Batch Verification for Statistical Zero Knowledge Proofs

Inbar Kaslasi[1(✉)], Guy N. Rothblum[2], Ron D. Rothblum[1], Adam Sealfon[3], and Prashant Nalini Vasudevan[3]

[1] Technion - Israel Institute of Technology, Haifa, Israel
{inbark,rothblum}@cs.technion.ac.il
[2] Weizmann Institute, Rehovot, Israel
rothblum@alum.mit.edu
[3] UC Berkeley, Berkeley, USA
{asealfon,prashvas}@berkeley.edu

Abstract. A statistical zero-knowledge proof (SZK) for a problem Π enables a computationally unbounded prover to convince a polynomial-time verifier that $x \in \Pi$ without revealing any additional information about x to the verifier, in a strong information-theoretic sense.

Suppose, however, that the prover wishes to convince the verifier that k separate inputs $x_1, \dots, x_k$ all belong to Π (without revealing anything else). A naive way of doing so is to simply run the SZK protocol separately for each input. In this work we ask whether one can do better – that is, is efficient *batch verification* possible for SZK?

We give a partial positive answer to this question by constructing a batch verification protocol for a natural and important subclass of SZK – all problems Π that have a *non-interactive* SZK protocol (in the common random string model). More specifically, we show that, for every such problem Π, there exists an honest-verifier SZK protocol for batch verification of k instances, with communication complexity $\mathsf{poly}(n) + k \cdot \mathsf{poly}(\log n, \log k)$, where poly refers to a fixed polynomial that depends only on Π (and not on k). This result should be contrasted with the naive solution, which has communication complexity $k \cdot \mathsf{poly}(n)$.

Our proof leverages a new NISZK-complete problem, called *Approximate Injectivity*, that we find to be of independent interest. The goal in this problem is to distinguish circuits that are nearly injective, from those that are non-injective on almost all inputs.

1 Introduction

Zero-knowledge proofs, introduced in the seminal work of Goldwasser, Micali and Rackoff [GMR89], are a remarkable and incredibly influential notion. Loosely speaking, a zero-knowledge proof lets a prover P convince a verifier V of the validity of some statement without revealing any additional information.

In this work we focus on *statistical zero-knowledge proofs*. These proof-systems simultaneously provide unconditional *soundness* and *zero-knowledge*:

The full version is available on ECCC [KRR+20].

R. Pass and K. Pietrzak (Eds.): TCC 2020, LNCS 12551, pp. 139–167, 2020.
https://doi.org/10.1007/978-3-030-64378-2_6

- Even a *computationally unbounded* prover P^* cannot convince V to accept a false statement (except with some negligible probability).
- Any efficient, but potentially malicious, verifier V^* learns nothing in the interaction (beyond the validity of the statement) in the following strong, statistical, sense: there exists an algorithm, called the simulator, which can efficiently simulate the entire interaction between V^* and P based only on the input x, so that the simulation is indistinguishable from the real interaction even to a *computationally unbounded* distinguisher.

The class of promise problems[1] having a statistical zero-knowledge proof is denoted by SZK. This class contains many natural problems, including many of the problems on which modern cryptography is based, such as (relaxations of) integer factoring [GMR89], discrete logarithm [GK93, CP92] and lattice problems [GG00, MV03, PV08, APS18].

Since the study of SZK was initiated in the early 80's many surprising and useful structural properties of this class have been discovered (see, e.g., [For89, AH91, Oka00, SV03, GSV98, GV99, NV06, OV08]), and several applications have been found for hard problems in this (and related) classes (for example, see [Ost91, OW93, BDRV18a, BDRV18b, KY18, BBD+20]). It is known to be connected to various cryptographic primitives [BL13, KMN+14, LV16, PPS15] and algorithmic and complexity-theoretic concepts [Dru15], and has consequently been used to show conditional impossiblility results. In particular, a notable and highly influential development was the discovery of natural complete problems for SZK [SV03, GV99].

In this work we are interested in the following natural question. Suppose that a particular problem Π has an SZK protocol. This means that there is a way to efficiently prove that $x \in \Pi$ in zero-knowledge. However, in many scenarios, one wants to be convinced not only that a single instance belongs to Π but rather that k different inputs $x_1, \ldots, x_k$ all belong to Π. One way to do so is to simply run the underlying protocol for Π k times, in sequence, once for each input x_i.[2] However, it is natural to ask whether one can do better. In particular, assuming that the SZK protocol for Π has communication complexity m, can one prove (in statistical zero-knowledge) that $x_1, \ldots, x_k \in \Pi$ with communication complexity $\ll k \cdot m$? We refer to this problem as *batch verification for* SZK.

We view batch verification of SZK as being of intrinsic interest, and potentially of use in the study of the structure of SZK. Beyond that, batch verification of SZK may be useful to perform various cryptographic tasks, such as batch verification of digital signature schemes [NMVR94, BGR98, CHP12] or batch verification of well-formedness of public keys (see, e.g., [GMR98]).

[1] Recall that a promise problem Π consists of two ensembles of sets $\mathrm{YES} = (\mathrm{YES}_n)_{n\in\mathbb{N}}$ and $(\mathrm{NO}_n)_{n\in\mathbb{N}}$, such that the YES_n's and NO_n's are disjoint. Instances in YES are called YES instances and those in NO are called NO instances.

[2] The resulting protocol can be shown to be zero-knowledge (analogously to the fact that *sequential* repetition preserves statistical zero-knowledge).

1.1 Our Results

We show that non-trivial batch verification is possible for a large and natural subset of languages in SZK. Specifically, we consider the class of promise problems having *non-interactive statistical zero-knowledge proofs.* A non-interactive statistical zero-knowledge proof [BFM88] is a variant of SZK in which the verifier and the prover are given access to a uniformly random *common random string* (CRS). Given this CRS and an input x, the prover generates a proof string π which it sends to the verifier. The verifier, given x, the CRS, and the proof string π, then decides whether to accept or reject. In particular, no additional interaction is allowed other than the proof π. Zero-knowledge means that it is possible to simulate the verifier's view (which consists of the CRS and proof π) so that the simulation is *statistically* indistinguishable from the real interaction. The corresponding class of promise problems is abbreviated as NISZK.

Remark 1.1. *An* NISZK *for a problem Π is equivalent to a two-round public-coin* honest-verifier SZK *protocol. Recall that* honest-verifier *zero-knowledge, means that the* honest verifier *learns essentially nothing in the interaction, but a* malicious verifier *may be able to learn non-trivial information.*

The class NISZK contains many natural and basic problems such as: variants of the quadratic residuosity problem [BSMP91, DSCP94], lattice problems [PV08, APS18], etc. It is also known to contain *complete* problems [SCPY98, GSV99], related to the known complete problems for SZK.

Our main result is an honest-verifier statistical zero-knowledge protocol for batch verification of any problem in NISZK. In order to state the result more precisely, we introduce the following definition.

Definition 1.2. *Let $\Pi = (\mathrm{YES}, \mathrm{NO})$ be a promise problem, where $\mathrm{YES} = (\mathrm{YES}_n)_{n\in\mathbb{N}}$ and $\mathrm{NO} = (\mathrm{NO}_n)_{n\in\mathbb{N}}$, and let $k = k(n) \in \mathbb{N}$. We define the promise problem $\Pi^{\otimes k} = \left(\mathrm{YES}^{\otimes k}, \mathrm{NO}^{\otimes k}\right)$, where $\mathrm{YES}^{\otimes k} = (\mathrm{YES}_n^{\otimes k})_{n\in\mathbb{N}}$, $\mathrm{NO}^{\otimes k} = (\mathrm{NO}_n^{\otimes k})_{n\in\mathbb{N}}$ and*

$$\mathrm{YES}_n^{\otimes k} = (\mathrm{YES}_n)^k$$

and

$$\mathrm{NO}_n^{\otimes k} = (\mathrm{YES}_n \cup \mathrm{NO}_n)^k \setminus (\mathrm{YES}_n)^k .$$

That is, instances of $\Pi^{\otimes k}$ are k instances of Π, where the YES instances are all in YES and the NO instances consist of at least one NO instances for Π.[3]

With the definition of $\Pi^{\otimes k}$ in hand, we are now ready to formally state our main result:

[3] This notion of composition is to be contrasted with that employed in the closure theorems for SZK under composition with formulas [SV03]. There, a composite problem similar to $\Pi^{\otimes k}$ is considered that does not require in its NO sets that all k instances satisfy the promise, but instead just that at least one of the instances is a NO instance of Π.

Theorem 1.3 (Informally Stated, see Theorem 3.1). *Suppose that* $\Pi \in$ NISZK*. Then, for every* $k = k(n) \in \mathbb{N}$*, there exists an (interactive)* honest-verifier SZK *protocol for* $\Pi^{\otimes k}$ *with communication complexity* $\mathsf{poly}(n) + k \cdot \mathsf{poly}(\log n, \log k)$*, where* n *refers to the length of a single instance and* poly *refers to a fixed polynomial independent of* k*.*

The verifier's running time is $k \cdot \mathsf{poly}(n)$ *and the number of rounds is* $O(k)$*.*

We emphasize that our protocol for $\Pi^{\otimes k}$ is interactive and *honest-verifier* statistical zero-knowledge (HVSZK). Since we start with an NISZK protocol (which as mentioned above is a special case of HVSZK), it is somewhat expected that the resulting batch verification protocol is only HVSZK. Still, obtaining a similar result to Theorem 1.3 that achieves *malicious-verifier* statistical zero-knowledge is a fascinating open problem (see Sect. 1.4 for additional open problems). We mention that while it is known [GSV98] how to transform any HVSZK protocol into a full-fledged SZK protocol (i.e., one that is zero-knowledge even wrt a malicious verifier), this transformation incurs a polynomial overhead that we cannot afford.

1.2 Related Works

Batch Verification via **IP** = PSPACE*.* A domain in which batch computing is particularly easy is bounded space computation - if a language $\mathcal{L}$ can be decided in space s then k instances of $\mathcal{L}$ can be solved in space $s + \log(k)$ (by reusing space). Using this observation, the **IP** = PSPACE theorem [LFKN92,Sha92] yields an efficient interactive proof for batch verification of any problem in PSPACE. However, the resulting protocol has several major drawbacks. In particular, it does not seem to preserve zero-knowledge, which makes it unsuitable for the purposes of our work.

Batch Verification with Efficient Prover. Another caveat of the **IP** = PSPACE approach is that it does not preserve the efficiency of the prover. That is, even if we started with a problem that has an interactive proof with an *efficient prover*, the batch verification protocol stemming from the **IP** = PSPACE theorem has an *inefficient* prover.

Reingold *et al.* [RRR16,RRR18] considered the question of whether batch verification of NP proofs with an efficiency prover is possible, assuming that the prover is given the NP witnesses as an auxiliary input. These works construct such an interactive batch verification protocol for all problems in UP $\subseteq$ NP (i.e., languages in NP in which YES instances have a unique proof). In particular, the work of [RRR18] yields a batch verification protocol for UP with communication complexity $k^{\delta} \cdot \mathsf{poly}(m)$, where m is the original UP witness length and $\delta > 0$ is any constant.

Note that it seems unlikely that the [RRR16,RRR18] protocols preserve zero-knowledge. Indeed, these protocols fundamentally rely on the so-called *unambiguity* (see [RRR16]) of the underlying UP protocol, which, at least intuitively, seems at odds with zero-knowledge.

Batch Verification with Computational Soundness. Focusing on protocols achieving only *computational soundness*, we remark that interactive batch verification can be obtained directly from Kilian's [Kil92] highly efficient protocol for all of NP (assuming collision resistant hash functions). A *non-interactive* batch verification protocol was given by Brakerski *et al.* [BHK17] assuming the hardness of learning with errors. Non-interactive batch verification protocols also follow from the existence of *succinct non-interactive zero-knowledge arguments (zkSNARGs)*, which are known to exist under certain strong, and non-falsifiable, assumptions (see, e.g. [Ish], for a recent survey).

Randomized Iterates. The *randomized iterate* is a concept introduced by Goldreich, Krawczyk, and Luby [GKL93], and further developed by later work [HHR11, YGLW15], who used it to construct pseudorandom generators from regular one-way functions. Given a function f, its randomized iterate is computed on an input x and descriptions of hash functions $h_1, \dots, h_m$ by starting with $x_0 = f(x)$ and iteratively computing $x_i = f(h_i(x_{i-1}))$. The hardcore bits of these iterates were then used to obtain pseudorandomness. While the randomized iterate was used for a very different purpose, this process of alternating the evaluation of a given function with injection of randomness (which is what the hash functions were for) is strongly reminiscent of our techniques. It would be very interesting if there is a deeper connection between our techniques and the usage of these iterates in relation to pseudorandom generators.

1.3 Technical Overview

Batch Verification for Permutations. As an initial toy example, we first consider batch verification for a specific problem in NISZK. Let PERM be the promise problem defined as follows. The input to PERM is a description of a Boolean circuit $C : \{0,1\}^n \to \{0,1\}^n$. The YES inputs consist of circuits that define permutations over $\{0,1\}^n$ whereas the NO inputs are circuits so that every element in the image has at least two preimages.[4] It is straightforward to see that PERM $\in$ NISZK.[5]

Our goal is, given as input k circuits $C_1, \dots, C_k$, to distinguish (via a zero-knowledge proof) the case that all of the circuits are permutations from the case

[4] PERM can be thought of as a variant of the *collision problem* (see [Aar04, Chapter 6]) in which the goal is to distinguish a permutation from a 2-to-1 function.

[5] A two round public-coin honest-verifier *perfect* zero-knowledge protocol for PERM can be constructed as follows. The verifier sends a random string $y \in \{0,1\}^n$ and the prover sends $x = C^{-1}(y)$. The verifier needs to check that indeed $y = C(x)$. It is straightforward to check that this protocol is *honest-verifier* perfect zero-knowledge and has soundness 1/2, which can be amplified by parallel repetition (while noting that honest-verifier zero-knowledge is preserved under parallel repetition).

This protocol can be viewed as a NIPZK by viewing the verifier's coins as the common random string. On the other hand, assuming that NISZK $\neq$ NIPZK, PERM is not NISZK-complete.

that one or more is 2-to-1. Such a protocol can be constructed as follows: the verifier chooses at random $x_1 \in \{0,1\}^n$, computes $x_{k+1} = C_k(C_{k-1}(\dots C_1(x_1)\dots))$ and sends x_{k+1} to the prover. The prover now responds with the preimage $x_1' = C_1^{-1}(C_2^{-1}(\dots C_k^{-1}(x_{k+1})\dots))$. The verifier checks that $x_1 = x_1'$ and if so it accepts, otherwise it rejects.[6]

Completeness follows from the fact that the circuits define permutations and so $x_1 = x_1'$. For soundness, observe that for a NO instance, x_{k+1} has at least two preimages under the composed circuit $C_k \circ \dots \circ C_1$. Therefore, a cheating prover can guess the correct preimage x_0 with probability at most 1/2 (and the soundness error can be reduced by repetition). Lastly observe that the protocol is (perfect) honest-verifier zero-knowledge: the simulator simply emulates the verifier while setting $x_1' = x_1$.

The Approximate Injectivity Problem. Unfortunately, as mentioned above, PERM is presumably not NISZK-complete and so we cannot directly use the above protocol to perform batch verification for arbitrary problems in NISZK. Instead, our approach is to identify a relaxation of PERM that is both NISZK-complete and amenable to batch verification, albeit via a significantly more complicated protocol.

More specifically, we consider the *Approximate Injectivity* (promise) problem. The goal here is to distinguish circuits that are almost injective, from ones that are highly non-injective. In more detail, let $\delta \in [0,1]$ be a parameter. We define AI_δ to be a promise problem in which the input is again a description of a Boolean circuit C mapping n input bits to $m \geq n$ output bits. YES instances are those circuits for which all but δ fraction of the inputs x have no collisions (i.e., $\Pr_x[|C^{-1}(C(x))| > 1] < \delta$). NO instances are circuits for which all but δ fraction of the inputs have at least one colliding input (i.e., $\Pr_x[|C^{-1}(C(x))| = 1] < \delta$).

Our protocol for batch verification of any problem $\Pi \in$ NISZK consists of two main steps:

- First, we show that AI_δ is NISZK-hard: i.e., there exists an efficient Karp reduction from Π to AI_δ.
- Our second main step is showing an efficient HVSZK batch verification protocol for AI_δ. In particular, the communication complexity of the protocol scales (roughly) additively with the number of instances k.

Equipped with the above, an HVSZK protocol follows by having the prover and verifier reduce the instances $x_1, \dots, x_k$ for Π to instances $C_1, \dots, C_k$ for AI_δ, and then engage in the batch verification protocol for AI_δ on common input $(C_1, \dots, C_k)$.

[6] A related but slightly different protocol, which will be less useful in our eventual construction, can be obtained by observing that (1) the mapping $(C_1, \dots, C_k) \mapsto C_k \circ \dots \circ C_1$ is a Karp-reduction from an instance of $\mathsf{PERM}^{\otimes k}$ to an instance of PERM with n input/output bits, and (2) that PERM has an NISZK protocol with communication complexity that depends only on n.

Before describing these two steps in detail, we remark that we find the identification of AI_δ as being NISZK-hard (in fact, NISZK-complete) to be of independent interest. In particular, while AI_δ bears some resemblance to problems that were already known to be NISZK-complete, the special almost-injective nature of the YES instances of AI_δ seems very useful. Indeed, this additional structure is crucial for our batch verification protocol.

AI_δ *is* NISZK-*hard.* We show that AI_δ is NISZK-hard by reducing to it from the Entropy Approximation problem (EA), which is known to be complete for NISZK [GSV99].[7] An instance of EA is a circuit C along with a threshold $k \in \mathbb{R}^+$, and the problem is to decide whether the Shannon entropy of the output distribution of C when given uniformly random inputs (denoted $H(C)$) is more than $k+10$ or less than $k-10$.[8]

For simplicity, suppose we had a stronger promise on the output distribution of C — that it is a flat distribution (in other words, it is uniform over some subset of its range). In this case, for any output y of C, the promise of EA tells us something about the *number of pre-images* of y. To illustrate, suppose C takes n bits of input. Then, in a YES instance of EA, the size of the set $\left|C^{-1}(y)\right|$ is at most $2^{n-(k+10)}$, and in the NO case it is at least $2^{n-(k-10)}$. Recall that for a reduction to AI_δ, we need to make the sizes of most inverse sets 1 for YES instances and more than 1 for NO instances. This can now be done by using a hash function to shatter the inverse sets of C.

That is, consider the circuit $\widehat{C}$ that takes as input an x and also the description of a hash function h from, say, a pairwise-independent hash family H, and outputs $(C(x), h, h(x))$. If we pick H so that its hash functions have output length $(n-k)$, then out of any set of inputs of size $2^{n-(k+10)}$, all but a small constant fraction will be mapped injectively by a random h from H. On the other hand, out of any set of inputs of size $2^{n-(k-10)}$, only a small constant fraction will be mapped injectively by a random h. Thus, in the YES case, it may be argued that all but a small constant fraction of inputs (x, h) are mapped injectively by $\widehat{C}$, and in the NO case only a small constant fraction of inputs are. So for some constant δ, this is indeed a valid reduction from EA to AI_δ. For smaller functions δ, the reduction is performed by first amplifying the gap in the promise of EA and then proceeding as above.

Finally, we can relax the simplifying assumption of flatness using the asymptotic equipartition property of distributions. In this case, this property states that, however unstructured C may be, its t-fold repetition $C^{\otimes t}$ (that takes an input tuple $(x_1, \ldots, x_t)$ and outputs $(C(x_1), \ldots, C(x_t))$) is "approximately flat" for large enough t. That is, with increasingly high probability over the output distribution of $C^{\otimes t}$, a sample from it will have a pre-image set of size close to its expectation, which is $2^{t\cdot(n-H(C))}$. Such techniques have been previously used for similar purposes in the SZK literature and elsewhere, for example as the

[7] In fact, we also show that AI_δ is in NISZK, and thus is NISZK-complete, by reducing back from it to EA.

[8] In the standard definition of EA [GSV99], the promise is that $H(C)$ is either more than $k+1$ or less than $k-1$, but this gap can be amplified easily by repetition of C.

flattening lemma of Goldreich *et al.* [GSV99] (see the full version for details and the proof).

Batch Verification for Exact Injectivity. For sake of simplicity, for this overview we focus on batch verification of the *exact* variant of AI_δ, that is, when $\delta = 0$. In other words, distinguishing circuits that are truly injective from those in which every image y has at least two preimages (with no exceptions allowed). We refer to this promise problem as INJ. Modulo some technical details, the batch verification protocol for INJ, presented next, captures most of the difficulty in our batch verification protocol for AI_δ.

Before proceeding we mention that the key difference between INJ and PERM is that YES instances of the former are merely injective whereas for the latter they are permutations. Interestingly, this seemingly minor difference causes significant complications.

Our new goal is, given as input circuits $C_1, \ldots, C_k : \{0,1\}^n \to \{0,1\}^m$, with $m \geq n$, to distinguish the case that all of the circuits are injective from the case that at least one is entirely non-injective.

Inspired by the batch verification protocol for PERM, a reasonable approach is to choose x_1 at random but then try to hash the output $y_i = C_i(x_i) \in \{0,1\}^m$, of each circuit C_i, into an input $x_{i+1} \in \{0,1\}^n$ for C_{i+1}. If a hash function could be found that was injective on the image of C_i then we would be done. However, it seems that finding such a hash function is, in general, extremely difficult.

Rather, we will hash each y_i by choosing a random hash function from a small hash function family. More specifically, for every iteration $i \in [k]$ we choose a random seed z_i for a (seeded) randomness extractor $\mathsf{Ext} : \{0,1\}^m \times \{0,1\}^d \to \{0,1\}^n$ and compute $x_{i+1} = \mathsf{Ext}(y_i, z_i)$. See Fig. 1 for a diagram describing this sampling process.

In case all the circuits are injective (i.e., a YES instance), a simple inductive argument can be used to show that each y_i is (close to) a distribution having min-entropy n, and therefore the output $x_{i+1} = \mathsf{Ext}(y_i, z_i)$ of the extractor is close to uniform in $\{0,1\}^n$. Note that for this to be true, we need a very good extractor that essentially extracts *all of the entropy*. Luckily, constructions of such extractors with a merely poly-logarithmic seed length are known [GUV07].

This idea leads us to consider the following strawman protocol. The verifier chooses at random $x_1 \in \{0,1\}^n$ and k seeds $z_1, \ldots, z_k$. The verifier then computes inductively: $y_i = C_i(x_i)$ and $x_{i+1} = \mathsf{Ext}(y_i, z_i)$, for every $i \in [k]$. The verifier sends $(x_{k+1}, z_1, \ldots, z_k)$ to the prover, who in turn needs to guess the value of x_1.

The major difficulty that arises in this protocol is in completeness: the honest prover's probability of predicting x_1 is very small. To see this, suppose that all of the circuits $C_1, \ldots, C_k$ are injective. Consider the job of the honest prover: given $(x_{k+1}, z_1, \ldots, z_k)$ the prover needs to find x_1. The difficulty is that x_{k+1} is likely to have many preimages under $\mathsf{Ext}(\cdot, z_k)$. While this statement depends on the specific structure of the extractor, note that even in a "dream scenario" in which $\mathsf{Ext}(\cdot, z_k)$ were a random function, a constant fraction of x_{k+1}'s would have more than one preimage (in the image of C_k).

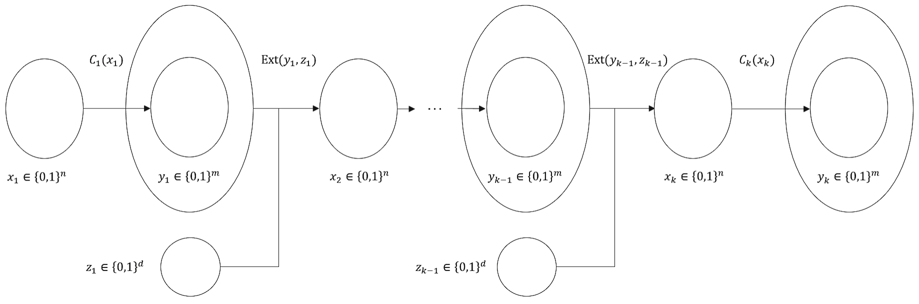

Fig. 1. The sampling process

A similar type of collision in the extractor is likely to occur in most of the steps $i \in [k]$. Therefore, the overall number of preimages x'_1 that are consistent with $(x_{k+1}, z_1, \ldots, z_k)$ is likely to be $2^{\Omega(k)}$ and the prover has no way to guess the correct one among them. The natural remedy for this is to give the prover some additional information, such as a hash of x_1, in order to help pick it correctly among the various possible options. However, doing so also helps a cheating prover find x_1 in the case where one of the circuits is non-injective. And it turns out that the distribution of the number of x_1's in the two cases – where all the C_i's are injective and where one is non-injective – are similar enough that it is not clear how to make this approach work as is.

Isolating Preimages via Interaction. We circumvent the issue discussed above by employing *interaction*. The key observation is that, even though the number of pre-images x_1 of the composition of all k circuits is somewhat similar in the case of all YES instance and the case of one NO instance among them, if we look at this composition circuit-by-circuit, the number of pre-images in an injective circuit is clearly different from that in a non-injective circuit. In order to exploit this, we have the verifier *gradually* reveal the y_i's rather than revealing them all at once.

Taking a step back, let us consider the following naive protocol:

1. For $i = k, \ldots, 1$:
   (a) The verifier chooses at random $x_i \in \{0,1\}^n$ and sends $y_i = C_i(x_i)$ to the prover.
   (b) The (honest) prover responds with $x'_i = C_i^{-1}(y_i)$.
   (c) The verifier immediately rejects if the prover answered incorrectly (i.e., $x'_i \neq x_i$).

It is not difficult to argue that this protocol is indeed an HVSZK protocol (with soundness error 1/2, which can be reduced by repetition). Alas, the communication complexity is at least $k \cdot n$, which is too large.

However, a key observation is that this protocol still works even if we generate the y_i's as in the strawman protocol. Namely, $x_{i+1} = \mathsf{Ext}(y_i, z_i)$ and $y_i = C_i(x_i)$, for every $i \in [k]$, where the z_i's are fresh uniformly distributed (short) seeds. This lets us significantly reduce the *randomness complexity* of the above naive protocol. Later we shall use this to also reduce the *communication complexity*, which is our main goal.

To see that the "derandomized" variant of the naive protocol works, we first observe that completeness and zero-knowledge indeed still hold. Indeed, since in a YES case all the circuits are injective, the honest prover always provides the correct answer - i.e., $x'_i = x_i$. Thus, not only does the verifier always accept (which implies completeness), but it can also easily simulate the messages sent by the prover (which guarantees honest-verifier statistical zero-knowledge).[9]

Arguing soundness is slightly more tricky. Let $i^* \in [k]$ be the smallest integer so that C_{i^*} is a NO instance. Recall that in the i^*-th iteration of the protocol, the prover is given y_{i^*} and needs to predict x_{i^*}. If we can argue that x_{i^*} is (close to) uniformly distributed then (constant) soundness follows, since C_{i^*} is a NO instance and therefore non-injective on every input.

We argue that x_{i^*} is close to uniform by induction. For the base case $i = 1$ this is obviously true since x_1 is sampled uniformly at random. For the inductive step, assume that x_{i-1} is close to uniform, with $i \leq i^*$. Since C_{i-1} is injective (since $i - 1 < i^*$), this means that $y_{i-1} = C_{i-1}(x_{i-1})$ is close to uniform in the image of C_1, a set of size 2^n. Thus, $\mathsf{Ext}(y_{i-1}, z_{i-1})$ is applied to a source (that is close to a distribution) with min-entropy n. Since Ext is an extractor, this means that x_i is close to uniform, concluding the induction.

Reducing Communication Complexity via Hashing. Although we have reduced the *randomness complexity* of the protocol, we have not reduced the *communication complexity* (which is still $k \cdot n$). We shall do so by, once more, appealing to hashing.

Let us first consider the verifier to prover communication. Using hashing, we show how the verifier can specify each y_i to the prover by transmitting only a *poly-logarithmic* number of bits. Consider, for example, the second iteration of the protocol. In this iteration the verifier is supposed to send y_{k-1} to the prover but can no longer afford to do so. Notice however that at this point the prover already knows x_k. We show that with all but negligible probability, the number of candidate pairs (y_{k-1}, z_{k-1}) that are consistent with x_k (and so that y_{k-1} is in the image of C_{i-1}) is very small. This fact (shown in Proposition 4.4 in Sect. 4), follows from the fact that Ext is an extractor with small seed length.[10] In more detail, we show that with all but negligible probability, the number of candidates is roughly (quasi-)polynomial. Thus, it suffices for the verifier to send a hash of poly-logarithmic length (e.g., using a pairwise independent hash

[9] Actually the protocol as described achieves perfect completeness and perfect honest-verifier zero-knowledge. However, the more general AI_δ problem will introduce some (negligible) statistical errors.

[10] This observation is simple in hindsight but we nevertheless find it somewhat surprising. In particular, it cannot be shown by bounding the expected number of collisions and applying Markov's inequality since the *expected* number of collisions in Ext is very large (see [Vad12, Problem 6.4]).

function) to specify the correct pair (y_{k-1}, z_{k-1}). This idea extends readily to all subsequent iterations.

Thus, we are only left with the task of reducing the communication from the prover to the verifier (which is currently $n \cdot k$). We yet again employ hashing. The observation is that rather than sending x_i in its entirety in each iteration, it suffices for the prover to send a short hash of x_i. The reason is that, in the case of soundness, when we reach iteration i^*, we know that y_i has two preimages: $x_i^{(0)}$ and $x_i^{(1)}$. The prover at this point has no idea which of the two is the correct one and so as long as the hashes of $x_i^{(0)}$ and $x_i^{(1)}$ differ, the prover will only succeed with probability 1/2. Thus, it suffices to use a pairwise independent hash function.

To summarize, after an initial setup phase in which the verifier specifies y_k and the different hash functions, the protocol simply consists of a "ping pong" of hash values between the verifier and the prover. In each iteration the verifier first reveals a hash of the pair (y_i, z_i), which suffices for the prover to fully recover y_i. In response, the prover sends a hash of x_i, which suffices to prove that the prover knows the correct preimage of y_i. For further details, a formal description of the protocol, and the proof, see Sect. 4.

1.4 Discussion and Open Problems

Theorem 1.3 gives a non-trivial batch verification protocol for any problem in NISZK. However, we believe that it is only the first step in the study of batch verification of SZK. In particular, and in addition to the question of obtaining *malicious verifier* zero-knowledge that was already mentioned, we point out several natural research directions:

1. As already pointed out, Theorem 1.3 only gives a batch verification protocol for problems in NISZK. Can one obtain a similar result for all of SZK?
   As a special interesting case, consider the problem of batch verification for the graph non-isomorphism problem: deciding whether or not there exists a pair of isomorphic graphs among k such pairs. Theorem 1.3 yields an efficient batch verification protocol for this problem under the assumption that the graphs have no non-trivial automorphisms. Handling the general case remains open and seems like a good starting point for a potential generalization of Theorem 1.3 to all of SZK.
2. Even though we started off with an NISZK protocol for Π, the protocol for $\Pi^{\otimes k}$ is highly interactive. As a matter of fact, the number of rounds is $O(k)$. Is there an NISZK batch verification protocol for any $\Pi \in \mathsf{NISZK}$?
3. While the communication complexity in the protocol for $\Pi^{\otimes k}$ only depends (roughly) *additively* on k, this additive dependence is still linear. Is a similar result possible with a sub-linear dependence on k?[11] For example, with $\mathsf{poly}(n, \log k)$ communication?

[11] While a linear dependence on k seems potentially avoidable, we note that a polynomial dependence on n seems inherent (even for just a single instance, i.e., when $k = 1$).

4. A different line of questioning follows from looking at prover efficiency. While in general one cannot expect provers in interactive proofs to be efficient, it is known that any problem in $\mathsf{SZK} \cap \mathsf{NP}$ has an SZK protocol where the honest prover runs in polynomial-time given the NP witness for the statement being proven [NV06]. Our transformations, however, make the prover quite inefficient. This raises the interesting question of whether there are batch verification protocols for languages in $\mathsf{SZK} \cap \mathsf{NP}$ (or even $\mathsf{NISZK} \cap \mathsf{NP}$) that are zero-knowledge and also preserve the prover efficiency. This could have interesting applications in, say, cryptographic protocols where the honest prover is the party that generated the instance in the first place and so has a witness for it (e.g., in a signature scheme where the signer wishes to prove the validity of several signatures jointly).

While the above list already raises many concrete directions for future work, one fascinating high-level research agenda that our work motivates is a *fine-grained* study of SZK. In particular, optimizing and improving our understanding of the concrete polynomial overheads in structural study of SZK.

Remark 1.4 (Using circuits beyond random sampling). *To the best of our knowledge, all prior works studying complete problems for* SZK *and* NISZK *only make a very restricted usage of the given input circuits. Specifically, all that is needed is the ability to generate random samples of the form* $(r, C(r))$*, where* r *is uniformly distributed random string and* C *is the given circuit (describing a probability distribution).*

In contrast, our protocol leverages the ability to feed a (hash of an) output of one circuit as an input to the next circuit. This type of adaptive usage escapes the "random sampling paradigm" described above. In particular, our technique goes beyond the (restrictive) black box model of Holenstein and Renner [HR05], who showed limitations for statistical distance polarization within this model (see also [BDRV19]).

1.5 Organization

We start with preliminaries in Sect. 2. The batch verification result for NISZK is formally stated in Sect. 3 and proved therein, based on results that are proved in the subsequent sections. In Sect. 4 we show a batch verification protocol for AI_δ. Due to lack of space, we defer the proof that AI_δ is NISZK-complete to the full version.

2 Preliminaries

2.1 Probability Theory Notation and Background

Given a random variable X, we write $x \leftarrow X$ to indicate that x is sampled according to X. Similarly, given a finite set S, we let $s \leftarrow S$ denote that s is selected according to the uniform distribution on S. We adopt the convention that when the same random variable occurs several times in an expression, all occurrences refer to a single sample. For example, $\Pr[f(X) = X]$ is defined to be the probability that when $x \leftarrow X$, we have $f(x) = x$. We write U_n to denote the random variable distributed uniformly over $\{0,1\}^n$. The support of a distribution D over a finite set U, denoted $Supp(D)$, is defined as $\{u \in U : D(u) > 0\}$.

The *statistical distance* of two distributions P and Q over a finite set U, denoted as $\Delta(P,Q)$, is defined as $\max_{S \subseteq U}(P(S) - Q(S)) = \frac{1}{2}\sum_{u \in U} |P(u) - Q(u)|$.

We recall some standard basic facts about statistical distance.

Fact 2.1 (Data processing inequality for statistical distance). *For any two distributions X and Y, and every (possibly randomized) process f:*

$$\Delta\left(f(X), f(Y)\right) \leq \Delta\left(X, Y\right)$$

Fact 2.2. *For any two distributions X and Y, and event E:*

$$\Delta\left(X, Y\right) \leq \Delta\left(X|_E, Y\right) + \Pr_X[\neg E],$$

where $X|_E$ denotes the distribution of X conditioned *on E.*

Proof. Let $p_u = \Pr[X = u]$ and $q_u = \Pr[Y = u]$. Also, let $p_{u|E} = \Pr_X[X = u|E]$ and $p_{u|\neg E} = \Pr_X[X = u|\neg E]$.

$$\begin{aligned}
\Delta\left(X, Y\right) &= \frac{1}{2}\sum_u \left|p_u - q_u\right| \\
&= \frac{1}{2}\sum_u \left|\Pr_X[E]\cdot p_{u|E} + \Pr_X[\neg E]\cdot \mathsf{p}_{u|\neg E} - \Pr_X[E]\cdot q_u - \Pr_X[\neg E]\cdot q_u\right| \\
&\leq \frac{1}{2}\sum_u \left(\left|\Pr_X[E]\cdot p_{u|E} - \Pr_X[E]\cdot q_u\right| + \left|\Pr_X[\neg E]\cdot p_{u|\neg E} - \Pr_X[\neg E]\cdot q_u\right|\right) \\
&= \Pr_X[E]\cdot \Delta\left(X|_E, Y\right) + \Pr_X[\neg E]\cdot \Delta\left(X|_{\neg E}, Y\right) \\
&\leq \Delta\left(X|_E, Y\right) + \Pr_X[\neg E].
\end{aligned}$$

We also recall Chebyshev's inequality.

Lemma 2.3 (Chebyshev's inequality). *Let X be a random variable. Then, for every $\alpha > 0$:*

$$\Pr\left[\left|X - E\left[X\right]\right| \geq \alpha\right] \leq \frac{Var\left[X\right]}{\alpha^2}.$$

2.2 Zero-Knowledge Proofs

We use $(\mathsf{P},\mathsf{V})(x)$ to refer to the *transcript* of an execution of an interactive protocol with prover P and verifier V on common input x. The $\mathsf{transcript}$ includes the input x, all messages sent by P to V in the protocol and the verifier's random coin tosses. We say that the transcript $\tau = (\mathsf{P},\mathsf{V})(x)$ is $\mathsf{accepting}$ if at the end of the corresponding interaction, the verifier accepts.

Definition 2.4 (HVSZK). *Let $c = c(n) \in [0,1]$, $s = s(n) \in [0,1]$ and $z = z(n) \in [0,1]$. An* Honest Verifier SZK Proof-System (HVSZK) with completeness error *c,* soundness error *s* and zero-knowledge error *z for a promise problem $\Pi = (\Pi_{\text{YES}}, \Pi_{\text{NO}})$, consists of a probabilistic polynomial-time verifier* V *and a computationally unbounded prover* P *such that following properties hold:*

- **Completeness:** *For any $x \in \Pi_{\text{YES}}$:*

$$\Pr[(\mathsf{P},\mathsf{V})(x) \text{ is accepting}] \geq 1 - c(|x|).$$

- **Soundness:** *For any (computationally unbounded) cheating prover* P^* *and any $x \in \Pi_{\text{NO}}$:*

$$\Pr[(\mathsf{P}^*,\mathsf{V})(x) \text{ is accepting}] \leq s(|x|).$$

- **Honest Verifier Statistical Zero Knowledge:** *There is a probabilistic polynomial-time algorithm* Sim *(called the* simulator*) such that for any $x \in \Pi_{\text{YES}}$:*

$$\Delta((\mathsf{P},\mathsf{V})(x), \mathsf{Sim}(x)) \leq z(|x|).$$

If the completeness, soundness and zero-knowledge errors are all negligible, we simply say that Π has an HVSZK protocol. We also use HVSZK to denote the class of promise problems having such an HVSZK protocol.

We also define *non-interactive zero knowledge proofs* as follows.

Definition 2.5 (NISZK). *Let $c = c(n) \in [0,1]$, $s = s(n) \in [0,1]$ and $z = z(n) \in [0,1]$. An* non-interactive statistical zero-knowledge proof (NISZK) with completeness error *c,* soundness error *s* andzero-knowledge error *z for a promise problem $\Pi = (\Pi_{\text{YES}}, \Pi_{\text{NO}})$, consists of a probabilistic polynomial-time verifier* V, *a computationally unbounded prover* P *and a polynomial $\ell = \ell(n)$ such that following properties hold:*

- **Completeness:** *For any $x \in \Pi_{\text{YES}}$:*

$$\Pr_{r \in \{0,1\}^{\ell(|x|)}}[\mathsf{V}(x,r,\pi) \text{ accepts}] \geq 1 - c(|x|),$$

where $\pi = \mathsf{P}(x,r)$.

- **Soundness:** *For any $x \in \Pi_{\text{NO}}$:*

$$\Pr_{r \in \{0,1\}^{\ell(|x|)}}[\exists \pi^* \text{ s.t. } \mathsf{V}(x,r,\pi^*) \text{ accepts}] \leq s(|x|).$$

- **Honest Verifier Statistical Zero Knowledge:** *There is a probabilistic polynomial-time algorithm* Sim *(called the* simulator*) such that for any $x \in \Pi_{\text{YES}}$:*

$$\Delta((U_\ell, \mathsf{P}(x, U_\ell)), \mathsf{Sim}(x)) \leq z(|x|).$$

As above, if the errors are negligible, we say that Π has a NISZK protocol and use NISZK to denote the class of all such promise problems.

2.3 Many-Wise Independent Hashing

Hash functions offering bounded independence are used extensively in the literature. We use a popular variant in which the output of the hash function is *almost* uniformly distributed on the different points. This relaxation allows us to save on the representation length of functions in the family.

Definition 2.6 (δ-almost ℓ-wise Independent Hash Functions). *For $\ell = \ell(n) \in \mathbb{N}$, $m = m(n) \in \mathbb{N}$ and $\delta = \delta(n) > 0$, a family of functions $\mathcal{F} = (\mathcal{F}_n)_n$, where $\mathcal{F}_n = \left\{ f : \{0,1\}^m \to \{0,1\}^n \right\}$ is δ-almost ℓ-wise independent if for every $n \in \mathbb{N}$ and distinct $x_1, x_2, \ldots, x_\ell \in \{0,1\}^m$ the distributions:*

- *$(f(x_1), \ldots, f(x_\ell))$, where $f \leftarrow \mathcal{F}_n$; and*
- *The uniform distribution over $(\{0,1\}^n)^\ell$,*

are δ-close in statistical distance.

When $\delta = 0$ we simply say that the hash function family is ℓ-wise independent. Constructions of (efficiently computable) many-wise hash function families with a very succinct representation are well known. In particular, when $\delta = 0$ we have the following well-known construction:

Lemma 2.7 (See, e.g., [Vad12, Section 3.5.5]). *For every $\ell = \ell(n) \in \mathbb{N}$ and $m = m(n) \in \mathbb{N}$ there exists a family of ℓ-wise independent hash functions $\mathcal{F}^{(\ell)}_{n,m} = \{f \colon \{0,1\}^m \to \{0,1\}^n\}$ where a random function from $\mathcal{F}^{(\ell)}_{n,m}$ can be selected using $O\big(\ell \cdot \max(n,m)\big)$ bits, and given a description of $f \in \mathcal{F}^{(\ell)}_{n.m}$ and $x \in \{0,1\}^m$, the value $f(x)$ can be computed in time $\mathsf{poly}(n,m,\ell)$.*

For $\delta > 0$, the seminal work of Naor and Naor [NN93] yields a highly succinct construction.

Lemma 2.8 ([NN93, Lemma 4.2]). *For every $\ell = \ell(n) \in \mathbb{N}$, $m = m(n) \in \mathbb{N}$ and $\delta = \delta(n) > 0$, there exists a family of δ-almost ℓ-wise independent hash functions $\mathcal{F}^{(\ell)}_{n,m} = \{f \colon \{0,1\}^m \to \{0,1\}^n\}$ where a random function from $\mathcal{F}^{(\ell)}_{n,m}$ can be selected using $O\big(\ell \cdot n + \log(m) + \log(1/\delta)\big)$ bits, and given a description of $f \in \mathcal{F}^{(\ell)}_{n.m}$ and $x \in \{0,1\}^m$, the value $f(x)$ can be computed in time $\mathsf{poly}(n,m,\ell,\log(1/\delta))$.*

2.4 Seeded Extractors

The min-entropy of a distribution X over a set $\mathcal{X}$ is defined as $H_\infty(X) = \min_{x \in \mathcal{X}} \log(1/\Pr[X = x])$. In particular, if $H_\infty(X) = k$ then, $\Pr[X = x] \leq 2^{-k}$, for every $x \in \mathcal{X}$.

Definition 2.9 ([NZ96]). *Let $k = k(n) \in \mathbb{N}$, $m = m(n) \in \mathbb{N}$, $d = d(n)$ and $\epsilon = \epsilon(n) \in [0,1]$. We say that the family of functions $\mathsf{Ext} = (\mathsf{Ext}_n)_{n \in \mathbb{N}}$, where $\mathsf{Ext}_n : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$, is a (k,ϵ)-extractor if for every $n \in \mathbb{N}$ and distribution X supported on $\{0,1\}^n$ with $H_\infty(X) \geq k$, it holds that $\Delta\big(\mathsf{Ext}(X, U_d), U_m\big) \leq \epsilon$, where U_d (resp., U_m) denotes the uniform distribution on d (resp., m) bit strings.*

Lemma 2.10 **([GUV07, Theorem 4.21]).** *Let $k = k(n) \in \mathbb{N}$, $m = m(n) \in \mathbb{N}$ and $\epsilon = \epsilon(n) \in [0,1]$ such that $k \leq n$, $m \leq k + d - 2\log(1/\epsilon) - O(1)$, $d = \log(n) + O(\log(k) \cdot \log(k/\epsilon))$ and the functions k, m and ϵ are computable in $\mathsf{poly}(n)$ time. Then, there exists a polynomial-time computable (k, ϵ)-extractor $\mathsf{Ext} = (\mathsf{Ext}_n)_{n\in\mathbb{N}}$ such that $\mathsf{Ext}_n : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$.*

3 Batch Verification for NISZK

In this section we formally state and prove our main result.

Theorem 3.1. *Let $\Pi \in \mathsf{NISZK}$ and $k = k(n) \in \mathbb{N}$ such that $k(n) = 2^{n^{o(1)}}$. Then, $\Pi^{\otimes k}$ has an $O(k)$-round HVSZK protocol with communication complexity $k \cdot \mathsf{poly}(\log n, \log k) + \mathsf{poly}(n)$ and verifier running time $k \cdot \mathsf{poly}(n)$.*

The proof of Theorem 3.1 is divided into two main steps:

1. As our first main step, we introduce a new NISZK-hard problem, called *approximate injectivity*. The problem is defined formally in Definition 3.2 below and its NISZK hardness is established by Lemma 3.3. Due to space restrictions, the proof of Lemma 3.3 is deferred to the full version.
2. The second step is constructing a batch verification protocol for approximate injectivity, as given in Theorem 3.4. The proof of Theorem 3.4 appears in Sect. 4.

We proceed to define the approximate injectivity problem and state its NISZK-hardness.

Definition 3.2. *Let $\delta = \delta(n) \to [0,1]$ be a function computable in $\mathsf{poly}(n)$ time. The* Approximate Injectivity problem *with* approximation δ, *denoted by AI_δ, is a promise problem* (YES, NO), *where* $\mathrm{YES} = (\mathrm{YES}_n)_{n\in\mathbb{N}}$ *and* $\mathrm{NO} = (\mathrm{NO}_n)_{n\in\mathbb{N}}$ *are sets defined as follows:*

$$\mathrm{YES}_n = \left\{ (1^n, C) \ : \ \Pr_{x \leftarrow \{0,1\}^n} \left[\left| C^{-1}(C(x)) \right| > 1 \right] < \delta(n) \right\}$$

$$\mathrm{NO}_n = \left\{ (1^n, C) \ : \ \Pr_{x \leftarrow \{0,1\}^n} \left[\left| C^{-1}(C(x)) \right| > 1 \right] > 1 - \delta(n) \right\}$$

where, in both cases, C is a circuit that takes n bits as input. The size *of an instance $(1^n, C)$ is n.*

Lemma 3.3. *Let $\delta = \delta(n) \in [0,1]$ be a non-increasing function such that $\delta(n) > 2^{-o(n^{1/4})}$. Then, AI_δ is NISZK-hard.*

As mentioned above, the proof of Lemma 3.3 appears in the full version. Our main technical result is a batch verification protocol for AI_δ.

Theorem 3.4. *For any $k = k(n) \in \mathbb{N}$, $\delta = \delta(n) \in [0, \frac{1}{100k^2}]$ and security parameter $\lambda = \lambda(n)$, the problem $\mathsf{AI}_\delta^{\otimes k}$ has an* HVSZK *protocol with communication complexity $O(n)+k\cdot\mathsf{poly}(\lambda, \log N, \log k)$, where N is an upper bound on the size of each of the given circuits (on n input bits). The completeness and zero-knowledge errors are $O(k^2 \cdot \delta + 2^{-\lambda})$ and the soundness error is a constant bounded away from* 1*.*

The verifier running time is $k \cdot \mathsf{poly}(N, \log k, \lambda)$ and the number of rounds is $O(k)$.

The proof of Theorem 3.4 appears in Sect. 4. With Lemma 3.3 and Theorem 3.4 in hand, the proof of Theorem 3.1 is now routine.

Proof (Proof of Theorem 3.1*).* Let $\Pi \in \mathsf{NISZK}$. We construct an HVSZK protocol for $\Pi^{\otimes k}$ as follows. Given as common input $(x_1, \ldots, x_k)$, the prover and verifier each first employ the Karp reduction of Lemma 3.3 to each instance to obtain circuits $(C_1, \ldots, C_k)$ wrt $\delta = \frac{1}{2^{\mathsf{poly}(\log n, \log k)}}$. The size of each circuit, as well as the number of inputs bits, is $\mathsf{poly}(n)$.

The parties then emulate a $\mathsf{poly}(\log n, \log k)$ parallel repetition of the SZK protocol of Theorem 3.4 on input $(C_1, \ldots, C_k)$ and security parameter $\lambda = \mathsf{poly}(\log n, \log k)$. Completeness, soundness and honest-verifier zero-knowledge follow directly from Lemma 3.3 and Theorem 3.4, with error $O(k \cdot \delta + 2^{-\lambda}) = negl(n, k)$, where we also use the fact that parallel repetition of interactive proofs reduces the soundness error at an exponential rate, and that parallel repetition preserves *honest verifier* zero-knowledge.

To analyze the communication complexity and verifier running time, observe that the instances C_i that the reduction of Lemma 3.3 generates have size $\mathsf{poly}(n)$. The batch verification protocol of Theorem 3.4 therefore has communication complexity $\mathsf{poly}(n) + k \cdot \mathsf{poly}(\log n, \log k)$ and verifier running time $k \cdot \mathsf{poly}(n)$.

4 Batch Verification for AI

In this section we prove Theorem 3.4 by constructing an HVSZK protocol for batch verification of the approximate injectivity problem AI_δ (see Definition 3.2 for the definition of AI_δ). For convenience, we restate Theorem 3.4 next.

Theorem 3.4. *For any $k = k(n) \in \mathbb{N}, \delta = \delta(n) \in [0, \frac{1}{100k^2}]$ and security parameter $\lambda = \lambda(n)$, the problem $\mathsf{AI}_\delta^{\otimes k}$ has an* HVSZK *protocol with communication complexity $O(n)+k$ poly $(\lambda, \log N, \log k)$, where N is an upper bound on the size of each of the given circuits (on n input bits). The completeness and zero-knowledge errors are $O(k^2 \cdot \delta + 2^{-\lambda})$ and the soundness error is a constant bounded away from 1.*

The verifier running time is $k \cdot \mathsf{poly}(N, \log k, \lambda)$ and the number of rounds is $O(k)$.

HVSZK Batch Verification Protocol for AI_δ

INPUT: Circuits $C_1, \ldots, C_k : \{0,1\}^n \to \{0,1\}^m$ and security parameter λ, where all circuits have size at most N, input length n, and output length $m \leq N$.

- Wlog we assume that all the circuits have the same output length $m \leq N$. This can be achieved by padding.

INGREDIENTS:

- Let $\mathsf{Ext} = \mathsf{Ext}_n$ be the explicit extractor from Lemma 2.10, where $\mathsf{Ext}_n : \{0,1\}^m \times \{0,1\}^d \to \{0,1\}^n$, so that Ext_n supports min-entropy $n-1$, has error $\epsilon = \frac{1}{k^2 \cdot 2^\lambda}$ and the seed length d is as guaranteed by Lemma 2.10.
- Let H_n be the explicit family of $\frac{1}{2^{2\lambda+d+2\log k}}$-almost pairwise-independent hash functions of Lemma 2.8, where $H_n : \{0,1\}^m \times \{0,1\}^d \to \{0,1\}^{2\lambda+d+2\log k}$ and d is the seed length of the extractor as specified above.
- Let G_n be the explicit family of pairwise-independent hash functions of Lemma 2.7, where $G_n : \{0,1\}^n \to \{0,1\}^\ell$ and $\ell = O(1)$ (e.g., $\ell = 3$ suffices).

THE PROTOCOL:

1. Setup for V:
   (a) Sample $h \leftarrow H_n$ and $g \leftarrow G_n$.
   (b) Sample $x_1 \leftarrow \{0,1\}^n$.
   (c) For $i = 1, ..., k$:
      i. Compute $y_i = C_i(x_i)$.
      ii. Sample $z_i \leftarrow \{0,1\}^d$.
      iii. Compute $x_{i+1} = \mathsf{Ext}(y_i, z_i)$.
2. V sends h, g, and x_{k+1} to P.
3. P sets $x'_{k+1} = x_{k+1}$.
4. For $i = k, ..., 1$:
   (a) V sends $\beta_i = h(y_i, z_i)$ to P.
   (b) P computes y'_i by finding the unique pair (y'_i, z'_i) s.t. $\mathsf{Ext}(y'_i, z'_i) = x'_{i+1}$ and $h(y'_i, z'_i) = \beta_i$. If such a pair (y'_i, z'_i) does not exist or is not unique, P sends a special abort symbol to V.
   (c) P computes x'_i by inverting C_i at y'_i and sends $\alpha_i = g(x'_i)$ to V. If an inverse of y'_i does not exist or is not unique, P sends a special abort symbol to V.
   (d) If V got an abort symbol or if $\alpha_i \neq g(x_i)$, then it rejects and aborts.
5. If all previous tests passed then V accepts.

Fig. 2. A batch SZK protocol for AI

Let k, δ and λ be as in the statement of Theorem 3.4. In order to prove the theorem we need to present an HVSZK protocol for $\mathsf{AI}_\delta^{\otimes k}$ with the specified parameters. The protocol is presented in Fig. 2. The rest of this section is devoted to proving that the protocol indeed satisfies the requirements of Theorem 3.4.

Section Organization. First, in Sect. 4.1 we prove several lemmas that will be useful throughout the analysis of the protocol. Using these lemmas, in Sects. 4.2 and 4.4, we, respectively, establish the completeness, honest-verifier statistical zero-knowledge and soundness properties of the protocol. Lastly, in Sect. 4.5 we analyze the communication complexity and verifier runtime.

4.1 Useful Lemmas

Let $C_1, \ldots, C_k : \{0,1\}^n \to \{0,1\}^m$ be the given input circuits (these can correspond to either a YES or NO instance of AI_δ). Throughout the proof we use $i^* \in [k+1]$ to denote the index of the first NO instance circuit, if such a circuit exists, and $i^* = k+1$ otherwise. That is, $i^* = \min\big(\{k+1\} \cup \{i \in [k] : C_i \text{ is a NO instance}\}\big)$.

For every $i \in [k]$ we introduce the following notations:

- We denote by X_i the distribution over the string $x_i \in \{0,1\}^n$ as sampled in the verifier's setup phase. That is, $X_1 = U_n$ and for every $i \in [k]$, it holds that $Y_i = C_i(X_i)$ and $X_{i+1} = \mathsf{Ext}(Y_i, Z_i)$, where each Z_i is an iid copy of U_d.
- We denote the subset of strings in $\{0,1\}^m$ having a unique preimage under C_i by S_i (i.e., $S_i = \big\{y_i : \big|C_i^{-1}(y_i)\big| = 1\big\}$). Abusing notation, we also use S_i to refer to the uniform distribution over the corresponding set.

For a function f, we define ν_f as $\nu_f(x) = |\{x' : f(x') = f(x)\}|$. We say that $x \in \{0,1\}^n$ has siblings under f, if $\nu_f(x) > 1$. When f is clear from the context, we omit it from the notation.

Lemma 4.1. *For every $i \le i^*$ it holds that $\Delta\left(X_i, U_n\right) \le \frac{1}{k \cdot 2^\lambda} + k \cdot \delta$.*

Proof. We show by induction on i that $\Delta\left(X_i, U_n\right) \le (i-1) \cdot \left(\frac{1}{k^2 \cdot 2^\lambda} + \delta\right)$. The lemma follows by the fact that $i \le k$.

For the base case (i.e., $i = 1$), since X_1 is uniform in $\{0,1\}^n$ we have that $\Delta\left(X_1, U_n\right) = 0$. Let $1 < i \le i^*$ and suppose that the claim holds for $i-1$. Note that $i-1 < i^*$ and so C_{i-1} is a YES instance circuit.

Claim 4.1.1. $\Delta\big(\mathsf{Ext}(S_{i-1}, U_d), U_n\big) \le \frac{1}{k^2 \cdot 2^\lambda}$.

Proof. By definition of AI_δ, the set S_{i-1} has cardinality at least $(1-\delta) \cdot 2^n$. Since $\delta < 1/2$, this means that the min-entropy of (the uniform distribution over) S_i is at least $n-1$. The claim follows by the fact that Ext is an extractor for min-entropy $n-1$ with error $\epsilon = \frac{1}{k^2 \cdot 2^\lambda}$.

We denote by W_i the distribution obtained by selecting (x_{i-1}, z_{i-1}) uniformly in $\{0,1\}^n \times \{0,1\}^d$ and outputting $\mathsf{Ext}\big(C_{i-1}(x_{i-1}), z_{i-1}\big)$.

Claim 4.1.2. $\Delta(W_i, U_n) \leq \frac{1}{k^2 \cdot 2^\lambda} + \delta$.

Proof. Consider the event that X_{i-1} has a sibling (under C_{i-1}). Since C_{i-1} is a YES instance, this event happens with probability at most δ. On the other hand, the distribution of $C_i(X_{i-1})$, conditioned on X_{i-1} not having a sibling, is simply uniform in S_i. The claim now follows by Claim 4.1.1 and Fact 2.2.

We are now ready to bound $\Delta(X_i, U_n)$, as follows:

$$\begin{aligned}
\Delta(X_i, U_n) &\leq \Delta(X_i, W_i) + \Delta(W_i, U_n) \\
&= \Delta\Big(\mathsf{Ext}(C_{i-1}(X_{i-1}), U_d), \mathsf{Ext}(C_{i-1}(U_n), U_d)\Big) + \Delta(W_i, U_n) \\
&\leq \Delta(X_{i-1}, U_n) + \frac{1}{k^2 \cdot 2^\lambda} + \delta \\
&\leq (i-1) \cdot \left(\frac{1}{k^2 \cdot 2^\lambda} + \delta\right),
\end{aligned}$$

where the first inequality is by the triangle inequality, the second inequality is by Fact 2.1 and Claim 4.1.2 and the third inequality is by the inductive hypothesis.

Definition 4.2. *We say that the tuple $(x_1, h, z_1, ...z_k)$ is* good *if the following holds, where we recursively define $y_i = C_i(x_i)$ and $x_{i+1} = \mathsf{Ext}(y_i, z_i)$, for every $i < i^*$:*

1. *For every $i < i^*$, there does not exist $x'_i \neq x_i$ s.t. $C_i(x_i) = C_i(x'_i)$ (i.e., x_i has no siblings).*
2. *For every $i < i^*$, there does not exist $(y'_i, z'_i) \neq (y_i, z_i)$ such that $y'_i \in S_i$, $\mathsf{Ext}(y'_i, z'_i) = \mathsf{Ext}(y_i, z_i)$ and $h(y'_i, z'_i) = h(y_i, z_i)$.*

Lemma 4.3. *The tuple $(x_1, h, z_1, ...z_k)$ sampled by the verifier V is good with probability at least $1 - O(k^2 \cdot \delta + 2^{-\lambda})$.*

In order to prove Lemma 4.3, we first establish the following proposition, which bounds the number of preimages of a random output of the extractor.

Proposition 4.4. *For any $S \subseteq \{0,1\}^m$ with $2^{n-1} \leq |S| \leq 2^n$ and any security parameter $\lambda > 1$, it holds that:*

$$\Pr_{y \leftarrow S, z \leftarrow U_d}\left[\nu_{\mathsf{Ext}}(y, z) > 2^{d+\lambda}\right] \leq \epsilon + \frac{1}{2^\lambda}.$$

Proof. Throughout the current proof we use ν as a shorthand for ν_{Ext}. Abusing notation, we also use S to refer to the uniform distribution over the set S.

For a given security parameter $\lambda > 1$, denote by H (for "heavy") the set of all $(y, z) \in S \times \{0,1\}^d$ that have $\nu(y, z) > |S| \cdot 2^{d-n+\lambda}$, and by $\mathsf{Ext}(H)$ the set $\{\mathsf{Ext}(y, z) : (y, z) \in H\}$. By definition, for any $z \in \mathsf{Ext}(H)$, we have that $\Pr[\mathsf{Ext}(S, U_d) = z] > 2^{-n+\lambda}$. This implies that:

$$|\mathsf{Ext}(H)| < 2^{n-\lambda}.$$

Note again that for any $z \in \mathsf{Ext}(H)$, the above probability is more than 2^{-n}, which is the probability assigned to z by the uniform distribution U_n. It then follows from the definition of statistical distance that:

$$\begin{aligned}\Delta\left(\mathsf{Ext}(S, U_d), U_n\right) &\geq \sum_{z \in \mathsf{Ext}(H)} \left(\Pr\left[\mathsf{Ext}(S, U_d) = z\right] - 2^{-n}\right) \\ &= \Pr\left[\mathsf{Ext}(S, U_d) \in \mathsf{Ext}(H)\right] - |\mathsf{Ext}(H)| \cdot 2^{-n} \\ &> \Pr\left[\mathsf{Ext}(S, U_d) \in \mathsf{Ext}(H)\right] - 2^{-\lambda}.\end{aligned}$$

Since $|S| \geq 2^{n-1}$, the min entropy of S is at least $n-1$, and therefore, it holds that $\Delta\left(\mathsf{Ext}(S, U_d), U_n\right) \leq \epsilon$. Together with the fact that $\Pr\left[\mathsf{Ext}(S, U_d) \in \mathsf{Ext}(H)\right] = \Pr\left[(S, U_d) \in H\right]$, we have:

$$\Pr_{y \leftarrow S, z \leftarrow U_d}\left[\nu(y, z) > |S| \cdot 2^{d-n+\lambda}\right] \leq \epsilon + \frac{1}{2^\lambda}.$$

And since $|S| \leq 2^n$ we have

$$\Pr_{y \leftarrow S, z \leftarrow U_d}\left[\nu(y, z) > 2^{d+\lambda}\right] \leq \epsilon + \frac{1}{2^\lambda}.$$

Using Proposition 4.4 we are now ready to prove Lemma 4.3.

Proof (Proof of Lemma 4.3). For any $i < i^*$, let E_i denote the event that *either* (1) there exists $x'_i \neq x_i$ such that $C_i(x_i) = C_i(x'_i)$, or (2) there exists $(y'_i, z'_i) \neq (y_i, z_i)$ such that $y'_i \in S_i$, $\mathsf{Ext}(y'_i, z'_i) = \mathsf{Ext}(y_i, z_i)$ and $h(y'_i, z'_i) = h(y_i, z_i)$, where $(x_1, \ldots, x_{k+1}, y_1, \ldots, y_k, z_1, \ldots, z_k)$ are as sampled by the verifier.

Lemma 4.3 follows from the following claim, and a union bound over all $i \in [k]$.

Claim 4.4.1. $\Pr[E_i] \leq (k+1) \cdot \delta + \frac{4}{k \cdot 2^\lambda}$, for every $i \in [k]$.

Proof. We first analyze the probability for the event E_i when x_i is sampled uniformly at random. By definition of AI_δ:

$$\Pr_{x_i \leftarrow U_n}\left[x_i \text{ has siblings }\right] \leq \delta.$$

Let us condition on x_i with no siblings being chosen. Under this conditioning, $C_i(x_i)$ is uniform in S_i. We note that $|S_i| \geq (1-\delta) \cdot 2^n \geq 2^{n-1}$ and $|S_i| \leq 2^n$. Thus, by Proposition 4.4 (using security parameter $\lambda + \log k$) it holds that:

$$\Pr_{y_i \leftarrow S_i, z \leftarrow U_d}\left[\nu_{\mathsf{Ext}}(y, z) > k \cdot 2^{\lambda+d}\right] \leq \epsilon + \frac{1}{k \cdot 2^\lambda} \leq \frac{2}{k \cdot 2^\lambda},$$

where the last inequality follows from the fact that $\epsilon = \frac{1}{k^2 \cdot 2^\lambda}$.

Let us therefore assume that the pair (y_i, z_i) has at most $k \cdot 2^{\lambda+d}$ siblings under Ext. We wish to bound the probability that there exists a preimage that collides with (y_i, z_i) under h. Since h is $2^{-(2\lambda+d+2\log k)}$-almost pairwise-independent (into a range of size $2^{2\lambda+d+2\log k}$), for any pair (y', z'), the probability that it collides with (y_i, z_i) under h is at most $\frac{2}{2^{2\lambda+d+2\log k}}$. Since y_i has at most $k \cdot 2^{\lambda+d}$ siblings (under Ext), by a union bound, the probability that any of them collide with (y_i, z_i) (under h) is at most $k \cdot 2^{\lambda+d} \cdot 2^{-(2\lambda+d+2\log k)} = \frac{1}{k \cdot 2^{\lambda}}$.

Thus, when x_i is sampled uniformly at random, the probability that it has a sibling (under C_i) or that there exist (y', z') such that $\mathsf{Ext}(y', z') = \mathsf{Ext}(y_i, z_i)$, where $y'_i \in S_i$ and $h(y', z') = h(y_i, z_i)$, is at most:

$$\delta + \frac{2}{k \cdot 2^{\lambda}} + \frac{1}{k \cdot 2^{\lambda}} = \delta + \frac{3}{k \cdot 2^{\lambda}}.$$

The claim follows by the fact that, by Lemma 4.1, the actual distribution of x_i is $\left(\frac{1}{k \cdot 2^{\lambda}} + k \cdot \delta\right)$-close to uniform.

This concludes the proof of Lemma 4.3.

4.2 Completeness

Let $C_1, \ldots, C_k \in \mathsf{AI}_\delta$. Assume first that V generates a *good* tuple $(x_1, h, z_1, \ldots, z_k)$ (as per Definition 4.2). Observe that in such a case, by construction of the protocol, it holds that $x'_i = x_i$ and $y'_i = y_i$, for every $i \in [k]$. Therefore, the verifier accepts in such a case (with probability 1).

By Lemma 4.3, the tuple $(x_1, h, g, z_1, \ldots, z_k)$ is good with all but $O\left(k^2 \cdot \delta + 2^{-\lambda}\right)$ probability. Thus, the completeness error is upper bounded by $O\left(k^2 \cdot \delta + 2^{-\lambda}\right)$.

4.3 Honest-Verifier Statistical Zero-Knowledge

The simulator is presented in Fig. 3.

The generation of $(x_1, h, g, z_1, ..., z_k)$ is identical for the verifier and for the simulator. Assuming that the tuple $(x_1, h, z_1, ..., z_k)$ is good, by construction the prover does not abort in the honest execution (as in the case of completeness). Moreover, in this case, each x'_i (resp., (y'_i, z'_i)) found by the prover is equal to x_i (resp., (y_i, z_i)) chosen by the verifier. Therefore, conditioned on the tuple $(x_1, h, z_1, ..., z_k)$ being good, the distributions of (1) the transcript generated in the honest execution, and (2) the simulated transcript are identically distributed. The fact that the protocol is honest-verifier statistical zero-knowledge now follows from Lemma 4.3, and by applying Fact 2.2 twice.

4.4 Soundness

Let $C_1, ..., C_k : \{0,1\}^n \to \{0,1\}^m$ be such that one of them is a NO instance of AI_δ. Recall that $i^* \in [k]$ denotes the index of the first such NO instance circuit (i.e., C_{i^*} is a NO instance of AI_δ but for every $i < i^*$, it holds that C_i is a YES instance).

We first make two simplifying assumptions. First, recall that value of y_{i^*} is specified by the verifier by having it send $x_{k+1}, \beta_k, \ldots, \beta_{i^*}$ to the prover. Instead,

```
Simulator for the $\mathsf{AI}_\delta$ Batch Verification Protocol

INPUT: $C_1, \dots, C_k$

THE SIMULATOR:

1. Sample $h \leftarrow H_n$, $g \leftarrow G_n$ and $x_1 \in \{0,1\}^n$.
2. For $i = 1, ..., k$:
   (a) Compute $y_i = C_i(x_i)$.
   (b) Sample $z_i \leftarrow \{0,1\}^d$.
   (c) Compute $\alpha_i = g(x_i)$.
   (d) Compute $\beta_i = h(y_i, z_i)$.
   (e) Compute $x_{i+1} = \mathsf{Ext}(y_i, z_i)$.
3. Output $\mathsf{transcript} = \Big( \big(C_1, \dots, C_k\big), \big(x_1, z_1, \dots, z_k, h, g\big), \big(\alpha_k, \dots, \alpha_1\big) \Big)$.
```

Fig. 3. Simulator for AI_δ batch verification

we will simply assume that the verifier sends y_{i^*} directly to the prover. Since y_{i^*} can be used to generate the verifier's distribution consistently, revealing y_{i^*} only makes the prover's job harder and therefore can only increase the soundness error. Second, we modify the protocol so that the verifier merely checks that $\alpha_{i^*} = g(x_{i^*})$ – if so it accepts and otherwise it rejects. Once again having removed the verifier's other tests can only increase the soundness error.

Thus, it suffices to bound the soundness error of the following protocol. The verifier samples x_{i^*} as in the real protocol, sends $y_{i^*} = C_{i^*}(x_{i^*})$ and the hash function g to the prover and expects to get in response $g(x_{i^*})$. We show that the prover's probability of making the verifier accept is bounded by a constant.

In order to bound the prover's success probability in the foregoing experiment, we first give an upper bound assuming that x_{i^*} is uniform in $\{0,1\}^n$, rather than as specified by the protocol (and, as usual, $y_{i^*} = C_{i^*}(x_{i^*})$). Later we shall remove this assumption using Lemma 4.1, which guarantees that x_{i^*} is actually *close* to uniform.

Let P^* be the optimal prover strategy. Namely, given g and y_{i^*}, the prover P^* outputs the hash value $\alpha_{i^*} \in \{0,1\}^\ell$ with the largest probability mass (i.e., that maximizes $|C_{i^*}^{-1}(y_{i^*}) \cap g^{-1}(\alpha)|$).

Let $\widehat{Y}_{i^*}$ denote the distribution obtained by sampling $x \in \{0,1\}^n$ uniformly at random, conditioned on x have a sibling under C_{i^*} and outputting $C_{i^*}(x)$. Using elementary probability theory we have that:

$$\begin{aligned}
\Pr_{\substack{g\leftarrow G_n\\ x_{i^*}\leftarrow\{0,1\}^n}}\left[\mathsf{P}^*(g,y_{i^*})=g(x_{i^*})\right] &\le \Pr_{\substack{g\leftarrow G_n\\ x_{i^*}\leftarrow\{0,1\}^n}}\left[\mathsf{P}^*(g,y_{i^*})=g(x_{i^*}) \mid x_{i^*} \text{ has siblings}\right]\\
&\quad + \Pr[x_{i^*} \text{ has no siblings}]\\
&\le \Pr_{\substack{g\leftarrow G_n\\ y_{i^*}\leftarrow\widehat{Y}_{i^*}\\ x_{i^*}\leftarrow C_{i^*}^{-1}(y_{i^*})}}\left[\mathsf{P}^*(g,y_{i^*})=g(x_{i^*})\right]+\delta\\
&= E_{y_{i^*}\leftarrow\widehat{Y}_{i^*}}\left[\Pr_{\substack{g\leftarrow G_n\\ x_{i^*}\leftarrow C_{i^*}^{-1}(y_{i^*})}}\left[\mathsf{P}^*(g,y_{i^*})=g(x_{i^*})\right]\right]+\delta,
\end{aligned} \tag{1}$$

where the second inequality follows from the fact that C_{i^*} is a NO instance.

Fix y_{i^*} in the support of $\widehat{Y}_{i^*}$ (i.e., $|C_{i^*}^{-1}(y_{i^*})| \ge 2$) and let $u = |C_{i^*}^{-1}(y_{i^*})|$. We show that $\Pr_{g\in G^n, x_{i^*}\leftarrow C_{i^*}^{-1}(y_{i^*})}\left[\mathsf{P}^*(g,y_{i^*})=g(x_{i^*})\right]$ is upper bounded by a constant.

Let E be the event (defined only over the choice of g) that for every hash value $\alpha \in \{0,1\}^\ell$, it holds that $|C_{i^*}^{-1}(y_{i^*}) \cap g^{-1}(\alpha)| \le \frac{7}{8}u$. That is, the event E means that no hash value has more than 7/8 fraction of the probability mass (when sampling x_{i^*} uniformly in $C_{i^*}^{-1}(y_{i^*})$ and outputting $g(x_{i^*})$).[12]

Claim 4.4.2. The event E occurs with probability a least 1/10.

Proof. Fix a hash value $\alpha \in \{0,1\}^\ell$, and let $X = |C_{i^*}^{-1}(y_{i^*}) \cap g^{-1}(\alpha)|$ be a random variable (over the randomness of g). Observe that X can be expressed as a sum of u pairwise independent Bernoulli random variables, each of which is 1 with probability $2^{-\ell}$ and 0 otherwise. Thus, the expectation of X is $u/2^\ell$ and the variance is $u \cdot 2^{-\ell} \cdot (1-2^{-\ell}) \le u \cdot 2^{-\ell}$. By Chebyshev's inequality (Lemma 2.3), it holds that

$$\begin{aligned}
\Pr\left[X > \frac{7}{8}u\right] &\le \Pr\left[\left|X - \frac{u}{2^\ell}\right| > \frac{3}{4}u\right]\\
&\le \frac{Var\,[X]}{(3/4)^2 \cdot u^2}\\
&\le \frac{16}{9u} \cdot \frac{1}{2^\ell},
\end{aligned}$$

where the first inequality follows from the fact that ℓ is a sufficiently large constant. Taking a union bound over all α's we have that the probability that there exists some α with more than 7/8 fraction of the preimages in U (under g) is less than $\frac{16}{9u} < 0.9$, where we use the fact that $u \ge 2$.

[12] We remark that the choice of 7/8 is somewhat but not entirely arbitrary. In particular, in case u is very small (e.g., $u = 2$) there may very well be a hash value that has 50% of the probability mass.

Observe that conditioned on the event E, the probability (over $x_{i^*} \leftarrow C_{i^*}^{-1}(y_{i^*})$) that $\mathsf{P}^*(g, y_{i^*}) = g(x_{i^*})$ is at most 7/8. Thus, by Claim 4.4.2 we obtain that:

$$\Pr_{g, x_{i^*} \leftarrow C_{i^*}^{-1}(y_{i^*})} \left[\mathsf{P}^*(g, y_{i^*}) \neq g(x_{i^*})\right] \geq \Pr[E] \cdot \Pr_{g, x_{i^*} \leftarrow C_{i^*}^{-1}(y_{i^*})} \left[\mathsf{P}^*(g, y_{i^*}) \neq g(x_{i^*}) | E\right] \geq 1/80.$$

Plugging this into Eq. (1), we have that the prover convinces the verifier to accept with probability at most $1 - \frac{1}{80} + \delta$, when x_{i^*} is sampled uniformly at random in $\{0,1\}^n$.

By Lemma 4.1 it holds that $\Delta\left(X_{i^*}, U_n\right) \leq \frac{1}{k \cdot 2^\lambda} + k \cdot \delta$. Therefore (using Fact 2.2), the probability that the verifier accepts when x_{i^*} is sampled as in the protocol is at most $1 - \frac{1}{80} + \frac{1}{k \cdot 2^\lambda} + (k+1) \cdot \delta$, which is bounded away from 1 since $\delta < \frac{1}{100k^2}$ and λ is sufficiently large.

4.5 Communication Complexity and Verifier Run Time

We first bound the amount of bits sent during the interaction:

- Sending x_{k+1} costs n bits.
- By Lemma 2.10, the seed length of the extractor is $d = \log(m) + O(\log n \cdot \log(\frac{n}{\epsilon})) = \log(N) + \lambda \cdot \mathit{polylog}(n,k)$ and therefore, the cost of sending $z_1, \ldots, z_k$ is $k \cdot (\log(N) + \lambda \cdot \mathit{polylog}(n,k))$.
- By Lemma 2.8, the description length of $h : \{0,1\}^m \times \{0,1\}^d \rightarrow \{0,1\}^{2\lambda + d + 2\log k}$, a $\frac{1}{2^{2\lambda+d+2\log k}}$-almost pairwise-independent hash function, is $O(\log(N) + \lambda + \mathit{polylog}(k))$. The cost of sending the hashes $\beta_1, \ldots, \beta_k$ is $k \cdot O(\lambda + d + \log k) = k \cdot (\log(N) + \lambda \cdot \mathit{polylog}(n,k))$.
- By Lemma 2.7, the description length of $g \in \{0,1\}^n \rightarrow \{0,1\}^\ell$, a pairwise independent hash function, is $O(n)$. The cost of sending the hashes $\alpha_1, \ldots, \alpha_k$ is $O(k)$.

In total, the communication complexity is $O(n) + k \cdot (\log(N) + \lambda \cdot \mathit{polylog}(n,k))$. As for the verifier run time, For each iteration i the verifier running time is as follows:

- Evaluating the circuit C_i takes time $\mathsf{poly}(N)$.
- By Lemma 2.10, evaluating Ext takes time $\mathsf{poly}(m,d) = \mathsf{poly}(N, \log k, \lambda)$.
- By Lemma 2.8, evaluating h on an input of size $m + d$ takes time $\mathsf{poly}(N, \log k, \lambda)$.
- By Lemma 2.7, evaluating g on an input of size n takes time $\mathsf{poly}(n)$.

In total, the verifier running time is $k \cdot \mathsf{poly}(N, \log k, \lambda)$.

Acknowledgments. We thank an anonymous TCC reviewer for pointing that our techniques fall outside the scope of the Holenstein-Renner [HR05] blackbox model (see Remark 1.4).

Inbar Kaslasi and Ron Rothblum were supported in part by a Milgrom family grant, by the Israeli Science Foundation (Grants No. 1262/18 and 2137/19), and the Technion Hiroshi Fujiwara cyber security research center and Israel cyber directorate.

Guy Rothblum has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819702).

Adam Sealfon was a PhD student at MIT for part of the duration of this project, and was supported in part by NSF CNS-1413920, Sloan/NJIT 996698, MIT/IBM W1771646, and NSF CNS-1804794.

Prashant Vasudevan was supported in part by AFOSR Award FA9550-19-1-0200, AFOSR YIP Award, NSF CNS Award 1936826, DARPA and SPAWAR under contract N66001-15-C-4065, a Hellman Award and research grants by the Okawa Foundation, Visa Inc., and Center for Long-Term Cybersecurity (CLTC, UC Berkeley). The views expressed are those of the authors and do not reflect the official policy or position of the funding agencies.

References

[Aar04] Aaronson, S.: Limits on efficient computation in the physical world. CoRR, abs/quant-ph/0412143 (2004)

[AH91] Aiello, W., Hastad, J.: Statistical zero-knowledge languages can be recognized in two rounds. J. Comput. Syst. Sci. **42**(3), 327–345 (1991)

[APS18] Alamati, N., Peikert, C., Stephens-Davidowitz, N.: New (and Old) Proof Systems for Lattice Problems. In: Abdalla, M., Dahab, R. (eds.) PKC 2018, Part II. LNCS, vol. 10770, pp. 619–643. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-76581-5_21

[BBD+20] Ball, M., et al.: Cryptography from information loss. In: Vidick, T. (ed.) 11th Innovations in Theoretical Computer Science Conference, ITCS 2020, Seattle, Washington, USA, 12–14 January 2020. LIPIcs, vol. 151, pp. 81:1–81:27. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020)

[BDRV18a] Berman, I., Degwekar, A., Rothblum, R.D., Vasudevan, P.N.: From laconic zero-knowledge to public-key cryptography. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part III. LNCS, vol. 10993, pp. 674–697. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96878-0_23

[BDRV18b] Berman, I., Degwekar, A., Rothblum, R.D., Vasudevan, P.N.: Multi-collision resistant hash functions and their applications. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part II. LNCS, vol. 10821, pp. 133–161. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78375-8_5

[BDRV19] Berman, I., Degwekar, A., Rothblum, R.D., Vasudevan, P.N.: Statistical difference beyond the polarizing regime. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019, Part II. LNCS, vol. 11892, pp. 311–332. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36033-7_12

[BFM88] Blum, M., Feldman, P., Micali, S.: Non-interactive zero-knowledge and its applications (extended abstract). In: Proceedings of the 20th Annual ACM Symposium on Theory of Computing, Chicago, Illinois, USA, 2–4 May 1988, pp. 103–112 (1988)

[BGR98] Bellare, M., Garay, J.A., Rabin, T.: Fast batch verification for modular exponentiation and digital signatures. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 236–250. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054130

[BHK17] Brakerski, Z., Holmgren, J., Kalai, Y.: Non-interactive delegation and batch NP verification from standard computational assumptions. In: Hatami, H., McKenzie, P., King, V. (eds.) Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, 19–23 June 2017, pp. 474–482. ACM (2017)

[BL13] Bogdanov, A., Lee, C.H.: Limits of provable security for homomorphic encryption. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 111–128. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_7

[BSMP91] Blum, M., De Santis, A., Micali, S., Persiano, G.: Noninteractive zero-knowledge. SIAM J. Comput. **20**(6), 1084–1118 (1991)

[CHP12] Camenisch, J., Hohenberger, S., Pedersen, M.Ø.: Batch verification of short signatures. J. Cryptol. **25**(4), 723–747 (2012)

[CP92] Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 89–105. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-48071-4_7

[Dru15] Drucker, A.: New limits to classical and quantum instance compression. SIAM J. Comput. **44**(5), 1443–1479 (2015)

[DSCP94] De Santis, A., Di Crescenzo, G., Persiano, G.: The knowledge complexity of quadratic residuosity languages. Theor. Comput. Sci. **132**(2), 291–317 (1994)

[For89] Fortnow, L.J.: Complexity-theoretic aspects of interactive proof systems. Ph.D. thesis, Massachusetts Institute of Technology (1989)

[GG00] Goldreich, O., Goldwasser, S.: On the limits of nonapproximability of lattice problems. J. Comput. Syst. Sci. **60**(3), 540–563 (2000)

[GK93] Goldreich, O., Kushilevitz, E.: A perfect zero-knowledge proof system for a problem equivalent to the discrete logarithm. J. Cryptol. **6**(2), 97–116 (1993)

[GKL93] Goldreich, O., Krawczyk, H., Luby, M.: On the existence of pseudorandom generators. SIAM J. Comput. **22**(6), 1163–1175 (1993)

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

[GMR98] Gennaro, R., Micciancio, D., Rabin, T.: An efficient non-interactive statistical zero-knowledge proof system for quasi-safe prime products. In: Gong, L., Reiter, M.K. (eds.) CCS 1998, Proceedings of the 5th ACM Conference on Computer and Communications Security, San Francisco, CA, USA, 3–5 November 1998, pp. 67–72. ACM (1998)

[GSV98] Goldreich, O., Sahai, A., Vadhan, S.: Honest-verifier statistical zero-knowledge equals general statistical zero-knowledge. In: STOC (1998)

[GSV99] Goldreich, O., Sahai, A., Vadhan, S.: Can statistical zero knowledge be made non-interactive? or on the relationship of SZK and *NISZK*. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 467–484. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_30

[GUV07] Guruswami, V., Umans, C., Vadhan, S.P.: Unbalanced expanders and randomness extractors from parvaresh-vardy codes. In: 22nd Annual IEEE Conference on Computational Complexity (CCC 2007), San Diego, California, USA, 13–16 June 2007, pp. 96–108. IEEE Computer Society (2007)

[GV99] Goldreich, O., Vadhan, S.P.: Comparing entropies in statistical zero knowledge with applications to the structure of SZK. In: CCC (1999)

[HHR11] Haitner, I., Harnik, D., Reingold, O.: On the power of the randomized iterate. SIAM J. Comput. **40**(6), 1486–1528 (2011)

[HR05] Holenstein, T., Renner, R.: One-way secret-key agreement and applications to circuit polarization and immunization of public-key encryption. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 478–493. Springer, Heidelberg (2005). https://doi.org/10.1007/11535218_29

[Ish] Ishai, Y.: Zero-knowledge proofs from information-theoretic proof systems. https://zkproof.org/2020/08/12/information-theoretic-proof-systems/

[Kil92] Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: Kosaraju, S.R., Fellows, M., Wigderson, A., Ellis, J.A. (eds.) Proceedings of the 24th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, 4–6 May 1992, pp. 723–732. ACM (1992)

[KMN+14] Komargodski, I., Moran, T., Naor, M., Pass, R., Rosen, A., Yogev, E.: One-way functions and (im)perfect obfuscation. In: 55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, 18–21 October 2014, pp. 374–383. IEEE Computer Society (2014)

[KRR+20] Kaslasi, I., Rothblum, G.N., Rothblum, R.D., Sealfon, A., Vasudevan, P.N.: Batch verification for statistical zero knowledge proofs. In: Electronic Colloquium on Computational Complexity (ECCC) (2020)

[KY18] Komargodski, I., Yogev, E.: On distributional collision resistant hashing. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part II. LNCS, vol. 10992, pp. 303–327. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96881-0_11

[LFKN92] Lund, C., Fortnow, L., Karloff, H.J., Nisan, N.: Algebraic methods for interactive proof systems. J. ACM **39**(4), 859–868 (1992)

[LV16] Liu, T., Vaikuntanathan, V.: On Basing Private Information Retrieval on NP-Hardness. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016, Part I. LNCS, vol. 9562, pp. 372–386. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49096-9_16

[MV03] Micciancio, D., Vadhan, S.P.: Statistical zero-knowledge proofs with efficient provers: lattice problems and more. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 282–298. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_17

[NMVR94] Naccache, D., M'Rahi, D., Vaudenay, S., Raphaeli, D.: Can D.S.A. be improved? — complexity trade-offs with the digital signature standard. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 77–85. Springer, Heidelberg (1995). https://doi.org/10.1007/BFb0053426

[NN93] Naor, J., Naor, M.: Small-bias probability spaces: Efficient constructions and applications. SIAM J. Comput. **22**(4), 838–856 (1993)

[NV06] Nguyen, M.-H., Vadhan, S.P.: Zero knowledge with efficient provers. In: Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, 21–23 May 2006, pp. 287–295 (2006)

[NZ96] Nisan, N., Zuckerman, D.: Randomness is linear in space. J. Comput. Syst. Sci. **52**(1), 43–52 (1996)

[Oka00] Okamoto, T.: On relationships between statistical zero-knowledge proofs. J. Comput. Syst. Sci. **60**(1), 47–108 (2000)

[Ost91] Ostrovsky, R.: One-way functions, hard on average problems, and statistical zero-knowledge proofs. In: Structure in Complexity Theory Conference, pp. 133–138 (1991)

[OV08] Ong, S.J., Vadhan, S.: An equivalence between zero knowledge and commitments. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 482–500. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78524-8_27

[OW93] Ostrovsky, R., Wigderson, A.: One-way fuctions are essential for non-trivial zero-knowledge. In: ISTCS, pp. 3–17 (1993)

[PPS15] Pandey, O., Prabhakaran, M., Sahai, A.: Obfuscation-based non-black-box simulation and four message concurrent zero knowledge for NP. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015, Part II. LNCS, vol. 9015, pp. 638–667. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46497-7_25

[PV08] Peikert, C., Vaikuntanathan, V.: Noninteractive statistical zero-knowledge proofs for lattice problems. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 536–553. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85174-5_30

[RRR16] Reingold, O., Rothblum, G.N., Rothblum, R.D.: Constant-round interactive proofs for delegating computation. In: Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, 18–21 June 2016, pp. 49–62 (2016)

[RRR18] Reingold, O., Rothblum, G.N., Rothblum, R.D.: Efficient batch verification for UP. In: 33rd Computational Complexity Conference, CCC 2018, San Diego, CA, USA, 22–24 June 2018, pp. 22:1–22:23 (2018)

[SCPY98] De Santis, A., Di Crescenzo, G., Persiano, G., Yung, M.: Image density is complete for non-interactive-SZK. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 784–795. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0055102

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

[SV03] Sahai, A., Vadhan, S.: A complete problem for statistical zero knowledge. J. ACM (JACM) **50**(2), 196–249 (2003)

[Vad12] Vadhan, S.P.: Pseudorandomness. Found. Trends Theor. Comput. Sci. **7**(1–3), 1–336 (2012)

[YGLW15] Yu, Yu., Gu, D., Li, X., Weng, J.: The randomized iterate, revisited - almost linear seed length PRGs from a broader class of one-way functions. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015, Part I. LNCS, vol. 9014, pp. 7–35. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46494-6_2