

¹ Robustness for Space-Bounded Statistical Zero Knowledge

³ **Eric Allender**  

⁴ Rutgers University, NJ, USA

⁵ **Jacob Gray**  

⁶ University of Massachusetts, MA, USA

⁷ **Saachi Mutreja** 

⁸ University of California, Berkeley, CA, USA

⁹ **Harsha Tirumala**  

¹⁰ Rutgers University, NJ, USA

¹¹ **Pengxiang Wang** 

¹² University of Michigan, MI, USA

¹³ **Abstract**

¹⁴ We show that the space-bounded Statistical Zero Knowledge classes SZK_L and NISZK_L are surprisingly
¹⁵ robust, in that the power of the verifier and simulator can be strengthened or weakened without
¹⁶ affecting the resulting class. Coupled with other recent characterizations of these classes [2], this
¹⁷ can be viewed as lending support to the conjecture that these classes may coincide with the
¹⁸ non-space-bounded classes SZK and NISZK , respectively.

¹⁹ **2012 ACM Subject Classification** Complexity Classes

²⁰ **Keywords and phrases** Interactive Proofs

²¹ **Funding** *Eric Allender*: Supported in part by NSF Grants CCF-1909216 and CCF-1909683.

²² *Jacob Gray*: Supported in part by NSF grants CNS-215018 and CCF-1852215

²³ *Saachi Mutreja*: Supported in part by NSF grants CNS-215018 and CCF-1852215

²⁴ *Harsha Tirumala*: Supported in part by NSF Grants CCF-1909216 and CCF-1909683.

²⁵ *Pengxiang Wang*: Supported in part by NSF grants CNS-215018 and CCF-1852215

26

1 Introduction

27 The complexity class SZK (Statistical Zero Knowledge) and its “non-interactive” subclass
 28 NISZK have been studied intensively by the research communities in cryptography and
 29 computational complexity theory. In [10], a space-bounded version of SZK , denoted SZK_L
 30 was introduced, primarily as a tool for understanding the complexity of estimating the
 31 entropy of distributions represented by very simple computational models (such as low-degree
 32 polynomials, and NC^0 circuits). There, it was shown that SZK_L contains many important
 33 problems previously known to lie in SZK , such as Graph Isomorphism, Discrete Log, and
 34 Decisional Diffie-Hellman. The corresponding “non-interactive” subclass of SZK_L , denoted
 35 NISZK_L , was subsequently introduced in [1], primarily as a tool for clarifying the complexity
 36 of computing time-bounded Kolmogorov complexity under very restrictive reducibilities (such
 37 as projections). Just as every problem in $\text{SZK} \leq_{tt}^{\text{AC}^0}$ reduces to problems in NISZK [12], so
 38 also every problem in $\text{SZK}_L \leq_{tt}^{\text{AC}^0}$ reduces to problems in NISZK_L , and thus NISZK_L contains
 39 intractable problems if and only if SZK_L does.

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

45 ▶ **Theorem 1.** [2] Let A be a decidable promise problem. Then

- 46 ■ $A \in \text{NISZK}$ if and only if A is reducible to \tilde{R}_K by randomized polynomial time reductions.
- 47 ■ $A \in \text{NISZK}_L$ if and only if A is reducible to \tilde{R}_K by randomized AC^0 or logspace reductions.
- 48 ■ $A \in \text{SZK}$ if and only if A is reducible to \tilde{R}_K by randomized polynomial time “Boolean
 49 formula” reductions.
- 50 ■ $A \in \text{SZK}_L$ if and only if A is reducible to \tilde{R}_K by randomized logspace “Boolean formula”
 51 reductions.

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

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

60 This motivates a closer examination of SZK_L and NISZK_L , to answer questions that have
 61 not been addressed by earlier work on these classes.

62 Our main results are:

- 63 1. **The verifier and simulator may be very weak.** NISZK_L and SZK_L are defined in
 64 terms of three algorithms: (1) A logspace-bounded *verifier*, who interacts with (2) a
 65 computationally-unbounded *prover*, following the usual rules of an interactive proof, and
 66 (3) a logspace-bounded *simulator*, who ensures the zero-knowledge aspects of the protocol.
 67 (More formal definitions are to be found in Section 2.) We show that the verifier and
 68 simulator can be restricted to lie in AC^0 . Let us explain why this is surprising.
 69 The proof presented in [1], showing that EA_{NC^0} is complete for NISZK_L , makes it clear
 70 that the verifier and simulator can be restricted to lie in $\text{AC}^0[\oplus]$ (as was observed in [22]).

71 But the proof in [1] (and a similar argument in [12]) relies heavily on hashing, and it is
 72 known that, although there are families of universal hash functions in $\text{AC}^0[\oplus]$, no such
 73 families lie in AC^0 [17]. We provide an alternative construction, which avoids hashing,
 74 and allows the verifier and simulator to be very weak indeed.

75 **2. The verifier and simulator may be somewhat stronger.** The proof presented in
 76 [1], showing that EA_{NC^0} is complete for NISZK_L , also makes it clear that the verifier and
 77 simulator can be as powerful as $\oplus L$, without leaving NISZK_L . This is because the proof
 78 relies on the fact that logspace computation lies in the complexity class PREN of functions
 79 that have *perfect randomized encodings* [5], and $\oplus L$ also lies in PREN. Applebaum,
 80 Ishai, and Kushilevitz defined PREN and the somewhat larger class SREN (for *statistical*
 81 *randomized encodings*), in proving that there are one-way functions in SREN if and only
 82 if there are one-way functions in NC^0 . They also showed that other important classes
 83 of functions, such as NL and GapL , are contained in SREN.¹ We initially suspected that
 84 NISZK_L could be characterized using verifiers and simulators computable in GapL (or
 85 even in the slightly larger class DET, consisting of problems that are $\leq_T^{\text{NC}^1}$ reducible to
 86 GapL), since DET is known to be contained in NISZK_L [1]. However, we were unable to
 87 reach that goal.

88 We were, however, able to show that the simulator and verifier can be as powerful as NL,
 89 without making use of the properties of SREN. In fact, we go further in that direction.
 90 We define the class PM, consisting of those problems that are \leq_T^L -reducible to the Perfect
 91 Matching problem. PM contains NL [16], and is not known to lie in (uniform) NC (and it
 92 is not known to be contained in SREN). We show that statistical zero knowledge protocols
 93 defined using simulators and verifiers that are computable in PM yield only problems in
 94 NISZK_L .

95 **3. The complexity of the simulator is key.** As part of our attempt to characterize
 96 NISZK_L using simulators and verifiers computable in DET, we considered varying the
 97 complexity of the simulator and the verifier separately. Among other things, we show
 98 that the verifier can be as complex as DET if the simulator is logspace-computable.
 99 In most cases of interest, the NISZK class defined with verifier and simulator lying in
 100 some complexity class remains unchanged if the rules are changed so that the verifier is
 101 significantly stronger or weaker.

102 We also establish some additional closure properties of NISZK_L and SZK_L , some of which are
 103 required for the characterizations given in [2].

104 The rest of the paper is organized as follows: Section 3 will show how NISZK_L can be
 105 defined equivalently using an AC^0 verifier and simulator. Section 4 will show that increasing
 106 the power of the verifier and simulator to lie in PM does not increase the size of NISZK_L
 107 (where PM is the class of problems (containing NL) that are logspace Turing reducible to
 108 Perfect Matching). Section 5 expands the list of problems known to lie in NISZK_L . McKenzie
 109 and Cook [18] studied different formulations of the problem of solving linear congruences.
 110 These problems are not known to lie in DET, which is the largest well-studied subclass of P
 111 known to be contained in NISZK_L . However, these problems are randomly logspace-reducible
 112 to DET [6]. We show that NISZK_L is closed under randomized logspace reductions, and
 113 hence show that these problems also reside in NISZK_L . Section 6 shows that the complexity
 114 of the simulator is more important than the complexity of the verifier, in non-interactive
 115 zero-knowledge protocols. In particular, the verifier can be as powerful as DET, while still

¹ This is not stated explicitly for GapL , but it follows from [15, Theorem 1]. See also [9, Section 4.2].

116 defining only problems in NISZK_L . Finally Section 7 will show that SZK_L is closed under
 117 logspace Boolean formula truth-table reductions.

118 **2 Preliminaries**

119 We assume familiarity with basic complexity classes L , NL , $\oplus L$ and P , and circuit complexity
 120 classes NC^0 and AC^0 . We assume knowledge of m-reducibility (many-one-reducibility) and
 121 Turing-reducibility. $\#L$ is the class of functions that count the number of accepting paths
 122 of NL machines, and $\text{GapL} = \{f - g : f, g \in \#L\}$. The determinant is complete for GapL ,
 123 and the complexity class DET is the class of languages NC^1 -Turing reducible to functions in
 124 GapL .

125 Many of the problems we consider deal with entropy (also known as Shannon entropy).
 126 The *entropy* of a distribution X (denoted $H(X)$) is the expected value of $\log(1/\Pr[X = x])$.
 127 Given two distributions X and Y , the *statistical difference* between the two is denoted
 128 $\Delta(X, Y)$ and is equal to $\sum_{\alpha} |\Pr[X = \alpha] - \Pr[Y = \alpha]|/2$. Equivalently, for finite domains D ,
 129 $\Delta(X, Y) = \max_{S \subseteq D} \{|\Pr_X[S] - \Pr_Y[S]|\}$. This quantity is also known as the *total variation*
 130 *distance* between X and Y . The *support* of X , denoted $\text{supp}(X)$, is $\{x : \Pr[X = x] > 0\}$.

131 ► **Definition 2.** *Promise Problem: a promise problem Π is a pair of disjoint sets (Π_Y, Π_N)*
 132 *(the "YES" and "NO" instances, respectively). A solution for Π is any set S such that*
 133 $\Pi_Y \subseteq S$, and $S \cap \Pi_N = \emptyset$.

134 ► **Definition 3.** *A branching program is a directed acyclic graph with a single source and*
 135 *two sinks labeled 1 and 0, respectively. Each non-sink node in the graph is labeled with a*
 136 *variable in $\{x_1, \dots, x_n\}$ and has two edges leading out of it: one labeled 1 and one labeled 0.*
 137 *A branching program computes a Boolean function f on input $x = x_1 \dots x_n$ by first placing*
 138 *a pebble on the source node. At any time when the pebble is on a node v labeled x_i , the*
 139 *pebble is moved to the (unique) vertex u that is reached by the edge labeled 1 if $x_i = 1$ (or*
 140 *by the edge labeled 0 if $x_i = 0$). If the pebble eventually reaches the sink labeled b , then*
 141 $f(x) = b$. Branching programs can also be used to compute functions $f : \{0, 1\}^m \rightarrow \{0, 1\}^n$,
 142 *by concatenating n branching programs p_1, \dots, p_n , where p_i computes the function $f_i(x) =$*
 143 *the i -th bit of $f(x)$. For more information on the definitions, backgrounds, and nuances of*
 144 *these complexity classes, circuits, and branching programs, see the text by Vollmer [24].*

145 ► **Definition 4.** *Non-interactive zero-knowledge proof (NISZK) [Adapted from [1, 12]]: A*
 146 *non-interactive statistical zero-knowledge proof system for a promise problem Π is defined*
 147 *by a pair of deterministic polynomial time machines² (V, S) (the verifier and simulator,*
 148 *respectively) and a probabilistic routine P (the prover) that is computationally unbounded,*
 149 *together with a polynomial $r(n)$ (which will give the size of the random reference string σ),*
 150 *such that:*

151 1. *(Completeness): For all $x \in \Pi_Y$, the probability (over random σ , and over the random*
 152 *choices of P) that $V(x, \sigma, P(x, \sigma))$ accepts is at least $1 - 2^{-O(|x|)}$.*

153 2. *(Soundness): For all $x \in \Pi_N$, and for every possible prover P' , the probability that*
 154 $V(x, \sigma, P'(x, \sigma))$ accepts is at least $2^{-O(|x|)}$. (Note P' here can be malicious, meaning it
 155 *can try to fool the verifier)*

² In prior work on NISZK [12, 1], 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.

156 3. (Zero Knowledge): For all $x \in \Pi_Y$, the statistical distance between the following two
 157 distributions is bounded by $2^{-|x|}$:

158 a. Choose $\sigma \leftarrow \{0, 1\}^{r(|x|)}$ uniformly random, $p \leftarrow P(x, \sigma)$, and output (p, σ) .
 159 b. $S(x, r)$ (where the coins r for S are chosen uniformly at random).

160 It is known that changing the definition, to have the error probability in the soundness and
 161 completeness conditions and in the simulator's deviation be $\frac{1}{n^{\omega(1)}}$ results in an equivalent
 162 definition [1, 12]. (See the comments after [1, Claim 39].) We will occasionally make use of
 163 this equivalent formulation, when it is convenient.

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

166 NISZK_C denotes the class of problems in NISZK where the verifier V and simulator S lie
 167 in complexity class C .

168 ▶ **Definition 5.** [1, 12] (EA and EA_{NC^0}). Consider Boolean circuits $C_X : \{0, 1\}^m \rightarrow \{0, 1\}^n$
 169 representing distribution X . The promise problem EA is given by:

$$170 \quad \text{EA}_{Y_{es}} := \{(C_X, k) : H(X) > k + 1\}$$

$$172 \quad \text{EA}_{N_o} := \{(C_X, k) : H(X) < k - 1\}$$

173 EA_{NC^0} is the variant of EA where the distribution C_x is an NC^0 circuit with each output bit
 174 depending on at most 4 input bits.

175 ▶ **Definition 6** (SDU and SDU_{NC^0}). Consider Boolean circuits $C_X : \{0, 1\}^m \rightarrow \{0, 1\}^n$
 representing distributions X . The promise problem

$$\text{SDU} = (\text{SDU}_{YES}, \text{SDU}_{NO})$$

175 is given by

$$176 \quad \text{SDU}_{YES} \stackrel{\text{def}}{=} \{C_X : \Delta(X, U_n) < 1/n\}$$

$$177 \quad \text{SDU}_{NO} \stackrel{\text{def}}{=} \{C_X : \Delta(X, U_n) > 1 - 1/n\}.$$

178 SDU_{NC^0} is the analogous problem, where the distributions X are represented by NC^0
 179 circuits where no output bit depends on more than four input bits.

180 ▶ **Theorem 7.** [1, 2]: EA_{NC^0} and SDU_{NC^0} are complete for NISZK_L . EA_{NC^0} remains complete,
 181 even if k is fixed to $k = n - 3$.

182 ▶ **Definition 8.** [10, 23] (SD and SD_{BP}). Consider a pair of Boolean circuits $C_1, C_2 : \{0, 1\}^m \rightarrow \{0, 1\}^n$ representing distributions X_1, X_2 . The promise problem SD is given by:

$$184 \quad \text{SD}_{Y_{es}} := \{(C_1, C_2) : \Delta(X_1, X_2) > 2/3\}$$

$$186 \quad \text{SD}_{N_o} := \{(C_1, C_2) : \Delta(X_1, X_2) < 1/3\}.$$

187 SD_{BP} is the variant of SD where the distributions X_1, X_2 are represented by branching
 188 programs.

2.1 Perfect Randomized Encodings

189 We will make use of the machinery of *perfect randomized encodings* [5].

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

- 193 ■ **Input independent:** for every $x, x' \in \{0, 1\}^n$ such that $f(x) = f(x')$, the random
194 variables $\hat{f}(x, U_m)$ and $\hat{f}(x', U_m)$ are identically distributed.
- 195 ■ **Output Disjoint:** for every $x, x' \in \{0, 1\}^n$ such that $f(x) \neq f(x')$, $\text{supp}(\hat{f}(x, U_m)) \cap$
196 $\text{supp}(\hat{f}(x', U_m)) = \emptyset$.
- 197 ■ **Uniform:** for every $x \in \{0, 1\}^n$ the random variable $\hat{f}(x, U_m)$ is uniform over the set
198 $\text{supp}(\hat{f}(x, U_m))$.
- 199 ■ **Balanced:** for every $x, x' \in \{0, 1\}^n$ $|\text{supp}(\hat{f}(x, U_m))| = |\text{supp}(\hat{f}(x', U_m))| = b$

200 The following property of perfect randomized encodings is established in [10].

201 ▶ **Lemma 10 (entropy).** Let $f : \{0, 1\}^n \rightarrow \{0, 1\}^\ell$ be a function and let $\hat{f} : \{0, 1\}^n \times$
202 $\{0, 1\}^m \rightarrow \{0, 1\}^s$ be a perfect randomized encoding of f with blowup b . Then $H(\hat{f}(U_n, U_m)) =$
203 $H(f(U_n)) + \log b$

204 3 Simulators and Verifiers in AC^0

205 In this section, we show that NISZK_L can be defined equivalently using verifiers and simulators
206 that are computable in AC^0 . The standard complete problems for NISZK and NISZK_L take a
207 circuit C as input, where the circuit is viewed as representing a probability distribution X ;
208 the goal is to approximate the entropy of X , or to estimate how far X is from the uniform
209 distribution. Earlier work [13, 1, 22] that had presented non-interactive zero-knowledge
210 protocols for these problems had made use of the fact that the verifier could compute hash
211 functions, and thereby convert low-entropy distributions to distributions with small support.
212 But an AC^0 verifier cannot compute hash functions [17].

213 Our approach is to “delegate” the problem of computing hash functions to a logspace
214 verifier, and then to make use of the uniform encoding of this verifier to obtain the desired
215 distributions via an AC^0 reduction. To this end, we begin by defining a suitably restricted
216 version of SDU_{NC^0} and show that this restricted version remains complete for NISZK_L under
217 AC^0 reductions (and even under projections).

218 With this new complete problem in hand, we provide a $\text{NISZK}_{\text{AC}^0}$ protocol for the complete
219 problem, to conclude $\text{NISZK}_L = \text{NISZK}_{\text{AC}^0}$.

220 ▶ **Definition 11.** Consider an NC^0 circuit $C : \{0, 1\}^m \rightarrow \{0, 1\}^n$ and the probability distri-
221 bution X on $\{0, 1\}^n$ defined as $C(U_m)$ - where U_m denotes m uniformly random bits. For
222 some fixed $\epsilon > 0$ (chosen later in Remark 16), we define:

$$223 \text{SDU}'_{\text{NC}^0, Y} = \{X : \Delta(C, U_n) < \frac{1}{2^{n^\epsilon}}\}$$

$$224 \text{SDU}'_{\text{NC}^0, N} = \{X : |\text{supp}(X)| \leq 2^{n-n^\epsilon}\}$$

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

228 ▶ **Claim 12.** $\text{SDU}'_{\text{NC}^0} \leq_m^{\text{proj}} \text{SDU}_{\text{NC}^0}$, and thus $\text{SDU}'_{\text{NC}^0} \in \text{NISZK}_L$.

229 **Proof.** On a given probability distribution X defined on $\{0,1\}^n$ for $\text{SDU}'_{\text{NC}^0}$, we claim that
 230 the identity function $f(X) = X$ is a reduction of $\text{SDU}'_{\text{NC}^0}$ to SDU_{NC^0} . If X is a YES instance
 231 for $\text{SDU}'_{\text{NC}^0}$, then $\Delta(X, U_n) < \frac{1}{2^{n\epsilon}}$, which clearly is a YES instance of SDU_{NC^0} . If X is a
 232 NO instance for $\text{SDU}'_{\text{NC}^0}$, then $|\text{supp}(X)| \leq 2^{n-n^\epsilon}$. Thus, if we let T be the complement of
 233 $\text{supp}(X)$, we have that, under the uniform distribution, a string α is in T with probability
 234 $\geq 1 - \frac{1}{2^{n\epsilon}}$, whereas this event has probability zero under X . Thus $\Delta(X, U_n) \geq 1 - \frac{1}{2^{n\epsilon}}$, easily
 235 making it a NO instance of SDU_{NC^0} . \blacktriangleleft

236 **3.1 Hardness for $\text{SDU}'_{\text{NC}^0}$**

237 \blacktriangleright **Theorem 13.** $\text{SDU}'_{\text{NC}^0}$ is hard for NISZK_L under \leq_m^{proj} reductions.

238 **Proof.** In order to show that $\text{SDU}'_{\text{NC}^0}$ is hard for NISZK_L , we will show that the reduction
 239 given in [1] proving the hardness of SDU_{NC^0} for NISZK_L actually produces an instance of
 240 $\text{SDU}'_{\text{NC}^0}$.

241 Let Π be an arbitrary promise problem in NISZK_L with proof system (P, V) and simulator
 242 S . Let x be an instance of Π . Let $M_x(r)$ denote a machine that simulates $S(x)$ with
 243 randomness r to obtain a transcript (σ, p) - if $V(x, \sigma, p)$ accepts then $M_x(r)$ outputs σ ; else
 244 it outputs $0^{|\sigma|}$. We will assume without loss of generality that $|\sigma| = n^k$ for some constant k .
 245

246 It was shown in [13, Lemma 3.1] that for the promise problem EA , there is an NISZK
 247 protocol with completeness error, soundness error and simulator deviation all bounded from
 248 above by 2^{-m} for inputs of length m . Furthermore, as noted in the paragraph before Claim
 249 38 in [1], the proof carries over to show that EA_{BP} has an NISZK_L protocol with the same
 250 parameters. Thus, any problem in NISZK_L can be recognized with exponentially small
 251 error parameters by reducing the problem to EA_{BP} and then running the above protocol for
 252 EA_{BP} on that instance. In particular, this holds for EA_{NC^0} . In what follows, let M_x be the
 253 distribution described in the preceding paragraph, assuming that the simulator S and verifier
 254 V yield a protocol with these exponentially small error parameters.

255 \blacktriangleright **Claim 14.** 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
 256 $|\text{supp}(M_x(r))| \leq 2^{n^k-n^{\epsilon k}}$.

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

259 For $x \in \Pi_{NO}$, from the soundness guarantee of the NISZK_L protocol for EA_{NC^0} , we know
 260 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
 261 message p that the prover can send that will cause V to accept. Thus there are at most
 262 2^{n^k-n} outputs of $M_x(r)$ other than 0^{n^k} . For $\epsilon < \frac{1}{k}$, we have $|\text{supp}(M_x(r))| \leq 2^{n^k-n^{\epsilon k}}$. \blacktriangleleft

263 The above claim talks about the distribution $M_x(r)$ where M is a logspace machine. We
 264 will instead consider an NC^0 distribution with similar properties that can be constructed
 265 using projections. This distribution (denoted by C_x) is a perfect randomized encoding of
 266 $M_x(r)$. We make use of the following construction:

267 \blacktriangleright **Lemma 15.** [1, Lemma 35]. There is a function computable in AC^0 (in fact, it can be
 268 a projection) that takes as input a branching program Q of size l computing a function f
 269 and produces as output a list p_i of NC^0 circuits, where p_i computes the i -th bit of a function
 270 \hat{f} that is a perfect randomized encoding of f that has blowup $2^{\binom{l}{2}-1}2^{(l-1)^2-1}$. Each p_i
 271 depends on at most four input bits from (x, r) (where r is the sequence of random bits in the
 272 randomized encoding).

273 Since the simulator S runs in logspace, each bit of $M_x(r)$ can be simulated with a
 274 branching program Q_x . Furthermore, it is straightforward to see that there is an AC^0 -
 275 computable function that takes x as input and produces an encoding of Q_x as output, and it
 276 can even be seen that this function can be a projection. Let the list of NC^0 circuits produced
 277 from Q_x by the construction of Lemma 15 be denoted C_x .

278 We show that this distribution C_x is an instance of $\text{SDU}'_{\text{NC}^0}$ if $x \in \Pi$. For $x \in \Pi_{YES}$, we
 279 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
 280 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

$$\begin{aligned} \Delta(C_x(r), U_{\log b+n^k}) &= \sum_{\alpha\beta} \left| \Pr[C_x = \alpha\beta] - \frac{1}{2^{n^k+b}} \right|/2 = \sum_{\beta} \sum_{\alpha} \left| \Pr[M_x = \alpha] \frac{1}{2^b} - \frac{1}{2^b} \frac{1}{2^{n^k}} \right|/2 \\ &= \sum_{\alpha} \left| \Pr[M_x = \alpha] - \frac{1}{2^{n^k}} \right|/2 = \Delta(M_x(r), U_{n^k}). \end{aligned}$$

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

282 For $x \in \Pi_{NO}$, Claim 14 shows that $|\text{supp}(M_x(r))| \leq 2^{n^k-n}$. Since the NC^0 circuit C_x is
 283 a perfect randomized encoding of $M_x(r)$, we have that the support of C_x for $x \in \Pi_{NO}$ is
 284 bounded from above by $b \times 2^{n^k-n}$. Note that $\log b$ is polynomial in n ; let $q(n) = \log b$. Let
 285 $r(n)$ denote the length of the output of C ; $r(n) = q(n) + n^k$. Thus the size of $\text{supp}(C_x) \leq$
 286 $2^{n^k-n+q(n)} = 2^{r(n)-n} < 2^{r(n)-r(n)^\epsilon}$ (if $1/\epsilon$ is chosen to be greater than the degree of r), and
 287 hence C_x is a NO instance for $\text{SDU}'_{\text{NC}^0}$. \blacktriangleleft

288 \blacktriangleright **Remark 16.** Here is how we pick ϵ in the definition of $\text{SDU}'_{\text{NC}^0}$. SDU_{NC^0} is in NISZK_L via
 289 some simulator and verifier, where the error parameters are exponentially small, and the
 290 shared reference strings σ have length n^k on inputs of length n . Now we pick $\epsilon > 0$ so that
 291 $\epsilon < 1/k$ (as in Claim 14) and also $1/\epsilon$ is greater than the degree of r (as in the last sentence
 292 of the proof of Theorem 13).

293 3.2 $\text{NISZK}_{\text{AC}^0}$ protocol for $\text{SDU}'_{\text{NC}^0}$ on input X represented by circuit C

294 3.2.1 Non Interactive proof system

- 295 1. Let C take inputs of length m and produce outputs of length n , and let σ be the reference
 296 string of length n .
- 297 2. If there is no r such that $C(r) = \sigma$, then the prover sends \perp . Otherwise, the prover picks
 298 an element r uniformly at random from $p \sim \{r \mid C(r) = \sigma\}$ and sends it to the verifier.
- 299 3. V accepts iff $C(r) = \sigma$.

300 3.2.2 Simulator for $\text{SDU}'_{\text{NC}^0}$ proof system, on input X represented by 301 circuit C

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

304 3.3 Proofs of Zero Knowledge, Completeness and Soundness

305 3.3.1 Completeness

306 \blacktriangleright **Claim 17.** If $X \in \text{SDU}'_{\text{NC}^0, Y}$, then the verifier accepts with probability $\geq 1 - \frac{1}{2^{n^\epsilon}}$.

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

309

310 3.3.2 Soundness

311 \triangleright **Claim 18.** If $X \in \text{SDU}'_{\text{NC}^0, N}$, then for every prover, the probability that the verifier
 312 accepts is at most $\frac{1}{2^{n\epsilon}}$.

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

315 3.3.3 Zero Knowledge

316 \triangleright **Claim 19.** For $X \in \text{SDU}'_{\text{NC}^0, Y}$, $\Delta((p, \sigma), (s, \gamma)) = O(\frac{1}{2^{n\epsilon}})$.

317 **Proof.** Recall that $\sigma \sim \{0, 1\}^n$, $s \sim \{0, 1\}^m$, $p \sim \{r : C(r) = \sigma\}$ and $\gamma = C(s)$. In order
 318 to provide an upper bound on $\Delta((p, \sigma), (s, \gamma))$, we consider the element wise probability of
 319 each distribution and show that for $X \in \text{SDU}'_{\text{NC}^0, Y}$ the claim holds. For $a \in \{0, 1\}^m$ and
 320 $b \in \{0, 1\}^n$ we have :

$$321 \quad \Delta((p, \sigma), (s, \gamma)) = \sum_{(a, b)} \frac{1}{2} |\Pr[(p, \sigma) = (a, b)] - \Pr[(s, \gamma) = (a, b)]|$$

322 Let us consider an element $b \in \{0, 1\}^n$. Let $A_b = \{a_1, a_2, \dots, a_{k_b}\}$ be the pre-images of b under
 323 C i.e. 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 $k_b 2^{-m} = \beta_b$
 324 (since exactly k_b elements of $\{0, 1\}^m$ are mapped to b under C). Let $B = \{b \mid \neg \exists y : C(y) = b\}$.
 325 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 :

$$326 \quad \begin{aligned} \Delta((p, \sigma), (s, \gamma)) &= \sum_{(a, b)} \frac{1}{2} (|\Pr[(p, \sigma) = (a, b)] - \Pr[(s, \gamma) = (a, b)]|) \\ 327 &= \frac{1}{2} \sum_{(a, b): b \in B} |\Pr[(p, \sigma) = (a, b)] - \Pr[(s, \gamma) = (a, b)]| \\ 328 &\quad + \frac{1}{2} \sum_{(a, b): b \notin B} |\Pr[(p, \sigma) = (a, b)] - \Pr[(s, \gamma) = (a, b)]| \end{aligned}$$

329 For (a, b) satisfying $b \in B$, we have $\Pr[(s, \gamma) = (a, b)] = \Pr[(p, \sigma) = (a, b)] = 0$. For $b \notin B$
 330 and a satisfying $C(a) \neq b$ we again have $\Pr[(s, \gamma) = (a, b)] = \Pr[(p, \sigma) = (a, b)] = 0$. For
 331 $(a, b) : C(a) = b$ we have $\Pr[(s, \gamma) = (a, b)] = 2^{-m}$ since $s \sim U_m$ and picking s fixes b . We
 332 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
 333 A_b . This gives us

$$335 \quad \begin{aligned} \Delta((p, \sigma), (s, \gamma)) &= \frac{1}{2} \sum_{(a, b): C(a) = b} \left| 2^{-m} - \frac{2^{-n}}{k_b} \right| \\ 336 &= \frac{1}{2} \sum_{(a, b): C(a) = b} \left| 2^{-m} - \frac{2^{-m-n}}{\beta_b} \right| \\ 337 &= \frac{1}{2} \sum_{(a, b): C(a) = b} \frac{2^{-m}}{\beta_b} |\beta_b - 2^{-n}| \\ 338 &\leq \frac{1}{2} \sum_{(a, b): C(a) = b} |\beta_b - 2^{-n}| = \Delta(C(U_m), U_n) \leq \frac{1}{2^{n\epsilon}} \end{aligned}$$

339

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

341
$$\Delta((p, \sigma), (s, \gamma)) = O\left(\frac{1}{2^{n^\epsilon}}\right).$$

342

◀

343 **4 Simulator and Verifier in PM**

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

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

352
$$\text{PM} := \{A : A \leq_T^L \text{Perfect Matching}\}$$

353 It is known that $\text{NL} \subseteq \text{PM}$ [16].

354 Our argument proceeds by first observing³ that $\text{NISZK}_L = \text{NISZK}_{\oplus L}$, and then making
 355 use of the details of the argument that Perfect Matching is in $\oplus L/\text{poly}$ [4].

356 ▶ **Proposition 20.** $\text{NISZK}_{\oplus L} = \text{NISZK}_L$

357 **Proof.** It suffices to show $\text{NISZK}_{\oplus L} \subseteq \text{NISZK}_L$. We do this by showing that the problem
 358 EA_{NC^0} is hard for $\text{NISZK}_{\oplus L}$; this suffices since EA_{NC^0} is complete for NISZK_L . The proof
 359 of [1, Theorem 26] (showing that EA_{NC^0} is complete for NISZK_L involves (a) building a
 360 branching program to simulate a logspace computation called M_x that is constructed from a
 361 logspace-computable simulator and verifier, and (b) constructing an NC^0 -computable perfect
 362 randomized encoding of M_x , using the fact that $L \subset \mathcal{PREN}$, where \mathcal{PREN} is the class
 363 defined in [5], consisting of all problems with perfect randomized encodings. But Theorem
 364 4.18 in [5] shows the stronger result that $\oplus L$ lies in \mathcal{PREN} , and hence the argument of
 365 [1, Theorem 26] carries over immediately, to reduce any problem in $\text{NISZK}_{\oplus L}$ to EA_{NC^0} (by
 366 modifying step (a), to build a *parity* branching program for M_x that is constructed from a
 367 $\oplus L$ simulator and verifier). ◀

368 We also rely on the following lemma:

369 ▶ **Lemma 21.** *Adapted from [4, Section 3] and [19, Section 4]: Let $W = (w_1, w_2, \dots, w_{n^{k+3}})$
 370 be a sequence of n^{k+3} weight functions, where each $w_i : [n] \rightarrow [4n^2]$ is a distinct weight
 371 assignment to edges in n -vertex graphs. Let (G, w_i) denote the result of weighting the edges
 372 of G using weight assignment w_i . Then there is a function f in GapL , such that, if (G, w_i)
 373 has a unique perfect matching of weight j , then $f(G, W, i, j) \in \{1, -1\}$, and if G has no
 374 perfect matching, then for every (W, i, j) , it holds that $f(G, W, i, j) = 0$. Furthermore, if W
 375 is chosen uniformly at random, then with probability $\geq 1 - 2^{-n^k}$, for each n -vertex graph G :*

376 □ *If G has no perfect matching then $\forall i \forall j f(G, W, i, j) = 0$.*

³ This equality was previously observed in [22].

377 ■ If G has a perfect matching then $\exists i$ such that (G, w_i) has a unique minimum-weight
 378 matching, and hence $\exists i \exists j f(G, W, i, j) \in \{1, -1\}$.

379 Thus if we define $g(G, W)$ to be $1 - \Pi_{i,j}(1 - f(G, W, i, j)^2)$, we have that $g \in \text{GapL}$ and with
 380 probability $\geq 1 - 2^{-n^k}$ (for randomly-chosen W), $g(G, W) = 1$ if G has a perfect matching,
 381 and $g(G, W) = 0$ otherwise.

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

385 ▶ **Corollary 22.** For every language $A \in \text{PM}$ there is a language $B \in \oplus\text{L}$ such that, if $x \in A$,
 386 then $\Pr_{W \leftarrow [4n^2]^{n^5}}[(x, W) \in B] \geq 1 - 2^{-n^2}$, and if $x \notin A$, then $\Pr_{W \leftarrow [4n^2]^{n^5}}[(x, W) \in B] \leq$
 387 2^{-n^2} .

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

393 Let $C = \{(G, W) : g(G, W) \equiv 1 \pmod{2}\}$, where g is the function from Lemma 21. Clearly,
 394 $C \in \oplus\text{L}$. Now, a logspace oracle machine with input (x, W) and oracle C can simulate
 395 the computation of M^P on x ; each time M poses the query “Is $G \in P$ ”, instead we ask if
 396 $(G, W) \in C$. Then with high probability (over the random choice of W) all of the queries
 397 will be answered correctly and hence this routine will accept if and only if $x \in A$, by
 398 Lemma 21. Let B be the language accepted by this logspace oracle machine. We see that
 399 $B \in \text{L}^C \subseteq \text{L}^{\oplus\text{L}} = \oplus\text{L}$, where the last equality is from [14]. ◀

400 ▶ **Theorem 23.** $\text{NISZK}_{\text{L}} = \text{NISZK}_{\text{PM}}$

401 **Proof.** We show that $\text{NISZK}_{\text{PM}} \subseteq \text{NISZK}_{\oplus\text{L}}$, and then appeal to Proposition 20.

402 Let Π be an arbitrary problem in NISZK_{PM} , and let (S, P, V) be the PM simulator, prover,
 403 and verifier for Π , respectively. Let S' and V' be the $\oplus\text{L}$ languages that are probabilistic
 404 realizations of S, V , respectively, guaranteed by Corollary 22. We now define a NISZK_{L}
 405 protocol (S'', P'', V'') for Π .

406 On input x with shared randomness σW , the prover P'' sends the same message $p =$
 407 $P(x, \sigma)$ as the original prover sends. The verifier V'' , returns the value of $V'((x, \sigma, p), W)$,
 408 which with high probability is equal to $V(x, \sigma, p)$. The simulator S'' , given as input x and
 409 random sequence rW , executes $S'((x, r, i), W)$ for each bit position i to obtain a bit that
 410 (with high probability) is equal to the i^{th} bit of $S(x, r)$, which is a string of the form (σ, p) ,
 411 and outputs $(\sigma W, p)$.

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

413 ■ Completeness: Suppose $x \in \Pi_Y$, then $\Pr_{\sigma}[V(x, \sigma, P(x, \sigma)) = 1] \geq 1 - 2^{-O(n)}$. Since
 414 $\forall y \in \{0, 1\}^n : \Pr_W[V(y) = V'(y, W)] \geq 1 - 2^{-n^k}$ we have:

$$415 \Pr_{\sigma W}[V'((x, \sigma, P''(x, \sigma)), W) = 1] \geq [1 - 2^{-O(n)}][1 - 2^{-n^k}] = 1 - 2^{-O(n)}$$

416 ■ Soundness: Suppose $x \in \Pi_N$, then $\Pr_{\sigma}[V(x, \sigma, p) = 0] \geq 1 - 2^{-O(n)}$. Since
 417 $\forall y \in \{0, 1\}^n : \Pr_W[V(y) = V'(y, W)] \geq 1 - 2^{-n^k}$, we have:

$$418 \Pr_{\sigma W}[\forall p : V'((x, \sigma, p), W) = 0] \geq [1 - 2^{-O(n)}][1 - 2^{-n^k}] = 1 - 2^{-O(n)}$$

419 ■ Statistical Zero-Knowledge: Suppose $x \in \Pi_Y$. Let S^* denote the distribution on strings
 420 of the form (σ, p) that $S(x, r)$ produces, where r is uniformly generated, and let P^* denote
 421 the distribution on strings given by $(\sigma, P(x, \sigma))$ where σ is chosen uniformly at random.
 422 Similarly, let S''^* denote the distribution on strings of the form $(\sigma W, p)$ that $S''(x, rW)$
 423 produces, where r and W are chosen uniformly, and let P''^* be the distribution given by
 424 $(\sigma W, P''(x, \sigma W))$. Let $A = \{(\sigma W, p) : \exists i \exists r S(x, r)_i \neq S'((x, r, i), W)\}$.
 425 Since $\Pr_W [\forall i \forall r : S(x, r)_i = S'((x, r, i), W)] \geq 1 - 2^{-O(n)}$ we have:

$$\begin{aligned} 426 \quad \Delta(S''^*, P''^*) &= \frac{1}{2} \sum_{(\sigma W, p)} |\Pr[S''^* = (\sigma W, p)] - \Pr[P''^* = (\sigma W, p)]| \\ 427 \quad &\leq \frac{1}{2} (2^{-O(n)} + \sum_{(\sigma W, p) \in \bar{A}} |\Pr[S''^* = (\sigma W, p)] - \Pr[P''^* = (\sigma W, p)]|) \\ 428 \quad &= \frac{1}{2} (2^{-O(n)} + \sum_{(\sigma W, p) \in \bar{A}} |\Pr[S^* = (\sigma, p)] - \Pr[P^* = (\sigma, p)]| \Pr[W]) \\ 429 \quad &\leq 2^{-O(n)} + \sum_W \Pr[W] \frac{1}{2} \sum_{(\sigma, p)} |\Pr[S^* = (\sigma, p)] - \Pr[P^* = (\sigma, p)]| \\ 430 \quad &= 2^{-O(n)} + \Delta(S^*, P^*) = 2^{-O(n)} \end{aligned}$$

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

433 5 Additional problems in \mathbf{NISZK}_L

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

437 The following definition is from [2]:

438 ► **Definition 24.** A promise problem $A = (Y, N)$ is $\leq_m^{\mathbf{BPL}}$ -reducible to $B = (Y', N')$ with
 439 threshold θ if there is a logspace-computable function f and there is a polynomial p such that

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

442 Note, in particular, that the logspace machine computing the reduction has two-way access
 443 to the random bits r ; this is consistent with the model of probabilistic logspace that is used
 444 in defining \mathbf{NISZK}_L .

445 ► **Theorem 25.** \mathbf{NISZK}_L is closed under $\leq_m^{\mathbf{BPL}}$ reductions with threshold $1 - \frac{1}{n^{\omega(1)}}$.

446 **Proof.** Let $\Pi \leq_m^{\mathbf{BPL}} \mathbf{EA}_{\mathbf{NC}^0}$, via logspace-computable function f . Let (S_1, V_1, P_1) be the \mathbf{NISZK}_L
 447 proof system for $\mathbf{EA}_{\mathbf{NC}^0}$.

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

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

449 ■ **Algorithm 2** Prover $P(x, (\sigma, \sigma'))$

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

450 ■ **Algorithm 3** Verifier $V(x, (\sigma, \sigma'), p)$

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

451 We now claim that (S, P, V) is a NISZK_L protocol for Π .

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

454 ■ Completeness: Suppose x is YES instance of Π . Then with probability $1 - \frac{1}{n^{\omega(1)}}$ (over
455 randomness of σ'): $f(x, \sigma')$ is a YES instance of EA_{NC^0} . Thus for a randomly chosen σ :

$$456 \Pr[V_1(f(x, \sigma'), \sigma, P_1(f(x, \sigma'), \sigma)) = 1] \geq 1 - \frac{1}{n^{\omega(1)}}$$

457 ■ Soundness: Suppose x is NO instance of Π . Then with probability $1 - \frac{1}{n^{\omega(1)}}$ (over
458 randomness of σ'): $f(x, \sigma')$ is a NO instance of EA_{NC^0} . Thus for a randomly chosen σ :

$$459 \Pr[V_1(f(x, \sigma'), \sigma, P_1(f(x, \sigma'), \sigma)) = 0] \geq 1 - \frac{1}{n^{\omega(1)}}$$

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

466

◀

467 McKenzie and Cook [18] defined and studied the problems **LCON**, **LCONX** and **LCONNUL**.
468 **LCON** is the problem of determining if a system of linear congruences over the integers mod
469 q has a solution. **LCONX** is the problem of finding a solution, if one exists, and **LCONNUL**
470 is the problem of computing a spanning set for the null space of the system.

471 These problems are known to lie in uniform NC^3 [18], but are not known to lie in uniform
472 NC^2 , although Arvind and Vijayaraghavan showed that there is a set B in $\text{L}^{\text{GapL}} \subseteq \text{DET} \subseteq \text{NC}^2$
473 such that $x \in \text{LCON}$ if and only if $(x, W) \in B$, where B is a randomly-chosen weight function
474 [6]. (The probability of error is exponentially small.) The mapping $x \mapsto (x, W)$ is clearly a
475 \leq_m^{BPL} reduction. Since $\text{DET} \subseteq \text{NISZK}_L$ [1], it follows that

$$476 \text{LCON} \in \text{NISZK}_L$$

477 The arguments in [6] carry over to **LCONX** and **LCONNUL** as well.

478 ▶ **Corollary 26.** $\text{LCON} \in \text{NISZK}_L$. $\text{LCONX} \in \text{NISZK}_L$. $\text{LCONNUL} \in \text{NISZK}_L$.

479 6 Varying the Power of the Verifier

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

487 We define $\text{NISZK}_{A,B}$ as the class of problems with a NISZK protocol where the simulator
488 is in A and the verifier is in B (and hence $\text{NISZK}_A = \text{NISZK}_{A,A}$). We will consider the
489 case where $A \subseteq B \subseteq \text{NISZK}_A$ and A, B are both classes of functions that are closed under
490 composition.

491 \blacktriangleright **Theorem 27.** $\text{NISZK}_{A,B} = \text{NISZK}_A$

492 **Proof.** Let Π be an arbitrary promise problem in $\text{NISZK}_{A,B}$ with (S_1, V_1, P_1) being the A
 493 simulator, B verifier, and prover for Π 's proof system, where the reference string has length
 494 $p_1(|x|)$ and the prover's messages have length $q_1(|x|)$. Since $V_1 \in B \subseteq \text{NISZK}_A$, $L(V_1)$ has
 495 a proof system (S_2, V_2, P_2) , where the reference string has length $p_2(|x|)$ and the prover's
 496 messages have length $q_2(|x|)$.

497 \blacktriangleright **Lemma 28.** *We may assume without loss of generality that $p_1(n) > p_2(n) + q_2(n)$.*

498 **Proof.** If it is not the case that $p_1(n) > p_2(n) + q_2(n)$, then let $r(n) = p_2(n) + q_2(n) - p_1(n)$.
 499 Consider a new proof system (S'_1, V'_1, P'_1) that is identical to (S_1, V_1, P_1) , except that the
 500 reference string now has length $p_1(n) + r(n)$ (where P'_1 and V'_1 ignore the additional $r(n)$
 501 random bits). The simulator S'_1 uses an additional $r(n)$ random bits and simply appends
 502 those bits to the output of S_1 . The language $L(V'_1)$ is still in NISZK_A , with a proof system
 503 (S'_2, V'_2, P'_2) where the reference string still has length $p_2(n)$, since membership in $L(V'_1)$ does
 504 not depend on the "new" $r(n)$ random bits, and hence S'_2, V'_2 and P'_2 , given input $(x, \sigma r, p)$
 505 behave exactly as S_2, V_2 and P_2 behave when given input (x, σ, p) . \blacktriangleleft

506 Then Π has the following NISZK_A proof system:

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

Data: $x \in \Pi_{Y_{es}} \cup \Pi_{N_o}$
 507 $(\sigma, p) \leftarrow S_1(x, r_1);$
 $(\sigma', p') \leftarrow S_2((x, \sigma, p), r_2);$
return $((\sigma, \sigma'), (p, p'))$;

Algorithm 5 Prover $P(x, \sigma\sigma')$

Data: $x \in \Pi_{Y_{es}} \cup \Pi_{N_o}, \sigma \in \{0, 1\}^{p_1(|x|)}, \sigma' \in \{0, 1\}^{p_2(|x|)}$
if $x \in \Pi_{Y_{es}}$ **then**
 508 | $p \leftarrow P_1(x, \sigma);$
 | $p' \leftarrow P_2((x, \sigma, p), \sigma');$
 | **return** $(p, p');$
else
 | **return** \perp, \perp ;
end

Algorithm 6 Verifier $V(x, (\sigma, \sigma'), (p, p'))$

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

510 \blacksquare **Correctness:** Suppose $x \in \Pi_{Y_{es}}$, then given random σ , with probability $(1 - \frac{1}{2^{O(|x|)})})$:
 511 $(x, \sigma, P_1(x, \sigma)) \in L(V_1)$ which means with probability $(1 - \frac{1}{2^{O(|x|+p_1(|x|)+|p|)})})$ it holds that
 512 $((x, \sigma, p), \sigma', P_2(x, \sigma, P_1(x, \sigma)) \in L(V_2)$. So the probability that V accepts is at least:

$$513 \quad (1 - \frac{1}{2^{O(|x|)})})(1 - \frac{1}{2^{O(|x|+p_1(|x|)+q_1(|x|))})}) = 1 - \frac{1}{2^{O(|x|)}}$$

514 \blacksquare **Soundness:** Suppose $x \in \Pi_{N_o}$. When given a random σ , we have that with probability less
 515 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
 516 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
 517 σ'). So the probability that V rejects is at least:

$$518 \quad (1 - \frac{1}{2^{O(|x|)}})(1 - \frac{1}{2^{O(|x|+p(|x|)+|p|)}}) = 1 - \frac{1}{2^{O(|x|)}}$$

519 ■ Statistical Zero-Knowledge: Let P_1^* denote the distribution that samples σ and outputs
 520 $(\sigma, P_1(x, \sigma))$. Similarly, let $P_2^*(\sigma, p)$ denote the distribution that samples σ' and outputs
 521 $(\sigma\sigma', P_2((x, \sigma, p), \sigma'))$. P^* will be defined as the distribution $((\sigma\sigma'), P(x, \sigma, \sigma'))$ where σ
 522 and σ' are chosen uniformly at random. In the same way, let S^* refer to the distribution
 523 produced by S on input x , let S_1^* refer to the distribution produced by $S_1(x)$, and let
 524 $S_2^*(\sigma, p)$ be the distribution induced by S_2 on input (x, σ, p) . Now we can partition the
 525 set of possible outcomes $((\sigma, \sigma'), (p, p'))$ of S^* and P^* into 3 blocks:

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

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

$$530 \quad \Delta(S^*, P^*) = \frac{1}{2} \sum_{j \in \{1, 2, 3\}} \sum_{y \in A_j} \left| \Pr_{S^*}[y] - \Pr_{P^*}[y] \right|$$

$$531 = \frac{1}{2} \sum_{y \in A_1} \left| \Pr_{S^*}[y] - \Pr_{P^*}[y] \right| + \frac{1}{2} \sum_{j \in \{2, 3\}} \sum_{y \in A_j} \left[\Pr_{S^*}[y] + \Pr_{P^*}[y] \right]$$

533 We concentrate first on A_1 .

$$534 \quad \sum_{y \in A_1} \left| \Pr_{S^*}[y] - \Pr_{P^*}[y] \right|$$

$$535 = \sum_{(\sigma', p')} \left(\sum_{\{(\sigma, p): y=((\sigma, \sigma'), (p, p')) \in A_1\}} \left| \Pr_{S^*}[y|\sigma', p'] \Pr_{S^*}[(\sigma', p')] - \Pr_{P^*}[y|\sigma', p'] \Pr_{P^*}[(\sigma', p')] \right| \right) \quad (*)$$

537 Here

$$538 \quad \Pr_{S^*}[(\sigma', p')] = \sum_{(\sigma, p)} \Pr_{S^*}[(\sigma, \sigma'), (p, p')]$$

539 and

$$540 \quad \Pr_{P^*}[(\sigma', p')] = \sum_{(\sigma, p)} \Pr_{P^*}[(\sigma, \sigma'), (p, p')].$$

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

$$543 \quad \left| \Pr_{S^*}[y|\sigma', p'] \Pr_{S^*}[(\sigma', p')] - \Pr_{P^*}[y|\sigma', p'] \Pr_{P^*}[(\sigma', p')] \right|$$

$$544 = \left| \left(\Pr_{S^*}[y|\sigma', p'] \Pr_{S^*}[(\sigma', p')] - \Pr_{P^*}[y|\sigma', p'] \Pr_{S^*}[(\sigma', p')] \right) + \right.$$

$$545 \quad \left. \left(\Pr_{P^*}[y|\sigma', p'] \Pr_{P^*}[(\sigma', p')] - \Pr_{S^*}[y|\sigma', p'] \Pr_{P^*}[(\sigma', p')] \right) \right|$$

$$546 = \left| \left(\Pr_{S_1^*}[(\sigma, p)] - \Pr_{P_1^*}[(\sigma, p)] \right) \Pr_{S^*}[(\sigma', p')] + \Pr_{P_1^*}[(\sigma, p)] \left(\Pr_{S^*}[(\sigma', p')] - \Pr_{P^*}[(\sigma', p')] \right) \right|$$

$$547 \leq \left| \Pr_{S_1^*}[(\sigma, p)] - \Pr_{P_1^*}[(\sigma, p)] \right| \Pr_{S^*}[(\sigma', p')] + \Pr_{P_1^*}[(\sigma, p)] \left| \Pr_{S^*}[(\sigma', p')] - \Pr_{P^*}[(\sigma', p')] \right|$$

$$548 = \left| \Pr_{S_1^*}[(\sigma, p)] - \Pr_{P_1^*}[(\sigma, p)] \right| \Pr_{S^*}[(\sigma', p')] + \Pr_{P_1^*}[(\sigma, p)] \delta(\sigma', p')$$

550 Thus (*) is no more than

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

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

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

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

$$\begin{aligned}
565 \quad \delta(\sigma', p') &= \left| \Pr_{S^*}[\sigma', p'] - \Pr_{P^*}[\sigma', p'] \right| \\
566 \quad &= \left| \sum_{(\sigma, p)} \Pr_{S_2^*(\sigma, p)}[(\sigma', p')] \Pr_{S_1^*}[(\sigma, p)] - \sum_{(\sigma, p)} \Pr_{P_2^*(\sigma, p)}[(\sigma', p')] \Pr_{P_1^*}[(\sigma, p)] \right| \\
567 \quad &= \left| \sum_{(\sigma, p)} \Pr_{S_2^*(\sigma, p)}[(\sigma', p')] \Pr_{S_1^*}[(\sigma, p)] - \sum_{(\sigma, p)} \Pr_{P_2^*(\sigma, p)}[(\sigma', p')] \Pr_{S_1^*}[(\sigma, p)] \right. \\
568 \quad &\quad \left. + \sum_{(\sigma, p)} \Pr_{P_2^*(\sigma, p)}[(\sigma', p')] \Pr_{S_1^*}[(\sigma, p)] - \sum_{(\sigma, p)} \Pr_{P_2^*(\sigma, p)}[(\sigma', p')] \Pr_{P_1^*}[(\sigma, p)] \right| \\
569 \quad &= \left| \sum_{(\sigma, p)} \left(\Pr_{S_2^*(\sigma, p)}[(\sigma', p')] - \Pr_{P_2^*(\sigma, p)}[(\sigma', p')] \right) \Pr_{S_1^*}[(\sigma, p)] \right. \\
570 \quad &\quad \left. + \sum_{(\sigma, p)} \Pr_{P_2^*(\sigma, p)}[(\sigma', p')] (\Pr_{S_1^*}[(\sigma, p)] - \Pr_{P_1^*}[(\sigma, p)]) \right| \\
571 \quad &\leq \sum_{(\sigma, p)} \left| \Pr_{S_2^*(\sigma, p)}[(\sigma', p')] - \Pr_{P_2^*(\sigma, p)}[(\sigma', p')] \right| \Pr_{S_1^*}[(\sigma, p)] \\
572 \quad &\quad + \sum_{(\sigma, p)} \Pr_{P_2^*(\sigma, p)}[(\sigma', p')] \left| \Pr_{S_1^*}[(\sigma, p)] - \Pr_{P_1^*}[(\sigma, p)] \right| \\
573 \quad &= \sum_{(\sigma, p)} 2\Delta(S_2^*(\sigma, p), P_2^*(\sigma, p)) \Pr_{S_1^*}[(\sigma, p)] \\
574 \quad &\quad + \sum_{(\sigma, p)} \Pr_{P_2^*(\sigma, p)}[(\sigma', p')] \left| \Pr_{S_1^*}[(\sigma, p)] - \Pr_{P_1^*}[(\sigma, p)] \right| \\
575 \quad &\leq \sum_{(\sigma, p)} \frac{2}{2^{|(x, \sigma, p)|}} \Pr_{S_1^*}[(\sigma, p)] + \sum_{(\sigma, p)} \Pr_{P_2^*(\sigma, p)}[(\sigma', p')] \left| \Pr_{S_1^*}[(\sigma, p)] - \Pr_{P_1^*}[(\sigma, p)] \right| \\
576 \quad &= \frac{2}{2^{|x| + p_1(|x|) + q_1(|x|)}} + \sum_{(\sigma, p)} \Pr_{P_2^*(\sigma, p)}[(\sigma', p')] \left| \Pr_{S_1^*}[(\sigma, p)] - \Pr_{P_1^*}[(\sigma, p)] \right|
\end{aligned}$$

578 where the last inequality holds, since the summation in $(**)$ is taken over tuples, such
579 that each (x, σ, p) is a YES instance of $L(V_1)$.

580 Replacing each term in $(**)$ with this upper bound, thus yields the following upper bound
581 on $(*)$:

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

591 where the last inequality follows from Lemma 28. Thus, A_1 contributes only a negligible
592 quantity to $\Delta(S^*, P^*)$.

593 We now move on to consider A_2 and A_3 .

594
$$\Pr_{P^*}[y \in A_2] = \sum_{\{(\sigma,p):(x,\sigma,p) \in L(V_1)\}} \Pr[V_2(x, \sigma, p) \text{ rejects}] \leq \sum_{(\sigma,p)} \frac{1}{2^{|x|+|\sigma|+|p|}} \leq \frac{1}{2^{|x|}}.$$

595
$$\Pr_{S^*}[y \in A_2] = \sum_{\{(\sigma,p):(x,\sigma,p) \in L(V_1)\}} (\Pr[V_2(x, \sigma, p) \text{ rejects}] + \Delta(S_2^*(\sigma, p), P_2^*(\sigma, p))) \leq \frac{2}{2^{|x|}}.$$

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

598

599 ▶ **Corollary 29.** $\mathbf{NISZK_L} = \mathbf{NISZK_{AC^0}} = \mathbf{NISZK_{AC^0,DET}} = \mathbf{NISZK_{NL,DET}}$

600 The proof of Theorem 27 did not make use of the condition that the verifier is at least as
601 powerful as the simulator. Thus, maintaining the condition that $A \subseteq B \subseteq \mathbf{NISZK}_A$, we also
602 have the following corollary:

603 ▶ **Corollary 30.** $\mathbf{NISZK}_B = \mathbf{NISZK}_{B,A}$

604 ▶ **Corollary 31.** $\mathbf{NISZK}_{A,B} \subseteq \mathbf{NISZK}_{B,A}$

605 ▶ **Corollary 32.** $\mathbf{NISZK_{DET}} = \mathbf{NISZK_{DET,AC^0}}$

606 **7 $\mathbf{SZK_L}$ closure under \leq_{bf-tt}^L reductions**

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

609 The authors of [10], after defining the class $\mathbf{SZK_L}$, wrote:

610 We also mention that all the known closure and equivalence properties of \mathbf{SZK} (e.g.
611 closure under complement [20], equivalence between honest and dishonest verifiers
612 [13], and equivalence between public and private coins [20]) also hold for the class
613 $\mathbf{SZK_L}$.

614 In this section, we consider a variant of a closure property of \mathbf{SZK} (closure under \leq_{bf-tt}^P
615 [23]), and show that it also holds⁴ for $\mathbf{SZK_L}$. Although our proof follows the general approach
616 of the proof of [23, Theorem 4.9], there are some technicalities with showing that certain
617 computations can be accomplished in logspace (and for dealing with distributions represented
618 by branching programs instead of circuits) that require proof. (The characterization of $\mathbf{SZK_L}$
619 in terms of reducibility to the Kolmogorov-random strings presented in [2] relies on this
620 closure property.)

⁴ We observe that open questions about closure properties of \mathbf{NISZK} also translate to open questions about $\mathbf{NISZK_L}$. \mathbf{NISZK} is not known to be closed under union [21], and neither is $\mathbf{NISZK_L}$. Neither is known to be closed under complementation. Both are closed under conjunctive logspace-truth-table reductions.

621 ► **Definition 33.** (From [23, Definition 4.7]) For a promise problem Π , the characteristic
 622 function of Π is the map $\mathcal{X}_\Pi : \{0, 1\}^* \rightarrow \{0, 1, *\}$ given by

$$623 \quad \mathcal{X}_\Pi(x) = \begin{cases} 1 & \text{if } x \in \Pi_{Yes}, \\ 0 & \text{if } x \in \Pi_{No}, \\ * & \text{otherwise.} \end{cases}$$

624 ► **Definition 34.** Logspace Boolean formula truth-table reduction (\leq_{bf-tt}^L reduction): We
 625 say a promise problem Π **logspace Boolean formula truth-table reduces** to Γ if there
 626 exists a logspace-computable function f , which on input x produces a tuple (y_1, \dots, y_m) and
 627 a Boolean formula ϕ (with m input gates) such that:

$$628 \quad x \in \Pi_{Yes} \implies \phi(\mathcal{X}_\Gamma(y_1), \dots, \mathcal{X}_\Gamma(y_m)) = 1$$

$$629 \quad x \in \Pi_{No} \implies \phi(\mathcal{X}_\Gamma(y_1), \dots, \mathcal{X}_\Gamma(y_m)) = 0$$

631 We begin by proving a logspace analogue of a result from [23], used to make statistically
 632 close pairs of distributions closer and statistically far pairs of distributions farther.

633 ► **Lemma 35.** (Polarization Lemma, adapted from [23, Lemma 3.3]) There is a logspace-
 634 computable function that takes a triple $(P_1, P_2, 1^k)$, where P_1 and P_2 are branching programs,
 635 and outputs a pair of branching programs (Q_1, Q_2) such that:

$$636 \quad \Delta(P_1, P_2) < \frac{1}{3} \implies \Delta(Q_1, Q_2) < 2^{-k}$$

$$637 \quad \Delta(P_1, P_2) > \frac{2}{3} \implies \Delta(Q_1, Q_2) > 1 - 2^{-k}$$

639 To prove this, we adapt the same method as in [23] and alternate two different procedures,
 640 one to drive pairs with large statistical distance closer to 1, and one to drive distributions
 641 with small statistical distance closer to 0. The following lemma will do the former:

642 ► **Lemma 36.** (Direct Product Lemma, from [23, Lemma 3.4]) Let X and Y be distributions
 643 such that $\Delta(X, Y) = \epsilon$. Then for all k ,

$$644 \quad k\epsilon \geq \Delta(\otimes^k X, \otimes^k Y) \geq 1 - 2 \exp(-k\epsilon^2/2)$$

645 The proof of this statement follows from [23]. To use this for Lemma 35, we note that a
 646 branching program for $\otimes^k P$ can easily be created in logspace from a branching program P
 647 by simply copying and concatenating k independent copies of P together.

648 We now introduce a lemma to push close distributions closer:

649 ► **Lemma 37.** (XOR Lemma, adapted from [23, Lemma 3.5]) There is a logspace-computable
 650 function that maps a triple $(P_0, P_1, 1^k)$, where P_0 and P_1 are branching programs, to a pair
 651 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
 652 are defined as follows:

$$653 \quad Q_0 = \bigotimes_{i \in [k]} P_{y_i} : y \leftarrow_R \{y \in \{0, 1\}^k : \bigoplus_{i \in [k]} y_i = 0\}$$

$$654 \quad Q_1 = \bigotimes_{i \in [k]} P_{y_i} : y \leftarrow_R \{y \in \{0, 1\}^k : \bigoplus_{i \in [k]} y_i = 1\}$$

656 **Proof.** The proof that $\Delta(Q_0, Q_1) = \Delta(P_0, P_1)^k$ follows from [23, Proposition 3.6]. To finish
 657 proving this lemma, we show a logspace-computable mapping between $(P_0, P_1, 1^k)$ and
 658 (Q_0, Q_1) .

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

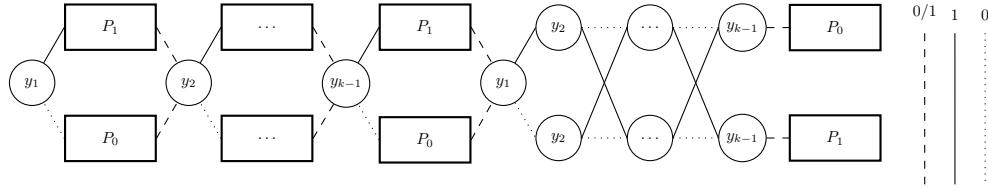


Figure 1 Branching program for Q_0 of Lemma 37

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

669

670 We now have the tools to prove Lemma 35.

671 **Proof.** From [23, Section 3.2], we know that we can polarize $(P_0, P_1, 1^k)$ by:

- 672 ■ Letting $l = \lceil \log_{4/3} 6k \rceil$, $j = 3^{l-1}$
- 673 ■ Applying Lemma 37 to $(P_0, P_1, 1^l)$ to get (P'_0, P'_1)
- 674 ■ Applying Lemma 36: $P''_0 = \otimes^j P'_0$, $P''_1 = \otimes^j P'_1$
- 675 ■ Applying Lemma 37 to $(P''_0, P''_1, 1^k)$ to get (Q_0, Q_1)

676 Each step is computable in logspace, and since logspace is closed under composition, this
 677 completes our proof. \blacktriangleleft

678 We also mention the following lemma, which will be useful in evaluating the Boolean
 679 formula given by the $\leq_{\text{bf-tt}}^L$ reduction.

680 ► **Lemma 38.** *There is a function in NC^1 that takes as input a Boolean formula ϕ (with m
 681 input bits) and produces as output an equivalent formula ψ with the following properties:*

- 682 1. *The depth of ψ is $O(\log m)$.*
- 683 2. *ψ is a tree with alternating levels of AND and OR gates.*
- 684 3. *The tree's non-leaf structure is always the same for a fixed input length.*
- 685 4. *All NOT gates are located just before the leaves.*

686 **Proof.** Although this lemma does not seem to have appeared explicitly in the literature,
 687 it is known to researchers, and is closely related to results in [11] (see Theorems 5.6 and
 688 6.3, and Lemma 3.3) and in [3] (see Lemma 5). Alternatively, one can derive this by using
 689 the fact that the Boolean formula evaluation problem lies in NC^1 [7, 8], and thus there is
 690 an alternating Turing machine M running in $O(\log n)$ time that takes as input a Boolean

691 formula ψ and an assignment α to the variables of ψ , and returns $\psi(\alpha)$. We may assume
 692 without loss of generality that M alternates between existential and universal states at each
 693 step, and that M runs for exactly $c \log n$ steps on each path (for some constant c), and that
 694 M accesses its input (via the address tape that is part of the alternating Turing machine
 695 model) only at a halting step, and that M records the sequence of states that it has visited
 696 along the current path in the current configuration. Thus the configuration graph of M , on
 697 inputs of length n , corresponds to a formula of $O(\log n)$ depth having the desired structure,
 698 and this formula can be constructed in NC^1 . Given a formula ϕ , an NC^1 machine can thus
 699 build this formula, and hardwire in the bits that correspond to the description of ϕ , and
 700 identify the remaining input variables (corresponding to M reading the bits of α) with the
 701 variables of ϕ . The resulting formula is equivalent to ϕ and satisfies the conditions of the
 702 lemma. \blacktriangleleft

703 **► Definition 39.** (From [23, Definition 4.8]) For a promise problem Π , we define a new
 704 promise problem $\Phi(\Pi)$ as follows:

$$705 \quad \Phi(\Pi)_{\text{Yes}} = \{(\phi, x_1, \dots, x_m) : \phi(\mathcal{X}_\Pi(x_1), \dots, \mathcal{X}_\Pi(x_m)) = 1\}$$

$$707 \quad \Phi(\Pi)_{\text{No}} = \{(\phi, x_1, \dots, x_m) : \phi(\mathcal{X}_\Pi(x_1), \dots, \mathcal{X}_\Pi(x_m)) = 0\}$$

708 **► Theorem 40.** SZK_L is closed under $\leq_{\text{bf-tt}}^L$ reductions.

709 To begin the proof of this theorem, we first note that as in the proof of [23, Lemma 4.10],
 710 given two SD_{BP} pairs, we can create a new pair which is in $\text{SD}_{\text{BP}, \text{No}}$ if both of the original
 711 two pairs are (which we will use to compute ANDs of queries.) We can also compute in
 712 logspace the OR query for two queries by creating a pair $(P_1 \otimes S_1, P_2 \otimes S_2)$. We prove that
 713 these operations produce an output with the correct statistical difference with the following
 714 two claims:

715 **► Claim 41.** $\{(y_1, y_2) | \mathcal{X}_{\text{SD}_{\text{BP}}}(y_1) \vee \mathcal{X}_{\text{SD}_{\text{BP}}}(y_2) = 1\} \leq_m^L \text{SD}_{\text{BP}}$.

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

$$718 \quad (A_i, B_i) \in \text{SD}_{\text{BP}, Y} \implies \Delta(A_i, B_i) > 1 - p$$

$$719 \quad (A_i, B_i) \in \text{SD}_{\text{BP}, N} \implies \Delta(A_i, B_i) < p$$

721 Then consider:

$$722 \quad y = (A_1 \otimes A_2, B_1 \otimes B_2)$$

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

- 724 ■ YES: $\Delta(A_1 \otimes A_2, B_1 \otimes B_2) \geq \max\{\Delta(A_1 \otimes B_2, B_1 \otimes B_2), \Delta(B_1 \otimes A_2, B_1 \otimes B_2)\} =$
 725 $\max\{\Delta(A_1, B_1), \Delta(A_2, B_2)\} > 1 - p$.
- 726 ■ NO: $\Delta(A_1 \otimes A_2, B_1 \otimes B_2) \leq \Delta(A_1, B_1) + \Delta(A_2, B_2) < 2p$.

727 The second equality is from [23, Fact 2.3]. \blacktriangleleft

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

$$731 \quad < 2^{\frac{d+1}{2}} \frac{1}{2^m} = \frac{m^{O(1)}}{2^m} < 1/3.$$

732 \triangleright Claim 42. $\{(y_1, y_2) | \mathcal{X}_{\text{SD}_{\text{BP}}}(y_1) \wedge \mathcal{X}_{\text{SD}_{\text{BP}}}(y_2) = 1\} \leq_m^L \text{SD}_{\text{BP}}.$

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

735 $(A_i, B_i) \in \text{SD}_{\text{BP}, Y} \implies \Delta(A_i, B_i) > 1 - p$

736

737 $(A_i, B_i) \in \text{SD}_{\text{BP}, N} \implies \Delta(A_i, B_i) < p$

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

739 $\Delta(A_1, B_1) \cdot \Delta(A_2, B_2)$

740 The pair (A, B) we construct is analogous to (Q_0, Q_1) in Lemma 37, and can be created
741 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
742 $b_0 = 1$, while $B = (A_1, B_2)$ if b_2 is 0 and (A_2, B_1) if $b_1 = 1$.

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

744 ■ YES: $\Delta(A_1, B_1) \cdot \Delta(A_2, B_2) > (1 - p)^2$.

745 ■ NO: $\Delta(A_1, B_1) \cdot \Delta(A_2, B_2) \leq \max\{\Delta(A_1, B_1), \Delta(A_2, B_2)\} < p$.

746

◀

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

$$750 > (1 - \frac{1}{2^m})^{2^{\frac{d+1}{2}}} = (1 - \frac{1}{2^m})^{m^{O(1)}} > (1 - \frac{1}{2^m})^{2^m/m} \approx (\frac{1}{e})^{1/m} > \frac{2}{3}.$$

751 **Proof.** (of Theorem 40) Now suppose that we are given a promise problem Π such that
752 $\Pi \leq_{\text{bf-tt}}^L \text{SD}_{\text{BP}}$. We want to show $\Pi \leq_m^L \text{SD}_{\text{BP}}$, which by SZK_L 's closure under \leq_m^L reductions
753 implies $\Pi \in \text{SZK}_L$.

754 We follow the steps below on input x to create an SD_{BP} instance (F_0, F_1) which is in
755 $\text{SD}_{\text{BP}, Y}$ if $x \in \Pi_Y$:

756 1. Run the L machine for the $\leq_{\text{bf-tt}}^L$ reduction on x to get queries (q_1, \dots, q_m) and the
757 formula ϕ .

758 2. Build ψ from ϕ using Lemma 38. Replace queries $\neg q_i$ that would be negated with the
759 reduction from $\text{SD}_{\text{BP}, Y}$ to $\text{SD}_{\text{BP}, N}$ on q_i , and then apply Lemma 35 (the Polarization
760 Lemma) with $k = n$ on these queries to get (y_1, \dots, y_k) . Pad the output bits of each
761 branching program so each branching program has m output bits.

762 3. Build the template tree T . At the leaf level, for each variable in ψ , we will plug in the
763 corresponding query y_i . By Lemma 38 the tree is full.

764 4. Given x and designated output position j of F_0 or F_1 , there is a logspace computation
765 which finds the original output bit from $y_1 \dots y_m$ that bit j was copied from. This machine
766 traverses down the template tree from the output bit and records the following:

767 ■ The node that the computation is currently at on the template tree, with the path
768 taken depending on j .

769 ■ The position of the random bits used to decide which path to take when we reach
770 nodes corresponding to AND.

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

773 For step 4, we give an algorithm $\text{Eval}(x, j, \psi, y_1, \dots, y_m)$ to compute the j th output bit of
 774 F_0 or F_1 on x , for a formula ψ satisfying the properties of Lemma 38, a list of SD_{BP} queries
 775 (y_1, \dots, y_m) , and j . Without loss of generality, we lay out the algorithm to compute only
 776 $F_0(x)$.

777 Outline of $\text{Eval}(x, j, \psi, y_1, \dots, y_m)$:

778 The idea is to compute the j th output bit of F_0 by recursively calculating which query
 779 output bit it was copied from. To do this, first notice that the AND and OR operations
 780 produce branching programs where each output bit is copied from exactly one output bit of
 781 one of the query branching programs, so composing these operations together tells us that
 782 every output bit in F_0 is copied from exactly one output bit from one query. By Lemma 38
 783 and our AND and OR operations preserving the number of output bits, we also have that
 784 if every BP has l output bits, F_0 will have $2^a l = |\psi| l$ output bits, where a is the depth of
 785 ψ . This can be used to recursively calculate which query the j th bit is from: for an OR
 786 gate, divide the output bits into fourths, and decide which fourth the j th bit falls into (with
 787 each fourth corresponding to one BP, or two fourths corresponding to a subtree.) For an
 788 AND gate, divide the output into fourths, decide which fourth the j th bit falls into, and
 789 then use the 4 random bits for the XOR operation to compute which fourth corresponds to
 790 which branching programs (2 fourths will correspond to 1 BP or subtree, and the other 2
 791 fourths will correspond to the 2 BPs from the other subtree.) If j is updated recursively,
 792 then at the query level, we can directly return the j 'th output bit. This can be done in
 793 logspace, requiring a logspace path of “lefts” and “rights” to track the current gate, logspace
 794 to record and update j' , logspace to compute $2^a l$ at each level, and logspace to compute
 795 which subtree/query the output bit comes from at each level.

796 The resulting BP will be two distributions that will be in $\text{SD}_{\text{BP}, Y} \iff x \in \Pi_Y$. By this
 797 process $\Pi \leq_m^L \text{SD}_{\text{BP}}$. ◀

798 Acknowledgments

799 EA and HT were supported in part by NSF Grants CCF-1909216 and CCF-1909683. This
 800 work was carried out while JG, SM, and PW were participants in the 2022 DIMACS REU
 801 program at Rutgers University, supported by NSF grants CNS-215018 and CCF-1852215.
 802 We thank Yuval Ishai for helpful conversations about SREN, and we thank Markus Lohrey,
 803 Sam Buss, and Dave Barrington for useful discussions about Lemma 38.

804 References

- 805 1 Eric Allender, John Gouwar, Shuichi Hirahara, and Caleb Robelle. Cryptographic hardness
 806 under projections for time-bounded Kolmogorov complexity. *Theoretical Computer Science*,
 807 940:206–224, 2023. doi:10.1016/j.tcs.2022.10.040.
- 808 2 Eric Allender, Shuichi Hirahara, and Harsha Tirumala. Kolmogorov complexity characterizes
 809 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
 810 Informatik, 2023. doi:10.4230/LIPIcs.ITCS.2023.3.
- 812 3 Eric Allender and Ian Mertz. Complexity of regular functions. *Journal of Computer and
 813 System Sciences*, 104:5–16, 2019. Language and Automata Theory and Applications - LATA
 814 2015. doi:https://doi.org/10.1016/j.jcss.2016.10.005.
- 815 4 Eric Allender, Klaus Reinhardt, and Shiyu Zhou. Isolation, matching, and counting uniform
 816 and nonuniform upper bounds. *Journal of Computer and System Sciences*, 59(2):164–181,
 817 1999. doi:https://doi.org/10.1006/jcss.1999.1646.

818 5 Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0 . *SIAM Journal*
 819 *on Computing*, 36(4):845–888, 2006. doi:10.1137/S0097539705446950.

820 6 V. Arvind and T. C. Vijayaraghavan. Classifying problems on linear congruences and abelian
 821 permutation groups using logspace counting classes. *computational complexity*, 19(1):57–98,
 822 November 2009. doi:10.1007/s00037-009-0280-6.

823 7 Samuel R. Buss. The Boolean formula value problem is in ALOGTIME. In *Proceedings of the*
 824 *19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 123–131. ACM, 1987.
 825 doi:10.1145/28395.28409.

826 8 Samuel R Buss. Algorithms for Boolean formula evaluation and for tree contraction. *Arithmetic,*
 827 *Proof Theory, and Computational Complexity*, 23:96–115, 1993.

828 9 Ronald Cramer, Serge Fehr, Yuval Ishai, and Eyal Kushilevitz. Efficient multi-party com-
 829 putation over rings. In *Proc. International Conference on the Theory and Applications of*
 830 *Cryptographic Techniques; Advances in Cryptology (EUROCRYPT)*, volume 2656 of *Lecture*
 831 *Notes in Computer Science*, pages 596–613. Springer, 2003. doi:10.1007/3-540-39200-9_37.

832 10 Zeev Dvir, Dan Gutfreund, Guy N Rothblum, and Salil P Vadhan. On approximating the
 833 entropy of polynomial mappings. In *Second Symposium on Innovations in Computer Science*,
 834 pages 460–475. Tsinghua University Press, 2011.

835 11 Moses Ganardi and Markus Lohrey. A universal tree balancing theorem. *ACM Transactions*
 836 *on Computation Theory*, 11(1):1:1–1:25, 2019. doi:10.1145/3278158.

837 12 Oded Goldreich, Amit Sahai, and Salil Vadhan. Can statistical zero knowledge be made
 838 non-interactive? or On the relationship of SZK and NISZK. In *Annual International Cryptology*
 839 *Conference*, pages 467–484. Springer, 1999. doi:10.1007/3-540-48405-1_30.

840 13 Oded Goldreich, Amit Sahai, and Salil P. Vadhan. Honest-verifier statistical zero-knowledge
 841 equals general statistical zero-knowledge. In *Proceedings of the 30th Annual ACM Symposium on*
 842 *the Theory of Computing (STOC)*, pages 399–408. ACM, 1998. doi:10.1145/276698.276852.

843 14 Ulrich Hertrampf, Steffen Reith, and Heribert Vollmer. A note on closure properties of
 844 logspace MOD classes. *Information Processing Letters*, 75(3):91–93, 2000. doi:10.1016/
 845 S0020-0190(00)00091-0.

846 15 Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect
 847 randomizing polynomials. In *Proc. International Conference on Automata, Languages, and*
 848 *Programming (ICALP)*, volume 2380 of *Lecture Notes in Computer Science*, pages 244–256.
 849 Springer, 2002. doi:10.1007/3-540-45465-9_22.

850 16 Richard M. Karp, Eli Upfal, and Avi Wigderson. Constructing a perfect matching is in random
 851 NC . *Combinatorica*, 6(1):35–48, 1986. doi:10.1007/BF02579407.

852 17 Nathan Linial, Yishay Mansour, and Noam Nisan. Constant depth circuits, Fourier transform,
 853 and learnability. *J. ACM*, 40(3):607–620, 1993. doi:10.1145/174130.174138.

854 18 Pierre McKenzie and Stephen A. Cook. The parallel complexity of Abelian permutation group
 855 problems. *SIAM Journal on Computing*, 16(5):880–909, 1987. doi:10.1137/0216058.

856 19 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix
 857 inversion. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*
 858 (*STOC*), pages 345–354. ACM, 1987. doi:10.1145/28395.383347.

859 20 Tatsuaki Okamoto. On relationships between statistical zero-knowledge proofs. *Journal of*
 860 *Computer and System Sciences*, 60(1):47–108, 2000. doi:10.1006/jcss.1999.1664.

861 21 Chris Peikert and Vinod Vaikuntanathan. Noninteractive statistical zero-knowledge proofs
 862 for lattice problems. In *Proc. Advances in Cryptology: 28th Annual International Cryptology*
 863 *Conference (CRYPTO)*, volume 5157 of *Lecture Notes in Computer Science*, pages 536–553.
 864 Springer, 2008. doi:10.1007/978-3-540-85174-5_30.

865 22 Vishal Ramesh, Sasha Sami, and Noah Singer. Simple reductions to circuit minimization:
 866 DIMACS REU report. Technical report, DIMACS, Rutgers University, 2021. Internal
 867 document.

868 23 Amit Sahai and Salil P. Vadhan. A complete problem for statistical zero knowledge. *J. ACM*,
 869 50(2):196–249, 2003. doi:10.1145/636865.636868.

870 24 Heribert Vollmer. *Introduction to circuit complexity: a uniform approach*. Springer Science &
871 Business Media, 1999. doi:10.1007/978-3-662-03927-4.