complexity of the simulator is more important than the computational complexity of the verifier, in non-interactive protocols. The results in this section were motivated by our attempts to show that $\mathsf{NISZK_L} = \mathsf{NISZK_{DET}}$. Although we were unable to reach this goal, we were able to show that the verifier could be as powerful as DET, if the simulator was restricted to be no more powerful than NL. The general approach here is to replace a powerful verifier with a weaker verifier, by requiring the prover to provide a proof to convince a weak verifier that the more powerful verifier would accept.

We define $\mathsf{NISZK}_{A,B}$ as the class of problems with a NISZK protocol where the simulator is in A and the verifier is in B (and hence $\mathsf{NISZK}_A = \mathsf{NISZK}_{A,A}$).

▶ **Theorem 29.** *Let A and B be classes of functions that are closed under composition, where $A \subseteq B \subseteq \mathsf{NISZK}_A$. Then $\mathsf{NISZK}_{A,B} = \mathsf{NISZK}_A$.*

Proof. Let Π be an arbitrary promise problem in $\mathsf{NISZK}_{A,B}$ with (S_1, V_1, P_1) being the A simulator, B verifier, and prover for Π's proof system, where the reference string has length $p_1(|x|)$ and the prover's messages have length $q_1(|x|)$. Since $V_1 \in B \subseteq \mathsf{NISZK}_A$, $L(V_1)$ has a proof system (S_2, V_2, P_2), where the reference string has length $p_2(|x|)$ and the prover's messages have length $q_2(|x|)$.

▶ **Lemma 30.** *We may assume without loss of generality that $p_1(n) > p_2(n) + q_2(n)$.*

Proof. If it is not the case that $p_1(n) > p_2(n) + q_2(n)$, then let $r(n) = p_2(n) + q_2(n) - p_1(n)$. Consider a new proof system (S_1', V_1', P_1') that is identical to (S_1, V_1, P_1), except that the reference string now has length $p_1(n) + r(n)$ (where P_1' and V_1' ignore the additional $r(n)$ random bits). The simulator S_1' uses an additional $r(n)$ random bits and simply appends those bits to the output of S_1. The language $L(V_1')$ is still in NISZK_A, with a proof system (S_2', V_2', P_2') where the reference string still has length $p_2(n)$, since membership in $L(V_1')$ does not depend on the "new" $r(n)$ random bits, and hence S_2', V_2' and P_2', given input $(x, \sigma r, p)$ behave exactly as S_2, V_2 and P_2 behave when given input (x, σ, p). ◀

Then Π has the following NISZK_A proof system:

Algorithm 4 Simulator $S(x, r_1, r_2)$

Data: $x \in \Pi_{Yes} \cup \Pi_{No}$
$(\sigma, p) \leftarrow S_1(x, r_1)$;
$(\sigma', p') \leftarrow S_2((x, \sigma, p), r_2)$;
return $((\sigma, \sigma'), (p, p'))$;

Algorithm 5 Verifier $V(x, (\sigma, \sigma'), (p, p'))$

---

return $V_2((x, \sigma, p), \sigma', p')$

Algorithm 6 Prover $P(x, \sigma\sigma')$

Data: $x \in \Pi_{Yes} \cup \Pi_{No}, \sigma \in \{0,1\}^{p_1(|x|)}, \sigma' \in \{0,1\}^{p_2(|x|)}$
if $x \in \Pi_{Yes}$ **then**
  $p \leftarrow P_1(x, \sigma)$;
  $p' \leftarrow P_2((x, \sigma, p), \sigma')$;
  return (p, p');
else
  return $\perp, \perp$;
end

- Correctness: Suppose $x \in \Pi_{Yes}$, then given random σ, with probability $(1 - \frac{1}{2^{O(|x|)}})$, we have that $(x, \sigma, P_1(x, \sigma)) \in L(V_1)$, which means with probability $(1 - \frac{1}{2^{O(|x|+p_1(|x|)+|p|)}})$ it holds that $((x, \sigma, p), \sigma', P_2(x, \sigma, P_1(x, \sigma)) \in L(V_2)$. So the probability that V accepts is at least:

$$(1 - \frac{1}{2^{O(|x|)}})(1 - \frac{1}{2^{O(|x|+p_1(|x|)+q_1(|x|))}}) = 1 - \frac{1}{2^{O(|x|)}}$$

- Soundness: Suppose $x \in \Pi_N$. When given a random σ, we have that with probability less than $\frac{1}{2^{O(|x|)}}$: $\exists p$ such that $(x, \sigma, p) \in L(V_1)$. For $(x, \sigma, p) \notin L(V_1)$, the probability that there is a p such that $((x, \sigma, p), \sigma', p') \in L(V_2)$ is at most $\frac{1}{2^{O(|x|+p_1(|x|)+|p|)}}$ (given random σ'). So the probability that V rejects is at least:

$$(1 - \frac{1}{2^{O(|x|)}})(1 - \frac{1}{2^{O(|x|+p(|x|)+|p|)}}) = 1 - \frac{1}{2^{O(|x|)}}$$

- Statistical Zero-Knowledge: Let P_1^* denote the distribution that samples σ and outputs $(\sigma, P_1(x, \sigma))$. Similarly, let $P_2^*(\sigma, p)$ denote the distribution that samples σ' and outputs $(\sigma\sigma', P_2((x, \sigma, p), \sigma')$. P^* will be defined as the distribution $((\sigma\sigma'), P(x, \sigma, \sigma')))$ where σ and σ' are chosen uniformly at random. In the same way, let S^* refer to the distribution produced by S on input x, let S_1^* refer to the distribution produced by $S_1(x)$, and let $S_2^*(\sigma, p)$ be the distribution induced by S_2 on input (x, σ, p). Now we can partition the set of possible outcomes $((\sigma, \sigma'), (p, p'))$ of S^* and P^* into 3 blocks:

1. $((\sigma, \sigma'), (p, p'))$ such that $V_1(x, \sigma, p)$ accepts and $V_2((x, \sigma, p), \sigma', p')$ accepts.
2. $((\sigma, \sigma'), (p, p'))$ such that $V_1(x, \sigma, p)$ accepts and $V_2((x, \sigma, p), \sigma', p')$ rejects.
3. $((\sigma, \sigma'), (p, p'))$ such that $V_1(x, \sigma, p)$ rejects.

We will call these blocks A_1, A_2, and A_3 respectively. Then by definition:

$$\begin{aligned}\Delta(S^*, P^*) &= \frac{1}{2} \sum_{j \in \{1,2,3\}} \sum_{y \in A_j} \big| \Pr_{S^*}[y] - \Pr_{P^*}[y] \big| \\ &= \frac{1}{2} \sum_{y \in A_1} \big| \Pr_{S^*}[y] - \Pr_{P^*}[y] \big| + \frac{1}{2} \sum_{j \in \{2,3\}} \sum_{y \in A_j} \big[\Pr_{S^*}[y] + \Pr_{P^*}[y] \big]\end{aligned}$$

We concentrate first on A_1.

$$\sum_{y \in A_1} \big| \Pr_{S^*}[y] - \Pr_{P^*}[y] \big|$$

$$= \sum_{(\sigma', p')} \left(\sum_{\{(\sigma,p): y=((\sigma,\sigma'),(p,p')) \in A_1\}} \big| \Pr_{S^*}[y|\sigma', p'] \Pr_{S^*}[(\sigma', p')] - \Pr_{P^*}[y|\sigma', p'] \Pr_{P^*}[(\sigma', p')] \big| \right) \quad (*)$$

Here

$$\Pr_{S^*}[(\sigma', p')] = \sum_{(\sigma,p)} \Pr_{S^*}[((\sigma, \sigma'), (p, p'))]$$

and

$$\Pr_{P^*}[(\sigma', p')] = \sum_{(\sigma,p)} \Pr_{P*}[((\sigma, \sigma'), (p, p'))].$$

We define $\delta(\sigma',p') := \left|\Pr_{S^*}[(\sigma',p')] - \Pr_{P^*}[(\sigma',p')]\right|$. Let us examine a single term of the sum $(*)$, for $y = ((\sigma,\sigma'),(p,p'))$:

$$
\begin{aligned}
& \left|\Pr_{S^*}[y|\sigma',p']\Pr_{S^*}[(\sigma',p')] - \Pr_{P^*}[y|\sigma',p']\Pr_{P^*}[(\sigma',p')]\right| \\
&= \left|(\Pr_{S^*}[y|\sigma',p']\Pr_{S^*}[(\sigma',p')] - \Pr_{P^*}[y|\sigma',p']\Pr_{S^*}[(\sigma',p')]) + \right. \\
&\quad \left.(\Pr_{P^*}[y|\sigma',p']\Pr_{S^*}[(\sigma',p')] - \Pr_{P^*}[y|\sigma',p']\Pr_{P^*}[(\sigma',p')])\right| \\
&= \left|(\Pr_{S_1^*}[(\sigma,p)] - \Pr_{P_1^*}[(\sigma,p))\Pr_{S^*}[(\sigma',p')] + \Pr_{P_1^*}[(\sigma,p)](\Pr_{S^*}[(\sigma',p')] - \Pr_{P^*}[(\sigma',p')])\right| \\
&\le \left|\Pr_{S_1^*}[(\sigma,p)] - \Pr_{P_1^*}[(\sigma,p)]\right|\Pr_{S^*}[(\sigma',p')] + \Pr_{P_1^*}[(\sigma,p)]\left|\Pr_{S^*}[(\sigma',p')] - \Pr_{P^*}[(\sigma',p')]\right| \\
&= \left|\Pr_{S_1^*}[(\sigma,p)] - \Pr_{P_1^*}[(\sigma,p)]\right|\Pr_{S^*}[(\sigma',p')] + \Pr_{P_1^*}[(\sigma,p)]\delta(\sigma',p')
\end{aligned}
$$

Thus (*) is no more than

$$
\begin{aligned}
& \sum_{(\sigma',p')}\sum_{(\sigma,p)}\left|\Pr_{S_1^*}[(\sigma,p)] - \Pr_{P_1^*}[(\sigma,p)]\right|\Pr_{S^*}[(\sigma',p')] \\
&\quad + \sum_{(\sigma',p')}\ \sum_{\{(\sigma,p):y=((\sigma,\sigma'),(p,p'))\in A_1\}}\Pr_{P_1^*}[(\sigma,p)]\delta(\sigma',p') \\
&\le \sum_{(\sigma,p)}\left|\Pr_{S_1^*}[(\sigma,p)] - \Pr_{P_1^*}[(\sigma,p)]\right| + \sum_{\{(\sigma',p'):\exists(\sigma,p)\ ((\sigma,\sigma'),(p,p'))\in A_1\}}\delta(\sigma',p') \\
&= 2\Delta(S_1^*(x),P_1^*(x)) + \sum_{\{(\sigma',p'):\exists(\sigma,p)\ ((\sigma,\sigma'),(p,p'))\in A_1\}}\delta(\sigma',p') \\
&\le \frac{2}{2^{|x|}} + \sum_{\{(\sigma',p'):\exists(\sigma,p)\ ((\sigma,\sigma'),(p,p'))\in A_1\}}\delta(\sigma',p') \quad (**)
\end{aligned}
$$

Let us consider a single term $\delta(\sigma',p')$ in the summation in $(**)$. Recalling that the probability that $S(x) = ((\sigma,\sigma'),(p,p'))$ is equal to the probability that $S_1(x) = (\sigma,p)$ and $S_2(x,\sigma,p) = (\sigma',p')$, we have

$$
\begin{aligned}
\Pr_{S^*}[(\sigma',p')] &= \sum_{(\sigma,p)}\Pr_{S^*}[((\sigma,\sigma'),(p,p'))] \\
&= \sum_{(\sigma,p)}\Pr_{S^*}[((\sigma,\sigma'),(p,p'))|(\sigma,p)]\Pr_{S^*}[(\sigma,p)] \\
&= \sum_{(\sigma,p)}\Pr_{S_2^*(\sigma,p)}[(\sigma'p')]\Pr_{S_1^*}[(\sigma,p)]
\end{aligned}
$$

and similarly $\Pr_{P^*}[(\sigma',p')] = \sum_{(\sigma,p)}\Pr_{P_2^*(\sigma,p)}[(\sigma'p')]\Pr_{P_1^*}[(\sigma,p)]$. Thus

$$\begin{aligned}
\delta(\sigma',p') &= \big|\Pr_{S^*}[\sigma',p'] - \Pr_{P^*}[\sigma',p']\big| \\
&= \big|\sum_{(\sigma,p)} \Pr_{S_2^*(\sigma,p)}[(\sigma',p')]\Pr_{S_1^*}[(\sigma,p)] - \sum_{(\sigma,p)} \Pr_{P_2^*(\sigma,p)}[(\sigma',p')]\Pr_{P_1^*}[\sigma,p]\big| \\
&= \big|\sum_{(\sigma,p)} \Pr_{S_2^*(\sigma,p)}[(\sigma',p')]\Pr_{S_1^*}[(\sigma,p)] - \sum_{(\sigma,p)} \Pr_{P_2^*(\sigma,p)}[(\sigma',p')]\Pr_{S_1^*}[(\sigma,p)] \\
&\qquad + \sum_{(\sigma,p)} \Pr_{P_2^*(\sigma,p)}[(\sigma',p')]\Pr_{S_1^*}[(\sigma,p)] - \sum_{(\sigma,p)} \Pr_{P_2^*(\sigma,p)}[(\sigma',p')]\Pr_{P_1^*}[(\sigma,p)]\big| \\
&= \big|\sum_{(\sigma,p)} (\Pr_{S_2^*(\sigma,p)}[(\sigma',p')] - \Pr_{P_2^*(\sigma,p)}[(\sigma',p')])\Pr_{S_1^*}[(\sigma,p)] \\
&\qquad + \sum_{(\sigma,p)} \Pr_{P_2^*(\sigma,p)}[(\sigma',p')](\Pr_{S_1^*}[(\sigma,p)] - \Pr_{P_1^*}[(\sigma,p)])\big| \\
&\leq \sum_{(\sigma,p)} \big|\Pr_{S_2^*(\sigma,p)}[(\sigma',p')] - \Pr_{P_2^*(\sigma,p)}[(\sigma',p')]\big|\Pr_{S_1^*}[(\sigma,p)] \\
&\qquad + \sum_{(\sigma,p)} \Pr_{P_2^*(\sigma,p)}[(\sigma',p')]\big|\Pr_{S_1^*}[(\sigma,p)] - \Pr_{P_1^*}[(\sigma,p)]\big| \\
&= \sum_{(\sigma,p)} 2\Delta(S_2^*(\sigma,p),P_2^*(\sigma,p))\Pr_{S_1^*}[(\sigma,p)] \\
&\qquad + \sum_{(\sigma,p)} \Pr_{P_2^*(\sigma,p)}[(\sigma',p')]\big|\Pr_{S_1^*}[(\sigma,p)] - \Pr_{P_1^*}[(\sigma,p)]\big| \\
&\leq \sum_{(\sigma,p)} \frac{2}{2^{|(x,\sigma,p)|}}\Pr_{S_1^*}[(\sigma,p)] + \sum_{(\sigma,p)} \Pr_{P_2^*(\sigma,p)}[(\sigma',p')]\big|\Pr_{S_1^*}[(\sigma,p)] - \Pr_{P_1^*}[(\sigma,p)]\big| \\
&= \frac{2}{2^{|x|+p_1(|x|)+q_1(|x|)}} + \sum_{(\sigma,p)} \Pr_{P_2^*(\sigma,p)}[(\sigma',p')]\big|\Pr_{S_1^*}[(\sigma,p)] - \Pr_{P_1^*}[(\sigma,p)]\big|
\end{aligned}$$

where the last inequality holds, since the summation in $(**)$ is taken over tuples, such that each (x,σ,p) is a YES instance of $L(V_1)$.
Replacing each term in $(**)$ with this upper bound, thus yields the following upper bound on $(*)$:

$$\begin{aligned}
&\frac{2}{2^{|x|}} + \sum_{(\sigma',p')}\left(\frac{2}{2^{|x|+p_1(|x|)+q_1(|x|)}} + \sum_{(\sigma,p)} \Pr_{P_2^*(\sigma,p)}[(\sigma',p')]\big|\Pr_{S_1^*}[(\sigma,p)] - \Pr_{P_1^*}[(\sigma,p)]\big|\right) \\
&= \frac{2}{2^{|x|}} + \frac{2\cdot 2^{p_2(|x|)+q_2(|x|)}}{2^{|x|+p_1(|x|)+q_1(|x|)}} + \sum_{(\sigma',p')}\sum_{(\sigma,p)} \Pr_{P_2^*(\sigma,p)}[(\sigma',p')]\big|\Pr_{S_1^*}[(\sigma,p)] - \Pr_{P_1^*}[(\sigma,p)]\big|) \\
&= \frac{2}{2^{|x|}} + \frac{2\cdot 2^{p_2(|x|)+q_2(|x|)}}{2^{|x|+p_1(|x|)+q_1(|x|)}} + 2\Delta(S_1^*,P_1^*) \\
&\leq \frac{2}{2^{|x|}} + \frac{2\cdot 2^{p_2(|x|)+q_2(|x|)}}{2^{|x|+p_1(|x|)+q_1(|x|)}} + \frac{2}{2^{|x|}} \\
&\leq \frac{2}{2^{|x|}} + \frac{2}{2^{|x|}} + \frac{2}{2^{|x|}}
\end{aligned}$$

where the last inequality follows from Lemma 30. Thus, A_1 contributes only a negligible quantity to $\Delta(S^*,P^*)$.

We now move on to consider A_2 and A_3.

$$\Pr_{P^*}[y \in A_2] = \sum_{\{(\sigma,p):(x,\sigma,p)\in L(V_1)\}} \Pr[V_2(x,\sigma,p) \text{ rejects}] \leq \sum_{(\sigma,p)} \frac{1}{2^{|x|+|\sigma|+|p|}} \leq \frac{1}{2^{|x|}}.$$

$$\Pr_{S^*}[y \in A_2] = \sum_{\{(\sigma,p):(x,\sigma,p)\in L(V_1)\}} (\Pr[V_2(x,\sigma,p) \text{ rejects}] + \Delta(S_2^*(\sigma,p), P_2^*(\sigma,p))) \leq \frac{2}{2^{|x|}}.$$

A similar and simpler calculation shows that $\Pr_{P^*}[y \in A_3] \leq \frac{1}{2^{|x|}}$ and $\Pr_{S^*}[y \in A_3] \leq \frac{2}{2^{|x|}}$, to complete the proof.

◀

▶ **Corollary 31.** $\mathsf{NISZK_L} = \mathsf{NISZK_{AC^0}} = \mathsf{NISZK_{AC^0,DET}} = \mathsf{NISZK_{NL,DET}}$

Proof. DET contains AC^0 and is contained in $\mathsf{NISZK_L}$. By Theorem 11, $\mathsf{NISZK_L} = \mathsf{NISZK_{AC^0}}$, and thus by Theorem 29 $\mathsf{NISZK_{AC^0,DET}} = \mathsf{NISZK_{AC^0}}$. Also, since $\mathsf{AC}^0 \subseteq \mathsf{NL} \subseteq \mathsf{PM}$ and $\mathsf{NISZK_L} = \mathsf{NISZK_{PM}}$ (by Theorem 25), it follows that $\mathsf{NISZK_{NL}} \subseteq \mathsf{NISZK_{PM}} = \mathsf{NISZK_{AC^0}} = \mathsf{NISZK_{NL}}$. Thus, again by Theorem 29, $\mathsf{NISZK_{NL,DET}} = \mathsf{NISZK_{NL}} = \mathsf{NISZK_L}$. ◀

The proof of Theorem 29 did not make use of the condition that the verifier is at least as powerful as the simulator. Thus, maintaining the condition that $A \subseteq B \subseteq \mathsf{NISZK}_A$, we also have the following corollaries:

▶ **Corollary 32.** $\mathsf{NISZK}_B = \mathsf{NISZK}_{B,A}$

▶ **Corollary 33.** $\mathsf{NISZK}_{A,B} \subseteq \mathsf{NISZK}_{B,A}$

▶ **Corollary 34.** $\mathsf{NISZK_{DET}} = \mathsf{NISZK_{DET,AC^0}}$

7 $\mathsf{SZK_L}$ closure under $\leq^{\mathsf{L}}_{\mathrm{bf-tt}}$ reductions

Although our focus in this paper has been on $\mathsf{NISZK_L}$, in this section we report on a closure property of the closely-related class $\mathsf{SZK_L}$.

The authors of [15], after defining the class $\mathsf{SZK_L}$, wrote:

> We also mention that all the known closure and equivalence properties of SZK (e.g. closure under complement [25], equivalence between honest and dishonest verifiers [18], and equivalence between public and private coins [25]) also hold for the class $\mathsf{SZK_L}$.

In this section, we consider a variant of a closure property of SZK (closure under $\leq^{\mathsf{P}}_{\mathrm{bf-tt}}$ [28]), and show that it also holds[10] for $\mathsf{SZK_L}$. Although our proof follows the general approach of the proof of [28, Theorem 4.9], there are some technicalities with showing that certain computations can be accomplished in logspace (and for dealing with distributions represented by branching programs instead of circuits) that require proof. (The characterization of $\mathsf{SZK_L}$ in terms of reducibility to the Kolmogorov-random strings presented in [5, Theorem 34] relies on this closure property.)

[10] We observe that open questions about closure properties of NISZK also translate to open questions about $\mathsf{NISZK_L}$. NISZK is not known to be closed under union [26], and neither is $\mathsf{NISZK_L}$. Neither is known to be closed under complementation. Both are closed under conjunctive logspace-truth-table reductions.

▶ **Definition 35.** *(From [28, Definition 4.7]) For a promise problem* Π*, the characteristic function of* Π *is the map* $\mathcal{X}_\Pi : \{0,1\}^* \to \{0,1,*\}$ *given by*

$$\mathcal{X}_\Pi(x) = \begin{cases} 1 & \text{if } x \in \Pi_{Yes}, \\ 0 & \text{if } x \in \Pi_{No}, \\ * & \text{otherwise.} \end{cases}$$

▶ **Definition 36.** *Logspace Boolean formula truth-table reduction* ($\leq^{\mathsf{L}}_{\text{bf-tt}}$ *reduction): We say a promise problem* Π ***logspace Boolean formula truth-table reduces*** *to* Γ *if there exists a logspace-computable function* f*, which on input* x *produces a tuple* $(y_1, \ldots, y_m)$ *and a Boolean formula* ϕ *(with* m *input gates) such that:*

$$x \in \Pi_{Yes} \implies \phi(\mathcal{X}_\Gamma(y_1), \ldots, \mathcal{X}_\Gamma(y_m)) = 1$$

$$x \in \Pi_{No} \implies \phi(\mathcal{X}_\Gamma(y_1), \ldots, \mathcal{X}_\Gamma(y_m)) = 0$$

We begin by proving a logspace analogue of a result from [28], used to make statistically close pairs of distributions closer and statistically far pairs of distributions farther.

▶ **Lemma 37.** *(Polarization Lemma, adapted from [28, Lemma 3.3]) There is a logspace-computable function that takes a triple* $(P_1, P_2, 1^k)$*, where* P_1 *and* P_2 *are branching programs, and outputs a pair of branching programs* (Q_1, Q_2) *such that:*

$$\Delta(P_1, P_2) < \frac{1}{3} \implies \Delta(Q_1, Q_2) < 2^{-k}$$

$$\Delta(P_1, P_2) > \frac{2}{3} \implies \Delta(Q_1, Q_2) > 1 - 2^{-k}$$

To prove this, we adapt the same method as in [28] and alternate two different procedures, one to drive pairs with large statistical distance closer to 1, and one to drive distributions with small statistical distance closer to 0. The following lemma will do the former:

▶ **Lemma 38.** *(Direct Product Lemma, from [28, Lemma 3.4]) Let* X *and* Y *be distributions such that* $\Delta(X, Y) = \epsilon$*. Then for all* k*,*

$$k\epsilon \geq \Delta(\otimes^k X, \otimes^k Y) \geq 1 - 2\exp(-k\epsilon^2/2)$$

The proof of this statement follows from [28]. To use this for Lemma 37, we note that a branching program for $\otimes^k P$ can easily be created in logspace from a branching program P by simply copying and concatenating k independent copies of P together.

We now introduce a lemma to push close distributions closer:

▶ **Lemma 39.** *(XOR Lemma, adapted from [28, Lemma 3.5]) There is a logspace-computable function that maps a triple* $(P_0, P_1, 1^k)$*, where* P_0 *and* P_1 *are branching programs, to a pair of branching programs* (Q_0, Q_1) *such that* $\Delta(Q_0, Q_1) = \Delta(P_0, P_1)^k$*. Specifically,* Q_0 *and* Q_1 *are defined as follows:*

$$Q_0 = \bigotimes_{i \in [k]} P_{y_i} : y \sim \{y \in \{0,1\}^k : \oplus_{i \in [k]} y_i = 0\}$$

$$Q_1 = \bigotimes_{i \in [k]} P_{y_i} : y \sim \{y \in \{0,1\}^k : \oplus_{i \in [k]} y_i = 1\}$$

Proof. The proof that $\Delta(Q_0, Q_1) = \Delta(P_0, P_1)^k$ follows from [28, Proposition 3.6]. To finish proving this lemma, we show a logspace-computable mapping between $(P_0, P_1, 1^k)$ and (Q_0, Q_1).

Let ℓ and w be the max length and width between P_0 and P_1. We describe the structure of Q_0, with Q_1 differing in a small step: to begin with, Q_0 reads the $k-1$ random bits $y_1, \ldots, y_{k-1}$. For each of the random bits, it can pick the correct of two different branches, one having P_0 built in at the end and the other having P_1. We will read y_1, branch to P_0 or P_1 (and output the distribution accordingly), then unconditionally branch to reading y_2 and repeat until we reach y_{k-1} and branch to P_0 or P_1. We then unconditionally branch to y_1 and start computing the parity, and at the end we will be able to decide the value of y_k which will allow us to branch to the final copy of P_0 or P_1.

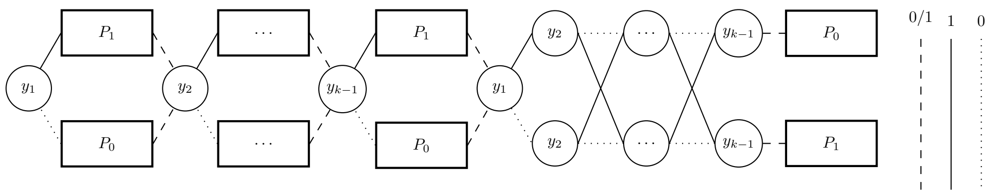

Figure 1 Branching program for Q_0 of Lemma 39

Creating (Q_0, Q_1) can be done in logspace, requiring logspace to create the section to compute y_k and logspace to copy the independent copies of P_0 and P_1.

◀

We now have the tools to prove Lemma 37.

Proof. (of Lemma 37) From [28, Section 3.2], we know that we can polarize $(P_0, P_1, 1^k)$ by:
- Letting $l = \lceil \log_{4/3} 6k \rceil$, $j = 3^{l-1}$
- Applying Lemma 39 to $(P_0, P_1, 1^l)$ to get (P'_0, P'_1)
- Applying Lemma 38: $P''_0 = \otimes^j P'_0$, $P''_1 = \otimes^j P'_1$
- Applying Lemma 39 to $(P''_0, P''_1, 1^k)$ to get (Q_0, Q_1)

Each step is computable in logspace, and since logspace is closed under composition, this completes our proof. ◀

We also mention the following lemma, which will be useful in evaluating the Boolean formula given by the $\leq^{\mathsf{L}}_{\mathrm{bf-tt}}$ reduction.

▶ **Lemma 40.** *There is a function in* NC^1 *that takes as input a Boolean formula* ϕ *(with* m *input bits) and produces as output an equivalent formula* ψ *with the following properties:*
1. *The depth of* ψ *is* $O(\log m)$*.*
2. ψ *is a tree with alternating levels of AND and OR gates.*
3. *The tree's non-leaf structure is always the same for a fixed input length, and is a complete binary tree.*
4. *All NOT gates are located just before the leaves.*

Proof. Although this lemma does not seem to have appeared explicitly in the literature, it is known to researchers, and is closely related to results in [16] (see Theorems 5.6 and 6.3, and Lemma 3.3) and in [6] (see Lemma 5).

The Boolean formula that is given as input may be encoded in the usual infix notation over the alphabet $\{0, 1, x,), (\}$, where leaf nodes connected to variable x_i are expressed by the string (xb) (where the string b is the binary representation of the number i), and where leaf nodes connected to the constants 0 and 1 are expressed by the strings (0) and (1), respectively, and more complicated expressions can be built from formulae α and β as $(\alpha \vee \beta)$, $(\alpha \wedge \beta)$, and $(\neg\alpha)$. Since the formula produced as output has a very restricted form (with an AND gate at the root, and alternating layers of AND and OR gates forming a full binary tree) the output formula can simply be encoded as a list of 2^d leaf nodes. Thus $0, \neg x10, x11, 1$ would be a representation of the formula $(((0) \vee (\neg(x_2))) \wedge ((x_3) \vee (1)))$.

The lemma is proved by using the fact that the Boolean formula evaluation problem lies in NC^1 [11, 12], and thus there is an alternating Turing machine M running in $O(\log n)$ time that takes as input a Boolean formula ψ and an assignment α to the variables of ψ, and returns $\psi(\alpha)$. We may assume without loss of generality that M alternates between existential and universal states at each step, and that M runs for exactly $c \log n$ steps on each path (for some constant c), and that M accesses its input (via the address tape that is part of the alternating Turing machine model) only at a halting step, and that M records the sequence of states that it has visited along the current path in the current configuration. Thus the configuration graph of M, on inputs of length n, corresponds to a formula of $O(\log n)$ depth having the desired structure, and this formula can be constructed in NC^1. Given a formula ϕ, an NC^1 machine can thus build this formula, and hardwire in the bits that correspond to the description of ϕ, and identify the remaining input variables (corresponding to M reading the bits of α) with the variables of ϕ. The resulting formula is equivalent to ϕ and satisfies the conditions of the lemma. ◀

▶ **Definition 41.** *(From [28, Definition 4.8]) For a promise problem* Π*, we define a new promise problem* $\Phi(\Pi)$ *as follows:*

$$\Phi(\Pi)_{Yes} = \{(\phi, x_1, \ldots, x_m) : \phi(\mathcal{X}_\Pi(x_1), \ldots, \mathcal{X}_\Pi(x_m)) = 1\}$$

$$\Phi(\Pi)_{No} = \{(\phi, x_1, \ldots, x_m) : \phi(\mathcal{X}_\Pi(x_1), \ldots, \mathcal{X}_\Pi(x_m)) = 0\}$$

▶ **Theorem 42.** $\mathsf{SZK_L}$ *is closed under* $\leq^{\mathsf{L}}_{\mathrm{bf-tt}}$ *reductions.*

To begin the proof of this theorem, we first note that as in the proof of [28, Lemma 4.10], given two $\mathsf{SD_{BP}}$ pairs, we can create a new pair which is in $\mathsf{SD}_{\mathsf{BP},No}$ if both of the original two pairs are (which we will use to compute ANDs of queries.) We can also compute in logspace the OR query for two queries by creating a pair $(P_1 \otimes S_1, P_2 \otimes S_2)$. We prove that these operations produce an output with the correct statistical difference with the following two claims:

▷ Claim 43. $\{(y_1, y_2) | \mathcal{X}_{\mathsf{SD_{BP}}}(y_1) \vee \mathcal{X}_{\mathsf{SD_{BP}}}(y_2) = 1\} \leq^{\mathsf{L}}_{\mathrm{m}} \mathsf{SD_{BP}}$.

Proof. Let $y_1 = (A_1, B_1)$ and $y_2 = (A_2, B_2)$. Let $p > 0$ be a parameter, where we are guaranteed that:

$$(A_i, B_i) \in \mathsf{SD}_{\mathsf{BP},Y} \implies \Delta(A_i, B_i) > 1 - p$$

$$(A_i, B_i) \in \mathsf{SD}_{\mathsf{BP},N} \implies \Delta(A_i, B_i) < p$$

Then consider:

$$y = (A_1 \otimes A_2, B_1 \otimes B_2)$$

Let us analyze the Yes and No instance of $\mathcal{X}_{\mathsf{SD_{BP}}}(y_1) \vee \mathcal{X}_{\mathsf{SD_{BP}}}(y_2)$:

- YES: $\Delta(A_1 \otimes A_2, B_1 \otimes B_2) \geq \max\{\Delta(A_1 \otimes B_2, B_1 \otimes B_2), \Delta(B_1 \otimes A_2, B_1 \otimes B_2)\} = \max\{\Delta(A_1, B_1), \Delta(A_2, B_2)\} > 1 - p$.
- NO[11]: $\Delta(A_1 \otimes A_2, B_1 \otimes B_2) \leq \Delta(A_1, B_1) + \Delta(A_2, B_2) < 2p$.

◀

In our Boolean formula, we will have only $d = O(\log m)$ depth, so we have this OR operation for at most $\frac{d+1}{2}$ levels (and the soundness gap doubles at every level). Since $p = \frac{1}{2^m}$ at the beginning, the gap (for NO instance) will be upper bounded at the end by:

$$< 2^{\frac{d+1}{2}} \frac{1}{2^m} = \frac{m^{O(1)}}{2^m} < 1/3.$$

▷ Claim 44. $\{(y_1, y_2) | \mathcal{X}_{\mathsf{SD_{BP}}}(y_1) \wedge \mathcal{X}_{\mathsf{SD_{BP}}}(y_2) = 1\} \leq^{\mathsf{L}}_{\mathrm{m}} \mathsf{SD_{BP}}$.

Proof. Let $y_1 = (A_1, B_1)$ and $y_2 = (A_2, B_2)$. Let $p > 0$ be a parameter, where we are guaranteed that:

$$(A_i, B_i) \in \mathsf{SD}_{\mathsf{BP},Y} \implies \Delta(A_i, B_i) > 1 - p$$

$$(A_i, B_i) \in \mathsf{SD}_{\mathsf{BP},N} \implies \Delta(A_i, B_i) < p$$

We can construct a pair of BPs $y = (A, B)$ whose statistical difference is exactly

$$\Delta(A_1, B_1) \cdot \Delta(A_2, B_2)$$

The pair (A, B) we construct is analogous to (Q_0, Q_1) in Lemma 39, and can be created in logspace with 2 random bits b_0, b_1. We have $A = (A_1, A_2)$ if $b_0 = 0$ and $A = (B_1, B_2)$ if $b_0 = 1$, while $B = (A_1, B_2)$ if b_2 is 0 and (A_2, B_1) if $b_1 = 1$.

Let us analyze the Yes and No instance of $\mathcal{X}_{\mathsf{SD_{BP}}}(y_1) \wedge \mathcal{X}_{\mathsf{SD_{BP}}}(y_2)$:

- YES: $\Delta(A_1, B_1) \cdot \Delta(A_2, B_2) > (1 - p)^2$.
- NO: $\Delta(A_1, B_1) \cdot \Delta(A_2, B_2) \leq \min\{\Delta(A_1, B_1), \Delta(A_2, B_2)\} < p$.

◀

In our Boolean formula we will have only $d = O(\log m)$ depth, so we have this AND operation for at most $\frac{d+1}{2}$ levels (and the completeness gap squares itself at every level). Since $p = \frac{1}{2^m}$ at the beginning, the gap (for YES instance) will be lower bounded at the end by:

$$> (1 - \frac{1}{2^m})^{2^{\frac{d+1}{2}}} = (1 - \frac{1}{2^m})^{m^{O(1)}} > (1 - \frac{1}{2^m})^{2^m/m} \approx (\frac{1}{e})^{1/m} > \frac{2}{3}.$$

Proof. (of Theorem 42) Now suppose that we are given a promise problem Π such that $\Pi \leq^{\mathsf{L}}_{\mathrm{bf-tt}} \mathsf{SD_{BP}}$. We want to show $\Pi \leq^{\mathsf{L}}_{\mathrm{m}} \mathsf{SD_{BP}}$, which by $\mathsf{SZK_L}$'s closure under $\leq^{\mathsf{L}}_{\mathrm{m}}$ reductions implies $\Pi \in \mathsf{SZK_L}$.

We follow the steps below on input x to create an $\mathsf{SD_{BP}}$ instance (F_0, F_1) which is in $\mathsf{SD}_{\mathsf{BP},Y}$ if $x \in \Pi_Y$, and is in $\mathsf{SD}_{\mathsf{BP},N}$ if $x \in \Pi_N$:

1. Run the L machine for the $\leq^{\mathsf{L}}_{\mathrm{bf-tt}}$ reduction on x to get queries $(q_1, \ldots, q_m)$ and the formula ϕ.

[11] For the first inequality here, see [28, Fact 2.3].

2. Build ψ from ϕ using Lemma 40. Recalling that there is a $\leq_m^L$ reduction f reducing $\mathsf{SD_{BP}}$ to its complement, replace each negated query $\neg q_i$ with $f(q_i)$, so that we can now view ψ as a *monotone* Boolean formula reducing Π to $\mathsf{SD_{BP}}$. Since the Polarization Lemma (Lemma 37) maps YES instances to YES instances and NO instances to NO instances, we can also use the same formula ψ on the polarized instances that we obtain by applying Lemma 37 with $k = n$ to these queries, to obtain a new list of queries $(y_1, \ldots, y_m)$. Furthermore we may pad these queries, so that each query y_i consists of a pair of branching programs (instances of $\mathsf{SD_{BP}}$) where all of the branching programs have the same number of output bits.
3. Using the formula ψ, build a "template tree" T. At the leaf level, for each variable in ψ, we will plug in the corresponding query y_i; interior nodes are labeled AND or OR. By Lemma 40 the tree T is full. Using Claims 43 and 44, each node of the template tree is associated with a pair of branching programs, with the pair (F_0, F_1) at the root being the output of our $\leq_m^L$ reduction. It is important to note that the constructions in Claims 43 and 44 produce distributions, where each output bit is simply a copy of one of the output bits of the distributions that feed into it. Thus each output bit of F_0 and F_1 is simply a copy of one of the output bits of one of the pairs of branching programs that constitute one of the input queries y_i.
4. Given x and designated output position j of F_0 or F_1, there is a logspace computation which finds the original output bit from $y_1 \ldots y_m$ that bit j was copied from. This machine traverses down the template tree from the output bit and records the following:
   - The node that the computation is currently at on the template tree, with the path taken depending on j.
   - The position of the random bits used to decide which path to take when we reach nodes corresponding to AND.

   This takes $O(\log m)$ space. We can use this algorithm to copy and compute each output bit of F_0 and F_1, creating (F_0, F_1) in logspace.

For step 4, we give an algorithm $\mathsf{Eval}(x, j, \psi, y_1, \ldots, y_m)$ to compute the jth output bit of F_0 or F_1 on x, for a formula ψ satisfying the properties of Lemma 40, a list of $\mathsf{SD_{BP}}$ queries $(y_1, \ldots, y_m)$, and j. Without loss of generality, we lay out the algorithm to compute only $F_0(x)$.

Outline of $\mathsf{Eval}(x, j, \psi, y_1, \ldots, y_m)$:

The idea is to compute the jth output bit of F_0 by recursively calculating which query output bit it was copied from. To do this, first notice that the AND and OR operations produce branching programs where each output bit is copied from exactly one output bit of one of the query branching programs, so composing these operations together tells us that every output bit in F_0 is copied from exactly one output bit from one query. By Lemma 40 and our AND and OR operations preserving the number of output bits, we also have that if every BP has l output bits, F_0 will have $2^a l = |\psi| l$ output bits, where a is the depth of ψ. This can be used to recursively calculate which query the jth bit is from: for an OR gate, divide the output bits into fourths, and decide which fourth the jth bit falls into (with each fourth corresponding to one BP, or two fourths corresponding to a subtree.) For an AND gate, divide the output into fourths, decide which fourth the jth bit falls into, and then use the 4 random bits for the XOR operation to compute which fourth corresponds to which branching programs (2 fourths will correspond to 1 BP or subtree, and the other 2 fourths will correspond to the 2 BPs from the other subtree.) If j is updated recursively, then at the query level, we can directly return the j'th output bit. This can be done in logspace, requiring a logspace path of "lefts" and "rights" to track the current gate, logspace to record and update j', logspace to compute $2^a l$ at each level, and logspace to compute which subtree/query the output bit comes from at each level.

The resulting BP will be two distributions that will be in $\mathsf{SD}_{\mathsf{BP},Y} \iff x \in \Pi_Y$. By this process $\Pi \leq^{\mathsf{L}}_{\mathrm{m}} \mathsf{SD}_{\mathsf{BP}}$. ◀

8 Open Questions

The main open question is whether NISZK is equal to $\mathsf{NISZK}_{\mathsf{L}}$. Partial progress on this problem can be achieved by finding additional subclasses of P that lie in $\mathsf{NISZK}_{\mathsf{L}}$ (extending the work presented in Section 5).

On a more concrete level, can the results of Section 6 be improved, in order to show that $\mathsf{NISZK}_{\mathsf{L}} = \mathsf{NISZK}_{\mathsf{DET}}$? Or, more ambitiously, given the role that randomized encodings play in our results, is it possible that all problems in the class SREN (problems with statistical randomized encodings) lie in $\mathsf{NISZK}_{\mathsf{L}}$, or even (as suggested by the referees) that $\mathsf{NISZK}_{\mathsf{L}} = \mathsf{NISZK}_{\mathsf{SREN}}$?

The referees have also suggested that it would be interesting to consider classes defined in terms of non-uniform verifiers and simulators.

Acknowledgments

This work was done in part while EA and HT were visiting the Simons Institute for the Theory of Computing. This work was carried out while JG, SM, and PW were participants in the 2022 DIMACS REU program at Rutgers University. We thank Yuval Ishai for helpful conversations about SREN, and we thank Markus Lohrey, Sam Buss, and Dave Barrington for useful discussions about Lemma 40. We also thank the anonymous referees for helpful comments.

References

1 Eric Allender. Guest column: Parting thoughts and parting shots (read on for details on how to win valuable prizes! *SIGACT News*, 54(1):63–81, 2023. doi:10.1145/3586165.3586175.

2 Eric Allender, John Gouwar, Shuichi Hirahara, and Caleb Robelle. Cryptographic hardness under projections for time-bounded Kolmogorov complexity. *Theoretical Computer Science*, 940:206–224, 2023. doi:10.1016/j.tcs.2022.10.040.

3 Eric Allender, Jacob Gray, Saachi Mutreja, Harsha Tirumala, and Pengxiang Wang. Robustness for space-bounded statistical zero knowledge. In Nicole Megow and Adam Smith, editors, *Proc. International Workshop on Randomization and Computation (RANDOM 2023)*, volume 275 of *LIPIcs*, pages 56:1–56:21, Dagstuhl, Germany, 2023. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.APPROX/RANDOM.2023.56.

4 Eric Allender and Shuichi Hirahara. New insights on the (non-) hardness of circuit minimization and related problems. *ACM Transactions on Computation Theory (TOCT)*, 11(4):1–27, 2019.

5 Eric Allender, Shuichi Hirahara, and Harsha Tirumala. Kolmogorov complexity characterizes statistical zero knowledge. In *14th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 251 of *LIPIcs*, pages 3:1–3:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ITCS.2023.3.

6 Eric Allender and Ian Mertz. Complexity of regular functions. *Journal of Computer and System Sciences*, 104:5–16, 2019. Language and Automata Theory and Applications - LATA 2015. doi:https://doi.org/10.1016/j.jcss.2016.10.005.

7 Eric Allender and Mitsunori Ogihara. Relationships among PL, #L, and the determinant. *RAIRO Theor. Informatics Appl.*, 30(1):1–21, 1996. doi:10.1051/ita/1996300100011.

8 Eric Allender, Klaus Reinhardt, and Shiyu Zhou. Isolation, matching, and counting uniform and nonuniform upper bounds. *Journal of Computer and System Sciences*, 59(2):164–181, 1999. doi:https://doi.org/10.1006/jcss.1999.1646.

9 Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. *SIAM Journal on Computing*, 36(4):845–888, 2006. doi:10.1137/S0097539705446950.

10 V. Arvind and T. C. Vijayaraghavan. Classifying problems on linear congruences and abelian permutation groups using logspace counting classes. *computational complexity*, 19(1):57–98, November 2009. doi:10.1007/s00037-009-0280-6.

11 Samuel R. Buss. The Boolean formula value problem is in ALOGTIME. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 123–131. ACM, 1987. doi:10.1145/28395.28409.

12 Samuel R Buss. Algorithms for Boolean formula evaluation and for tree contraction. *Arithmetic, Proof Theory, and Computational Complexity*, 23:96–115, 1993.

13 Ronald Cramer, Serge Fehr, Yuval Ishai, and Eyal Kushilevitz. Efficient multi-party computation over rings. In *Proc. International Conference on the Theory and Applications of Cryptographic Techniques; Advances in Cryptology (EUROCRYPT)*, volume 2656 of *Lecture Notes in Computer Science*, pages 596–613. Springer, 2003. doi:10.1007/3-540-39200-9_37.

14 Alfredo De Santis, Giovanni Di Crescenzo, Giuseppe Persiano, and Moti Yung. Image density is complete for non-interactive-SZK (extended abstract). In *Proc. International Conference on Automata, Languages, and Programming (ICALP)*, volume 1443 of *Lecture Notes in Computer Science*, pages 784–795. Springer, 1998. This paper claims that NISZK is closed under complement, but this claim was later retracted. doi:10.1007/BFb0055102.

15 Zeev Dvir, Dan Gutfreund, Guy N Rothblum, and Salil P Vadhan. On approximating the entropy of polynomial mappings. In *Second Symposium on Innovations in Computer Science*, pages 460–475. Tsinghua University Press, 2011.

16 Moses Ganardi and Markus Lohrey. A universal tree balancing theorem. *ACM Transactions on Computation Theory*, 11(1):1:1–1:25, 2019. doi:10.1145/3278158.

17 Oded Goldreich, Amit Sahai, and Salil Vadhan. Can statistical zero knowledge be made non-interactive? or On the relationship of SZK and NISZK. In *Annual International Cryptology Conference*, pages 467–484. Springer, 1999. doi:10.1007/3-540-48405-1_30.

18 Oded Goldreich, Amit Sahai, and Salil P. Vadhan. Honest-verifier statistical zero-knowledge equals general statistical zero-knowledge. In *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 399–408. ACM, 1998. doi:10.1145/276698.276852.

19 Ulrich Hertrampf, Steffen Reith, and Heribert Vollmer. A note on closure properties of logspace MOD classes. *Information Processing Letters*, 75(3):91–93, 2000. doi:10.1016/S0020-0190(00)00091-0.

20 Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *Proc. International Conference on Automata, Languages, and Programming (ICALP)*, volume 2380 of *Lecture Notes in Computer Science*, pages 244–256. Springer, 2002. doi:10.1007/3-540-45465-9_22.

21 Richard M. Karp, Eli Upfal, and Avi Wigderson. Constructing a perfect matching is in random NC. *Combinatorica*, 6(1):35–48, 1986. doi:10.1007/BF02579407.

22 Nathan Linial, Yishay Mansour, and Noam Nisan. Constant depth circuits, Fourier transform, and learnability. *J. ACM*, 40(3):607–620, 1993. doi:10.1145/174130.174138.

23 Pierre McKenzie and Stephen A. Cook. The parallel complexity of Abelian permutation group problems. *SIAM Journal on Computing*, 16(5):880–909, 1987. doi:10.1137/0216058.

24 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 345–354. ACM, 1987. doi:10.1145/28395.383347.

25 Tatsuaki Okamoto. On relationships between statistical zero-knowledge proofs. *Journal of Computer and System Sciences*, 60(1):47–108, 2000. doi:10.1006/jcss.1999.1664.

26 Chris Peikert and Vinod Vaikuntanathan. Noninteractive statistical zero-knowledge proofs for lattice problems. In *Proc. Advances in Cryptology: 28th Annual International Cryptology Conference (CRYPTO)*, volume 5157 of *Lecture Notes in Computer Science*, pages 536–553. Springer, 2008. `doi:10.1007/978-3-540-85174-5\_30`.

27 Vishal Ramesh, Sasha Sami, and Noah Singer. Simple reductions to circuit minimization: DIMACS REU report. Technical report, DIMACS, Rutgers University, 2021. Internal document.

28 Amit Sahai and Salil P. Vadhan. A complete problem for statistical zero knowledge. *J. ACM*, 50(2):196–249, 2003. `doi:10.1145/636865.636868`.

29 Jacobo Torán. On the hardness of graph isomorphism. *SIAM Journal on Computing*, 33(5):1093–1108, 2004. `doi:10.1137/S009753970241096X`.

30 Heribert Vollmer. *Introduction to circuit complexity: a uniform approach*. Springer Science & Business Media, 1999. `doi:10.1007/978-3-662-03927-4`.