On Black-Box Constructions of Time and Space Efficient Sublinear Arguments from Symmetric-Key Primitives

Laasya Bangalore[1], Rishabh Bhadauria[2], Carmit Hazay[2], and Muthuramakrishnan Venkitasubramaniam[1]

[1] Georgetown University, USA
[2] Bar Ilan University, Israel

Abstract. Zero-knowledge proofs allow a prover to convince a verifier of a statement without revealing anything besides its validity. A major bottleneck in scaling sub-linear zero-knowledge proofs is the high space requirement of the prover, even for NP relations that can be verified in a small space.

In this work, we ask whether there exist complexity-preserving (i.e. overhead w.r.t time and space are minimal) succinct zero-knowledge arguments of knowledge with minimal assumptions while making only black-box access to the underlying primitives. We design the first such zero-knowledge system with sublinear communication complexity (when the underlying NP relation uses non-trivial space) and provide evidence why existing techniques are unlikely to improve the communication complexity in this setting. Namely, for every NP relation that can be verified in time T and space S by a RAM program, we construct a public-coin zero-knowledge argument system that is black-box based on collision-resistant hash-functions (CRH) where the prover runs in time $\widetilde{O}(T)$ and space $\widetilde{O}(S)$, the verifier runs in time $\widetilde{O}(T/S+S)$ and space $\widetilde{O}(1)$ and the communication is $\widetilde{O}(T/S)$, where $\widetilde{O}()$ ignores polynomial factors in $\log T$ and κ is the security parameter. As our construction is public-coin, we can apply the Fiat-Shamir heuristic to make it non-interactive with sample communication/computation complexities. Furthermore, we give evidence that reducing the proof length below $\widetilde{O}(T/S)$ will be hard using existing symmetric-key based techniques by arguing the space-complexity of constant-distance error correcting codes.

1 Introduction

Zero-knowledge proofs, introduced by Goldwasser, Micali and Rackoff [20] are powerful cryptographic objects that allow a prover to convince a verifier of a statement while revealing nothing beyond the validity of the statement. Succinct non-interactive zero-knowledge arguments (ZK-SNARKs and ZK-SNARGs) are variants of zero-knowledge proof systems that offer very efficient verification, namely, proof lengths and verification times that are polylogarithmic in the size of the instance. ZK-SNARKs have been the focus of intense

research from both theory and practice in the past few years as they are becoming an indispensable tool to bringing privacy and efficiency to blockchains (see [23, 35] for two recent surveys).

While the initial constructions of concretely efficient ZK-SNARKs suffered from significantly high prover times, recent works have shown how to improve the computational complexity to essentially linear in the time taken compute the underlying relation (for an NP-language) [14, 36, 37, 33, 34, 26, 12, 27]. However, these works come with a steep price in terms of *space*, namely, for computations that take time T and space S, the space complexity of the prover is $\Omega(T)$. Notably, only a few works provide time and space efficient constructions that we discuss next. This fact turns out to be a major bottleneck in scaling up zero-knowledge proofs to larger and larger computations.

To make the context precise, we focus on the task of proving that a non-deterministic RAM machine M accepts a particular instance x, i.e. uniform non-deterministic computations. The goal here is if M accepts/rejects x in time T and space S the resulting ZK proof system preserves these complexities on the prover's side and polylogarithmic in T (i.e. succinct) or even sublinear on the verifier's side.

When considering designated verifier ZK-SNARKs, complexity preserving solutions (i.e. poly-logarithmic overhead in space and time) have been constructed by Bitansky and Chiesa [8] and by Holmgren and Rothblum [22] in the non-interactive setting. The work of Ephraim et al. [18] show that assuming the existence of standard (circuit) SNARKs one can construct a non-interactive succinct argument of knowledge (i.e. SNARK) for parallel RAM computations where the complexities are preserved on the prover's side and the verifier requires polylogarithmic in T time and space based on collision-resistant hash functions (CRHF), where the underlying CRHF and SNARK is used in a non-black-box manner. Publicly-verifiable ZK-SNARKs with similar overheads can be accomplished via recursive composition [13, 16, 15]. However, these constructions have significant overheads as they typically rely on non-black-box usage of the underlying primitives. Imposing black-box access to the underlying primitives is an important step to obtain practically viable constructions [28, 1, 21].

More recently, two works by Block et al. [9, 10], designed the first black-box construction of a ZK-SNARKs with polylogarithmic overhead in space and time based on "more standard" assumptions. The first work assumes hardness of discrete logarithm in prime-order groups and relies on the random oracle to construct a public-coin zero-knowledge argument where the proof length is $\mathsf{polylog}(T)$, the prover is complexity preserving and the verifier runtime is $T \cdot \mathsf{polylog}(T)$ while using $\mathsf{polylog}(T)$ space. The second work improves the verifier's runtime from $T \cdot \mathsf{polylog}(T)$ to $n \cdot \mathsf{polylog}(T)$, where n is the input length, under hardness assumptions on hidden order groups. We note that these works make extensive use of public-key operations - e.g., the prover needs to do $\Omega(T)$ exponentiations, and public-key operations are typically orders of magnitude more expensive than symmetric key operations.

Thus, the prior works leave the following question open:

> *Is it possible to design a complexity preserving (zero-knowledge) argument system based on minimal assumptions (eg, symmetric-key primitives) with a succinct verifier where the underlying primitives are used in a black-box manner?*

As noted above, the problem is solved if we are willing to assume (and extensively use) public-key primitives. We further highlight that the problem can be solved if we relax either the computation or the space requirements of the prover. The works of [5, 7] demonstrate a ZK-SNARK with succinct proofs and verification (i.e. polylogarithmic in T), where the prover's running time and space are quasilinear in T. If we relax the time but restrict the space of the prover, it is easy to see how to extend the same constructions of [5, 7] by observing that a Reed-Solomon encoding of streaming data of size T can be computed in time polynomial in T with space $\mathsf{polylog}(T)$. Finally, if we relax the black-box requirement, recursive composition can be used to construct (ZK-)SNARKS [13, 16, 15].

1.1 Our Results

Theorem 1. *Assume that collision-resistant hash functions exist. Then, every NP relation that can be verified by a time T and space S RAM machine has a public-coin zero-knowledge argument-system such that:*

1. *The prover runs in time $T \cdot \mathsf{poly}(\log(T), \lambda)$ and uses space $S \cdot \mathsf{poly}(\log(T), \lambda)$.*
2. *The verifier runs in time $(T/S + S) \cdot \mathsf{poly}(\log(T), \lambda)$ and uses space $\mathsf{poly}(\log(T), \lambda)$.*
3. *The communication complexity is $(T/S) \cdot \mathsf{poly}(\log(T), \lambda)$ and number of rounds is constant.*
4. *The protocol has perfect completeness and negligible soundness error.*

where λ is the computational security parameter. Moreover, applying the Fiat-Shamir heuristic results in a non-interactive sublinear zero-knoweldge argument of knowledge with the same asymptotic efficiencies.

We remark that our construction could lead to concretely efficient complexity preserving ZK-SNARKs that are possibly post-quantum secure since it is based on symmetric-key primitives and is black-box in the underlying primitives.

Next we complement our upper bound with a lower bound. We prove that any constant-distance code with an encoding algorithm that runs in time quasi-linear in the input length n must require space at least $\tilde{\Omega}(n)$. More formally, we prove the following theorem.

Theorem 2 (Informal). *Suppose that a a code over $\mathbb{F}$ with message length n, codeword length m and minimum relative distance δ (i.e. $[m, n, \delta m]$ code) can be encoded via a RAM machine with space S while making r passes over the input message, then $S \in \Omega(\delta n/r \cdot \log |\mathbb{F}|)$.*

Interpreting theorem 2 in the context of proof systems, we note that most IOP/PCP constructions use constant-distance codes to encode the computation-transcript, which is of size $\widetilde{O}(T)$. Our lower bound implies that encoding an $\widetilde{O}(T)$ message with space S will have distance $\widetilde{O}(S/T)$ which implies a query complexity (and consequently proof length) of $\Omega(T/S)$ for the IOP/PCPs that encode the transcript and this matches our upper bound.

1.2 Technical Overview

The most common approach to design a ZK-SNARK black-box from symmetric-key primitives in a black-box way is to first design an interactive oracle proof (IOP) system [6, 31], then compile it to an succinct interactive zero-knowledge proof system (honest-verifier) using collision-resistant hash functions and finally relying on the Fiat-Shamir heuristic [19] to make it non-interactive.

Interactive oracle proofs and probabilistically checkable proofs encode the computation in such a way that the verifier needs to query only few bits to verify its validity. These proofs typically involve encoding the computation transcript using some constant-rate constant-distance error-correcting codes. Computing these codes on a computational transcript of size T can be done efficiently, i.e. in time $\widetilde{O}(T)$ using FFTs. Unfortunately, all FFTs are believed to require a high space complexity. In fact, it was shown in a specific computational model that computing Fourier transforms on a domain of size n with time T and space S requires $T \cdot S \in \Omega(n^2)$ [32]. This means that if $S << T$, then the running time to compute Fourier transforms will no longer be quasi-linear in n. As mentioned above, we demonstrate that even designing codes with constant-distance requires significant space.

Our starting point for our upper bound is the Ligero ZK argument system [1] which is an instantiation of the IOP framework (based on the MPC-in-the-head paradigm [24]) but provides a trade-off between size of the Fourier transforms and proof length. Given a parameter β, for a computation of size T, the Ligero proof system provides a $O(T/\beta + \beta)$-sized proof and requires executing several $O(T/\beta)$ FFTs on size β. However, the proof system as we describe below still requires a space complexity of $O(T)$. Our main contribution is a new proof system that follows the blueprint of the Ligero proof system and preserves time and space efficiency.

We provide a high-level description of the Ligero proof system in the IOP model and identify the bottlenecks in making it time and space efficient. Given an arithmetic circuit C over a field $\mathbb{F}$, the Ligero system proves satisfiability of C as follows:

1. **Preparing the proof oracle:** In the first step in Ligero, the prover computes an "extended" witness (of size $O(|C|)$) that incorporates all intermediate computations (namely, output of each "gate") and encodes it using an Interleaved Reed-Solomon code. This code is set as the proof oracle.
2. **Testing the encoding:** Next, the verifier tests if the prover set the oracle with a valid encoding of some message. The Interleaved Reed-Solomon

Code can be interpreted as a matrix U where each row is a Reed-Solomon code of some message. The verifier challenges the prover with a set of random elements (one for each row of U) and the prover responds with a random linear combination of the rows based on the randomness provided by the prover. The verifier rejects if this combination is not a valid (Reed Solomon) code. The idea is that if each row of U is a valid Reed Solomon code, then by linearity the random linear combination provided by the prover must also be a valid Reed Solomon code.

3. **Testing linear constraints:** Linear constraints incorporate all the addition gates and circuit wiring in C. The verifier tests these constraints by providing randomness and obtaining an encoding of a random linear combination of the result of all the linear constraints applied to the extended witness. Given the prover's response the verifier checks if the response encodes values that sum up to 0. The idea here is that even if one of the linear relations do not hold, then the values encoded in the random linear combination will not sum up to 0 with very high probability.
4. **Testing quadratic constraints:** Quadratic constraints incorporate all the multiplication gates in C. The verifier tests these constraints analogously to the linear constraints. Specifically, the verifier checks if the prover's response encodes a vector of all zeros. This test utilizes the strong multiplicative property of the Reed-Solomon encoding [30].
5. **Column check:** Finally, the verifier checks if the responses provided by the prover in the three tests presented above are consistent with the code in the proof oracle. Since all the tests can be performed via row operations on the matrix, the verifier selects a random subset of the columns of the matrix and recomputes the results of the tests for these columns and checks if they are consistent with the responses.

Compiling the IOP system to a sublinear argument is achieved by replacing the proof oracle with the root hash of a Merkle hash tree with leaves as the elements of the code matrix and providing Merkle decommitments along with the elements (columns) revealed in the column check step [25, 6].

Next, we analyze the space complexity of the Ligero system, describe the obstacles to make it space-efficient and then explain our approach to overcome these obstacles.

1. The first step of the argument system involves the prover computing the code generated by encoding the witness where this codeword serves as the proof oracle. This is followed by computing the Merkle hash tree of the code. The size of the code is $O(|C|)$ and if we naively compute the Merkle tree it will require holding the entire code in memory. However, if the Interleaved Reed Solomon code can be computed one row at a time then the Merkle hash tree can be computed with space proportional to the length of the code (i.e. number of columns in the matrix) as the hash of the leaves can be iteratively aggregated using the Merkle-Damgard construction [17]. We remark that computing the code one row at a time is not straight forward as the Ligero proof system actually requires the extended witness

to be arranged in a specific structure. The verifier on the other hand can check the Merkle decommitments of the κ columns in space proportional to κ and $\mathsf{polylog}(T)$.

2. In the code test, the prover computes a random linear combination of the rows of the matrix. Once again, if we assume that the code matrix can be computed one row at a time, then the linear combination can also be computed in space proportional to the length of the code by maintaining a running aggregate.
3. The linear test is one of the main bottlenecks in terms of space complexity. As the wiring in the circuit C can be arbitrary the linear constraints can involve values encoded in arbitrary rows of the matrix. This means even if the code can be computed one row at a time, computing the response to the linear constraint could involve recomputing the entire code for each constraint to access different rows of the code and this blows up the running time of the prover beyond quasi-linear in the worst case. The issue of the wiring in the circuit C being arbitrary (as described in the previous step) poses a challenge to improving the verifier's space complexity as well. In addition to the same issues as discussed above, the verifier has more stringent space restrictions, and verifying the prover's response to the linear test in small space is non-trivial. We discuss our approach for the linear test below.
4. In the quadratic test, the verifier checks the correctness of all the multiplication gates. The prover prepares the extended witness in a specific way where the multiplication gates are batched and the wire values are aligned so that they can be tested for correctness as follows: the verifier provides randomness and the prover provides an aggregate computed via row operations which the verifier checks if it encodes the all 0's string. Making this space-efficient requires arranging each batch of multiplication gates in neighboring rows.
5. In the final step, the verifier queries the proof oracle on a subset of the columns and verifies if the responses provided by the prover for the code, linear and quadratic tests are consistent with the columns. In the Ligero system, all these tests are results of row operations on the encoded matrix. Hence the verifier can check the correctness by simply recomputing the row operations on the subset of columns opened by the verifier and checking against the prover's responses. If the tests can be computed by the prover in a space efficient manner, then the verifier can rely on a similar approach to recompute the responses for the columns in a space-efficient manner.

1.2.1 Our Approach We want to design a space-efficient ZK-SNARK for RAM computations. First, we fix the RAM model of computation as a machine that has (multi-pass) unidirectional input tapes and a work tape with RAM access. Our first step is to rely on the transformation from [4, 11] to transform the RAM computation into a (succinct) circuit C. We modify the compiler to generate directly a constraint system that can be consumed by the Ligero system.

In slightly more detail, the Ligero constraint system is over a $m \times \ell$ matrix X that represents the "extended witness" and instantiated via a linear constraint (A, b) and quadratic constraint system specified by tuples of rows (i_l, i_r, i_o) on the matrix X. The linear constraint requires that $Ax = b$ where x is the flattening of the matrix X (namely concatenating the rows of X) and the quadratic constraint (i_l, i_r, i_o) requires that for every $j \in [\ell]$, $X_{i_l,j} \cdot X_{i_r,j} = X_{i_o,j}$.

By relying on the transformation of [4, 11], we will obtain a Ligero constraint system over a $\widetilde{O}(T/S) \times \widetilde{O}(S)$ matrix X where we can decompose X into $\widetilde{O}(T/S)$ blocks where a block, denoted by X_i, contains $\mathsf{polylog}(T)$ rows of X with the following properties:

1. First, a block can be stored in space $\widetilde{O}(S)$ (as opposed to storing X which requires $\widetilde{O}(T)$ space). The transformation will allow the prover to generate and encode X block-wise as needed by the Ligero proof system while using only $\widetilde{O}(S)$ space.
2. The linear and quadratic constraints over the extended witness X will be localized to a block or consecutive blocks i.e. these constraints only involve values within a block or consecutive blocks of X. We will show that this allows us to test constraints block-wise in a space efficient manner.

Next, we explain the main technical novelty of our approach - implementing the linear and quadratic tests.

Linear Test. In this step, the prover convinces the verifier that the extended witness X satisfies all the linear constraints. We observe from [4, 11] that the linear constraints are "localized" to blocks of size $\widetilde{O}(S)$ and "uniform" i.e., the set of the constraints applied to each block are the same. The efficiency of the linear constraints relies on these two properties. In more detail, we express the linear constraints for each block as $Ay_i = b$ where A is a public matrix of size $\widetilde{O}(S) \times \widetilde{O}(S)$ extracted from the transformation, b is a public vector of size $\widetilde{O}(S)$ and y_i is $\widetilde{O}(S)$-sized flattened vector corresponding to block Y_i that is obtained by concatenating the rows of Y_i.

We briefly describe the linear test for "uniform" constraints. To verify these constraints, the witness is split into blocks y_i and the verifier verifies that $r^{\mathbb{T}}(Ay_i) = r^{\mathbb{T}}b$ for all blocks y_i, where r is a random challenge it provides of length $\widetilde{O}(S)$. We explain the rest of the test for a specific block. To apply batching, the output of the batched test is takes as the random linear combination of the individual tests. In such a test, the prover rearranges the vector $r^{\mathbb{T}}A$ as an $\widetilde{O}(1) \times \widetilde{O}(S)$ matrix and computes its Interleaved Reed Solomon encoding, denoted by R. Then, instead of sending $r^{\mathbb{T}}(Ay_i)$, the prover sends the vector $q = (\mathbf{1}_m)^{\mathbb{T}}(R \odot U_i)$ where U_i is an encoding of Y_i, $\mathbf{1}_m$ is the all ones length m vector and $\odot$ denotes pointwise product. By the multiplicative property of Reed-Solomon codes, it follows that checking whether $r^{\mathbb{T}}(Ay_i) = r^{\mathbb{T}}b$ is equivalent to checking whether the decoding of q satisfies that the sum of the decoded values equals $r^{\mathbb{T}}b$. Towards making this test efficient in terms of both time and space, the following three steps need to be computed efficiently.

1. The prover and the verifier need to compute $r^{\mathbb{T}}A$. Note that naively storing the entire matrix A requires space $\widetilde{O}(S^2)$. Instead, we observe the matrix A benefits from the following properties of the circuit (which is obtained from the RAM-to-Circuit reduction of [4, 11]): (a) each wire of the circuit is involved in at most polylogarithmic linear and quadratic constraints and (b) all constraints involving a particular wire can be efficiently identified. This translates into the following properties for A: (a) A is a sparse matrix i.e., the number of non-zero elements in A is $\widetilde{O}(S)$ and (b) all the non-zero elements of a column can be efficiently computed in time $\widetilde{O}(1)$ and space $\widetilde{O}(1)$. To perform the matrix-vector multiplication, we just need to query the non-zero values for each column of A in time $\widetilde{O}(1)$ and then multiply each of these non-zero values with the appropriate randomness in r. The randomness associated with i^{th} row is set to s^i where s is a randomly generated seed. Hence, we can compute each element of $r^{\mathbb{T}}A$ in time $\widetilde{O}(1)$ and space $\widetilde{O}(1)$.
2. Next, both the prover and the verifier need to compute the encoding of $r^{\mathbb{T}}A$. The prover rearranges the $\widetilde{O}(S)$-length vector into a $\widetilde{O}(1) \times \widetilde{O}(S)$ matrix and then encodes each row using an RS encoding, denoted by R. The prover can do this by first interpolating each row i of the matrix to generate a polynomial $r_i(\cdot)$ and then evaluate $r_i(\cdot)$ at $\widetilde{O}(S)$ evaluation points; performing interpolation followed by evaluation (of size $\widetilde{O}(S)$) is done efficiently using iFFT followed by FFT and requires space $\widetilde{O}(S)$. The prover can perform these operations, but the verifier has much less space i.e., $\mathsf{poly}(\log T, \kappa)$. First, note that the verifier needs to compute only at $O(\kappa)$ columns of R (as opposed to the prover who needs to compute the entire codeword, which is of size $\widetilde{O}(S)$). However, this does not directly reduce the space to $\widetilde{O}(1)$ as interpolation followed by evaluation requires space $\widetilde{O}(S)$ to store the interpolated polynomials. By exploiting the structure of FFTs, we present an algorithm DEval that can implicitly evaluate the polynomial without storing all the coefficients at a particular point using $\widetilde{O}(1)$ space given an input of size $\widetilde{O}(S)$. This algorithm will allow the verifier to recompute the result of the linear test on the $\widetilde{O}(1)$ columns in $\widetilde{O}(1)$ space.
3. Lastly, the verifier needs to check if the prover's response in the linear test encodes values that sum up to $r^{\mathbb{T}}b$. Suppose $q(\cdot)$ is the polynomial associated with the prover's response, then the verifier needs to evaluate $q(\cdot)$ at ℓ points and check if they sum up to $r^{\mathbb{T}}b$ i.e., $\sum_{i\in[\ell]} q(\zeta_i) = r^{\mathbb{T}}b$ where $\{\zeta_i\}_{i\in[\ell]}$ are the interpolation points. It is non-trivial to ensure that both the time and space are optimal for this check as evidenced by the following two approaches where one is optimal in time but not in space and vice-versa.

   (a) If we use FFTs to evaluate the polynomial at ℓ points, then the check is optimal in time but not space i.e., this approach requires time $O(\ell \log \ell)$ and space $O(\ell)$.

(b) Alternately, instead of storing all ℓ evaluations of $q(\cdot)$ and then adding them up, we can compute the running aggregate of the values encoded by $q(\cdot)$ while simultaneously evaluating the polynomial at all ℓ points. This approach updates the running partial aggregate as the terms of the polynomial are computed and just needs to store 1 field element. But the time to evaluate a t degree polynomial at ℓ points is at least $O(t\ell)$, which is $O(\ell^2)$ when $t = \ell$. Hence, this approach is optimal in terms of space but not time i.e., it requires space $O(1)$ and time $O(\ell^2)$ (if the degree of $q(\cdot)$ is ℓ).

We address this issue by setting the interpolation points to be the ℓ^{th} roots of unity. It turns out that the sum of the values encoded by $q(\cdot)$ is equal to $\ell(c_0 + c_\ell)$ i.e., $\sum_{i\in[\ell]} q(\zeta_i) = \ell(c_0 + c_\ell)$ where c_0 and c_ℓ are the coefficients of $q(\cdot)$. Our time and space-optimal approach is as follows. The prover sends only the coefficients c_0 and c_ℓ during the linear test. The verifier sums up the two coefficients and checks if it is equal to $r^{\mathbb{T}}b$ i.e., $c_0 + c_\ell = r^{\mathbb{T}}b$, which requires time $O(1)$ and space $O(1)$.

Quadratic Test. Similar to the linear constraints, the quadratic constraints are "localized" to a block i.e., the constraints involve only values within a block of X. Further, the quadratic constraints require the rows of X to be aligned in a specific way: the left, right, and output wire values of multiplication gates are aligned in corresponding rows of a block. During the test, The verifier provides a vector r' of length $\widetilde{O}(1)$ and tries to verify the following for all blocks $i \in [O(T/S)]$, $r'^T(Y_i^{\text{left}} \odot Y_i^{\text{right}} - Y_i^{\text{out}}) = [0]_{1\times\widetilde{O}(S)}$ where Y_i^{left}, Y_i^{right} and Y_i^{out} are submatrices of X is size $\widetilde{O}(1) \times \widetilde{O}(S)$ corresponding to left, right and output wire values respectively (and they are all aligned). Towards this, the prover computes the encoding of $r'^T(Y_i^{\text{left}} \odot Y_i^{\text{right}} - Y_i^{\text{out}})$ for each block and then combines them by taking a random linear combination of such encodings.

Similar to the linear test, the verifier needs to additionally check if the prover's response encodes a vector of all zeros. A similar challenge as described in the linear test arises here as well. The verifier needs to evaluate the prover's response to the quadratic test, say $q(\cdot)$, at ℓ points and check if each of them is 0. Like the previous solution for linear test, we set the interpolation points to be the ℓ^{th} roots of unity. However, the solution for the previous step cannot be directly applied here as we need to check if each of the values is 0 (instead of the sum being 0). Instead, we observe that the polynomial $q(\cdot)$ can be expressed as a product of two polynomials $q'(\cdot)$ and $z(\cdot)$ such that $z(\cdot)$ evaluates to zero at all the interpolation points. We modify the quadratic test so that the prover sends $q'(\cdot)$ instead instead of $q(\cdot)$ and the verifier computes $q(\cdot)$ from $q'(\cdot)$ and $z(\cdot)$ where $z(\cdot)$ is a publicly known polynomial. This entirely avoids the need to check if $q(\cdot)$ encodes all 0 values.

IPCP to ZK-SNARK. We compile an IPCP to a ZKSNARK in two steps. First we compile an IPCP to a ZKIPCP and then transform ZKIPCP to a ZKSNARK.

In the first step, we need to ensure that the information revealed to the verifier is "zero-knowledge". Recall that information regarding the extended witness is revealed in each of the code, linear and quadratic tests and in the symbols (i.e. columns of the U matrix) queried by the verifier. The columns revealed can be protected by adding redundancy to the encoding. More precisely, we instantiate the Reed-Solomon code so that the columns of U matrix provide t-privacy as a secret sharing scheme where t is the number of symbols opened by the verifier. To make sure that the result of the tests leak no information, it suffices to mask the results by adding additional rows to the U matrix that blind the results of the tests. The IPCP protocol can be converted into ZKIPCP protocol without any additional overhead by : 1) adding blinding random codewords to encoded witness U and 2) adding randomness while generating U. This compilation only incurs a constant multiplicative overhead.

In the second step we rely on the compilation of Ben-Sasson et al. [6] (which in turn is based on [25]) using Merkle trees. We argue that this step affects the asymptotic computation or communication complexity only by a multiplicative factor proportional to $\mathsf{poly}(\kappa)$ where κ is computational security parameter.

Efficiency. To get the target space and time efficiency we will set the β parameter (length of a block) of the proof system to be $\widetilde{O}(S)$ and get a proof length of $\widetilde{O}((T/S)+S)$. The prover requires $\widetilde{O}(T)$-time and $\widetilde{O}(S)$-space, which is complexity-preserving. Further, the verifier is "succinct" and will require $\widetilde{O}(T/S+S)$-time and $\widetilde{O}(1)$-space to verify the proof.

Improving proof length. To improve the proof length, the protocol does not send the polynomials $q(\cdot)$ in the test. Instead, the polynomials generate a codeword which will be used as an oracle. The prover proves the degree of the polynomial using a low-degree testing protocol FRI [2] which requires polylogarithmic communication in the degree of polynomial, thereby reducing the proof length to $\widetilde{O}(T/S)$ and preserving the time and space complexities of both the prover and the verifier.

1.2.2 A Matching Lower Bound We complement our positive result with a lower bound that demonstrates why getting a proof length better than $\widetilde{O}(T/S)$ will be hard using current techniques. As mentioned above, all techniques involve codes with constant distance in one form or another. We show that any code that makes polylogarithmic passes on an input message of length n and produces a code with constant distance must require space $\widetilde{O}(n)$. Interpreting the result in the context of proof systems, if we want to generate a code of a message of length T in quasilinear time, it will require space $\Omega(T)$. A slightly more refined implication is that with space S, encoding a T-length message in quasilinear time (in T), can yield a code of distance at most $S \cdot \mathsf{polylog}(T)/T$. Testing such codes typically requires queries inversely proportional to the distance i.e. $T/(S \cdot \mathsf{polylog}(T))$. Hence, any proof system that employs such a code and encodes a length T message, will need a query complexity of at

least $T/(S \cdot \mathsf{polylog}(T))$, implying that the proof length will also be at least $T/(S \cdot \mathsf{polylog}(T))$.

The high-level idea of the lower bound is to prove that for any constant-distance code over a field $\mathbb{F}$, the encoding algorithm requires space $S > (n/(r\delta) - O(\log m)) \cdot \log |\mathbb{F}|$, where n is the message length, m is the codeword length, δ is the distance and r is the number of passes. We prove this in two steps.

First, consider an encoding algorithm that reads each block (i.e., contiguous portion) of a message and outputs a portion of the codeword. We show that there must be a message M that consists of a block of length $O(n/\delta)$ such that the number of elements output by the encoding algorithm corresponding to that block is $\delta m/2$.

Next, consider the set of all messages m' that agree with m everywhere except on that block (there are $|\mathbb{F}|^{O(n/\delta)}$ such messages). We show that there will be a subset of messages, say D, of size at least $|\mathbb{F}|^{O(n/\delta) - rS/\log|\mathbb{F}| - O(r\log m)}$ such that the encoding of any two messages will differ only in at most $\delta m/2$-elements where r is the number of passes made by the encoding algorithm on the input. If this set has at least two messages, then the encodings of these messages will differ in at most $\delta m/2$ locations, thereby violating the distance property of the codeword whose minimum distance is δm. To evade this contradiction, we require the size of D to be at most 1, i.e., $|\mathbb{F}|^{O(n/\delta) - rS/\log|\mathbb{F}| - O(r\log m)} \leq 1$ which implies that the space $S > (n/(r\delta) - O(\log m)) \cdot \log |\mathbb{F}|$.

1.3 A Comparison with Related Work

Related to the design of sub-linear zero-knowledge arguments, the work of Mohassel, Rosulek and Scafuro [29] constructs zero-knowledge arguments when modeling the NP relation via a RAM program, that are sublinear in a different sense. More precisely, they considered the scenario of a prover that commits to a large database of size M, and later wishes to prove several statements of the form $\exists w$ such that $\mathcal{R}_i(M, w) = 1$. After an initial setup with a computational cost of $O(M)$ only on the prover's side, they achieve computation and communication complexities for both parties that are proportional to $\widetilde{O}(T)$ where T is the running time of the RAM program implementing the relation and $\widetilde{O}$ hides a factor of $\mathsf{poly}(\log(T), \kappa)$.

Previously, the two works [9, 10] also designed black-box constructions of ZK-SNARKs with polylogarithmic overhead in time and space. These works rely on the hardness of discrete logarithm and hidden order groups. Our protocol, on the other hand, relies on symmetric key operations and requires collision-resistant hash functions. The prover's time and space complexities of [9, 10] match our complexity. This is the case for the verifier's space complexity as well. The verifier's running time in [9] is $\widetilde{O}(T)$ and $\widetilde{O}(n)$ in [10] where n is the input length. On the other hand, our verifier's complexity is $\widetilde{O}(T/S + S)$. Finally, the communication complexity of prior works is $\widetilde{O}(1)$ while we achieve $\widetilde{O}(T/S)$. We summarize these results in Table 1.

	$\mathcal{P}$ time	$\mathcal{P}$ space	$\mathcal{V}$ time	$\mathcal{V}$ space
[9]	$\widetilde{O}(T)$	$\widetilde{O}(S)$	$\widetilde{O}(T)$	$\widetilde{O}(1)$
[10]	$\widetilde{O}(T)$	$\widetilde{O}(S)$	$\widetilde{O}(n)$	$\widetilde{O}(1)$
Theorem 1	$\widetilde{O}(T)$	$\widetilde{O}(S)$	$\widetilde{O}(T/S+S)$	$\widetilde{O}(1)$

Table 1. The complexity analysis of black-box ZKSNARKs. T and S are the respective time and space complexities required by the RAM program to verify the NP relation. n is the input length and $\widetilde{O}(\cdot)$ ignores polynomial factors of $\log T$.

2 Preliminaries

Basic notations. Let κ be the security parameter. We use lower-case letters such as x,y to represent vectors, and $x[i]$ denotes the i^{th} element in vector x. We use capital letters such as X, Y to represent matrices. Also, $X[j]$ denotes the j^{th} column and $X_{i,j}$ denote the element in i^{th} row and j^{th} column in matrix X. We use the notation $\widetilde{O}(.)$ to ignore $\mathsf{polylog}(.)$ terms. A matrix X is said to be *flattened* into a vector x (i.e. denoted by the lower-case letters of the corresponding matrix), if x is a rearrangement of the matrix X row-wise i.e., $x = (X_{1,1}, \ldots, X_{1,n}, \ldots, X_{m,1}, X_{m,n})$ where X is of size $m \times n$.

We also X_i or Y_i to denote matrices, especially when there many such matrices and i identifies a specific matrix in the set $\{X_i\}_{i\in[n]}$. Note that the flattened vector associated with X_i and Y_i are denoted by corresponding lower-case letters i.e. x_i and y_i respectively.

2.1 Circuit Notations

A arithmetic circuit C is defined over a field $\mathbb{F}$ and has input gates, output gates, intermediate gates, and directed wires between them. Each gate computes addition or multiplication over $\mathbb{F}$. We define the notion of a *transcript* for an arithmetic circuit C to be an assignment of values to the gates where the gates are ordered in a lexicographic order; each gate in circuit C will have a gate id $\mathsf{g_{id}}$ and will have two input wires and one output wire. Each wire will also have a wire id $\mathsf{w_{id}}$ and in case a wire value is an output wire of gate $\mathsf{g_{id}}$, then the wire id $\mathsf{w_{id}} = \mathsf{g_{id}}$. Each element in the transcript W is of the form $(\mathsf{g_{id}}, \mathsf{type}, \gamma)$ where $\mathsf{g_{id}}$ is the gate label, $\mathsf{type} \in \{\mathsf{inp}, \mathsf{add}, \mathsf{mult}, \mathsf{out}\}$ is the type of the gate and γ is the output wire value of gate $\mathsf{g_{id}}$.

2.2 Zero-Knowledge Arguments

A zero-knowledge argument system for an NP relationship $\mathcal{R}$ is a protocol between a computationally-bounded prover $\mathcal{P}$ and a verifier $\mathcal{V}$. At the end of the protocol, $\mathcal{V}$ is convinced by $\mathcal{P}$ that there exists a witness w such that $(x; w) \in \mathcal{R}$ for some input x, and learns nothing beyond that. We focus on arguments of knowledge which have the stronger property that if the prover convinces the verifier of the statement validity, then the prover must know w.

Formally, consider the definition below, where we assume $\mathcal{R}$ is known to $\mathcal{P}$ and $\mathcal{V}$.

Definition 1. *Let $\mathcal{R}(x,w)$ be an NP relation corresponding to an NP language L. A tuple of algorithm $(\mathcal{P},\mathcal{V})$ is an argument of knowledge for $\mathcal{R}$ if the following holds.*

- ***Correctness**. For every $(x,w) \in R$ and auxiliary input $z \in \{0,1\}^*$, it holds:*

$$\langle \mathcal{P}(w), \mathcal{V}(z)\rangle(x) = 1$$

- ***Soundness**. For every $x \notin L$, every (unbounded) interactive machine $\mathcal{P}^*$, and every $w,z \in \{0,1\}^*$ and a large enough security parameter λ,*

$$\mathsf{Pr}[\langle \mathcal{P}(w), \mathcal{V}(z)\rangle(x) = 1] \leq \mathsf{negl}(\lambda)$$

It is a zero-knowledge argument of knowledge it additionally satisfies:

- ***Zero knowledge**. There exists a PPT simulator $\mathcal{S}$ such that for any PPT algorithm $\mathcal{V}^*$, auxiliary input $z \in \{0,1\}^*$, and $(x;w) \in \mathcal{R}$, it holds that*

$$\mathsf{View}(\langle \mathcal{P}(w), \mathcal{V}(z)^*\rangle(x)) \approx \mathcal{S}^{\mathcal{V}^*}(x,z)$$

Here $\mathcal{S}^{\mathcal{V}^*}$ denotes that the simulator $\mathcal{S}$ sees the randomness from a polynomial-size space of $\mathcal{V}^*$.

Succinct vs. Sublinear Arguments. We say an argument of knowledge is *succinct* if there exists a fixed polynomial $p(\cdot)$ such that the length of the proof is is bounded by $p(\lambda + \log|C|)$ where C is the circuit corresponding to the NP relation. Similarly, we say an argument of knowledge is *sublinear* if the proof length is $o_\lambda(|C|)$ where $o_\lambda(\cdot)$ hides multiplicative factors dependent on the security parameter λ.

2.3 Random-Access Machines (RAM)

A Random-Access Machines (RAM) comprises of a finite set of instructions that are executed sequentially on a finite set of registers and can make arbitrary memory accesses. We assume that each time step during the execution of a RAM program executes a single instruction or accesses the memory locations. We model the RAM as a Reduced-Instruction Set Computer (RISC) which more closely models programs compiled from high-level languages such as Java, C++. We adopt the formal notation for RAM from [4].

Definition 2. *(The RAM Model [4]) A random-access machine (RAM) is a tuple $M = (w,k,\mathbb{A},\mathcal{C},\mathcal{T})$, where:*

- $\mathsf{w} \in \mathbb{N}$ *is the register size;*
- $k \in \mathbb{N}$ *is the number of registers;*
- $\mathcal{C} = (I_0,\ldots,I_{n-1})$ *is a set of instructions (or the code for the RAM program), where $n \in \{1,\ldots 2^{\mathsf{w}}\}$ and each I_i is an instruction.*

- $\mathcal{I}$ *is a set of tapes which consists of a constant number of unidirectional input tapes with read-only access and single unidirectional output tape.*
- $\mathbb{W}$ *is a work tape with arbitrary read and write accesses.*

Consider a RAM program M that runs in time $T(n)$ and uses $S(n)$ memory cells on input x with n-bits. For simplicity, we use T and S instead of $T(n)$ and $S(n)$ as the input length can be easily inferred.

The RAM program M has arbitrary access to a work tape[3]. At any time step, the RAM program may read from or write into the cells (also referred to as memory cells) of the work tape using the load and store instructions respectively. We say a *RAM program uses space* S, if at most S memory cells of the tape were accessed during an execution of the M.

2.4 Succinct Matrix

We define succinct matrices which will be used in our zero-knowledge argument system.

Definition 3 (Succinct Matrix). *A succinct matrix A is a matrix of dimension $n_1 \times n_2$ with the following properties:*

- *There are $n_1 \cdot \mathsf{polylog}(n_1)$ non-zero values.*
- *There exists an algorithm getColumn$(\cdot)$ that takes input j and outputs a list L. The list L contains all non-zero elements of column j where each non-zero element is represented as a tuple (k, val) where k represents the row number and val represents the non-zero value. This algorithm runs in $\mathsf{polylog}(n_1)$.*

3 Lower Bound for Space-Efficient Encoding Schemes

In this section, we present our lower bound on space-efficient constant-distance codes. This lower bound provides evidence for why it is unlikely for current proof systems to be complexity preserving (in both time and space) when the underlying RAM machine uses space $S << T$ for non-trivial space.

3.1 Interpreting the Lower Bound in the Context of Proof Systems

As mentioned in the technical overview, all constructions of succinct non-interactive arguments based on symmetric-key primitives that are black-box in the underlying assumptions rely on constant-distance codes [25, 5, 3, 1]. In slightly more detail, all constructions first rely transforming the circuit evaluation to an "execution" transcript that is proportional to the size of the circuit and then encoding the transcript via a constant-distance code. For a RAM machine, such a transformation typically results in a transcript of size T where T is the running time of the RAM computation. In this section, we will show that

[3] Generally, the RAM program has access to memory which we model as a work tape.

encoding a T-element message via a constant-distance code will require space $\Omega(T/r)$, where r is the number of passes taken by the algorithm on the input tape. In other words, current techniques for constructing a time-preserving ZK-SNARK, i.e. r is at most $\mathsf{polylog}(T)$, will require prover's space of $\Omega(T/r)$. In particular if the space of the underlying RAM machine is $S << T$, it is unlikely to get such a proof system that is complexity preserving (in time and space).

3.2 Warm Up: A Simple Lower Bound

As a warm up, we first present a lower bound where we assume a small restriction on the encoding algorithm and then prove a more general result. We begin with some notation that will help in our lower bounds.

Notation. We consider an encoding algorithm executed via a RAM machine with space S that encodes a message of length n. The encoding algorithm has unidirectional (i.e. linear) access to the input tape and can make multiple passes on the input. The machine also has a unidirectional output tape. Further, the the encoding algorithm has RAM access to a work tape of size S bits (or equivalently $S/\log|\mathbb{F}|$ field elements). To keep track of the current position of the read head of the encoding algorithm on the input tape, we introduce the notion of *head*. Specifically, we use read head and write head to denote the heads in the input tape and output tapes, respectively (where the message to be encoded is read from the input tape and the codeword is written on to the output tape). Note that the contents of the work tape of the encoding algorithm differs depending on the position of the read head. It will be convenient to divide a msg into contiguous blocks of equal length. We will denote by by $\mathsf{msg}[i]$ the i^{th} block of msg. We denote by $\mathsf{c_{msg}}$ the output of Enc on input msg. Let $\mathsf{c_{msg}}[i,j]$ denote the part of the codeword output by the encoding algorithm when it reads the block $\mathsf{msg}[i]$ during the j^{th} pass, i.e. when the read head moves from the left end to the right end of the block $\mathsf{msg}[i]$ in the j^{th} pass. Let $\mathsf{c_{msg}}[i]$ be the concatenation of $\{\mathsf{c_{msg}}[i,1],\ldots,\mathsf{c_{msg}}[i,r]\}$. We will drop the subscript when the msg is understood from the context.

We present a high-level overview of the simplified version of the lower bound, which imposes certain restrictions on the encoding algorithm. Note that the encoding algorithm reads a certain portion of the message (referred to as a block), outputs a portion of the codeword (associated with this block) and then proceeds to the next message block. We make a simplifying assumption that the length of the codeword portion associated with any message block is independent of the contents of the message. This is formally stated in assumption 1 below.

Assumption 1 *The position of the read head and write head at any step during the encoding is independent of the message.*

As a corollary we have the following: Suppose we divide the input message into $\lceil 2/\delta \rceil$ blocks of equal length. Given any two messages $\mathsf{msg}, \mathsf{msg}' \in \mathbb{F}^n$, the

output of the encoding algorithm satisfies $|\mathsf{c}_{\mathsf{msg}}[i,j]| = |\mathsf{c}_{\mathsf{msg}'}[i,j]|$ for all blocks $i \in [2/\delta]$ and passes $j \in [r]$.

We begin with a proof overview. On a high-level the idea is to identify a set of messages whose encoding violate the minimum distance property. First, by a simple counting argument we can argue that there must be a message block t of length $O(n/2\delta)$ such that the total number of elements output by the encoding algorithm when the read head passes through block t (i.e. $\sum_j |c[t,j]|$) is at most $\delta m/2$. Observe that block t will have the same property for any message by our Assumption 1. Next, we will focus on messages that are identical everywhere except on block t; if we fix the remaining blocks then there are $|\mathbb{F}|^{\delta n/2}$ such messages as each block is of size $\delta n/2$. Out of these $|\mathbb{F}|^{\delta n/2}$ messages, we identify a subset of messages that result in identical work tapes after the encoding algorithm reads block t in each pass. These messages have property that the code can only differ in the portions output when reading block t, namely $c[t,j]$. We conclude by showing that there exist at least two messages in this set when $S \leq (\delta n/2r) \cdot \log |\mathbb{F}|$. Since the codewords corresponding to these messages only differ in at most $\delta m/2$ locations, but the minimum distance of the code is δm, we arrive at a contradiction.

Theorem 3. *Let $\mathcal{C}$ be a $[m,n,\delta m]$ code over $\mathbb{F}$ with message length n, codeword length m and minimum relative distance δ. Also, let $\mathsf{Enc}_{(T,S,r)} : \mathbb{F}^n \to \mathbb{F}^m$ be a Turing machine that on input $\mathsf{msg} \in \mathbb{F}^n$ outputs an encoding of msg in time T with a work tape of size S while making r passes on the message. Suppose Assumption 1 holds, then $S \geq (\delta n/2r) \cdot \log |\mathbb{F}|$.*

Proof. Assume for contradiction that there exists a $[m,n,\delta m]$ code $\mathcal{C}$ over $\mathbb{F}$ with an encoding algorithm $\mathsf{Enc}_{(T,S,r)}$. Consider an arbitrary message msg. Let's partition it into $2/\delta$ blocks each of length $\delta n/2$ elements.

Assumption 1 implies that the length of the output of the encoding algorithm associated with message block i, which is denoted by $|\mathsf{c}[t]|$, is the same for all messages. Here, $\mathsf{c}[i]$ is a concatenation of $\{\mathsf{c}[i,1],\ldots,\mathsf{c}[i,r]\}$ for some message msg. We drop the subscript for $|\mathsf{c}_{\mathsf{msg}}[t]|$ as the length is the same for all messages. Next, We show that there exists a message block t such that $\mathsf{c}[t]$ is of length at most $\delta m/2$.

Lemma 1. *There exists a $t \in [2/\delta]$ such that $|\mathsf{c}[t]| \leq \delta m/2$.*

Proof. Assume for contradiction, for every t, $|\mathsf{c}[t]| > \delta m/2$. Then,

$$|\mathsf{c}| \ = \sum_i |\mathsf{c}[i]| > 2/\delta \times \delta m/2 > m$$

which is a contradiction.

Lemma 2. *Given message block $t \in [2/\delta]$ and pass $k \in [r]$, there exists a set of messages D_k of size at least $|\mathbb{F}|^{\frac{\delta n}{2} - kS/\log|\mathbb{F}|}$ such that for any two messages $\mathsf{msg}, \mathsf{msg}' \in \mathsf{D}_k$ the following holds:*

1. $\mathsf{msg}[i] = \mathsf{msg}'[i]$ *for all* $i \neq t$.
2. *At the end of the* k^{th} *pass,* $\mathsf{c}_{\mathsf{msg}}$ *and* $\mathsf{c}_{\mathsf{msg}'}$ *differ only at positions occupied by* $\mathsf{c}[t,1], \ldots, \mathsf{c}[t,k]$. *Furthermore, the contents of the work tape of the encoding algorithm at the end of the* k^{th} *pass for messages* msg *and* msg' *will be identical.*

Proof. Consider an arbitrary message msg, define D_0 to be the set of all messages that are identical to msg in every block $i \neq t$, but differ in block t. D_0 contains $|\mathbb{F}|^{\frac{\delta n}{2}}$ messages. We prove the claim via an induction on the number of passes.

Base case: In the first pass, we show that there exists $D_1 \subseteq D_0$, such that the properties of the claim hold. By an averaging argument, there must exist a subset of D_0, say D_1, of size at least $|\mathbb{F}|^{\frac{\delta n}{2} - \frac{S}{\log|\mathbb{F}|}}$ with the following property: the contents of the work tapes are identical for any two messages in D_1 after the encoding algorithm finishes reading block t message during the first pass.

We now show that the codewords $\mathsf{c}_{\mathsf{msg}}$ and $\mathsf{c}_{\mathsf{msg}'}$ differ only in the codeword portions $\mathsf{c}[t,1]$ for any two messages $\mathsf{msg}, \mathsf{msg}' \in D_1$. Since all messages in D_0 are identical except for block t, the encoding will be identical before the codeword portion $\mathsf{c}[t,1]$. Next, since the contents of work tapes after reading block t are identical and the remaining part of the message (i.e., after message block t) are the same, the rest of the output until the encoding finishes the first pass will be the identical as well. Furthermore, the work tapes will be identical when the encoding finishes the first pass.

Induction: Suppose that there exists a set of messages D_k for which the conditions of the claim holds at the end of the k^{th} pass. Then in the $(k+1)^{st}$ pass, the encoding algorithm starts with identical contents on the work tape for every message in D_k, so it will output the same elements until the encoding reaches block t. Applying another averaging argument, there must exists a subset $D_{k+1} \subseteq D_k$ of size at least $|\mathbb{F}|^{\frac{\delta n}{2} - \frac{kS}{\log|\mathbb{F}|}} / |\mathbb{F}|^{S/\log|\mathbb{F}|} = |\mathbb{F}|^{\frac{\delta n}{2} - \frac{(k+1)S}{\log|\mathbb{F}|}}$ such that the work tape will be identical when the encoding finishes reading block t in the $(k+1)^{st}$ pass. Similarly the contents of the work tapes and the output for these messages will also be identical for every two messages in D_{k+1} until the end of the $(k+1)^{st}$ pass. This completes the induction step.

Finally, we combine lemmas 1 and 2 to prove theorem 3 via contradiction. As per lemma 1, there exists a message block t such that the encodings of messages differing only at this block have a distance of at most $\delta m/2$. If we instantiate lemma 2 for block t, we get that there exists a set of messages D_r of size $|\mathbb{F}|^{\frac{\delta n}{2} - \frac{rS}{\log|\mathbb{F}|}}$ such that the encodings of any two messages in D_r differ in at most $\delta m/2$ locations. If we set $S < (\delta n/2r) \cdot \log|\mathbb{F}|$, then D_r will contain at least 2 messages whose encodings differ in at most $\delta m/2$ locations. This contradicts the distance requirements of the codeword $C_{\mathbb{F},n,m,\delta}$ whose minimum distance is at least δm.

3.3 Lower Bound for Multi-Pass Space-Efficient Encoding Schemes

In this section, we extend the lower bound where we do not make Assumption 1. Without this assumption, for two different messages, the portion of the code affected by different blocks of the message could be different. The main idea to deal with the general case is to show that there exist sufficiently many messages for which Assumption 1 holds and then apply the preceding argument.

Theorem 4. *Let $\mathcal{C}$ be a $[m, n, \delta m]$ code over $\mathbb{F}$ with message length n, codeword length m and minimum relative distance δ. Also, let $\mathsf{Enc}_{(T,S,r)} : \mathbb{F}^n \to \mathbb{F}^m$ be a Turing machine that on input $\mathsf{msg} \in \mathbb{F}^n$ outputs an encoding of msg in time T with a work tape of size S while making r passes on the message. Then $S \geq (\delta n/4r - 2(\log_{|\mathbb{F}|} m) - 2/r) \cdot \log |\mathbb{F}|$.*

Proof. Assume for contradiction that there exists a code $\mathcal{C}$ and encoding algorithm Enc. We partition the message msg into $4/\delta$ blocks each of length $\delta n/4$. We first show that there exists a subset containing $|\mathbb{F}|^{\delta n/4-2}$ messages, say D, and an index t such that for each message msg in D, we have $|\mathsf{c}_{\mathsf{msg}}[t]| \leq \delta m/2$. Note however, that since Assumption 1 does not hold, the corresponding code blocks for these messages might not be aligned.

Lemma 3. *There exists a set of messages D of size at least $|D| \geq |\mathbb{F}|^{\delta n/4-2}$ and $t \in [4/\delta]$ such that for any two $\mathsf{msg}, \mathsf{msg}' \in D$ the following holds:*

1. $\mathsf{msg}[i] = \mathsf{msg}'[i]$ *for all* $i \neq t$.
2. $|\mathsf{c}_{\mathsf{msg}}[t]| \leq \delta m/2$.

Proof. Assume for contradiction that such a set D does not exist. Given a message msg, let $A_t[\mathsf{msg}]$ be the set of all possible messages that agree with msg on all blocks except block t. We know that the size of $A_t[\mathsf{msg}]$ is $|\mathbb{F}|^{\delta n/4}$. By our assumption, we have that for more than $|\mathbb{F}|^{\delta n/4} - |\mathbb{F}|^{\delta n/4-2}$ of the messages in $A_t[\mathsf{msg}]$, it holds that $\mathsf{c}[t]$ is of length bigger than $\delta m/2$. We will now compute

$$\sum_{s \in \{0,1\}^{n-\delta n/4}} \sum_{t} \sum_{s' \in \{0,1\}^{\delta n/4}} |\mathsf{c}_{Combine(s,s',t)}[t]|$$

where $Combine(a, b, i)$ denotes the string obtained by inserting b into string a at position $t \times \delta n/4$. Observe that the sum above, counts the sum total of the lengths of the encodings of every message, which should be equal to $m \times |\mathbb{F}|^n$. By our assumption, we can lower bound the sum as

$$\begin{aligned} |\mathbb{F}|^{n-\delta n/4} \times 4/\delta \times (|\mathbb{F}|^{\delta n/4} - |\mathbb{F}|^{\delta n/4-2}) \times \delta m/2 &= 2 \times |\mathbb{F}|^n \times (1 - 1/|\mathbb{F}|^2) \times m \\ &> |\mathbb{F}|^n \times m \end{aligned}$$

where the last step holds for $|\mathbb{F}| \geq 2$. This is a contradiction.

Next, we show that there are sufficiently many messages in D and indices t, such that the message block t influences identical portions of the codeword. In other words, the assumption we made for the warm-up proof holds for a subset of the messages in D.

Lemma 4. *Given any index $t \in [4/\delta]$, set D of messages there exists a subset of messages $D' \subseteq D$ of size at least $|D|/m^{2r}$ such that for all messages $\mathsf{msg}', \mathsf{msg}''$ in D', the starting and ending positions of $\mathsf{c}_{\mathsf{msg}'}[t,i]$ and $\mathsf{c}_{\mathsf{msg}''}[t,i]$ w.r.t the code are identical for every $i \in [r]$.*

Proof. There are overall $2r$ positions considering the starting and ending points of $\mathsf{c}_{\mathsf{msg}}[t,1], \ldots, \mathsf{c}_{\mathsf{msg}}[t,r]$ w.r.t the code. The number of possibilities for these $2r$ points is exactly $\binom{m}{2r}$ (because selection of $2r$ positions can be assigned as starting and ending positions uniquely to the code blocks). By an averaging argument there must be at least $\frac{|D|}{\binom{m}{2r}} \geq \frac{|D|}{m^{2r}}$ messages in D for which these $2r$ locations will be identical.

Combining Lemmas 3 and 4, we get that there exists a set B of size at least $|\mathbb{F}|^{\frac{\delta n}{4} - 2r\log_{|\mathbb{F}|}(m) - 2}$ and index t that satisfy the conditions in both the lemmas.

Lemma 5. *There exists a set of messages $D \subseteq B$ of size at least $|\mathbb{F}|^{\frac{\delta n}{4} - 2r(\log m) - \frac{rS}{\log|\mathbb{F}|} - 2}$ where the following properties hold for any messages $\mathsf{msg}, \mathsf{msg}' \in D$:*

1. $\mathsf{msg}[i] = \mathsf{msg}'[i]$ *for all* $i \neq t$.
2. *At the end of the k^{th} pass, $\mathsf{c}_{\mathsf{msg}}$ and $\mathsf{c}_{\mathsf{msg}'}$ differ only in the portions occupied by the blocks $\mathsf{c}[t,1], \ldots, \mathsf{c}[t,k]$. Furthermore, the contents of the work tape of the encoding algorithm at the end of the k^{th} pass will be identical.*

Proof. Observe that all messages in B have the property that they are identical on all blocks except at block t. Moreover, the starting and ending positions w.r.t the code when the encoding algorithm reads block t are identical for all messages in B. We can now follow essentially the same argument as Claim 2 to prove this claim.

We conclude the proof of Theorem 4 by observing that if D has at least two messages we arrive at a contradiction because for every message in D, $\mathsf{c}[t]$ is at most $\delta m/2$ and for any two messages the corresponding codes only differ in these locations. Thus, if $\frac{\delta n}{4} - 2r(\log_{|\mathbb{F}|} m) - \frac{rS}{\log|\mathbb{F}|} - 2 > 0$, we arrive at a contradiction.

4 Main Construction

In this section, we present a short overview of our space-efficient zero-knowledge argument system for RAM programs based on collision-resistant hash-functions. Please refer to the full version for a more detailed presentation.

The first main step in our construction is transforming a RAM program to the Ligero constraint system. This is summarized in the Lemma below.

Lemma 6. *Let M be an arbitrary (non-deterministic) Random Access Machine that on input strings (x, w) runs in time T and space S. Then, (M, x) can be transformed into the following system of constraints over a $m \times \ell$ matrix X:*

1. *X is a $m \times \ell$ matrix that is subdivided into sub-matrices or blocks $X_1, \ldots, X_B$ where each X_i is a $m' \times \ell$ matrix, $X = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_B \end{bmatrix}$ and $B = O\left(\frac{T}{S}\right), m' = \mathsf{polylog}(T)$, $m = m' \cdot B$ and $\ell = S \cdot polylog(T)$. We denote by x_i the "flattened" vector corresponding to matrix X_i (namely, x_i is the vector obtained by concatenating the rows of X_i).*
2. *(Intra-block Linear Constraints) A is of size $(m' \cdot \ell) \times (m' \cdot \ell)$ and b is a length $(m' \cdot \ell)$-vector and $Ax_i = b$ for all $i \in [B]$.*
3. *(Inter-block Linear Constraints) A' is a $(2m' \cdot \ell) \times (2m' \cdot \ell)$ matrix and b' is a length $(2m' \cdot \ell)$-vector and $A' \begin{bmatrix} x_i \\ x_{i+1} \end{bmatrix} = b'$ for all $i \in [B-1]$.*
4. *(Input-Consistency Constraint) A'' is a $|x| \times (m' \cdot \ell)$ matrix and $A''x_1 = x$ where $|x|$ is the size of x.*
5. *(Quadratic Constraints) For each $i \in [B]$, $X_i^{left} \odot X_i^{right} = X_i^{out}$ where $\odot$ denotes point-wise products and $X_i = \begin{bmatrix} X_i^{inp} \\ X_i^{left} \\ X_i^{right} \\ X_i^{out} \end{bmatrix}$ where X_i^{inp} is $m_{inp} \times \ell$ matrix and $X_i^{left}, X_i^{right}, X_i^{out}$ are $m_{mult} \times \ell$ matrices.*

Efficiency. *Furthermore, the matrices A, A' and A'' are succinct according to Definition 3 and an input-witness pair (x, w) that makes M accept can be mapped to an extended witness X by a RAM machine in $T \cdot polylog(T)$ and space $S \cdot polylog(T)$.*
Equivalency. *Any X that satisfies the system of constraints can be mapped to a w such that M accepts (x, w).*

The core of our construction is a space-efficient IPCP for the linear and quadratic tests. We will only focus on linear test in this version of the paper. For the full description of all the elements of our protocol, we refer the reader to the full version. A formal description of the linear test is given below.

Lemma 7. *Protocol 1 is an IOP/IPCP for testing linear constraints with the following properties:*

- **Completeness:** *If $U \in L^m$ is an encoding of a $m \times \ell$ matrix X such that, for every $i \in [B]$, $Ay_i = b$ where y_i is the flattened vector corresponding to Y_i and block Y_i is a $m' \times \ell$ submatrix of X starting at the $I[i]^{th}$ row of X and the $\mathcal{P}$ is honest, then $\mathcal{V}$ accepts with probability 1.*

Protocol 1 (Testing linear constraints over Interleaved RS Codes)
Input: *L^m-codeword U, #blocks B, vectors $\{y_i\}_{i\in[B]}$ each of length $m'\ell$, indices $\{I[i]\}_{i\in[B]}$, matrix A of size $m_a \times m'\ell$, vector b of length m_a.*
Oracle: *A purported L^m-codeword U that should encode $m \times \ell$ matrix X such that, for every $i \in [B]$ we have $Ay_i = b$ where y_i is the flattened vector corresponding to Y_i and block Y_i is a $m' \times \ell$ submatrix of X starting at the $I[i]^{th}$ row of X.*
Linear Test:

1. *$\mathcal{V}$ picks two random seeds $s, s' \in \mathbb{F}$ and sends it to $\mathcal{P}$.*
2. *$\mathcal{P}$ sends $q(\cdot) = \sum_{i\in[B]} r'[i]q_i(\cdot)$ to $\mathcal{V}$ where $r = (1, s, s^2, \ldots, s^{m_a-1})$, $r' = (1, s', s'^2, \ldots, s'^{B-1})$, $r^{\mathbb{T}}A = (r_{1,1}, \ldots r_{1,\ell}, \ldots, r_{m',1}, \ldots, r_{m',\ell})$ and $q_i(\cdot) = \sum_{k\in[m']} r_k(\cdot)p_{I[i]+k-1}(\cdot)$, and $r_i(\cdot)$ is the polynomial of degree $< \ell$ such that $r_i(\zeta_j) = r_{i,j}$ for all $j \in [\ell]$.*
3. *$\mathcal{V}$ queries a random subset $Q \subseteq [n]$ of size t to obtain the columns of U corresponding Q.*
4. *$\mathcal{V}$ accepts if*
   (a) *$q(\cdot)$ is of degree $< 2\ell - 1$.*
   (b) *$\sum_{k\in[\ell]} q(\zeta_k) = \sum_{i\in[B]} r'[i]r^{\mathbb{T}}b$.*
   (c) *For every $i \in Q$,*
   $\sum_{j\in[B],k\in[m']} r'[j] \cdot r_k(\eta_i)U_{k+I[j],i} = q(\eta_i)$.

Fig. 1. Protocol for Linear Test.

– **Soundness:** *Let e be a positive integer such that $e < d/2$ where d is minimal distance of Reed-Solomon code. Suppose that a badly formed matrix U^* is e-close to a codeword U that encodes a matrix X such that $\exists i \in [B]$, $Ay_i \neq b$ where y_i is the flattened vector corresponding to Y_i and block Y_i is a $m' \times \ell$ submatrix of X starting at the $I[i]^{th}$ row of X. Then for any malicious $\mathcal{P}^*$ strategy, $\mathcal{V}$ will reject except with probability $((e + 2\ell)/n)^t + (m_a + B)/|\mathbb{F}|$.*
– **Complexity:**
$\mathcal{P}$ has X on its input tape and has a work tape of size $O(m'\ell)$. In this model, $\mathcal{P}$ makes a single pass on the input tape. We denote by $m\ell$ the length of X, the number of blocks as B and y_i is a flattened vector of a block within X of size $m' \times \ell$. Given that $\mathcal{P}$ is provided with a one-way linear access to X, matrix A is a public succinct matrix of dimension $m_a \times m'\ell$ as defined in Definition 3 then the following complexities are obtained:
  - *Prover's Time = $m'\ell\,\mathsf{poly}\log m_a + O(m'\ell B \log \ell)$.*
  - *Verifier's Time = $m'\ell\,\mathsf{poly}\log m_a + O(m'\ell\kappa + Bm'\kappa)$*
  - *Prover's Space = $O(m'\ell)$.*
  - *Verifier's Space = $O(\kappa m' + m_a)$.*
  - *Communication Complexity = $O(\ell)$.*
  - *Query Complexity = $O(\kappa)$.*

Our IPCP protocol Given a RAM program M, we construct a zero-knowledge argument system for $\mathsf{BH}_{\mathsf{RAM}}(M)$ by composing the following two components.

1. A complexity-preserving reduction from $\mathsf{BH}_{\mathsf{RAM}}$ to extended witness X that satisfies the system of constraints defined in lemma 6.
2. The protocols for testing interleaved linear codes, linear constraints and quadratic constraints require oracle access to L^m-codeword U that encodes the extended witness X. The prover computes the outputs of the tests by processing X block-by-block. The verifier has a "succinct" representation of the system of constraints imposed on X and can therefore check the outputs of the tests in a space-efficient manner.

We compose these two components as follows. At a high level, the prover generates the extended witness X block-by-block as described in the reduction from $\mathsf{BH}_{\mathsf{RAM}}$ to X. As and when a block is generated, the prover processes this block to compute "running partial outputs" for each of the three tests. The prover only needs to store a few blocks in memory at a time rather than the entire extended witness.

Theorem 5. *Fix parameters $m, m', m_{\mathsf{mult}}, n, \ell, B, t, e, d$ such that $e < d/3$ and $d = n - \ell + 1$. For every NP relation that can be verified by a time T and space S RAM machine M with input x has a public-coin IOP/IPCP with the following properties:*

1. *Completeness: If there exist an witness w such that $M(x, w)$ with time T and space S is accepted and $\mathcal{P}$ generated the oracle U honestly, then $\mathcal{V}$ accepts with probability 1.*
2. *Soundness: Let there exist no witness w such that $M(x, w)$ is accepted in time T and space S, then for every unbounded prover strategy $\mathcal{P}^*$, $\mathcal{V}$ will reject except with $(1 - e/n)^t + 4((e + 2\ell)/n)^t + (d + 3m'\ell + m_{mult} + |x| + 3B)/|\mathbb{F}|$.*
3. *Complexity: The complexities are in terms of the number of field operations performed or number of field elements over a field $\mathbb{F}$ below.*
   (a) The prover runs in time $T \cdot \mathsf{poly}(\log T, \kappa)$ and uses space $S \cdot \mathsf{poly}(\log T, \kappa)$.
   (b) The verifier runs in time $(T/S + S) \cdot \mathsf{poly}(\log T, \kappa)$ and uses space $\mathsf{poly}(\log T, \kappa)$.
   (c) The communication complexity is $S \cdot \mathsf{poly}(\log T, \kappa)$, query complexity of the verifier is $(T/S) \cdot \mathsf{poly}(\log T, \kappa)$ and number of rounds is a constant.

where κ is the statistical security parameter.

In the full version, we also show how to modify our IPCP to obtain zero-knowledge and then how to improve the communication to $\widetilde{O}(T/S)$.

5 Space-Efficient Affine Code Testing for Interleaved Reed Solomon Codes

We begin by providing a high-level overview of our IPCP system. It follows the same blueprint of the Ligero system (described in the introduction). In a nutshell, our construction is a space-efficient variant of each phase of the Ligero blueprint. The main steps involved in our IPCP system are as follows.

1. **RAM to circuit reduction:** Given a RAM program M, we transform the RAM program to circuits by relying on the transformation of [4, 11] where the resulting circuit C has a "succinct" representation.
2. **Preparing the proof oracle:** Next, the prover evaluates the circuit C on the private witness w to compute all wire values (input, intermediate and output) and arranges the values in a specific way in a $m \times \ell$ matrix X referred to as the extended witness. The prover then encodes the matrix using a Interleaved Reed Solomon (IRS) code to obtain a $m \times n$ matrix U, namely, each row of U is an encoding of a corresponding row in X using a Reed Solomon code; this is the proof oracle. The prover computes U in a row-by-row manner, which is space-efficient.
3. **Testing the encoding:** This step involves testing interleaved linear codes in a space efficient manner. This essentially follows as in the previous step as $r^{\mathbb{T}}U$ can be computed by recomputing U row-by-row and maintaining a running partial aggregate of $\sum_j r_j U_{j,\cdot}$.
4. **Testing linear constraints:** This step shows how the linear test can be performed in a space efficient manner. This is the non-trivial part of the construction as we need to utilize the succinct representation of C and the arrangement of the extended witness in U to compute the response in a space-efficient manner.
5. **Testing quadratic constraints:** This step relies on ideas from the previous two steps to obtain a space-efficient version of the quadratic test.

In this section, we focus on our space-efficient IPCP for the linear test (mentioned in step 4 above), which is one of the core aspects of our construction. We defer to the reader to the full version of our paper for the descriptions of the rest of the steps mentioned above.

This test checks if the linear constraints imposed by the addition gates and the circuit's structure are satisfied. The linear check is performed over blocks, where each block Y_i starts at the row $I[i]$ of the extended witness X and is of size $m' \times \ell$ for all $i \in B$. Precisely, given a public matrix A of size $(m' \cdot \ell) \times (m' \cdot \ell)$ and a vector b of size $(m' \cdot \ell)$, the linear constraints are $Ay_i = b$ for each $i \in [B]$ where y_i is the flattened vector of Y_i and is of size $(m' \cdot \ell)$. Note that linear constraints imposed on each block are the same, which are captured by same parameters A and b for all blocks.

Recall that the linear test in Ligero handles all the linear constraints over the extended witness X in a single shot (represented as a linear equation $A'X = b'$ for some public matrix A' and vector b'). Whereas we consider a variant of the linear test where the same set of linear constraints repeat over different sections (i.e., blocks) of the extended witness, which is represented as linear equations $Ay_i = b$ for all $i \in [B]$.

We first describe a simple algorithm for the new variant of linear test and later show how to further improve the verifier's time and space costs. At a high level, we apply Ligero's linear test on each block and then take a random linear combination of the outputs of the test for each block. At the prover's end, naively computing $r^{\mathbb{T}}A$ is expensive as A is a large matrix. By observing

Parameter	Description
Y_i	Blocks associated with the linear constraints
y_i	Flattened vector corresponding to block Y_i
B	#Blocks associated with the linear test
m'	#Rows in each block Y_i
$I[i]$	Index of the first row of X included in Y_i
s	Seed 1 for randomness
s'	Seed 2 for randomness
r	randomness vector 1
r'	randomness vector 2
A	Linear constraint matrix
b	linear constraint vector
m_a	#Linear constraint for each y_i
$r_{i,j}$	the value of the matrix at position (i,j) when parsing $r^T A$ into a matrix
$r_i(\cdot)$	i^{th} polynomial generated by encoding $r^T A$

Table 2. Description of the parameters.

that the matrix A is sparse (more precisely, A is succinct), we reduce the time and space required significantly by efficiently computing the positions of the non-zero elements of A.

Roughly, the protocol for linear test proceeds as follows. The verifier provides two random seeds $s, s' \in \mathbb{F}$, from which the prover and verifier can generate random vectors r and r'. We require two randomness vectors where one is used as random linear combiners for rows within a block, while the other is used as random linear combiners across blocks. The prover computes the polynomial encoding $q_i(\cdot)$ of $(r^{\mathbb{T}} A)y_i$ for each block Y_i and then computes the polynomial encoding $q(\cdot) = \sum_{i \in [B]} r'[i] \cdot q_i(\cdot)$ of all the blocks. Lastly, the verifier checks the consistency of $q(\cdot)$ with $\sum_{i \in [B]} r'[i] \cdot (r^{\mathbb{T}} b)$ and U on t randomly chosen columns. Refer to Fig. 1 for the formal description of the protocol.

Algorithm $\mathsf{DEval}(v, R)$**:** On input (v, R), this algorithm outputs an evaluation vector $e = \{p(\eta_j)\}_{\eta_j \in R}$ where the polynomial $p(\cdot)$ is defined such that $p(\zeta_i) = v[i]$ for all $i \in [\ell]$, ζ_i are the interpolation points, η_j are the evaluation points and R is the set of query points. The input vector v is provided to the algorithm in an input tape where the algorithm individually reads and processes each element in vector v. The algorithm repeats until all the elements are read from the input tape. The input v is a vector of size ℓ which needs to be interpolated. We denote the set of interpolation points to be ζ and set of evaluation points to be η. Note that the set R needs to be a subset of η i.e. $R \subseteq \eta$.

We set the evaluation points and interpolation points to be related to the roots of unity. In more detail, let w be a primitive $2n^{th}$ root of unity where $w^{2n} = 1$ but $w^m \neq 1$ for $0 < m < 2n$. We set the variable $f = n/\ell$, $\zeta = \{1, w^{2f}, w^{4f}, \ldots, w^{2f(\ell-1)}\}$ and $\eta = \{w, w^3, \ldots w^{2f(\ell-1)+1}\}$. Each individual in-

terpolation point and evaluation point can be represented as $\zeta_i = w^{2(i-1)f}$ and $\eta_j = w^{2j-1}$ respectively. The algorithm is as follows:

1. Check if $R \subseteq \eta$. Abort if the check fails. If the check succeeds, initialize $e_j = 0$ for all j such that $\eta_j \in R$.
2. Upon receiving an element $v[k]$ from the input tape, update the running partial sum $e_j = e_j + \frac{1}{\ell}\frac{w^{(2j-1)\ell}-1}{w^{2k-1-2fk+2f}-1}v[k]$ for all j such that $\eta_j \in R$.
3. After processing each element from the vector v, output all $e_j's$ for all j such that $\eta_j \in R$.

Proof. **Correctness:** Define a vector c such that each element of c represents the coefficient of the polynomial $p(\cdot)$ and v is the vector which is being interpolated. We represent the relation between c and v as $v = Xc$ where X is a public matrix and the i^{th} row of X can be represented as $X[i] = (\zeta_i^0, \zeta_i^1, \ldots \zeta_i^{\ell-1})$. Each element in X can be represented as $X[i,j] = \zeta_i^{j-1} = w^{2(i-1)(j-1)f}$. Another way to represent the same equation is $c = X^{-1}v$ where X^{-1} is the inverse of the matrix X i.e. $XX^{-1} = I$ and I is an identity matrix.

It now follows that $X^{-1}[i,j] = \frac{1}{\ell}w^{-2(i-1)(j-1)f}$. Lastly, to evaluate $p(\cdot)$ at η_j, we define a vector $w = (1, \eta_j, \ldots, \eta_j^{\ell-1})$ and represent $p(\eta_j)$ as $p(\eta_j) = w^T c = w^T X^{-1} v$. We calculate vector $w^T X^{-1}$ as:

$$\begin{aligned} w^T X^{-1}[k] &= \sum_{l=1}^{\ell} w[l] \cdot X^{-1}[l,k] \\ &= \sum_{l=1}^{\ell} \eta_j^{l-1} \cdot \frac{1}{\ell} w^{-2(l-1)(k-1)f} \\ &= \frac{1}{\ell} \sum_{l=1}^{\ell} w^{(2j-1)(l-1)} \cdot w^{-2(l-1)(k-1)f} \\ &= \frac{1}{\ell} \sum_{l=1}^{\ell} w^{(l-1)(2j-1-2kf+2f)} \\ &= \frac{1}{\ell} \frac{w^{(2j-1)\ell}-1}{w^{2j-1-2kf+2f}-1} \end{aligned}$$

$$\begin{aligned} p(\eta_j) &= w^T X^{-1} v \\ &= \sum_{k=1}^{\ell} w^T X^{-1}[k] v[k] \\ &= \frac{1}{\ell} \sum_{k=1}^{\ell} \frac{w^{(2j-1)\ell}-1}{w^{2j-1-2kf+2f}-1} v[k] \end{aligned}$$

The algorithm reads each element of the vector v sequentially from the input tape. Initialising $e_j = 0$ and after reading the element $v[k]$, the algorithm updates $e_j = e_j + \frac{w^{(2j-1)\ell}-1}{w^{2j-1-2kf+2f}-1} v[k]$. After processing the whole vector v, e_j will satisfy $e_j = p(\eta_j)$. Hence it shows the correctness of the algorithm.

Efficiency Analysis: For each element of v, the algorithm performs $O(1)$ operations per evaluation point. Thus, the overall computational cost is $O(t\ell)$ where t is the number of evaluations and ℓ is the size of v. The algorithm requires only $O(t)$ space to store only e_j's and the t evaluation points.

Lemma 8. *Protocol 1 is an IOP/IPCP for testing linear constraints with the following properties:*

- **Completeness:** *If $U \in L^m$ is an encoding of a $m \times \ell$ matrix X such that, for every $i \in [B]$, $Ay_i = b$ where y_i is the flattened vector corresponding to Y_i and block Y_i is a $m' \times \ell$ submatrix of X starting at the $I[i]^{th}$ row of X and the $\mathcal{P}$ is honest, then $\mathcal{V}$ accepts with probability 1.*
- **Soundness:** *Let e be a positive integer such that $e < d/2$ where d is minimal distance of Reed-Solomon code. Suppose that a badly formed matrix U^* is e-close to a codeword U that encodes a matrix X such that $\exists i \in [B]$, $Ay_i \neq b$ where y_i is the flattened vector corresponding to Y_i and block Y_i is a $m' \times \ell$ submatrix of X starting at the $I[i]^{th}$ row of X. Then for any malicious $\mathcal{P}^*$ strategy, $\mathcal{V}$ will reject except with probability $((e+2\ell)/n)^t + (m_a + B)/|\mathbb{F}|$.*
- **Complexity:**
  $\mathcal{P}$ has X on its input tape and has a work tape of size $O(m'\ell)$. In this model, $\mathcal{P}$ makes a single pass on the input tape. We denote by $m\ell$ the length of X, the number of blocks as B and y_i is a flattened vector of a block within X of size $m' \times \ell$. Given that $\mathcal{P}$ is provided with a one-way linear access to X, matrix A is a public succinct matrix of dimension $m_a \times m'\ell$ as defined in Definition 3 then the following complexities are obtained:
  - *Prover's Time* = $O(m'\ell(poly \log m_a + B \log \ell))$.
  - *Verifier's Time* = $O(m'\ell(poly \log m_a + \kappa) + Bm'\kappa)$.
  - *Prover's Space* = $O(m'\ell)$.
  - *Verifier's Space* = $O(\kappa m' + m_a)$.
  - *Communication Complexity* = $O(\ell)$.
  - *Query Complexity* = $O(\kappa)$.

Refer to the full version for the proof of completeness and soundness. Next, we discuss the time and space complexities for the prover and the verifier.

Prover's Time Complexity. Each column of of A can be computed in time $O(poly \log m_a)$ and computing each element of $r^{\mathbb{T}}A$ requires the same complexity. As length of vector $r^{\mathbb{T}}A$ is $m'\ell$, computing $r^{\mathbb{T}}A$ needs $O(m'\ell poly \log m_a)$ time. The polynomials $r_i(\cdot)$ generated in Step 2 can be constructed using inverse-FFT in $O(m'\ell \log \ell)$ time. The intermediate proof polynomial $q_i(\cdot)$ is composed by multiplying m' pairs of polynomial, each multiplication costs $O(\ell \log \ell)$ using FFT. The proof polynomial $q(\cdot)$ requires $O(Bm'\ell \log \ell)$ time

as it is generated by taking a random linear combination of all the intermediate proof polynomials. The prover's total time is $O(m'\ell(poly\log m_a + B\log \ell))$.

Before proceeding to the next analysis, we introduce a new lemma. This lemma allows the verifier to efficiently evaluate the sum of evaluation of a polynomial given the evaluation points are roots of unity.

Lemma 9. *Given a polynomial $p(\cdot)$ of degree t and H be the ℓ^{th} roots of unity, then $\sum_{a\in H} p(a) = \ell \sum_{i mod \ell=0} c_i$.*

We refer the readers to the full version for the proof.

Verifier's Time Complexity. The verifier upon receiving the proof polynomial $q(\cdot)$, needs to execute two checks. The first check is to check whether $\sum_{j\in[\ell]} q(\zeta_j) = \sum_{i\in[B]} r'[i]r^{\mathbb{T}}b$. To optimise the check, we leverage the structure of interpolation points ζ. We show that $\ell(c_0 + c_\ell) = \sum_{j\in[\ell]} q(\zeta_j)$ where c_0 and c_ℓ are the constant and ℓ^{th} coefficient of the polynomial $q(\cdot)$. To prove this, we directly use Lemma 9. This Lemma states that if a polynomial $p(\cdot)$ is evaluated at ℓ^{th} roots of unity, then the evaluation of the polynomial at all the roots of unity sums up to $\ell \sum_{i \mod \ell=0} c_i$ where c_i is the i^{th} coefficient of the polynomial $p(\cdot)$. Therefore, we can verify whether $\ell(c_0 + c_\ell) = \sum_{i\in[B]} r'[i]r^{\mathbb{T}}b$. This requires $O(m_a + B)$ time. To verify the second check, the verifier needs to generate t evaluations of polynomial $r_i(\cdot)$ defined in Step 2. To compute it, the verifier first evaluates $r^T A$ element by element. For each vector $v = (r_{i,1}, \ldots, r_{i,\ell})$ which is computed element by element and stored in the input tape of algorithm DEval. The algorithm DEval outputs t evaluation $r_i(\cdot)$ where $r_i(\cdot)$ can be generated using v. Evaluating v requires $O(\ell poly\log m_a)$ time and $t = O(\kappa)$ evaluations is generated in $O(\ell\kappa)$. As there are total m' polynomials, all evaluations are completed in $O(m'\ell(poly\log m_a + \kappa))$ time. In addition, the verifier needs $O(\ell\kappa)$ for evaluating $q(\cdot)$ at $t = \kappa$ evaluations and require $O(Bm'\kappa)$ operations to verifying the consistency between $q(\cdot)$ and U. Therefore, the total time to verify this check is $O((Bm' + \ell)\kappa)$. The verifier's total time is $O(m'\ell(poly\log m_a + \kappa) + Bm'\kappa + m_a)$.

Prover's Space Complexity. Firstly, the prover computes $r^{\mathbb{T}}A$ and the polynomials $r_i(\cdot)$ defined in Step 2 and stores them to be used for each $i \in [B]$ which requires $O(m'\ell)$ space in the work tape. To compute the polynomial $q(\cdot)$ in Step 2 in a space-efficient manner while making a single pass on input tape X, we implement Step 2 by maintaining a running partial aggregate. More precisely, the prover initialises the polynomial $\mathsf{agg}(\cdot) = 0$. Next, the prover processes X row by row. It keeps track of the blocks Y_i that contain the current row. There can be at most m' blocks Y_i that contain X as there can be at most one block that starts from any row of X. Let min_i and max_i denote the first and last block indices which contain the i^{th} row of X. The polynomial $\mathsf{agg}(\cdot)$ is

updated as follows:

$$\mathsf{agg}(\cdot) = \mathsf{agg}(\cdot) + p_i(\cdot) \sum_{l=\mathsf{min}_i}^{\mathsf{max}_i} r'[l] r_{i-I[l]}(\cdot) \tag{1}$$

After every row of X is processed, we set our proof polynomial as $q(\cdot) = \mathsf{agg}(\cdot)$. The space required to generate the polynomial $q(\cdot)$ is the space for storing $p_i(\cdot)$, $\mathsf{agg}(\cdot)$ and the product of two polynomials of degree $< \ell$. Since we can multiply polynomials via FFT, the space required is $O(\ell)$. Therefore, the overall space complexity of the prover in the interactive phase is dominated by storing the $r_i(\cdot)$ polynomials which is $O(m'\ell)$.

Verifier's Space Complexity. Upon receiving the proof polynomial $q(\cdot)$, the verifier performs the following three checks:

- The degree of q is at most $k + \ell - 1$. This can be done by simply counting the number of coefficients.
- The polynomial q satisfies $\sum_{j\in[\ell]} q(\zeta_j) = \sum_{i\in[B]} r'[i] r^{\mathbb{T}} b$. Following the optimization mentioned in the time complexity analysis, the verifier simply checks if $\ell \cdot (c_0 + c_\ell) = \sum_{i\in[B]} r'[i] r^T b$ where c_0 and c_l are the constant and ℓ^{th} coefficient of the polynomial $q(\cdot)$. This requires $O(m_a)$ space to store b.
- Finally, the verifier needs to compute $t = O(\kappa)$ evaluations on polynomials $r_i(\cdot)$ generated in Step 2. As we described in the beginning of this section, the verifier will rely on the DEval algorithm is executed to generate these evaluations. The verifier needs $O(m'\kappa)$ space where m' is the number of polynomials. For each query $j \in Q$, the verifier initialises each variable $\mathsf{agg}_j = 0$. When the verifier processes the i^{th} element (or i^{th} row of U) of each column of U queried, it computes public variables min_i and max_i just as the prover and each variable agg_j for all $j \in Q$ is updated as follows:

$$\mathsf{agg}_j = \mathsf{agg}_j + U_{i,j} \sum_{l=\mathsf{min}_i}^{\mathsf{max}_i} r'[l] r_{i-I[l]}(\eta_j) \tag{2}$$

  As all $r_i(\eta_j)$ are already stored by the verifier, the verifier requires to store only the agg_j variable for each j. Overall the verifier's space is $O(m'\kappa + m_a)$.

6 Acknowledgements

We thank the anonymous TCC'22 reviewers for their helpful comments. The first author conducted research during her internship at JP Morgan. The second and third authors are supported by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office and ISF grant No. 1316/18. The third and fourth authors are supported by DARPA under Contract No. HR001120C0087. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA.

References

1. Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: Lightweight sublinear arguments without a trusted setup. In: CCS. pp. 2087–2104 (2017)
2. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Fast reed-solomon interactive oracle proofs of proximity. In: ICALP. pp. 14:1–14:17 (2018)
3. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable zero knowledge with no trusted setup. In: CRYPTO. pp. 701–732 (2019)
4. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E.: Fast reductions from rams to delegatable succinct constraint satisfaction problems: extended abstract. In: ITCS. pp. 401–414 (2013)
5. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: Transparent succinct arguments for R1CS. In: EUROCRYPT. pp. 103–128 (2019)
6. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: TCC. pp. 31–60 (2016)
7. Bhadauria, R., Fang, Z., Hazay, C., Venkitasubramaniam, M., Xie, T., Zhang, Y.: Ligero++: A new optimized sublinear IOP. In: CCS. pp. 2025–2038 (2020)
8. Bitansky, N., Chiesa, A.: Succinct arguments from multi-prover interactive proofs and their efficiency benefits. In: CRYPTO. pp. 255–272 (2012)
9. Block, A.R., Holmgren, J., Rosen, A., Rothblum, R.D., Soni, P.: Public-coin zero-knowledge arguments with (almost) minimal time and space overheads. In: TCC. pp. 168–197 (2020)
10. Block, A.R., Holmgren, J., Rosen, A., Rothblum, R.D., Soni, P.: Time- and space-efficient arguments from groups of unknown order. In: CRYPTO. pp. 123–152 (2021)
11. Blumberg, A.J., Thaler, J., Vu, V., Walfish, M.: Verifiable computation using multiple provers. IACR Cryptol. ePrint Arch. p. 846 (2014)
12. Bootle, J., Chiesa, A., Liu, S.: Zero-knowledge succinct arguments with a linear-time prover. IACR Cryptol. ePrint Arch. p. 1527 (2020)
13. Bowe, S., Grigg, J., Hopwood, D.: Halo: Recursive proof composition without a trusted setup. IACR Cryptol. ePrint Arch. p. 1021 (2019)
14. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA. pp. 315–334. IEEE Computer Society (2018)
15. Bünz, B., Chiesa, A., Mishra, P., Spooner, N.: Recursive proof composition from accumulation schemes. In: TCC. pp. 1–18 (2020)
16. Chiesa, A., Ojha, D., Spooner, N.: Fractal: Post-quantum and transparent recursive proofs from holography. In: EUROCRYPT. pp. 769–793 (2020)
17. Coron, J.S., Dodis, Y., Malinaud, C., Puniya, P.: Merkle-damgård revisited: How to construct a hash function. In: CRYPTO. pp. 430–448 (2005)
18. Ephraim, N., Freitag, C., Komargodski, I., Pass, R.: Sparks: Succinct parallelizable arguments of knowledge. In: EUROCRYPT. pp. 707–737 (2020)
19. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: CRYPTO. pp. 186–194 (1986)
20. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems (extended abstract). In: STOC. pp. 291–304 (1985)
21. Hazay, C., Ishai, Y., Marcedone, A., Venkitasubramaniam, M.: Leviosa: Lightweight secure arithmetic computation. In: CCS

22. Holmgren, J., Rothblum, R.: Delegating computations with (almost) minimal time and space overhead. In: Thorup, M. (ed.) 59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018. pp. 124–135. IEEE Computer Society (2018). https://doi.org/10.1109/FOCS.2018.00021, https://doi.org/10.1109/FOCS.2018.00021
23. Ishai, Y.: Zero-knowledge proofs from information-theoretic proof systems (2020), https://zkproof.org/2020/08/12/information-theoretic-proof-systems
24. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: STOC. pp. 21–30 (2007)
25. Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: Proceedings of the 24th Annual ACM Symposium on Theory of Computing. pp. 723–732 (1992)
26. Kothapalli, A., Masserova, E., Parno, B.: A direct construction for asymptotically optimal zksnarks. IACR Cryptol. ePrint Arch. p. 1318 (2020)
27. Lee, J., Setty, S.T.V., Thaler, J., Wahby, R.S.: Linear-time zero-knowledge snarks for R1CS. IACR Cryptol. ePrint Arch. p. 30 (2021)
28. Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. In: EUROCRYPT. pp. 52–78 (2007)
29. Mohassel, P., Rosulek, M., Scafuro, A.: Sublinear zero-knowledge arguments for RAM programs. In: EUROCRYPT. pp. 501–531 (2017)
30. Reed, I.S., Solomon, G.: Polynomial codes over certain finite fields. Journal of the Society for Industrial and Applied Mathematics **8**(2), 300–304 (1960)
31. Reingold, O., Rothblum, G.N., Rothblum, R.D.: Constant-round interactive proofs for delegating computation. In: STOC. pp. 49–62 (2016)
32. Savage, J., Swamy, S.: Space-time trade-offs on the fft algorithm. IEEE Transactions on Information Theory **24**(5), 563–568 (1978). https://doi.org/10.1109/TIT.1978.1055938
33. Setty, S.T.V.: Spartan: Efficient and general-purpose zksnarks without trusted setup. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO. pp. 704–737 (2020)
34. Setty, S.T.V., Lee, J.: Quarks: Quadruple-efficient transparent zksnarks. IACR Cryptol. ePrint Arch. p. 1275 (2020)
35. Thaler, J.: Proofs, arguments, and zero-knowledge (2021), https://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.html
36. Xie, T., Zhang, J., Zhang, Y., Papamanthou, C., Song, D.: Libra: Succinct zero-knowledge proofs with optimal prover computation. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO. pp. 733–764 (2019)
37. Zhang, J., Xie, T., Zhang, Y., Song, D.: Transparent polynomial delegation and its applications to zero knowledge proof. In: S&P. pp. 859–876. IEEE (2020)