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.

Table of Contents

1 Introduction

Zero-knowledge proofs, introduced by Goldwasser, Micali and Rackoff [23] 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 [26, 43] 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) [16, 44, 45, 40, 41, 31, 14, 32]. 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. polylogarithmic overhead in space and time) have been constructed by Bitansky and Chiesa [10] and by Holmgren and Rothblum [25] in the non-interactive setting. The work of Ephraim et al. [20] 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 [15, 18, 17]. 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 [33, 1, 24].

More recently, two works by Block et al. [11, 12], 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 [6, 9] 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 [6, 9] 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 [15, 18, 17].

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 [7, 37], 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 [21] 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 $T \cdot S \in \Omega(n^2)$ [38]. 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 [27]) 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 [36].
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 [30, 7].

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 [19]. 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 [5, 13] 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 [5, 13], 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 [5, 13] 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 (Definition 3), 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 [5, 13]): (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. [7] (which in turn is based on [30]) 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 [3] which requires poly-logarithmic 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 Towards a 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 $S > ((\delta n)/r - O(\log m)) \cdot \log|\mathbb{F}|$, where n is the message length, m is the codeword length, δ is the distance and r is the number of passes. A similar lower bound on the space was shown by [2] for the restricted case where the encoding algorithm made only a single pass over the message i.e., $r = 1$. We prove our lower bound 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 $O(\delta n)$ 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 $|\mathbb{F}|^{O(\delta n)}$ such messages). We show that there will be a subset of messages, say D, of size at least $|\mathbb{F}|^{O(\delta n)-rS/\log|\mathbb{F}|-O(r\log m)}$ 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., $|\mathbb{F}|^{O(\delta n)-rS/\log|\mathbb{F}|-O(r\log m)} \leq 1$ which implies that the space $S > ((\delta n)/r - O(\log m)) \cdot \log|\mathbb{F}|$.

1.3 A Comparison with Related Work

Related to the design of sub-linear zero-knowledge arguments, the work of Mohassel, Rosulek and Scafuro [35] 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 [11, 12] also designed black-box constructions of ZK-SNARKs with poly-logarithmic 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 [11, 12] match our complexity. This is the case for the verifier's space complexity as well. The verifier's running time in [11] is $\widetilde{O}(T)$ and $\widetilde{O}(n)$ in [12] 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
[11]	$\widetilde{O}(T)$	$\widetilde{O}(S)$	$\widetilde{O}(T)$	$\widetilde{O}(1)$
[12]	$\widetilde{O}(T)$	$\widetilde{O}(S)$	$\widetilde{O}(n)$	$\widetilde{O}(1)$
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 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 see Section 5 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 1. *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:*

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.

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 (c.f. Figure 3)*

.

Efficiency. *Furthermore, the matrices A, A' and A'' are succinct according to Definition 10 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 only focus on the linear test in this section. A formal description of the linear test and its corresponding lemma are given below. For the full description of all the elements of our protocol, we refer the reader to the section 5.

Lemma 2. *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 10 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. In section 5.2.1, we present our complexity-preserving reduction from $\mathsf{BH}_{\mathsf{RAM}}$ to extended witness X that satisfies the system of constraints defined in lemma 10.
2. In sections 5.2.2, 5.2.3 and 5.2.4, we present protocols for testing interleaved linear codes, linear constraints and quadratic constraints given 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 3. *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. Then the Protocol 4 is 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 section 5.3, we show how to modify our IPCP to obtain zero-knowledge and then in section 5.5 how to improve the communication to $\widetilde{O}(T/S)$.

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. In more detail, for a space-efficient constant-distance code that maps n field elements (i.e., message) to m field elements (i.e., codeword) and has a minimum relative distance of δ, we show that an encoding algorithm that makes r passes over the message requires space $S > O(((\delta n)/r - \log m) \cdot \log |\mathbb{F}|)$. Previously, [2] showed a similar lower bound for the restricted case where the encoding algorithm made only a single pass over the message (as opposed to r passes in our case).

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 [30, 6, 4, 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 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 $O(\delta n/2)$ 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 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. 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 3. *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 4. *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 3 and 4 to prove theorem 4 via contradiction. As per lemma 3, 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 4 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 5. *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 5. *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 6. *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 5 and 6, 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 7. *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 4 to prove this claim.

We conclude the proof of Theorem 5 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 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.

4.1 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 g_{id} and will have two input wires and one output wire. Each wire will also have a wire id w_{id} and in case a wire value is an output wire of gate g_{id}, then the wire id $\mathsf{w}_{\mathsf{id}} = \mathsf{g}_{\mathsf{id}}$. Each element in the transcript W is of the form $(\mathsf{g}_{\mathsf{id}}, \mathsf{type}, \gamma)$ where 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 g_{id}.

A circuit descriptor, denote by ϕ, is an algorithm that, given as input a gate number g_{id} of C, outputs the type of the gate type, the input gates to g_{id} and the output gates to g_{id}. We refer to a circuit that has a *short* descriptor (say, of size $\mathsf{polylog}(|C|)$ where C is the size of the circuit) as a *succinct* circuit.

4.2 Secret Sharing Schemes

A secret sharing scheme is a fundamental building block used in multi-party computation. Roughly, a (t, r, n)-secret sharing scheme ($t < r \leq n$) takes a secret s as input and outputs n shares with the following properties: the secret s can be efficiently reconstructed from any subset of r shares (*correctness*), while ensuring that a subset of t shares does not reveal anything about the secret s (*privacy*). A secret sharing scheme consists of the two algorithms $(\mathsf{Sh}, \mathsf{Rec})$, which we instantiate with Shamir's secret sharing scheme [42] as follows.

- **Sharing:** Given any input $s \in \mathbb{F}$, the sharing algorithm Sh outputs n shares $\mathbf{c}$. The syntax for this is $\mathbf{c} \leftarrow \mathsf{Sh}(m)$. Pick a random polynomial p of degree t such that $p(0) = s$ and output $c := (p(1), \ldots, p(n))$.
- **Reconstruction:** The algorithm Rec takes as an input the shares $\mathbf{c} = \{c'_i\}_{i \in S} \in \mathbb{F}^n$ and a set S to reconstruct the secret s where none of c'_i are $\perp$ and $|S| > t$. This is denoted by $s = \mathsf{Rec}(\mathbf{c})$. The reconstruction algorithm reconstructs the polynomial $p'(\cdot)$ using Lagrange interpolation as follows and outputs $p'(0)$.

$$p'(x) = \sum_{i \in S} c'_i \prod_{j \in S \setminus \{i\}} \frac{x - j}{i - j}$$

Packed secret-sharing. Packed secret-sharing was introduced by Franklin and Yung [22] in order to reduce the communication complexity of secure multi-party protocols, and is an extension of standard secret-sharing. In [22] the authors considered Shamir's secret-sharing with the difference that the number of secrets $s_1, \ldots, s_\ell$ is now ℓ instead of a single secret, evaluated by a polynomial $p(\cdot)$ on ℓ distinct points. To ensure privacy in case of t corrupted parties, the random polynomial must have degree at least $t + \ell$. We use packed secret-sharing in our underlying protocol to save on communication complexity. We denote a packed secret sharing scheme for ℓ secrets by the pair of algorithms $(\mathsf{Sh}_\ell, \mathsf{Rec}_\ell)$ and extend the security definitions from the standard secret sharing scheme accordingly.

4.3 Reed-Solomon Encoding

For a linear code $\mathsf{C} \subseteq \Sigma^n$ and vector $v \in \Sigma^n$, we use $d(\mathsf{C}, v)$ to denote the minimal distance of v from C. Formally $d(\mathsf{C}, v) = \min_{c \in \mathsf{C}} h(c, v)$, where $h(c, v)$ is the hamming distance between c and v.

Definition 1 (Reed-Solomon code). *For positive integers n, k, field $\mathbb{F}$ and vector $\eta = (\eta_1, \ldots, \eta_n) \in \mathbb{F}^n$ of distinct field elements, the Reed-Solomon (RS) code $RS_{\mathbb{F},n,k,\eta}$ is the $[n, k, n-k+1]$ linear code over $\mathbb{F}$ that consists of all n-tuples $(p(\eta_1), \ldots, p(\eta_n))$ where p is a polynomial of degree $< k$ over $\mathbb{F}$.*

Definition 2 (Encoded message). *Let $L = RS_{\mathbb{F},n,k,\eta}$ be an RS code and $\zeta = (\zeta_1, \ldots, \zeta_k)$ be a sequence of distinct elements in $\mathbb{F}$. For a codeword $u \in L$, we define the message $Dec_\zeta(u)$ to be $(p_u(\zeta_1), \ldots, p_u(\zeta_k))$, where p_u is the polynomial (of degree $< k$) corresponding to codeword u. For $U \in L^m$ with rows $u_1, \ldots, u_m \in L$, we let $Dec_\zeta(U)$ be the length-mk vector $x = (x_{11}, \ldots, x_{1k}, \ldots, x_{m1}, \ldots, x_{mk})$ such that $(x_{i1}, \ldots, x_{ik}) = Dec_\zeta(u^i)$ for $i \in [m]$. Finally we say that U encodes x if $x = Dec_\zeta(U)$, we use $Dec(U)$ when ζ is clear from the context.*

In our protocol, we set the interpolation point $\zeta_i = \omega^{2(i-1)f}$ and evaluation point $\eta_i = \omega^{2i-1}$ where ω be a primitive $2n^{th}$ root of unity i.e. $\omega^{2n} = 1$ but $\omega^m \neq 1$ for $0 < m < 2n$. We can evaluate $(p(\eta_1), \ldots, p(\eta_n))$ using the fast Fourier transform (FFT), which takes $O(n \log n)$ field operations. We use $\mathsf{RS}(a)$ to denote the RS encoding of message a.

Further, Reed-Solomon encoding satisfies a strong multiplicative property which states that the "product" of two Reed-Solomon codewords is a Reed-Solomon code.

Definition 3 (Strong Multiplicative Property). *Let $L = RS_{\mathbb{F},n,k,\eta}$ be an RS code and $\zeta = (\zeta_1, \ldots, \zeta_k)$ be a sequence of distinct elements in $\mathbb{F}$. For any two codeword $u_1, u_2 \in L$, then $u_1 \odot u_2 \in RS_{\mathbb{F},n,2k-1,\eta}$ where is $\odot$ denote pointwise product.*

4.4 Schwartz-Zippel Lemma

Lemma 8. *[39] Let $\mathbb{F}$ be the field and f be a polynomial of degree d.*

$$\mathsf{Pr}_{x \leftarrow \mathbb{F}}[f(x) = 0] \leq d/|\mathbb{F}|$$

This lemma states that if x is chosen uniformly at random from the field $\mathbb{F}$, then the probability that $f(x) = 0$ is at most $d/|\mathbb{F}|$.

4.5 Collision-Resistant Hashing and Merkle Trees

Let $\{\mathcal{H}_\kappa\}_{\kappa \in \mathbb{N}} = \{H : \{0,1\}^{p(\kappa)} \to \{0,1\}^{p'(\kappa)}\}_\kappa$ be a family of hash functions, where $p(\cdot)$ and $p'(\cdot)$ are polynomials so that $p'(\kappa) \leq p(\kappa)$ for sufficiently large $\kappa \in \mathbb{N}$. When we need the hash computations to be space efficient, we make use of the Merkle-Damgard paradigm [19]. In this paradigm, a hash of an arbitrary length message can be computed on fixed length blocks (say of size S) iteratively, which requires space $O(S)$. For a message $m = (m_1, \ldots, m_n)$, the the Merkle-Damgard hash function is updated iteratively for each m_i which we denote by $h_i \leftarrow \mathsf{MD.Update}(h_{i-1}, m_i)$[3].

For a hash function $H \leftarrow \mathcal{H}_\kappa$, a Merkle hash tree [34] is a data structure that allows to commit to $\ell = 2^d$ messages by a single hash value h such that revealing any message requires only to reveal $O(d)$ hash values. A Merkle hash tree is represented by a binary tree of depth d where the ℓ messages $m_1, \ldots, m_\ell$ are assigned to the leaves of the tree; the values assigned to the internal nodes are computed using the underlying hash function H that is applied on the values assigned to the children, whereas the value h that commits to $m_1, \ldots, m_\ell$ is assigned to the root of the tree. To open the commitment to a message m_i, one reveals m_i together with all the values assigned to nodes on the path from the root to m_i, and the values assigned to the siblings of these nodes. We denote the algorithm of committing to ℓ messages $m_1, \ldots, m_\ell$ by $h := \mathsf{MT.Commit}(m_1, \ldots, m_\ell)$ and the opening of m_i by $(m_i, \pi_i) := \mathsf{MT.Open}(h, i)$. Verifying the opening of m_i is carried out by essentially recomputing the entire path bottom-up and comparing the final outcome (i.e., the root) to the value given at the commitment phase.

The binding property of a Merkle hash tree is due to collision-resistance. Intuitively, this says that it is infeasible to efficiently find a pair (x, x') so that $H(x) = H(x')$, where $H \leftarrow \mathcal{H}_\kappa$ for sufficiently large κ. In fact, one can show that collision-resistance of $\{\mathcal{H}_\kappa\}_{\kappa \in \mathbb{N}}$ carries over to the Merkle hashing. Formally, we say that a family of hash functions $\{\mathcal{H}_\kappa\}_\kappa$ is collision-resistant if for any PPT adversary $\mathcal{A}$ the following experiment outputs 1 with probability $negl(\kappa)$: (i) A hash function H is sampled from $\mathcal{H}_\kappa$; (ii) The adversary $\mathcal{A}$ is given H and outputs x, x'; (iii) The experiment outputs 1 if and only if $x \neq x'$ and $H(x) = H(x')$.

4.6 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 4. *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,*

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

[3] Here, h_0 is defined as per the specific instantiation of the Merkle-Damgard construction.

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.

4.7 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 [5].

Definition 5. *(The RAM Model [5]) 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.*
- $\mathcal{T}$ *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. During the execution of M, we refer to the "local" state of M at some time step $\tau \in [T(n)]$ as the configuration which consists of the program counter and the values of all the registers at timestep τ.

Definition 6. *Let $M = (w, k, \mathbb{A}; \mathcal{C}, \mathcal{T}, \mathbb{W})$ be a random-access machine. A configuration of M is a tuple $\mathcal{S} = (pc, r_0, \ldots, r_{k-1})$ where I_{pc} is the next instruction to be executed in code $\mathcal{C}$ and $r_0, \ldots, r_{k-1}$ are the current w-bit values of the k registers.*

The execution of the RAM program involves executing a sequence of instructions pointed to by the program counter pc. Specifically, the instruction I_{pc} is executed at every timestep of the RAM program. The instruction may modify the registers, the work tape or the program counter as per the RAM program's code. If the program counter is modified by the current instruction,[4], then the new instruction pointed by pc is executed next; otherwise the program counter proceeds to the next instruction i.e., $pc := pc + 1$. An execution of the RAM program is represented by a sequence of (time-ordered) configurations $\overrightarrow{\mathcal{S}} = (\mathcal{S}_0, \ldots, S_{T-1})$. If the final configuration $\mathcal{S}_{T-1}$ outputs the instruction corresponding to it is $(\mathsf{out}, \mathsf{accept})$, then $\mathcal{S}_{T-1}$ is an accepting configuration and the sequence of configurations $\overrightarrow{\mathcal{S}}$ are accepting. Similarly, if the final configuration $\mathcal{S}_{T-1}$ outputs the instruction corresponding to it is $(\mathsf{out}, \mathsf{reject})$, then $\mathcal{S}_{T-1}$ is a rejecting configuration and and the sequence of configurations $\overrightarrow{\mathcal{S}}$ are rejecting.

The RAM program M has arbitrary access to a work tape[5]. 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

[4] For instance, jump or goto instructions modify the program counter i.e. set pc to point to an arbitrary instruction.
[5] Generally, the RAM program has access to memory which we model as a work tape.

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. For a sequence of configurations $\overrightarrow{\mathcal{S}}$ to be valid, we require a value read from the work tape to be the same as the value previously written into the same location. In other words, we need the configurations with a load instruction to be consistent with previous configuration with a store instruction modifying the same memory cell, which we refer to as memory consistency.

4.8 RAM to Circuit Reduction

Circuits are a model of computation that can efficiently capture highly structured computation such as Fourier transforms, arithmetic operation on inputs, etc. However, circuits are not convenient to represent certain problems. Constraint satisfaction problems are one such examples with a strong combinatorial or algebraic flavor such as the number of Hamiltonian cycles, the graph coloring problem and the existence of a low-degree polynomial with certain roots. Also, the correct execution of programs in written in high-level languages such as C++, Java can easily be reduced to the correct execution of RAM programs using modern compilers, rather than circuits. Since a wide range of problems can be conveniently expressed as time and space bounded RAM programs, we express our problems as constraint satisfaction problems over RAM programs, which can be modeled as a Bounded Halting Problem on a RAM. Following the formalization of [5], a Bounded Halting Problem on a RAM program M, denoted by $\mathsf{BH}_{\mathsf{RAM}}$, is the language of all triples (x, T, S) such that there is a witness w for which $M(x, w)$ accepts within T time steps using at most S memory cells (formally stated in [5]).

Definition 7. *[5] Let M be a random-access machine. The language $\mathsf{BH}_{\mathsf{RAM}}(M)$ consists of instances $(x, T(n), S(n))$, where x is a binary string of length n, with $|x| \leq T(n)$, such that there exists a binary string w of length at most $T(n)$ for which $M(x, w)$ accepts within $T(n)$ steps and accesses at most $S(n)$ distinct addresses. Furthermore, we denote by $\mathsf{BH}_{\mathsf{RAM}}$ the language of all quadruples $(M, x, T(n), S(n))$ such that $(x, T(n), S(n)) \in \mathsf{BH}_{\mathsf{RAM}}(M)$.*

Next, we describe the circuit satisfiability (CSAT) which is the language of all satisfiable circuits. The succinct variant of circuit *sCSAT* consists of (potentially short) circuit descriptors[6] instead of circuits, which is defined as follows.

Definition 8. *[5] (sCSAT) For a family of circuit descriptors $\Phi = \{\phi_{T,S}\}_{T,S \in \mathbb{N}}$, $sCSAT(\Phi)$ is the language of (x, T, S) such that there exists a witness w for which $C(x, w)$ accepts, where C is the circuit described by $\phi_{T,S}$. The language sCSAT is the set of all (ϕ, x, T, S) such that $(x, T, S) \in sCSAT(\phi)$.*

We rely on the RAM to circuit reduction of [5] to transform a time T, space S RAM program M that accepts input x into a (non-deterministic) arithmetic circuit C over a field $\mathbb{F}$ where C has a succinct descriptor ϕ of size $\mathsf{polylog}(T)$. Namely, for an instance (x, T, S), we have that $(x, T, S) \in \mathsf{BH}_{\mathsf{RAM}}(M)$ if and only if $(x, T, S) \in sCSAT(\phi)$.

Zero Knowledge for RAM. Given a succinct zero-knowledge argument system (P, V) for circuit satisfiability (CSAT) and the reduction from $\mathsf{BH}_{\mathsf{RAM}}$ to Circuit satisfiability (given in [5]), we construct a succinct zero-knowledge argument system (P', V') for $\mathsf{BH}_{\mathsf{RAM}}$. The prover P' reduces the proof for statements of the form "Is (x, T, S) satisfiable?" to statements of the form "Is the circuit C satisfiable?". Then the prover P' invokes the prover P for CSAT. The reduction from $\mathsf{BH}_{\mathsf{RAM}}$ to CSAT preserves succinctness i.e., the reduction from $\mathsf{BH}_{\mathsf{RAM}}$ to CSAT outputs a succinct representation of the NP statement for CSAT.

We model the prover P and the verifier V in our zero-knowledge argument system as RAM programs. Both P and V have a read-only unidirectional input tape with the instance (x, T, S), a unidirectional output tape and a work tape. Additionally, P has read-only unidirectional access to a witness tape with the witness w and V has read-only unidirectional access to a proof tape with the proof written by P.

[6] Recall that circuit descriptors are algorithms, given an input g_{id}, output all the information related to the gate such as the type, inputs and outputs of g_{id}.

4.9 Routing Networks

We model the routing network as a graph with a distinguished set of source and sink nodes each of size N. The goal of these routing networks is to find node-disjoint paths from a source node i to the sink node $\pi(i)$ for each $i \in [N]$ where π is permutation over $[N]$. If such node-disjoint paths exist for every permutation from sources to sinks, then such a routing network is said to have the property of *rearrangability*.

Extended De Bruijn networks are a class of routing networks, which can be modeled as a directed graph with $O(\kappa)$ layers each containing 2^κ nodes where each vertex is connected to two nodes in the following layer. The sources and sinks correspond to the set of nodes in the first and last layer respectively. We use the same notation as [8, 5] for De Bruijn graphs, which is formally stated in Definition 9.

Definition 9. *(Extended De-Bruijn Graphs [5]) Let κ and L be two positive integers. The (κ, L) extended De Brujn graph, denoted $\mathsf{DB}(\kappa, L)$, is a directed 2-regular graph with L layers numbered $0, \ldots, L-1$ each containing 2^κ vertices identified by κ-bit strings. A vertex in layer $i \in \{0, \ldots, L-1\}$ with identifier $w \in \{0,1\}^\kappa$ has two neighbors in layer $(i+1) \bmod L$ with identifier $sr(w)$ and $sr(w) \oplus e_1$ where sr denotes the cyclic "shift right" bit operation and e_1 denotes w-bit string such that only the most significant (i.e., rightmost) bit is 1 and all other bits are 0. In other words, the edge set is induced by the following two neighbor functions:*

$$\Gamma_1((i,w)) = ((i+1 \bmod L), sr(w))$$

$$\Gamma_2((i,w)) = ((i+1 \bmod L), sr(w) \oplus e_1)$$

It has been shown in [5] that the extended De-Bruijn graph has the rearrangability if it has "sufficiently" many layers. In more detail, they can route any given permutation over $\{0,1\}^\kappa$ from the source to the sink nodes in time and space $O(\kappa \cdot 2^\kappa)$, which is stated formally in lemma 9

Lemma 9. *[5] Let κ be a positive integer and $\pi : \{0,1\}^\kappa \leftarrow \{0,1\}^\kappa$ be a permutation. There exists a set S_π of 2^κ node-disjoint paths such that each node $(0,w)$ in $\mathsf{DB}(\kappa; 4\kappa-1)$ is connected to $(0, \pi(w))$. Moreover, S_π can be found in time and space $O(\kappa \cdot 2^\kappa)$ or parallel time $O(\kappa^2)$.*

Suppose a routing network outputs a path p_i from the source node i to the sink node $\pi(i)$ for each $i \in [N]$ where π is a permutation. Let each of the source nodes have information of size $\{0,1\}^\kappa$ with it, which is referred to as a packet. The packets associated with a source node i are "forwarded" from i to $\pi(i)$ along the path p_i. Since the paths are node-disjoint, no two packets will cross paths through a same node.

4.10 Succinct Matrix

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

Definition 10 (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 $\mathsf{polylog}(n_1)$.*

5 Sublinear Zero-Knowledge Arguments for RAM Programs

In this section, we present our space-efficient zero-knowledge argument system for RAM programs based on collision-resistant hash-functions. Our construction is a variant of the Ligero argument system [1] and we begin with a brief overview of the construction.

5.1 Overview of Ligero ZK Argument [1]

Given an arithmetic circuit C, the Ligero system[1] allows proving satisfiability of C. Recall that in the Ligero proof system, the satisfiability is expressed as a system of linear constraints (A, b) and quadratic constraints (defined via a set of tuples containing row indices (i_l, i_r, i_o)) over a matrix X that contains the wire values of an evaluation of the circuit C on a satisfying input arranged in a specific way.

In more detail, the proof system proceeds as follows:

1. **Merkle tree:** In the first step, the prover commits an "extended witness" to the verifier. In more detail, it 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. Finally, the prover generates a Merkle hash tree where the leaves contain the n columns of U and sends the Merkle root to the verifier.
2. **Testing IRS encoding:** In the next step, the prover provides evidence attesting that the matrix U is a valid encoding of some matrix X. Namely, in this step, the verifier sends a random vector $r \in \mathbb{F}^m$ as a challenge to the prover and the prover responds with the vector $r^{\mathbb{T}}U$. The verifiers accepts if the vector is a valid RS codeword.
3. **Testing linear Constraints:** In this step, the prover convinces the verifier of all the linear constraints on the extended witness. Namely, these include the circuit wiring topological constraints and addition gates. Let x be the $m \times \ell$-length flattened vector corresponding to X obtained by concatenating the rows of X. Then the linear constraints can be expressed as $Ax = 0$ where A is a public matrix. The verifier provides a vector r' of length $m \times \ell$ and tries to verify that $r'^{\mathbb{T}}(Ax) = b$. Towards this, the prover rearranges the vector $r'^{\mathbb{T}}A$ as an $m \times \ell$ matrix and computes an Interleaved Reed Solomon encoding R. Instead of sending $r'^{\mathbb{T}}(Ax)$, the prover sends the vector $q = (\mathbf{1}_m)^{\mathbb{T}}(R \odot U)$ where $\mathbf{1}_m$ is the all ones length m vector and $\odot$ denotes pointwise product. By the multiplicative property of Reed-Solomon codes (c.f. Definition 3)), it follows that checking $r'^{\mathbb{T}}(Ax) = r\mathbb{T}'b$ is equivalent to decoding q to a vector of values and checking if the sum of those values is $r'^{\mathbb{T}}b$.
4. **Testing Quadratic Constraints:** Quadratic constraints encode all the multiplication gates in the circuit. The special arrangement of the wire values in X require the inputs/outputs of multiplication gates be batched ℓ at a time and have the left wire values, right wire values and output wire values in separate rows of X such that the j^{th} columns of each of these rows correspond to the wire values of j^{th} multiplication gate of that batch. Given a tuple of row indices (i_l, i_r, i_o) corresponding to a batch of multiplication gates, we have $X_{i_l,\cdot} \odot X_{i_r,\cdot} - X_{i_o,\cdot}$ is the all o's vector if the prover is honest. To check this, in this step, the verifier provides a random vector r''. The prover responds with
$$\sum_j r''_j \cdot (U_{i^j_l,\cdot} \odot U_{i^j_r,\cdot} - U_{i^j_o,\cdot})$$
and the verifier decodes and checks if the resulting vector encodes the all 0 string. This step relies again on the strong multiplicative property of Reed Solomon codes.
5. **Column check:** In this step, the verifier ensures that the responses to the previous tests are consistent with the actual code committed to at the beginning via the root of the Merkle tree. More precisely, the verifier samples and sends a random subset $I \subseteq [n]$ of κ columns. The prover responds with the corresponding columns in the U matrix along with Merkle decommitments. Next, the verifier recomputes the result of each of the tests above for those columns and checks if the response agrees with the recomputed values on those columns. This step utilizes the fact that every test can be expressed as row operations on the U matrix.

5.2 Our IPCP Construction

In this section, we provide a description of our IPCP system. It follows the same blueprint of the Ligero system. On a high-level we will provide a space-efficient variant of each phase of the Ligero blueprint.

1. In Section 5.2.1, we describe our RAM to circuit reduction. We rely on the transformation of [5, 13] that transforms a RAM program to a circuit C with a "succinct" representation.

2. In Section 5.2.2, we describe how to test 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 and maintaining a running partial aggregate of $\sum_j r_j U_{j,\cdot}$.
3. In Section 5.2.3, we describe 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.
4. In Section 5.2.4, we describe how the quadratic test can be performed in a space-efficient manner. This will rely on ideas from the previous two steps.
5. In Section 5.2.5, we integrate all of the previous subsections from Section 5.2.1-5.2.4 to construct a space-efficient IPCP.

5.2.1 RAM to Circuit Reduction We rely on the space-efficient RAM-to-Circuit reduction of [5, 13], which transforms a RAM program M with running time T and space S into a circuit C of size $\tilde{O}(T)$ where the transformation runs in time $\tilde{O}(T)$ and space $\tilde{O}(S)$. The resulting circuit C of size $T \cdot \mathsf{polylog}(T)$ can be generated in time $\mathsf{polylog}(T)$. Given a (input, witness) pair (x, w), we show that the wire values in $C(x, w)$ can be arranged in a specific manner that would later be useful in our zero-knowledge argument system (as it allows us to efficiently check if the constraints imposed by the circuit are satisfied). We describe the extended witness[7] X for the circuit C and the corresponding constraints in Lemma 10.

Parameter	Description
X	Extended Witness
U	Encoded Extended Witness
$p_i(\cdot)$	i^{th} polynomial generated by encoding X
m	#Rows in the extended witness (& oracle)
ℓ	#Columns in the extended witness
n	#Columns in the oracle
t	#queries on U
B	#blocks in X
m'	#Rows in a block
X_i^{inp}	Sub-block of X_i (associated with the non-deterministic inputs)
X_i^{left}	Sub-block of X_i (associated with left wire values)
X_i^{right}	Sub-block of X_i (associated with right wire values)
X_i^{out}	Sub-block of X_i (associated with output wire values)
m_{inp}	#Rows in X_i
m_{mult}	#Rows in each of X_i^{left}, X_i^{right} and X_i^{out}

Table 2. Description of the parameters (Part 1)

Lemma 10. *[Lemma 1 restated] 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).

[7] Recall that the extended witness is an arrangement of the wire values of a circuit in a specific order.

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 (c.f. Figure 3).

Efficiency. *Furthermore, the matrices A, A' and A'' are succinct according to Definition 10 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).*

Notation. Recall a configuration consists of the contents of all the registers and the program counter at a particular timestep (see Definition 6). Given inputs (x, w), we associate (timestep, configuration) pair for each of the T timesteps during the execution of M. The set of all such T (timestep, configuration) pairs is referred to as the transcript of M, which we denote as follows $\overrightarrow{S} = \{(1, S_1), \ldots, (T, S_T)\}$. The transcript can be ordered by timesteps or memory access addresses (with ties broken by timesteps), which we denote by $\overrightarrow{S}_{\mathsf{time}}$ and $\overrightarrow{S}_{\mathsf{mem}}$ respectively. Lastly, we refer a (timestep, configuration) pair as an element of the transcript.

RAM Programs. Our starting point is a time T and space S RAM program M that takes as input (x, w). Similar to [5], we transform any given M in order to simplify the RAM to circuit satisfaction. Precisely, we first transform M into a RAM program M' which is then transformed into M'' as described below.

1. M' proceeds as per M except that it first reads the input x and writes it into the work tape. If the input is of size $|x|$, then the first $2|x|$ timesteps involve reading x from the input tape and writing it into the work tape one field element at a time.
2. M'' is the similar as M' except that, after every S timesteps of M', M'' makes a pass over the entire work tape (i.e., $O(S)$ memory cells) before and after the execution of the S timesteps of M'.[8]

For the rest of the description, we assume that the RAM program M has undergone the two transformation described above. The time T and space S for M include the overheads incurred by the above transformations. We now describe the structure of the circuit C obtained from the RAM-to-Circuit reduction[5] followed by the generation of extended witness X for C.

Circuit Description. The circuit C obtained by transforming a RAM program M (as per [5]) takes time T and requires space S. We assume C consists of addition and multiplication gates over the field $\mathbb{F}$.

The circuit C is equivalent to M and can be evaluated gate-by-gate in $\widetilde{O}(T)$ and space $\widetilde{O}(S)$. The inputs to C are (x, w') where w' is a non-deterministic input. At a high-level, C takes the entire transcript (ordered by time) as input and checks if the transcript satisfies time and memory consistency checks. Time-consistency check ensures that the configuration at timestep τ follows the configuration at timestep $\tau - 1$ by executing a single instruction. On the other hand, memory-consistency check imposes that the value read from a memory location at any timestep is consistent with value that was last written into the same memory location. To verify whether these two checks hold, C takes both time-ordered and memory-ordered transcripts, say $\overrightarrow{S}_{\mathsf{time}}$ and $\overrightarrow{S}_{\mathsf{mem}}$, as inputs. However, this is not

[8] This transformation increases the running time of the RAM program by a factor of 3.

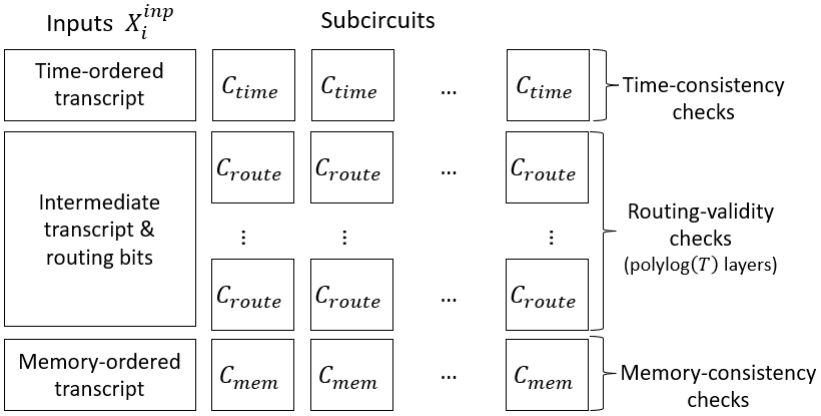

Fig. 2. A depiction of the i^{th} part of the circuit (i.e., $\mathcal{P}_i$) along with its inputs. Here, $C_{time}, C_{route}, C_{mem}$ are the subcircuits for time-consistency, routing-validity and memory-consistency checks respectively. The inputs X_i^{inp} to each type of subcircuits are shown on the left of the corresponding subcircuits.

sufficient as the time and memory-ordered transcript need not correspond to the same execution of the M. In other words, we need to check if $\overrightarrow{\mathcal{S}}_{\mathsf{time}}$ and $\overrightarrow{\mathcal{S}}_{\mathsf{mem}}$ are permutations of each other; this check can be done using routing networks by sorting the time-ordered transcript into a memory ordered transcript. We additionally provide the routing bits and the intermediate permutations of the transcripts as non-deterministic inputs to C and then check that the routing is valid (which we call the routing validity check).

The routing network requires $T \cdot \mathsf{polylog}(T)$ space to sort a transcript of size T. When S is smaller than T, the overhead in space required for the RAM-to-Circuit reduction is not desirable. In this scenario, the space complexity can be improved to $S \cdot \mathsf{polylog}(T)$ by adopting the techniques of [5]; they observe that permutation from $\overrightarrow{\mathcal{S}}_{\mathsf{time}}$ to $\overrightarrow{\mathcal{S}}_{\mathsf{mem}}$ has special locality properties that can be used to obtain a space-efficient routing network. In more detail, no element needs to be routed for more than $2S$ distance from its position in $\overrightarrow{\mathcal{S}}_{\mathsf{time}}$. This locality property is obtained by transforming the RAM program M (as described above): every S consecutive timesteps of M are preceded and followed by reading all S memory locations, thereby increasing the time by a factor of 3. Each memory location is accessed at least once in a window of $2S$ timesteps.

A detailed description of the structure of C is as follows:

1. We divide the (time-ordered) transcript into B parts such that each part comprises of $2S$ consecutive elements i.e. (timestep, configuration) pairs of the transcript and any two consecutive parts have S overlapping elements. Namely, the i^{th} part of the transcript consists of $2S$ elements starting at timestep $i \cdot S$ to $(i \cdot S + 2S - 1)$ in the time-ordered transcript $\overrightarrow{\mathcal{S}}_{\mathsf{time}}$ for $i \in [B]$. Here, we set $B = T/S$.
2. We process each part of the transcript independently and ensure that it satisfies time-consistency, memory-consistency and routing-validity checks. So, the circuit associated with each part of the transcript is described modularly, denoted by $\mathcal{P}_i$. While describing the checks below, we use the term transcript (and also the notation $\overrightarrow{\mathcal{S}}_{\mathsf{time}}$ and $\overrightarrow{\mathcal{S}}_{\mathsf{mem}}$) to refer to only parts of the transcript associated with $\mathcal{P}_i$. The inputs and the checks for $\mathcal{P}_i$ are as follows.
   (a) The (non-deterministic) inputs for the three checks are: the time-ordered transcript $\overrightarrow{\mathcal{S}}_{\mathsf{time}}$, memory-ordered transcript $\overrightarrow{\mathcal{S}}_{\mathsf{mem}}$, $\mathsf{polylog}(T)$ intermediate transcripts and routing bits. Note that each of these transcripts are permutations of $O(S)$ (timestep, configuration) pairs. The intermediate transcript is an arbitrary permutation of the $O(S)$ (time, configuration) pairs and corresponds to an intermediate layer in a routing network.
   (b) *Time-consistency check* verifies whether two given (timestamp, configuration) pairs are ordered by time. Let C_{time} be the subcircuit that checks time-consistency. On input $(\tau, \mathcal{S})$ and $(\tau', \mathcal{S}')$, C_{time} outputs 1 if and only if $\tau' = \tau + 1$ and configuration $\mathcal{S}'$ is obtained by running a one step of M on $\mathcal{S}$; otherwise outputs 0.

(c) *Memory-consistency check* verifies whether two given (timestamp, configuration) pairs are (i) ordered by the memory locations accessed in that step with ties broken based on time and (ii) consistent with respect to memory accesses i.e. the values read are consistent with last written value into the same memory location. Let the subcircuit associated with this check be C_{mem}.

(d) *Routing-validity check* verifies whether the routing from the time-ordered transcript to the memory-ordered transcript has been done correctly. The non-deterministic part of the input consists of routing bits and intermediate transcripts corresponding to each layer of the routing network. The intermediate transcripts simulate the "forwarding" of elements in a routing network by having copies of the elements at the position the element has been forwarded to; in other words, all the elements in the intermediate transcripts along the routing path are the same. The routing validity check ensures that: (i) every element of the transcript is forwarded as per the routing bits i.e. the element at the current position is a copy of the element at the forwarded position and (ii) no two elements are forwarded to the same position. Let C_{route} be the subcircuit associated with this check.

3. We also need to ensure that the time-ordered transcript associated with two consecutive parts $\mathcal{P}_i$ and $\mathcal{P}_{i+1}$ are consistent with each other for $i \in [B-1]$. This can be done using a simple check that verifies if the overlapping portions of the time-ordered transcript in $\mathcal{P}_i$ and $\mathcal{P}_{i+1}$ match.
4. *Input-consistency check* verifies whether the input x of the instance is consistent with the sequence of configurations. Specifically, x is consistent with the values read and written in the first $O(|x|)$ configurations as M first loads the input into the work tape during the first $O(|x|)$ configurations and never reads the input tape[9].
5. Lastly, C outputs 1 is all the checks pass; otherwise outputs 0.

The subcircuits and their corresponding inputs in each $\mathcal{P}_i$ are depicted in Figure 2. We claim that each gate in the C serves as input to at most $\mathsf{polylog}(T)$ gates. This claim will be used later to show that matrices for linear constraints are succinct. First, note that each of the subcircuits is of size $\mathsf{polylog}(T)$ and takes inputs of size $\mathsf{polylog}(T)$. Given a gate g_{id}, we count the number of gates that g_{id} is an input to. Suppose g_{id} belongs to subcircuit C_{type}, then we have three cases:

- If g_{id} is an internal gate of C_{type}, then it serves as input to only the gates within C_{type} and there are at most $\mathsf{polylog}(T)$ such gates.
- If g_{id} is an input gate of C_{type}, then there are at most a constant number of subcircuits it is a part of. Specifically, each configuration in ST are inputs to C_{time} as well as C_{route}[10]. The configurations in the intermediate transcripts are inputs to constant number of C_{route} subcircuits (as per the structure of the routing network). Lastly, the configurations in $\overrightarrow{S}_{\mathsf{mem}}$ are inputs to C_{mem} as well as C_{route}. Hence, the total number of gates that g_{id} is a input of is the upper bounded by the total number of gates in all of the subcircuits it is a part of, which is at most $\mathsf{polylog}(T)$.
- If g_{id} is an output gate of C_{type}, then there are at most a constant number of subcircuits it is a part of and hence is an input to at most $\mathsf{polylog}(T)$ following similar reasoning as the previous case.

Next, we describe the extended witness corresponding to the circuit C obtained from the RAM program M.

Extended Witness. The extended witness X is an arrangement of the wire values of C on inputs (x, w'). Recall that C was divided into B parts, each of which can be evaluated in a modular manner. Let X_i denote the evaluation of part $\mathcal{P}_i$ of the circuit where X_i is a matrix of size $m' \times \ell$. Then, X is essentially a matrix of size $(B \cdot m') \times \ell$, which is as follows.

$$X = \begin{bmatrix} X_1 \\ X_2 \\ \dots \\ X_B \end{bmatrix}$$

[9] Without loss of generality, we focus on RAM machines that first read the entire input into memory and then never access the input tape. This implies that $|x| < S$.

[10] More precisely, $\overrightarrow{S}_{\mathsf{time}}$ is an input to the first layer of C_{route} subcircuits where the routing network can be viewed as a layered graph and there's a C_{route} associated with each layer as depicted in Fig. 2.

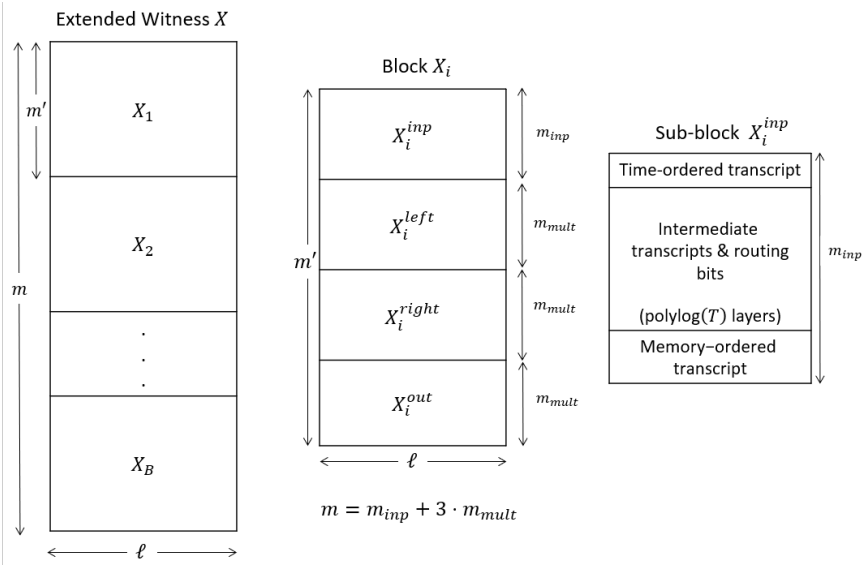

Fig. 3. A depiction of the extended witness X (left), a block X_i (centre) and the sub-block X_i^{inp} corresponding to each block (right).

We now describe the arrangement of wire values in each block X_i. Each X_i comprises of the non-deterministic inputs and wire values for multiplication gates associated with $\mathscr{P}_i$. The non-deterministic inputs comprise of the time-ordered transcript, memory-ordered transcript, intermediate transcripts for each layer of the routing network and routing bits. Overall, these inputs are represented as matrix X_i^{inp} of size $m_{\text{inp}} \times \ell$ where $m_{\text{inp}} = \text{polylog}(T)$. The wire values for multiplication gates are arranged as three matrices $X_i^{left}, X_i^{right}, X_i^{out}$, representing the left input, right input and output wires respectively. Each of these matrices is of size $m_{\text{mult}} \times \ell$ such that $X_i^{left} \odot X_i^{right} = X_i^{out}$. Refer to Figure 3 for depictions of the extended witness X, a block X_i and inputs X_i^{inp} within each block.

Given an extended witness X, we can can compute the encoded version of X by encoding each row of X using an RS code. We refer to the encoded extended witness as an *oracle*, which is denoted by U. The formal definition of an oracle is given below.

Definition 11. *[Oracle U] Let ζ and η be a set of interpolation and evaluation points (of sizes ℓ and n, respectively). To encode the extended witness X into an oracle U, each row of X is encoded into an RS code. In more details, to encode i^{th} row of X, a polynomial $p_i(\cdot)$ is generated where $p_i(\zeta_j) = X_{i,j}$ for all $j \in [\ell]$ (where $X_{i,j}$ is the element of X in the i^{th} row and j^{th} column). Next, the polynomial $p_i(\cdot)$ is evaluated at ζ to generate u_i which corresponds to the i^{th} row of U. More formally, $U_{i,j} = p_i(\eta_j)$ for all $j \in [n]$.*

In the description of X, we have divided the extended witness of X into *blocks*, where a block is essentially a collection of consecutive rows of X. A block in X can be specified by its size i.e. the number of rows and a starting point i.e., the index of the first row in X included in the block. A formal definition of a block is presented below.

Definition 12. *Given a matrix X of size $m \times \ell$, Y is said to be a block in X if Y is a matrix of size $m' \times \ell$ and consists of m' consecutive rows of X starting from* idxth *row.*

We will later show how X can be generated in space $\widetilde{O}(S)$ (in the proof sketch of lemma 10).

Linear Constraints. Note that there are two types of linear constraints for circuit C, the intra-block and inter-block linear constraints, which are as follows.

1. *Intra-block linear constraints* are expressed as $Ax_i = b$ for each block X_i where x_i is the flattened version of X_i and is of size $m' \cdot \ell$, A is a matrix of size $(m' \cdot \ell) \times (m' \cdot \ell)$ and b is a vector is size

$m' \cdot \ell$. In particular, these constraints ensure that (a) output wire values are the sum of inputs for every addition gate and (b) two wire values are equal if they are copies of each other. The latter constraint occurs as we require the wire values in the extended witness to be aligned in specific manner for multiplication gates. For instance, a multiplication gate that takes the 5^{th} and 6^{th} wires as input and the 10^{th} wire as output, then the 5^{th} wires value occurs in X_i^{inp} and X_i^{left}, and the 6^{th} wire value occurs in X_i^{inp} and X_i^{right}. Hence, multiple copies of the 5^{th} and 6^{th} wires values occur to meet the specific alignment requirements for multiplication gates and we can ensure the equality of these copies via linear constraints. Thus, an addition gate can be expressed as a single linear constraint and a multiplication gate requires at most three linear constraints (to check equality of copies of its left, right or output wires).

2. *Inter-block linear constraints* are expressed as $A' \begin{bmatrix} x_i \\ x_{i+1} \end{bmatrix} = b'$ for each blocks X_i, X_{i+1} where x_i and x_{i+1} is the flattened versions of X_i and X_{i+1} respectively, A' is a matrix of size $(2m' \cdot \ell) \times (2m' \cdot \ell)$ and b' is a vector is size $2m' \cdot \ell$. Recall that there are S common pairs of (timestep, configuration) that occur in the time-ordered transcript of blocks X_i and X_{i+1}. Specifically, the second half of the first row of X_i is the same as the first half of the first row of X_{i+1} and this is checked using $\widetilde{O}(S)$ linear constraints (as each (timestep, configuration) pair is of size $\mathsf{polylog}(T)$).

We next provide a sketch of the proof of Lemma 10 which states that the extended witness X (described above) can be generated efficiently for any given RAM program M and satisfies linear and quadratic constraints imposed by M. Further, this lemma states that the matrices A and A' associated with the linear constraints are succinct.

Proof Sketch of Lemma 10. Recall that intra-block linear constraints are represented as $Ax_i = b$ where x_i is the flattened vector corresponding to block X_i.

Generating Extended Witness X. We now discuss how to generate each of the blocks X in space $\widetilde{O}(S)$. First, we focus on generating the non-deterministic inputs. The time-ordered transcript $\overrightarrow{\mathcal{S}}_{\mathsf{time}}$ is generated by evaluating the ℓ steps of the RAM program associated with $\mathscr{P}_i$ where $\ell = 2S$. Then, the memory-ordered transcript $\overrightarrow{\mathcal{S}}_{\mathsf{mem}}$ is obtained by sorting the time-ordered transcript. Lastly, we need to compute a routing from $\overrightarrow{\mathcal{S}}_{\mathsf{time}}$ to $\overrightarrow{\mathcal{S}}_{\mathsf{mem}}$ which can be done in space $\widetilde{O}(S)$ as per lemma 9. The routing is captured by the routing bits and the intermediate transcripts, one per layer of the routing network. Note that $\overrightarrow{\mathcal{S}}_{\mathsf{time}}$ and $\overrightarrow{\mathcal{S}}_{\mathsf{time}}$ are each of size $\widetilde{O}(S)$. The routing network has $\mathsf{polylog}(T)$ layers and each layer has $\widetilde{O}(S)$ elements. Also, there are $\widetilde{O}(S)$ routing bits. Hence, the time and space to process non-deterministic inputs X_i^{inp} is $\widetilde{O}(S)$ per block.

Next, we generate the matrices $X_i^{left}, X_i^{right}, X_i^{out}$ which correspond to the left input, right input and output wire values of all multiplication gates in $\mathscr{P}_i$. Recall that the circuit and inputs associated with each part $\mathscr{P}_i$ are each of size $\widetilde{O}(S)$. Therefore, evaluating all the subcircuits $C_{time}, C_{mem}, C_{route}$ associated with $\mathscr{P}_i$ on input X_i^{inp} can be done in space $\widetilde{O}(S)$. Then, the wire values associated with the multiplication gates can be stored in matrices $X_i^{left}, X_i^{right}, X_i^{out}$.

Succinctness of A, A′ and A″. First, A has $\widetilde{O}(S)$ non-zero values. Specifically, we prove the claim that the number of non-zero values in A is proportional to size of the circuit (i.e., total number of addition and multiplication gates) associated with part $\mathscr{P}_i$. As described under linear constraints, an addition gate is expressed as a single linear constraint over three variables (i.e., two input and one output wire values). A multiplication gate is expressed as three linear constraint over two variables Hence, the total number of non-zero values is equal to the number of rows in A times the number of non-zero values per row; this is proportional to the size of circuit $\mathscr{P}_i$ which is $\widetilde{O}(S)$.

Each column of A is associated with a gate in $\mathscr{P}_i$. Recall that each gate in $\mathscr{P}_i$ serves as an input to at most $\mathsf{polylog}(T)$ gates in $\mathscr{P}_i$ (see the circuit description in Section 5.2.1). Thus, each column associated with a gate has at most $\mathsf{polylog}(T)$ non-zero values. Given a column j in A, we now show how to determine all the non-zero values in column j in time $\mathsf{polylog}(T)$. Since the circuit $\mathscr{P}_i$ comprises repeated subcircuits connected to each other (as shown in Figure 2) where each subcircuit has a fixed number of gates, it is straightforward to compute the exact subcircuit the j^{th} gate g_{id} belongs to, in time $\mathsf{polylog}(T)$ (for example, we can compute 56^{th} gate belongs to the 3^{rd} copy of subcircuit C_{time} if

Protocol 2 (Interleaved Linear Code Test) *Let $L[n, \ell, d]$ be the $RS_{\mathbb{F},n,\ell,\eta}$ code and L^m be the interleaved RS code.*
Input: *U.*
Oracle: *A purported L^m-codeword U that should encode $m \times \ell$ matrix X.*
Interleaved Test:

1. *$\mathcal{V}$ sends a random vector $r \in \mathbb{F}^m$ and sends it to $\mathcal{P}$.*
2. *$\mathcal{P}$ computes $q(\cdot) = \sum_{i\in[m]} r[i]p_i(\cdot)$ and sends it to $\mathcal{V}$ where $p_i(\cdot)$ is the polynomial representation of i^{th} row of U.*
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 q is of degree $< \ell$ and for every $i \in Q$,*

$$\sum_{j\in[m]} r[j]U_{j,i} = q(\eta_i).$$

Fig. 4. Interleaved Linear Code Test

each C_{time} has 20 gates each). All the constraints that the gate $\mathsf{g_{id}}$ (i.e. the output wire of the gate $\mathsf{g_{id}}$) is a part of can be computed by checking, for each gate $\mathsf{g_{id}}'$ in the subcircuit, whether $\mathsf{g_{id}}$ is an input to $\mathsf{g_{id}}'$ or not; this takes $\mathsf{polylog}(T)$ time since each subcircuit has $\mathsf{polylog}(T)$ gates. Thus, we can output a set of all the non-zero values in the j^{th} column of A along with the rows these non-zero values belong to. This completes the proof of succinctness of A. A similar argument is used to show that A' and A'' are succinct matrices.

5.2.2 Interleaved Linear Code Test In this section, we present the steps for testing interleaved linear codes in Figure 2. Similarly to Ligero [1], this check determines whether the encoded matrix U is an interleaved linear code. Upon committing to U, the verifier sends a randomly sampled $s \in \mathbb{F}$ to the prover. Let $p_i(\cdot)$ denote the polynomial associated with the i^{th} row of X and r be a random vector sent by the verifier. The prover generates the proof polynomial $q(\cdot) = \sum_{i\in[m]} r[i]p_i(\cdot)$ and sends it to the verifier. The verifier then checks that polynomial $q(\cdot)$ is consistent with a randomly selected t columns of U.

Lemma 11. *Protocol 2 is an IOP/IPCP for interleaved linear code test with the following properties:*

- **Completeness:** *If $U \in L^m$ and the $\mathcal{P}$ is honest, then $\mathcal{V}$ accepts with probability 1.*
- **Soundness:** *Let e be a positive integer such that $e < d/3$ where d is the minimal distance of the Reed-Solomon code. Suppose that a badly formed matrix U^* satisfies $d(U^*, L^m) > e$. Then, for any malicious $\mathcal{P}^*$ strategy, $\mathcal{V}$ will reject except with probability $(1 - e/n)^t + d/|\mathbb{F}|$.*
- **Complexity:**
  $\mathcal{P}$ has X on its input tape. In this model, $\mathcal{P}$ makes a single pass on the input tape. We denote by $m\ell$ the length of X, then the following complexities are obtained:
  - *Prover's Time = $O(m\ell)$.*
  - *Verifier's Time = $O(\kappa m)$.*
  - *Prover's Space = $O(\ell)$.*
  - *Verifier's Space = $O(\kappa)$.*
  - *Communication Complexity = $O(\ell)$.*
  - *Query Complexity = $O(\kappa)$.*

Proof. **Completeness:** Completeness of the interleaved linear test directly follows from the properties of the linear code.

Soundness: Let $\mathcal{P}^*$ be a malicious strategy for the prover and let $d(U^*, L^m) > e$. We analyse the soundness error by considering two cases.

In the first case, the codeword $w^* = r^{\mathbb{T}}U^*$ has $d(w^*, L) = e' > e$. Note that $d(w^*, L) = e'$ implies that any proof polynomial $q(\cdot)$ sent by prover in Step 2 will have at least e' evaluation points which differ from w^* i.e $q(\eta_j) \neq w^*[j]$ for $j \in E$ and $|E| > e'$. In this case, the verifier will only accept the interleaved test if all the t queries are from remaining $n - e'$ positions. The probability of that is $\frac{\binom{n-e'}{t}}{\binom{n}{t}}$. This can be generalized to get the probability,

$$\Pr[\mathcal{V}\text{accepts} \mid d(w^*, L) > e] \leq \frac{\binom{n-e-1}{t}}{\binom{n}{t}}$$

For the second case where $d(w^*, L) < e$, then it is shown in [1] that the probability of this event happening is $d/|\mathbb{F}|$.

Lemma 12. *[1] Let L be a Reed-Solomon code with minimal distance $d = n - \ell + 1$ and e be a positive integer such that $e < d/3$. Suppose $d(U, L^m) > e$. Then, for a random w^* in the row-span of U, we have*

$$\Pr[d(w^*, L) \leq e] \leq d/|\mathbb{F}|.$$

Combining both cases using a union bound argument, we can bound the soundness error as,

$$\begin{aligned}
\Pr[\mathcal{V} \text{ accepts } U^*] &< \Pr[\mathcal{V} \text{ accepts } U^* | d(w^*, L) > e] + \Pr[\mathcal{V} \text{ accepts } U^* | d(w^*, L) \leq e] \\
&< \Pr[\mathcal{V} \text{ accepts } U^* | d(w^*, L) > e] + \Pr[d(w^*, L) \leq e]] \\
&< \frac{\binom{n-e-1}{t}}{\binom{n}{t}} + d/|\mathbb{F}| \\
&< (1 - e/n)^t + d/|\mathbb{F}|
\end{aligned}$$

Complexity: The prover needs to compute a random linear combination of m rows, each row is of size ℓ and therefore the prover's time is $O(m\ell)$. The verifier runs a consistency check where it is required to evaluate a polynomial of degree less than ℓ at $t = O(\kappa)$ points. Additionally, the verifier needs to compute $r^{\mathbb{T}} U[j]$ on t columns where $U[j]$ is the j^{th} column of U. Both of these operations require $O(\kappa m)$ overhead. The prover sends a polynomial of degree less than ℓ and t columns of U where each column is of size m. Therefore the communication complexity is $O(m\kappa + \ell)$. The prover needs to store a polynomial $p_i(\cdot)$ at a time as well as the proof polynomial $q(\cdot)$. Therefore the prover's space is $O(\ell)$. Next, we require the explain the verifier's space.

The prover writes a polynomial $q(\cdot)$, which is of size ℓ, on to the proof tape of the verifier. Instead of the storing $q(\cdot)$ into the work tape, the verifier can first sample the set $Q \subset [n]$ of size t and then only store the evaluations of $q(\cdot)$ at $\{\eta_i\}_{i \in Q}$; this requires space $O(\kappa)$ as $t = O(\kappa)$. The verifier sends a random vector of size m to the prover. To make it space efficient, the verifier generates a seed s and both the prover and verifier can generate the vector r by evaluating using a PRF. More formally, given a PRF function f, each element of vector r can be set as $r[i] = PRF_k(s, i)$ where k is the PRF key. Upon receiving the rows of U restricted to columns in Q, the verifier computes the random linear combination of these rows by maintaining a running partial aggregate. Precisely, the verifier initializes a vector $\mathsf{agg} := 0$ of size t where agg_i corresponds to column $i \in Q$. Upon receiving the j^{th} element of column i, the verifier multiplies $U_{j,i}$ with the appropriate random multiplier and then updates agg_i i.e. $\mathsf{agg}_i = \mathsf{agg}_i + r_j U_{j,i}$. After processing all the columns in Q, the verifier needs to check if $\mathsf{agg}_i = q(\eta_i)$, similar to the protocol given in Figure 2. Hence, the random linear combination of the rows of U restricted to Q can be computed in space $O(\kappa)$ to store agg. The overall space required by the verifier is $O(\kappa)$ to store $O(\kappa)$ evaluations of the polynomial $q(\cdot)$ and the running partial aggregate agg.

5.2.3 Linear Test 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 $i \in [B]$.

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 3. Description of the parameters (Part 2).

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 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 interpolation 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.

We first prove that each element of X^{-1} is $X^{-1}[i,j] = \frac{1}{\ell} w^{-2(i-1)(j-1)f}$ by showing $XX^{-1}[i,j]$ is equal to 1 if $i = j$ and 0 otherwise.

$$\begin{aligned} XX^{-1}[i,j] = \sum_{k=1}^{\ell} X[i,k] \cdot X^{-1}[k,j] &= \sum_{k=1}^{\ell} w^{2(i-1)(k-1)f} \cdot \frac{1}{\ell} w^{-2(j-1)(k-1)f} \\ &= \frac{1}{\ell} \sum_{k=1}^{\ell} w^{2(k-1)f(i-j)} \\ &= \begin{cases} \frac{1}{\ell} \frac{w^{2\ell f(i-j)}-1}{w^{2(i-j)f}-1} & \text{if } i \neq j \\ \frac{1}{\ell} \sum_{k=1}^{\ell} 1 & \text{if } i = j \end{cases} \\ &= \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \end{cases} \end{aligned}$$

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 13. *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 10 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* = $O(\kappa)$.

Proof. **Completeness:** The completeness property directly follows from the properties of Reed-Solomon Codes.

Firstly, we show that if $Ay_i = b$ for all $i \in [B]$ and the prover constructs the proof polynomial $q(\cdot)$ correctly, then the check in Step 4b will be accepted by the verifier with probability 1. Next, we show that if U is computed correctly and the proof polynomial $q(\cdot)$ is generated honestly, then the check in Step 4c will be accepted by the verifier with probability 1.

Check in Step 4b is accepted with probability 1*:* To show that the check will be accepted the verifier, we show that $\sum_{j\in[\ell]} q(\zeta_j) = \sum_{k\in[B]} r'[j] r^{\mathbb{T}} b$ if the linear constraints are satisfied.

$$\begin{aligned}
\sum_{j\in[\ell]} q(\zeta_j) &= \sum_{i\in[B]} \sum_{j\in[\ell]} q_i(\zeta_j) \\
&= \sum_{i\in[B]} \sum_{j\in[\ell]} \sum_{k\in[m']} r'[i] \cdot r_k(\zeta_j) p_{I[i]+k-1}(\zeta_j) \\
&= \sum_{i\in[B]} r'[i] \sum_{j\in[\ell]} \sum_{k\in[m']} r_k(\zeta_j) p_{I[i]+k-1}(\zeta_j) \\
&= \sum_{i\in[B]} r'[i] \sum_{j\in[\ell]} \sum_{k\in[m']} r^{\mathbb{T}} A[(k-1)\ell + j] y_i[(k-1)\ell + j] \\
&= \sum_{i\in[B]} r'[i] \sum_{l\in[N]} r^{\mathbb{T}} A[l] y_i[l] && \text{(Set } l = (k-1)\ell + j \text{ and } N = m'\ell\text{)} \\
&= \sum_{i\in[B]} r'[i] r^{\mathbb{T}} A y_i \\
&= \sum_{i\in[B]} r'[i] r^{\mathbb{T}} b && (Ay_i = b)
\end{aligned}$$

Check in Step 4c is accepted with probability 1*:* We show that if U and $q(\cdot)$ are constructed correctly, then the verifier will accept the check with probability 1. To show this, we show that $q(\eta_i) = \sum_{j\in[B]} \sum_{k\in[m']} r'[j] r_k(\eta_i) U_{k+I[j],i}$.

$$\begin{aligned}
q(\eta_i) &= \sum_{j\in[B]} q_j(\eta_i) \\
&= \sum_{j\in[B]} r'[j] \sum_{k\in[m']} r_k(\eta_i) p_{I[j]+k-1}(\eta_i) \\
&= \sum_{j\in[B]} \sum_{k\in[m']} r'[j] r_k(\eta_i) U_{k+I[j],i}
\end{aligned}$$

This will hold if $U_{i,j} = p_i(\eta_j)$ which will always be true when U is constructed correctly. Hence, the verifier will accept the check in Step 4c with probability-1.

Soundness: Let $\mathcal{P}^*$ be a malicious strategy for the prover and let $d(U^*, L^m) < e$. We analyse the soundness error by considering two cases.

In the first case, we consider $q(\cdot)$ be the polynomial generated in Step 2 following the honest $\mathcal{P}$ strategy on input U. We first analyse the probability $r^{\mathbb{T}}Ay_i - r^{\mathbb{T}}b = 0$ given that $Ay_i \neq b$. We argue this probability to be $m_a/|\mathbb{F}|$. To show, we first define a vector $v = Ay_i - b$ and $r = (1, s, s^2, s^{m_a-1})$. Modelling v as the coefficients of a polynomial $o(\cdot)$, we can rewrite $r^{\mathbb{T}}v = o(s)$. Using Schwartz-Zippel lemma (Lemma 8), the probability that the polynomial $o(s)$ evaluates to 0 given that s is chosen randomly is $m_a/|\mathbb{F}|$. If the same vector r is chosen for all constraints, then the probability that for all i where $Ay_i \neq b$, the $r^{\mathbb{T}}Ay_i = r^{\mathbb{T}}b$ is satisfied for all i is bounded by $m_a/|\mathbb{F}|$. Next, we analyse the probability that if there exist an i such that $r^{\mathbb{T}}Ay_i \neq r^{\mathbb{T}}b$ but $\sum_{i\in[B]} r'[i]r^{T}Ay_i = \sum_{i\in[B]} r'[i]r^{\mathbb{T}}b$. As vector $r' = (1, s', s'^2, s'^{B-1})$, we can apply the same argument as before to say that probability of this event is $B/|\mathbb{F}|$.By applying the union-bound on these two events, we get the probability that the randomness are chosen such that a malicious prover passes the test is $M/|\mathbb{F}| + B/|\mathbb{F}|$.

Next, we analyze the probability that a malicious prover strategy is rejected conditioned on $q(\cdot)$ failing as above. Let $q'(\cdot)$ be the polynomial sent by the prover. Using the fact that $q(\cdot)$ and $q'(\cdot)$ are of degree at most $2\ell - 2$, the number of indices on which the polynomials agree on is at most $2\ell - 2$. Let Q' be the set of indices on which the polynomials agree on. Now as U^* is e-close to a codeword U, the verifier will reject the test if at least one position is chosen from the error positions $E = \Delta(U^*, L^m$ or from the positions where the polynomials disagree on $\overline{Q'}$. This will not happen with probability at most $\binom{e+2\ell-2}{t}/\binom{n}{t} \leq ((e+2\ell)/n)^t$. This is the bound on the probability that the malicious prover's strategy is accepted.

By combining the two cases using union bound, the verifier will reject except with probability $m_a/|\mathbb{F}| + B/|\mathbb{F}| + ((e+2\ell)/n)^t$.

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 $O(m'\ell(poly \log m_a + B \log \ell))$.

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 18. 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 $O(m'\kappa + m_a)$.

5.2.4 Quadratic Test This test checks if the quadratic constraints imposed by the circuit are satisfied. Precisely, this test verifies if the extended witness X, encoded as U, satisfies $Y_i^{\text{left}} \odot Y_i^{\text{right}} = Y_i^{\text{out}}$ for every $i \in [B]$ where $Y_i^{\text{left}}, Y_i^{\text{right}}$ and Y_i^{out} are $m_{\text{mult}} \times \ell$ submatrices of Y_i and Y_i is a $m' \times \ell$ submatrix of X starting at the $I[i]^{th}$ row of X and $\odot$ denotes pointwise product. We refer to Y_i as a *block* and $Y_i^{\text{left}}, Y_i^{\text{right}}$ and Y_i^{out} as *sub-blocks* of Y_i. We assume that positions of the blocks and sub-blocks within X can be efficiently computed by both the prover and verifier. Let $I_1[i]$ denote the index of the first row of X contained in Y_i^{left}. Similarly, we define $I_2[i]$ and $I_3[i]$ for sub-blocks Y_i^{right} and Y_i^{out} respectively.

The quadratic constraints associated with a block Y_i can be reduced to checking whether $r^{\mathbb{T}}(Y_i^{\text{left}} \odot Y_i^{\text{right}} - Y_i^{\text{out}}) = [0]_{1\times\ell}$ for all $i \in [B]$ where r is set $r = (1, s, s^2, \ldots, s^{m_{\text{mult}}-1})$. Further, the checks for all the blocks can be combined using vector $r' = (1, s', s'^2, \ldots, s'^{B-1})$ as the linear combiner and the resulting final constraint is presented in Equation 3.

$$\sum_{i\in[B]} r'[i] r^{\mathbb{T}}(Y_i^{\text{left}} \odot Y_i^{\text{right}} - Y_i^{\text{out}}) = [0]_{1\times\ell} \tag{3}$$

In the protocol, the prover encodes $r^{\mathbb{T}}(Y_i^{\text{left}} \odot Y_i^{\text{right}} - Y_i^{\text{out}})$ as a polynomial $q_i(\cdot)$ and combines all these polynomials as $q(\cdot) = \sum_{i\in[B]} r'[i] q_i(\cdot)$. The prover sends the polynomial $q(\cdot)$ and the verifier

Protocol 3 (Testing quadratic constraints over Interleaved RS Codes)
Input: U, $\{Y_i\}_{i\in[B]}$, $\{I[i]\}_{i\in[B]}$, $\{I_1[i]\}_{i\in[B]}$,$\{I_2[i]\}_{i\in[B]}$,$\{I_3[i]\}_{i\in[B]}$,
Oracle: *A purported L^m-codeword U that should encode $m \times \ell$ matrix X such that, for every $i \in [B]$ we have $Y_i^{left} \odot Y_i^{right} = Y_i^{out}$ where Y_i^{left}, Y_i^{right} and Y_i^{out} are $m_{mult} \times \ell$ submatrices of Y_i and Y_i is a $m' \times \ell$ submatrix of X starting at the $I[i]^{th}$ row of X. For each $i \in [B]$, let $I_1[i]$ denote the index of the first row of X contained in Y_i^{left}. Similarly, define $I_2[i]$ and $I_3[i]$ for Y_i^{right} and Y_i^{out} respectively. Further, we assume that two consecutive blocks are non-overlapping i.e., $I[i+1] \geq I]i] + m'$ for all $i \in [B-1]$*
Quadratic Test:

1. *$\mathcal{V}$ samples 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*

$$r = (1, s, s^2, \ldots, s^{m_{mult}-1}),$$
$$r' = (1, s', s'^2, \ldots, s'^{B-1}),$$
$$q_i(\cdot) = \sum_{j\in[m_{mult}]} r[j]\Big(p_{I_1[i]+j-1}(\cdot) \cdot p_{I_2[i]+j-1}(\cdot) - p_{I_3[i]+j-1}(\cdot)\Big)$$

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) *$q(\zeta_k) = 0$ for every $k \in [\ell]$*
   (c) *For every $k \in Q$,*

$$\sum_{i\in[B],j\in[m_{mult}]} r'[i]r[j](U_{I_1[i]+j-1,i} \cdot U_{I_2[i]+j-1,i} - U_{I_3[i]+j-1,i}) = q(\eta_i)$$

Fig. 5. Protocol for Quadratic Test.

checks if the quadratic constraints are satisfies by verifying $q(\zeta_j) = 0$ for all $j \in [\ell]$. Additionally, the verifier needs to verify the consistency among $q(\cdot)$, $Y_i^{\text{left}}, Y_i^{\text{right}}, Y_i^{\text{out}}$ for all $i \in [B]$. This can be achieved by executing a check involving $q(\cdot)$ and U.

Lemma 14. *Protocol 3 is an IOP/IPCP for testing quadratic constraints with the following properties:*

- **Completeness:** *Let U be a valid encoding of a $m \times \ell$ matrix X, such that for every $i \in [B]$, $Y^{left} \odot Y^{right} = Y^{out}$ where block Y_i is a $m' \times \ell$ submatrix of X starting at the $I[i]^{th}$ row of X, Y_i^{left}, Y_i^{right} and Y_i^{out} are $m_{mult} \times \ell$ submatrices of Y_i 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 $\exists i \in [B], Y_i^{left} \odot Y_i^{right} \neq Y_i^{out}$ where block Y_i is a $m' \times \ell$ submatrix of X starting at the $I[i]^{th}$ row of X and Y_i^{left}, Y_i^{right} and Y_i^{out} are $m_{mult} \times \ell$ submatrices of Y_i. Then for any malicious $\mathcal{P}^*$ strategy, $\mathcal{V}$ will reject except with probability $((e + 2\ell)/n)^t + (m_{mult} + B)/|\mathbb{F}|$.*
- **Complexity:**
  $\mathcal{P}$ has X on its input tape and has a work tape of size $O(m_{mult}\ell)$. 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, block Y_i is a $m' \times \ell$ submatrix of X starting at the $I[i]^{th}$ row of X and Y_i^{left}, Y_i^{right} and Y_i^{out} are $m_{mult} \times \ell$ submatrices of Y_i. Given that $\mathcal{P}$ is provided with a one-way linear access to X, then the following complexities are obtained:
  - *Prover's Time = $O(Bm_{mult}\ell \log \ell)$.*
  - *Verifier's Time = $O((Bm_{mult} + \ell)\kappa)$.*
  - *Prover's Space = $O(m_{mult}\ell)$.*
  - *Verifier's Space = $O(m_{mult}\kappa)$.*
  - *Communication Complexity = $O(\ell)$.*
  - *Query Complexity = $O(\kappa)$.*

Proof. **Completeness:** The completeness property directly follows from the properties of Reed-Solomon Codes.

Firstly, we show that if $Y_i^{\text{left}} \odot Y_i^{\text{right}} - Y_i^{\text{out}} = [0]_{m_{\text{mult}} \times \ell}$ for all $i \in [B]$ and the prover constructs proof polynomial $q(\cdot)$ correctly, then check in Step 4b will be accepted by the verifier with probability 1. Next, we show that if U is computed correctly and the proof polynomial $q(\cdot)$ is generated honestly, then the check in Step 4c will be accepted by the verifier with probability 1.

Check in Step 4b is accepted with probability 1*:* To show that the check will be accepted the verifier, we show that$q(\zeta_j) = 0$ for all $j \in [\ell]$ if all the quadratic constraints are satisfied.

$$\begin{aligned}
q(\zeta_j) &= \sum_{i \in [B]} q_i(\zeta_j) \\
&= \sum_{i \in [B]} \sum_{k \in [m_{\text{mult}}]} r'[i] r[j] \left(p_{I_1[i]+k-1}(\zeta_j) \cdot p_{I_2[i]+k-1}(\zeta_j) - p_{I_3[i]+k-1}(\zeta_j) \right) \\
&= \sum_{i \in [B]} \sum_{k \in [m_{\text{mult}}]} r'[i] r[j] \left(Y_i^{\text{left}}[(k-1), j] \odot Y_i^{\text{right}}[(k-1), j] - Y_i^{\text{out}}[(k-1), j] \right) \\
&= \sum_{i \in [B]} \sum_{k \in [m_{\text{mult}}]} r'[i] r[j] 0 \\
&= 0
\end{aligned}$$

Check in Step 4c is accepted with probability 1*:* We show that if U and $q(\cdot)$ are constructed correctly, then the verifier will accept the check with probability 1. To show this, we show that $\sum_{i \in [B], j \in [m_{\text{mult}}]} r'[i] r[j] (U_{I_1[i]+j-1,i} \cdot U_{I_2[i]+j-1,i} - U_{I_3[i]+j-1,i}) = q(\eta_i)$ for all $j \in Q$.

$$\begin{aligned}
q(\eta_j) &= \sum_{i \in [B]} q_i(\zeta_j) \\
&= \sum_{i \in [B]} \sum_{k \in [m_{\text{mult}}]} r'[i] r[j] (p_{I_1[i]+k-1}(\eta_j) \cdot p_{I_2[i]+k-1}(\eta_j) - p_{I_3[i]+k-1}(\eta_j)) \\
&= \sum_{i \in [B]} \sum_{k \in [m_{\text{mult}}]} r'[i] r[j] (U_{I_1[i]+k-1,j} \cdot U_{I_2[i]+k-1,j} - U_{I_3[i]+k-1,j})
\end{aligned}$$

This will hold if $U_{i,j} = p_i(\eta_j)$ which will always be true when U is constructed correctly. Hence, the verifier will accept the check in Step 4c with probability 1.

Soundness:

In the first case, we consider $q(\cdot)$ be the polynomial generated in Step 2 following the honest $\mathcal{P}$ strategy on input U. We first analyse the probability $r^{\mathbb{T}}(Y_i^{\text{left}} \odot Y_i^{\text{right}} - Y_i^{\text{out}}) = [0]_{1 \times \ell}$ given that $Y_i^{\text{left}} \odot Y_i^{\text{right}} - Y_i^{\text{out}} \neq [0]_{m_{\text{mult}} \times \ell}$. We argue this probability to be $m_{\text{mult}}/|\mathbb{F}|$.

To show, we first define a matrix $V = Y_i^{\text{left}} \odot Y_i^{\text{right}} - Y_i^{\text{out}}$ and $r = (1, s, s^2, s^{m_a - 1})$. Define $V[j]$ as a vector which is j^{th} column of matrix V. Assume, there exist an index j such that $V[j]$ is not a zero vector. We analyse the probability that $r^{\mathbb{T}} V[j] = 0$. Modelling $V[j]$ as the coefficients of a polynomial $o(\cdot)$, we can rewrite $r^{\mathbb{T}} v = o(s)$. Using Schwartz-Zippel lemma (Lemma 8), the probability that the polynomial $o(s)$ evaluates to 0 at a randomly chosen s is $m_{\text{mult}}/|\mathbb{F}|$. If the same vector r is chosen for all constraints, then the probability that for all j where $V[j]$ is not a zero-vector and the $r^{\mathbb{T}} V[j] = 0$ is satisfied for all i is bounded by $m_{\text{mult}}/|\mathbb{F}|$. Next, we analyse the probability that if there exist an i such that $r^{\mathbb{T}}(Y_i^{\text{left}} \odot Y_i^{\text{right}} - Y_i^{\text{out}}) \neq [0]_{1 \times \ell}$ but $\sum_{i \in [B]} r'[i] r^{\mathbb{T}}(Y_i^{\text{left}} \odot Y_i^{\text{right}} - Y_i^{\text{out}}) = [0]_{1 \times \ell}$. As vector $r' = (1, s', s'^2, s'^{B-1})$, we can apply the same argument as before to say that probability of this event is $B/|\mathbb{F}|$.By applying the union-bound on these two events, we get the probability that the randomness are chosen such that a malicious prover passes the test is $m_{\text{mult}}/|\mathbb{F}| + B/|\mathbb{F}|$.

Next, we analyze the probability that a malicious prover strategy is rejected conditioned on $q(\cdot)$ failing as above. Let $q'(\cdot)$ be the polynomial sent by the prover. Using the fact that $q(\cdot)$ and $q'(\cdot)$ are of degree at most $2\ell - 2$, the number of indices on which the polynomials agree on is at most $2\ell - 2$.

Let Q' be the set of indices on which the polynomials agree on. Now as U^* is e-close to a codeword U, the verifier will reject the test if at least one position is chosen from the error positions $E = \Delta(U^*, L^m)$ or from the positions where the polynomials disagree on $\overline{Q'}$. This will not happen with probability at most $\binom{e+2\ell-2}{t}/\binom{n}{t} \leq ((e+2\ell)/n)^t$. This is the bound on the probability that the malicious prover's strategy is accepted.

By combining the two cases using union bound, the verifier will reject except with probability $(m_{\text{mult}} + B)/|\mathbb{F}| + ((e+2\ell)/n)^t$.

Time Complexity:

Prover: The intermediate proof polynomial $q_i(\cdot)$ is composed by multiplying m_{mult} pairs of polynomial, each multiplication costs $O(\ell \log \ell)$ using FFT.The proof polynomial $q(\cdot)$ requires $O(Bm_{\text{mult}}\ell \log \ell)$ time as it is generated by taking a random linear combination of all the intermediate proof polynomials. The prover's total time is $O(Bm_{\text{mult}}\ell \log \ell)$.

Verifier: The verifier upon receiving the proof polynomial $q(\cdot)$, needs to execute two checks. The first check requires to verify whether $q(\zeta_j) = 0$ for all $j \in Q$. This check can be removed altogether. The prover sends the polynomial $q'(\cdot)$ in Step 2 instead of polynomial $q(\cdot)$. The polynomial $q'(\cdot)$ is a polynomial of degree $< \ell - 1$ where $q(\cdot) = q'(\cdot)z(\cdot)$ and $z(\cdot)$ is a ℓ-degree polynomial such that $z(\zeta_j) = 0$. If the verifier generates the $q(\cdot)$ polynomial using $q'(\cdot)$ and $z(\cdot)$ assuming polynomial $q'(\cdot)$ is of right degree, then $q(\zeta_j) = 0$ will always be satisfied for all $j \in [\ell]$. For verifying the second check, the verifier needs to evaluate $t = O(\kappa)$ evaluations on polynomial $q(\cdot)$. This can be easily computed by evaluating $q'(\cdot)$ and $z(\cdot)$ at those t evaluations and is done in $O(\ell\kappa)$ time. Additionally, the verifier requires $O(Bm_{\text{mult}}\kappa)$ time to verify the consistency between $q(\cdot)$ and U. The verifier's total time is $O((Bm_{\text{mult}} + \ell)\kappa)$.

Space Complexity:

Prover: To compute the polynomial $q(\cdot)$ in Step 2 in a space-efficient manner while making a single pass on the input tape X, the prover will store each sub-block $Y_i^{\text{left}}, Y_i^{\text{right}}$ and Y_i^{out} in the work tape and the space required is $O(m_{\mathsf{mult}}\ell)$. We compute $q(\cdot)$ by maintaining a running partial aggregate. The prover initialises the polynomial $\mathsf{agg}(\cdot) = 0$. Next, the prover processes each block Y_i and split it into triples of sub-block $(Y_i^{\text{left}}, Y_i^{\text{right}}, Y_i^{\text{out}})$ and computes $q_i(\cdot)$. The polynomial $\mathsf{agg}(\cdot)$ is updated as $\mathsf{agg}(\cdot) = \mathsf{agg}(\cdot) + q_i(\cdot)$ and requires $O(\ell)$ space. Therefore, the overall space complexity of the prover is $O(m_{\mathsf{mult}}\ell)$.

Verifier: The verifier upon receiving the proof polynomial $q(\cdot)$ executes three checks. But as explained previously, we can eliminate a check where instead of receiving $q(\cdot)$, the prover sends a polynomial $q'(\cdot)$ such that $q'(\cdot)z(\cdot) = q(\cdot)$ where $z(\zeta_j) = 0$ for all $j \in [\ell]$. If the verifier reconstructs $q(\cdot)$ by multiplying $q'(\cdot)$ and $z(\cdot)$, then the polynomial $q(\cdot)$ is evaluate to 0 at all interpolation points ζ_j, given that the degree of polynomial $q'(\cdot)$ is of right degree. The verifier needs to check whether polynomial $q'(\cdot)$ is atmost $k-1$ degree. For the last check, the verifier needs to evaluate $q(\cdot)$ at $t = O(\kappa)$ evaluations. This can be easily computed by evaluating $q'(\cdot)$ and $z(\cdot)$ at those t evaluations and requires only $O(\kappa)$ space. Each block Y_i is non-overlapping with other blocks and all the Y_i blocks are ordered based on $I[i]$ i.e $I[i] < I[i+1]$ where $I[i]$ is the first row of X contained in block Y_i. To verify the last check, the verifier computes as follows.For each query $j \in Q$, the verifier initialises each variable $\mathsf{agg}_j = 0$. The verifier process each block Y_i sequentially. While processing each block Y_i, the verifier processes each of the columns of U queried row by row. The verifier stores the columns of U from $m_1'[i]^{th}$ row of U to $(m_3'[i] + m_{\text{mult}} - 1)^{th}$ row of U and this requires $O(m_{\text{mult}}\kappa)$ space. The verifier then updates each variable agg_j for all $j \in Q$ as follows:

$$\mathsf{agg}_j = \mathsf{agg}_j + \sum_{k \in [m_{\text{mult}}]} r'[i]r[k]((U_{I_1[i]+j-1,i} \cdot U_{I_2[i]+j-1,i} - U_{I_3[i]+j-1,i}) \tag{4}$$

The verifier only requires to store all the agg_j for each j. Overall the verifier's space complexity is $O(m_{\text{mult}}\kappa)$.

5.2.5 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. In section 5.2.1, we presented a complexity-preserving reduction from $\mathsf{BH}_{\mathsf{RAM}}$ to extended witness X that satisfies the system of constraints defined in lemma 10. Further, this reduction is space-efficient as it sub-divides X into blocks and generates X block-by-block where each block requires space $\widetilde{O}(S)$.
2. In sections 5.2.2, 5.2.3 and 5.2.4, we present protocols for testing interleaved linear codes, linear constraints and quadratic constraints given 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 6. *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. Then the Protocol 4 is 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 exists 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 $O(\kappa)$ and number of rounds is a constant.

where κ is the security parameter.

Proof. **Completeness:** The Completeness follows from the correctness of RAM-to-circuit transformation as well as Completeness property in Lemma 11, 13 and 14.
Soundness: The soundness follows from the soundness property in Lemma 11, 13 and 14. The overall soundness follows by applying a union bound on the soundness of the test. More precisely, we argue the soundness by applying the union bound on the soundness of the two cases:

- $d(U, L^m) > e/4$: Since $e < d/3$, we can conclude from Lemma 11 that the verifier rejects Step 3 except with probability $(1 - e/n)^t + d/|\mathbb{F}|$.
- $d(U, L^m) \leq e$: We can conclude from Lemma 13 that the verifier rejects the linear test with probability $((e + 2\ell)/n)^t + (m_a + B)/|\mathbb{F}|$. As the value of m_a and B are different in each step where linear test is invoked. Substituting the correct value of m_a and B for each step, the verifier rejects Step 4a, 4b and 4c except with probability $((e + 2\ell)/n)^t + (|x| + 1)/|\mathbb{F}|$, $((e + 2\ell)/n)^t + (m'\ell + B)/|\mathbb{F}|$ and $((e + 2\ell)/n)^t + (2m'\ell + B - 1)/|\mathbb{F}|$ respectively. Similarly, we can conclude from Lemma 14, the verifier rejects Step 5 except with probability $((e + 2\ell)/n)^t + (m_{\mathsf{mult}} + B)/|\mathbb{F}|$.

Applying union bound on all the steps in both cases, the overall soundness error is $(1 - e/n)^t + 4((e + 2\ell)/n)^t + (d + 3m'\ell + m_{\mathsf{mult}} + |x| + 3B)/|\mathbb{F}|$.
Complexity: Given a RAM program M that runs in time T and uses space S, we set parameters as follows:

- $m = \widetilde{O}(T/S)$
- $m' = \mathsf{poly}(\kappa, \log(T))$

- $\ell = \widetilde{O}(S)$
- $B = O(T/S)$
- $m_a = \widetilde{O}(S)$
- $m_{\text{mult}} = \mathsf{poly}(\kappa, \log(T))$

where $\widetilde{O}(\cdot)$ ignores $\mathsf{polylog}(T)$ factor. We describe the complexities for the prover and verifier next.

Time Complexity:

Prover: We argue that the overall time complexities are the sum of the times complexities for the RAM to extended witness reduction, interleaved linear test, (intra-block and inter-block) linear tests, and quadratic test. This is because the prover generates X block-by-block where each block is an $m' \times \ell$ submatrix of X. Upon generating a block, the prover performs the steps of each of the tests corresponding to a block (as the test processes X block-by-block). So, the prover makes a single pass through X to compute the final outputs of each of the tests simultaneously. Thus, the overall time required by $\mathcal{P}$ is $\widetilde{O}(T)$.

Verifier: The verifier's time is the sum of the times required to execute all the tests on blocks. By plugging in the parameters as stated above, we get that the time for interleaved code test, linear test, and quadratic tests are $\widetilde{O}((T/S))$, $\widetilde{O}((T/S + S))$, and $\widetilde{O}((T/S + S))$ respectively. The verifier obtains $\widetilde{O}(1)$ columns by querying the oracle, which is written onto the verifier's proof tape row-wise. The verifier makes a single pass over these columns and computes the outputs of each of the tests.

Space Complexity:

Prover: The prover requires space $\widetilde{O}(S)$ to generate and store a block of X. As described earlier, upon generating a block, the prover invokes each of the tests on this block. By plugging in the parameters described above, we obtain that the prover uses $\widetilde{O}(S)$ space to process each of the tests. Hence, the overall space required by the prover is $\widetilde{O}(S)$.

Verifier: The overall space required by the verifier is the sum of space required for each of the three tests. This is because the verifier makes a single pass over each of the $\widetilde{O}(\kappa)$ columns (received by querying oracle) on its proof tape, which are obtained by querying the oracle (as described in the verifier's time complexity). In more detail, by substituting the parameters as described above, we get that the interleaved code test and quadratic tests require a space of $O(\kappa)$; and the linear test requires a space of $O(\kappa + |b|)$ where $|b|$ is the size of the vector used in linear test to express linear constraints. The additional $O(S)$ is required to store the vector b used in the linear tests. However, b can be efficiently generated using space $\widetilde{O}(1)$ because of the "succinct" representation of the circuit obtained from the RAM to circuit reduction. This reduces the space for the linear test is just $\widetilde{O}(1)$. Hence the overall space for the verifier is $\widetilde{O}(1)$.

5.3 Achieving Zero-Knowledge

In this section, we modify our protocol to achieve zero-knowledge using techniques from [1, 9]. In particular, there are two parts which leak information about the extended witness. Taking Interleaved test in Section 5.2.2 as an example, 1) the verifier receives a polynomial $q(\cdot)$ which is constructed by taking random linear combiners of each row of U in the the polynomial form. This polynomial leaks information about the extended witness. 2) Secondly, to check consistency between $q(\cdot)$ and U, the verifier receives t randomly chosen (by the verifier) columns of U. These columns also leak information about the extended witness as well.

To mitigate the first leakage, the prover adds a blinding codeword in U such that no information is leaked by taking the random linear combiner of the rows in U. To migitate the second leakage, the prover increases the degree of the RS code in U by t such that opening of t columns does not reveal anything about the polynomials and extended witness.

Below, we provide different modification for different parts of the protocol:

- **Oracle Generation:** Previously defined in Definition 11, the encoding of the extended witness is generated as follows. Each row i of X (denoted by x_i) is encoded into a polynomial $p_i(\cdot)$ i.e.

Protocol 4 (Our new argument construction for arithmetic circuits.) *Let X be the extended witness.*

1. *Process the extended witness X as follows.*
   (a) *Define m, ℓ be parameters such that X is a matrix of size $m \times \ell$.*
   (b) *Define blocks X_i of size $m' \times \ell$ starting at the $I[i]^{th}$ row of X for each $i \in [B]$.*
   (c) *For each block X_i, define sub-blocks as follows.*
   - X_i^{inp} *is submatrix of X_i and is of size $m_{inp} \times \ell$*
   - $X_i^{left}, X_i^{right}, X_i^{out}$ *are submatrices of X_i of size $m_{mult} \times \ell$.*

   Let $I_1[i], I_2[i], I_3[i]$ be the indices of the first row of X included in sub-blocks $X_i^{left}, X_i^{right}, X_i^{out}$ respectively.
   (d) *The following succinct matrices and vectors are used to express linear constraints.*
   - *A is a $(m' \cdot \ell) \times (m' \cdot \ell)$ matrix and b is a length $(m' \cdot \ell)$-vector.*
   - *A' is a $(2m' \cdot \ell) \times (2m' \cdot \ell)$ matrix and b' is a length $(2m' \cdot \ell)$-vector.*
   - *A'' is a $|x| \times (2m' \cdot \ell)$ matrix. $(2m' \cdot \ell)$-vector*
2. *Encode X and generate the oracle U as given in Definition 11.*
3. *Test U is e-close to a valid interleaved code by engaging in Protocol 2 on the following input: U*
4. *Test linear constraint by engaging in three executions of Protocol 1 on the following inputs:*
   (a) *Inputs for Execution 1 (Input consistency): U, 1, $\{y_1\}_{i\in[1]}$, $\{I[1]\}_{i\in[B]}$, A'', x.*
   (b) *Inputs for Execution 2 (Intra-block Linear constraints): U, B, $\{x_i\}_{i\in[B]}$, $\{I[i]\}_{i\in[B]}$, A, b.*
   (c) *Inputs for Execution 3 (Inter-block Linear constraints): U, $B-1$, $\{[x_i \;\; x_{i+1}]\}_{i\in[B-1]}$, $\{I[i]\}_{i\in[B-1]}$, A', b'.*

   where x_i is a flattened vector of block X_i (namely, x_i is the vector obtained by concatenating the rows of X_i).
5. *Test quadratic constraints by engaging in Protocol 3 on the following input: U, B, $\{X_i\}_{i\in[B]}$, $\{I[i]\}_{i\in[B]}$, $\{I_1[i]\}_{i\in[B]}$,$\{I_2[i]\}_{i\in[B]}$,$\{I_3[i]\}_{i\in[B]}$.*

Fig. 6. A full description of our space-efficient IPCP.

$p_i(\zeta_j) = x_i[j]$ for all $j \in [\ell]$ and ζ_j is a interpolation point. Each row i of the encoded witness U (denoted by u_i) is generated by evaluating polynomial $p_i(\cdot)$ at $n-$evaluation points. To increase the degree of polynomial by t by adding randomness. For each row i in X, we generate a new polynomial $p'_i(\cdot) = p_i(\cdot) + \mathsf{r}_i(\cdot)z(\cdot)$ where $\mathsf{r}_i(\cdot)$ is a random polynomial of degree t and $z(\cdot)$ is a polynomial which evaluates to 0 at set of all interpolation points ζ_j. This new polynomial $p'_i(\cdot)$ will be used to generate the encoded witness U. Note that the polynomial is randomised by using $\mathsf{r}_i(\cdot)$ but the evaluation of new polynomial at the interpolation points remain the same i.e. $p_i(\zeta_j) = p'_i(\zeta_j)$. This allows this encoding to be directly plugged in all the test without modifying them.

- **Interleaved Linear Test:** The verifier in the interleaved linear test verifies that the encoded witness U is constructed correctly. The prover generate a polynomial $p'(\cdot)$ of degree k and it's corresponding RS codeword u' and append u' in U.
- **Linear Test:** The verifier in the linear test verifies that $Ay_i = b$ for all $i \in [B]$. Using the two randomness vector r and r', the verifier verifies whether $\sum_{i\in[B]} r'[i]r^{\mathbb{T}}Ay_i = \sum_{i\in[B]} r'[i]r^{\mathbb{T}}b$. Encoding $r^{\mathbb{T}}A$ and y_i as polynomials, we can compute a proof polynomial $q(\cdot)$. The verifier checks if $\sum_{k\in[\ell]} q(\zeta_k) = \sum_{i\in[B]} r'[i]r^{\mathbb{T}}b$. The polynomial $q(\cdot)$ will leak information about the extended witness. Therefore, we generate an RS codeword u' by encoding a random message $\sigma = (\sigma_1, \ldots, \sigma_\ell)$ such that $\sum_{k\in\ell} \sigma_i = 0$. We append u' to mask the oracle U and append σ to each y_i. To verify that σ is constructed correctly, we modify A such it adds an additional constraint to verify the masked message σ is constructed correctly i.e. $\sum_{k\in\ell} \sigma_i = 0$.
- **Quadratic Test:** The verifier in the quadratic test verifies that $Y_i^{\text{left}} \odot Y_i^{\text{right}} - Y_i^{\text{out}} = [0]_{m_{\text{mult}}\times\ell}$ for $i \in [B]$. We add three messages a, b, c to $Y_i^{\text{left}}, Y_i^{\text{right}}, Y_i^{\text{out}}$ for all $i \in [B]$ such that $a \odot b - c = 0$. The encoding of these three messages $u'^{\text{left}}, u'^{\text{right}}, u'^{\text{out}}$ are appended to U.

After applying these modifications, we can convert the IPCP protocol into a Zero-Knowledge IPCP protocol.

5.4 From ZKIPCP to ZK

As addressed in [1, 9], transforming a ZKIPCP protocol into a ZK protocol can be achieved using Merkle Trees. Using ZKIPCP protocol, the honest-verifier ZK is constructed as follows. Each entry of the proof oracle (here each column of U) is committed using a statistically-hiding commitment scheme and is compressed using a Merkle hash tree. Each of these commitments can be instantiated using a family of collision-resistant hash functions. The rest of the steps in the honest-verifier ZK protocol will be the same as in the ZKIPCP protocol, except the verifier querying the oracle. In the ZK protocol, the prover will open the committed values corresponding to the verifier's oracle queries. Malicious verifiers can be handled using standard techniques [28, 29].

To use this transformation, the verifier needs to store the whole proof oracle U in the work tape. To optimise the space-efficiency of the prover, we compute the partial commitment of each column of U. As discussed in Section 4.5, we use the Merkle-Damgard hash function to commit to each column of U. This only requires space in the order of the number of columns of U, denoted by $n = \widetilde{O}(S)$. The prover initialises n variable $\mathsf{agg}_i = \hbar_0$[11], and computes each row i of the proof oracle U row by row and updates $\mathsf{agg}_j \leftarrow \mathsf{MD.Update}(\mathsf{agg}_j, U_{i,j})$. After processing each row of U, each agg_j denotes the commitment of each column of U. Next, the prover will compute the Merkle tree hash as described previously. To open the committed values corresponding the verifier's oracle, the prover will repeat the whole process to generate the commitments for each column of U. Next, the prover will provide the authentication path of the merkle tree for each verifier's query to confirm that the merkle tree was generated correctly and is consistent with the queries.

We can also compile our protocol directly to a non-interactive ZK protocol by applying the Fiat-Shamir transformation [21] in the random oracle model where the verifier's messages are emulated by applying the random oracle on the partial transcript in each round.

The communication complexity of our ZK protocol is $(S + (T/S)) \cdot \mathsf{poly}(\log T, \kappa)$. The communication complexity of the IPCP protocol is $S \cdot \mathsf{poly}(\log T, \kappa)$. Additionally, the prover sends a merkle tree root which is $O(1)$ element and the prover responds to t queries of the verifier. In each query, the verifier sends a column of U having $O(m) = (T/S) \cdot \mathsf{poly}(\log T, \kappa)$ elements as well as the authentication paths which consist of $O(\log n) = \mathsf{poly}(\log T, \kappa)$ elements. In total, for $t = O(\kappa)$ queries, the prover sends $(T/S) \cdot \mathsf{poly}(\log T, \kappa)$ elements.

5.5 Improving the proof length from $\widetilde{O}(T/S + S)$ to $\widetilde{O}(T/S)$

The proof length or the communication complexity of the ZK protocol described above is $(S + (T/S)) \cdot \mathsf{poly}(\log T, \kappa)$. In this section, we show how to improve the proof length to $(T/S) \cdot \mathsf{poly}(\log T, \kappa)$.

In our construction, recall that the prover encodes the extended witness X as U and performs three test - *Interleaved test*, *Linear Test* and *Quadratic Test*. Each test requires sending a polynomial $q(\cdot)$ of size $\widetilde{O}(S)$; this adds an $\widetilde{O}(S)$ term to the proof length. We will reduce this additive term to $\widetilde{O}(1)$ (i.e., $\mathsf{poly}(\log T, \kappa)$) by recursively using a symmetric-key based SNARK, namely the FRI protocol of [3]. At a high level, we need to prove that the polynomial $q(\cdot)$ is of the right degree where $q(\cdot)$ is of size $\widetilde{O}(S)$; Using the FRI protocol, the prover' time and space are $\widetilde{O}(S)$, while the verifier's time and space are polylogarithmic in S. In more detail, in each of the tests, the verifier checks:

1. The polynomial $q(\cdot)$ is of right degree.
2. For all $i \in [\ell]$, the interpolation points ζ_i and $q(\cdot)$ satisfies the following constraints:
   (a) *Linear Test:* The polynomial $q(\cdot)$ satisfies $\sum_{i \in [\ell]} q(\zeta_i) = a$ where a is a value known to the verifier.
   (b) *Quadratic Test:* The polynomial $q(\cdot)$ satisfies $q(\zeta_i) = 0$ for all $i \in [\ell]$.
3. For $j \in Q$, $q(\eta_j) = \beta_j$ where the query set Q satisfies $Q \subseteq [n]$ and $|Q| = t = O(\kappa)$ and β_j are known to the verifier.

Let's define $q_1(\cdot), q_2(\cdot), q_3(\cdot)$ to be the polynomial sent in the Interleaved linear test, Linear Test and Quadratic Test respectively. The FRI protocol provides a mechanism to commit to a polynomial where (a) the verifier can verify the degree of the polynomial and (b) the verifier can make a few queries to learn the value of the polynomial at the points needed for Step 3 above. This will suffice for the code

[11] Here, $\hbar_0$ is defined as per the specific instantiation of the Merkle-Damgard construction.

test. For the additional check on the linear test (resp., quadratic test), using ideas from [3], we can reduce the checks on the polynomials $q_2(\cdot)$ (resp., $q_3(\cdot)$ to checking the degree of related polynomials $o_1(\cdot), o_2(\cdot)$ (resp., polynomial $o_3(\cdot)$) along with a consistency check between the polynomials and $q_2(\cdot)$ (resp., $q_3(\cdot)$). All of this can be enabled via the FRI protocol.

First, we briefly recall the FRI protocol. The FRI protocol allows the verifier to be convinced that the polynomial is of right degree without sending the whole polynomial. In more detail, given a codeword C generated by a polynomial $q(\cdot)$, the prover proves that C is a codeword $\delta-$close to a codeword generated by a polynomial of degree d. So with some soundness error, the prover can convince the verifier that C is generated by evaluating a polynomial $q(\cdot)$ of degree d.

We presented our previous ZK protocol in the ZKIPCP model where the prover only commits to a single oracle followed by an interactive protocol. For the improved proof length, we will move to the ZKIOP model as the prover will commit to multiple oracles (required by the FRI protocol). The FRI prover requires running time $O(d)$ where d is the degree of the polynomial and FRI verifier requires running time $O(t \log d)$ where t is number of queries.

The FRI protocol is divided into two phases, the commit phase and the query phase. On input a polynomial $q(\cdot)$ (only known to verifier), a codeword C (verifier has oracle access) and d (denotes degree of polynomial claimed), the protocol is executed as follows. In the commit phase, the prover outputs a codeword as an oracle in each round i where i^{th} codeword generated by a polynomial $p_i(\cdot)$. The size of codeword and the polynomial in round i is reduced by factor of 2 in each round. The prover needs to store all the codewords. The total space for all the codewords is $2n$ where n is the size of the 1^{st} codeword. As the size of codeword C is $O(d)$, then the space required for the prover will be $O(d)$ in the IOPP model. In the query phase, the verifier queries $O(t)$ entries in each oracle generated by the prover in commit phase. For consistency between oracles generated in i and $i+1$, a simple consistency check is performed requiring $O(1)$ space for each query. While running a consistency check between two adjacent oracles, the verifier needs to store only queries for those oracles. Therefore the verifier needs to store only $O(t)$ query responses. Therefore in the IOPP model, the prover needs $O(d)$ space and verifier needs $O(t \log d)$ space.

While instantiating the protocols, the oracles can be committed using the Merkle Tree hash which will add a multiplicative overhead of $O(\log d)$. The proof size will be $t\mathsf{poly}(\log d)$ where t is number of queries.

Recalling that we can verify the first and second checks of the tests by checking the degree of polynomials $q_1(\cdot)$ and $o_1(\cdot), o_2(\cdot), o_3(\cdot)$. To carry out the third check, the prover outputs codeword of $q_1(\cdot), o_1(\cdot), o_2(\cdot), o_3(\cdot)$ as an oracle which is constructed by evaluating the polynomials at the evaluation points η_j for all $j \in [n]$. The verifier can query these oracles to verify the thrid check. Note that as $o_1(\cdot)$ and $o_2(\cdot)$ satisfy a constraint with $q_2(\cdot)$, therefore the codeword of $q_2(\cdot)$ can be generated using codeword of $o_1(\cdot)$ and $o_2(\cdot)$ and the same applies for $o_3(\cdot)$ and $q_3(\cdot)$.

Reducing $q_2(\cdot)$ to $o_1(\cdot)$ and $o_2(\cdot)$:

Denote $z(\cdot)$ is a polynomial of degree ℓ such that $z(\zeta_i) = 0$ for all $i \in [\ell]$ and $z'(\cdot)$ is a polynomial of degree 1 where the constant coefficient is 0 and the first coefficient is 1.

To verify that $\sum_{i\in[\ell]} q_2(\zeta_i) = a$, we first define a new polynomial $q'(\cdot)$ where $q'(\cdot) = q_2(\cdot) - a$. Next, we write the polynomial $q'(\cdot)$ as :

$$q'(\cdot) = w_1(\cdot) + w_2(\cdot)z(\cdot)$$

where $q'(\cdot)$, $w_1(\cdot)$ and $w_2(\cdot)$ are polynomials of degree $< 2\ell - 1$, $< \ell$ and $< \ell - 1$. We note that $w_1(\zeta_i) = q'(\zeta_i)$ for all $i \in [\ell]$ as the term $w_2(\zeta_i)z(\zeta_i) = 0$.

Notice that $\sum_{i\in[\ell]} w_1(\zeta_i) = 0$. As ζ_i are the interpolation points which are ℓ^{th} root of unity, we can apply Lemma 18 and show that the constant coefficient of the polynomial $w_1(\cdot)$ is 0. Therefore we define another polynomial $w_3(\cdot)$ of degree $< \ell - 1$ that satisfies $w_1(\cdot) = z'(\cdot) \cdot w_3(\cdot)$.

Therefore we can represent $q_2(\cdot)$ as:

$$\begin{aligned} q_2(\cdot) &= q'(\cdot) + a \\ &= w_1(\cdot) + o_2(\cdot)z(\cdot) + a \\ &= z'(\cdot) \cdot w_3(\cdot) + w_2(\cdot)z(\cdot) + a \end{aligned}$$

where $w_3(\cdot)$ and $w_2(\cdot)$ are polynomials of degree $< \ell - 1$ and $< \ell - 1$, $z(\cdot)$ is a polynomial of degree ℓ. We set $o_1(\cdot) = w_3(\cdot)$ and $o_2(\cdot) = w_2(\cdot)$.

To prove that the polynomial $q_2(\cdot)$ satisfies $\sum_{i\in[\ell]} q(\zeta_i) = a$, the prover needs to prove that $o_1(\cdot)$ and $o_2(\cdot)$ are polynomials of degree $\ell - 2$ and satisfies the constraint $q_2(\cdot) = z'(\cdot) \cdot o_1(\cdot) + o_2(\cdot)z(\cdot) + a$.

Reducing $q_3(\cdot)$ to $o_3(\cdot)$: To verify $q_3(\zeta_i) = 0$ for all $i \in [\ell]$, we first a new polynomial $o_3(\cdot)$ such that $q_3(\cdot) = o_3(\cdot)z(\cdot)$ where $z(\cdot)$ is a polynomial of degree ℓ such that $z(\zeta_k) = 0$ for all $i \in [\ell]$. To prove that the polynomial $q_3(\cdot)$ satisfies $q_3(\zeta_i)$ for all $i \in [\ell]$, the prover needs to prove that $o_3(\cdot)$ is a polynomial of degree $\ell - 2$ and satisfies the constraint $q_3(\cdot) = o_3(\cdot)z(\cdot)$.

Modified protocol.

Denote $z(\cdot)$ is a polynomial of degree ℓ such that $z(\zeta_i) = 0$ for all $i \in [\ell]$ and $z'(\cdot)$ is a polynomial of degree 1 where the constant coefficient is 0 and the first coefficient is 1. We provide the modifications to the test as follows:

1. **Interleaved Linear Test**: Instead of sending the polynomial $q_1(\cdot)$, the prover generates a codeword C by evaluating $q_1(\cdot)$ on the evaluation points η_i for $i \in [n]$. The verifier will query C at $j \in Q$ and verify if $C[j] = \beta_j$. The verifier will also execute a FRI protocol with inputs $(q_1(\cdot), C, \ell - 1)$ to verify the degree of $q_1(\cdot)$ where $q_1(\cdot)$ is claimed to be of degree $\ell - 1$ by the prover.
2. **Linear Test** : Instead of sending the polynomial $q_2(\cdot)$, the prover generates two codeword C_1, C_2 by evaluating $o_1(\cdot), o_2(\cdot)$ on the evaluation points η_i for $i \in [n]$. The verifier will query C_1, C_2 at $j \in Q$ and verify if $z'(\eta_j)C_1[j] + C_2[j]z(\eta_j) + a = \beta_j$. The verifier will also execute a FRI protocol with inputs $(o_1(\cdot), C_1, \ell - 2)$ and $(o_2(\cdot), C_2, \ell - 2)$to verify the degree of $o_1(\cdot)$ and $o_2(\cdot)$ where both are claimed to be of degree $\ell - 2$.
3. **Quadratic Test**: Instead of sending the polynomial $q_2(\cdot)$, the prover generates a codeword C by evaluating $o_3(\cdot)$ on the evaluation points η_i for $i \in [n]$. The verifier will query C at $j \in Q$ and verify if $z(\eta_j)C[j] = \beta_j$. The verifier will also execute a FRI protocol with inputs $(o_3(\cdot), C, \ell - 2)$ to verify the degree of $o_3(\cdot)$ and $o_2(\cdot)$ where is claimed to be of degree $\ell - 2$.

Recall that the Proof length or the communication complexity of our ZK protocol is $(S + (T/S)) \cdot \mathsf{poly}(\log T, \kappa))$. Sending a polynomial $q(\cdot)$ in each test required to send $d = S \cdot \mathsf{polylog}T$ elements. This part of the communication complexity is reduced to $t \cdot \mathsf{polylog}d = \kappa \cdot \mathsf{polylog}S$, thereby reducing the proof length to be $(T/S) \cdot \mathsf{poly}(\log T, \kappa)$. Hence we have the following theorem.

Theorem 7. *Assume that collision-resistant hash functions exist. For every NP relation that can be verified by a time T and space S RAM machine M with input x has a zero-knowledge argument-system 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 $\mathcal{P}$ is honest, then $\mathcal{V}$ accepts with probability 1.*
2. *Soundness: If there exists 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 negligible probability.*
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 $(T/S + S) \cdot \kappa \cdot \mathsf{poly}(\log T, \kappa)$ and uses space $\kappa \cdot \mathsf{poly}(\log T, \kappa)$.
   (c) The communication complexity is $T/S \cdot \mathsf{polylog}(T)$, and the number of rounds is $O(\log S)$.

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

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. Bar-Yossef, Z., Trevisan, L., Reingold, O., Shaltiel, R.: Streaming computation of combinatorial objects. In: Proceedings of the 17th Annual IEEE Conference on Computational Complexity, Montréal, Québec, Canada, May 21-24, 2002. pp. 165–174. IEEE Computer Society (2002). https://doi.org/10.1109/CCC.2002.1004352, https://doi.org/10.1109/CCC.2002.1004352
3. 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)
4. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable zero knowledge with no trusted setup. In: CRYPTO. pp. 701–732 (2019)
5. 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)
6. 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)
7. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: TCC. pp. 31–60 (2016)
8. Ben-Sasson, E., Sudan, M.: Short pcps with polylog query complexity. SIAM J. Comput. **38**(2), 551–607 (may 2008). https://doi.org/10.1137/050646445, https://doi.org/10.1137/050646445
9. Bhadauria, R., Fang, Z., Hazay, C., Venkitasubramaniam, M., Xie, T., Zhang, Y.: Ligero++: A new optimized sublinear IOP. In: CCS. pp. 2025–2038 (2020)
10. Bitansky, N., Chiesa, A.: Succinct arguments from multi-prover interactive proofs and their efficiency benefits. In: CRYPTO. pp. 255–272 (2012)
11. 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)
12. 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)
13. Blumberg, A.J., Thaler, J., Vu, V., Walfish, M.: Verifiable computation using multiple provers. IACR Cryptol. ePrint Arch. p. 846 (2014)
14. Bootle, J., Chiesa, A., Liu, S.: Zero-knowledge succinct arguments with a linear-time prover. IACR Cryptol. ePrint Arch. p. 1527 (2020)
15. Bowe, S., Grigg, J., Hopwood, D.: Halo: Recursive proof composition without a trusted setup. IACR Cryptol. ePrint Arch. p. 1021 (2019)
16. 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)
17. Bünz, B., Chiesa, A., Mishra, P., Spooner, N.: Recursive proof composition from accumulation schemes. In: TCC. pp. 1–18 (2020)
18. Chiesa, A., Ojha, D., Spooner, N.: Fractal: Post-quantum and transparent recursive proofs from holography. In: EUROCRYPT. pp. 769–793 (2020)
19. Coron, J.S., Dodis, Y., Malinaud, C., Puniya, P.: Merkle-damgård revisited: How to construct a hash function. In: CRYPTO. pp. 430–448 (2005)
20. Ephraim, N., Freitag, C., Komargodski, I., Pass, R.: Sparks: Succinct parallelizable arguments of knowledge. In: EUROCRYPT. pp. 707–737 (2020)
21. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: CRYPTO. pp. 186–194 (1986)
22. Franklin, M.K., Yung, M.: Communication complexity of secure computation (extended abstract). In: STOC. pp. 699–710 (1992)
23. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems (extended abstract). In: STOC. pp. 291–304 (1985)

24. Hazay, C., Ishai, Y., Marcedone, A., Venkitasubramaniam, M.: Leviosa: Lightweight secure arithmetic computation. In: CCS
25. 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
26. Ishai, Y.: Zero-knowledge proofs from information-theoretic proof systems (2020), https://zkproof.org/2020/08/12/information-theoretic-proof-systems
27. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: STOC. pp. 21–30 (2007)
28. Ishai, Y., Mahmoody, M., Sahai, A.: On efficient zero-knowledge pcps. In: TCC. pp. 151–168 (2012)
29. Ishai, Y., Weiss, M.: Probabilistically checkable proofs of proximity with zero-knowledge. In: TCC. pp. 121–145 (2014)
30. 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)
31. Kothapalli, A., Masserova, E., Parno, B.: A direct construction for asymptotically optimal zksnarks. IACR Cryptol. ePrint Arch. p. 1318 (2020)
32. 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)
33. Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. In: EUROCRYPT. pp. 52–78 (2007)
34. Merkle, R.C.: A certified digital signature. In: CRYPTO. pp. 218–238 (1989)
35. Mohassel, P., Rosulek, M., Scafuro, A.: Sublinear zero-knowledge arguments for RAM programs. In: EUROCRYPT. pp. 501–531 (2017)
36. 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)
37. Reingold, O., Rothblum, G.N., Rothblum, R.D.: Constant-round interactive proofs for delegating computation. In: STOC. pp. 49–62 (2016)
38. 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
39. Schwartz, J.T.: Fast probabilistic algorithms for verification of polynomial identities. J. ACM pp. 701–717 (1980)
40. Setty, S.T.V.: Spartan: Efficient and general-purpose zksnarks without trusted setup. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO. pp. 704–737 (2020)
41. Setty, S.T.V., Lee, J.: Quarks: Quadruple-efficient transparent zksnarks. IACR Cryptol. ePrint Arch. p. 1275 (2020)
42. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (nov 1979). https://doi.org/10.1145/359168.359176, https://doi.org/10.1145/359168.359176
43. Thaler, J.: Proofs, arguments, and zero-knowledge (2021), https://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.html
44. 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)
45. 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)

A Additional Lemmas for Testing Linear Constraints

Lemma 15. *Relation between co-efficient and roots of polynomial:*
Given a polynomial $p(\cdot)$ of $deg(P) = n$:

$$\begin{aligned} p(x) &= c_n \cdot X^n + c_{n-1} \cdot X^{n-1} + \ldots + c_1 \cdot X + a_0 \\ &= c_n \cdot (x - \alpha_1) \cdot (x - \alpha_2) \ldots (x - \alpha_{n-1}) \cdot (x - \alpha_n) \end{aligned}$$

where $\alpha_1, \alpha_2, \ldots \alpha_n$ are roots of the polynomial.
Sum of n roots is $(-1) \cdot c_{n-1}$

Lemma 16. *If roots of the polynomial $p(\cdot)$ is ℓ^{th} roots of unity, then :*

$$p(X) = X^{\ell} - 1$$

Combining the two Lemmas, we can observe that the sum of all ℓ^{th} root of unity is 0:

Corollary 1. *If H is a set of ℓ^{th} roots of unity, then the sum of all elements in H is 0.*

Lemma 17. *If monomial $p(\cdot) = X^t$, H be a set of ℓ^{th} roots of unity and $G = \{p(h)\}_{h \in H}$, then G is also a set of root of unity.*

Proof: If H is a set of ℓ^{th} roots of unity, then every element $h \in H$ satisfies $h^\ell = 1$. This is because every element h is a root of polynomial $X^\ell - 1$. Each element $g \in G$ also satisfies $g^\ell = 1$ and is shown mathematically below :

$$\begin{aligned} g^\ell &= (h^t)^\ell \\ &= (h^\ell)^t \\ &= 1^t \\ &= 1 \end{aligned}$$

As each element $g \in G$ satisfies $g^\ell = 1$, therefore all element in G are roots of unity.

Lemma 18. *Given a polynomial $p(\cdot)$ of degree t and H be the ℓ^{th} roots of unity, then $\sum_{a \in H} p(a) = \ell \sum_{i mod \ell = 0} c_i$.*

Proof: We use the corollary 1 and Lemma 17 to show that if H is a set consisting of ℓ^{th} roots of unity, then $\sum_{a \in H} p(a) = \ell \sum_{i \bmod \ell = 0} c_i$.

$$\begin{aligned} \sum_{a \in H} p(a) &= \sum_{a \in H} \sum_{i \in [0,t]} c_i a^i \\ &= \sum_{i \in [0,t]} c_i \sum_{a \in H} a^i \\ &= \sum_{i \bmod \ell = 0} c_i \sum_{a \in H} a^i + \sum_{i \bmod \ell \neq 0} c_i \sum_{a \in H} a^i \\ &= \sum_{i \bmod \ell = 0} c_i \sum_{a \in H} 1 + \sum_{i \bmod \ell \neq 0} c_i \sum_{a \in H} a^i \\ &= \sum_{i \bmod \ell = 0} \ell c_i + 0 \\ &= \ell \sum_{i \bmod \ell = 0} c_i \end{aligned}$$