How to prove a statement jointly? Distributed-prover Zero-Knowledge Protocols

Abstract: Traditional zero-knowledge protocols have been studied and optimized for the setting where a single prover holds the complete witness and tries to convince a verifier about a predicate on the witness, without revealing any additional information to the verifier. In this work, we the study the notion of distributed proof generation (DPZK) for arbitrary predicates where the witness is shared among multiple mutually distrusting provers and they want to convince a verifier that their shares together satisfy the predicate. We provide a new security definition, discuss new efficiency parameters, and propose a state-of-art construction for this paradigm: (i) we propose a new MPC-style security definition to capture the adversarial settings possible for different collusions between the provers and the verifier, (ii) we discuss new efficiency parameters for distributed proof generation on the number of rounds r_{pr} and the amount of communication c_{pr} among the provers, (iii) we propose a compiler that converts Interactive Oracle Proofs (IOP)-based ZK protocols into a DPZK protocol, (iv) we present a new single-prover zero-knowledge protocol Graphene in the Interactive Probabilistically Checkable Proofs (IPCP) paradigm which admits $O(N^{1/c})$ proof size. And, building on it, our DPZK protocol DP-Graphene, with N provers, generate the same proof size with a communication complexity c_{pr} of $O(\mathsf{N} \cdot N^{1-2/c} + \mathsf{N}^2 \cdot \max(N_s, N^{1-2/c}))$ and a round complexity r_{pr} of 2, where N_s denotes the number of multiplication gates that takes witnesses of more than one provers, in the arithmetic circuit representing the prover's computation in Graphene. This is achieved by compiling a Ligero-style protocol followed by several optimizations.

- Implementation or de-emphasize the protocol.
- Why this work is relevant to PETS. Added another application – Correctness in secure SOC.
- Add related work that clearly differentiates from prior works.
- Add a paragraph regarding UCZK.
- Add details in simulator.
- Check if the indistinguishability requires anything in the Ideal Functionality.
- Add details for the malicious setting.
- Length of inputs for provers.
- Comparison with Ligero and Graphene. (Soundness, performance, encoding) soundness is yet to be added.
- Motivate Honest verifier and add compiler reference to obtain malicious verifier from honest verifier.
- Update contribution.
- Add the proof of Bounded Independence.
- Update proofs in Appendix:C.
- Compiler from arbitrary IOP-based protocol to DPZK from. Updated the comparison table.

1 Introduction

A zero knowledge protocol tries to convince a verifier about the truth of a statement, without revealing any additional information. These include proving the correctness of transactions in cryptocurrencies such as [46] or validating sensitive web browser data reported during telemetry [17, 24]. Still in the dream of a decentralized world, there are multiple co-opetitive entities i.e., collaborating but mutually distrusting entities interacting with each other to obtain insights and maximize their goals. We envisage applications of zero knowledge proofs to enable these mutually distrusting entities to prove a claim on their joint data. In this setting, the traditional zero knowledge protocols are restrictive since they require a single prover to have the entire witness needed to generate the proof.

In this work, we study the general setting of *distributed prover zero-knowledge protocols* (DPZK) where multiple co-opetitive entities each possessing their own secret data want to prove to a verifier that their secret data together satisfies a predicate of common interest. This is to be done without revealing any information about their sensitive data to each other or to the verifier. We illustrate through some real-world applications:

- In privacy-preserving server outsourced computation, it is assumed that the number of corrupted parties can be at most t, where $t < n/2$, for the honest majority setting and $t < n-1$ for the dishonest majority setting and n is the number of servers. The protocols in these settings do not consider the scenario where the corruption exceeds t. In such a case, neither correctness nor privacy is ensured. So, suppose a user gets the output from the servers. In that case, there is no way to verify the correctness of the output, especially if the function is evaluated on inputs of multiple parties. In such a scenario, protocols can be enhanced by attaching a proof produced by all the servers stating the correctness of the computation. Here the statement would be the evaluated function, and the transcripts of each server become the share of the global witness. If the proof is valid, then the user is ensured that the output is indeed correct.
- A simple but pertinent scenario is of a *joint loan application* by an association of companies from a particular industry. The loan issuer has a set of financial requirements that it wants the association to satisfy, but there is no single trusted entity to act as the prover whom all the companies are willing to provide their sensitive business information with.
- In cryptocurrency settings [13, 42, 46], this would enable a *multi-wallet transaction* where the wallets are held by different parties or a *proof of joint stake* where the stake is held by different parties. These in turn enable secure collaboration applications by design on a blockchain network.
- In trade logistics business networks [22, 31, 49], a major reason for businesses to enter these networks is to benefit from cross-industry statistics. Publishing these statistics in a publicly verifiable manner without having a single trusted entity is another embodiment of this setting.

Formally, we have multiple provers $\mathcal{P}_1, \ldots, \mathcal{P}_{\mathsf{N}}$ respectively possessing witnesses $\mathsf{w}_1, \ldots, \mathsf{w}_{\mathsf{N}}$. For a predicate C the provers wish to prove to a verifier that $C(\mathsf{w}_1, \cdots, \mathsf{w}_{\mathsf{N}}) = 1$. A natural solution for distributed proof generation is to start with the prover algorithm with a single prover protocol and run this algorithm between multiple provers using multi-party computation (MPC). This generic construction was discussed by Pedersen in [45]. However, the generic construction is likely to be concretely inefficient, as implementing high level abstractions such as fields, groups and generic algorithms like FFT etc to compute the next prover message as an arithmetic circuit is challenging, and would lead to large circuit representations. Running multiparty computation over such circuits with several parties would incur more overheads. To get around the inefficiency of a generic construction, some prior works have proposed efficient distributed proof generation in restricted settings. These include simple predicates involved in threshold signatures [44], some sigma protocols [37], combined range proofs [12], or under trusted setup with a weaker adversarial model of majority of parties being honest [47]. The goal of this paper is to address the aforementioned shortcomings.

Our contributions

In this work, we provide a formal definition of distributed prover zero-knowledge (DPZK) in the real-ideal world paradigm. Furthermore, we identify and motivate relevant efficiency parameters to measure the efficiency of a DPZK protocol. In the next section, we elaborate more on the new efficiency parameters. We present a compiler that takes any IOP-based zero-knowledge protocol and converts it to an "MPC-friendly" zero-knowledge that supports DPZK. Finally, we produce a protocol obtained via compiling a Ligero-style proof system using our compiler. It is optimized further to attain comparable efficiency with state-of-the-art protocols while achieving DPZK. The discussions will focus on *public-coin honest-verifier* protocols with a transparent setup, which is essentially the setting for real-world applications we envisage: 1) protocols with such verifiers can be converted to succinct non-interactive arguments (zk-SNARGs) via the Fiat-Shamir and like transforms [6, 25], and 2) generating the parameters of the protocol should ideally *not* involve a trusted third party. In this work, we provide an honest verifier zero-knowledge protocol motivated by applications, however using known transformations, such as [30], we can obtain a zero-knowledge protocol while preserving the round complexity and the efficiency of both prover and verifier.

2 Overview of our work

2.1 Efficiency parameters for DPZK

In the single-prover setting, the efficiency of a zero-knowledge protocol is usually measured in terms of:

– *prover complexity*, denoted by $t_{\mathcal{P}}$, which represents the time complexity of the prover algorithm,

- *proof/argument size*, denoted by c_{zk} that refers to the amount of communication from the prover to the verifier,
- *verifier complexity*, denoted by $t_{\mathcal{V}}$, which represents the time complexity of the verifier algorithm, and
- *round complexity*, denoted as r_{zk}, which represents rounds of interaction between the prover and the verifier.

But, these parameters do not capture the core bottleneck in the setting of multiple provers. As a first step in our work, we identify additional parameters with significant impact in distributed proof generation. A lower value of each parameter enhances the practicality of the distributed protocol.

Proof-generation communication (c_{pr}): This parameter quantifies the amount of communication between the provers during the distributed proof generation. This is meant to capture additional MPC communication that provers would incur while computing the messages to the verifier. Using secret-sharing based MPC protocols [3, 18, 27], this depends on the number of multiplications between inputs from different parties (*cross-multiplications*), that need to be performed to compute the messages to the verifier.

Proof generation rounds (r_{pr}): This indicates the number of rounds of communication between the provers during the distributed proof generation. Informally, it denotes the number of rounds the provers need to run MPC protocols. This is orthogonal to r_{zk} and may not have any correlation with it. The parameter r_{pr} is also of cryptographic interest. If there is more than one round of prover message generation which requires MPC, care has to be taken to ensure a secure composition of the individual MPCs to prove the complete protocol is secure.

Shared circuit complexity: We will now elaborate a bit more on the number of cross-multiplications in the circuit that needs to be evaluated distributedly for proof generation since that is of practical relevance and influences prover communication (c_{pr}) and proof generation rounds (r_{pr}). In applications where the initial witness is canonically *partitioned* among the provers, the term *shared circuit* denotes sub-circuit consisting of wires, whose values are functions of inputs from more than one prover. Consider the earlier example of a trade-logistics business network, now on top of a blockchain where the hashes (or any other commitment) of large number of data points from different participants are stored. When parties come together to prove a statistic on their combined data, they need to establish that each data point used to compute the statistic corresponds to a hash entry in the blockchain, and the correctness of the result of the statistic computation on these data points. But here, the values of the circuit wires connecting each data point to its respective hash depends only on witness bits from a single party which owns the data point. Only the part of the circuit computing the statistic takes witness bits from multiple parties, and thus only this part needs to be considered for MPC while computing shares of extended witness. It would be desirable, though not evident, that the complexity of the MPC for *proof generation* also depends favorably on the size of the shared circuit. With the standard SHA hash function having around 30,000 gates, the practicality of distributed proof generation will be greatly improved if its parameters (like c_{pr} and r_{pr}) are linear in the size of the shared circuit N_s and not the total circuit size N.

In summary, the efficiency of a DPZK in the public-verifiable and non-interactive setting (which is our concern) will be measured via c_{pr}, r_{pr}, in addition to the parameters for the single prover protocol.

2.2 On the formal Definition of DPZK

Just like the efficiency considerations, the security definition for a DPZK protocol needs to account for additional interaction. In particular, the security definition needs to capture the fact that interaction among the provers to generate the proof does not leak knowledge about their respective witnesses to other provers. We come up with an MPC-style definition based on real-world ideal world paradigm [14, 16, 26, 40] that takes the above issues into account. In the ideal-world, the provers deliver their respective part of the witnesses and the functionality (that is parametrised with a language) combines them and check the assertion of a statement. In the real protocol, the provers participate in instances of MPC for 'proof-generating functions' to generate messages for the verifier. To keep the proof-size independent of the number of provers, one of the provers enacts in a special role called aggregator that prepares the message for the verifier taking into account communication from all its fellow provers and communicates the same to the verifier on behalf of all the provers. We say our protocol is secure if whatever an adversary (corrupting various subset of provers and verifier) can do in real execution can be done in the ideal execution. Lastly, keeping in mind the final goal of non-interactive and publicly-verifiable proofs/arguments, we explicitly mention the exact corruption scenario that we handle for our

constructions. For example, it is enough to tackle a semi-honestly corrupt verifier (since a non-interactive proof does not give any additional scope of misbehaviour to a maliciously corrupt verifier over a semi-honestly corrupt one). We refer to Section 4 for complete details. Next, we move on to discuss the single-prover construction on which we build our DPZK. To preserve the privacy, we assume that provers are not going to deviate from the protocol, since that may lead to rejection of the proof, though our protocol can withstand the scenario if all the provers are maliciously corrupt together due to the soundness property of the underlying ZK protocol.

2.3 Graphene: an MPC-friendly zero-knowledge protocol

We present an MPC-friendly single-prover protocol Graphene with a transparent setup which allows an efficient DPZK. Graphene achieves proof size and verification complexity competitive with the state-of-art zero knowledge protocols, while achieving efficient distributed proof generation. In particular, for a statement represented as a circuit with N multiplication gates, Graphene admits a proof size of $O(N^{1/c})$ for arbitrary $c \geq 2$, with a verification complexity of $O(N^{1-2/c})$ public key operations and $O(N)$ field operations. Graphene is a public-coin, honest verifier zero-knowledge protocol in the Interactive PCP (IPCP) [1, 28, 35, 36] paradigm. In IPCP paradigm, the verifier reads a small number of bits from PCP created by the prover and exchanges a small number of bits with the prover. There is known a transformation that converts ZK designed with IPCP to an interactive ZK via collision-resistant hash functions [33, 34] which in turn can be made non-interactive [6, 25].

Graphene follows the broad outline of Ligero [1]: (i) encoding the 'extended' witness via a suitable error-correcting code, where the extended witness is the concatenation of the wire values of the circuit (for the statement) in a topological order evaluated on the witness, (ii) using the encoding to check linear and quadratic relations on the extended witness, and (iii) a check to ensure that the extended witness is correctly encoded. The extended witness (henceforth witness) encoding is committed as an "oracle" and is queried during the linear and quadratic checks. Our key innovations are listed below.

New encoding and oracle: A core technical contribution of our work is a novel way to encode the witness and design the three checks in the zero-knowledge protocol around the new encoding. The *witness* is represented as a *3D matrix* of dimensions $p \times m \times s$ whose product is $O(N)$. The *witness encoding* is obtained by encoding every $m \times s$ 'slice' of the 3D matrix using a carefully selected product of Reed-Solomon codes. The *oracle* is now obtained by committing each $O(m)$-sized 'column' in the encoded witness using homomorphic vector commitments [45]. This results in an oracle that is a $O(p \cdot s)$-sized matrix of commitments. The use of homomorphic vector commitments helps in two aspects: (i) enabling distributed proof generation and (ii) achieving proof sizes independent of the largest dimension of the matrix. The latter is crucially dependent on the use of product codes to achieve soundness with very small *query complexity* (the size of the oracle revealed to the verifier). As we will see later, we will be able to regulate the three dimensions such that both the communication between the provers and the aggregator and that between aggregator and the verifier (argument size) are *sub-linear* in circuit-size N, while keeping low verification time. Lastly, we present a new coding theoretic result, which appears in Section 3.3 and proved in Appendix A.1, and leverage it to prove the soundness of our two stage "compression" as explained in the next paragraph.

Checking Linear and Quadratic Relations: As in other oracle-based protocols [1, 5], checking the linear and quadratic relations on the witness reduces to checking identities on the polynomials derived from polynomials encoding the witness. To ensure that the size of the final polynomial, which can be shared with the verifier is proportional to the smallest dimension of the three dimensional witness matrix, we need to achieve two stages of "compression". To ensure soundness for both stages of compression, we introduce an intermediate structure (a two dimensional matrix) to which the prover commits. This intermediate matrix enforces consistency between the three dimensional encoding, and one dimensional polynomial message to the verifier as it can be checked to be consistent with both. Having an extra dimension (over Ligero) also allows us to shave off the size of the largest dimension over which we run (expensive) inner-product arguments. This allows us to achieve inner-products over vectors of size $N^{1-2/c}$, which makes our verifier quite efficient despite using public key operations.

2.4 DP-Graphene: A new DPZK protocol

Building on Graphene, we design our DPZK protocol DP-Graphene with c_{pr} linear in size of the NP verification circuit for the language and with a constant r_{pr}. To be more precise, the communication for a N-prover proof generation for an N-multiplication gate circuit would be $O(\mathsf{N}^2 \cdot \max(N_s, N^{1-2/c}) + \mathsf{N} \cdot N^{1-2/c})$, where N_s is the shared circuit size as introduced in Section 2.1. Moreover, although r_{zk} is $O(\log N)$, r_{pr} is constant, limited to a single instance of secret-sharing based MPC on a circuit of size $\max(N_s, N^{1-2/c})$ and multiplicative depth as 1. Here we can see another benefit of viewing the witness as a 3 dimensional matrix. The largest dimension can be picked in such a way that, it can cover the complete shared circuit (N_s) if the shared circuit is not too big, which is true in many applications. To emphasize the non-trivial properties attained by our protocol, we discuss the roadblocks in naively adapting existing state-of-art zero-knowledge protocols to the distributed prover setting.

Roadblocks in achieving DPZK: Intuitively, to satisfy the desiderata of distributed proof generation with minimal prover communication, large parts of the messages to the verifier must be computable *locally* by the provers using their respective shares of the witness, which then must be efficiently aggregatable later to obtain the final message. Thus, the prover messages must have some "homomorphic" properties with respect to the witness. This requirement precludes naively extending recent efficient zero-knowledge protocols based on interactive oracles like [1, 5]. Realizing the oracle while constructing a non-interactive argument in the Random Oracle Model (ROM) uses hash functions, which exhibit no such homomorphic properties. We address this roadblock in our construction by making the oracle homomorphic. However, constructing such an oracle incurs expensive public key operations for the prover. For the restricted case of IPCP, where only the initial message is an oracle, the additional overhead though substantial is tolerable. For more general interactive oracle protocols, such as Aurora [5], our methods may not be the most efficient. Nevertheless, we pose efficient DPZK construction from general interactive oracle protocols as an interesting open question. Even if the messages in the protocol are largely homomorphic, small amount of non-homomorphic computation in several rounds would require repeated interaction among the provers. Intuitively, one way to avoid repeated prover interaction is to reduce the initial statement, to a statement involving "random witnesses", which can then be "handed off" to the aggregator to complete the remaining rounds of interaction with the verifier. It is essential that the witnesses in the reduced statement are "random" or "simulatable" to ensure that the aggregator gains no knowledge of provers' witnesses from them. In general, in such constructions one must ensure that in addition to verifier's view, the aggregator's view also leaks no knowledge. We did not see an obvious way to overcome this roadblock for Spartan [48], without resorting to expensive MPC among the provers. Among the existing zero-knowledge protocols, we were only able to obtain an efficient DPZK from Bulletproofs [12] with moderate effort and from Spartan [48] but without privacy guarantee. These constructions appear in Appendix D and serves as a baseline for comparison with DP-Graphene. Even for the Bulletproofs based construction, we are unable to achieve sub-linear prover communication for a sub-linear shared circuit size N_s. We extensively compare the two constructions across both the DPZK parameters, and the single prover parameters in Section 9.

Finally, we mention that the provers in DP-Graphene are semi-honest in trying to learn each other's witnesses while constructing messages for the verifier. This view fits well with the kind of applications we foresee, where incentive to make a joint claim is greater than that to deviate from the protocol. Even for this weaker adversarial model, proving the security of this construction is interesting because of a couple of reasons. The witness shares that the provers start with are additive, but they need not be from a secret sharing of a "global" witness. And, even if the provers' shares are secret shares of a global witness with a threshold of $\mathsf{N} - 1$, the aggregator in our construction sees messages from *all* the N provers. We prove the security of DP-Graphene taking care of these aspects when the provers and the aggregator are semi-honest.

2.5 Related Work

As mentioned earlier in the introduction, Pedersen [45] defines the notion of distributed proof generation and proposes a generic construction using MPC. Over the years, works have discussed distributed proof generation for a restricted class of predicates: Desmedt et al. [20] proposing proofs for graph isomorphism, various works [19, 39, 44] for proposing threshold signatures, Keller et al. [37] for a class of sigma protocols. Bulletproof [12] presents a "distributed" protocol to produce an aggregation of range proofs on inputs held by different parties. However, their completeness property is

weaker than ours, i.e, a set of provers with valid witness can only prove the statement only if each input gate is exclusive to one prover. Trinocchio [47] proposes a distributed proof generation method for Pinocchio [43] but only when there is a honest majority of semi-honest parties. Moreover, it needs the trusted setup model. The work DIZK [50] provides a distributed implementation of the proof generation of a non-interactive zero-knowledge protocol (zkSNARK). In Table 1, we compare the different metrics of efficiency for both the available DPZK protocols.

3 Notations and Preliminaries

3.1 Basic Notations

In this section, we describe the notation that will be followed in rest of the paper. We will often recall the notation in appropriate places. We use $[n]$ to denote the set $\{1, \ldots, n\}$. All the arithmetic circuits and constraint systems are assumed to be defined over finite field denoted by $\mathbb{F}$. For a vector $x \in \mathbb{F}^m$, x_i and $x[i]$ represent the i^{th} component of x for $i \in [m]$. For an $m \times n$ matrix over $\mathbb{F}$, we use (a) $A[i, j]$ to denote $(i, j)^{th}$ entry, (b) $A[i, \cdot]$ to denote the i^{th} row of A. Similarly, for an $p \times m \times n$ matrix X, we use (a) $X[i, j, k]$ to denote the $(i, j, k)^{th}$ entry, (b) $X[i, \cdot, \cdot]$ to denote the $m \times n$ matrix Y given by $Y[j, k] := X[i, j, k]$, (c) and analogously $X[\cdot, j, \cdot]$ and $X[\cdot, \cdot, k]$ to denote $n \times p$ and $p \times m$ matrices respectively. We will refer to sub-matrices $X[i, \cdot, \cdot]$ as *slices* of X, sub-matrices $X[\cdot, j, \cdot]$ as *slabs* of X and sub-matrices $X[\cdot, \cdot, k]$ as the *planes* of X. X^T denotes transpose of a two dimensional matrix X. Inner-product of two vectors X, Y is denoted as $\langle X, Y \rangle$. Inner-product of two three or two dimensional matrices X and Y, denoted as $\langle X, Y \rangle$, implies the inner-product of the corresponding one-dimensional vectors obtained from unrolling the matrices.

Throughout the paper we use several distance metrics on vectors and matrices, which we now introduce. For two vectors x, y of length n, we define $\Delta(x, y) = |\{i \in [n] : x_i \neq y_i\}|$. For two $m \times n$ matrices A, B we define $\Delta_1(A, B) = |\{j \in [n] : A[\cdot, j] \neq B[\cdot, j]\}|$ and $\Delta_2(A, B) = |\{i \in [m] : A[i, \cdot] \neq B[i, \cdot]\}|$. For $p \times m \times n$ matrices X and Y, we define $\Delta_1(X, Y) = \{k \in [n] : X[\cdot, \cdot, k] \neq Y[\cdot, \cdot, k]\}$ (no of planes that are different between the two matrices). Similarly we define $\Delta_2(X, Y) = |\{j \in [m] : X[\cdot, j, \cdot] \neq Y[\cdot, j, \cdot]\}|$ (no of slabs that are different between the two matrices). Let u be a vector and S be a set of vectors. We use $d(u, S) := \min\{\Delta(u, v) : v \in S\}$ to denote distance between u and S. Similarly for a matrix (two or three dimensional) U and a set of matrices $\mathcal{M}$, we define $d_i(U, \mathcal{M}) := \min\{\Delta_i(U, M) : M \in \mathcal{M}\}$ for $i = 1, 2$.

We use $\leftarrow_{\$}$ to denote drawing from a distribution uniformly at random.

3.2 Linear Codes

Definition 1. *Let n, k, d be positive integers with $n \geq k$ and $\mathbb{F}$ be a finite field. We call $L \subseteq \mathbb{F}^n$ to be an $[n, k, d]$ linear code if L is a k-dimensional subspace of $\mathbb{F}^n$ and d is the minimum value of $\Delta(x, y)$ for distinct $x, y \in L$. Elements of L are conventionally called* codewords.

For an $[n, k, d]$ code L, an $n \times k$ matrix $\mathcal{G}$ is called a *generator matrix* for L iff (i) $\mathcal{G}x \in L$ for all $x \in \mathbb{F}^k$ and (ii) $\mathcal{G}x \neq \mathcal{G}y$ for $x \neq y$. Clearly, such a matrix $\mathcal{G}$ has rank k. Similarly an $n \times (n - k)$ matrix $\mathcal{H}$ such that $y^T\mathcal{H} = 0$ for all $y \in L$ is called a *parity check* matrix for L. It is known that the above two matrices exist for any $[n, k, d]$ linear code L. We will assume that description of the linear code L includes a generator matrix $\mathcal{G}$ and a parity check matrix $\mathcal{H}$.

Definition 2 (Interleaved Code)**.** *For an $[n, k, d]$ linear code L and a positive integer m, we define a* row interleaved code $\mathsf{RIC}(L, m)$ *to be the set of $m \times n$ matrices A such that each row of A is a codeword in L. Similarly, we define a* column interleaved code $\mathsf{CIC}(L, m)$ *to be the set of $n \times m$ matrices B such that each column of B is a codeword in L.*

For an $[n, k, d]$ linear code L over $\mathbb{F}$, one can view $\mathsf{RIC}(L, m)$ as an $[mn, mk, d]$ linear code over $\mathbb{F}$ or alternatively, as an $[n, k, d]$ code over $\mathbb{H}$, where $\mathbb{H} \cong \mathbb{F}^m$.

Analogous to the row interleaved code, a column interleaved code $\mathsf{CIC}(L, m)$ can be viewed as an $[n, k, d]$ code over $\mathbb{F}^m$ by viewing each row of the codeword $B \in \mathsf{CIC}(L, m)$ as an element in $\mathbb{F}^m$. Here the distance metric for $\mathsf{RIC}(L, m)$ is given by the matrix distance Δ_1, while the distance metric for $\mathsf{CIC}(L, m)$ is given by the matrix distance Δ_2 as defined in Section 3.1.

Definition 3 (Product Code)**.** *Let L_i be an $[n_i, k_i, d_i]$-linear code for $i = 1, 2$. We define the product code $L_1 \otimes L_2$ to be the code consisting of $n_2 \times n_1$ matrices A such that each row of A is a codeword in L_1 and each column of A is a codeword in L_2.*

Protocols	c_{zk}	r_{zk}	$t_{\mathcal{P}}$	$t_{\mathcal{V}}$	r_{pr}	c_{pr}
D-Ligero	$O(\sqrt{N})$	$O(1)$	$O(N)E + O(N\log(N))M$	$O(N)E + O(N)M$	2	$O(\mathsf{N}\cdot N + \mathsf{N^2}\cdot\max(N_s, \sqrt{N}))$
D-Ligero++	$O(\log^2(N))$	$O(\log(N))$	$O(N\log(N))E + O(N\log(N))M$	$O(N)E + O(N)M$	$\log(N)$	$O((\mathsf{N}\cdot N + \mathsf{N^2}\cdot N)\log(N))$
D-Aurora	$O(\log^2(N))$	$O(\log(N))$	$O(N\log(N))E + O(N\log(N))M$	$O(N\log(N))E + O(N)M$	$\log(N)$	$O((\mathsf{N}\cdot N + \mathsf{N^2}\cdot N)\log(N))$
D-Bulletproofs	$O(\log(N))$	$O(\log(N))$	$O(N)E$	$O(N)E$	2	$O(\mathsf{N}\cdot N + \mathsf{N^2}\cdot N_s)$
Graphene	$O(N^{1/c})$	$O(\log(N))$	$O(\frac{N}{\log(N)})E + O(N\log(N))M$	$O(N^{1-2/c})E + O(N)M$	2	$O(\mathsf{N}\cdot N^{1-2/c} + \mathsf{N^2}\cdot\max(N_s, N^{1-2/c}))$

Table 1. Comparison amongst the DPZKs. Here N is the size of the circuit and, c is a positive integer of our choice. N is the number provers in the DPZK setting. M is to indicate the amount of computation required for a single multiplication on the field elements, and E is to indicate the amount of computation required for single exponentiation on the group elements, in which Dlog is assumed to be hard. And N_s denotes the size of the shared circuit.

Note that by definition, the product code $L_1 \otimes L_2$ is a row interleaved code of L_1 and a column interleaved code of L_2, i.e $L_1 \otimes L_2 = \mathsf{RIC}(L_1, n_2) \cap \mathsf{CIC}(L_2, n_1)$. For $A, A' \in L_1 \otimes L_2$, we define $\Delta_1(A, A') = |\{i \in [n_1] : A[\cdot, i] \neq A'[\cdot, i]\}|$ and $\Delta_2(A, A') = |\{i \in [n_2] : A[i, \cdot] \neq A'[i, \cdot]\}|$. The distance Δ_1 corresponds to distance function of the code $\mathsf{RIC}(L, n_2)$, where we view A, A' as codewords in $\mathsf{RIC}(L, n_2)$. Similarly, the distance Δ_2 corresponds to the distance function of the code $\mathsf{CIC}(L, n_1)$.

Definition 4 (Reed-Solomon Code)**.** *An $[n, k]$-Reed Solomon Code $L \subseteq \mathbb{F}^n$ consists of vectors $(p(\eta_1), \ldots, p(\eta_n))$ for polynomials $p \in \mathbb{F}[x]$ of degree less than k where $\eta_1, \ldots, \eta_n$ are distinct points in $\mathbb{F}$. We will use $\mathsf{RS}_{\boldsymbol{\eta}}[n, k]$ to denote the Reed Solomon code with $\boldsymbol{\eta} = (\eta_1, \ldots, \eta_n)$ and $deg(p) < k$.*

Lemma 3.1 ([41])**.** *Suppose a matrix $U^* \in \mathbb{F}^{h\times n}$ satisfies $d_1(U^*, \mathsf{RIC}(\mathsf{RS}_{\boldsymbol{\eta}}[n, \ell], h)) < e_1 \leq n - \ell$ and $d_2(U^*, \mathsf{CIC}(\mathsf{RS}_{\boldsymbol{\alpha}}[h, m], n)) < e_2 \leq h - m$. Then, there exists $U \in \mathsf{RS}_{\boldsymbol{\eta}}[n, \ell] \otimes \mathsf{RS}_{\boldsymbol{\alpha}}[h, m]$ and sets $S_1 \subseteq [n]$, $S_2 \subseteq [h]$ with $|S_1| > n - e_1$ and $|S_2| > h - e_2$ such that $U^*[i, j] = U[i, j]$ for $(i, j) \in S_1 \times S_2$.*

3.3 Coding Theory Results for our Constructions

The following coding theoretic result is the key to testing proximity of a three-dimensional matrix to a well-formed encoding. The proof appears in Appendix A.1.

Proposition 3.2 (3D Compression)**.** *Let L be an $[n, k, d]$ code over $\mathbb{F}$, and $\mathcal{C} := \mathsf{RIC}(L, m)$ be a row interleaved code of L. Let $\mathcal{C}^p$ denote the set of $p \times m \times n$ matrices U over $\mathbb{F}$ such that $U[i, \cdot, \cdot] \in \mathcal{C}$ for all $i \in [p]$. Let U^* be a $p \times m \times n$ matrix such that $d_1(U^*, \mathcal{C}^p) > e$. Then for any $m \times n$ matrix u_0 over $\mathbb{F}$, we have*

$$\Pr\left[d_1\left(u_0 + \sum_{i=1}^{p} r_i U^*[i, \cdot, \cdot], \mathcal{C}\right) \leq e\right] \leq \frac{d}{|\mathbb{F}|}$$

for a uniformly sampled $(r_1, \ldots, r_p) \leftarrow_{\$} \mathbb{F}^p$.

3.4 Vector Commitment Schemes and Inner-product Arguments

We define an interactive protocol that allows proving inner-product relation over committed vectors. For this we are using Pedersen vector commitment scheme Com.

We now define the language for inner-product w.r.t. the above as $\mathcal{L} \subseteq \mathbb{F}^m \times \mathcal{C} \times \mathbb{F}$ given by:

$$\mathcal{L} = \{(x, c, z) : \exists\, y, r \text{ s.t. } c = \mathsf{Com}(y, r) \text{ and } \langle x, y\rangle = z\}$$

Our language for inner-product is a simpler version of the one defined in [12], where the first argument is also hidden in a commitment. The inner product argument is used to reduce the communication.

We use inner-product argument from Bulletproofs [12]. Notably, we use a variant of inner-product where one of the vector is public and the other one is private. $\mathsf{InnerProduct}(\mathsf{pp}, x, \mathsf{cm}, v; z)$ is the notation we will be using through out the paper, where the public parameter pp consists of group description and generator, x is a public vector and cm is the commitment of the private vector z such that $\langle x, z\rangle = v$.

4 Definition for Distributed Proof Zero-Knowledge

This discussion will formally define a zero-knowledge protocol which supports multiple provers and distributed proof generation. With the eventual goal of protocols that are non-interactive and publicly-verifiable, we keep our focus on public-coin (where the verifier only sends truly random messages) protocols in mind. Our definition can be extended to private-coin protocols. Note that, the non-interactiveness concerns the communication between the set of provers and the verifier. We may still need multiple rounds of communication

just amongst the provers for proof preparation. Consider a language $L \in \mathsf{NP}$ and the corresponding relation R such that $x \in L \Leftrightarrow R(x, \mathsf{w}) = 1$ for some witness w. Let $\mathcal{P}_1, \ldots, \mathcal{P}_\mathsf{N}$ be N provers and $\mathcal{V}$ be the verifier. A public- coin DPZK protocol consists of four probabilistic polynomial time algorithms: $(\mathsf{SetUp}, \Pi, \mathsf{A}, \mathcal{V})$ as defined below.

- SetUp takes as input the security parameter 1^λ and optionally a trapdoor τ and outputs the public parameters σ of the system. The trapdoor input as well as the public parameter can possibly be empty.
- For a R-move DPZK, Π is defined by a sequence of R N-input and N-output functions/algorithms $\{\pi_i\}_{i\in[\mathsf{R}]}$, where a move indicates a one-shot communication from the provers to the verifier. The ith function takes the states of the provers at ith state i.e. $\pi_i(\mathsf{st}_1^i, \ldots, \mathsf{st}_\mathsf{N}^i)$. st_j^1 is set to $\mathcal{P}_j$'s share of the witness w_j, randomness r_j, the public parameters σ and is updated at the end of each move. For every, π_i there is a corresponding aggregator algorithm A_i that takes the outputs of π_i and generates a single message m_i for $\mathcal{V}$ for the ith move. Therefore, A for a R-move DPZK is defined as $\{\mathsf{A}_i\}_{i\in[\mathsf{R}]}$. m_i can possibly be a message in response to a random challenge that $\mathcal{V}$ outputs for ith step. Recall that $\mathcal{V}$ only outputs uniform random challenge in a public-coin DPZK. Finally, the output of $\mathcal{V}$ is either an `accept` (1) or a `reject` (0).

We note that computing π_i, without leaking the states to each other, may require interaction amongst the provers. It is important to note that the output of A alone is sent to the verifier on behalf of all the provers. In simple terms, A is an aggregator algorithm for the provers messages and is the key in making the proof-size delivered to the verifier independent of the number of provers. The task of accomplishing A can be assigned to (a) one or a constant-size subset of the provers, (b) an external entity or even (c) a hardware token. In this work, we take the route of allowing one of the provers to execute A.

Intuitively, a DPZK protocol for a language L will satisfy the following properties: (a) correctness: when the provers and the verifier are good, the verifier should accept if and only if the provers hold a valid witness; (b) soundness: the corrupt provers cannot make the verifier accept without holding a valid (joint) witness; (c) zero-knowledge: a corrupt verifier does not learn any information, except the assertion of the statement; (d) witness-hiding: the witness of the honest provers remain hidden from a collusion of a corrupt verifier and a subset of provers. Below, we formally prove security of such protocols based on real/ideal world paradigm.

4.1 Real-Ideal world definition for DPZK

We prove the security of our protocols based on the standard real/ideal world paradigm. Essentially, the security of a protocol is analyzed by comparing what an adversary can do in the real execution of the protocol to what it can do in an ideal execution, that is considered secure by definition (in the presence of an incorruptible trusted party). In an ideal execution, each party sends its input to the trusted party over a secure channel, the trusted party computes the function based on these inputs and sends to each party its respective output. Informally, a protocol is secure if whatever an adversary can do in the real protocol (where no trusted party exists) can be done in the above described ideal computation. We refer to [14, 16, 26, 40] for details regarding the security model.

The ideal execution: The "ideal" world execution of DPZK involves parties in $\mathcal{P}$ that includes N provers $\{\mathcal{P}_1, \ldots, \mathcal{P}_\mathsf{N}\}$ and a verifier $\mathcal{V}$, an ideal adversary $\mathcal{S}$ who may corrupt various subset of parties in $\mathcal{P}$, and a functionality $\mathcal{F}_{\text{DPZK}}$. The functionality is parametrized with an NP language L and the corresponding relation/verification function R. Each prover $\mathcal{P}_i$ sends (x, w_i) to $\mathcal{F}_{\text{DPZK}}$, where x is the statement and w_i is the ith part of the witness. The verifier $\mathcal{V}$ sends x. $\mathcal{F}_{\text{DPZK}}$ computes $\mathsf{w} = \mathsf{w}_1 \oplus \mathsf{w}_2 \ldots \oplus \mathsf{w}_\mathsf{N}$, where $\oplus$ is the combining function of the parts of the witness held by the provers distributedly, and sends $R(x, \mathsf{w})$ to everyone. $\mathcal{F}_{\text{DPZK}}$ is described below. Let the corrupt set be denoted as I. We let $\text{IDEAL}_{\mathcal{F}_{\text{DPZK}}, \mathcal{S}(z), I}(\vec{x})$ denote the random variable consisting of the output pair of the honest parties and $\mathcal{S}$ controlling the corrupt parties in I upon inputs $\vec{x} = (x_1, \ldots, x_\mathsf{N}, x_\mathcal{V})$ for the parties and auxiliary input z for $\mathcal{S}$.

The real execution: In the real model, the parties run Π_{DPZK} protocol. We consider a synchronous network with private point-to-point channels amongst the provers, and an authenticated broadcast channel. This means that the computation proceeds in rounds, and in each round parties can send private messages to other parties and can broadcast a message to all other parties. We stress that the adversary cannot read or modify messages sent over the point-to-point channels, and that the broadcast channel is authenticated, meaning that all parties know who sent the message and the adversary cannot tamper with it in any way. Nevertheless,

Functionality (The Distributed Prover Zero Knowledge Functionality $\mathcal{F}_{\text{DPZK}}$)

The functionality is parametrized with an NP relation R of an NP language L.

- Upon receiving input $x_i = (x, \mathsf{w}_i)$ from $\mathcal{P}_i$ $\forall i \in [\mathsf{N}]$ and x from $\mathcal{V}$, do the following: if x_i is an empty string or falls outside the range of the domain of $\mathcal{P}_i$'s input, reset $x_i = \texttt{abort}$.
- $\mathcal{F}_{\text{DPZK}}$ sends $|x_i|$ for all $\mathcal{P}_i \in [\mathsf{N}]$ and $\{x_i\}_{i \in C}$ to the simulator $\mathcal{S}$, where for all $i \in C$, $\mathcal{P}_i$ is a corrupt prover.
- If any of the x_i is abort, send abort to everyone in $\mathcal{P}$. Otherwise, compute $\mathsf{w} = \mathsf{w}_1 \oplus \mathsf{w}_2 \ldots \oplus \mathsf{w}_{\mathsf{N}}$, where $\oplus$ is the combining function of the parts of the witness held by the provers distributedly and send $R(x, \mathsf{w})$ to $\mathcal{S}$.
- $\mathcal{S}$ sends a command abort or continue to $\mathcal{F}_{\text{DPZK}}$. If $\mathcal{F}_{\text{DPZK}}$ receives abort it sends abort to all, otherwise it sends $R(x, \mathsf{w})$ to everyone.

Fig. 1. Ideal Functionality $\mathcal{F}_{\text{DPZK}}$

the adversary is assumed to be rushing, meaning that in every given round it can see the messages sent by the honest parties before it determines the messages sent by the corrupted parties.

As per Π_{DPZK}, the parties first produce the output of SetUp. Next, in ith move, the provers run a distributed protocol to compute π_i with respective states and a single prover, runs A_i on the outputs of π received from the provers, computes m_i and sends the output m_i of the computation to $\mathcal{V}$. m_i can possibly be a message in response to a challenge that $\mathcal{V}$ broadcasted. The protocol ends after R steps.

In summary, the "real" world execution involves the PPT parties in $\mathcal{P}$, and a real world adversary $\mathcal{A}$ who may corrupt a set of parties in I maliciously. Let $\text{REAL}_{\Pi,\mathcal{A}(z),I}(\vec{x})$ denote the random variable consisting of the output pair of the honest parties and the adversary $\mathcal{A}$ controlling the corrupt parties in I in the real execution, upon inputs $\vec{x}$ (defined in the same way as the ideal world) for the parties and auxiliary input z for $\mathcal{A}$.

Security via Indistinguishability of Real and Ideal world: The definition is given below

Definition 5. *Let $\mathcal{F}_{DPZK}$ be a $(\mathsf{N}+1)$-party functionality and let Π_{DPZK} be a $(\mathsf{N}+1)$-party protocol involving $\mathcal{P}$ for DPZK. We say that Π_{DPZK}* securely realizes *$\mathcal{F}_{DPZK}$ if for every* PPT *probabilistic real-world adversary $\mathcal{A}$, there exists an* PPT *expected polynomial-time ideal-world adversary $\mathcal{S}$, such that for every $I \subset \mathcal{P}$, every $\vec{x} \in (\{0,1\}^*)^{\mathsf{N}}$ where $|x_1| = \ldots = |x_{\mathsf{N}}|$, and every $z \in \{0,1\}^*$, it holds that: $\left\{\text{IDEAL}_{\mathcal{F}_{DPZK},\mathcal{S}(z),I}(\vec{x})\right\} \equiv \left\{\text{REAL}_{\Pi_{DPZK},\mathcal{A}(z),I}(\vec{x})\right\}$.*

Note that if the inputs of provers are not of the same length, then by padding zeros, inputs can be made of the same length. However, the verifier does not have any private input, so this requirement is exclusive to the provers.

When I is $\{\mathcal{P}_1, \ldots, \mathcal{P}_{\mathsf{N}}\}$, $\{\mathcal{V}\}$, a proper t-size subset of $\{\mathcal{P}_1, \ldots, \mathcal{P}_{\mathsf{N}}\}$, a proper t-size subset of $\{\mathcal{P}_1, \ldots, \mathcal{P}_{\mathsf{N}}\}$ plus $\mathcal{V}$, the above indistinguishability is referred as Soundness with Witness Extraction (SoWE), Zero-Knowledge (ZK), witness-hiding (WH), witness-hiding with collusion (WHwC) *respectively.*

While we give a very strong definition as above, motivated by several practical aspects, we relax the assumption on adversarial power and will focus on a subset of the above properties for our constructions. We elaborate below on this.

4.2 Our setting for DPZK

Our final goal is to produce arguments/proofs that is non-interactive and publicly-verifiable. We stress that the non-interactiveness concerns the communication between the set of provers and the verifier. We may still need multiple rounds of communication just amongst the provers for proof preparation. Most of the applications we foresee work best with these features. With these features as end-goal, we will design public-coin honest-verifier protocols that via generalized Fiat-Shamir heuristic [6, 25] can be turned into non-interactive and publicly-verifiable proofs/arguments. This setting has quite a many interesting bearing for us. First, in the honest-verifier setting, the verifier is considered to be only semi-honestly corrupt and so the ZK property needs to be proven keeping such a weaker adversary in mind. Second, non-interactiveness and publicly-verifiability make the two properties WH and WHwC equivalent, since the verifier has nothing more to add to the view of the corrupt provers in the case of collusion (in fact, the view of a corrupt aggregator itself subsumes the view of a verifier).

Our next relaxation comes in the form of imposing a semi-honest behaviour on the prover that acts as an aggregator. We justify the reason as follows. First, our aggregation function is deterministic. Having a deterministic aggregation procedure reduces the task to just combining the inputs and has a better promise to be executed through a hardware token that may not have randomness sampling capability. The determinism also in part helps the function to be reproducible by every prover when the information sent to the aggregator is sent over a public broadcast channel. This allows an easy check on the behaviour of the aggregator and a strong deterrent for the aggregation to be carried out honestly, as reputation may be at stake and applications typically bind the provers to act rationally (and hence honestly or semi-honestly) for the common cause of coming up with an accepting proof.

In summary, we will prove three properties for our protocols: (a) ZK assuming a semi-honest verifier, (b) SoWE tolerating malicious provers and (c) WH tolerating semi-honest provers and aggregator. Our protocol can easily be modified to achieve privacy from the maliciously corrupt provers. But we do not have a provably secure DPZK when the aggregator is maliciously corrupt. A DPZK which is provably secure when the provers and the aggregator is maliciously corrupt remains open.

As for the form of witness partition across the prover's, we assume that the provers jointly hold an additive sharing of the witness w. This is general enough and well supported by secret sharing protocols.

4.3 Related Notions

Closely related to our notion is the work of [37] which discusses threshold proofs. This work distributes the prover's side of interactive proofs of knowledge over multiple parties for: (i) improving the security against theft of the user's identity, (ii) improving robustness by ensuring that only a restricted size subset of the provers may be corrupted. The work of [37] primarily captures sigma protocols without allowing interaction among the provers. Our definition captures a broader class of protocols via allowing interaction amongst the provers. In other words, threshold proofs are a sub-class of the protocols we cater to. Also, other than increasing security via a decentralisation of the prover (or distributing prover's task), our goal is to capture scenarios where multiple provers would like to jointly prove a statement. Our real-ideal world based definition is inline with MPC-style definitions and is more powerful. [45] also defines distributed proofs. [45] considers the provers' witnesses to always be secret shares of a "global" witness and hence the provers' witnesses are *random* when the distributed proof generation begins. But, our definition allows for any distribution on the provers' witnesses, and this impacts some of our security proofs. In other words, any individual share of the witness used in [45] does not have any stand alone significance. On the other hand, in our security definition, any individual share of the witness is a private input of the corresponding shareholder, and it provides privacy of such individual shares. Another notion, called Multi-prover interactive proofs (MIP), was introduced by Goldwasser et al. [2]. Here, multiple provers hold a common witness corresponding to a statement and provide proofs. MIP is used to reduce the soundness error. However, the setting in MIP [2, 8] is entirely different from our notion since, in the DPZK setting, no provers hold a complete witness, and all the provers jointly provide a proof. A linking paradigm between zero-knowledge and MPC, called MPC-in-the-head, was introduced by Ishai et al. [32], where the prover runs a multiparty protocol in its head and use the view of the emulated parties to provide a proof. This process requires the opening of the views/inputs of the parties performing the MPC. Therefore, this approach cannot be used to obtain a DPZK protocol since, in this setting, opening a party's view to the verifier implies compromising that party's privacy. Moreover, the efficiency of the zero-knowledge construction from MPC-in-the-head-paradigm depends on the MPC protocol emulated by the prover, i.e., the protocol decides the number of parties, corruption scenario (honest majority or dishonest majority/semi-honest or malicious). In an arbitrary setting, using MPC-in-the-head-paradigm can be inefficient. [PKP:*It is starting abruptly*] One might ask to run MPC to generate the trusted setup and then use a protocol with trusted setup. But to generate trusted setup via MPC requires verifier to take part in the MPC, which precludes publicly verifiable non-interactive proofs. [PKP:*To add: Check the UC security definition and address the comment.*]

5 DPZK Compiler

In Ligero [1], an extended witness is viewed as a matrix of size $\sqrt{N} \times \sqrt{N}$. Then each row of the matrix is encoded. Then in that protocol, the prover is required to communicate messages, one of which is the size of a

row and $O(1)$ many columns of the encoded extended witness. This imposes a natural restriction on Ligero as they cannot skew the dimensions to reduce the proof size. Among the two dimensions, the largest dimension becomes the dominating factor that makes $\sqrt{N}$ is the best achievable communication complexity in Ligero. This bottleneck was resolved in Ligero++ [7]. They replace the communication of one dimension by using the inner product argument. More precisely, they rely on the inner product argument proposed in Virgo [51], which in turn relies on the Aurora proof system. Our optimization over Ligero is in a similar direction. We present the extended witness as a cube (3D matrix). One extra dimension brings more flexibility to a better trade-off between the communication complexity and the verification time. The key differences from Ligero/Ligero++ are the following. (i) We use bulletproofs based inner product argument on reducing the communication complexity, and also, this is mainly used to facilitate distributed prover version that others fail to achieve. (ii) We split the extended witness into three dimensions that aids in verification time while retaining the communication complexity proportional to the smallest dimension.

Further, we give a generic construction from an IOP-based proof system to obtain another proof system that offers DPZK. For most of the protocols, the communication cost among the provers is expensive. Our protocol Graphene provides a much efficient construction of the distributed prover version.

Lemma 5.1 (Compiler)**.** *Let $\langle \mathcal{P}, \mathcal{V} \rangle$ be an IOP-based zero-knowledge proof system with proof of knowledge property, where* Or *is the oracle construction algorithm. Then $\exists$ an IOP-based zero-knowledge proof system $\langle \mathcal{P}', \mathcal{V}' \rangle$ with an homomorphic oracle construction algorithm* H *such that it supports DPZK.*

Proof. Let $\langle \mathcal{P}, \mathcal{V} \rangle$ be an r round protocol. Furthermore, consider at the ith round $\mathcal{P}$ sends m_i to $\mathcal{V}$ and sets π_i as the oracle.

Note that m_i and π_i are functions of the statement x, (extended) witness w, $\mathcal{P}$'s randomness r_i, previous messages and oracles $(m_1, \ldots, m_{i-1}, \pi_1, \ldots, \pi_{i-1})$, and $\mathcal{V}$'s challenges.

In the new protocol $\langle \mathcal{P}', \mathcal{V}' \rangle$, $\mathcal{P}'$ sets $m'_i = m_i$ and $\pi'_i = \mathsf{Com}(\pi_i, r_i)$. Here $\mathsf{Com}(\cdot)$ is any homomorphic commitment scheme. In particular, we instantiate Com by using Pedersen commitment scheme. Therefore, the new oracle construction algorithm is $\mathsf{H}(\pi_i) = \mathsf{Or}(\mathsf{Com}(\pi_i, r_i))$. Similarly, the verification algorithm changes in the following way: If $\mathcal{V}$ queries jth position to the oracle π_i, then $\mathcal{V}'$ queries the same position and receives $\mathsf{Open}(\pi'_i[j])$, followed by the same computation of $\mathcal{V}$.

Completeness of $\langle \mathcal{P}', \mathcal{V}' \rangle$ holds directly from the completeness of $\langle \mathcal{P}, \mathcal{V} \rangle$. Consider an instance x, where $R(x, \mathsf{w}) = 1$ for the Relation R and $\mathcal{P}'$ has the witness w. Now in $\langle \mathcal{P}, \mathcal{V} \rangle$, $\mathcal{P}$ sends the messages m_i that is the same as the messages of $\langle \mathcal{P}', \mathcal{V}' \rangle$, and for the oracle instead of $\mathsf{H}(\pi_i)$, $\mathcal{P}$ executes $\mathsf{Or}(\pi_i)$. Since $\langle \mathcal{P}, \mathcal{V} \rangle$ is complete, and the proof is correct. Therefore $\mathcal{V}$ accepts the proof. Hence, the proof given by $\mathcal{P}'$ is also accepting, as the verification algorithm is the same. This ensures the completeness property.

Proof of knowledge of $\langle \mathcal{P}', \mathcal{V}' \rangle$ holds due to the proof of knowledge property of $\langle \mathcal{P}, \mathcal{V} \rangle$ and the binding property of the commitment scheme.

Let $\mathcal{E}$ be the extractor that interacts with a prover $\mathcal{P}$ and if the interaction is accepting then $\mathcal{E}$ outputs a witness. We will use $\mathcal{E}$ to design an extractor $\mathcal{E}'$ for $\langle \mathcal{P}', \mathcal{V}' \rangle$. $\mathcal{E}'$ works in the following way:

- $\mathcal{E}'$ interacts with the prover $\mathcal{P}'$ on behalf of the verifier $\mathcal{V}'$, and interacts with $\mathcal{E}$ on behalf of the prover $\mathcal{P}$.
- If $\mathcal{P}'$ sends a message m' then $\mathcal{E}'$ sends $m = m'$ to $\mathcal{E}$.
- If $\mathcal{E}$ makes any oracle query, then $\mathcal{E}'$ does the same oracle query. Let $\mathcal{E}$ queries for j. Then $\mathcal{E}'$ queries j to π' and receives $\pi^*[j] = \mathsf{Open}(\pi'[j])$. Note that, $\pi^*[j] = \pi[j]$ with high probability. If not, that is, either $\mathcal{P}'$ commits to $\pi[j]$ but opens to a different value $\pi^*[j]$ with non-negligible probability which means $\mathcal{P}'$ breaks the binding property of the commitment scheme, or if $\mathcal{P}'$ does not commit to $\pi[j]$ that leads to verification failed in due to the soundness property of the underlying protocol $\langle \mathcal{P}, \mathcal{V} \rangle$. Therefore $\mathcal{E}'$ sends π^* to $\mathcal{E}$ as oracle response.
- If $\mathcal{E}$ sends a challenge c to $\mathcal{E}'$, then $\mathcal{E}'$ forwards c to $\mathcal{P}'$.
- Finally, $\mathcal{E}'$ outputs whatever $\mathcal{E}$ outputs.

Zero-knowledge of $\langle \mathcal{P}', \mathcal{V}' \rangle$ holds directly from the zero-knowledge property of $\langle \mathcal{P}, \mathcal{V} \rangle$.

Since, $\langle \mathcal{P}, \mathcal{V} \rangle$ has zero-knowledge property, therefore $\exists$ a simulator $\mathcal{S}$ that generates a transcript that is indistinguishable from a real transcript. Using $\mathcal{S}$, we will construct a new simulator $\mathcal{S}'$.

$\mathcal{S}'$ executes $\mathcal{S}$. If τ is the transcript generated by $\mathcal{S}$, and if π_i be the part of τ which is an oracle set by $\mathcal{P}$ at ith round of the protocol. Then $\mathcal{S}'$ commits to π_i and gets π'_i. Finally the simulated transcript, τ', generated by $\mathcal{S}'$ by replacing π_i with π'_i is indistinguishable

from a real execution of $\langle \mathcal{P}', \mathcal{V}' \rangle$. This ensures the zero-knowledge property.

Construction of DPZK version: Let $m_i(\cdot)$ be the function to generate m_i and correspondingly $\pi_i(\cdot)$ for π_i. Let $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_{\mathsf{N}}\}$ be the set of N provers, where at most $t(< n)$ can be corrupted. Parties in $\mathcal{P}$ execute a t-secure MPC for the function $m_i(\cdot)$ and $\pi_i(\cdot)$ to get $\langle m_i \rangle$ and $\langle \pi_i \rangle$, where $\langle x \rangle$ represents t-out of n sharing of x. Provers locally compute $\langle \pi_i' \rangle$ and send $\langle m_i \rangle, \langle \pi_i' \rangle$ to an aggregator A.

A obtains $m_i = \sum \langle m_i \rangle$ and $\pi_i' = \prod \langle \pi_i' \rangle$. It sets π_i' as the oracle and sends m_i to $\mathcal{V}$.

Soundness with Witness Extraction:

Claim: The above construction of the DPZK from $\langle \mathcal{P}', \mathcal{V}' \rangle$ has Soundness with Witness Extraction (SoWE) property if $\langle \mathcal{P}', \mathcal{V}' \rangle$ has proof of knowledge property.

Let $\mathcal{E}$ be the extractor of $\langle \mathcal{P}', \mathcal{V}' \rangle$. Therefore $\mathcal{P}'$ interacts with $\mathcal{E}$ and $\mathcal{E}$ outputs a correct witness if the interaction is accepting. Using $\mathcal{E}$, we will construct $\mathcal{E}_{DP}$ for the distributed version.

$\mathcal{E}_{DP}$ works as follows: If $\mathcal{E}$ sends c to $\mathcal{P}'$, then $\mathcal{E}_{DP}$ sends c to all the provers (broadcasts). If $\mathcal{E}_{DP}$ receives m from the aggregator A, then $\mathcal{E}_{DP}$ forwards m to $\mathcal{E}$. Finally, $\mathcal{E}_{DP}$ outputs $\mathcal{E}$'s output.

Note that $\mathcal{E}$ can extract a correct witness with high probability. Therefore, $\mathcal{E}_{DP}$'s output is also a correct witness with high probability.

Zero-Knowledge:

Since in the DPZK of $\langle \mathcal{P}', \mathcal{V}' \rangle$, the verifier's view does not change, therefore the same simulator works for the distributed prover setting also. Hence the zero-knowledge property is obvious.

Witness Hiding:

Let at the ith round, provers run a secure MPC, Π_{m_i}, to obtain $\langle m_i \rangle$ such that $\sum \langle m_i \rangle = m_i$.

Corresponding to a corrupted set C of t provers, $\mathcal{S}$ does the following:

- $\mathcal{S}$ calls the zero-knowledge simulator, $\mathcal{S}_{ZK}$ and obtains an indistinguishable transcript τ.
- On behalf of the verifier, $\mathcal{S}$ sets the challenge c_i obtained from the transcript τ, and corresponding response m_i.
- If provers run an MPC, Π_{m_i} to obtain $\langle m_i \rangle$, and $\mathcal{S}_{m_i}$ be the corresponding simulator. Consider $\{st_j^i\}_{j \in C}$ be the inputs of the corrupted parties to Π_{m_i}. Then $\mathcal{S}$ executes $\mathcal{S}_{m_i}$ with inputs $\{st_j^i\}_{j \in C}$ and $\langle m_i \rangle$.
- Finally, $\mathcal{S}$ sends $\{\langle m_i \rangle_j\}_{j \notin C}$ to A.

Here note that, the view generated by $\mathcal{S}$ is indistinguishable from a real execution of the protocol. This can be established by the following argument:

$$\begin{aligned} &\mathsf{SView}_1 || \ldots || \mathsf{SView}_i || \mathsf{RView}_{i+1} || \ldots || \mathsf{RView}_r \\ \approx\ &\mathsf{SView}_1 || \ldots || \mathsf{RView}_i || \mathsf{RView}_{i+1} || \ldots || \mathsf{RView}_r \end{aligned}$$

Where SView_i represents the simulated view of the ith round and analogously RView_i represents the real view of the ith round. Now using hybrid argument we get *simulated view* $\approx$ *real view*. □

The compiler preserves the proof size and round complexity of the underlying protocol. The overhead of the computational complexity depends on the oracle size and round complexity of the protocol. If $\langle \mathcal{P}, \mathcal{V} \rangle$ has an oracle of size $|\pi_i|$ in the ith round, then the prover's complexity in $\langle \mathcal{P}', \mathcal{V}' \rangle$ incurs an additional $|\pi_i|$ group exponentiations in the ith round. Similarly, if the verifier in $\langle \mathcal{P}, \mathcal{V} \rangle$ makes t many queries to the oracle π_i, that adds t group exponentiations in the verifier's complexity in $\langle \mathcal{P}', \mathcal{V}' \rangle$.

In Aurora [5], the size of the oracle is $O(N)$, the proof size is $O(\log^2 N)$, and number of rounds is $O(\log N)$. Aurora is an IOP-based proof system where almost all the messages from $\mathcal{P}$ to $\mathcal{V}$ are set as oracles, and $\mathcal{V}$ makes a few oracle-queries complete the verification. We can convert Aurora using our compiler such that it supports DPZK, we call the distributed version to be D-Aurora. The compiled version retains the Oracle size, proof size, and number of rounds. However, the prover time and the verification time increase by $O(N \log N)$ group exponentiations. Since all the oracle construction and validation need $O(N)$ group exponentiations in every round. The distributed version of it needs secure evaluation of a depth one circuit with $O(N)$ multiplication gates in each round.

Similarly, for Ligero [1], Ligero++ [7], our compiler provides protocols with prover time $O(N \log N)$ field multiplication plus $O(N)$ group exponentiations, and verification time $O(N)$ field multiplication plus $O(N)$ group exponentiations. Note that the proof size of the protocols remains the same in the new compiled protocols. We call the distributed versions as D-Ligero and D-Ligero++ respectively.

We provide construction of a new proof system which can be obtained from Ligero/Ligero++ [1, 7] using our compiler and some additional optimization. The rationale behind opting for Ligero style proof system is that it does not have many oracles. Therefore the conversion is less costly. Using the compiler directly on Ligero++ [7] gives 2D version of our construction where the Virgo [51] inner product is replaced by Bulletproofs [12] inner product. We optimize further in our

construction by adjoining additional dimension which aids in better trade-off between proof size and verification time. We use a similar approach to [10, 11], where the witness is viewed as a multi-dimensional matrix. In our construction, we restrict to 3 dimensions since, among these dimensions, only the smallest one contributes to the proof size, while the remaining two aid in better verification time. In our setting, we explored the scenarios by increasing the number of dimensions. Those approaches lead to more complicated and costlier proof-generation and verification protocols without any more improvements. [11] provides linear-time prover with poly-logarithmic verification, but a major drawback of this work is that the soundness error is $O(1)$. Furthermore, [10] obtain linear-time prover by using linear-time encodable codes. For that, they use a linear code provided by [21], whose decoding is conjectured intractable. Due to this property, [10] does not satisfy the proof of knowledge property.

6 Primitives for Graphene

This section provides the main primitives that we use extensively to construct Graphene and its components, such as LinearCheck and QuadraticCheck. The primitives are i) Witness Encoding, ii) Codes and matrices, iii) Commitment of product codewords, iv) Oracle construction, and v) Witness decoding.

Our protocol proves membership in the NP-complete language specified by *rank one constraint system* (R1CS). An R1CS over a field $\mathbb{F}$ is specified by $M \times N$ matrices A, B and C, where the associated language $\mathcal{L}(A, B, C)$ consists of $\mathsf{w} \in \mathbb{F}^N$ such that $A\mathsf{w} \circ B\mathsf{w} = C\mathsf{w}$, where $\circ$ defines element-wise product. Several recent zero knowledge constructions [5, 15, 29], and circuit compilers such as [23] naively support the R1CS formulation. The arithmetic circuit representation can be expressed as an R1CS by introducing a constraint for each multiplication gate. We follow the broad outline of the interactive PCP construction in [1] which includes: (i) extending the witness, denoted as extended witness, to the concatenation of values over all the wires in topological order of an arithmetic circuit representing the statement, (ii) encoding the extended witness via suitable error-correcting code, (iii) checking linear relations on the extended witness, and (iv) checking quadratic relations on the extended witness (for the values concerning the wires going into and coming out from the multiplication gates). Both the linear and quadratic checks require sub-linear (oracle) access to the witness encoding. However, one runs into challenges in converting the IPCP to a non-interactive argument when the witness is distributed across several provers. The naive approach of provers sharing their encoded witnesses with an aggregator, which constructs the oracle breaches privacy among the provers, as these encodings only support *bounded independence*, i.e, they maintain privacy only when a bounded part of the witness is revealed. We overcome this, and several other technical challenges in our construction. The core techniques behind our construction are summarized below:

- We use homomorphic commitment over witness encoding and provide oracle access to the commitment. This facilitates oracle realization through aggregation.
- We design protocols for checking linear and quadratic constraints with oracle access to the commitment.
- We come up with new witness encoding scheme, to facilitate smaller argument size, and reduce communication amongst the provers during distributed proof generation and verification time.

While our use of homomorphic commitments introduces expensive cryptographic operations, we keep their usage strictly sub-linear. In particular, for an argument size of $O(N^{1/c})$, the verifier incurs $O(N^{1-2/c})$ exponentiations. The prover incurs $O(N/\log N)$ exponentiations essentially while setting up the oracle. We report the concrete performance of our new zero-knowledge argument Graphene in Section:9.

6.1 Witness Encoding

We start by describing a randomized encoding of the prover's extended witness $\mathsf{w} \in \mathbb{F}^N$ (henceforth referred as witness), where N denotes the number of wires in the arithmetic circuit representing the NP relation. Let p, m and s be integers such that $N = pms$. We canonically view the witness w as $p \times m \times s$ matrix with entries $\mathsf{w}[i, j, k]$ for $i \in [p]$, $j \in [m]$ and $k \in [s]$. The encoding is specified by an independence parameter b, integers $\ell := s + \mathsf{b}$, $h > 2m$, $n > 2\ell$, and sequences $\boldsymbol{\zeta}, \boldsymbol{\eta}, \boldsymbol{\alpha}$ of distinct points in $\mathbb{F}$ with cardinality ℓ, n, h respectively. We write $\boldsymbol{\zeta} = (\zeta_1, \ldots, \zeta_\ell)$, $\boldsymbol{\eta} = (\eta_1, \ldots, \eta_n)$ and $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_h)$. Next we define the interpolation domain G as $G = \{(\alpha_j, \zeta_k) : j \in [m], k \in [\ell]\}$ and evaluation domain H as $H = \{(\alpha_j, \eta_k) : j \in [h], k \in [n]\}$. Finally, we encode w as follows and denote the below

randomized computation as $\mathsf{U} \leftarrow \mathsf{Enc}(\mathsf{w})$, where $\mathsf{Enc}(\mathsf{w})$ is the random variable denoting the encodings of w:

(i) First we embed w into a $p \times m \times \ell$ matrix $\hat{\mathsf{w}}$ where $\hat{\mathsf{w}}[i,j,k] = \mathsf{w}[i,j,k]$ for $k \leq s$, while the entries $\hat{\mathsf{w}}[i,j,k]$ for $k > s$ are sampled from $\mathbb{F}$ uniformly at random.

(ii) We construct bivariate polynomials $Q^i(x,y)$ with $deg_x(Q) < m$ and $deg_y(Q) < \ell$ such that Q^i interpolates the slice $\hat{\mathsf{w}}[i,\cdot,\cdot]$ on G, i.e, $Q^i(\alpha_j, \zeta_k) = \hat{\mathsf{w}}[i,j,k]$.

(iii) Let U denote the $p \times h \times n$ matrix, where the slice $\mathsf{U}[i,\cdot,\cdot]$ consists of evaluations of Q^i on H, i.e, $\mathsf{U}[i,j,k] = Q^i(\alpha_j, \eta_k)$ for $i \in [p], j \in [h]$ and $k \in [n]$. Then U is a randomized encoding of w.

It is easily seen that $\mathsf{U}[i,\cdot,\cdot] \in \mathsf{RS}_{\boldsymbol{\eta}}[n,\ell] \otimes \mathsf{RS}_{\boldsymbol{\alpha}}[h,m]$. We remark that the above encoding can be computed using $O(N \log N)$ field operations by applying FFT along the rows and columns of the slices.

The encoding Enc satisfies the following *bounded independence* property. The proof is elementary and it will be presented in the full version.

Lemma 6.1 (Bounded Independence)**.** *Let $B \subseteq [n]$ be a set of size b. Let $\mathcal{U}(p,h,b)$ denote the set of $p \times h \times b$ matrices X such that $\mathsf{X}[i,\cdot,k]$ is a codeword in $\mathsf{RS}_{\boldsymbol{\alpha}}[h,m]$ for all $i \in [p], k \in [\mathsf{b}]$. Then for any $p \times m \times s$ matrix w, the random variable $\mathsf{U}_B := \{\mathsf{U}[\cdot,\cdot,B] : \mathsf{U} \leftarrow_{\$} \mathsf{Enc}(\mathsf{w})\}$ is distributed uniformly on $\mathcal{U}(p,h,b)$.*

Proof. It is sufficient to prove that an arbitrary element from $\mathcal{U}(p,h,b)$ is in U_B suffices the lemma.

Let $U \in \mathcal{U}(p,h,b)$. Set U such that $\mathsf{U}[\cdot,\cdot,k] = U[\cdot,\cdot,k]$ for all $k \in B$.

For $i = 1$ to p,

- Construct a bivariate polynomial $Q^i(x,y)$ such that $Q^i(\alpha_j, \zeta_k) = \mathsf{w}[i,j,k]$, $\forall j \in [m], k \in [s]$, and $Q^i(\alpha_j, \eta_k) = \mathsf{U}[i,j,k]$, $\forall j \in [m], k \in B$.
- $Q^i(x,y)$ is a polynomial such that $deg_x(Q^i) < m$ and $deg_y(Q^i) < \ell$, where $\ell = s + b$.
- Since, $\mathsf{U}[i,\cdot,k] \in \mathsf{RS}_{\boldsymbol{\alpha}}[h,m]$, therefore $Q^i(\alpha_j, \eta_k) = \mathsf{U}[i,j,k]$, forall $j \in [h], k \in B$.
- Set $\mathsf{U}[i,j,k] = Q^i(\alpha_j, \eta_k)$, forall $j \in [m], k \in [n] \setminus B$.

Note that, $\mathsf{Dec}(\mathsf{U}) = \mathsf{w}$, therefore $\mathsf{U}[\cdot,\cdot,B] \in \mathsf{U}_B$. This completes the proof. □

6.2 Codes and Matrices

For code $\mathsf{RS}_{\boldsymbol{\eta}}[n,\ell]$, let $\Lambda_{n,\ell}$ denote the $n \times \ell$ matrix for the linear transformation that maps a vector $x \in \mathbb{F}^\ell$ to the unique codeword y in $\mathsf{RS}_{\boldsymbol{\eta}}[n,\ell]$ such that $y_i = x_i$ for $i \in [\ell]$. For codes $\mathsf{RS}_{\boldsymbol{\alpha}}[h,m]$, $\mathsf{RS}_{\boldsymbol{\eta}}[n, s+\ell-1]$, $\mathsf{RS}_{\boldsymbol{\eta}}[n, 2\ell-1]$, and $\mathsf{RS}_{\boldsymbol{\alpha}}[h, 2m-1]$, let $\Lambda_{h,m}, \Lambda_{n,s+\ell-1}, \Lambda_{n,2\ell-1}$ and $\Lambda_{h,2m-1}$ be similar matrices. We denote the corresponding parity-check matrices as $\mathcal{H}_{n,\ell}, \mathcal{H}_{h,m}, \mathcal{H}_{n,s+\ell-1}, \mathcal{H}_{n,2\ell-1}$.

We notate the set of three dimensional $p \times h \times n$ matrices as $\mathcal{M}_{p,h,n}$ and the set of two dimensional $h \times n$ matrices as $\mathcal{M}_{h,n}$. We assume standard distance metrics on the sets $\mathcal{M}_{p,h,n}$ and $\mathcal{M}_{h,n}$. Let $\mathcal{W}_1$ denote the set of matrices U in $\mathcal{M}_{p,h,n}$ such that the n-length vector $U[i,j,\cdot]$ is a codeword in $\mathsf{RS}_{\boldsymbol{\eta}}[n,\ell]$ for all i,j. Similarly let $\mathcal{W}_2$ denote the set of matrices U such that the h-length vector $U[i,\cdot,k]$ is a codeword in $\mathsf{RS}_{\boldsymbol{\alpha}}[h,m]$ for all i,k. Let $\mathcal{W} = \mathcal{W}_1 \cap \mathcal{W}_2$. $\mathcal{W}$ denotes the set of *well-formed* encodings and consists of U such that each slice $U[i,\cdot,\cdot] \in \mathsf{RS}_{\boldsymbol{\eta}}[n,\ell] \otimes \mathsf{RS}_{\boldsymbol{\alpha}}[h,m]$.

6.3 Commitment of product codewords

We now discuss the commitment scheme for matrices in $\mathsf{RS}_{\boldsymbol{\eta}}[n,\ell] \otimes \mathsf{RS}_{\boldsymbol{\alpha}}[h,m]$. Similar scheme works for other product codes also. A matrix U in the above product code is completely determined by the sub-matrix $\overline{U} = \{U[j,k] : j \in [m], k \in [\ell]\}$ and can be expressed as $U = \Lambda_{h,m} \overline{U} \Lambda^T_{n,\ell}$. A commitment to U is defined as a vector of commitments $\boldsymbol{c} = (c_1, \ldots, c_\ell)$ where $c_k = \mathsf{Com}(\overline{U}[\cdot,k])$ is the vector commitment for k^{th} column of $\overline{U}$ for $k \in [\ell]$. We use the notation $\boldsymbol{c} = (c_1, \ldots, c_\ell) \leftarrow \mathsf{pcCom}(U)$. Given $\boldsymbol{c}$, commitment to kth column of U for $k > \ell$ can be computed as $c_k = \prod_{a \in [\ell]} (c_a)^{\Lambda^T_{n,\ell}[a,k]}$, thanks to the homomorphicity.

6.4 Oracle Construction

Unlike prior IOP constructions such as [1, 5], we additionally obtain a homomorphic commitment on the encoded witness $\mathsf{Enc}(\mathsf{w})$ and provide oracle access to the commitment. For all $(i,k) \in [p] \times [n]$, we compute the commitment $c_{ik} = \mathsf{Com}(V_{ik})$ for the vector $V_{ik} = (\mathsf{U}[i,1,k], \ldots, \mathsf{U}[i,m,k]) \in \mathbb{F}^m$. Note that we commit to the m-length vector (and not the h-length one) to save on the number of exponentiations. Finally we define $p \times n$ matrix $\mathcal{O}$ as $\mathcal{O}[i,k] = c_{ik}$. We write this as $\mathcal{O} \leftarrow \mathsf{oCom}(\mathsf{U})$. We provide oracle access to $\mathcal{O}$ where for

a query $Q \subseteq [n]$, the oracle responds with columns $\mathcal{O}[\cdot, k]$ for $k \in Q$. Note that i^{th} row of the matrix $\mathcal{O}$ commits to the i^{th} slice of U. This is different from commitment of a product codeword where only ℓ columns are committed.

6.5 Witness Decoding

We describe a decoding procedure Dec for obtaining a witness w from an encoding U. Let $\mathsf{U} \in \mathcal{W}$ be a well-formed encoding. Such an encoding can be decoded slice by slice, i.e, for each $i \in [p]$, we interpolate bivariate polynomial $Q^i \in \mathbb{F}[x, y]$ with $deg_x(Q^i) < m$ and $deg_y(Q^i) < \ell$ such that Q^i interpolates $\mathsf{U}[i, \cdot, \cdot]$ on evaluation domain $H = \{(\alpha_j, \eta_k) : j \in [h], k \in [n]\}$. This can be accomplished using standard algorithms. The decoded witness w is then given by $\mathsf{w}[i, j, k] = Q^i(\alpha_j, \zeta_k)$. We extend the above decoding procedure to recover from slightly malformed encodings. Let $\mathsf{U}^* \in \mathcal{M}_{p,h,n}$ be such that $d_1(\mathsf{U}^*, \mathcal{W}_1) < e_1 < (n-\ell)/2$ and $d_2(\mathsf{U}^*, \mathcal{W}_2) < e_2 < (h-m)/2$. In this case, from the distance property of the codes $\mathsf{RS}_{\boldsymbol{\eta}}[n, \ell]$ and $\mathsf{RS}_{\boldsymbol{\alpha}}[h, m]$ it follows that there is at most one $\mathsf{U} \in \mathcal{W}$ such that $\Delta_1(\mathsf{U}^*, \mathsf{U}) < e_1$ and $\Delta_2(\mathsf{U}^*, \mathsf{U}) < e_2$. Such a U may be efficiently "recovered" from U^* using algorithms for Reed-Solomon decoding (c.f. [41]). We then define $\mathsf{Dec}(\mathsf{U}^*)$ as $\mathsf{Dec}(\mathsf{U})$.

7 Graphene

7.1 Linear Check

In this section, we will give an overview of the working of the linear check protocol that is used to construct the Graphene. In the linear check protocol a prover proves knowledge of a witness w satisfying a linear constraint of the form $A\mathsf{w} = b$ for some *public* $A \in \mathbb{F}^{M \times N}$ and $b \in \mathbb{F}^M$. The verifier needs to check that $A\mathsf{w} = b$ holds, this check can be probabilistically reduced to check $r^T A\mathsf{w} = r^T b$, for a randomly chosen r. To do that, the verifier $\mathcal{V}$ picks a random $r \leftarrow_\$ \mathbb{F}^M$ and sends to the prover $\mathcal{P}$ as a challenge. Now $\mathcal{P}$ needs to show that $\langle r^T A, \mathsf{w} \rangle = r^T b$. Lets call $R = r^T A$. Further, this check is reduced to check inner product arguments. To facilitate such checks, $\mathcal{P}$ encodes and commits to its witness w, using the encoding and commitment described in Section 6.1, and Section 6.3 respectively. $\mathcal{P}$ constructs an oracle as described in Section 6.4. As $\mathcal{P}$ receives r from $\mathcal{V}$, both $\mathcal{P}$ and $\mathcal{V}$ computes R, and view it as a cube of dimension $p \times m \times s$. Then interpolate and obtain bivariate polynomials $R^i(x, y)$ for $i \in [p]$ with $deg_x(R^i) < m$ and $deg_y(R^i) < s$ satisfying $R^i(\alpha_j, \zeta_k) = R[i, j, k]$. Let Q^i, $i \in [p]$ denote the polynomials used in interpolating (and encoding) witness w. Then $\mathsf{w}[i, j, k] = Q^i(\alpha_j, \zeta_k)$. Therefore the check $\langle R, \mathsf{w} \rangle = r^T b$ reduces to $\sum_{i,j,k} R^i(\alpha_j, \zeta_k) Q^i(\alpha_j, \zeta_k) = r^T b$ where i, j and k run over indices in $[p], [m]$ and $[s]$ respectively. $\mathcal{P}$ computes $p_j(y) = \sum_{i \in [p]} R^i(\alpha_j, y) \cdot Q^i(\alpha_j, y)$, for $j \in [h]$. Here all these polynomials are of degree less than $s + \ell - 1$. To reduce the communication instead of sending all $p_j(y)$ polynomials, $\mathcal{P}$ constructs an intermediate matrix P of dimension $h \times n$ such that $P[j, k] = p_j(\eta_k)$ for $j \in [h], k \in [n]$. Note that P is a product codeword from $\mathsf{RS}_{\boldsymbol{\eta}}[n, s + \ell - 1] \otimes \mathsf{RS}_{\boldsymbol{\alpha}}[h, 2m - 1]$ and fully determined by sub-matrix $\overline{P}$ consisting of the first $2m - 1$ rows and the first $s + \ell - 1$ columns of P.

Reduction to inner product argument:Now $\mathcal{P}$ commits to the matrix P using commitment for product codeword, described in Section 6.3 that is, $(c_1, \ldots, c_{s+\ell-1}) \leftarrow \mathsf{pcCom}(P)$, and sends $(c_1, \ldots, c_{s+\ell-1})$ to $\mathcal{V}$, which they use later in the inner product argument. The check $\langle R, \mathsf{w} \rangle = r^T b$ can be viewed as a following inner product $\langle *, \overline{P}\varphi \rangle = r^T b$, where $*$ and φ are public vectors and so the commitment for $\overline{P}\varphi$ can be computed given the commitment of $\overline{P}$. Let Φ be a $s \times (s + \ell - 1)$ matrix such that $[p_j(\zeta_1), \ldots, p_j(\zeta_s)]^T = \Phi[p_j(\eta_1), \ldots, p_j(\eta_{s+\ell-1})]^T$ for $j \in [h]$. We have:

$$\big[\sum_{j \in [m]} p_j(\zeta_1), \ldots, \sum_{j \in [m]} p_j(\zeta_s) \big]^T = \Phi \overline{P}^T [1^m || 0^{m-1}]^T$$

So the check reduces to $\sum_{k \in [s], j \in [m]} p_j(\zeta_k) = r^T b$ reduces to

$$\langle [\sum_{j \in [m]} p_j(\zeta_1), \ldots, \sum_{j \in [m]} p_j(\zeta_s)]^T, [1^s]^T \rangle = r^T b$$
$$\Rightarrow \langle [1^m || 0^{m-1}]^T, \overline{P} \times \Phi^T \times [1^s]^T \rangle = r^T b$$

Here, $* = [1^m || 0^{m-1}]^T$ and $\varphi = \Phi^T \times [1^s]^T$ are public vectors. Given commitment of P, $c_1, \ldots, c_{s+\ell-1}$, the commitment to $\overline{P}\varphi$ can be computed as $\mathsf{cm} = \prod_{k=1}^{s+\ell-1} (c_k)^{\varphi_k}$. In the protocol, the prover initially commits to a random $P_0 \in \mathbb{F}^{2m-1}$ subject to $\langle 1^m || 0^{m-1}, P_0 \rangle = 0$, and uses $\beta P_0 + \overline{P}\varphi$ as witness in the inner product protocol. Here $\beta \leftarrow_\$ \mathbb{F} \backslash \{0\}$ is randomly chosen by the verifier. This randomization precludes the need for the inner-product argument to be zero-knowledge, and as we will later see, helps reduce interaction among provers in its distributed variant.

Consistency of oracle π and P via U: The verifier additionally needs to determine if the committed P and the oracle π are consistent or not. The verifier proceeds

to check the consistency at randomly sampled t positions given by $\{(j_u, k_u) : u \in [t]\}$ from $[h] \times [n]$ for a $t = O(\lambda)$. It queries the oracle for the columns $\pi[\cdot, k_u]$ and the prover for vectors $X_u = \mathsf{U}[\cdot, j_u, k_u]$ for $u \in [t]$. For $u \in [t]$, let $\mathbf{1}_{j_u}$ denote the unit vector in $\mathbb{F}^h$ with 1 in the position j_u. The prover and the verifier run inner-product arguments to establish the following:

1. $\langle \mathbf{1}_{j_u}, P[\cdot, k_u] \rangle = \sum_{i \in [p]} R^i(\alpha_{j_u}, \eta_{k_u}) X_u[i]$ for $u \in [t]$, with c_{k_u} (which can be computed from $c_1, \ldots, c_{s+\ell-1}$, as in Section 6.3) as the commitment of $P[\cdot, k_u]$.
2. $\langle \mathbf{1}_{j_u}, \mathsf{U}[i, \cdot, k_u] \rangle = X_u[i]$ with $\pi[i, k_u]$ as the commitment for $\mathsf{U}[i, \cdot, k_u]$. A minor technicality arises since the commitment is for $V_{ik_u} = [\mathsf{U}[i, 1, k_u], \ldots, \mathsf{U}[i, m, k_u]]$ and not for $\mathsf{U}[i, \cdot, k_u]$. Since $\mathsf{U}[i, \cdot, k_u] = \Lambda_{h,m} V_{ik_u}$, the above inner-product reduces to $\langle \mathbf{1}_{j_u}^T \Lambda_{h,m}, V_{ik_u} \rangle = X_u[i]$.

The checks for each $u \in [t]$ can be aggregated, leading to one inner-product check for each $u \in [t]$.

Proximity Check for Oracle: This check forces a prover to commit to an encoding which is "close" to well-formed encoding $\mathcal{W}$. To check proximity, the verifier initially sends a vector $\rho \leftarrow_\$ \mathbb{F}^p$ and asks the prover to send commitments $\tilde{\boldsymbol{c}} = (\tilde{c}_1, \ldots, \tilde{c}_\ell)$ to $\tilde{\mathsf{U}} = \sum_{i \in [p]} \rho_i \mathsf{U}[i, \cdot, \cdot]$ computed as $\tilde{c}_k = \prod_{i \in [p]} (\pi[i, k])^{\rho_i} \ \forall k \in [\ell]$. It then checks:

$$\prod_{a=1}^{\ell} (\tilde{c}_a)^{\Lambda^T_{n,\ell}[a, k_u]} = \prod_{i=1}^{p} (\pi[i, k_u])^{\rho_i} \text{ for } u \in [t]. \quad (1)$$

It can be seen that for an honest computation, both the commitments open to the vector $\sum_{i \in [p]} \rho_i V_{iu}$ where $V_{iu} = (\mathsf{U}[i, 1, k_u], \ldots, \mathsf{U}[i, m, k_u])$.

7.2 Quadratic Check

Here we will describe the quadratic check protocol, which aids in validating the honest evaluation of the circuit's multiplication gates. $\mathcal{P}$ proves the knowledge of w_x, w_y and w_z in $\mathbb{F}^N$ satisfying $\mathsf{w}_x \circ \mathsf{w}_y = \mathsf{w}_z$. This protocol is similar to the linear check protocol, described in Section 7.1. Here also it has 3 stages, i) Reduction to inner product argument, ii) Consistency of the oracle and the intermediate matrix P, and iii) Proximity check. Here $\mathcal{V}$ checks if the polynomials $Q^i = Q^i_x \cdot Q^i_y - Q^i_z$ interpolate to $\mathbf{0}^{m \times s}$ on the set $\{(\alpha_j, \zeta_k) : j \in [m], k \in [s]\}$ for all $i \in [p]$. In three simple steps, we derive a simple probabilistic check below compressing in all the three dimensions. First, the above check reduces to checking that the polynomial $F := \sum_{i \in [p]} r_i Q^i$ interpolates $\mathbf{0}^{m \times s}$ on the same set for a randomly sampled $r \in \mathbb{F}^p$. Second, this further reduces to checking $p(\cdot) := \sum_{j \in [m]} \gamma_j F(\alpha_j, \cdot)$ interpolates $\mathbf{0}^s$ on $\overline{\boldsymbol{\zeta}}$ for randomly sampled $\gamma = (\gamma_1, \ldots, \gamma_m) \in \mathbb{F}^m$. Third, this further reduces to checking $\langle \tau, p(\overline{\boldsymbol{\zeta}}) \rangle = 0$ for a random $\tau \in \mathbb{F}^s$. Similar to the linear check protocol, $\mathcal{P}$ constructs the intermediate matrix P, which is in the product code $\mathsf{RS}_{\boldsymbol{\eta}}[n, 2\ell - 1] \otimes \mathsf{RS}_{\boldsymbol{\alpha}}[h, 2m - 1]$, and commits to it similarly. Let $(c_1, \ldots, c_{2\ell-1}) \leftarrow \mathsf{pcCom}(P)$. Similar to the linear check, quadratic check reduces to $\langle [\gamma || 0^{m-1}]^T, z \rangle = 0$, where $z = \beta P_0 + \overline{P}\varphi$ and P_0 is a random vector such that $P_0[j] = 0$ for $j \in [m]$, and $\varphi = \Phi \times \tau$. The differences are: (a) $p(\cdot) = \sum_{j \in [m]} \gamma_j F_j(\alpha_j, \cdot)$ is a polynomial of degree $< 2\ell - 1$, (b) Φ is a $s \times 2\ell - 1$ matrix that takes $p(\overline{\boldsymbol{\eta}})$ to $p(\overline{\boldsymbol{\zeta}})$, where $p(\overline{\boldsymbol{\eta}}) = [p(\eta_1), \ldots, p(\eta_{2\ell-1})]$ and $p(\overline{\boldsymbol{\zeta}}) = [p(\zeta_1), \ldots, p(\zeta_s)]$. Commitment c_0 of P_0 is sent to the verifier a-priori. Therefore, commitment to z is $\mathsf{cm} = c_0^\beta \prod_{k=1}^{2\ell-1} (c_k)^{\varphi_k}$.

Consistency of the oracles and P: Similar to linear check, the verifier checks the consistency at randomly sampled positions $Q = \{(j_u, k_u) : u \in [t]\} \subset [h] \times [n]$. It queries the oracles for the columns $\pi_a[\cdot, k_u]$ for $a \in \{x, y, z\}$ and $u \in [t]$. Then it queries the prover for vectors $X_u = \mathsf{U}_x[\cdot, j_u, k_u], Y_u = \mathsf{U}_y[\cdot, j_u, k_u], Z_u = \mathsf{U}_z[\cdot, j_u, k_u]$ for $u \in [t]$. Let $\mathbf{1}_{j_u}$ denote the unit vector in $\mathbb{F}^h$ with 1 in the j_u^{th} position. The prover and verifier run inner-product arguments to establish:

1. $\langle \mathbf{1}_{j_u}, P[\cdot, k_u] \rangle = \sum_{i \in [p]} r_i [X_u[i] \cdot Y_u[i] - Z_u[i]]$ for $u \in [t]$ with c_{k_u} which can be computed from $c_1, \ldots, c_{2\ell-1}$, as in Section 6.3) as the commitment of $P[\cdot, k_u]$.
2. $\langle \mathbf{1}_{j_u}, \mathsf{U}_x[i, \cdot, k_u] \rangle = X_u[i]$, $\langle \mathbf{1}_{j_u}, \mathsf{U}_y[i, \cdot, k_u] \rangle = Y_u[i]$ and $\langle \mathbf{1}_{j_u}, \mathsf{U}_z[i, \cdot, k_u] \rangle = Z_u[i]$. These inner-products check the consistency of the vectors sent by the prover with the respective oracles. These can be verified in a similar manner to that in the linear check protocol.

Proximity Check for Oracle: We combine the proximity check for the three encodings $\mathsf{U}_x, \mathsf{U}_y$ and U_z. The verifier sends a vector $\rho \leftarrow_\$ \mathbb{F}^{3p}$ and asks the prover to commit to the matrix $\tilde{\mathsf{U}} = \sum_{i=1}^{p} [\rho_i \mathsf{U}_x[i, \cdot, \cdot] + \rho_{p+i} \mathsf{U}_y[i, \cdot, \cdot] + \rho_{2p+i} \mathsf{U}_z[i, \cdot, \cdot]]$ by sending commitments $\tilde{\boldsymbol{c}} = (\tilde{c}_1, \ldots, \tilde{c}_\ell)$, which are computed as $\tilde{c}_k = \prod_{i=1}^{p} (\pi_x[i, k])^{\rho_i} \cdot (\pi_y[i, k])^{\rho_{p+i}} \cdot (\pi_z[i, k])^{\rho_{2p+i}} \ \forall k \in [\ell]$.

Thereafter, for $u \in [t]$, the verifier checks:

$$\prod_{a\in[\ell]} \tilde{c}_a^{\Lambda^T_{n,\ell}[a,k_u]} = \prod_{i=1}^{p} (\pi_x[i,k_u])^{\rho_i} (\pi_y[i,k_u])^{\rho_{p+i}} (\pi_z[i,k_u])^{\rho_{2p+i}} \tag{2}$$

7.3 Graphene

We use the linear and quadratic check protocols from Section 7.1 and Section 7.2 respectively to describe an efficient protocol for rank one constraint system (R1CS). Let A, B and C be $M \times N$ matrices. We prove existence of $\mathsf{w} \in \mathbb{F}^N$ satisfying $A\mathsf{w} \circ B\mathsf{w} = C\mathsf{w}$ by showing existence of $\mathsf{w}_x, \mathsf{w}_y, \mathsf{w}_z$ and $\mathsf{w} \in \mathbb{F}^N$ satisfying the linear relations $A\mathsf{w} = \mathsf{w}_x$, $B\mathsf{w} = \mathsf{w}_y$ and $C\mathsf{w} = \mathsf{w}_z$ and a quadratic relation $\mathsf{w}_x \circ \mathsf{w}_y = \mathsf{w}_z$. We probabilistically reduce the three linear relations to the linear relation:

$$\left[\gamma_x I \,|\gamma_y I \,|\gamma_z I \,| - (\gamma_x A + \gamma_y B + \gamma_z C)\right] \begin{bmatrix} \mathsf{w}_x \\ \mathsf{w}_y \\ \mathsf{w}_z \\ \mathsf{w} \end{bmatrix} = \mathbf{0} \tag{3}$$

for $\gamma_x, \gamma_y, \gamma_z \leftarrow\!\!\$\ \mathbb{F}$. As before, we view $\mathsf{w}_x, \mathsf{w}_y, \mathsf{w}_z$ and w as $p \times m \times s$ matrices. Let $\overline{\mathsf{w}}$ denote the $4p \times m \times s$ matrix formed by stacking $\mathsf{w}_x, \mathsf{w}_y, \mathsf{w}_z$ and w along "slices". The encoding $\mathsf{U} \leftarrow \mathsf{Enc}(\overline{\mathsf{w}})$ and commitment to the encoding $\mathcal{O} \leftarrow \mathsf{Com}(\mathsf{U})$ are computed as in Sections 6.1 and 6.4. Note that $\mathcal{O}$ is a $4p \times n$ matrix. Let D be the $M \times 4N$ sized matrix given in Equation 3. For the R1CS instance (A, B, C), $\mathcal{P}$ and $\mathcal{V}$ run Linear Check protocol 7.1 for $D\overline{\mathsf{w}} = \mathbf{0}$ and run Quadratic Check protocol for $\mathsf{w}_x \circ \mathsf{w}_y = \mathsf{w}_z$. Here the following lemmas ensure the soundness and zero-knowledge property of Graphene. Details of proof of the protocol is given in Appendix B.

Lemma 7.1 (Soundness)**.** *For all polynomially bounded provers P^* and all $\pi \in \mathbb{G}^{4p\times n}$, there exists an expected polynomial time extractor $\mathcal{E}$ with rewinding access to the transcript oracle $\mathsf{tr} = \langle P^*, \mathcal{V}^\pi \rangle$ such that either $\mathcal{E}$ breaks the commitment binding, or it outputs a witness with overwhelming probability whenever P^* succeeds, i.e.,*

$$\Pr\left[\begin{array}{c|c} [\mathsf{U}_x||\mathsf{U}_y||\mathsf{U}_z||\mathsf{U}] = \mathsf{Open}(\pi) \wedge & \sigma \leftarrow \mathsf{Gen}(1^\lambda) \\ \mathsf{w}_z = \mathsf{w}_x \circ \mathsf{w}_y \wedge \mathsf{w}_x = A\mathsf{w} & [\mathsf{U}_x||\mathsf{U}_y||\mathsf{U}_z||\mathsf{U}] \leftarrow \mathcal{E}^{\mathsf{tr}}(\sigma) \\ \mathsf{w}_y = B\mathsf{w} \wedge \mathsf{w}_z = C\mathsf{w} & \mathsf{w}_a = \mathsf{Dec}(\mathsf{U}_a), a \in \{x,y,z\} \\ & \mathsf{w} = \mathsf{Dec}(\mathsf{U}) \end{array}\right] \geq \epsilon(P^*) - \kappa_{\mathrm{qd}}(\lambda) - \kappa_{\mathrm{lc}}(\lambda)$$

Where $\kappa_{\mathrm{lc}}(\lambda)$ and $\kappa_{\mathrm{qd}}(\lambda)$ are the negligible soundness error of LinearCheck *and* QuadraticCheck *respectively. And $\epsilon(P^*)$ is the success probability of the prover.*

Lemma 7.2 (Zero-knowledge)**.** *There exists a simulator $\mathcal{S}$ that outputs a perfectly indistinguishable extended view of the verifier in honest execution of the protocol* GrapheneR1CS *for $t \leq b$.*

8 DP-Graphene: Distributed Prover Variant

We now describe distributed protocol to produce a Graphene proof for a statement, when the witness is shared between N provers $\mathcal{P}_1, \ldots, \mathcal{P}_\mathsf{N}$. For $\xi \in [\mathsf{N}]$, let $\langle \mathsf{w} \rangle^\xi$ denote the prover $\mathcal{P}_\xi$'s share of the witness w. We assume that the sharing is additive, i.e, $\sum_{\xi\in[\mathsf{N}]} \langle \mathsf{w} \rangle^\xi = \mathsf{w}$. Recall from Section 4, that there is an algorithm A which aggregates the messages received from provers $\mathcal{P}_1, \ldots, \mathcal{P}_\mathsf{N}$ and constructs the message to be sent to $\mathcal{V}$. We assume one of the provers executes A. We specify the algorithm A implicitly by describing the construction of message to $\mathcal{V}$ from the provers' messages for each round. We first discuss a protocol secure against semi-honest provers and then briefly discuss how to ensure the privacy of the honest provers when the corrupt provers are malicious (barring the one who runs A).

8.1 Distributed Oracle Setup:

In distributed setting, each prover $\mathcal{P}_\xi$ encodes her share $\langle \mathsf{w} \rangle^\xi$ as $\langle \mathsf{U} \rangle^\xi = \mathsf{Enc}(\langle \mathsf{w} \rangle^\xi)$ and computes the commitment $\langle \mathcal{O} \rangle^\xi = \mathsf{oCom}(\langle \mathsf{U} \rangle^\xi)$. The provers then share $\langle \mathcal{O} \rangle^\xi$ with the aggregator A which sets the oracle π as $\pi := \mathsf{Combine}(\langle \mathcal{O} \rangle^\xi)$, where Combine simply multiplies the corresponding commitments. The homomorphism and the fact that the witnesses are additively shared ensure that π contains commitment to the witness.

8.2 Distributed Linear Check:

The messages sent by the prover to the verifier in the linear check protocol include:

- Commitments $\tilde{\boldsymbol{c}} = (\tilde{c}_1, \ldots, \tilde{c}_\ell)$ to the matrix $\tilde{\mathsf{U}} = \sum_{i\in[p]} \rho_i \mathsf{U}[i,\cdot,\cdot]$ for verifier's challenge $\rho \leftarrow\!\!\$\ \mathbb{F}^p$.
- Commitments $c_0, \ldots, c_{s+\ell-1}$ where c_0 is a commitment to random vector $P_0 \in \mathbb{F}^{2m-1}$ satisfying $\langle 1^m||0^{m-1}, P_0 \rangle = 0$ and $c_1, \ldots, c_{s+\ell-1}$ are commitments to the $h \times n$ matrix P.
- The vectors $X_u = \mathsf{U}[\cdot, j_u, k_u]$ for $u \in [t]$, for verifier's query $Q = \{(j_u, k_u) : u \in [t]\}$.

We see that given verifier's challenges, each of the messages is linear function of the encoding (which itself is a linear function of the witness). Hence, the provers compute the respective messages on their shares, which can be trivially combined by A. In addition to above messages, we also want A to receive witnesses to the inner-product protocols namely, the vectors $\overline{P}[\cdot, k_u]$, $W_u = \sum_{i\in[p]} \delta_i \mathsf{U}[i, \cdot, k_u]$ for $u \in [t]$, $z = \beta P_0 + \overline{P}\varphi$ and the randomness used to commit the vectors. Each of these can again be obtained by combining the respective shares. Note that the share $\langle z\rangle^\xi$ leaks $r^T A\langle \mathsf{w}\rangle^\xi = \langle 1^m||0^{m-1}, \langle z\rangle^\xi\rangle$, which is non-trivial knowledge about an individual witness share. Thus provers use a random share $\langle 0^{2m-1}\rangle^\xi$ to randomize their share of z, and send $\langle z\rangle^\xi = \beta\langle P_0\rangle^\xi + \langle\overline{P}\rangle^\xi\varphi + \langle 0^{2m-1}\rangle^\xi$. We provide the complete distributed linear protocol in Figure 6.

8.3 Distributed Quadratic Check:

Here the distributed variant requires an additional interaction among the provers. Recall that in response to $\mathcal{V}$'s challenge $r \in \mathbb{F}^p$, the provers need to compute P as:

$$\begin{aligned} P[j,k] &= \sum_{i\in[p]} r_i\big(Q_x^i(\alpha_j,\eta_k).Q_y^i(\alpha_j,\eta_k) - Q_z^i(\alpha_j,\eta_k)\big) \\ &= \sum_{i\in[p]} r_i(\mathsf{U}_x[i,j,k].\mathsf{U}_y[i,j,k] - \mathsf{U}_z[i,j,k]) \end{aligned}$$

where U_x, U_y and U_z are the encodings of the witness vectors w_x, w_y and w_z respectively. Since the matrix P above is completely determined by it's first $2m-1$ rows and first $2\ell-1$ columns, the provers need to obtain the shares $\langle \mathsf{U}_x[i,j,k].\mathsf{U}_y[i,j,k]\rangle^\xi$ for $i \in [p], j \in [2m-1], k \in [2\ell-1]$. From the shares of witness $\langle \mathsf{w}_x\rangle^\xi, \langle \mathsf{w}_y\rangle^\xi$ and $\langle \mathsf{w}_z\rangle^\xi$ the provers can locally compute shares $\langle \mathsf{U}_x\rangle^\xi, \langle \mathsf{U}_y\rangle^\xi, \langle \mathsf{U}_z\rangle^\xi$ of $\mathsf{U}_x, \mathsf{U}_y$ and U_z. Now, the provers call $\mathcal{F}_{\mathsf{Mult}}$ on inputs $\langle \mathsf{U}_x\rangle^\xi$, $\langle \mathsf{U}_y\rangle^\xi$ and obtain $\langle \mathsf{U}_x[i,j,k].\mathsf{U}_y[i,j,k]\rangle^\xi$. We can instantiate $\mathcal{F}_{\mathsf{Mult}}$ with any state-of-the-art dishonest majority protocol for arithmetic circuits. This requires evaluation of $p.(2m-1).(2\ell-1) \approx 4N$ multiplication gates, and depth 1, to obtain the shares $\langle \mathsf{U}_x[i,j,k].\mathsf{U}_y[i,j,k]\rangle^\xi$ for $i \in [p], j \in [2m-1], k \in [2\ell-1]$. Thereafter, each prover obtains a share of matrix P, and the remaining protocol proceeds as the distributed linear check protocol. The complete protocol for distributed quadratic check appears in Figure 7 , we discuss how to optimize MPC overhead when the size of the shared circuit (see Section 2.1) is small.

We show that the messages received by the aggregator in both the distributed protocols can be efficiently simulated, independent of their views in the preceding MPC protocols, provided the parties follow the protocol and jointly possess a valid witness. We present the proofs of (i) Soundness with Witness Extraction (SoWE), (ii) Zero-Knowledge (Zk), and (iii) Witness-Hiding with Collusion (WHwC), which is the same as the plain Witness Hiding in our setting, in Appendix C.

9 Performance Evaluation

In Section 7.3, we present protocol for an R1CS instance. Here we summarize several performance parameters attained by our protocol for R1CS. Let $N = pms$, $\ell = s+t$ and h be as in the previous sections. Let $\mathsf{C}_{\mathbb{F}}$ denote the time taken for a field operation, $\mathsf{C}_{\mathsf{FFT}}(x)$ denote the time to compute FFT of a x length vector, $\mathsf{C}_{\mathsf{MXP}}(x)$ denote the time taken for a multi-exponentiation of length x, and $\mathsf{C}_{\mathsf{EXP}}$ denote the time required for an exponentiation. Let $\mathsf{B}_{\mathbb{F}}$ and $\mathsf{B}_{\mathbb{G}}$ denote the number of bits required to represent an element of $\mathbb{F}$ and $\mathbb{G}$ respectively. Our protocol Graphene achieves following efficiency parameters:

- *Number of rounds*: $r_{\mathsf{zk}} = O(\log N)$.
- *Argument size*: $c_{\mathsf{zk}} = 4pt\mathsf{B}_{\mathbb{F}} + (4pt + 8t\log m + 4\ell + s + 8t + 4)\mathsf{B}_{\mathbb{G}}$.
- *Prover complexity*: $t_{\mathcal{P}} = 4p((m+n)\mathsf{C}_{\mathsf{FFT}}(m) + m\mathsf{C}_{\mathsf{FFT}}(l) + n\mathsf{C}_{\mathsf{FFT}}(h) + 4p\ell\mathsf{C}_{\mathsf{MXP}}(m) + (n-\ell)\mathsf{C}_{\mathsf{MXP}}(\min(\ell,m)) + (s+3\ell)\mathsf{C}_{\mathsf{MXP}}(2m) + (48tm + 32m)\mathsf{C}_{\mathsf{EXP}}$
- Verifier complexity: $t_{\mathcal{V}} = O(N)\mathsf{C}_{\mathbb{F}} + 4pm\mathsf{C}_{\mathsf{FFT}}(s) + (2t+2)\mathsf{C}_{\mathsf{MXP}}(2m) + 2t\mathsf{C}_{\mathsf{MXP}}(m) + t\mathsf{C}_{\mathsf{MXP}}(4p+\ell)$
- Soundness error $\kappa_{\mathrm{gr}} = (1-e/n)^t + 2\big(2m/h + (1 - 2m/h)(2\ell+e)/n\big)^t$.

In Appendix F.1, we give similar expressions for Ligero and Bulletproofs protocols.

For $c \geq 2$, setting $p = s = O(N^{1/c})$, $m = O(N^{1-2/c})$, $t = O(\lambda)$, $n = O(\ell)$ and $h = O(m)$, we get $\kappa_{gr} = \mathrm{negl}(\lambda)$ with argument size $O(N^{1/c})$, verifier's complexity as $O(N)$ field operations and $O(N^{1-2/c})$ exponentiations.

N	Arg. Size(c_{zk}) MB			Verifier Time($t_{\mathcal{V}}$) sec			Prover Time($t_{\mathcal{P}}$) sec		
	G	L	B	G	L	B	G	L	B
2^{19}	0.728	3.5	0.001	45.4	55.2	89.1	802	137	662
2^{20}	0.759	4.9	0.001	49.8	205.57	178.2	1514	291	1324
2^{21}	0.884	6.95	0.001	55.97	212.17	356.5	2877	582	2648
2^{22}	1.28	9.8	0.001	108.306	805.3	713	5558	1258	5297
2^{23}	1.31	13.8	0.001	137.072	833.66	1426	10757	2516	10595

Fig. 2. Comparison of Graphene(G), Ligero(L) and Bulletproofs(B) in single prover setting for 80 bits of security

In Figure 2, we compare Graphene with Ligero [1] and Bulletproofs [12] in single prover setting based on the expressions in Appendix F.1. The concrete estimates were obtained by timing the FFT operations, exponentiations and multiexponentiations for different sizes, in a single threaded setting using libff library. Parameters for Graphene were optimized to yield best proving time, while those for Ligero were optimized to yield best proof size. From the table in Figure 2, we see that our protocol offers much more practical argument sizes compared to Ligero, while still attaining low verifier complexity.

Performance Evaluation of DP-Graphene. We now illustrate DP-Graphene's performance for a concrete example. We assume two provers P_1 and P_2 who wish to produce a proof of holding private coins $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ with serial numbers sn_1 and sn_2 on the Zcash blockchain, which are unspent and have combined value above some threshold v. The verification circuit consists of following major components:

Ensure the coins $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ are in the Merkle tree of coins. Each coin authentication takes around 1.8×10^6 gates (see [4, Section 5.2.2]).

Check that sn_1 and sn_2 are correctly computed from $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$. This take around $54,000$ gates.

Check that $v_1 + v_2 > v$ and $v_1 + v_2 < 2^{64}$, where v_1 and v_2 are values of the coins. This takes around 66 constraints.

For the above circuit, we consider $N \approx 4.0 \times 10^6$. For different values of parameters of our protocol, we set $N_s = \max(66, 4m\ell)$. For Bulletproofs, we take $N_s = 66$. Optimizing for total prover communication, our protocol achieves a total communication of 83.64 MB, with a proof size of 4.2 MB. Prover and verification time for our protocol is 9100 sec and 30 sec respectively. The distributed variant of Bulletproofs yields total prover communication of 168 MB, with proof size of 0.001 MB, proving and verification time of 5297 sec and 713 sec respectively. We used state-of-the-art dishonest majority semi-honest protocol (Oblivious Transfer based) MASCOT [38] for the MPC communication among the provers. All the experiments were done on 8-core Ubuntu Linux 18.04 Machine with 2 GHz processor and 32GB memory. We used libff and libqfft libraries for finite field and elliptic curve implementation, and also for the FFT and multi-exp algorithms. The elliptic curve used was 181-bit Edwards curve.

References

[1] S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *CCS*, pages 2087–2104, 2017.

[2] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson. Multi-prover interactive proofs: How to remove intractability assumptions. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pages 373–410. 2019.

[3] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10, 1988.

[4] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. Cryptology ePrint Archive, Report 2014/349, 2014.

[5] E. Ben-Sasson, A. Chiesa, M. Riabzev, N. Spooner, M. Virza, and N. P. Ward. Aurora: Transparent succinct arguments for R1CS. In *EUROCRYPT Part I*, pages 103–128, 2019.

[6] E. Ben-Sasson, A. Chiesa, and N. Spooner. Interactive oracle proofs. In *TCC 2016-B Part II*, pages 31–60, 2016.

[7] R. Bhadauria, Z. Fang, C. Hazay, M. Venkitasubramaniam, T. Xie, and Y. Zhang. Ligero++: A new optimized sublinear iop. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 2025–2038, 2020.

[8] A. J. Blumberg, J. Thaler, V. Vu, and M. Walfish. Verifiable computation using multiple provers. *IACR Cryptol. ePrint Arch.*, 2014:846, 2014.

[9] J. Bootle, A. Cerulli, P. Chaidos, J. Groth, and C. Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *EUROCRYPT Part II*, pages 327–357, 2016.

[10] J. Bootle, A. Chiesa, and J. Groth. Linear-time arguments with sublinear verification from tensor codes. In *Theory of Cryptography Conference*, pages 19–46. Springer, 2020.

[11] J. Bootle, A. Chiesa, and S. Liu. Zero-knowledge succinct arguments with a linear-time prover. 2020.

[12] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *IEEE SP*, pages 315–334, 2018.

[13] V. Buterin. Ethereum, 2013.

[14] R. Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

[15] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. P. Ward. Marlin: Preprocessing zksnarks with universal and updatable SRS. *IACR Cryptology ePrint Archive*, 2019:1047, 2019.

[16] R. Cohen and Y. Lindell. Fairness versus guaranteed output delivery in secure multiparty computation. In *ASIACRYPT Part II*, pages 466–485, 2014.

[17] H. Corrigan-Gibbs and D. Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, pages 259–282, 2017.

[18] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, pages 643–662, 2012.

[19] Y. Desmedt. *Threshold Cryptography*, pages 1288–1293. 2011.

[20] Y. Desmedt, G. D. Crescenzo, and M. Burmester. Multiplicative non-abelian sharing schemes and their application to threshold cryptography. In *ASIACRYPT*, pages 21–32, 1994.

[21] E. Druk and Y. Ishai. Linear-time encodable codes meeting the gilbert-varshamov bound and their cryptographic applications. In *Proceedings of the 5th conference on Innovations in theoretical computer science*, pages 169–182, 2014.

[22] E2open. Strategic digital supply chain. https://www.e2open.com/.

[23] J. Eberhardt and S. Tai. Zokrates - scalable privacy-preserving off-chain computations. In *IEEE International Conference on Internet of Things (iThings)*, pages 1084–1091, 2018.

[24] S. Englehardt. Privacy-preserving mozilla telemetry with prio. https://blog.mozilla.org/security/2019/06/06/next-steps-in-privacy-preserving-telemetry-with-prio/.

[25] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.

[26] O. Goldreich. *The Foundations of Cryptography - Volume 1: Basic Techniques*. Cambridge University Press, 2001.

[27] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

[28] V. Goyal, Y. Ishai, M. Mahmoody, and A. Sahai. Interactive locking, zero-knowledge pcps, and unconditional cryptography. In *CRYPTO*, pages 173–190, 2010.

[29] J. Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT*, pages 305–326, 2016.

[30] P. Hubáček, A. Rosen, and M. Vald. An efficiency-preserving transformation from honest-verifier statistical zero-knowledge to statistical zero-knowledge. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 66–87. Springer, 2018.

[31] IBM. Sterling supply chain business network. https://www.ibm.com/in-en/products/supply-chain-business-network.

[32] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge from secure multiparty computation. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 21–30, 2007.

[33] Y. Ishai, M. Mahmoody, and A. Sahai. On efficient zero-knowledge pcps. In *TCC*, pages 151–168, 2012.

[34] Y. Ishai and M. Weiss. Probabilistically checkable proofs of proximity with zero-knowledge. In *TCC*, volume 8349, pages 121–145. Springer, 2014.

[35] Y. T. Kalai and R. Raz. Interactive PCP. In *ICALP*, pages 536–547, 2008.

[36] Y. T. Kalai and R. Raz. Probabilistically checkable arguments. In *CRYPTO*, pages 143–159, 2009.

[37] M. Keller, G. L. Mikkelsen, and A. Rupp. Efficient threshold zero-knowledge with applications to user-centric protocols. In *ICITS*, pages 147–166, 2012.

[38] M. Keller, E. Orsini, and P. Scholl. MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In *ACM CCS*, pages 830–842, 2016.

[39] B. King. An efficient implementation of a threshold RSA signature scheme. In *ACISP*, pages 382–393, 2005.

[40] Y. Lindell. How to simulate it - A tutorial on the simulation proof technique. In *Tutorials on the Foundations of Cryptography*, pages 277–346. 2017.

[41] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error Correcting Codes*. North-Holland Publishing Company, 1978.

[42] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[43] B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: Nearly practical verifiable computation. In *IEEE SP*, pages 238–252. IEEE, 2013.

[44] T. P. Pedersen. Distributed provers with applications to undeniable signatures. In *EUROCRYPT*, pages 221–242, 1991.

[45] T. P. Pedersen. Distributed provers and verifiable secret sharing based on the discrete logarithm problem. *DAIMI Report Series*, 21(388), 1992. PhD Thesis.

[46] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *IEEE SP*, pages 459–474, 2014.

[47] B. Schoenmakers, M. Veeningen, and N. de Vreede. Trinocchio: Privacy-preserving outsourcing by distributed verifiable computation. In M. Manulis, A.-R. Sadeghi, and S. Schneider, editors, *ACNS*, pages 346–366, 2016.

[48] S. Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. Cryptology ePrint Archive, Report 2019/550, 2019. https://eprint.iacr.org/2019/550.

[49] Tradelens. Digitizing global supply chains. https://www.tradelens.com/.

[50] H. Wu, W. Zheng, A. Chiesa, R. A. Popa, and I. Stoica. DIZK: A distributed zero knowledge proof system. *IACR Cryptology ePrint Archive*, 2018:691, 2018.

[51] J. Zhang and T. Xie. Virgo: Zero knowledge proofs system without trusted setup. 2019.

Appendix

∗*We got approval from the chairs on relaxing the page limit for the appendices, hence we have added some more details on the protocol and the proofs here using extra pages. If accepted, we will remove these details from the proceedings version and will add the citation to the full version.*

A Preliminaries

A.1 Coding Theory Results

Proof of Proposition 3.2

We need several observations to prove the proposition, which we present next. Throughout this section, let L be a linear $[n, k, d]$ code over a field $\mathbb{H}$ and let $\mathbb{F} \subseteq \mathbb{H}$ be a subfield. Let $m \geq 1$ be an integer, and let $\mathcal{C}$ denote the row interleaved code $\mathsf{RIC}(L, m)$. For a matrix $U \in \mathbb{H}^{m\times n}$ and a vector $u_0 \in \mathbb{H}^n$, let $\mathrm{Aff}(u_0, U)$ denote the affine space as:

$$\mathrm{Aff}(u_0, U) := \{u_0 + r^T U : r \in \mathbb{F}^m\} \tag{4}$$

Note that in the above, the scalars in the linear combination come from $\mathbb{F}$.

The following result was proved in [1] for the setting $\mathbb{H} = \mathbb{F}$. For completeness, we present an adaptation of the proof to the setting where $\mathbb{F}$ is a subfield of $\mathbb{H}$.

Lemma A.1. *Let L and $\mathcal{C}$ be codes as defined, and let e be a positive integer such that $e+2 \leq |\mathbb{F}|$. Then for any $u_0 \in \mathbb{H}^n$ and any $U^* \in \mathbb{H}^{m\times n}$ such that $d(U^*, \mathcal{C}) > e$, there exists $v \in \mathrm{Aff}(u_0, U^*)$ such that $d(v, L) > e$.*

Proof. For sake of contradiction, assume that $d(u, L) \leq e$ for all $u \in \mathrm{Aff}(u_0, U^*)$. Let x be the point in $\mathrm{Aff}(u_0, U^*)$ such that $d(x, L)$ is maximum. By assumption $d(x, L) \leq e$. Let $v \in L$ be such that $\Delta(x, v) = d(x, L)$. Let $E \subseteq [n]$ be the set of positions where x and v differ. Since $d(U^*, \mathcal{C}) > e$, there exists row R of U^* and there is some position $j \in [n]\backslash E$ such that $R_j \neq 0$. Let $\alpha_1, \ldots, \alpha_{e+1}$ be distinct non zero elements in $\mathbb{F}$. Let E_k for $k = 1, \ldots, e+1$ denote the set of positions where $x + \alpha_k R$ and v differ. Fix a position $i \in E$. Then there exists at most one $k \in [e+1]$ such that $i \notin E_k$. By pigeon hole principle, there exists $k \in [e+1]$ such that $E \subseteq E_k$. We also observe that since $\alpha_k \neq 0$, $j \in E_k$. Thus $d(x + \alpha_k, v) > d(x, v)$, contradicting the choice of x. This proves the lemma. □

Next we prove a result about lines with respect to linear codes. The result was proved in [1] for the case $\mathbb{H} = \mathbb{F}$ and $e < d/4$. The authors in [1] conjectured the result for $e < d/3$. Here we prove the result for $e < d/3$ when $\mathbb{F}$ can be an arbitrary subfield of $\mathbb{H}$.

Lemma A.2 (Affine Line)**.** *Let L be the linear code as defined. Define an affine line $\ell_{u,v}$ in $\mathbb{H}^n$ as $\ell_{u,v} := \{u + \alpha v : \alpha \in \mathbb{F}\}$ for $u, v \in \mathbb{H}^n$. Then for $e < d/3$ and any affine line $\ell_{u,v}$:*

(i) *$d(x, L) \leq e$ for all $x \in \ell_{u,v}$, or*
(ii) *$d(x, L) > e$ for at most d points in $\ell_{u,v}$.*

Proof. In this proof, let $\delta(x, y)$ denote the set of positions where the vectors x and y differ, and let $\mathbf{wt}(x)$ denote $|\delta(x, \mathbf{0})|$. The above is equivalent to proving that if there exist $d+1$ distinct points $X = \{x_1, \ldots, x_{d+1}\} \subseteq \ell_{u,v}$ such that $d(x_i, L) \leq e$ for all $i \in [d+1]$, then $d(x, L) \leq e$ for all $x \in \ell_{u,v}$. Assume that there exists such a set X of $d+1$ points. Let ℓ_i denote the point in L closest to x_i for $i \in [d+1]$. Set $\eta_i = x_i - l_i$ for all i and let $\boldsymbol{\eta} = \{\eta_1, \ldots, \eta_{d+1}\}$. By assumption we have $\mathbf{wt}(\eta_i) \leq e$ for all $i \in [d+1]$. Since $\ell_{u,v}$ is contained in affine span of any two distinct points on it, we have tuples $\{(\alpha_i, \beta_i)\}_{i\in[d+1]} d$ such that $\alpha_i + \beta_i = 1$ and $x_i = \alpha_i x_1 + \beta_i x_2$ for $i \in [d+1]$. Note that $(\alpha_1, \beta_1) = (1, 0)$ and $(\alpha_2, \beta_2) = (0, 1)$. Observe that α_i's and β_i's must be distinct for all $i \in [d+1]$. We call X as *degenerate* if there exist $i \neq j$ satisfying $\alpha_j = \gamma\alpha_i$ and $\beta_j = \gamma\beta_j$ for some $\gamma \in \mathbb{F}\backslash 0$. We consider two cases:

X is degenerate: Degeneracy implies there exist $i \neq j$ such that $x_i = \gamma x_j$ for some $\gamma \in \mathbb{F}\backslash\{0\}$. In this case we have $0 = \frac{1}{1-\gamma} x_i - \frac{\gamma}{1-\gamma} x_j \in \ell_{u,v}$. This implies $\ell_{u,v} = \{\alpha x_i : \alpha \in \mathbb{F}\}$ and hence $d(x, L) = d(x_i, L) \leq e$ for all $x \in \ell_{u,v}$. Thus the statement of the Lemma holds in this case.

X is not degenerate: We first prove that $\ell_i = \alpha_i \ell_1 + \beta_i \ell_2$ and $\eta_i = \alpha_i \eta_1 + \beta_i \eta_2$ for all $i \in [d+1]$. We have

$$\begin{aligned} \ell_i + \eta_i = x_i = \alpha_i x_1 + \beta_i x_2 &= \alpha_i(\ell_1 + \eta_1) + \beta_i(\ell_2 + \eta_2) \\ &= (\alpha_i \ell_1 + \beta_i \ell_2) + (\alpha_i \eta_1 + \beta_i \eta_2) \end{aligned}$$

Rearranging we have, $\ell_i - (\alpha_i \ell_1 + \beta_i \ell_2) = \alpha_i \eta_1 + \beta_i \eta_2 - \eta_i$. In the above equation we see that LHS is a vector in L. Further $\mathbf{wt}(\alpha_i \eta_1 + \beta_i \eta_2 - \eta_i) \leq \mathbf{wt}(\eta_1) + \mathbf{wt}(\eta_2) + \mathbf{wt}(\eta_i) \leq 3e < d$. Thus the LHS must be equal to 0 and hence $\ell_i = \alpha_i \ell_1 + \beta_i \ell_2$ and $\eta_i = \alpha_i \eta_1 + \beta_i \eta_2$. Observe that any $x^* \in \ell_{u,v}$ can be written as $x^* = \alpha^* x_1 + \beta^* x_2$. Thus $d(x^*, L) = d(\alpha^* x_1 + \beta^* x_2, L) \leq \mathbf{wt}(\alpha^* \eta_1 + \beta^* \eta_2) \leq |E|$ where E denotes the set $\delta(\eta_1, 0) \cup \delta(\eta_2, 0)$. Our final effort will be to show that $|E| \leq e$.

Claim: $|E| \leq e$ where $E = \delta(\eta_1, 0) \cup \delta(\eta_2, 0)$. We consider the partition of E into sets $E_0 = \delta(\eta_1, 0)\backslash\delta(\eta_2, 0)$, $E_1 = \delta(\eta_2, 0)\backslash\delta(\eta_1, 0)$ and $E_{01} = \delta(\eta_1, 0) \cap \delta(\eta_2, 0)$. Let $t = |E|$. Consider a $t \times (d+1)$ matrix $M = (m_{ij})$ where $m_{ij} = 0$ if j^{th} coordinate (η_i^j) of η_i is zero, and $m_{ij} = 1$ otherwise. We will show that each row of M has at most one 0. Assume that there exists i such that $m_{ip} = 0$ and $m_{iq} = 0$ for $p \neq q$. We consider three cases:

– If $i \in E_0$, the above condition implies $\alpha_p \eta_1^i = \alpha_q \eta_1^i = 0$, or $\alpha_p = \alpha_q = 0$ as $\eta_1^i \neq 0$ for $i \in E_0$. This contradicts the fact that X is not degenerate.

- The case $i \in E_1$ is similar to above.
- If $i \in E_{01}$ we have $\alpha_p \eta_1^i + \beta_p \eta_2^i = \alpha_q \eta_1^i + \beta_q \eta_2^i = 0$ which implies $\alpha_p/\beta_p = \alpha_q/\beta_q = -\eta_2^i/\eta_1^i$, or $\alpha_p/\alpha_q = \beta_p/\beta_q$ which contradicts the fact that X is not degenerate. Note that all denominators can be argued to be non-zero for $i \in E_{01}$.

From the above, we conclude that each row has at least d entries as 1. Counting by columns, we have that each column has at most e entries as 1 (since $\mathbf{wt}(\eta_i) \leq e$ for all $i \in [d]$. Comparing the lower and upper bounds on the number of 1 entries in the matrix we have $td \leq e(d+1)$ which implies $t \leq e + e/d < e + 1$. Thus $t \leq e$, as we wanted to show. This completes the proof. □

The following result underlies proximity protocols in our work and in [1]. Intuitively the result states that if a matrix is far away from the code $\mathcal{C}$, a random linear combination of its rows is far away from a codeword in L, and thus the proximity of the matrix to $\mathcal{C}$ may be tested by testing the proximity of a random linear combination of its rows to L.

Lemma A.3 (Affine Subspace)**.** *Let the codes L and $\mathcal{C}$ be as defined and $e < d/3$ be an integer. Let $U \in \mathbb{H}^{m \times n}$ be a matrix such that $d(U, \mathcal{C}) > e$. Then for any $u_0 \in \mathbb{H}^n$, $\Pr\left[d(u_0 + r^T U, L) \leq e\right] \leq d/|\mathbb{F}|$ for uniformly sampled $r \leftarrow_{\$} \mathbb{F}^m$.*

Proof. From Lemma A.1, there exists $u \in \mathrm{Aff}(u_0, U)$ such that $d(u, L) > e$. Now we can write $\mathrm{Aff}(u_0, U)$ as union of affine lines passing through u. Applying lemma A.2 to each line, we see that at most d points x on each affine line satisfy $d(x, L) \leq e$. Thus, a randomly sampled point x in $\mathrm{Aff}(u_0, U)$, equivalently obtained as $u_0 + r^T U$ for a randomly sampled vector $r \in \mathbb{F}^m$ satisfies $d(x, L)$ with probability at most $d/|\mathbb{F}|$. □

We are now in a position to prove Proposition 3.2.

Proof of Proposition 3.2. Let $\mathbb{H}$ denote the field $\mathbb{F}^m$. Then u_0 can be viewed as a point in $\mathbb{H}^n$. Similarly U can be viewed as $p \times n$ matrix over $\mathbb{H}$. We consider $\mathcal{C}$ as $[n, k, d]$ code over $\mathbb{H}$ and $\mathcal{C}^p$ as the interleaved code of $\mathcal{C}$ over the field $\mathbb{H}$. Then by applying Lemma A.3 with $\mathbb{H} = \mathbb{F}^m$ and codes $\mathcal{C}$ and $\mathcal{C}^p$ in place of codes L and $\mathcal{C}$, we have the desired bound. □

Commitment and Inner-Product Argument

As described in Section 3, in our work we are using Pedersen vector commitment scheme, Com with message space $\mathbb{F}^m$, commitment space $\mathbb{G}$ and randomness space $\mathbb{F}$. For a set of generators $\boldsymbol{g} = (g_1, \ldots, g_m, h)$, the commitment of vector $x \in \mathbb{F}^m$ with commitment randomness $r \in \mathbb{F}$ is given by $\mathsf{Com}(x, r) = h^r \prod_{i=1}^m (g_i)^{x_i}$. This commitment is homomorphic with $\mathcal{C} = \mathbb{G}$ as the commitment space. This is a non-interactive perfectly hiding and computationally binding commitment scheme.

Definition 6 (Inner-product Argument [9, 12])**.** *We call an interactive protocol $\langle \mathsf{P}_{\mathrm{ip}}, \mathsf{V}_{\mathrm{ip}} \rangle$ consisting of* PPT *interactive algorithms P_{ip} and V_{ip} an inner-product argument for commitment scheme $(\mathsf{Gen}, \mathsf{Com})$ if it recognizes the language $\mathcal{L}$ as defined previously. Namely, $\langle \mathsf{P}_{\mathrm{ip}}, \mathsf{V}_{\mathrm{ip}} \rangle$ satisfies the following:*

(i) **Completeness***: For all $\mathcal{A}$, the following should be* 1

$$\Pr\left[\begin{array}{l|l} (x, \mathsf{w}) \in \mathcal{R} \wedge & \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda); \\ \langle \mathsf{P}_{\mathrm{ip}}(\mathsf{pp}, x, \mathsf{w}), \mathsf{V}_{\mathrm{ip}}(\mathsf{pp}, x) \rangle = 1 & (x, \mathsf{w}) \leftarrow \mathcal{A}(1^\lambda, \mathsf{pp}) \end{array}\right]$$

(ii) **Soundness***: For all deterministic polynomial time $\mathcal{P}^*$ and* PPT *$\mathcal{A}$, the following should be at most* $\mathrm{negl}(\lambda)$.

$$\Pr\left[\begin{array}{l|l} x \notin \mathcal{L} \wedge & \sigma \leftarrow \mathsf{Gen}(1^\lambda); \\ \langle \mathcal{P}^*(\mathsf{pp}, x, s), \mathsf{V}_{\mathrm{ip}}(\mathsf{pp}, u) \rangle = 1 & (x, s) \leftarrow \mathcal{A}(1^\lambda, \mathsf{pp}) \end{array}\right]$$

B Graphene Protocol and Security Proofs

B.1 Linear Check Protocol

Linear Check Soundness

Lemma B.1 (Soundness)**.** *For all polynomially bounded provers P^* and all $\pi \in \mathbb{G}^{p \times n}$, $A \in \mathbb{F}^{N \times N}, b \in \mathbb{F}^N$, there exists an expected polynomial time extractor $\mathcal{E}$ with rewinding access to transcript $\mathsf{tr} = \langle P^*(\cdot), \mathcal{V}^\pi(\cdot) \rangle$ such that $\mathcal{E}$ either breaks the commitment binding or outputs a witness with overwhelming probability whenever P^* succeeds, i.e,*

$$\Pr\left[\begin{array}{l|c} \mathsf{U} = \mathsf{Open}(\pi) \wedge & \sigma \leftarrow \mathsf{Gen}(1^\lambda) \\ A\mathsf{w} = b & \mathsf{U} \leftarrow \mathcal{E}^{\mathsf{tr}}(\boldsymbol{x}, \sigma) \\ & \mathsf{w} \leftarrow \mathsf{Dec}(\mathsf{U}) \end{array}\right] \geq \epsilon(P^*) - \kappa_{lc}(\lambda)$$

where $\epsilon(P^*) := \Pr\left[\langle P^*(\boldsymbol{x}, \sigma), \mathcal{V}^\pi(\boldsymbol{x}, \sigma) \rangle = 1 \,\middle|\, \sigma \leftarrow \mathsf{Gen}(1^\lambda)\right]$ denotes the success probability of P^*, κ_{lc} denotes a negligible function, and $\boldsymbol{x}$ denotes the tuple (A, b, M, N).

Proof. Suppose the oracle π commits to U. U can be "extracted" by the appropriate extractor. Note that $\mathsf{U} \in (\mathcal{W}_2)^p$ as a commitment implicitly corresponds to such a matrix. Let $e < (n - \ell)/3$ be a parameter. First

$\mathsf{LinearCheck}(\mathsf{pp}, A \in \mathcal{M}_{M,N}, b \in \mathbb{F}^N, [\pi]; \mathsf{U})$:
Relation: $\mathsf{U} = \mathsf{Open}(\pi) \wedge A\mathsf{w} = b$ for $\mathsf{w} = \mathsf{Dec}(\mathsf{U})$.

1. (i) $\mathcal{V} \to \mathcal{P}$: $\rho \leftarrow\!\$\, \mathbb{F}^p$. (ii) $\mathcal{P}$ computes: $\tilde{\mathsf{U}} = \sum_{i\in[p]} \rho_i \mathsf{U}[i,\cdot,\cdot]$, commitments $\tilde{c}_1, \ldots, \tilde{c}_\ell$ as $\tilde{c}_k = \prod_{i\in[p]} (\pi[i,k])^{\rho_i}\ \forall k \in [\ell]$. (iii) $\mathcal{P} \to \mathcal{V}$: $\tilde{\boldsymbol{c}} = (\tilde{c}_1, \ldots, \tilde{c}_\ell)$.
2. $\mathcal{V} \to \mathcal{P}$: $r \leftarrow\!\$\, \mathbb{F}^M$.
3. $\mathcal{P}$ and $\mathcal{V}$ compute: Polynomials R^i, $i \in [p]$ interpolating $R = r^T A$ as in Section 7.1.
4. $\mathcal{P}$ (a) computes matrix P from R and U as described in Section 7.1, (b) samples $P_0 \leftarrow\!\$\, \mathbb{F}^{2m-1}$, $\omega_0 \leftarrow\!\$\, \mathbb{F}$ and computes $c_0 \leftarrow \mathsf{Com}(P_0, \omega_0)$, (c) computes $(c_1, \ldots, c_{s+\ell-1}) \leftarrow \mathsf{pcCom}(P)$.
5. $\mathcal{P} \to \mathcal{V}$: $c_0, c_1, \ldots, c_{s+\ell-1}$.
6. $\mathcal{V} \to \mathcal{P}$: $Q = \{(j_u, k_u) : u \in [t]\}$ for $(j_u, k_u) \leftarrow\!\$\, [h] \times [n]$ for $u \in [t]$.
7. $\mathcal{V} \to \pi$: $\{k_u : u \in [t]\}$.
8. $\mathcal{P} \to \mathcal{V}$: $\mathsf{U}[\cdot, j_u, k_u]$ for $u \in [t]$.
9. $\pi \to \mathcal{V}$: $\pi[\cdot, k_u]$ for $u \in [t]$.
10. $\mathcal{V} \to \mathcal{P}$: $\delta \leftarrow\!\$\, \mathbb{F}^p$, $\beta \leftarrow\!\$\, \mathbb{F}\backslash\{0\}$.
11. $\mathcal{P}$ and $\mathcal{V}$ run inner product arguments to check:
    (a) $\mathsf{InnerProduct}(\mathsf{pp}, \mathbf{1}_{j_u}^T \Lambda_{h,2m-1}, \mathsf{cm}_{k_u}, v_u; \overline{P}[\cdot, k_u])$ for $u \in [t]$ where $\mathsf{cm}_{k_u} = \prod_{a=1}^{s+\ell-1} c_a^{\Lambda^T_{n,s+\ell-1}[a,k_u]}$, $v_u = \sum_{i=1}^p R^i(\alpha_{j_u}, \eta_{k_u}) U[i, j_u, k_u]$ (check consistency of P with π).
    (b) $\mathsf{InnerProduct}(\mathsf{pp}, 1^m || 0^{m-1}, \mathsf{cm}, r^T b; z)$ where $z = \beta P_0 + \overline{P}\varphi$ and $\mathsf{cm} = c_0^\beta \cdot \prod_{a=1}^{s+\ell-1} c_k^{\varphi_k}$ (check the condition $r^T A w = r^T b$).
    (c) $\mathsf{InnerProduct}(\mathsf{pp}, \mathbf{1}_{j_u}^T \Lambda_{h,m}, C_u, \langle \delta, X_u \rangle; V_u)$ for $u \in [t]$ where $C_u = \prod_{i=1}^p (\pi[i,k_u])^{\delta_i}$ and $V_u = \sum_{i\in[p]} \delta_i \mathsf{U}[i,\cdot,k_u]$ (consistency of X_u with π).
12. $\mathcal{V}$ checks: $\prod_{a=1}^{\ell} (\tilde{c}_a)^{\Lambda^T_{n,\ell}[a,k_u]} = \prod_{i=1}^p (\pi[i,k_u])^{\rho_i}$ for $u \in [t]$ (check proximity of U to $\mathcal{W}_1$).
13. $\mathcal{V}$ accepts if all the checks succeed.

Fig. 3. Linear Check Protocol

we show that an adversarial prover succeeds with negligible probability if $d(\mathsf{U}, (\mathcal{W}_1)^p) > e$. Second, we show that for $d(\mathsf{U}, (\mathcal{W}_1)^p) \le e$, the prover succeeds with negligible probability when $A\mathsf{w} \neq b$ where $\mathsf{w} = \mathsf{Dec}(\mathsf{U})$. Consider the case when $d(\mathsf{U}, (\mathcal{W}_1)^p) > e$. Then for $\tilde{\mathsf{U}} = \sum_{i\in[p]} \rho_i \mathsf{U}[i,\cdot,\cdot]$, by Proposition 3.2, we have $d(\tilde{\mathsf{U}}, \mathcal{C}_1) > e$ with probability $1-o(1)$. Let $\tilde{\boldsymbol{c}} = (\tilde{c}_1, \ldots, \tilde{c}_\ell)$ be the commitments to $\tilde{\mathsf{U}}$ sent by the prover (Step 3 in Figure 3). Define the vector $\tilde{\boldsymbol{C}} = (\tilde{C}_1, \ldots, \tilde{C}_n)$ where $\tilde{C}_k = \prod_{a=1}^{\ell} (\tilde{c}_a)^{\Lambda^T_{n,\ell}[a,k]}$ for $k \in [n]$. Let $\hat{\boldsymbol{C}} = (\hat{C}_1, \ldots, \hat{C}_n)$ where $\hat{C}_k = \prod_{i=1}^p (\pi[i,k])^{\rho_i}$. Now if $\Delta(\tilde{\boldsymbol{C}}, \hat{\boldsymbol{C}}) > e$, we see that the prover succeeds in the proximity check equation (1) with probability at most $(1 - e/n)^t$. If $\Delta(\tilde{\boldsymbol{C}}, \hat{\boldsymbol{C}}) \le e$, while $d(\tilde{\mathsf{U}}, \mathcal{C}_1) > e$, then it is easy to break the binding property commitment scheme. Thus an adversarial prover succeeds with probability at most $(1-e/n)^t$ when $d(\mathsf{U}, (\mathcal{W}_1)^p) > e$.

We now consider the case when $d(\mathsf{U}, (\mathcal{W}_1)^p) \le e$. From Lemma 3.1, there exists (unique) $\mathsf{U}^* \in (\mathcal{W})^p$ such that $\Delta_1(\mathsf{U}, \mathsf{U}^*) \le e$. Let $\mathsf{w} = \mathsf{Dec}(\mathsf{U}) = \mathsf{Dec}(\mathsf{U}^*)$. We consider the prover's success probability when $A\mathsf{w} \neq b$, and thus with overwhelming probability $r^T A\mathsf{w} \neq r^T b$. Let P^* denote the correctly computed intermediate matrix from U^* and let $\hat{P}$ denote the correctly computed intermediate matrix from U. We note that $\Delta_1(\hat{P}, P^*) \le e$. Let $c_1, \ldots, c_{s+\ell-1}$ be the commitments to the intermediate matrix sent by the prover. If these commitments correspond to a matrix $P = P^*$, the inner product check $\mathsf{InnerProduct}(\mathsf{pp}, [1^m || 0^{m-1}], \mathsf{cm}, r^T b)$ fails when using the commitment $\mathsf{cm} = \prod_{k=1}^{s+\ell-1} c_k^{\varphi_k}$ for $\varphi = \Phi \times [1^s]$. This is because $\langle 1^m || 0^{m-1}, P^*[1 : 2m-1, 1 : s+\ell-1] \times \varphi \rangle = r^T A\mathsf{w} \neq r^T b$, as discussed in the protocol. If the commitments correspond to a matrix $P \neq P^*$, we have $\Delta_1(P, P^*) \ge n - s - \ell$ by distance property of the code $\mathsf{RS}_{\boldsymbol{\eta}}[n, s+\ell-1]$. (We note that a prover implicitly commits to a matrix in $\mathsf{RS}_{\boldsymbol{\eta}}[n, s+\ell-1] \otimes \mathsf{RS}_{\boldsymbol{\alpha}}[h, 2m-1]$). Thus there exists a set E of at least $n-s-\ell-e$ columns, such that for $k \in E$, $\hat{P}[\cdot, k] = P^*[\cdot, k] \neq P[\cdot, k]$. The checks $\langle \mathbf{1}_{j_u}, \mathsf{U}[i,\cdot,k_u] \rangle = X_u[i]$ for $i \in [p]$ and $u \in [t]$ force the prover to provide vectors $X_u = \mathsf{U}[\cdot, j_u, k_u]$ with overwhelming probability. Then the consistency check succeeds for the uniformly sampled query point (j_u, k_u) when:

$$P[j_u, k_u] = \sum_{i\in[p]} R^i(\alpha_{j_u}, \eta_{k_u}) X_u[i] = \hat{P}[j_u, k_u]$$

For $k_u \in E$, the above holds when the distinct codewords $P^*[\cdot, k_u]$ and $P[\cdot, k_u]$ in $\mathsf{RS}_{\boldsymbol{\alpha}}[h, 2m-1]$ agree on the position j_u, which happens with probability at most $2m/h$. Thus probability $\Pr[\mathcal{E}]$ that a corrupt prover with U such that $d(\mathsf{U}, (\mathcal{W}_1)^p) \le e$ succeeds is bounded by:

$$\begin{aligned}\Pr[\mathcal{E}] &\le \frac{\binom{s+\ell+e}{t}}{\binom{n}{t}} + \frac{n-s-\ell-e}{n} \cdot \frac{\binom{2m}{t}}{\binom{h}{t}} \\ &= \left(\frac{s+\ell+e}{n}\right)^t + \left(\frac{n-s-\ell-e}{n}\right) \cdot \left(\frac{2m}{h}\right)^t\end{aligned}$$

The above probability is smaller than a constant $\epsilon < 1$ for suitable choices of parameters. Hence, the overall probability of prover's success is $\mathrm{negl}(\lambda)$ for $t = O(\lambda)$.

Extraction:

Let $\mathcal{E}_{ip}$ be the extractor of the inner product argument which takes, in expectation, $\mathsf{p}_{ip}(n)$ amount of time,

where $\mathsf{p}_{ip}(\cdot)$ is polynomial, to output the private vector of length or breaking the binding of the commitment scheme. We will design an extractor $\mathcal{E}$ for LinearCheck, which uses $\mathcal{E}_{ip}$.

If $\mathcal{P}^*$ fails in the proximity check (step 12) then $\mathcal{V}$ outputs reject, and so the extractor $\mathcal{E}$ terminates with abort. If $\mathcal{P}^*$ succeeds in the proximity check (step 12) then $\mathsf{U} \in \mathcal{W}_1$ and $\mathcal{E}$ proceeds in the following way:

$\mathcal{E}$ plays the role of the verifier and rewinds the provers polynomially many times if required.

$\mathcal{E}$ runs the protocol till step 8, sends Q and receives $\mathsf{U}[\cdot, j_u, k_u]$.

$\mathcal{E}$ picks random $\delta \in \mathbb{F}^p$ and $\beta \in \mathbb{F}^*$ and proceeds to run the inner product arguments. $\mathcal{E}$ uses $\mathcal{E}_{ip}$ and gets:

- $\overline{P}[\cdot, k_u]$ in expected time $\mathsf{p}_{ip}(m)$, $\forall u \in [t]$.
- $z = \beta P_0 + \overline{P}\varphi$ in expected time $\mathsf{p}_{ip}(m)$.
- $V_u = \sum_{i\in[p]} \delta_i \mathsf{U}[i, \cdot, k_u]$ in expected time $\mathsf{p}_{ip}(m)$, $\forall u \in [t]$.

Then $\mathcal{E}$ rewinds δ, β $O(p\log(p))$ times and gets P_0 from z and from V_u, $\mathcal{E}$ gets $\mathsf{U}[\cdot, \cdot, k_u]$. $\mathcal{E}$ checks $\mathsf{U}[\cdot, j_u, k_u]$ received in step 8, matches with the extracted $\mathsf{U}[\cdot, \cdot, k_u]$'s j_u position or not. If does not, then $\mathcal{E}$ gets two opening of $\pi[\cdot, k_u]$ and outputs abort, otherwise $\mathcal{E}$ proceeds in the following way: $\mathcal{E}$ rewinds $\mathcal{P}^*$ and sends uniformly random Q and keeps repeating until all the Q's together cover all the columns. It takes $n\log(n)$ rewindings in expectation. Then $\mathcal{E}$ extracts the whole U and checks if all the vectors are consistent or not. If not, that gives the break for the binding of the commitment scheme and if consistent, then it decodes U and outputs correct witness w. Therefore, expected Run time of $\mathcal{E}$ is $O(n\log(n)(O(p\log(p))(\mathsf{p}_{ip}(m)) + \mathsf{p}_{ip}(m) + \mathsf{p}_{ip}(m)))$, which is polynomial in the size of the circuit. □

B.2 Quadratic Check Protocol

Quadratic Check Soundness

Lemma B.2 (Soundness)**.** *For all polynomially bounded provers P^* and all $\pi \in \mathbb{G}^{3p\times n}$, there exists an expected polynomial time extractor $\mathcal{E}$ with rewinding access to the transcript oracle $\mathsf{tr} = \langle P^*(\cdot), \mathcal{V}^{\pi}(\cdot)\rangle$ such that either $\mathcal{E}$ breaks the commitment binding, or it outputs a witness with overwhelming probability whenever P^* succeeds, i.e,*

$$\Pr\left[\begin{array}{c|c} [\mathsf{U}_x||\mathsf{U}_y||\mathsf{U}_z] = \mathsf{Open}(\pi) \wedge & \sigma \leftarrow \mathsf{Gen}(1^\lambda) \\ \mathsf{w}_z = \mathsf{w}_x \circ \mathsf{w}_y & [\mathsf{U}_x||\mathsf{U}_y||\mathsf{U}_z] \leftarrow \mathcal{E}^{\mathsf{tr}}(\sigma) \\ & \mathsf{w}_a = \mathsf{Dec}(\mathsf{U}_a), a \in \{x,y,z\} \end{array}\right] \geq \epsilon(P^*) - \kappa_{\mathrm{qd}}(\lambda)$$

QuadraticCheck$(\mathsf{pp}, [\pi_x], [\pi_y], [\pi_z]; \mathsf{U}_x, \mathsf{U}_y, \mathsf{U}_z)$:
Relation: $\mathsf{U}_a = \mathsf{Open}(\pi_a)$ for $a \in \{x,y,z\}$, $\mathsf{w}_x \circ \mathsf{w}_y = \mathsf{w}_z$ where $\mathsf{w}_a = \mathsf{Dec}(\mathsf{U}_a)$ for $a \in \{x,y,z\}$.

1. $\mathcal{V} \to \mathcal{P}$: $\rho \leftarrow\$ \mathbb{F}^{3p}$.
2. $\mathcal{P}$ computes: (a) $\tilde{\mathsf{U}} = \sum_{i=1}^{p}[\rho_i \mathsf{U}_x[i,\cdot,\cdot] + \rho_{p+i}\mathsf{U}_y[i,\cdot,\cdot] + \rho_{2p+i}\mathsf{U}_z[i,\cdot,\cdot]]$, (b) commitments $\tilde{c}_1, \ldots, \tilde{c}_\ell$ as $\tilde{c}_k = \prod_{i=1}^{p}(\pi_x[i,k])^{\rho_i} \cdot (\pi_y[i,k])^{\rho_{p+i}} \cdot (\pi_z[i,k])^{\rho_{2p+i}}$ $\forall k \in [\ell]$.
3. $\mathcal{P} \to \mathcal{V}$: $\tilde{\boldsymbol{c}} = (\tilde{c}_1, \ldots, \tilde{c}_\ell)$.
4. $\mathcal{V} \to \mathcal{P}$: $r \leftarrow\$ \mathbb{F}^p$.
5. $\mathcal{P}$ computes: (a) $p_j(\cdot) = \sum_{i\in[p]} r_i[Q_x^i(\alpha_j,\cdot)Q_y^i(\alpha_j,\cdot) - Q_z^i(\alpha_j,\cdot)]$ $\forall j \in [h]$, (b) matrix P such that $P[j,k] = p_j(\eta_k)$ as described in Section 7.2, (c) computes commitments $c_1, \ldots, c_{2\ell}$ from P. $\mathcal{P}$ also samples $P_0 \leftarrow\$ \mathbb{F}^{2m-1}$ with $P_0[j] = 0^m$ for $j \in [m]$ and computes commitment c_0 to P_0.
6. $\mathcal{P} \to \mathcal{V}$: $c_0, c_1, \ldots, c_{2\ell-1}$.
7. $\mathcal{V} \to \mathcal{P}$: $Q = \{(j_u, k_u) : u \in [t]\}$ for $Q \leftarrow\$ [h] \times [n]$, $u \in [t]$ and $\tau \leftarrow\$ \mathbb{F}^s$, $\gamma \leftarrow\$ \mathbb{F}^m$.
8. $\mathcal{V} \to \pi$: $\{k_u : u \in [t]\}$.
9. $\mathcal{P} \to \mathcal{V}$: $X_u = \mathsf{U}_x[\cdot, j_u, k_u]$, $Y_u = \mathsf{U}_y[\cdot, j_u, k_u]$ and $Z_u = \mathsf{U}_z[\cdot, j_u, k_u]$ for $u \in [t]$.
10. $\pi \to \mathcal{V}$: $\pi[\cdot, k_u]$ for $u \in [t]$.
11. $\mathcal{V} \to \mathcal{P}$: $\delta \leftarrow\$ \mathbb{F}^p$, $\beta_x \leftarrow\$ \mathbb{F}$, $\beta_y \leftarrow\$ \mathbb{F}$, $\beta_z \leftarrow\$ \mathbb{F}$, $\beta \leftarrow\$ \mathbb{F}\backslash\{0\}$.
12. $\mathcal{P}$ computes: $V_u = \sum_{i=1}^{p} \delta_i \big(\beta_x \mathsf{U}_x[i,\cdot,k_u] + \beta_y \mathsf{U}_y[i,\cdot,k_u] + \beta_z \mathsf{U}[i,\cdot,k_u]\big)$.
13. $\mathcal{P}$ and $\mathcal{V}$ compute:
    - $W_u = \beta_x X_u + \beta_y Y_u + \beta_z Z_u$ for $u \in [t]$.
    - $T_u = (C_u)^{\beta_x} \cdot (D_u)^{\beta_y} \cdot (E_u)^{\beta_z}$, for $u \in [t]$ where $C_u = \prod_{i=1}^{p}(\pi_x[i,k_u])^{\delta_i}$, $D_u = \prod_{i=1}^{p}(\pi_y[i,k_u])^{\delta_i}$ and $E_u = \prod_{i=1}^{p}(\pi_y[i,k_u])^{\delta_i}$.
14. $\mathcal{P}$ and $\mathcal{V}$ run inner-product arguments to check:
    (a) $\mathsf{InnerProduct}(\mathsf{pp}, \mathbf{1}_{j_u}^T \Lambda_{h,2m-1}, \mathsf{cm}_{k_u}, v_u; \overline{P}[\cdot, k_u])$ for $u \in [t]$ where $\mathsf{cm}_{k_u} = \prod_{a=1}^{2\ell-1}(c_a)^{\Lambda^T_{n,2\ell-1}[a,k_u]}$, $v_u = \sum_{i=1}^{p} r_i[X_u[i] \cdot Y_u[i] - Z_u[i]]$ (check consistency of P with π).
    (b) $\mathsf{InnerProduct}(\mathsf{pp}, \gamma||0^{m-1}, \mathsf{cm}, 0; z)$ where $z = \beta P_0 + \overline{P}\varphi$, $\varphi = \Phi^T\tau$ and $\mathsf{cm} = (c_0)^{\beta} \cdot \prod_{a=1}^{2\ell-1}(c_a)^{\varphi_a}$.
    (c) $\mathsf{InnerProduct}(\mathsf{pp}, \mathbf{1}_{j_u}^T \Lambda_{h,m}, T_u, w_u; \overline{V}_u])$, where $\overline{V}_u$ stands for first m entries of V_u and $w_u = \langle \delta, W_u \rangle$ (consistency of X_u, Y_u, Z_u with π).
15. $\mathcal{V}$ checks proximity of $\mathsf{U}_x, \mathsf{U}_y$ and U_z according to Eqn (2).
16. $\mathcal{V}$ accepts if all the checks succeed.

Fig. 4. Quadratic Check Protocol

for some negligible function κ_{qd}. In the above, $\epsilon(P^*)$ denotes the success probability of the prover P^* as before.

Proof. Suppose the oracle π commits to $\mathsf{U}_x, \mathsf{U}_y, \mathsf{U}_z$. $\mathsf{U}_x, \mathsf{U}_y, \mathsf{U}_z$ can be "extracted" by the appropriate extractor. Note that $\mathsf{U}_x||\mathsf{U}_y||\mathsf{U}_z \in (\mathcal{W}_2)^{3p}$, where $\mathsf{U} = \mathsf{U}_x||\mathsf{U}_y||\mathsf{U}_z$ is the juxtaposing along p direction, as a commitment implicitly corresponds to such a matrix. Let $e < (n-\ell)/3$ be a parameter. First we show that an adversarial prover succeeds with negligible probability if $d(\mathsf{U}, (\mathcal{W}_1)^{3p}) > e$. Second, we show that for $d(\mathsf{U}, (\mathcal{W}_1)^{3p}) \leq e$, the prover succeeds with negligible probability when $\mathsf{w}_x \circ \mathsf{w}_y \neq \mathsf{w}_z$ where $\mathsf{w}_a = \mathsf{Dec}(\mathsf{U}_a)$ for $a \in \{x, y, z\}$. Consider the case when $d(\mathsf{U}, (\mathcal{W}_1)^{3p}) > e$. Then for $\tilde{\mathsf{U}} = \sum_{i\in[3p]} \rho_i \mathsf{U}[i, \cdot, \cdot]$, by Proposition 3.2, we have $d(\tilde{\mathsf{U}}, \mathcal{C}_1) > e$ with probability $1 - o(1)$. Let $\tilde{\boldsymbol{c}} = (\tilde{c}_1, \ldots, \tilde{c}_\ell)$ be the commitments to $\tilde{\mathsf{U}}$ sent by the prover (Step 3 in Figure 4). Define the vector $\tilde{\boldsymbol{C}} = (\tilde{C}_1, \ldots, \tilde{C}_n)$ where $\tilde{C}_k = \prod_{a=1}^{\ell} (\tilde{c}_a)^{\Lambda^T_{n,\ell}[a,k]}$ for $k \in [n]$. Let $\hat{\boldsymbol{C}} = (\hat{C}_1, \ldots, \hat{C}_n)$ where $\hat{C}_k = \prod_{i=1}^{3p} (\pi[i,k])^{\rho_i}$. Now if $\Delta(\tilde{\boldsymbol{C}}, \hat{\boldsymbol{C}}) > e$, we see that the prover succeeds in the proximity check equation (1) with probability at most $(1 - e/n)^t$. If $\Delta(\tilde{\boldsymbol{C}}, \hat{\boldsymbol{C}}) \leq e$, while $d(\tilde{\mathsf{U}}, \mathcal{C}_1) > e$, then it is easy to break the binding property commitment scheme. Thus an adversarial prover succeeds with probability at most $(1 - e/n)^t$ when $d(\mathsf{U}, (\mathcal{W}_1)^p) > e$.

We now consider the case when $d(\mathsf{U}, (\mathcal{W}_1)^{3p}) \leq e$. From Lemma 3.1, there exists (unique) $\mathsf{U}^* \in (\mathcal{W})^{3p}$ such that $\Delta_1(\mathsf{U}, \mathsf{U}^*) \leq e$. Let $\mathsf{w}_a = \mathsf{Dec}(\mathsf{U}_a) = \mathsf{Dec}(\mathsf{U}^*_a)$. We consider the prover's success probability when $\mathsf{w}_x \circ \mathsf{w}_y = \mathsf{w}_z$, and thus with overwhelming probability $\sum_{i\in[p]} r_i \cdot (\mathsf{w}_x[i,\cdot,\cdot] \cdot \mathsf{w}_y[i,\cdot,\cdot] - \mathsf{w}_z[i,\cdot,\cdot]) = [\mathbf{0}]^{ms}$. Let P^* denote the correctly computed intermediate matrix from U^* and let $\hat{P}$ denote the correctly computed intermediate matrix from U. We note that $\Delta_1(\hat{P}, P^*) \leq e$. Let $c_1, \ldots, c_{2\ell-1}$ be the commitments to the intermediate matrix sent by the prover. If these commitments correspond to a matrix $P = P^*$, the inner product check $\mathsf{InnerProduct}(\mathsf{pp}, [\gamma||0^{m-1}], \mathsf{cm}, 0)$ fails when using the commitment $\mathsf{cm} = \prod_{k=1}^{2\ell-1} c_k^{\varphi_k}$ for $\varphi = \Phi \times \tau$. This is because $\langle \gamma||0^{m-1}, P^*[1:2m-1, 1:2\ell-1] \times \varphi\rangle = \sum_{j\in[m]} \gamma_j \sum_{k\in[s]} \tau_k \sum_{i\in[p]} r_i[\mathsf{w}_x[i,j,k] \cdot \mathsf{w}_y[i,j,k] - \mathsf{w}_z[i,j,k]] \neq 0$, as discussed in the protocol. If the commitments correspond to a matrix $P \neq P^*$, we have $\Delta_1(P, P^*) \geq n - 2\ell$ by distance property of the code $\mathsf{RS}_{\boldsymbol{\eta}}[n, 2\ell-1]$. (We note that a prover implicitly commits to a matrix in $\mathsf{RS}_{\boldsymbol{\eta}}[n, 2\ell-1] \otimes \mathsf{RS}_{\boldsymbol{\alpha}}[h, 2m-1]$). Thus there exists a set E of at least $n - 2\ell - e$ columns, such that for $k \in E$, $\hat{P}[\cdot, k] = P^*[\cdot, k] \neq P[\cdot, k]$. The checks $\langle \mathbf{1}_{j_u}, W_u\rangle = \sum_{i\in[p]} \delta_i(\beta_x X_u + \beta_y Y_u + \beta_z Z_u)$ for $i \in [p]$ and $u \in [t]$ force the prover to provide vectors $X_u = \mathsf{U}_x[\cdot, j_u, k_u], Y_u = \mathsf{U}_y[\cdot, j_u, k_u], Z_u = \mathsf{U}_z[\cdot, j_u, k_u]$ with overwhelming probability. Then the consistency check succeeds for the uniformly sampled query point (j_u, k_u) when:

$$P[j_u, k_u] = \sum_{i\in[p]} r_i(X_u[i] \cdot Y_u[i] - Z_u[i]) = \hat{P}[j_u, k_u]$$

For $k_u \in E$, the above holds when the distinct codewords $P^*[\cdot, k_u]$ and $P[\cdot, k_u]$ in $\mathsf{RS}_{\boldsymbol{\alpha}}[h, 2m-1]$ agree on the position j_u, which happens with probability at most $2m/h$. Thus probability $\Pr[\mathcal{E}]$ that a corrupt prover with U such that $d(\mathsf{U}, (\mathcal{W}_1)^{3p}) \leq e$ succeeds is bounded by:

$$\begin{aligned}\Pr[\mathcal{E}] &\leq \frac{\binom{2\ell+e}{t}}{\binom{n}{t}} + \frac{n-2\ell-e}{n} \cdot \frac{\binom{2m}{t}}{\binom{h}{t}} \\ &= \left(\frac{2\ell+e}{n}\right)^t + \left(\frac{n-2\ell-e}{n}\right)\left(\frac{2m}{h}\right)^t\end{aligned}$$

The above probability is smaller than a constant $\epsilon < 1$ for suitable choices of parameters. Hence, the overall probability of prover's success is $\mathrm{negl}(\lambda)$ for $t = O(\lambda)$. We will set $2n > 5\ell$.

Extraction:

Let $\mathcal{E}_{ip}$ be the extractor of the inner product argument which takes, in expectation, $\mathsf{p}_{ip}(n)$ amount of time, where $\mathsf{p}_{ip}(\cdot)$ is polynomial, to output the private vector of length or breaking the binding of the commitment scheme. We will design an extractor $\mathcal{E}$ for $\mathsf{QuadraticCheck}$, which uses $\mathcal{E}_{ip}$.

If $\mathcal{P}^*$ fails in the proximity check (step 15) then $\mathcal{V}$ outputs reject, and so the extractor $\mathcal{E}$ terminates with `abort`. If $\mathcal{P}^*$ succeeds in the proximity check (step 15) then $\mathsf{U} \in \mathcal{W}_1$ and $\mathcal{E}$ proceeds in the following way:

$\mathcal{E}$ plays the role of the verifier and rewinds the provers polynomially many times if required.

$\mathcal{E}$ runs the protocol till step 7, sends Q, γ, τ and receives $\mathsf{U}_x[\cdot, j_u, k_u], \mathsf{U}_y[\cdot, j_u, k_u], \mathsf{U}_z[\cdot, j_u, k_u]$. $\mathcal{E}$ picks random $\delta \in \mathbb{F}^p$, $\beta_x, \beta_y, \beta_z \in \mathbb{F}$ and $\beta \in \mathbb{F}^*$ and proceeds to run the inner product arguments. $\mathcal{E}$ uses $\mathcal{E}_{ip}$ and gets:

- $\overline{P}[\cdot, k_u]$ in expected time $\mathsf{p}_{ip}(m)$, $\forall u \in [t]$.
- $z = \beta P_0 + \overline{P}\varphi$ in expected time $\mathsf{p}_{ip}(m)$.
- $V_u = \sum_{i\in[p]} \delta_i(\beta_x \mathsf{U}_x[i,\cdot,k_u] + \beta_y \mathsf{U}_y[i,\cdot,k_u] + \beta_z \mathsf{U}_z[i,\cdot,k_u])$ in expected time $\mathsf{p}_{ip}(m)$, $\forall u \in [t]$.

Then $\mathcal{E}$ rewinds $\delta, \beta_x, \beta_y, \beta_z, \beta$ $O(p\log(p))$ times and gets V_u, $\mathcal{E}$ gets $\mathsf{U}_x[\cdot,\cdot,k_u], \mathsf{U}_y[\cdot,\cdot,k_u], \mathsf{U}_z[\cdot,\cdot,k_u]$ by solving a system of equation with $3p$ unknowns. $\mathcal{E}$ checks

$\mathsf{U}_a[\cdot, j_u, k_u]$ received in step 7, matches with the extracted $\mathsf{U}_a[\cdot,\cdot,k_u]$'s j_u position or not, for $a \in \{x,y,z\}$. If does not, then $\mathcal{E}$ gets two opening of $\pi_a[\cdot,k_u]$, for some $a \in \{x,y,z\}$ and outputs `abort`, otherwise $\mathcal{E}$ proceeds in the following way: $\mathcal{E}$ rewinds $\mathcal{P}^*$ and sends uniformly random Q and keeps repeating until all the Q's together cover all the columns. It takes $n\log(n)$ rewindings in expectation. Then $\mathcal{E}$ extracts the whole $\mathsf{U}_x, \mathsf{U}_y, \mathsf{U}_z$ and checks if all the vectors are consistent or not. If not, that gives the break for the binding of the commitment scheme and if consistent, then it decodes U_a and outputs correct witness w_a for $a \in \{x,y,z\}$.

Therefore, expected Run time of $\mathcal{E}$ is $O(n\log(n)(O(p\log(p))(\mathsf{p}_{ip}(m)) + \mathsf{p}_{ip}(m) + \mathsf{p}_{ip}(m)))$, which is polynomial in the size of the circuit. □

B.3 Graphene Protocol

For a cheating prover either $d(\mathsf{U}, (\mathcal{W}_1)^{4p}) > e$, or $d(\mathsf{U}, (\mathcal{W}_1)^{4p}) \le e$ for some e. Here we will set $e < (n-\ell)/3$. Then for the first case, that is, if $d(\mathsf{U}, (\mathcal{W}_1)^{4p}) > e$ then either the hamming distance of the commitment matrix will be more than e, which leads to fail in the proximity check with very high probability. Otherwise, if the hamming distance of the commitment matrix are not $> e$, then this leads to opening of a commitment to multiple values, which breaks the binding property of the commitment scheme. For the latter case, if $d(\mathsf{U}, (\mathcal{W}_1)^{4p}) \le e$, then there exists unique codeword U^* such that $\Delta_1(\mathsf{U}, \mathsf{U}^*) \le e$ and let $\mathsf{w} = \mathsf{Dec}(\mathsf{U}) = \mathsf{Dec}(\mathsf{U}^*)$. Since the prover is cheating, we assume that w is not the correct witness, i.e., either it fails to satisfy the linear constraint or the quadratic constraint or both. In such a case, either of the inner product check fails.

To prove of Lemma 7.1, we can construct an extractor $\mathcal{E}$ using the extractors $\mathcal{E}_{lc}$ and $\mathcal{E}_{qd}$ of LinearCheck and QuadraticCheck described in B.1 and B.2 respectively. $\mathcal{E}$ plays the role of the verifier and $\mathcal{E}_{lc}$, $\mathcal{E}_{qd}$ use the queries generated by $\mathcal{E}$.

B.4 Zero-knowledge

Now we will discuss the zero knowledge property of GrapheneR1CS, which has the similar idea of Linear Check and Quadratic Check. The verifier's extended view for the Graphene protocol consists of:

– *Verifier Randomness:* $\mathcal{V}$'s messages consist of: $\{\gamma_x, \gamma_y, \gamma_z, r_{lc}, r_{qd}, \rho, Q = \{(j_u, k_u)\}_{u\in[t]}, \delta_{lc}, \delta_{qd}, \beta_{lc}, \beta_x, \beta_y, \beta_z, \beta_{qd}, \tau_{qd}, \gamma_{qd}\}$.

GrapheneR1CS($\mathsf{pp}, A, B, C, [\pi]; \mathsf{w}_x, \mathsf{w}_y, \mathsf{w}_z, \mathsf{w}$):
Relation: $A\mathsf{w} \circ B\mathsf{w} = C\mathsf{w}$.
Oracle Setup: Compute $\mathcal{O}$ as described above. Set $\pi := \mathcal{O}$.

1. $\mathcal{V} \to \mathcal{P}$: $\gamma_x, \gamma_y, \gamma_z \leftarrow\$ \mathbb{F}$, $r_{lc} \leftarrow\$ \mathbb{F}^M$, $r_{qd} \leftarrow\$ \mathbb{F}^p$, $\rho \leftarrow\$ \mathbb{F}^{4p}$.
2. $\mathcal{P} \to \mathcal{V}$: $\mathcal{P}$ computes $\tilde{\mathsf{U}} = \sum_{i\in[4p]} \rho_i \mathsf{U}[i,\cdot,\cdot]$ and sends commitments $\tilde{c}_1, \ldots, \tilde{c}_\ell$ to $\tilde{\mathsf{U}}$.
3. $\mathcal{P} \leftrightarrow \mathcal{V}$ compute: $R = r_{lc}^T W$ for $W = [\gamma_x I \,||\gamma_y I \,||\gamma_z I \,|| -(\gamma_x A + \gamma_y B + \gamma_z C)]$, polynomials $R^i, i \in [4p]$ interpolating the slices of R viewed as a $4p \times m \times n$ matrix.
4. $\mathcal{P}$ computes:
   – Polynomials Q_x^i, Q_y^i, Q_z^i, Q^i for $i \in [p]$, where polynomials Q_a^i, $i \in [p]$ correspond to w_a for $a \in \{x,y,z\}$ and polynomials $Q^i, i \in [p]$ correspond to w.
   – $h \times n$ matrices P_{lc} and P_{qd} as "P" matrices for the linear check and quadratic check respectively. Note that p_j polynomial for P_{lc} is given by $p_j(\cdot) = \sum_{i=1}^{p}(R^i(\alpha_j,\cdot)\cdot Q_x^i(\alpha_j,\cdot) + R^{p+i}(\alpha_j,\cdot)\cdot Q_y^i(\alpha_j,\cdot) + R^{2p+i}(\alpha_j,\cdot)\cdot Q_z^i(\alpha_j,\cdot) + R^{3p+i}(\alpha_j,\cdot)\cdot Q^i(\alpha_j,\cdot))$. The p_j polynomials for the matrix P_{qd} are given by $p_j(\cdot) = \sum_{i=1}^{p} r_{qd}[i](Q_x^i(\alpha_j,\cdot)Q_y^i(\alpha_j,\cdot) - Q_z^i(\alpha_j,\cdot))$.
   – Blinding vectors $U_{lc}, U_{qd} \in \mathbb{F}^{2m-1}$ for linear and quadratic check protocols respectively, and commitments c_0, d_0 to vectors U_{lc} and U_{qd}.
5. $\mathcal{P} \to \mathcal{V}$: Commitments $c_0, c_1, \ldots, c_{s+\ell-1}$ for the matrix P_{lc} and commitments $d_0, d_1, \ldots, d_{2\ell-1}$ for matrix P_{qd}.
6. $\mathcal{V} \to \mathcal{P}$: $Q = \{(j_u, k_u) : u \in [t]\}$.
7. $\mathcal{V} \to \pi$: $\{k_u : u \in [t]\}$.
8. $\mathcal{P} \to \mathcal{V}$: $S_u = \mathsf{U}[\cdot, j_u, k_u]$ for $u \in [t]$.
9. $\pi \to \mathcal{V}$: $\pi[\cdot, k_u]$, $u \in [t]$.
10. $\mathcal{P}$ and $\mathcal{V}$ run the linear and quadratic check protocols in parallel, parsing the vectors S_u into X_u, Y_u, Z_u, W_u as needed.
11. Check proximity as: $\prod_{a=1}^{\ell}(\tilde{c}_a)^{\Lambda_{n,\ell}^T[a,k_u]} = \prod_{i=1}^{4p}(\pi[i,k_u])^{\rho_i}$ for $u \in [t]$.
12. $\mathcal{V}$ accepts if all the subprotocols accept.

Fig. 5. GrapheneR1CS Protocol

– *Commitments:* $\mathcal{P}$ sends the following commitments to $\mathcal{V}$: $\{\tilde{c_u}\}_{u\in[\ell]}$, $\{c_u\}_{u\in\{0,1,\ldots,s+\ell-1\}}$, $\{d_u\}_{u\in\{0,1,\ldots,2\ell-1\}}$ and oracle responses: $\pi[\cdot,k_u]$, $\pi_x[\cdot,k_u]$, $\pi_y[\cdot,k_u]$, $\pi_z[\cdot,k_u]$ for all $u \in [t]$
– *Vectors:* $\mathcal{P}$ sends the following vectors to $\mathcal{V}$: $z_{lc} = \beta_{lc}U_{lc} + \overline{P}_{lc}\varphi_{lc}$, $z_{qd} = \beta_{qd}U_{qd} + \overline{P}_{qd}\varphi_{qd}$ $\mathsf{U}[\cdot,\cdot,k_u]$, $\mathsf{U}_x[\cdot,\cdot,k_u]$, $\mathsf{U}_y[\cdot,\cdot,k_u]$, $\mathsf{U}_z[\cdot,\cdot,k_u]$ for all $u \in [t]$.
– *Commitment Randomness:*

- $w_{u_{lc}}$, $w_{u_{qd}}$, w_{lc}, and w_{qd} for $\overline{P}_{lc}[\cdot, k_u]$, $\overline{P}_{qd}[\cdot, k_u]$, z_{lc}, and z_{qd} respectively.
- $O[\cdot, k_u]$, $O_x[\cdot, k_u]$, $O_y[\cdot, k_u]$, and $O_z[\cdot, k_u]$ for $\mathsf{U}[\cdot, \cdot, k_u]$, $\mathsf{U}_x[\cdot, \cdot, k_u]$, $\mathsf{U}_y[\cdot, \cdot, k_u]$, and $\mathsf{U}_z[\cdot, \cdot, k_u]$ respectively.

Proof. **Simulator:** $\mathcal{S}$ picks $\{\gamma_x, \gamma_y, \gamma_z, r_{lc}, r_{qd}, \rho, Q = \{(j_u, k_u)\}_{u\in[t]}, \delta_{lc}, \delta_{qd}, \beta_{lc}, \beta_x, \beta_y, \beta_z, \beta_{qd}, \tau_{qd}, \gamma_{qd}\}$ uniformly at random from their respective domains. Then it picks $\mathsf{U}[\cdot, \cdot, k_u]$ and $\mathsf{U}_a[\cdot, \cdot, k_u]$ for $a \in \{x, y, z\}$, $i \in [p]$, $u \in [t]$ uniformly such that $\mathsf{U}[i, \cdot, k_u] \in \mathsf{RS}_{\boldsymbol{\alpha}}[h, 2m-1]$, $\mathsf{U}_a[i, \cdot, k_u] \in \mathsf{RS}_{\boldsymbol{\alpha}}[h, 2m-1]$ $\forall i \in [p], a \in \{x, y, z\}$. $\mathcal{S}$ picks uniform z_{lc}, z_{qd} from $\mathbb{F}^{2m-1}$ such that $\sum_{j\in[m]} z_{lc}[j] = r_{lc}^T b$ and $\langle \gamma_{qd} || 0^{m-1}, z_{qd}\rangle = 0$. $\mathcal{S}$ picks uniform U_{lc} and U_{qd} from $\mathbb{F}^{2m-1}$ such that $\langle 1^m || 0^{m-1}, U_{lc}\rangle = 0$ and $U_{qd}[j] = 0$ $\forall j \in [m]$. $\mathcal{S}$ picks w_{lc}, $w_{0_{lc}}$, $w_{qd}, w_{0_{qd}}$ and computes the following commitments:

$$c_0 \leftarrow \mathsf{Com}(U_{lc}, w_{0_{lc}}),\ d_0 \leftarrow \mathsf{Com}(U_{qd}, w_{0_{qd}})$$

$$\mathsf{cm}_{lc} \leftarrow \mathsf{Com}(z_{lc}, w_{lc}),\ \mathsf{cm}_{qd} \leftarrow \mathsf{Com}(z_{qd}, w_{qd})$$

$\mathcal{S}$ picks O, O_x, O_y, O_z uniformly at random. $\mathcal{S}$ computes for all $u \in [t]$:

$$\begin{aligned}
\tilde{\mathsf{U}}[\cdot, k_u] &= \sum_{i\in[p]} \rho_i \mathsf{U}[i, \cdot, k_u] + \rho_{p+i}\mathsf{U}_x[i, \cdot, k_u] + \rho_{2p+i}\mathsf{U}_y[i, \cdot, k_u] \\
&\quad + \rho_{3p+i}\mathsf{U}_z[i, \cdot, k_u] \\
\tilde{O}[k_u] &= \sum_{i\in[p]} \rho_i O[i, k_u] + \rho_{p+i} O_x[i, k_u] + \rho_{2p+i} O_y[i, k_u] \\
&\quad + \rho_{3p+i} O_z[i, k_u] \\
&\quad \tilde{c}_{k_u} \leftarrow \mathsf{Com}(\tilde{\mathsf{U}}'[\cdot, k_u], \tilde{O}[k_u]) \\
&\quad \pi[i, k_u] \leftarrow \mathsf{Com}(\mathsf{U}'[i, \cdot, k_u], O[k_u]) \\
&\quad \pi_a[i, k_u] \leftarrow \mathsf{Com}(\mathsf{U}'_a[i, \cdot, k_u], O[k_u])\ \forall a \in \{x, y, z\}
\end{aligned}$$

$\mathcal{S}$ picks $w_{u_{lc}}, w_{u_{qd}}$ uniformly at random for all $u \in [t]$ and computes $c_{k_u} \leftarrow \mathsf{Com}(\overline{P}_{lc}[\cdot, k_u], w_{u_{lc}})$ and $d_{k_u} \leftarrow \mathsf{Com}(\overline{P}_{qd}[\cdot, k_u], w_{u_{qd}})$. $\mathcal{S}$ picks $\tilde{c}_1, \ldots, \tilde{c}_\ell$ such that $\tilde{c}_{k_u} = \prod_{a\in[\ell]} (\tilde{c}_a)^{\Lambda^T_{n,\ell}[a,k_u]}$ $\forall u \in [t]$, it can be done efficiently since the number of unknowns is more than the number of constraints, and the coefficient matrix has full row rank. $\mathcal{S}$ picks $\pi[\cdot, k]$ such that

$$\tilde{c}_k = \prod_{i\in[p]} (\pi[i, k])^{\rho_i} \cdot (\pi_x[i, k])^{\rho_{p+i}} \cdot (\pi_y[i, k])^{\rho_{2p+i}} \cdot (\pi_z[i, k])^{\rho_{3p+i}}$$

for all $k \notin \{k_u : u \in [t]\}$. Finally $\mathcal{S}$ picks $c_1, c_2, \ldots, c_{s+\ell-1}$ and $d_1, d_2, \ldots, d_{2\ell-1}$ subject to the following constraints:

$$\begin{aligned}
c_{k_u} &= \prod_{a\in[s+\ell-1]} (c_a)^{\Lambda^T_{n,s+\ell-1}[a,k_u]}\ \forall u \in [t] \\
\mathsf{cm}_{lc} &= c_0^{\beta_{lc}} \cdot \prod_{a\in[s+\ell-1]} (c_a)^{\varphi_{lc_a}}, \\
d_{k_u} &= \prod_{a\in[2\ell-1]} (d_a)^{\Lambda^T_{n,2\ell-1}[a,k_u]}\ \forall u \in [t] \\
\mathsf{cm}_{qd} &= d_0^{\beta_{qd}} \cdot \prod_{a\in[2\ell-1]} (c_a)^{\varphi_{qd_a}}
\end{aligned}$$

$\mathcal{S}$ can efficiently pick such $c_1, \ldots, c_{s+\ell-1}$ and $d_1, \ldots, d_{2\ell-1}$.

The output of $\mathcal{S}$ is perfectly indistinguishable from the extended view of an honest execution of the protocol. □

C DP-Graphene Security Proofs

In the definition of DPZK, in Section 4, we discussed that according to the corrupted parties, there are 4 possible corruptions:

- All the provers are corrupt and do not have a valid witness. Together they try to cheat so that the verifier accepts the proof. If a DPZK protocol withstands this corruption model, then we call it has *Soundness with Witness Extraction (SoWE)* property.
- The verifier tries to learn about the provers' secrets. A protocol that prevents such an adversary has *Zero-Knowledge (ZK)* property.
- Among all the provers, t are corrupted and try to learn about the honest provers' secrets. Security against such adversary is called *Witness Hiding (WH)* property.
- Along with t provers, the verifier colludes and tries to learn honest provers' secrets. Security against this adversary is called *Witness Hiding with Collusion (WHwC)* property.

Soundness with Witness Extraction: A DPZK protocol has SoWE property if there exists an (expected) polynomial-time extractor $\mathcal{E}$ that interacts with the prover. If the interaction is accepting, $\mathcal{E}$ outputs a valid witness with a very high probability corresponding to the statement. The proof of knowledge property of the single prover version suffices the SoWE property of the DPZK protocol. The following lemma ensures the above claim.

Lemma C.1. *Let $\langle \mathcal{P}, \mathcal{V} \rangle$ be a zero-knowledge protocol and if exists, let Π be the distributed version of $\langle \mathcal{P}, \mathcal{V} \rangle$. Then Π has SoWE property if and only if $\langle \mathcal{P}, \mathcal{V} \rangle$ has proof of knowledge property.*

Proof. Note that, if Π has SoWE property, then $\langle \mathcal{P}, \mathcal{V} \rangle$ has proof of knowledge property, as $\langle \mathcal{P}, \mathcal{V} \rangle$ is a special case of Π, where $\mathsf{N} = 1$.

Let $\langle \mathcal{P}, \mathcal{V} \rangle$ has proof of knowledge property, that is, there exists an extractor $\mathcal{E}_{POK}$ for $\langle \mathcal{P}, \mathcal{V} \rangle$ such that it can extract a witness with very high probability. We will use this to build an extractor $\mathcal{E}_{SoWE}$ for Π. $\mathcal{E}_{SoWE}$ interacts with the provers as a verifier and interacts with $\mathcal{E}_{POK}$ as a prover. Suppose the aggregator A sends m to $\mathcal{E}_{SoWE}$, it forwards the same m to $\mathcal{E}_{POK}$. If $\mathcal{E}_{POK}$ sends a challenge c, $\mathcal{E}_{SoWE}$ publishes the same challenge to all the provers. Furthermore, if $\mathcal{E}_{POK}$ asks to rewind the prover, $\mathcal{E}_{SoWE}$ does the same. Finally, if the interaction is accepting, $\mathcal{E}_{POK}$ outputs a valid witness with a very high probability. Therefore, $\mathcal{E}_{SoWE}$ returns $\mathcal{E}_{POK}$'s output. Hence, $\mathcal{E}_{SoWE}$ outputs a valid witness with a very high probability. □

In Figure 6, Figure 7, we described Distributed Linear Check and Distributed Quadratic Check respectively. The single prover versions of the above checks are described in Figure 3 and Figure 4 respectively. We proved that Linear check and Quadratic Check have proof of knowledge property, see Lemma B.1 and Lemma B.2. Then by Lemma C.1, Distributed Linear Check and Distributed Quadratic Check has Soundness with Witness Extraction property (SoWE). Our protocol Graphene, Figure 5, has the distributed version, DP-Graphene, Figure 8, for R1CS circuits. We proved in Lemma 7.1 that Graphene has Proof of Knowledge property, then by Lemma C.1, DP-Graphene has Soundness with Witness Extraction property.

Lemma C.2. *For the protocol DP-Graphene given in Figure: 8, there exists an (expected) polynomial-time extractor $\mathcal{E}$, if $\mathcal{V}$ accepts the proof given by a set of provers, then $\mathcal{E}$ can output a witness with overwhelming probability.*

Proof. Proof of the above Lemma is evident from Lemma C.1 and Lemma 7.1. □

Zero Knowledge: If a verifier in a DPZK protocol learns nothing but the statement is true, it has the zero-knowledge (ZK) property. In other words, there exists a simulator $\mathcal{S}_{DP}$ that generates a transcript τ, without having a witness such that τ is indistinguishable from a real execution of the protocol. τ consists of the verifier's messages and the provers' messages. The following lemma ensures that the single prover protocol's zero-knowledge property is equivalent to its distributed version's zero-knowledge property.

Lemma C.3. *Let $\langle \mathcal{P}, \mathcal{V} \rangle$ be a single prover zero-knowledge protocol and Π be the distributed version of $\langle \mathcal{P}, \mathcal{V} \rangle$. Then Π has zero-knowledge property if and only if $\langle \mathcal{P}, \mathcal{V} \rangle$ has zero-knowledge property.*

Proof. Note that, if Π has ZK property, then $\langle \mathcal{P}, \mathcal{V} \rangle$ has zero knowledge property, as $\langle \mathcal{P}, \mathcal{V} \rangle$ is a special case of Π, where $\mathsf{N} = 1$.

Let $\langle \mathcal{P}, \mathcal{V} \rangle$ has zero-knowledge property, that is, there is a polynomial-time simulator $\mathcal{S}_{SP}$ which, without knowing the witness, can generate a transcript that is indistinguishable from a transcript of the protocol execution.

Since the verifier's view in the DPZK protocol is the same as the view in the single prover protocol. Therefore the simulator for the single prover version works for the DPZK version as well. □

In Section B.4, we proved that the single prover protocol Graphene described in Figure 5 has zero-knowledge property, see Lemma 7.2.

Lemma C.4. *There exists a simulator $\mathcal{S}$ that outputs a perfectly indistinguishable extended view of the verifier in an honest execution of the protocol DP-*GrapheneR1CS *for $t \leq b$.*

Proof. Proof of the Lemma is evident from Lemma C.3 and Lemma 7.2. □

Witness Hiding and Witness Hiding with Collusion: In our work, we are mainly concern about building a public coin proof system, more specifically NIZK, so that anyone can verify the proof, so any prover can play a verifier's role and learn nothing. Therefore Witness Hiding and Witness Hiding with Collusion are the same in this setting.

We will separate the set of provers into 2 classes: honest(H) and corrupt(C). Note that $|H| + |C| = \mathsf{N}$ and $|C| < t$, for some $t < \mathsf{N}$. In Linear Check, since there is no interaction among the provers other than the aggregator, it suffices to generate the view of the aggregator. For the Quadratic check, we need that the multiplications among the provers are executed securely.

Lemma C.5. *For the Distributed Linear Check protocol (Figure 6), there exists a polynomial-time simulator $\mathcal{S}$ corresponding to a subset of t provers out of N ($t < \mathsf{N}$) provers controlled by a semi-honest PPT adversary $\mathcal{A}$ such that $\mathcal{S}$ generates a view of $\mathcal{A}$ which is indistinguishable from a real view of $\mathcal{A}$.*

Proof. Let C be the set of all corrupt provers and H be the set of all honest provers. Case:1 Let the output of the protocol be "reject", then $\mathcal{S}$ picks all the components of the view of the corrupt parties uniformly at random from the appropriate domains, which is indistinguishable from an honest execution.

Case 2: Let the output of the protocol be "accept". Then input for $\mathcal{S}$ consists of $\langle \mathsf{U} \rangle^{\xi}$ $\forall \xi \in C$ and shares of 0^{2m-1}. Without loss of generality, we can assume that there are 2 parties, C and H. Also, assume that the adversary controls the aggregator. The view of $\mathcal{A}$ consists of:

verifier's randomness: $\{\rho, r, Q = \{(j_u, k_u) : u \in [t]\}, \delta, \beta\}$

commitments: $\langle \mathcal{O} \rangle^H$, $\langle \tilde{c}_1 \rangle^H, \ldots, \langle \tilde{c}_\ell \rangle^H$, $\langle c_0 \rangle^H, \langle c_1 \rangle^H, \ldots, \langle c_{s+\ell-1} \rangle^H, \langle d_0 \rangle^H$

vectors: $\langle \mathsf{U} \rangle^H[\cdot, j_u, k_u], \langle P \rangle^H[\cdot, k_u], \langle z \rangle^H$ and $\langle V_u \rangle^H$

$\mathcal{S}$ does the following:

- $\mathcal{S}$ picks uniformly at random the challenges on behalf of the verifier from the respective domains i.e. $\{\rho, r, Q = \{(j_u, k_u) : u \in [t]\}, \delta, \beta\}$.
- Then $\mathcal{S}$ picks $\mathsf{U}[\cdot, \cdot, k_u]$ such that $\mathsf{U}[i, \cdot, k_u] \in \mathsf{RS}_{\boldsymbol{\alpha}}[h, 2m-1]$ and computes $\langle \mathsf{U} \rangle^H[\cdot, \cdot, k_u] = \mathsf{U}[\cdot, \cdot, k_u] - \langle \mathsf{U} \rangle^C[\cdot, \cdot, k_u]$.
- $\mathcal{S}$ computes $\langle \mathcal{O} \rangle^H[\cdot, k_u]$ which is commitment of $\langle \mathsf{U} \rangle^H[\cdot, \cdot, k_u]$
- $\mathcal{S}$ computes $\langle \tilde{\mathsf{U}} \rangle^H[\cdot, k_u] = \sum_{i \in [p]} \rho_i \langle \mathsf{U} \rangle^H[i, \cdot, k_u]$ and $\langle \tilde{c}_{k_u} \rangle^H$ which is commitment of $\langle \tilde{\mathsf{U}} \rangle^H[\cdot, k_u]$, for all $u \in [t]$.
- $\mathcal{S}$ picks $\langle \tilde{c}_1 \rangle^H, \ldots, \langle \tilde{c}_\ell \rangle^H$ such that $\langle \tilde{c}_{k_u} \rangle^H = \prod_{a \in [\ell]} (\langle \tilde{c}_a \rangle^H)^{\Lambda^T_{n,\ell}[a, k_u]}$ $\forall u \in [t]$. $\mathcal{S}$ can do this efficiently since $\Lambda^T_{n,\ell}$ is a full rank matrix.
- $\mathcal{S}$ picks $\langle \mathcal{O} \rangle^H[\cdot, k]$ such that $\langle \tilde{c}_k \rangle^H = \prod_{i \in [p]} (\langle \mathcal{O} \rangle^H[i, k])^{\rho_i}$ $\forall k \notin \{k_u : u \in [t]\}$.
- $\mathcal{S}$ sends $\langle \mathcal{O} \rangle^H, \langle \tilde{c}_1 \rangle^H, \ldots, \langle \tilde{c}_\ell \rangle^H$ to the aggregator.
- $\mathcal{S}$ computes $\langle P \rangle^H[j, k_u] = \sum_{i \in [p]} R^i(\alpha_j, \eta_{k_u}) \cdot \langle U \rangle^H[i, j, k_u]$ $\forall u \in [t]$ and $\langle c_{k_u} \rangle^H$ which is commitment of $\langle P \rangle^H[\cdot, k_u]$.
- $\mathcal{S}$ picks $\langle c_0 \rangle^H$, $\langle d_0 \rangle^H$ uniformly at random and picks $\langle c_1 \rangle^H, \ldots, \langle c_{s+\ell-1} \rangle^H$ subject to the following constraints:
  $\langle c_{k_u} \rangle^H = \prod_{a \in [s+\ell-1]} (\langle c_a \rangle^H)^{\Lambda^T_{n,s+\ell-1}[a, k_u]}$ $\forall u \in [t]$
  $\langle \mathsf{cm} \rangle^H = (\langle c_0 \rangle^H)^{\beta} \prod_{a \in [s+\ell-1]} (\langle c_a \rangle^H)^{\varphi_a}$
  $\mathcal{S}$ can efficiently perform this since $\Lambda^T_{n,s+\ell-1}$ is full rank matrix.
- $\mathcal{S}$ sends $\langle c_0 \rangle^H, \langle c_1 \rangle^H, \ldots, \langle c_{s+\ell-1} \rangle^H, \langle d_0 \rangle^H$ to the aggregator.
- According to the challenge Q, $\mathcal{S}$ sends $\langle \mathsf{U} \rangle^H[\cdot, j_u, k_u]$, $\langle P \rangle^H[\cdot, k_u]$ to the aggregator.
- $\mathcal{S}$ computes $\langle z \rangle^H$ in the following way: $\mathcal{S}$ computes $\sum_{j \in [m]} \langle z \rangle^C[j] = \sum_{j \in [m]} (\beta \langle P_0 \rangle^C + \langle \overline{P} \rangle^C \varphi + \langle 0 \rangle^C)[j]$ $\mathcal{S}$ knows $\langle 0 \rangle^C$ and from $\langle \mathsf{U} \rangle^C$ and r, $\mathcal{S}$ can obtain $\langle P \rangle^C$ completely. Only unknown term is $\langle P_0 \rangle^C$, but note that $\sum_{j \in [m]} \langle P_0 \rangle^C[j] = 0$, hence $\mathcal{S}$ can compute $\sum_{j \in [m]} \langle z \rangle^C[j] = L$ (say). $\mathcal{S}$ picks $\langle z \rangle^H$ uniformly from $\mathbb{F}^{2m-1}$ satisfying $\sum_{j \in [m]} \langle z \rangle^H[j] = r^T b - L$.
- Corresponding to the challenge δ, $\mathcal{S}$ computes $\langle V_u \rangle^H = \sum_{i \in [p]} \delta_i \cdot \langle \mathsf{U} \rangle^H[i, \cdot, k_u]$.
- Finally, $\mathcal{S}$ sends $\langle z \rangle^H, \langle V_u \rangle^H$ to the aggregator.

The transcript generated by $\mathcal{S}$ is perfectly indistinguishable from an honest execution of the protocol. Hence the linear check described in Figure: 6 has witness hiding property. □

The distributed quadratic check protocol described in Figure: 7 has witness hiding property if the multiplication of the secrets held by the provers is performed securely. Let Π_{Mult} is a secure $\mathcal{F}_{\mathsf{Mult}}$ protocol used there, which can withstand t corrupted parties, then distributed quadratic has witness hiding property against t corrupted provers. The following lemma ensures the claim.

Lemma C.6. *Let Π_{Mult} be a N-party t-secure protocol where the inputs of the ξth party are $\langle \mathbf{a} \rangle^{\xi}, \langle \mathbf{b} \rangle^{\xi}$ and the protocol outputs $\langle (\sum_{\xi \in [\mathsf{N}]} \langle \mathbf{a} \rangle^{\xi}) \cdot (\sum_{\xi \in [\mathsf{N}]} \langle \mathbf{b} \rangle^{\xi}) \rangle^{\xi}$ to $\mathcal{P}_{\xi}$, where $\mathbf{a}, \mathbf{b}$ are vectors of length $2m-1$. Then corresponding to a subset of t provers corrupted semi-honestly by an PPT adversary $\mathcal{A}$, there exists a polynomial-time simulator $\mathcal{S}$ that takes private values of the provers controlled by $\mathcal{A}$ as input and the output is indistinguishable from the view of $\mathcal{A}$ in the Distributed Quadratic Check protocol, Figure 7.*

Proof. Since Π_{Mult} is a secure protocol, then there is a simulator $\mathcal{S}_M$ which can generate a view that is indistinguishable from an honest execution of the protocol. $\mathcal{S}$ takes inputs of the corrupt parties and output of the protocol to simulate the view. Similar to the Lemma: C.5, we will consider C as the set of corrupt parties and H as the set of honest parties. We will follow the same notations.

The view of $\mathcal{A}$ in the distributed quadratic check consists of :

Verifier's Randomness: $\rho, r, Q, \tau, \gamma, \beta, \delta, \beta_x, \beta_y, \beta_z$

Commitments: $\langle \mathcal{O}_a \rangle^H \ \forall a \in \{x, y, z\}$, $\langle \tilde{c_1} \rangle^H$, $\ldots$, $\langle \tilde{c}_\ell \rangle^H$, $\langle c_0 \rangle^H, \ldots, \langle c_{2\ell-1} \rangle^H$

Vectors:$\langle \mathsf{U}_a \rangle^H[\cdot, \cdot, k_u] \ \forall a \in \{x, y, z\}$, $\langle P \rangle^H[\cdot, k_u]$, $\langle z \rangle^H$, $\langle V_u \rangle^H$.

$\mathcal{S}$ does the following:

- $\mathcal{S}$ picks $\rho, r, Q, \tau, \gamma, \beta, \delta, \beta_x, \beta_y, \beta_z$ uniformly at random from their respective domains.
- $\mathcal{S}$ picks $\mathsf{U}_a[\cdot, \cdot, k_u] \in \mathsf{RS}_{\boldsymbol{\alpha}}[h, 2m-1]$ for all $a \in \{x, y, z\}$ and computes $\langle \mathsf{U}_a \rangle^H[\cdot, \cdot, k_u] = \mathsf{U}_a[\cdot, \cdot, k_u] - \langle \mathsf{U}_a \rangle^C[\cdot, \cdot, k_u] \ \forall a \in \{x, y, z\}, \ u \in [t]$.
- $\mathcal{S}$ computes $\langle \mathcal{O}_a \rangle^H[\cdot, k_u]$ which is commitment of $\langle \mathsf{U}_a \rangle^H[\cdot, \cdot, k_u] \ \forall u \in [t], a \in \{x, y, z\}$.
- $\mathcal{S}$ computes $\langle \tilde{\mathsf{U}} \rangle^H[\cdot, k_u] = \sum_{i \in [p]} \rho_i \langle \mathsf{U}_x \rangle^H[i, \cdot, k_u] + \rho_{p+i} \langle \mathsf{U}_y \rangle^H[i, \cdot, k_u] + \rho_{2p+i} \langle \mathsf{U}_z \rangle^H[i, \cdot, k_u]$ and $\langle c_{\tilde{k}_u} \rangle^H$, which is commitment of $\langle \tilde{\mathsf{U}} \rangle^H[\cdot, k_u]$, for all $u \in [t]$.
- $\mathcal{S}$ picks $\langle \tilde{c_1} \rangle^H, \ldots, \langle \tilde{c}_\ell \rangle^H$ such that $\langle \tilde{c}_{k_u} \rangle^H = \prod_{a \in [\ell]} (\langle \tilde{c}_a \rangle^H)^{\Lambda^T_{n,\ell}[a, k_u]} \ \forall u \in [t]$. $\mathcal{S}$ can pick such $\tilde{c}$ efficiently due to the fact that $\Lambda^T_{n,\ell}$ is a full rank matrix.
- $\mathcal{S}$ picks $\langle \mathcal{O}_a \rangle^H[\cdot, k]$ such that $\langle \tilde{c_k} \rangle^H = \prod_{i \in [p]} (\langle \mathcal{O}_x \rangle^H[i, k])^{\rho_i} \cdot (\langle \mathcal{O}_y \rangle^H[i, k])^{\rho_{p+i}} \cdot (\langle \mathcal{O}_z \rangle^H[i, k])^{\rho_{2p+i}} \ \forall k \notin \{k_u : u \in [t]\}$.
- $\mathcal{S}$ sends $\langle \mathcal{O}_a \rangle^H$ for all $a \in \{x, y, z\}$ and $\langle \tilde{c_1} \rangle^H, \ldots, \langle \tilde{c}_\ell \rangle^H$ to the aggregator.
- $\mathcal{S}$ picks z from $\mathbb{F}^{2m-1}$ such that $z[j] = 0 \ \forall j \in [m]$. Then $\mathcal{S}$ splits z into $\langle z \rangle^H$ and $\langle z \rangle^C$ such that $\langle z \rangle^H + \langle z \rangle^C = z$.
- $\mathcal{S}$ computes $\langle \mathsf{U}_x \cdot \mathsf{U}_y \rangle^C[\cdot, \cdot, k_u]$ from $\langle \mathsf{U}_x \rangle^C, \langle \mathsf{U}_y \rangle^C$ and $\langle \mathsf{U}_x \rangle^H[\cdot, \cdot, k_u]$ $\langle \mathsf{U}_y \rangle^H[\cdot, \cdot, k_u]$ and picks $\langle \mathsf{U}_x \cdot \mathsf{U}_y \rangle^C[i, \cdot, k] \in \mathsf{RS}_{\boldsymbol{\alpha}}[h, 2m-1]$ uniformly for $k \notin \{k_u : u \in [t]\}$ and compute such that $\{(\sum_{i \in [p]} r_i \langle \mathsf{U}_x \cdot \mathsf{U}_y \rangle^C[i, \cdot, \cdot])\varphi\}[j] = z[j] - \langle z \rangle^H[j] + \{(\sum_{i \in [p]} r_i \langle \mathsf{U}_z \rangle^C[i, \cdot, \cdot])\varphi\}[j] \ \forall j \in [m]$. It is easy to see that picking such $\langle \mathsf{U}_x \cdot \mathsf{U}_y \rangle^C[\cdot, \cdot, k]$ can be done efficiently.
- $\mathcal{S}$ calls $\mathcal{S}_M$ on input $\langle \mathsf{U}_x \rangle^C, \langle \mathsf{U}_y \rangle^C$ and $\langle \mathsf{U}_x \cdot \mathsf{U}_y \rangle^C$ and gets a valid transcript of the interaction among the provers.
- $\mathcal{S}$ computes $\langle P \rangle^H[\cdot, k_u]$ from $\langle \mathsf{U}_x \rangle^C$, $\langle \mathsf{U}_y \rangle^C$ and $\langle \mathsf{U}_x \rangle^H[\cdot, \cdot, k_u]$, $\langle \mathsf{U}_y \rangle^H[\cdot, \cdot, k_u]$ for all $u \in [t]$ and $\langle c_{k_u} \rangle^H$, commitments of $\langle P \rangle^H[\cdot, k_u]$.
- picks $\langle c_0 \rangle^H$, $\langle d_0 \rangle^H$ uniformly and picks $\langle c_1 \rangle^H, \ldots, \langle c_{2\ell-1} \rangle^H$ such that: $\langle c_{k_u} \rangle^H = \prod_{a \in [2\ell-1]} (\langle c_a \rangle^H)^{\Lambda^T_{n,2\ell-1}[a, k_u]} \ \forall u \in [t]$ $\mathsf{cm} = (\langle c_0 \rangle^H)^\beta \prod_{a \in [2\ell-1]} (\langle c_a \rangle^H)^{\varphi_a}$
- $\mathcal{S}$ sends $\langle c_0 \rangle^H, \langle c_1 \rangle^H, \ldots, \langle c_{2\ell-1} \rangle^H, \langle d_0 \rangle^H$ to the aggregator.
- $\mathcal{S}$ sends $\langle \mathsf{U}_a \rangle^H[\cdot, j_u, k_u]$ for all $a \in \{x, y, z\}$ and $\langle P \rangle^H[\cdot, k_u]$, as responses to the Q.
- $\mathcal{S}$ finally sends $\langle z \rangle^H$ and $\langle V_u \rangle^H$ corresponding to the challenges δ and $\beta_x, \beta_y, \beta_z$.

The view generated by $\mathcal{S}$ is indistinguishable from a transcript of an honest execution. □

Since DP-Graphene is the combination of linear check and quadratic check, the simulator for the complete protocol follows the same strategies of the simulators described in Lemma: C.5 and C.6. So, the DP-Graphene described in Figure: 8 has the witness hiding property.

Lemma C.7. *Let Π_{Mult} be a secure protocol described in Lemma: C.6, and Π_{Mult} is used in DP-*GrapheneR1CS *in step 5, then $\exists$ polynomial-time simulator $\mathcal{S}$ such that $\mathcal{S}$ can generate a view of the adversary $\mathcal{A}$ which is indistinguishable from the view of $\mathcal{A}$ in a real execution of the protocol.*

Proof. $\mathcal{S}$ starts with picking the verifier's randomness and does the same as simulators for the Linear check, Lemma: C.5 and the Quadratic check, Lemma: C.6 do. Therefore the transcript generated by $\mathcal{S}$ is indistinguishable from the adversary's view. □

D DPZK from existing protocols

D.1 DPZK of Bulletproof for R1CS

In [9], [12], authors present a proof for a Hadamard-product relation. Suppose C is a circuit of size N. In their formulation, $\boldsymbol{a}_L$, $\boldsymbol{a}_R$ and $\boldsymbol{a}_O$ denote the vectors corresponding to the left wire, right wire and output wire respectively. Then, $\boldsymbol{a}_L \circ \boldsymbol{a}_R = \boldsymbol{a}_O$ holds and satisfy the following linear constraints:

$$\langle \boldsymbol{w}_{L,q}, \boldsymbol{a}_L \rangle + \langle \boldsymbol{w}_{R,q}, \boldsymbol{a}_R \rangle + \langle \boldsymbol{w}_{O,q}, \boldsymbol{a}_O \rangle = c_q \, \forall q \in [Q]$$

with $\boldsymbol{w}_{L,q}, \boldsymbol{w}_{R,q}, \boldsymbol{w}_{O,q} \in \mathbb{Z}_p^N$ and $c_q \in \mathbb{Z}_p$. Consider $W_A = [\boldsymbol{w}_{A,1}, \ldots, \boldsymbol{w}_{A,Q}]^T$, for $A \in \{L, R, O\}$ and $\boldsymbol{c} = [c_1, \ldots, c_q]^T$

In [12], these above checks are reduced to a single inner product argument. $\mathcal{P}$ commits to the input vectors $\boldsymbol{a}_L, \boldsymbol{a}_R$, output vector $\boldsymbol{a}_O$ and sends the commitment values A_I and A_O to $\mathcal{V}$. $\mathcal{P}$ picks s_L and s_R uniformly at random and computes the commitment S and sends to $\mathcal{V}$. $\mathcal{V}$ gives random challenges y, z from $\mathbb{Z}_p^*$ to the prover. Then $\mathcal{P}$ computes vector polynomials $l(X)$ and $r(X)$ using $\boldsymbol{a}_L, \boldsymbol{a}_R, \boldsymbol{a}_O, s_L, s_R, y, z$ and other public vectors, and

computes a polynomial $t(X) = \langle l(X), r(X)\rangle$. The construction of $l(X)$ and $r(X)$ is such that the co-efficient of X^2 in $t(X)$ is independent of $\boldsymbol{a}_L, \boldsymbol{a}_R, \boldsymbol{a}_O, s_L, s_R$, which can be computed by $\mathcal{V}$, if $\boldsymbol{a}_L, \boldsymbol{a}_R, \boldsymbol{a}_O$ satisfy the Hadamard-product relation and linear constraints. To prove this, $\mathcal{P}$ commits to all the co-efficients of $t(X)$ other than the co-efficient of X^2, and verifier gives a random point x to evaluate $\boldsymbol{l} = l(x)$ and $\boldsymbol{r} = r(x)$. Which reduces to an inner product check whose witness is $\boldsymbol{l}$ and $\boldsymbol{r}$, where $\boldsymbol{l}$ and $\boldsymbol{r}$ are vectors of length N.

We give a DPZK version of Bulletproofs, where $\mathcal{P}_\xi$ starts the protocol with $\langle \boldsymbol{a}_L\rangle^\xi, \langle \boldsymbol{a}_R\rangle^\xi, \langle \boldsymbol{a}_O\rangle^\xi \ \forall \xi \in [\mathsf{N}]$ such that $\sum_{\xi\in[\mathsf{N}]}\langle \boldsymbol{a}_A\rangle^\xi = \boldsymbol{a}_A \ \forall A \in \{L, R, O\}$. A is an aggregator. $\mathcal{P}_\xi$ computes the commitment $\langle A_I\rangle^\xi, \langle A_O\rangle^\xi, \langle S\rangle^\xi$ and sends these values to A. A combines and sends A_I, A_O and S to $\mathcal{V}$. The verifier broadcasts the random challenge $y, z \leftarrow_{\$} \mathbb{Z}_p^*$. Then each prover computes $\langle l\rangle^\xi(X)$ and $\langle r\rangle^\xi(X)$. All the provers interact securely to compute shares of $t(x) = \langle l(X), r(X)\rangle$, where $l(X) = \sum_{\xi\in[\mathsf{N}]}\langle l\rangle^\xi(X)$ and $r(X) = \sum_{\xi\in[\mathsf{N}]}\langle r\rangle^\xi(X)$, i.e., output for $\mathcal{P}_\xi$ is $\langle t\rangle^\xi(X)$ such that $\sum_{\xi\in[\mathsf{N}]}\langle t\rangle^\xi(X) = t(X)$. $\mathcal{P}_\xi$ commits to all the coefficients of $\langle t\rangle^\xi(X)$ other than the coefficient of X^2. Sends these committed values to A. A combines and sends them to $\mathcal{V}$. $\mathcal{V}$ sends a random x. $\mathcal{P}_\xi$ evaluates $\langle l\rangle^\xi(x) = \langle \boldsymbol{l}\rangle^\xi$ and $\langle r\rangle^\xi(x) = \langle \boldsymbol{r}\rangle^\xi$ and sends these values to A. A computes $\boldsymbol{l} = \sum_{\xi\in[\mathsf{N}]}\langle \boldsymbol{l}\rangle^\xi$ and $\boldsymbol{r} = \sum_{\xi\in[\mathsf{N}]}\langle \boldsymbol{r}\rangle^\xi$. Finally, A and $\mathcal{V}$ run the inner product protocol, same as the single prover protocol, where the witness is $\boldsymbol{l}$ and $\boldsymbol{r}$. Since, $\boldsymbol{l}$ and $\boldsymbol{r}$ are vectors of length N, MPC is required for N multiplications or in other words, a MPC for a depth 1 circuit of size N.

D.2 DPZK of Spartan for R1CS

We will describe the distributed version of the interactive proof given in Spartan [48]. We do not have a proof of privacy of the distributed version of the protocol.

Let C be an R1CS circuit of size N. For every R1CS circuit $\exists$ matrices A, B, C and public input io which defines the circuit. If the instance is true, then $\exists$ a witness $\mathsf{w} \in \mathbb{F}^N$ such that $Az \circ Bz = Cz$ where $z = (io, 1, \mathsf{w})$ is the extended witness of C.

Now, the prover wants to convince the verifier that the prover knows z such that $Az \circ Bz = Cz$ holds. To that the prover defines a polynomial

$$\tilde{F}_{io}(x) = \left(\sum_{y\in\{0,1\}^s} \tilde{A}(x,y)\tilde{Z}(y)\right)\left(\sum_{y\in\{0,1\}^s} \tilde{B}(x,y)\tilde{Z}(y)\right) - \left(\sum_{y\in\{0,1\}^s} \tilde{C}(x,y)\tilde{Z}(y)\right)$$

Where $\tilde{A}(x,y) = A(x,y) \ \forall x, y$. $A(\cdot,\cdot)$ is defined as a function such that $A(x,y)$ is the $(x,y)th$ entry of the matrix A. Similarly, $B(\cdot,\cdot)$ and $C(\cdot,\cdot)$. And $\tilde{A}$, $\tilde{B}$, $\tilde{C}$ are the low-degree polynomial extensions of A, B, C respectively, defined in Spartan[48]. And $\tilde{Z}(y) = Z(y) \ \forall x$. $Z(\cdot)$ is defined as a function such that $Z(y)$ is the yth entry of the vector z, and $\tilde{Z}$ is the low-degree polynomial extensions of Z, defined in Spartan[48]. If $Az \circ Bz = Cz$ holds then $\tilde{F}_{io}(x) = 0 \ \forall x \in \{0,1\}^s$. That means the verifier needs to check for all $x \in \{0,1\}^s$ $\tilde{F}_{io}(x) = 0$. To get rid of such check a new polynomial $Q_{io}(t) = \sum_{x\in\{0,1\}^s} \tilde{F}_{io}(x) \cdot eq(t,x)$ where $eq(t,x) = \prod_{i\in[s]}[t_i \cdot x_i + (1-t_i)(1-x_i)]$. Note that if $\tilde{F}_{io}(x) = 0 \ \forall x \in \{0,1\}^s$ then Q_{io} is a zero polynomial i.e. $Q_{io}(\tau) = 0$ for any random τ.

Define $\mathcal{G}_{io,\tau}(x) = \tilde{F}_{io}(x) \cdot eq(\tau, x)$. Then $\sum_{x\in\{0,1\}^s} \mathcal{G}_{io,\tau}(x) = Q_{io}(\tau)$. Therefore, it is enough to check that $\sum_{x\in\{0,1\}^s} \mathcal{G}_{io,\tau}(x) = 0$. This check is done using the sum-check protocol described in Spartan [48].

In the DPZK version of Spartan [48] each prover holds a share of the witness w, say $\mathcal{P}_\xi$ has $\langle \mathsf{w}\rangle^\xi$. In other words, Z is distributedly shared among the provers. In the above protocol described in Spartan [48] only $\mathcal{G}_{io,\tau}$ generation is required interaction among the provers. Remaining all the messages can be generated by each prover locally, and an aggregator can combine the messages to obtain the corresponding message. Note that, $\mathcal{G}_{io,\tau}(x)$ computation requires $O(n^2)$ multiplications which are shared across the provers.

E Shared circuit complexity

E.1 Reducing MPC Overhead for Small Shared Circuits

Let C be a circuit with N wires (we assume a unique incoming wire for each input gate), let $\mathsf{w} = (w_1, \ldots, w_N)$ denote the vector of wire values in a satisfying assignment according to some ordering on the wires. We consider the case, when each input wire is "assigned" to a unique prover. Let M be the number of multiplication gates in C, and let A, B, C be $M \times N$ matrices such that

$\mathsf{w}_x = A\mathsf{w}$, $\mathsf{w}_y = B\mathsf{w}$ and $\mathsf{w}_z = C\mathsf{w}$ are vectors of *left*, *right* and *out* values of multiplication gates. For $i \in [m]$, we say that multiplication gate i is *isolated* if $\mathsf{w}_x[i], \mathsf{w}_y[i]$ and $\mathsf{w}_z[i]$ depend on inputs of only one prover. We call the remaining multiplication gates as *shared*. Let N_s be the number of shared gates. Without loss of generality, we can assume that the gates $1, \ldots, N_s$ are shared (rows of A, B, C can be permuted suitably). Now an additive sharing of the extended witness may be obtained by (i) each prover locally computing the wires for isolated gates from their inputs, while setting the shares of other parties for these wires to be 0 and (ii) running a secret sharing MPC on the subcircuit containing shared gates. Let $p_s = \lceil N_s/ms \rceil$. Then the multiplication MPC Mult in distributed quadratic check can be restricted to obtaining shares $\langle U_x[i,j,k].U_y[i,j,k] \rangle^{\xi}$ for $i \in [p_s]$, as each prover can canonically the shares of other slices of U_x and U_y. For slices, which depend on the provers inputs, the prover computes randomized encoding of the slice, for other slices (where its share is 0) it computes a deterministic encoding by setting the buffer columns to 0. This gives an MPC of depth 1 on circuit of size $O(\max(N_s, mn))$.

F Parameters

F.1 Performance Parameters for Ligero and Bulletproofs

Ligero: Let m, s be such that $N = ms$. Let t be a parameter, and let $\ell = s + t$, $n = O(\ell)$ and $e \leq (n - \ell)/4$ Then the performance parameters for Ligero are then given by: $c_{\mathsf{zk}} = n + 6\ell + 4s + 4mt + 5t - 5$, $t_{\mathcal{P}} = O(N)\mathsf{C}_{\mathbb{F}} + 4m(\mathsf{C}_{\mathsf{FFT}}(s) + \mathsf{C}_{\mathsf{FFT}}(n))$, $t_{\mathcal{V}} = O(N) + 4m\mathsf{C}_{\mathsf{FFT}}(s)$, $\kappa_{lg} = (1 - e/n)^t + 5((2\ell + e)/n)^t$.
Bulletproofs: $c_{\mathsf{zk}} = 2\log(N) + 13$, $t_{\mathcal{P}} = 9N\mathsf{C}_{\mathsf{EXP}} + 2\mathsf{C}_{\mathsf{MXP}}(2N) + 3\mathsf{C}_{\mathsf{MXP}}(N)$, $t_{\mathcal{V}} = N\mathsf{C}_{\mathsf{EXP}} + \mathsf{C}_{\mathsf{MXP}}(2N)$, $\kappa_{bp} = N/|\mathbb{F}|$.

For computing prover communication in distributed setting we use the expressions: $c_{\mathsf{pr}} = MPC(\mathsf{N}, \max(N_s, 4m\ell), 1) + \mathsf{N} \times [(4pn + 5\ell + s + 2)\mathsf{B}_{\mathbb{G}} + (4pt + 4m + h)\mathsf{B}_{\mathbb{F}}]$, for DP-Graphene and $c_{\mathsf{pr}} = MPC(\mathsf{N}, N_s, 1) + \mathsf{N} \times [8\mathsf{B}_{\mathbb{G}} + N\mathsf{B}_{\mathbb{F}}]$ for Bulletproofs. Here $MPC(\mathsf{N}, N_s, 1)$ denotes communication in an N party MPC with circuit size N_s and depth as 1.

$\mathsf{DistLinearCheck}(\mathsf{pp}, A \in \mathcal{M}_{M,N}, b \in \mathbb{F}^M, [\pi]; \langle \mathsf{U} \rangle^\xi, \langle 0^{2m-1} \rangle^\xi)$:
Relation: $\mathsf{U} = \mathsf{Open}(\pi) \wedge A\mathsf{w} = b$ for $\langle \mathsf{w} \rangle^\xi = \mathsf{Dec}(\langle \mathsf{U} \rangle^\xi)$ for all $\xi \in [\mathsf{N}]$ and $\sum_{\xi \in [\mathsf{N}]} \langle \mathsf{w} \rangle^\xi = \mathsf{w}$.

1. $\mathcal{V} \to \mathcal{P}_\xi$: $\rho \leftarrow_\$ \mathbb{F}^p$.
2. $\mathcal{P}_\xi$ computes: $\langle \tilde{\mathsf{U}} \rangle^\xi = \sum_{i \in [p]} \rho_i \langle \mathsf{U} \rangle^\xi [i, \cdot, \cdot]$, commitments $\langle \tilde{c}_1 \rangle^\xi, \ldots, \langle \tilde{c}_\ell \rangle^\xi$ as $\langle \tilde{c} \rangle^\xi_k = \prod_{i \in [p]} (\langle \mathcal{O}[i,k] \rangle^\xi)^{\rho_i} \ \forall k \in [\ell]$.
3. $\mathcal{P}_\xi \to \mathsf{A}$: $\langle \tilde{\boldsymbol{c}} \rangle^\xi = (\langle \tilde{c}_1 \rangle^\xi, \ldots, \langle \tilde{c}_\ell \rangle^\xi)$.
4. $\mathsf{A} \to \mathcal{V}$: $\tilde{\boldsymbol{c}} = \mathsf{Combine}(\langle \tilde{\boldsymbol{c}} \rangle^\xi)$.
5. $\mathcal{V} \to \mathcal{P}_\xi$: $r \leftarrow_\$ \mathbb{F}^M$.
6. $\mathcal{P}_\xi$ and $\mathcal{V}$ compute: Polynomials R^i, $i \in [p]$ interpolating $R = r^T A$ as in Section 7.1.
7. $\mathcal{P}_\xi$ computes: Matrix $\langle P \rangle^\xi$ from R and $\langle \mathsf{U} \rangle^\xi$ as described in Section 7.1. Samples $\langle P_0 \rangle^\xi \leftarrow_\$ \mathbb{F}^{2m-1}$, $\langle \omega_0 \rangle^\xi \leftarrow_\$ \mathbb{F}$ and $\langle c_0 \rangle^\xi \leftarrow \mathsf{Com}(\langle P_0 \rangle^\xi, \langle \omega_0 \rangle^\xi)$, and $\langle d_0 \rangle^\xi \leftarrow \mathsf{Com}(\langle 0^{2m-1} \rangle^\xi, \langle o \rangle^\xi)$ where $\langle o \rangle^\xi \leftarrow_\$ \mathbb{F}$. Computes commitments $\langle c_1 \rangle^\xi, \ldots, \langle c_{s+\ell-1} \rangle^\xi$ from $\langle P \rangle^\xi$.
8. $\mathcal{P}_\xi \to \mathsf{A}$: $\langle c_0 \rangle^\xi, \langle c_1 \rangle^\xi, \ldots, \langle c_{s+\ell-1} \rangle^\xi, \langle d_0 \rangle^\xi$.
9. $\mathsf{A} \to \mathcal{V}$: $c_k = \mathsf{Combine}(\langle c_k \rangle^\xi) \ \forall k \in [s + \ell - 1]$ and sends $c_0, c_1, \ldots, c_{s+\ell-1}$.
10. $\mathcal{V} \to \mathcal{P}_\xi$: $Q = \{(j_u, k_u) : u \in [t]\}$ for $Q \leftarrow_\$ [h] \times [n]$ for $u \in [t]$.
11. $\mathcal{V} \to \pi$: $\{k_u : u \in [t]\}$.
12. $\mathcal{P}_\xi \to \mathsf{A}$: $\langle X_u \rangle^\xi = \langle \mathsf{U} \rangle^\xi [\cdot, j_u, k_u], \langle P_u \rangle^\xi = \langle P \rangle^\xi [\cdot, k_u]$ for $u \in [t]$.
13. $\mathsf{A} \to \mathcal{V}$: $X_u = \sum_{\xi \in [\mathsf{N}]} \langle X_u \rangle^\xi, P_u = \sum_{\xi \in [\mathsf{N}]} \langle P_u \rangle^\xi$ and sends X_u for $u \in [t]$.
14. $\pi \to \mathcal{V}$: $\pi[\cdot, k_u]$ for $u \in [t]$.
15. $\mathcal{V} \to \mathcal{P}_\xi$: $\delta \leftarrow_\$ \mathbb{F}^p$, $\beta \leftarrow_\$ \mathbb{F} \backslash \{0\}$.
16. $\mathcal{P}_\xi \to \mathsf{A}$: $\langle z \rangle^\xi = \beta \langle P_0 \rangle^\xi + \langle \overline{P} \rangle^\xi \varphi + \langle 0^{2m-1} \rangle^\xi$, $\langle V_u \rangle^\xi = \sum_{i \in [p]} \delta_i \langle \mathsf{U} \rangle^\xi [i, \cdot, k_u]$ and sends $\langle z \rangle^\xi$, $\langle V_u \rangle^\xi$.
17. A computes $z = \sum_{\xi \in [\mathsf{N}]} \langle z \rangle^\xi$, $V_u = \sum_{\xi \in [\mathsf{N}]} \langle V_u \rangle^\xi$.
18. A and $\mathcal{V}$ run inner product arguments to check:
    (a) $\mathsf{InnerProduct}(\mathsf{pp}, \mathbf{1}^T_{j_u} \Lambda_{h,2m-1}, \mathsf{cm}_u, v_u; \overline{P}[\cdot, k_u])$ for $u \in [t]$ where $\mathsf{cm}_u = \prod_{a=1}^{s+\ell-1} (c_a)^{\Lambda_{n,s+\ell-1}[a,k_u]}$, $v_u = \sum_{i=1}^p R^i(\alpha_{j_u}, \eta_{k_u}) X_u[i]$ (check consistency of P with π).
    (b) $\mathsf{InnerProduct}(\mathsf{pp}, 1^m || 0^{m-1}, \mathsf{cm}, r^T b; z)$ where $z = \beta P_0 + \overline{P}\varphi$ and $\mathsf{cm} = (c_0)^\beta \cdot \prod_{a=1}^{s+\ell-1} (c_a)^{\varphi_a}$ (check the condition $r^T A w = r^T b$).
    (c) $\mathsf{InnerProduct}(\mathsf{pp}, \mathbf{1}^T_{j_u} \Lambda_{h,m}, C_u, w_u; V_u)$ for $u \in [t]$ where $C_u = \prod_{i=1}^p (\pi[i, k_u])^{\delta_i}$ and $w_u = \sum_{i \in [p]} \delta_i X_u[i]$(consistency of X_u with π).
19. $\mathcal{V}$ checks: $\prod_{a=1}^{\ell} (\tilde{c}_a)^{\Lambda^T_{n,\ell}[a,k_u]} = \prod_{i=1}^p (\pi[i, k_u])^{\rho_i}$ for $u \in [t]$ (check proximity of U to $\mathcal{W}_1$).

Fig. 6. Distributed Linear Check Protocol

$\mathsf{DistQuadraticCheck}(\mathsf{pp}, [\pi_x], [\pi_y], [\pi_z];\ \langle \mathsf{U}_x\rangle^\xi, \langle \mathsf{U}_y\rangle^\xi, \langle \mathsf{U}_z\rangle^\xi\ \langle 0^{2m}\rangle^\xi)$:
Relation: $[\mathsf{U}_x||\mathsf{U}_y||\mathsf{U}_z] = \mathsf{Open}(\pi) \wedge \mathsf{w}_x \circ \mathsf{w}_y = \mathsf{w}_z$ for $\langle \mathsf{w}_a\rangle^\xi = \mathsf{Dec}(\langle \mathsf{U}_a\rangle^\xi)$ for all $\xi \in [\mathsf{N}]$ and $\sum_{\xi\in[\mathsf{N}]}\langle \mathsf{w}_a\rangle^\xi = \mathsf{w}_a\ \forall a \in \{x, y, z\}$.

1. $\mathcal{V} \to \mathcal{P}_\xi$: $\rho \leftarrow_\$ \mathbb{F}^{3p}$.
2. $\mathcal{P}_\xi$ computes: $\langle \tilde{\mathsf{U}}\rangle^\xi = \sum_{i=1}^{p}[\rho_i\langle \mathsf{U}_x\rangle^\xi[i,\cdot,\cdot] + \rho_{p+i}\langle \mathsf{U}_y\rangle^\xi[i,\cdot,\cdot] + \rho_{2p+i}\langle \mathsf{U}_z\rangle^\xi[i,\cdot,\cdot]]$, commitments $\langle \tilde{c}_1\rangle^\xi, \ldots, \langle \tilde{c}_\ell\rangle^\xi$ as $\langle \tilde{c}\rangle_k^\xi = \prod_{i=1}^{p}(\langle \mathcal{O}_x\rangle^\xi[i,k])^{\rho_i} \cdot (\langle \mathcal{O}_y\rangle^\xi[i,k])^{\rho_{p+i}} \cdot (\langle \mathcal{O}_z\rangle^\xi[i,k])^{\rho_{2p+i}}\ \forall k \in [\ell]$.
3. $\mathcal{P}_\xi \to \mathsf{A}$: $\langle \tilde{c}\rangle^\xi = (\langle \tilde{c}_1\rangle^\xi, \ldots, \langle \tilde{c}_\ell\rangle^\xi)$.
4. $\mathsf{A} \to \mathcal{V}$: $\tilde{c} = \mathsf{Combine}(\langle \tilde{c}\rangle^\xi)$.
5. $\mathcal{V} \to \mathcal{P}_\xi$: $r \leftarrow_\$ \mathbb{F}^p$.
6. Provers invoke $\mathcal{F}_{\mathsf{Mult}}$: $\langle \mathsf{U}_x.\mathsf{U}_y\rangle^\xi \leftarrow \mathsf{Mult}(\langle \mathsf{U}_x\rangle^\xi, \langle \mathsf{U}_y\rangle^\xi)$ to obtain shares of the hadamard product of the encodings.
7. $\mathcal{P}_\xi$ computes: Matrix $\langle P\rangle^\xi$ from r and $\langle \mathsf{U}_x.\mathsf{U}_y\rangle^\xi$, $\langle \mathsf{U}_z\rangle^\xi$ as described in Section 7.2. Samples $\langle P_0\rangle^\xi \leftarrow_\$ \mathbb{F}^{2m-1}$ such that $P_0[j] = 0 \forall j \in [m]$, $\langle \omega_0\rangle^\xi \leftarrow_\$ \mathbb{F}$ and $\langle c_0\rangle^\xi \leftarrow \mathsf{Com}(\langle P_0\rangle^\xi, \langle \omega_0\rangle^\xi)$, and $\langle d_0\rangle^\xi \leftarrow \mathsf{Com}(\langle 0^{2m-1}\rangle^\xi, \langle o\rangle^\xi)$ where $\langle o\rangle^\xi \leftarrow_\$ \mathbb{F}$. Computes commitments $\langle c_1\rangle^\xi, \ldots, \langle c_{2\ell-1}\rangle^\xi$ from $\langle P\rangle^\xi$.
8. $\mathcal{P}_\xi \to \mathsf{A}$: $\langle c_0\rangle^\xi, \langle c_1\rangle^\xi, \ldots, \langle c_{2\ell-1}\rangle^\xi, \langle d_0\rangle^\xi$.
9. $\mathsf{A} \to \mathcal{V}$: $c_k = \mathsf{Combine}(\langle c_k\rangle^\xi)\ \forall k \in [2\ell-1]$ and sends $c_0, c_1, \ldots, c_{2\ell-1}$.
10. $\mathcal{V} \to \mathcal{P}_\xi$: $Q = \{(j_u, k_u) : u \in [t]\}$ for $Q \leftarrow_\$ [h] \times [n]$ for $u \in [t]$. And $\tau \leftarrow_\$ \mathbb{F}^s$, $\gamma \leftarrow_\$ \mathbb{F}^m, \beta \leftarrow_\$ \mathbb{F}^*$.
11. $\mathcal{V} \to \pi$: $\{k_u : u \in [t]\}$.
12. $\mathcal{P}_\xi \to \mathsf{A}$: $\langle X_u\rangle^\xi = \langle \mathsf{U}_x\rangle^\xi[\cdot, j_u, k_u]$, $\langle Y_u\rangle^\xi = \langle \mathsf{U}_y\rangle^\xi[\cdot, j_u, k_u]$, $\langle Z_u\rangle^\xi = \langle \mathsf{U}_z\rangle^\xi[\cdot, j_u, k_u]$, $\langle P_u\rangle^\xi = \langle P\rangle^\xi[\cdot, k_u]$ for $u \in [t]$.
13. $\mathsf{A} \to \mathcal{V}$: $A_u = \sum_{\xi\in[\mathsf{N}]}\langle A_u\rangle^\xi$ where $A \in \{X, Y, Z, P\}$ and sends X_u, Y_u, Z_u for $u \in [t]$.
14. $\pi \to \mathcal{V}$: $\pi[\cdot, k_u]$ for $u \in [t]$.
15. $\mathcal{V} \to \mathcal{P}_\xi$: $\delta \leftarrow_\$ \mathbb{F}^p$, $\beta_x \leftarrow_\$ \mathbb{F}$, $\beta_y \leftarrow_\$ \mathbb{F}$, $\beta_z \leftarrow_\$ \mathbb{F}$, $\beta \leftarrow_\$ \mathbb{F}\backslash\{0\}$.
16. $\mathcal{P}_\xi$ computes:$\langle V_u\rangle^\xi = \sum_{i=1}^{p}\delta_i\left(\beta_x\langle \mathsf{U}_x\rangle^\xi[i,\cdot,k_u] + \beta_y\langle \mathsf{U}_y\rangle^\xi[i,\cdot,k_u] + \beta_z\langle \mathsf{U}\rangle^\xi[i,\cdot,k_u]\right)$ and $\langle z\rangle^\xi = \beta \cdot P_0 + \langle \overline{P}\rangle^\xi\varphi + \langle 0^{2m-1}\rangle^\xi$.
17. $\mathcal{P}_\xi \to \mathsf{A}$: sends $\langle z\rangle^\xi, \langle V_u\rangle^\xi$.
18. A computes $z = \sum_{\xi\in[\mathsf{N}]}\langle z\rangle^\xi$ and $V_u = \sum_{\xi\in[\mathsf{N}]}\langle V_u\rangle^\xi$.
19. A and $\mathcal{V}$: Both compute $W_u = \beta_x X_u + \beta_y Y_u + \beta_z Z_u$ for $u \in [t]$. $T_u = (C_u)^{\beta_x} \cdot (D_u)^{\beta_y} \cdot (E_u)^{\beta_z}$, for $u \in [t]$ where $C_u = \prod_{i=1}^{p}(\pi_x[i,k_u])^{\delta_i}$, $D_u = \prod_{i=1}^{p}(\pi_y[i,k_u])^{\delta_i}$ and $E_u = \prod_{i=1}^{p}(\pi_y[i,k_u])^{\delta_i}$.
20. A and $\mathcal{V}$ run inner product arguments to check:
    (a) $\mathsf{InnerProduct}(\mathsf{pp}, \mathbf{1}_{j_u}^T\Lambda_{h,2m-1}, \mathsf{cm}_{k_u}, v_u; \overline{P}[\cdot, k_u])$ for $u \in [t]$ where $\mathsf{cm}_{k_u} = c_0^\beta\prod_{a=1}^{2\ell-1}(c_a)^{\Lambda^T_{n,2\ell-1}[a,k_u]}$, $v_u = \sum_{i=1}^{p} r_i[X_u[i] \cdot Y_u[i] - Z_u[i]]$ (check consistency of P with π).
    (b) $\mathsf{InnerProduct}(\mathsf{pp}, \gamma||0^{m-1}, \mathsf{cm}, 0; z)$ where $z = \beta \cdot P_0 + \overline{P} \times \varphi$ and $\varphi = \Phi^T\tau$ and $\mathsf{cm} = \prod_{a=1}^{2\ell-1}(c_a)^{\varphi_a}$.
    (c) $\mathsf{InnerProduct}(\mathsf{pp}, \mathbf{1}_{j_u}^T\Lambda_{h,m}, T_u, w_u; \overline{V}_u)$, where $\overline{V}_u$ stands for first m entries of V_u and $w_u = \langle \delta, W_u\rangle$ (consistency of X_u, Y_u, Z_u with π).
21. $\mathcal{V}$ checks proximity of $\mathsf{U}_x, \mathsf{U}_y$ and U_z according to the Equation (2).

Fig. 7. Distributed Quadratic Check Protocol

$DP - \mathsf{GrapheneR1CS}(\mathsf{pp}, A, B, C, [\pi]; \mathsf{w}_x, \mathsf{w}_y, \mathsf{w}_z, \mathsf{w})$:
Relation: $A\mathsf{w} \circ B\mathsf{w} = C\mathsf{w}$.
Oracle Setup: Compute $\mathcal{O}$ as described above. Set $\pi := \mathcal{O}$.

1. $\mathcal{V} \to \mathcal{P}_\xi$: $\gamma_x, \gamma_y, \gamma_z \leftarrow\$ \mathbb{F}$, $r_{lc} \leftarrow\$ \mathbb{F}^M$, $r_{qd} \leftarrow\$ \mathbb{F}^p$, $\rho \leftarrow\$ \mathbb{F}^{4p}$.
2. $\mathcal{P}_\xi \to \mathsf{A}$: $\mathcal{P}_\xi$ computes $\langle \tilde{\mathsf{U}} \rangle^\xi = \sum_{i \in [4p]} \rho_i \langle \mathsf{U} \rangle^\xi[i, \cdot, \cdot]$ and sends commitments $\langle \tilde{c}_1 \rangle^\xi, \ldots, \langle \tilde{c}_\ell \rangle^\xi$ to $\langle \tilde{\mathsf{U}} \rangle^\xi$.
3. $\mathsf{A} \to \mathcal{V}$: A computes $\tilde{c}_k = \prod_{\xi \in [\mathsf{N}]} \langle \tilde{c}_k \rangle^\xi$ and sends $\tilde{c}_1, \ldots, \tilde{c}_\ell$.
4. $\mathcal{P}_\xi \leftrightarrow \mathcal{V}$ compute: $R = r_{lc}^T W$ for $W = [\gamma_x I \,||\gamma_y I \,||\gamma_z I \,|| - (\gamma_x A + \gamma_y B + \gamma_z C)]$, polynomials R^i,$i \in [4p]$ interpolating the slices of R viewed as a $4p \times m \times n$ matrix.
5. $\mathcal{P}_\xi$ computes:
   - Polynomials $\langle Q_x^i \rangle^\xi, \langle Q_y^i \rangle^\xi, \langle Q_z^i \rangle^\xi, \langle Q^i \rangle^\xi$ for $i \in [p]$, where polynomials $\langle Q_a^i \rangle^\xi$, $i \in [p]$ correspond to $\langle \mathsf{w}_a \rangle^\xi$ for $a \in \{x, y, z\}$ and polynomials $\langle Q^i \rangle^\xi$,$i \in [p]$ correspond to $\langle \mathsf{w} \rangle^\xi$.
   - Provers invoke $\mathcal{F}_{\mathsf{Mult}}$ on inputs $\langle Q_x^i \rangle^\xi, \langle Q_y^i \rangle^\xi$ and $\mathcal{P}_\xi$ gets $\langle Q_{xy}^i \rangle^\xi$, share of the polynomial Q_{xy}^i.
   - $h \times n$ matrices $\langle P_{lc} \rangle^\xi$ and $\langle P_{qd} \rangle^\xi$ as "P" matrices for the linear check and quadratic check respectively. Note that $\langle p_j \rangle^\xi$ polynomial for $\langle P_{lc} \rangle^\xi$ is given by $\langle p_j \rangle^\xi(\cdot) = \sum_{i=1}^p (R^i(\alpha_j, \cdot) \cdot \langle Q_x^i \rangle^\xi(\alpha_j, \cdot) + R^{p+i}(\alpha_j, \cdot) \langle Q_y^i \rangle^\xi(\alpha_j, \cdot) + R^{2p+i}(\alpha_j, \cdot) \langle Q_z^i \rangle^\xi(\alpha_j, \cdot) + R^{3p+i}(\alpha_j, \cdot) \langle Q^i \rangle^\xi(\alpha_j, \cdot))$. The $\langle p_j \rangle^\xi$ polynomials for the matrix $\langle P_{qd} \rangle^\xi$ are given by $\langle p_j \rangle^\xi(\cdot) = \sum_{i=1}^p r_{qd}[i](\langle Q_{xy}^i \rangle^\xi(\alpha_j, \cdot) - \langle Q_z^i \rangle^\xi(\alpha_j, \cdot))$.
   - Blinding vectors $\langle U_{lc} \rangle^\xi, \langle U_{qd} \rangle^\xi \in \mathbb{F}^{2m-1}$ for linear and quadratic check protocols respectively, and commitments $\langle c_0 \rangle^\xi, \langle d_0 \rangle^\xi$ to vectors $\langle U_{lc} \rangle^\xi$ and $\langle U_{qd} \rangle^\xi$.
6. $\mathcal{P}_\xi \to \mathsf{A}$: Commitments $\langle c_0 \rangle^\xi, \langle c_1 \rangle^\xi, \ldots, \langle c_{s+\ell-1} \rangle^\xi$ for the matrix $\langle P_{lc} \rangle^\xi$ and commitments $\langle d_0 \rangle^\xi, \langle d_1 \rangle^\xi, \ldots, \langle d_{2\ell-1} \rangle^\xi$ for matrix $\langle P_{qd} \rangle^\xi$.
7. $\mathsf{A} \to \mathcal{V}$: A computes $c_k = \prod_{\xi \in [\mathsf{N}]} \langle c_k \rangle^\xi \ \forall k \in \{0, \ldots, s + \ell - 1\}$ and $d_k = \prod_{\xi \in [\mathsf{N}]} \langle d_k \rangle^\xi \ \forall k \in \{0, \ldots, 2\ell - 1\}$ and sends $c_0, c_1, \ldots, c_{s+\ell-1}, d_0, d_1, \ldots, d_{2\ell-1}$.
8. $\mathcal{V} \to \mathcal{P}_\xi$: $Q = \{(j_u, k_u) : u \in [t]\}$.
9. $\mathcal{V} \to \pi$: $\{k_u : u \in [t]\}$.
10. $\mathcal{P}_\xi \to \mathsf{A}$: $\langle S_u \rangle^\xi = \langle \mathsf{U} \rangle^\xi[\cdot, j_u, k_u]$, $\langle Plc_u \rangle^\xi = \langle P_{lc} \rangle^\xi[\cdot, k_u]$, $\langle Pqd_u \rangle^\xi = \langle P_{qd} \rangle^\xi[\cdot, k_u]$ for $u \in [t]$.
11. A computes $S_u = \sum_{\xi \in [\mathsf{N}]} \langle S_u \rangle^\xi$, $Plc_u = \sum_{\xi \in [\mathsf{N}]} \langle Plc_u \rangle^\xi$ and $Pqd_u = \sum_{\xi \in [\mathsf{N}]} \langle Pqd_u \rangle^\xi$ for $u \in [t]$.
12. $\mathsf{A} \to \mathcal{V}$: S_u for $u \in [t]$.
13. $\pi \to \mathcal{V}$: $\pi[\cdot, k_u]$, $u \in [t]$.
14. $\mathcal{V} \to \mathcal{P}_\xi$: $\delta_{lc} \leftarrow\$ \mathbb{F}^{4p}, \delta_{qd} \leftarrow\$ \mathbb{F}^p, \beta_{lc}, \beta_{qd} \leftarrow\$ \mathbb{F}^*, \beta_x, \beta_y, \beta_z \leftarrow\$ \mathbb{F}$.
15. $\mathcal{P}_\xi \to \mathsf{A}$: $\langle Vlc_u \rangle^\xi = \langle \delta_{lc}, \langle U[\cdot, j_u, k_u] \rangle^\xi \rangle$,
    $\langle Vqd_u \rangle^\xi = \langle \delta_{qd}, (\beta_x U_x[\cdot, j_u, k_u] + \beta_y U_y[\cdot, j_u, k_u] + \beta_z U_z[\cdot, j_u, k_u]) \rangle$.
16. A computes $Vlc_u = \sum_{\xi \in [\mathsf{N}]} Vlc_u$ and $Vqd_u = \sum_{\xi \in [\mathsf{N}]} Vqd_u$.
17. A and $\mathcal{V}$ run the linear and quadratic check protocols in parallel, parsing the vectors V_u into X_u, Y_u, Z_u, W_u as needed. And compute $A_u = \beta_x X_u + \beta_y Y_u + \beta_z Z_u$ for $u \in [t]$ and corresponding commitment.
18. Check proximity as: $\prod_{a=1}^{\ell} (\tilde{c}_a)^{\Lambda_{n,\ell}^T[a, k_u]} = \prod_{i=1}^{4p} (\pi[i, k_u])^{\rho_i}$ for $u \in [t]$.
19. $\mathcal{V}$ accepts if all the subprotocols accept.

Fig. 8. Distributed GrapheneR1CS Protocol