sciendo

Proceedings on Privacy Enhancing Technologies ; 2022 (2):517–556

Pankaj Dayama, Arpita Patra, Protik Paul*, Nitin Singh, and Dhinakaran Vinayagamurthy

How to prove any NP statement jointly? Efficient 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 study the notion of distributed-prover zero knowledge (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 make the following contributions to the notion of distributed proof generation: (i) we propose a new MPC-style security definition to capture the adversarial settings possible for different collusion models between the provers and the verifier, (ii) we discuss new efficiency parameters for distributed proof generation such as the number of rounds of interaction and the amount of communication among the provers, and (iii) we propose a compiler that realizes distributed proof generation from the zero-knowledge protocols in the Interactive Oracle Proofs (IOP) paradigm. Our compiler can be used to obtain DPZK from arbitrary IOP protocols, but the concrete efficiency overheads are substantial in general. To this end, we contribute (iv) a new zero-knowledge IOP Graphene which can be compiled into an efficient DPZK protocol. The $(\mathsf{D}+1)$-DPZK protocol D-Graphene, with D provers and one verifier, admits $O(N^{1/c})$ proof size with a communication complexity of $O(\mathsf{D}^2 \cdot (N^{1-2/c} + N_s))$, where N is the number of gates in the arithmetic circuit representing the predicate and N_s is the number of wires that depends on inputs from two or more parties. Significantly, only the distributed proof generation in D-Graphene requires interaction among the provers. D-Graphene compares favourably with the DPZK protocols obtained from the state-of-art zero-knowledge protocols, even those not modelled as IOPs.

Keywords: Zero-Knowledge Proofs, Secure Multi-Party Computation, Distributed-Prover Zero-Knowledge

DOI 10.2478/popets-2022-0055

Received 2021-08-31; revised 2021-12-15; accepted 2021-12-16.

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 [40] or validating sensitive web browser data reported during telemetry [18, 25]. 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 its own secret data, want to prove to a verifier that their secret data together satisfy 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.

Formally, we have multiple provers $\mathcal{P}_1, \ldots, \mathcal{P}_{\mathsf{D}}$ respectively possessing witnesses $\mathsf{w}_1, \ldots, \mathsf{w}_{\mathsf{D}}$. For a predicate C the provers wish to prove to a verifier that $C(\mathsf{w}_1, \cdots, \mathsf{w}_{\mathsf{D}}) = 1$. A natural solution for distributed

Pankaj Dayama: IBM Research, E-mail: pankajdayama@in.ibm.com, IBM
Arpita Patra: Indian Institute of Science Bangalore, E-mail: arpita@iisc.ac.in
***Corresponding Author: Protik Paul:** Indian Institute of Science Bangalore, E-mail: protikpaul@iisc.ac.in
Nitin Singh: IBM Research, E-mail: nitisin1@in.ibm.com
Dhinakaran Vinayagamurthy: IBM Research, E-mail: dvinaya1@in.ibm.com

BY-NC-ND

proof generation is to start with the prover algorithm of a single prover protocol and run this algorithm between multiple provers using multi-party computation (MPC). This generic construction was discussed by Pedersen [39]. However, the generic construction is unlikely to be efficient in practice. Known constructions of efficient zero-knowledge proofs involve expensive computation (e.g. FFT) over complex mathematical structures such as fields, groups and elliptic curves, which are expensive when expressed as an arithmetic circuit. Running multiparty computation over such circuits with several parties will be prohibitive. To get around the inefficiency of a generic construction, some prior works have proposed efficient distributed proof generation for restricted class of computations. These include simple predicates involved in threshold signatures [38], some sigma protocols [31], combined range proofs [13]. Distributed proof generation for more general computation has been considered in [41], but under weaker trust model involving a trusted setup and majority of the parties being honest [41]. The goal of this paper is to enable DPZK for general computation without requiring trusted setup or honest majority. We summarize our contributions below.

Our Contributions

- 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.
- We present a compiler that takes any IOP-based zero-knowledge protocol and converts it to an "MPC-friendly" zero-knowledge protocol which can then be used to obtain a DPZK protocol.
- We illustrate the application of our compiler for two single-oracle IOPs, which is the preferred setting for our compiler. We obtain the protocol D-Ligero by using our compiler with the Ligero protocol from [1]. Building upon the techniques used in Ligero, we construct a new single-oracle IOP which we call Graphene. The protocol Graphene admits smaller proof sizes than Ligero while ensuring efficient verification. Moreover we show that Graphene can also be efficiently compiled to yeild a DPZK protocol, which we call D-Graphene. The zero knowledge protocols are described as interactive protocols secure against honest verifiers. Using standard transforms such as [7, 26], they can be used to obtain succint non-interactive arguments of knowledge (SNARGs), which are secure against malicious verifiers in the Random Oracle model.

Before providing a detailed overview of our work, we discuss potential real-world applications where distributed proof generation can be useful.

2 Potential Application: Multi-Wallet Anonymous Payments

One application scenario that we highlight is in the context of decentralized anonymous payment networks similar to Zcash [40]. In this example, we consider a variant of Zcash network suitably modified to work with MPC-friendly zkSNARKs discussed in this paper. The users on the network create and consume coins via following transactions: (i) **Mint** transaction which allows users to introduce coins to the system (after equivalent funds are deposited to a backing pool), (ii) **Spend** transaction which allows a user to consume his unspent coins and create new coins for other users and (iii) **Redeem** transaction which allowed a user to redeem his unspent coins in exchange for equivalent funds in traditional banking systems. Our application is concerned with the **Spend** transaction. In general an n-ary spend transaction consumes n input coins and outputs n coins with matching cumulative value. The output coins may be assigned to different users. To ensure unlinkability of input and output coins, coins are created and spent using separate identifiers, known as *coin commitment* (cm) and *serial number* (sn) respectively. Roughly, cm and sn for a coin are linked via a trapdoor τ: $\mathsf{cm} = f_1(\tau)$ and $\mathsf{sn} = f_2(\tau)$ for one-way functions f_1 and f_2. Knowing one of them, it is infeasible to infer the other. As part of a spend transaction, the user supplies serial numbers $\mathsf{sn}_1, \ldots, \mathsf{sn}_n$ for the n coins being consumed along with the commitments for the output coins. Additionally, the user supplies a zero knowledge proof π showing that each serial number corresponds to a valid and unspent coin and that the cumulative value of the coins is equal to the cumulative value of the output coins. By verifying the above zero knowledge proof, the participants on the network can ensure the validity of the transaction.

The above setup does not allow different users to pool in their coins as inputs to a spend transaction, for example, to jointly pay off another entity. The limitation stems from having to generate a zero knowledge

proof, for which the user requires knowledge of all the private information associated with a coin. One way to work around the limitation is for users to transfer their respective coins (via spend transactions) to one designated user, who then initiates a spend transaction consuming all the received coins on their behalf. However, the operation is no longer *atomic* and the designated user may not go ahead with his transaction.

Using distributed proof generation, the set of pooling users supply the share of the witness corresponding to the coins they own, and use a distributed proof generation protocol proposed in this paper to output a proof that proves the validity of all the coins to the larger network. This results in an atomic payment between upto n senders and recipients.

Similarly, for auctions with joint bidding that use cryptocurrencies, a set of users can leverage DPZK to produce a proof that they together own coins amounting to the submitted bid.

3 Overview of Our Work

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.

However, 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 a 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, [4, 19, 28], 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 MPC executions between the provers during the distributed proof generation. 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 are more than one round of prover message generation that 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, that is the witness $\mathsf{w} = \mathsf{w}_1||\ldots||\mathsf{w}_{\mathsf{D}}$, the term *shared circuit* denotes a sub-circuit consisting of wires, whose values are functions of inputs from more than one prover.

Consider the circuit for verifying the validity of the **Spend** transaction in Section 2. Let rt denote the Merkle root over the list of coins on the ledger. Thus, to show that a coin commitment cm corresponds to a valid coin, one provides an authentication path $\boldsymbol{p}$ from cm to rt. Hence, the witness $\boldsymbol{w}_i$ involves the coin commitment cm_i, the trapdoor τ_i for the coin and authentication path $\boldsymbol{p}_i$ from cm_i to the root rt. This part of the witness is completely known to the owner of the i^{th} coin. Thus the witness $\boldsymbol{w}$ authenticating the coins against the serial numbers is canonically partitioned into $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_n$ and requires no interaction to compute. On the other hand, secure computation is required to compute shares of witness wires checking the equality $u_1 + \ldots + u_n == v_1 + \ldots + v_n$ where u_i is the value of the i^{th} output coin and v_i denotes the value of the i^{th} input coin. Clearly, this "shared circuit" constitutes only a fraction of the entire witness. It would be desirable, though not obvious if the secure computation for *computing the proof* also depends on the size of *shared circuit* instead of the that of the entire circuit. Since verifying merkle authentication paths using standard hash functions (like SHA2) incur upwards of $100K$ gates, the practicality of distributed proof generation in this case 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 con-

cern) will be measured via c_{pr}, r_{pr}, in addition to the parameters for the single prover protocol.

3.1 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, 17, 27, 35] 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 parametrized 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 subsets of provers and verifier) can do in real execution can be done in the ideal execution.

We formalize the security of a DPZK protocol and state the precise trust assumptions under which we realize it in Section 4. Here we foreshadow the important aspects: (i) *Witness Confidentiality*, i.e, a DPZK protocol ensures that other provers do not learn private inputs of a prover during distributed proof generation involving *semi-honest* provers, (ii) *Zero Knowledge*, i.e, an honest verifier learns nothing beyond the truth of the statement in a DPZK protocol involving semi-honest provers and (iii) *Soundness With Witness Extraction*, i.e, an honest verifier rejects a false statement even if the provers act maliciously. In practice, the restriction of honest verifier is overcome by using standard transformations to obtain non-interactive argument from the interactive proof. Realizing a DPZK protocol for a malicious set of provers in an interesting future work.

3.2 Compiler for IOP-based Proof Systems

Several recent constructions of efficient SNARKs [6, 7] are modeled in terms of *Interactive Oracle Proofs* (IOPs). In this work, we present a compiler that compiles an existing IOP, into an IOP that admits efficient proof generation by a distributed set of provers. This requires navigating several technical challenges: (i) IOPs use the abstraction of oracles, wherein the verifier does not receive the messages from the prover entirely, and only query a small number of positions and (ii) transforming IOPs to *Non-interactive Proof of Knowledge* (SNARKs) requires the use of collision-resistant hash functions to realize the oracles. The major challenge that we address is providing the abstraction of oracle by a distributed set of provers (with shares of the message), which is indistinguishable from the one provided by a single prover (with the complete message).

We use a common aggregator A to interact with the verifier on behalf of all the provers. Naively, to provide oracle access to a message, A itself needs to have access to the complete message. But this violates our privacy constraints: IOPs ensure privacy only when a "small" portion of the message is accessible. We resolve this conflict by using linear sharing and homomorphic commitments. In all rounds, provers maintain a linear sharing of the IOP message. They share commitments of their shares with A, which then has access to the commitment of complete IOP message. To the external verifier, A presents the commitment as the message and provides query access to it. While answering queries on the "committed oracle", A additionally opens the commitments that are queried. We present the above construction as two steps: the first step involves transforming an existing IOP into an IOP with "homomorphic" oracles. In the second step, a $(\mathsf{D}+1)$-DPZK protocol is obtained where D provers run the prover algorithm of the homomorphic IOP protocol ($\mathsf{D}+1$ indicates that the protocol involves D provers and one verifier). These provers jointly compute the proof where the provers start with a linear sharing of the witness. We elaborate more below and defer the details to the later section (Section 5).

Let $\langle \mathcal{P}, \mathcal{V} \rangle$ be an r round IOP-based proof system. $\mathcal{P}$ sends $f_i = (m_i, \pi_i)$ as the ith round message and $\mathcal{V}$ has oracle access to f_i where m_i is given to $\mathcal{V}$ in the clear and $\mathcal{V}$ makes bounded number of queries to π_i. In the compiled protocol, say $\langle \mathcal{P}', \mathcal{V}' \rangle$, $\mathcal{P}'$ sends $f'_i = (m_i, \pi'_i)$ at the ith round where π'_i is the commitment of π_i. In this step a homomorphic commitment is used. If $\mathcal{V}'$ queries some location to the oracle, oracle sends the committed value, and $\mathcal{P}'$ provides the corresponding opening. $\mathcal{V}'$ checks if the opening of π'_i on the queried locations are correct. If all the openings are correct then $\mathcal{V}'$ runs the verification algorithm of $\mathcal{V}$. Completeness and zero-knowledge of the newly obtained protocol follow directly from the under-

lying protocol, and soundness depends on the soundness property of the base protocol and the binding property of the commitment scheme.

To obtain the DPZK from the compiled protocol, we start with D copies of $\mathcal{P}'$. Each of these provers computes a linear sharing of (m_i, π_i). If the m_i and π_i are linear functions of the secret, then no interaction among the provers is required. That is, they locally compute these shares. If not, they perform secure evaluations of these values and obtain linear shares. All the provers locally obtain sharing of (m_i, π_i') from linear sharing of (m_i, π_i). A obtains shares of (m_i, π_i'), combines them, and sets them as the oracle. The verification algorithm is the same as in the single prover protocol.

This compiler preserves the proof size and the number of the rounds. However, the compiled protocol has an overhead of proof generation and verification time. This overhead depends on the oracle size, the number of oracles and the number of rounds.

3.3 Instantiations of Distributed Prover Zero-Knowledge

We instantiate a few distributed proof generation protocols. As discussed earlier, a protocol obtained from the above compiler is more likely to perform better in a distributed proof generation if the base protocol has less number of rounds and oracles. Thus we start with the compiled version of Ligero [1] since it has a constant number of rounds and only one oracle. Further, we provide a protocol Graphene by optimizing this newly obtained protocol from Ligero.

Additionally, we discuss the cost of the distributed protocol obtained from Aurora [6]. This gives a perspective of the efficiency of a compiled protocol where the base protocol is a multi-oracle IOP. We study the distributed proof generation for non-IOP protocols such as Bulletproofs [13] and Spartan [42]. Bulletproofs achieve the distributed proof generation naturally without requiring any additional primitives excluding secure multiplication (which is inherent in all the instantiations). In the distributed variant of Bulletproofs, provers securely evaluate multiplication of 2 vectors. The size of these vectors is $O(N_s)$, where N_s is the size of the shared circuit. Furthermore, each prover sends $O(N)$ field elements towards the aggregator. In the distributed version of Spartan, $O(N^2)$ multiplications are required among the provers. However, this construction does not ensure privacy among the provers. This shows the non-triviality of the distributed proof generation.

3.4 Graphene: an MPC-friendly Zero-Knowledge Protocol

To construct Graphene, we adjoin an additional dimension in the Ligero protocol and modify the encoding, the linear check, and the quadratic check accordingly. With this optimization, we break the $\sqrt{N}$ (N is the size of the circuit) barrier of Ligero, following a similar idea of the concurrent work of Ligero++ [8] where succinct inner product argument is used. Ligero++ uses inner product argument from Virgo [46] whereas, we use Bulletproofs' [13] inner product argument. Furthermore, for the soundness of Graphene, we provide novel results in coding theory. By adjoining an additional dimension, our protocol gets better flexibility to trade-off between proof size and verification time. Furthermore, we organize the witness so that if the shared circuit is small, then the communication among the provers is required for $N^{1-2/c}$ multiplications. In contrast, Ligero's compiled protocol would require $N^{1/2}$ multiplications. Also in distributed version of Ligero and Graphene, each prover sends $O(\sqrt{N})$ and $O(N^{1-2/c})$ field elements towards the aggregator respectively. Note that the 3-D encoding, obtained by adjoining an additional dimension, helps in reducing the proof size while keeping the verification time low. Also, if the size of the shared circuit is small, this additional dimension aids in reducing the communication among the provers as well. The details of Graphene are given in Appendix B.

3.5 Related Work

As mentioned earlier in the introduction, Pedersen [39] 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. [21] proposing proofs for graph isomorphism, various works [20, 33, 38] for proposing threshold signatures, Keller et al. [31] for a class of sigma protocols. More recently [13] presented a distributed variant of *range proofs* which allows several provers to compute a common proof showing their private inputs lie within an interval. However their protocol does not satisfy our stronger privacy requirements as the size of the proof depends on the number of provers.

Trinocchio [41] proposes a distributed proof generation method for Pinocchio [37] but only for honest majority setting. Moreover, it assumes a trusted setup. The work DIZK [44] provides a distributed implementation

of proof generation to reduce proving time. There is no notion of privacy among the provers in this setting.

In this work, we study how to achieve distributed proof generation from other existing works such as Ligero [1], Aurora [6], Bulletproofs [13], Spartan [42] and the corresponding challenges. To attain an efficient construction, we start with a Ligero-style protocol and use a similar approach to [11, 12], where the witness is viewed as a multi-dimensional matrix. However, 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, increasing the number of dimensions beyond three leads to more complicated and costlier proof-generation and verification protocols without improving any other parameters. [12] provides linear-time prover with polylogarithmic verification, but a significant drawback of this work is that the soundness error is $O(1)$. Furthermore, [11] obtain linear-time prover by using linear-time encodable codes. For this, they use a linear code provided by [23], whose decoding is conjectured intractable. Due to this property, [11] does not satisfy the proof of knowledge property. Moreover, constructions with higher dimensions require more rounds and oracles which incur higher overhead in the distributed proof generation. Ligero++ [8] overcomes Ligero's $O(\sqrt{N})$ proof size bottleneck by using Virgo [46] inner product argument, which uses Aurora style proof system. Thus the distributed proof generation of Ligero++ is similar to the distributed variant of Aurora. In our construction, we use inner product argument from Bulletproofs. In Table 1, we compare the different metrics of efficiency for all the existing DPZK protocols.

Another recent line of works such as Wolverine [43], Mac'n'Cheese [2], LPZK [22], QuickSilver [45] focuses on the scalability of the zero-knowledge proofs. These works take the "gate-by-gate" paradigm by relying on Vector Oblivious Linear Evaluations (VOLEs). Wolverine provides a ZK protocol with 4 field elements per multiplication gate. Mac'n'Cheese further improves it by providing a protocol with 3 elements per multiplication gate. While QuickSilver and LPZK both achieve a ZK protocol with 1 element per multiplication gate, LPZK is computationally heavier. All these works have linear communication complexity. Additionally, if the circuit satisfies a certain condition, called "weak notion of uniformity", QuickSilver achieves sub-linear communication. The gate-by-gate paradigm employed in these works, however, is restricted to interactive protocols. That is, it does not support non-interactiveness. Furthermore, a common aggregator approach does not work for these protocols since this would either require the verifier to communicate with all the provers or the aggregator to learn the whole witness. Overcoming the problems mentioned above and obtaining distributed variants of these works is an interesting area to explore, which we leave as future work.

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 $\mathcal{P}_1, \ldots, \mathcal{P}_{\mathsf{D}}$ be D 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 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 D-input and D-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. $\pi_i(\mathsf{st}_1^i, \ldots, \mathsf{st}_{\mathsf{D}}^i)$. st_j^1 is set to $\mathcal{P}_j$'s share of the witness w_j, randomness, the public parameters 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).

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

Table 1. Comparison amongst the DPZKs. Here N is the size of the circuit and, c is a positive integer of our choice. D 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.

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-confidentiality: 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, 17, 27, 35] for details regarding the security model. Extending the definition to Universal Composability (UC)[15] secure model and obtaining a secure protocol in this model is an interesting question that we leave as a future work.

The ideal execution: The "ideal" world execution of DPZK involves parties in $\mathcal{P}$ that includes D provers $\{\mathcal{P}_1, \ldots, \mathcal{P}_{\mathsf{D}}\}$ 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 $\mathcal{P}_\xi$ sends (x, w_ξ) to $\mathcal{F}_{\text{DPZK}}$, where x is the statement and w_ξ is the ξth part of the witness. The verifier $\mathcal{V}$ sends x. $\mathcal{F}_{\text{DPZK}}$ computes $\mathsf{w} = \mathsf{w}_1 \oplus \mathsf{w}_2 \ldots \oplus \mathsf{w}_{\mathsf{D}}$, 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 $\vec{x} = (x_1, \ldots, x_{\mathsf{D}}, x_{\mathcal{V}})$ for the parties and auxiliary input, such as trapdoor and additional information, 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 and the verifier communicate with a designated prover, referred as aggregator. 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, 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.

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 $\mathcal{P}_i$ $\forall i \in [\mathsf{D}]$ 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 $\mathcal{P}_i \in [\mathsf{D}]$ and $\{x_i\}_{i \in I}$ to the simulator $\mathcal{S}$, where for all $i \in I$, $\mathcal{P}_i$ is a corrupt prover.
- If any of the x_i is abort, send abort to everyone in $\mathcal{P}$. Otherwise, compute $\mathsf{w} = \mathsf{w}_1 \oplus \mathsf{w}_2 \ldots \oplus \mathsf{w}_{\mathsf{D}}$, 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 $\mathcal{F}_{\mathsf{DPZK}}$

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 1. *Let $\mathcal{F}_{DPZK}$ be a $(\mathsf{D}+1)$-party functionality and let Π_{DPZK} be a $(\mathsf{D}+1)$-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 $\vec{x} \in (\{0,1\}^*)^{\mathsf{D}}$ where $|x_1| = \ldots = |x_{\mathsf{D}}|$, 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 $\{\mathcal{P}_1, \ldots, \mathcal{P}_{\mathsf{D}}\}$, $\{\mathcal{V}\}$, a proper t-size subset of $\{\mathcal{P}_1, \ldots, \mathcal{P}_{\mathsf{D}}\}$, a proper t-size subset of $\{\mathcal{P}_1, \ldots, \mathcal{P}_{\mathsf{D}}\}$ plus $\mathcal{V}$, the above indistinguishability is referred as Soundness with Witness Extraction (SoWE), Zero-Knowledge (ZK), witness-confidentiality (WC), witness-confidentiality with collusion (WCwC) *respectively.*

Soundness with Witness Extraction (SoWE) Let $\mathcal{A}$ be the adversary corrupting parties in $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_{\mathsf{D}}\}$ and Π_{DPZK} be a secure protocol against $\mathcal{A}$. Then, by definition 1, there exists a simulator which emulates the verifier ($\mathcal{V}$) and outputs a view indistinguishable from the real-world view. If $\mathcal{A}$ interacts with $\mathcal{V}$ which causes $\mathcal{V}$ to accept, then the simulator extracts the input of $\mathcal{A}$ such that the output of the functionality is accept. We refer to this simulator as an "extractor". That is, a protocol Π_{DPZK} is said to have the *soundness with witness extraction* property, if corresponding to an adversary $\mathcal{A}$ that corrupts $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_{\mathsf{D}}\}$ and outputs an accepting proof for a statement x and relation R, there exists an extractor $\mathcal{E}$ that emulates $\mathcal{V}$ and with overwhelming probability extracts a witness w such that $R(x, \mathsf{w}) = 1$.

Zero-Knowledge (ZK) Let $\mathcal{A}$ be the adversary corrupting the verifier $\mathcal{V}$. A protocol Π_{DPZK} is said to have the *zero-knowledge* property, if there exists a simulator $\mathcal{S}$ that outputs a transcript which is indistinguishable from a real transcript. Note that a non-colluding verifier (verifier that does not collude with any of the provers) learns nothing more than the statement's assertion, not even the number of provers in the protocol. Therefore the simulated transcript remains independent of the number of provers.

Note that if the functionality in Figure 4.1 is modified such that the witness w is the input of only one prover and a protocol is secure against a verifier that learns nothing more than the assertion of the statement x, has the zero-knowledge property. The simulator for a corrupt verifier does not use the parameter D to simulate the view. Our construction satisfies this requirement.

Witness-Confidentiality (WC) A protocol Π_{DPZK} is said to have *witness confidentiality* property if an adversary $\mathcal{A}$, corrupting $t(< \mathsf{D})$ out of D provers, learns nothing about the honest provers' inputs. That is, there exists a simulator $\mathcal{S}$ acting on behalf of the honest provers and the verifier, which generates a view which is indistinguishable from the real-world view of the corrupt provers.

Witness-Confidentiality with Collusion (WCwC) A protocol Π_{DPZK} is said to have *witness confidentiality with collusion* property if there exists a simulator $\mathcal{S}$, corresponding to an adversary $\mathcal{A}$ corrupting t provers and the verifier, that plays the role of the honest provers and generates a view which is indistinguishable from the real-world view of the corrupt parties.

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 [7, 26] 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 WC and WCwC 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 dishonestly, 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) WC 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 [31] 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 [31] 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 decentralization 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. [39] also defines distributed proofs. [39] 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 [39] 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. [3]. 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 [3, 9] is entirely different from our notion since, in the DPZK setting, no prover holds 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. [30], where the prover runs a multiparty protocol in its head and uses the view of the emulated parties to provide a proof. This process requires opening 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 and corruption scenario (honest majority or dishonest majority/semi-honest or malicious). In an arbitrary setting, using MPC-in-the-head-paradigm can be inefficient. One might ask to run MPC to generate the trusted setup and then use a protocol with a trusted setup. However, generating a trusted setup via MPC requires the verifier to participate in the MPC, which precludes publicly verifiable non-interactive proofs.

5 DPZK Compiler

With recent advancements in zero-knowledge, protocols in IOP paradigm, such as Ligero [1], Ligero++ [8], Aurora [6] to name a few, bring efficiency in proof size, proof generation time and verification time. Most of the IOP-based protocols work with only symmetric key primitives, which leads to better efficiency. Therefore in search of efficient DPZK, we look into the IOP-based paradigm. Designing a DPZK directly from an IOP-based protocol faces the following challenges. (i) Each prover sets its oracle individually: in this case the verifier is required to communicate with each prover. It violates the zero-knowledge requirement of DPZK, specifically, the verifier learns the number of provers in the protocol. Furthermore, it increases the proof size and verification time by a factor of the number of provers. (ii) All the provers send their messages to one designated prover, called aggregator who aggregates and sets up the oracle in every round: in this case, the aggregator learns the private input of an honest prover, which defies the witness confidentiality property. In our construction, we use the second approach mentioned above where an aggregator sets up the oracle in every round. To ensure the witness confidentiality property it is now required that the aggregator be able to combine the messages from the provers without learning them in clear. Moreover, it is also required that a corrupt prover should not be able to deviate from the messages once the aggregator sets up the oracle. A homomorphic commitment scheme offers itself as a primitive which handles the above requirements. Such a scheme can be instantiated using Pedersen commitment or Pedersen vector commitment.

We encounter another roadblock as follows. In particular, it need not be the case that oracles in every round are linear functions of the witness. In cases where the oracle is a non-linear function, its construction may involve interactions among the provers. To tackle this, we use MPC in our construction. Therefore, the IOP protocols with fewer oracles/rounds are more likely to provide a better DPZK since the required number of MPC invocations would be lesser. Below we discuss a compiler that compiles a single prover ZK protocol to support distributed proof generation.

5.1 The Compiler

We give a generic construction from an IOP-based proof system to obtain another proof system that is suitable for distributed proof generation. The compiled protocol use homomorphic oracles. For this, the compiler requires a homomorphic functional commitment scheme for linear functions. A functional commitment scheme, FC, over a domain $\mathcal{D}$ is a tuple of four probabilistic polynomial-time algorithms - $(\mathsf{SetUp}, \mathsf{Com}, \mathsf{Open}, \mathsf{Verify})$. SetUp generates the key ck for the commitment. Com commits to a message $\mathbf{m}$, Open provides a witness for a partial or complete opening of the message $\mathbf{m}$, and Verify on input the commitment value and the witness outputs 1 if the opening is valid, and 0 otherwise. We refer the readers to [34] for more details on functional

commitments. We now recall the notion of IOP (refer [7] for more details).

Interactive Oracle Proofs (IOP)

Let $\langle \mathcal{P}, \mathcal{V} \rangle$ be a r-round IOP-based zero-knowledge proof system. Let $\mathsf{st}_0^p = \{x, \mathsf{w}\}$, $\mathsf{st}_0^v = \{x\}$, $f_0 = \perp$ and ρ_p, ρ_v be the randomness used by $\mathcal{P}$ and $\mathcal{V}$ respectively. Then for $i = 1$ to r, in the ith round: (i) $\mathcal{V}$ computes $(c_i, \mathsf{st}_i^v) = V^{f_0, f_1, \ldots, f_{i-1}}(\mathsf{st}_{i-1}^v, \rho_v)$ and sends c_i to $\mathcal{P}$. (ii) $\mathcal{P}$ computes $(f_i, \mathsf{st}_i^p) = P(c_1, \ldots, c_i, \mathsf{st}_{i-1}^p, \rho_p)$. The output of the protocol is $b := V^{f_0, f_1, \ldots, f_r}(\mathsf{st}_r^v, \rho_v)$, and b belongs to $\{0, 1\}$. $\langle \mathcal{P}, \mathcal{V} \rangle$ has the following properties.

Completeness: For every $(x, \mathsf{w}) \in R$, $\langle \mathcal{P}(x, \mathsf{w}), \mathcal{V}(x) \rangle$ outputs $b = 1$ with probability 1.

Proof of knowledge: $\langle \mathcal{P}, \mathcal{V} \rangle$ has proof of knowledge property if there exists a probabilistic polynomial-time algorithm $\mathcal{E}$ (extractor) and a negligible function μ such that, for every x and $\mathcal{P}^*$, $\Pr[(x, \mathcal{E}^{\mathcal{P}^*}(x)) \in R] \geq \Pr[\langle \mathcal{P}^*, \mathcal{V}(x) \rangle = 1] - \mu(|x|)$.

Zero knowledge: $\langle \mathcal{P}, \mathcal{V} \rangle$ has zero knowledge property if there exists a probabilistic polynomial-time algorithm $\mathcal{S}$ (simulator) such that for every $(x, \mathsf{w}) \in R$, it generates a transcript τ which is indistinguishable from a transcript of $\langle \mathcal{P}(x, \mathsf{w}), \mathcal{V}(x) \rangle$ to any PPT distinguisher.

Homomorphic IOP

Let $\langle \mathcal{P}, \mathcal{V} \rangle$ be an IOP-based zero-knowledge proof system with proof of knowledge property. In general, the prover in an IOP protocol provides oracle access to its messages in each round, while the verifier makes bounded number of queries to these oracles. Without loss of generality, we will write messages from the prover as $f_i = (m_i, \pi_i)$ for the ith round, where m_i denotes the part of message read in entirety by the verifier while π_i is the message to which the verifier has (bounded) query access.

The compiler takes an IOP-based zero knowledge protocol $\langle \mathcal{P}, \mathcal{V} \rangle$ and a homomorphic functional commitment scheme $\mathsf{Com}(\cdot)$. It outputs a protocol $\langle \mathcal{P}', \mathcal{V}' \rangle$, where $\langle \mathcal{P}', \mathcal{V}' \rangle$ is obtained in the following way: $\mathcal{P}'$ and $\mathcal{V}'$ run $\mathcal{P}$ and $\mathcal{V}$ algorithms respectively, in a state preserving manner. That is, $\mathcal{P}'$ runs $\mathcal{P}$'s first round and stores the state of $\mathcal{P}$. Upon receiving the challenge from $\mathcal{V}'$, $\mathcal{P}'$ runs the next round of $\mathcal{P}$ with the prior round's state and updates the state. $\mathcal{P}'$ follows this for all rounds. $\mathcal{V}'$ does the same for $\mathcal{V}$.

At ith round, $\mathcal{P}'$ sets $\pi_i' = \mathsf{Com}(\mathsf{ck}, \pi_i)$.

$\mathcal{P}'$ sends $f_i' = (m_i, \pi_i')$. $\mathcal{V}'$ has oracle access to f_i' such that it gets m_i in clear and queries limited number of locations of π_i'. According to the queried locations, $\mathcal{P}'$ provides opening of those locations. That is, if $\mathcal{V}'$ queries j then $\mathcal{P}'$ sends $\mathsf{Open}(\mathsf{ck}, \pi_i'[j], \mathsf{aux})$ to $\mathcal{V}'$, where aux denotes auxiliary information.

$\mathcal{V}'$ sets the challenge $c_i' = c_i$. When c_i consists of oracle query j, $\mathcal{V}'$ receives $\pi_i'[j]$ from the oracle and $W_{i,j} = \mathsf{Open}(\mathsf{ck}, \pi_i'[j], \mathsf{aux})$ from $\mathcal{P}'$. Further $\mathcal{V}'$ receives m_i. $\mathcal{V}'$ checks if $\mathsf{Verify}(\mathsf{ck}, \pi_i'[j], W_{i,j}, \pi_i[j]) = 1$, followed by the verification algorithm of $\mathcal{V}$.

Compiler from $\langle \mathcal{P}, \mathcal{V} \rangle$ to $\langle \mathcal{P}', \mathcal{V}' \rangle$:

1. $\mathcal{P}'$ initializes $\mathcal{P}$ with the input x and w and $\mathcal{V}'$ initializes $\mathcal{V}$ with the input x. Furthermore, $\mathcal{P}'$ and $\mathcal{V}'$ have common (untrusted) set up for FC (which consists of group description $\mathbb{G}$ and generators g, h for Pedersen commitment). $\mathcal{P}'$ sets $f_0' = \perp$, $st_0^p = \{x, \mathsf{w}\}$ and $\mathcal{V}'$ sets $\mathsf{st}_0^v = \{x\}$. Let ρ_p and ρ_v be the randomness of $\mathcal{P}$ and $\mathcal{V}$.
2. Let $r =$ number of rounds of $\langle \mathcal{P}, \mathcal{V} \rangle$. For $i = 1$ to r, $\mathcal{P}'$ and $\mathcal{V}'$ do the following:
   (a) $\mathcal{V}'$ computes $(c_i, \mathsf{st}_i^v) = V^{f_0', f_1', \ldots, f_{i-1}'}(st_{i-1}^v, \rho_v)$ and sends c_i to $\mathcal{P}'$.
   (b) $\mathcal{P}'$ computes $(f_i, \mathsf{st}_i^p) = P(c_1, \ldots, c_i, \mathsf{st}_{i-1}^p, \rho_p)$. Where $f_i = (m_i, \pi_i)$.
   (c) $\mathcal{P}'$ computes $\pi_i' = \mathsf{Com}(\mathsf{ck}, \pi_i)$.
   (d) $\mathcal{P}'$ sends $f_i' = (m_i, \pi_i')$.
   (e) $\mathcal{P}'$ sends $W_{i,j} = \mathsf{Open}(\mathsf{ck}, \pi_i'[j], \mathsf{aux})$ to $\mathcal{V}'$, for all $j \in J$, where J is the set of queried indices.
   (f) $\mathcal{V}'$ checks $\mathsf{Verify}(\mathsf{ck}, \pi_i'[j], W_{i,j}, \pi_i[j]) = 1$.
3. $\mathcal{V}'$ outputs $b = V^{f_0', f_1', \ldots, f_r}(\mathsf{st}_r^v, \rho_v)$.

Fig. 2. From IOP to Homomorphic IOP

Lemma 5.1. *Let $\langle \mathcal{P}, \mathcal{V} \rangle$ be an IOP-based zero-knowledge proof system with proof of knowledge property. Then the protocol $\langle \mathcal{P}', \mathcal{V}' \rangle$ obtained by using the above compiler is also an IOP-based zero-knowledge proof system with proof of knowledge property.*

Proof. We prove that $\langle \mathcal{P}', \mathcal{V}' \rangle$ is a secure IOP by proving completeness, knowledge-soundness and zero-knowledge.

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 $f_i = (m_i, \pi_i)$, where m_i is send to $\mathcal{V}$ in clear and $\mathcal{V}$ queries some of locations of π_i. Whereas, in $\langle \mathcal{P}', \mathcal{V}' \rangle$, $\mathcal{P}'$ sends $f_i' = (m_i, \pi_i')$. Simi-

lar to $\mathcal{V}$, $\mathcal{V}'$ receives m_i. In $\langle \mathcal{P}, \mathcal{V} \rangle$, $\mathcal{V}$ receives $\pi_i[j]$ depending on the $\mathcal{V}$'s choice of j. However, in $\langle \mathcal{P}', \mathcal{V}' \rangle$, $\mathcal{V}'$ receives $\pi'_i[j]$ corresponding to it's choice, where $\pi'_i[j]$ is the commitment of $\pi_i[j]$ and gets the corresponding opening from $\mathcal{P}'$. $\mathcal{V}'$ performs Verify to check if the above opening is correct or not, and for an honest prover, the opening is always correct. Now, $\mathcal{V}$ and $\mathcal{V}'$ hold the same data and perform the same verification. If $\mathcal{V}$ outputs $b = 1$, then $\mathcal{V}'$ also outputs $b = 1$. Therefore completeness holds for $\langle \mathcal{P}', \mathcal{V}' \rangle$.

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 $\mathcal{P}'^*$ be a prover such that $\langle \mathcal{P}'^*, \mathcal{V}'(x) \rangle = 1$. We claim that there is an PPT extractor $\mathcal{E}'$ that extracts a witness w corresponding to the statement x with very high probability.

$\mathcal{E}'$ rewinds $\mathcal{P}'^*$ sufficiently many times to extract π_i. Thereafter we consider following cases on how $\mathcal{P}'^*$ answers oracle queries for the remainder of the transcript.

Case I: $\mathcal{P}'^*$ opens $\pi'_i[j]$ to $\pi_i[j]$ for all i, j. Then consider a prover $\mathcal{P}^*$ sends $f_i = (m_i, \pi_i)$ where $\mathcal{P}'^*$ sends $f'_i = (m_i, \pi'_i)$ and $\pi_i = \mathsf{Open}(\pi'_i)$. $\mathcal{V}'$ in $\langle \mathcal{P}'^*, \mathcal{V}' \rangle$ and $\mathcal{V}$ in $\langle \mathcal{P}^*, \mathcal{V} \rangle$ performs the same verification. Since $\langle \mathcal{P}'^*, \mathcal{V}'(x) \rangle = 1$, therefore $\langle \mathcal{P}^*, \mathcal{V}(x) \rangle = 1$. By the proof of knowledge property of $\langle \mathcal{P}, \mathcal{V} \rangle$, there is an PPT extractor $\mathcal{E}$ that extracts a witness w with very high probability. $\mathcal{E}'$ runs the same algorithm and outputs w with the same probability.

Case II: $\mathcal{P}'^*$ opens at least one $\pi'_i[j]$ for some i, j to some value $\pi^*_i[j]$ such that $\pi^*_i[j] \neq \pi_i[j]$. This breaks the binding property of the commitment scheme.

Therefore, $\Pr[(x, \mathcal{E}'^{\mathcal{P}'^*}(x)) \in R] = \Pr[(x, \mathcal{E}^{\mathcal{P}^*}(x)) \in R] - O(|x|)\epsilon_c$, where ϵ_c is the probability that any PPT algorithm can break the binding property of Com. $\Pr[\langle \mathcal{P}'^*, \mathcal{V}'(x) \rangle = 1] = \Pr[\langle \mathcal{P}^*, \mathcal{V}(x) \rangle = 1]$. Due to the proof of knowledge property of $\langle \mathcal{P}, \mathcal{V} \rangle$, we have $\Pr[(x, \mathcal{E}^{\mathcal{P}^*}(x)) \in R] \geq \Pr[\langle \mathcal{P}^*, \mathcal{V}(x) \rangle = 1] - \mu(|x|)$. Hence, $\Pr[(x, \mathcal{E}'^{\mathcal{P}'^*}(x)) \in R] \geq \Pr[\langle \mathcal{P}'^*, \mathcal{V}'(x) \rangle = 1] - O(|x|)\epsilon_c - \mu(|x|)$. Set $\mu'(x) = O(|x|)\epsilon_c + \mu(|x|)$. Since μ' is a negligible function, therefore $\langle \mathcal{P}', \mathcal{V}' \rangle$ has the proof of knowledge property.

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 $\mathcal{S}$. Let $\pi_i[j]$ be the parts of the transcript τ corresponding to the ith round's oracle response when queried at the jth location. Then $\mathcal{S}'$ commits to π_i and gets π'_i. Finally the simulated transcript, τ', generated by $\mathcal{S}'$ by replacing $\pi_i[j]$ with $\pi'_i[j]$ and adding corresponding opening of $\pi'_i[j]$ is indistinguishable from a real execution of $\langle \mathcal{P}', \mathcal{V}' \rangle$. This ensures the zero-knowledge property. □

Distributed Proof Generation

Let $\langle \mathcal{P}, \mathcal{V} \rangle$ be an IOP-based zero-knowledge proof system. We define a $(\mathsf{D}+1)$ party protocol $\langle \mathsf{Dist}(\mathcal{P}, \mathsf{D}), \mathcal{V} \rangle$ with one verifier $\mathcal{V}$ and D identical copies of $\mathcal{P}$ algorithm, say $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_{\mathsf{D}}\}$. In this protocol $\mathcal{V}$ receives messages from one designated party, A, from $\mathcal{P}$ and parties in $\mathcal{P}$ can interact among themselves. In $\langle \mathsf{Dist}(\mathcal{P}, \mathsf{D}), \mathcal{V} \rangle$, $\mathcal{P}_\xi$ has input x, w_ξ for all $\xi \in \mathsf{D}$ and $\mathcal{V}$ has input x. Since $(f_i, \mathsf{st}^p_i) = P(c_1, \ldots, c_i, \mathsf{st}^p_{i-1}, \rho_p)$, parties in $\mathcal{P}$ jointly compute $P(\cdot)$ with public inputs $c_1, \ldots, c_i$ and $\mathcal{P}_\xi$'s private input $\{\mathsf{st}^{\mathcal{P}_\xi}_{i-1}, \rho_{\mathcal{P}_\xi}\}$. To obtain a linear sharing of (f_i, st^p_i), parties in $\mathcal{P}$ perform a t-secure MPC, where at most $t (< \mathsf{D})$ parties can be corrupted. For $\xi \in [\mathsf{D}]$, each $\mathcal{P}_\xi$ obtains $(\langle f_i \rangle, \langle \mathsf{st}^p_i \rangle)$ where $\langle \cdot \rangle$ is a t-out of n sharing. If $P(\cdot)$ is a linear function then no interaction is required among the provers. But $P(\cdot)$ need not be a linear function, in which case communication among the parties in $\mathcal{P}$ is required.

$\langle \mathsf{Dist}(\mathcal{P}, \mathsf{D}), \mathcal{V} \rangle$ **from** $\langle \mathcal{P}, \mathcal{V} \rangle$**:**

1. Initiate $\mathsf{D}+1$ party protocol with D copies of $\mathcal{P}$, say $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_{\mathsf{D}}\}$ and one $\mathcal{V}$. $\mathcal{P}_\xi$ in $\mathcal{P}$ starts the protocol with inputs (x, w_ξ) and $\mathcal{V}$'s input is x.
2. $\mathcal{P}_\xi$ sets $\mathsf{st}^{\mathcal{P}_\xi}_0 = \{x, \mathsf{w}_\xi\}$, A sends $f'_0 = \perp$ and $\mathcal{V}$ sets $\mathsf{st}^v_0 = \{x\}$.
3. Let r = number of rounds of $\langle \mathcal{P}, \mathcal{V} \rangle$. For $i = 1$ to r, parties do the following:
   (a) $\mathcal{V}$ computes $(c_i, \mathsf{st}^v_i) = V^{f'_0, f'_1, \ldots, f'_{i-1}}(st^v_{i-1}, \rho_v)$ and sends c_i to all parties in $\mathcal{P}$.
   (b) Parties in $\mathcal{P}$ jointly perform $(f_i, \mathsf{st}^p_i) = P(c_1, \ldots, c_i, \{\mathsf{st}^{\mathcal{P}_\xi}_{i-1}, \rho_{\mathcal{P}_\xi}\}_{\xi \in [\mathsf{D}]})$. $\mathcal{P}_\xi$ obtains a linear sharing of (f_i, st^p_i), say $(\langle f_i \rangle, \langle \mathsf{st}^p_i \rangle)$.
   (c) $\mathcal{P}_\xi$ sends $\langle f_i \rangle_\xi$ to A, a chosen party from $\mathcal{P}$.
   (d) A combines $\langle f_i \rangle$ to obtain f_i. A sends messages $f_i = (m_i, \pi_i)$.
4. $\mathcal{V}$ outputs $b = V^{f_0, f_1, \ldots, f_r}(\mathsf{st}^v_r, \rho_v)$.

Fig. 3. From single prover protocol to distributed prover protocol

Lemma 5.2. *Let $\langle \mathcal{P}', \mathcal{V}' \rangle$ be an homomorphic IOP-based protocol obtained by using the above mentioned compiler. If the witness, w, is shared among D provers $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_{\mathsf{D}}\}$ additively, then $\langle \mathsf{Dist}(\mathcal{P}', \mathsf{D}), \mathcal{V}' \rangle$ realizes $\mathcal{F}_{DPZK}$ functionality securely.*

Proof. In $\langle \mathsf{Dist}(\mathcal{P}', \mathsf{D}), \mathcal{V}' \rangle$, $\mathcal{P}'_\xi$ obtains $\langle f_i \rangle_\xi = (\langle m_i \rangle_\xi, \langle \pi_i \rangle_\xi)$. $\mathcal{P}'_\xi$ commits to $\langle \pi_i \rangle_\xi$ and obtains $\langle \pi'_i \rangle_\xi$. $\mathcal{P}'_\xi$ sends $\langle f'_i \rangle_\xi = (\langle m_i \rangle_\xi, \langle \pi'_i \rangle_\xi)$. Due to the linearity of $\langle \cdot \rangle$, the combine operation for A is simple. A computes $m_i = \sum_{\xi \in [\mathsf{D}]} \langle m_i \rangle$ and $\pi'_i = \prod_{\xi \in [\mathsf{D}]} \langle \pi'_i \rangle$ and sends $f'_i = (m_i, \pi'_i)$. Let J be the set of indices queried by $\mathcal{V}'$, where $|J|$ is bounded by the query complexity q of $\langle \mathcal{P}, \mathcal{V} \rangle$. In response to these queries, $\mathcal{P}'_\xi$ sends the corresponding openings. That is, $\mathcal{P}'_\xi$ sends $\mathsf{Open}(\langle \pi'_i \rangle[j]) = \langle \pi_i \rangle[j]$ to A, for all $j \in J$. A sends $\pi_i[j] = \sum_{\xi \in [\mathsf{D}]} \langle \pi_i \rangle[j]$ to $\mathcal{V}'$.

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

Let $\mathcal{P}^*$ be a set of D provers such that $\langle \mathsf{Dist}(\mathcal{P}'^*, \mathsf{D}), \mathcal{V}'(x) \rangle = 1$. Let $\mathcal{P}'^*$ be a prover that sends $f'_i = (m_i, \pi'_i)$ where A in $\langle \mathsf{Dist}(\mathcal{P}'^*, \mathsf{D}), \mathcal{V} \rangle$ sends the same f'_i. $\langle \mathcal{P}'^*, \mathcal{V}' \rangle = 1$ since the verification in both $\langle \mathsf{Dist}(\mathcal{P}', \mathsf{D}), \mathcal{V}' \rangle$ and $\langle \mathcal{P}', \mathcal{V}' \rangle$ is the same. Due to the proof of knowledge property of $\langle \mathcal{P}', \mathcal{V}' \rangle$, there is an PPT extractor $\mathcal{E}'$ that outputs w with high probability. $\mathcal{E}_{DP}$ be the extractor that has oracle access to $\mathcal{P}^*$ and runs $\mathcal{E}'$ algorithm which has oracle access to the $\mathcal{P}'^*$, where $\mathcal{P}'^*$ sends the same message as the A in $\langle \mathsf{Dist}(\mathcal{P}'^*, \mathsf{D}), \mathcal{V} \rangle$. If $\mathcal{E}'$ outputs w with probability p, $\mathcal{E}_{DP}$ outputs w with the same probability p.

Therefore, $\Pr[(x, \mathcal{E}_{DP}^{\mathcal{P}^*}(x)) \in R] = \Pr[(x, \mathcal{E}'^{\mathcal{P}'^*}(x)) \in R]$. Since $\langle \mathcal{P}'^*, \mathcal{V}' \rangle$ has proof of knowledge property, therefore $\Pr[(x, \mathcal{E}'^{\mathcal{P}'^*}(x)) \in R] \geq \Pr[\langle \mathcal{P}'^*, \mathcal{V}'(x) \rangle = 1] - \mu'(|x|)$ and $\Pr[\langle \mathsf{Dist}(\mathcal{P}'^*, \mathsf{D}), \mathcal{V}'(x) \rangle = 1] = \Pr[\langle \mathcal{P}'^*, \mathcal{V}'(x) \rangle = 1]$. Hence, $\Pr[(x, \mathcal{E}_{DP}^{\mathcal{P}^*}(x)) \in R] \geq \Pr[\langle \mathsf{Dist}(\mathcal{P}'^*, \mathsf{D}), \mathcal{V}'(x) \rangle = 1] - \mu'(|x|)$.

Zero-Knowledge: Since in $\langle \mathsf{Dist}(\mathcal{P}', \mathsf{D}), \mathcal{V}' \rangle$, 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 Confidentiality: Let at the ith round, provers run t-secure MPC, Π_P, to obtain $\langle f_i \rangle$ and $\langle \mathsf{st}_i^p \rangle$ such that $\sum \langle m_i \rangle = m_i$ and $\sum \langle \pi_i \rangle = \pi_i$, where $f_i = (m_i, \pi_i)$.

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 a transcript τ.
- On behalf of the verifier, $\mathcal{S}$ sets the challenge c_i obtained from the transcript τ, and corresponding response m_i and $\{\pi_i[j]\}_{j \in J}$, where J be the set of queried locations. $\mathcal{S}$ picks a π_i that is consistent with $\{\pi_i[j]\}_{j \in J}$. This is possible due to the bounded independence property, that is π_i remains random (independent of the witness w) even after revealing $|J|$ locations of π_i.
- If provers run MPC, Π_P to obtain $\langle f_i \rangle$ and $\langle \mathsf{st}_i^p \rangle$, and $\mathcal{S}_P$ be the simulator. Consider $\{st_{i-1}^j\}_{j \in C}$ be the inputs of the corrupted parties to the protocols Π_P. Then $\mathcal{S}$ executes $\mathcal{S}_P$ with inputs $\{st_{i-1}^j\}_{j \in C}$, $\langle f_i \rangle$ and $\langle \mathsf{st}_i^p \rangle$ correspondingly.
- Finally, $\mathcal{S}$ sends $\{\langle m_i \rangle_j\}_{j \notin C}$ and $\{\langle \pi'_i \rangle_j\}_{j \notin C}$to A.

Here note that the view generated by $\mathcal{S}$ is indistinguishable from a real execution of the protocol. We establish this by the following argument.

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

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*.

Privacy against A holds due to the hiding property of the commitment scheme and the zero-knowledge property of the underlying protocol. A's view consists of $(\langle m_i \rangle, \langle \pi'_i \rangle)$ and $\mathsf{Open}(\langle \pi'_i \rangle[j]) = \langle \pi_i \rangle[j]$ for all $j \in J$, where $|J|$ is bounded by the query complexity. The simulator runs the zero knowledge simulator internally and obtains m_i and $\pi_i[j]$. It creates additive sharing of m_i and $\pi_i[j]$, that is, $\langle m_i \rangle$ and $\langle \pi_i \rangle[j]$. The simulator picks $\langle \pi_i \rangle$ such that the obtained values are consistent. This is possible since the number of opened values are bounded by query complexity and that ensures it does not leak any information about the underlying message. Then the simulator commits to $\langle \pi_i \rangle$ and obtains $\langle \pi'_i \rangle$. This simulated view is indistinguishable from a real world A's 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 q many queries to the oracle π_i, that adds q group exponentiations in the verifier's complexity in $\langle \mathcal{P}', \mathcal{V}' \rangle$. In general, $\langle \mathsf{Dist}(\mathcal{P}', \mathsf{D}), \mathcal{V}' \rangle$ may require $O(N)$

multiplication in each round for most of the protocols. Note that our construction uses public key cryptography heavily. Reducing the usage of it will improve the efficiency significantly and will lead to a more practical solution. We believe that obtaining a practical solution of this problem is an interesting problem and will lead to further research.

6 DPZK Instantiation

In this section, we present the distributed proof generation from various protocols. We call the distributed variant of a protocol "X" as "D-X". We start with Ligero [1], where we discuss the efficiency of the distributed prover variant, D-Ligero. We optimize D-Ligero and provide a zero-knowledge protocol, named Graphene, that achieves better proof size, reduced verification time as well as communication among the provers. We explain the high-level idea of Graphene in this section and provide the details in Appendix B. The distributed version of Graphene is given in Appendix C. We then move on to the state-of-the-art IOP-based zero-knowledge protocol Aurora [6]. We discuss the efficiency of the compiled version of Aurora that enables distributed proof generation. This multi-oracle IOP shows the overhead of the compilation. We also discuss distributed proof generation from non-IOP protocols such as Bulletproofs [13] and Spartan [42]. We study the efficiency and challenges of the distributed prover variants of these protocols.

6.1 Protocols with Single Oracle

D-Ligero:
We compile the Ligero protocol, and instantiate the functional commitment scheme FC by Pedersen vector commitment scheme. Note that this FC has homomorphic property and the Open algorithm outputs the committed value $\mathbf{m}$ as the witness. The Verify algorithm can be replaced by an inner product argument where $\mathbf{m}$ is the witness.

The proof for an R1CS instance in Ligero constitutes of the check that verifies if the witness w satisfies the following condition: $x = A\mathsf{w} \wedge y = B\mathsf{w} \wedge z = C\mathsf{w} \wedge x \circ y = z$, where is A, B, C matrices are dependent on the R1CS instance. The above check is segmented into three checks: (i) Interleaved check, (ii) Linear check, and (iii) Quadratic check.

In Ligero, $\mathcal{P}$ rewrites the vectors w, x, y, z as matrices in a canonical way, that is, the first ℓ entries are set as the first row of the matrix, next ℓ entries form the second row and so on. This way a matrix is formed which has, say m rows and ℓ columns, where $m = \ell = O(\sqrt{N})$. This is referred to as the canonical matrix form. The prover encodes these matrices using Reed–Solomon(RS)-encoding to encode each row of the matrices. In the compiled Ligero protocol, $\mathcal{P}$ computes commitment of each column of these matrices. The commitment scheme takes m length messages. In the original Ligero protocol, t (depends on the query complexity) many columns and a linear combination of the rows are opened to the verifier. This enforces a bound on the proof size in Ligero. We circumvent this bound by replacing the openings with inner product checks.

In the interleaved check, $\mathcal{P}$ proves that the rows of the oracle matrices are codewords. $\mathcal{P}$ proves the claim probabilistically by showing that a random linear combination of the rows is a code word. $\mathcal{V}$ uniformly picks the coefficients for the linear combination. Therefore, if one or more rows in the matrices are not correct codewords, then the linear combination is not a codeword with high probability. Furthermore, $\mathcal{V}$ queries the oracle with few indices (the number of queries is bounded by the security parameter t for zero-knowledge) and receives the corresponding columns. Upon receiving the columns, $\mathcal{V}$ performs the same linear combination and checks that it is consistent with the row received from $\mathcal{P}$. In the linear check, $\mathcal{P}$ proves the knowledge of a vector x such that $Ax = b$ holds for public matrix A and vector b. This check is reduced to a probabilistic check $r^T Ax = r^T b$, where r is a random vector picked by $\mathcal{V}$. $\mathcal{P}$ computes a polynomial $p(x)$ such that the sum of the evaluations on publicly known points is equal to $r^T b$, and sends this $p(x)$ to $\mathcal{V}$. $\mathcal{V}$ checks if the above condition holds, and it further checks if $p(x)$ is constructed correctly from r, A, and w. $\mathcal{V}$ validates this by querying some locations of the oracle. In the quadratic check, $\mathcal{P}$ proves that $x \circ y = z$ ($\circ$ denotes the Hadamard product of two vectors), where the corresponding encoded values are set as oracles. $\mathcal{V}$ gives a randomly sampled challenge vector r to $\mathcal{P}$. $\mathcal{P}$ constructs a polynomial $p(x)$ using r and the encoding polynomials of x, y, z. This polynomial should evaluate to 0 on publicly known points. $\mathcal{P}$ sends the polynomial $p(x)$ to $\mathcal{V}$. Upon receiving $p(x)$, $\mathcal{V}$ checks if the above condition is true or not. Furthermore, it checks whether $p(x)$ is correctly constructed. For this, it queries some locations of the oracles.

In the compiled protocol, $\mathcal{V}$ obtains commitments corresponding to the queried columns, and instead of

sending the openings of those columns, we perform the inner product argument. Since the opening of the columns is not sent, we can elongate the size of the columns without increasing the proof size. The inner product argument requires communication of $O(\log N)$ elements, where N is the size of the witness vector. Therefore the newly obtained protocol has proof size $O(\ell+\log(m))$, where ℓ is the size of the row, and m is the size of the column. However, verifying these inner products requires $O(N)$ exponentiations for a witness vector of dimension N. Thus increasing the column size unreasonably may result in a better proof size but increases the verification time, which is undesirable. Furthermore, for the quadratic check protocol, the polynomial $p(x)$ depends non-linearly on the witness since it requires multiplication of the polynomials used for encoding x, y. Therefore, provers in $\mathcal{P}$ engage in a secure computation protocol where they obtain an additive sharing of $p(x)$. In general, this incurs communication for computing a circuit of depth 1 with $O(N)$ multiplications. Nevertheless, if the size of the shared circuit is sufficiently low (smaller than the row size in the canonical matrix), then it is possible to embed the shared circuit in a row. In this case, MPC is required only for the row that contains the shared circuit.

Graphene and D-Graphene:
In Graphene, we follow the same approach as in Ligero. We start with the extended witness w, consider it as a 3 dimensional matrix of size $p \times m \times s$ where p 2 dimensional matrices of size $m \times s$, called a slice, are stacked one after another. In contrast to Ligero, where each row is encoded separately, we encode each two-dimensional slice separately. Let $\ell = s + \mathsf{b}$, $h > 2m$, $n > 2\ell$ where b is the bounded independence parameter depending on the query complexity. Let $\boldsymbol{\zeta} = \{\zeta_1, \ldots, \zeta_\ell\}$, $\boldsymbol{\eta} = \{\eta_1, \ldots, \eta_n\}$ and $\boldsymbol{\alpha} = \{\alpha_1, \ldots, \alpha_h\}$ be public set of points in $\mathbb{F}$. We define $G = \{(\alpha_j, \zeta_k) : j \in [m], k \in [\ell]\}$ and $H = \{(\alpha_j, \eta_k) : j \in [h], k \in [n]\}$ to be the interpolation domain and evaluation domain respectively. To encode the ith slice, we construct a bivariate polynomial $Q^i(x, y)$ such that $Q^i(\alpha_j, \zeta_k) = \mathsf{w}[i, j, k]$ and $\deg_x(Q^i) < m$ and $\deg_y(Q^i) < \ell$. The encoded witness U is such that $\mathsf{U}[i, j, k] = Q^i(\alpha_j, \eta_k)$ for $i \in [p], j \in [h], k \in [n]$. Then each column of each slice is committed using Pedersen vector commitment. Here we instantiate the homomorphic FC with the Pedersen commitment scheme with message length h. Finally, $\mathcal{V}$ is provided oracle access to the committed matrix. We design interleaved check protocol over committed values which is performed together with the linear check and quadratic check. The details are presented in Appendix B.2. The linear check is similar to Ligero, where a probabilistic reduction is performed. $\mathcal{V}$ sends a random r followed by both $\mathcal{P}$ and $\mathcal{V}$ locally computing $R = r^T A$. Using $\boldsymbol{\alpha}, \boldsymbol{\eta}$ $\mathcal{P}$, $\mathcal{V}$ interpolate p polynomials $R^i(x, y)$ such that $R^i(\alpha_j, \zeta_k) = R[i, j, k]$ for $i \in [p], j \in [m], k \in [s]$ and $\deg_x(R^i) < m$, $\deg_y(R^i) < s$ for all i. $\mathcal{P}$ computes polynomial $p_j(y) = \sum_{i\in[p]} R^i(\alpha_j, y)Q^i(\alpha_j, y)$. If the witness is correct then $r^T A\mathsf{w} = r^T b \Rightarrow \langle R, \mathsf{w}\rangle = r^T b \Rightarrow \sum_{j\in[m],k\in[s]} p_j(\zeta_k) = r^T b$. Furthermore, $p_j(\eta_k) = \sum_{i\in[p]} R^i(\alpha_j, \eta)\mathsf{U}[i, j, k]$ due to the encoding. $\mathcal{P}$ constructs a matrix P of size $h \times n$ such that $P[j, k] = p_j(\eta)$ and commits to P. By providing partial opening using inner product arguments from Bulletproofs, $\mathcal{P}$ proves that the polynomials $p_j(y)$ satisfies the following conditions:

- $p_j(\eta_k) = \sum_{i\in[p]} R^i(\alpha_j, \eta)\mathsf{U}[i, j, k]$ for all $j \in [h], k \in [n]$
- $\sum_{j\in[m],k\in[s]} p_j(\zeta_k) = r^T b$

Similar to the linear check, in the quadratic check, $\mathcal{P}$ encodes x, y, z to obtain $\mathsf{U}_x, \mathsf{U}_y, \mathsf{U}_z$ and lets Q^i_x, Q^i_y, Q^i_z be the respective polynomials. $\mathcal{P}$ constructs polynomials Q^i such that $Q^i = Q^i_x \cdot Q^i_y - Q^i_z$. Since $x \circ y = z$, we have $Q^i(\alpha_j, \zeta_k) = 0$ for all $i \in [p], j \in [m], k \in [s]$. To check this, $\mathcal{V}$ sends a random vector $r \in \mathbb{F}^p$ as a challenge. $\mathcal{P}$ locally computes $p_j(\cdot) = \sum_{i\in[p]} r_i Q^i(\alpha_j, \cdot)$. Analogous to the linear check, $\mathcal{P}$ forms P matrix and commits to it, and further proves that P satisfies the following:

- $p_j(\eta_k) = \sum_{i\in[p]} r^i \left[\mathsf{U}_x[i, j, k] \cdot \mathsf{U}_y[i, j, k] - \mathsf{U}_z[i, j, k]\right]$ for all $j \in [h], k \in [n]$
- $p_j(\zeta_k) = 0$ for all $j \in [m], k \in [s]$

To obtain D-Graphene, parties in $\mathcal{P}$ execute secure multiplication to get an additive sharing of $\mathsf{U}_x[i, j, k] \cdot \mathsf{U}_y[i, j, k]$.The remaining steps in both linear and quadratic check do not require any interaction among the provers. Therefore, the provers interact to evaluate a circuit with $O(N)$ many multiplications with depth 1. However, with smaller shared circuit, where the shared circuit can be embedded into a column of a slice such that multiplication is required only for that column. The details of the protocol are given in Appendix B.

6.2 Multiple Oracle IOP

D-Aurora:
In Aurora [6], 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 oracle-queries to complete the verification. We can convert Aurora using our compiler, where we instantiate FC using Pedersen commitment, such that it supports DPZK. The compiled version retains the oracle size, proof size, and the number of rounds. However, the prover time and the verification time increase by $O(N \log N)$ group exponentiations due to oracle construction and validation costs. The distributed version needs secure evaluation of circuits with at least depth one and $O(N)$ multiplication gates in each round.

6.3 Non-IOP Protocols

D-Bulletproofs:
In Bulletproofs [13], the proof of an R1CS instance is reduced to an inner product argument which is proved recursively for the succinctness of the proof size. The prover constructs two vector polynomials $l(X), r(X)$ of degree 1 using the witness, statement and challenges from the verifier. It commits to the quadratic polynomial $t(X) = \langle l(X), r(X) \rangle$ where $t(X) = \sum_{i=0}^{d} \sum_{j=0}^{i} \langle l_i, r_j \rangle X^{i+j}$ such that l_i and r_j are $l(X)$'s ith and $r(X)$'s jth vector coefficients respectively. The verifier sends a random point x and prover obtains $\boldsymbol{l} = l(x)$, $\boldsymbol{r} = r(x)$ and $t = t(x)$. Using the inner product argument the prover proves that $\langle \boldsymbol{l}, \boldsymbol{r} \rangle = t$.

In D-Bulletproofs, provers start with additive sharings of $l(X), r(X)$ and perform secure multiplications to obtain additive sharing of $t(X)$. This requires $O(N)$ multiplications. Upon receiving x, each prover locally obtains a sharing of $\boldsymbol{l}, \boldsymbol{r}, t$ and sends these values to the aggregator. The aggregator performs the inner product argument with the verifier to complete the proof.

D-Spartan:
In Spartan [42], an R1CS circuit is represented by 3 matrices A, B, C. A witness w for the true instance satisfies $Az \circ Bz = Cz$, where $z = (x, 1, \mathsf{w})$ for public input x. In Spartan, prover obtains low-degree polynomial extensions of A, B, C, z. Upon receiving the challenges from the verifier, the prover constructs a polynomial f. Then, a sum-check is performed to ensure that f satisfies a consistency condition.

For the distributed proof generation, provers initiate the protocol with an additive sharing of the witness w and achieve an additive sharing of f by performing $O(N^2)$ secure multiplications. In Spartan, zero-knowledge is achieved by producing proofs of dot products. In the distributed setting, this either requires each prover to open its witness of the dot product to the aggregator or all the provers produce a distributed proof for the dot products. The latter case makes it a circular problem, whereas the former one has a privacy issue. Even if a similar protocol could overcome the privacy issue, it would likely still suffer from the high communication complexity of $O(N^2)$ secure multiplication. The above discussion further emphasizes the non-triviality of obtaining DPZK from any ZK protocol.

Protocols	r_{pr}	c_{pr}
D-Ligero	1	$O(\sqrt{N})$
D-Aurora	$\log(N)$	$O((N \log(N))$
D-Bulletproofs	1	$O(N_s)$
D-Graphene	1	$O(N^{1-2/c})$

Fig. 4. Comparison amongst the DPZKs. Here N is the size of the circuit and, c is a positive integer of our choice. $*$ indicates non-private variant of Spartan.

Here we provide a comparison of DPZK protocols in the setting where the size of the shared circuit is low ($N_s < \sqrt{N}$) and the number of provers is small ($O(1)$).

7 Conclusion

In this work, we presented a formal definition of distributed-prover zero-knowledge (DPZK) and a generic compiler that provides a DPZK protocol from any IOP-based zero-knowledge protocol. We discuss instantiations of this compiler with some recent IOP ZK protocols, and discuss DPZK versions of some state-of-the-art non-IOP ZK protocols.

Furthermore, we provided an efficient single-prover protocol Graphene that achieves DPZK with a minimal overhead of communication among the provers while retaining the proof size and verification time.

In all these protocols, we prove witness confidentiality against semi-honest provers. Extending this to malicious provers is left as an open problem. Another interesting future direction is to construct a universal composable (UC) DPZK.

Acknowledgement

We thank the reviewers for their valuable comments, which helped improve the paper. Arpita Patra would

like to acknowledge financial support from SERB MATRICS (Theoretical Sciences) Grant 2020, Google India AI/ML Research Award 2020, DST National Mission on Interdisciplinary Cyber-Physical Systems (NM-CPS) 2020 and National Security Council, India.

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] C. Baum, A. J. Malozemoff, M. B. Rosen, and P. Scholl. Mac'n'cheese: Zero-knowledge proofs for arithmetic circuits with nested disjunctions. *IACR Cryptol. ePrint Arch.*, 2020:1410, 2020.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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

[8] 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.

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

[10] 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.

[11] 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.

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

[13] 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.

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

[15] R. Canetti. Universally composable security: a new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145, 2001.

[16] 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.

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

[18] 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.

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

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

[21] 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.

[22] S. Dittmer, Y. Ishai, and R. Ostrovsky. Line-point zero knowledge and its applications. In *2nd Conference on Information-Theoretic Cryptography (ITC 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[23] 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.

[24] 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.

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

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

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

[28] 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.

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

[30] 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.

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

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

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

[34] B. Libert, S. Ramanna, and M. Yung. Functional commitment schemes: From polynomial commitments to pairing-based accumulators from simple assumptions. In *43rd International Colloquium on Automata, Languages and Programming (ICALP 2016)*, 2016.

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

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

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

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

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

[40] 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.

[41] 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.

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

[43] C. Weng, K. Yang, J. Katz, and X. Wang. Wolverine: fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1074–1091. IEEE, 2021.

[44] 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.

[45] K. Yang, P. Sarkar, C. Weng, and X. Wang. Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. *IACR Cryptol. ePrint Arch.*, 2021:76, 2021.

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

A Preliminaries

A.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.

A.2 Coding Theory primitives

Definition 2 (Linear Codes)**.** *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 3 (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 A.1.

Definition 4 (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.*

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 5 (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 A.1 ([36])**.** *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 $(i, j) \in S_1 \times S_2$.*

A.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.

Proposition A.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 $(r_1, \ldots, r_p) \leftarrow\!\!\$\, \mathbb{F}^p$.

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:

$$\mathrm{Aff}(u_0, U) := \{u_0 + r^T U : r \in \mathbb{F}^m\} \tag{1}$$

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.3. *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.4 (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.5 (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.3, 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.4 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 A.2.

Proof of Proposition A.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.5 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. □

A.4 Vector Commitment Schemes and Inner-product Arguments

Functional Commitment for linear functions

Let $\mathcal{D}$ be a domain and consider linear functions $\langle \cdot, \cdot \rangle : \mathcal{D}^n \times \mathcal{D}^n \to \mathcal{D}$ defined by $\langle \mathbf{x}, \mathbf{m} \rangle = \sum_{i=1}^n x_i m_i$ for $\mathbf{x}, \mathbf{m} \in \mathcal{D}^n$ with $\mathbf{x} = (x_1, \ldots, x_n)$, $\mathbf{m} = (m_1, \ldots, m_n)$. A functional commitment scheme FC is a tuple of four probabilistic polynomial-time algorithms - $(\mathsf{SetUp}, \mathsf{Com}, \mathsf{Open}, \mathsf{Verify})$.

$\mathsf{SetUp}(1^\lambda, 1^n)$: takes in a security parameter $\lambda \in \mathbb{N}$, size of the message $n \in \mathsf{poly}(\lambda)$ and outputs a commitment key CK.

$\mathsf{Com}(CK, \mathbf{m})$: takes commitment key CK and a n length message $\mathbf{m}$ as input. It outputs a commitment C for $\mathbf{m}$ and auxiliary information denotes aux.

$\mathsf{Open}(CK, C, \mathsf{aux}, \mathbf{x})$: takes as input the commitment key CK, a commitment C (to $\mathbf{m}$), auxiliary information (possibly containing $\mathbf{m}$) and a vector $\mathbf{x} \in \mathcal{D}^n$; computes a witness W_y for $y = \langle \mathbf{x}, \mathbf{m} \rangle$ i.e., W_y is a witness for the fact that the linear function defined by $\mathbf{x}$ when evaluated on $\mathbf{m}$ gives y.

$\mathsf{Verify}(CK, C, W_y, \mathbf{x}, y)$: takes as input the commitment key CK, a commitment C, a witness W_y, a vector $\mathbf{x} \in \mathcal{D}^n$ and $y \in \mathcal{D}$; outputs W_y is a witness for C being a commitment for some $\mathbf{m} \in \mathcal{D}^n$ such that $\langle \mathbf{x}, \mathbf{m} \rangle = y$ and outputs 0 otherwise.

Correctness: For every $CK \leftarrow \mathsf{SetUp}(1^\lambda, 1^n)$, for all $\mathbf{x}, \mathbf{m} \in \mathcal{D}^n$, if $(C, \mathsf{aux}) \leftarrow \mathsf{Com}(CK, \mathbf{m})$ and $W_y \leftarrow \mathsf{Open}(CK, C, \mathsf{aux}, \mathbf{x})$, then $\mathsf{Verify}(CK, C, W_y, \mathbf{x}, y) = 1$ with probability 1.

Hiding: For a key CK generated by an honest setup, for all $\mathbf{m_1}, \mathbf{m_2} \in \mathcal{D}^n$ with $\mathbf{m_1} \neq \mathbf{m_2}$, the two distributions $\{CK, \mathsf{Com}(CK, \mathbf{m_1})\}$ and $\{CK, \mathsf{Com}(CK, \mathbf{m_2})\}$ are indistinguishable.

Function binding: A functional commitment scheme $\mathsf{FC} = (\mathsf{SetUp}, \mathsf{Com}, \mathsf{Open}, \mathsf{Verify})$ for $(\mathcal{D}, n, \langle \cdot, \cdot \rangle)$ is said to be computationally binding if any PPT adversary $\mathcal{A}$ has negligible advantage in winning the following game.

1. The challenger generates CK by running $\mathsf{SetUp}(1^\lambda, 1^n)$ and gives CK to $\mathcal{A}$.
2. The adversary $\mathcal{A}$ outputs a commitment C, a vector $\mathbf{x} \in \mathcal{D}^n$, two values $y, y' \in \mathcal{D}$ and two witnesses $W_y, W_{y'}$. We say that $\mathcal{A}$ wins the game if the following conditions hold.
   - $y' = y$;
   - $\mathsf{Verify}(CK, C, W_y, \mathbf{x}, y)$ $= \mathsf{Verify}(CK, C, W_{y'}, \mathbf{x}, y') = 1$.

Functional commitment is generalization of vector commitments. In our constructions, vector commitment suffices. We initiate FC with Pedersen commitment scheme or Pedersen vector commitment scheme accordingly.

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 [13], 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 [13]. 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 $\langle x, z \rangle = v$.

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 [10, 13])**.** *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 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 [6, 16, 29], and circuit compilers such as [24] 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:G.

B.1.1 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 do-

main 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.

Lemma B.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. □

B.1.2 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 $U[i,\cdot,\cdot] \in \mathsf{RS}_{\boldsymbol{\eta}}[n,\ell] \otimes \mathsf{RS}_{\boldsymbol{\alpha}}[h,m]$.

B.1.3 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.

B.1.4 Oracle Construction

Unlike prior IOP constructions such as [1, 6], 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.

B.1.5 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. [36]). We then define $\mathsf{Dec}(\mathsf{U}^*)$ as $\mathsf{Dec}(\mathsf{U})$.

B.2 Graphene

B.2.1 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 B.1.1, and Section B.1.3 respectively. $\mathcal{P}$ constructs an oracle as described in Section B.1.4. 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 B.1.3 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 B.1.3) 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:

$$\prod_{a=1}^{\ell} (\tilde{c}_a)^{\Lambda^T_{n,\ell}[a, k_u]} = \prod_{i=1}^{p} (\pi[i, k_u])^{\rho_i} \text{ for } u \in [t]. \quad (2)$$

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 $V_{iu} = (\mathsf{U}[i, 1, k_u], \ldots, \mathsf{U}[i, m, k_u])$.

B.2.2 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 B.2.1. 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 B.1.3) 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:

$$\prod_{a\in[\ell]} \tilde{c}_a^{\Lambda^T_{n,\ell}[a,k_u]} = \prod_{i=1}^{p} (\pi_x[i,k_u])^{\rho_i} (\pi_y[i,k_u])^{\rho_{p+i}} (\pi_z[i,k_u])^{\rho_{2p+i}} \tag{3}$$

B.2.3 Graphene

We use the linear and quadratic check protocols from Section B.2.1 and Section B.2.2 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:

$$\left[\gamma_x I \,|\gamma_y I\,|\gamma_z I\,| - (\gamma_x A + \gamma_y B + \gamma_z C)\right] \begin{bmatrix} \mathsf{w}_x \\ \mathsf{w}_y \\ \mathsf{w}_z \\ \mathsf{w} \end{bmatrix} = \mathbf{0} \tag{4}$$

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 B.1.1 and B.1.4. Note that $\mathcal{O}$ is a $4p \times n$ matrix. Let D be the $M \times 4N$ sized matrix given in Equation 4. For the R1CS instance (A, B, C), $\mathcal{P}$ and $\mathcal{V}$ run linear check protocol B.2.1 for $D\overline{\mathsf{w}} = \mathbf{0}$ and run quadratic check protocol for $\mathsf{w}_x \circ \mathsf{w}_y = \mathsf{w}_z$.

B.2.4 Linear Check Protocol

Linear Check Soundness

Lemma B.2 (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,*

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

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).*

$\mathsf{LinearCheck}(\mathsf{pp}, A \in \mathcal{M}_{M,N}, b \in \mathbb{F}^N, [\pi]; \mathsf{U})$:
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 B.2.1.
4. $\mathcal{P}$ (a) computes matrix P from R and U as described in Section B.2.1, (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) $\mathsf{InnerProduct}(\mathsf{pp}, \mathbf{1}^T_{j_u} \Lambda_{h,2m-1}, \mathsf{cm}_{k_u}, v_u; \overline{P}[\cdot, k_u])$ 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) $\mathsf{InnerProduct}(\mathsf{pp}, \mathbf{1}^T_{j_u} \Lambda_{h,m}, C_u, \langle \delta, X_u\rangle; V_u)$ 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. 5. Linear Check Protocol

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 we show that an adversarial prover succeeds with negligible probability if $d(\mathsf{U}, (\mathcal{W}_1)^p) > e$. Second, we show that for $d(\mathsf{U}, (\mathcal{W}_1)^p) \leq e$, 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 A.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 5). 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 (2) 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 $d(\mathsf{U}, (\mathcal{W}_1)^p) \leq e$. From Lemma A.1, there exists (unique) $\mathsf{U}^* \in (\mathcal{W})^p$ such that $\Delta_1(\mathsf{U}, \mathsf{U}^*) \leq e$. 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 $\Delta_1(\hat{P}, P^*) \leq e$. 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 $\Delta_1(P, P^*) \geq n - s - \ell$ 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 $d(\mathsf{U}, (\mathcal{W}_1)^p) \leq e$ succeeds is bounded by:

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

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 $\mathsf{LinearCheck}$, 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.2.5 Quadratic Check Protocol

Quadratic Check Soundness

Lemma B.3 (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,*

$\mathsf{QuadraticCheck}(\mathsf{pp}, [\pi_x], [\pi_y], [\pi_z]; \mathsf{U}_x, \mathsf{U}_y, \mathsf{U}_z)$:
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 B.2.2, (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 $\mathsf{cm}_{k_u} = \prod_{a=1}^{2\ell-1} (c_a)^{\Lambda_{n, 2\ell-1}^T [a, k_u]}$, $v_u = \sum_{i=1}^{p} r_i [X_u[i] \cdot Y_u[i] - Z_u[i]]$ (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 (3).
16. $\mathcal{V}$ accepts if all the checks succeed.

Fig. 6. Quadratic Check Protocol

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

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 A.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 6). Define the vector $\tilde{\boldsymbol{C}} = (\tilde{C}_1, \ldots, \tilde{C}_n)$ where $\tilde{C}_k = \prod_{a=1}^{\ell} (\tilde{c}_a)^{\Lambda_{n, \ell}^T [a, k]}$ 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 (2) 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 A.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.2.6 Graphene Protocol

For a cheating prover either $d(\mathsf{U}, (\mathcal{W}_1)^{4p}) > e$, or $d(\mathsf{U}, (\mathcal{W}_1)^{4p}) \leq e$ 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 $d(\mathsf{U}, (\mathcal{W}_1)^{4p}) \leq e$, then there exists unique codeword U^* such that $\Delta_1(\mathsf{U}, \mathsf{U}^*) \leq e$ 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.

Lemma B.4 (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.,*

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

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. $\mathcal{V} \rightarrow \mathcal{P}$: $\gamma_x, \gamma_y, \gamma_z \leftarrow\$ \mathbb{F}$, $r_{lc} \leftarrow\$ \mathbb{F}^M$, $r_{qd} \leftarrow\$ \mathbb{F}^p$, $\rho \leftarrow\$ \mathbb{F}^{4p}$.
2. $\mathcal{P} \rightarrow \mathcal{V}$: $\mathcal{P}$ 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. $\mathcal{P} \rightarrow \mathcal{V}$: 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. $\mathcal{V} \rightarrow \mathcal{P}$: $Q = \{(j_u, k_u) : u \in [t]\}$.
7. $\mathcal{V} \rightarrow \pi$: $\{k_u : u \in [t]\}$.
8. $\mathcal{P} \rightarrow \mathcal{V}$: $S_u = \mathsf{U}[\cdot, j_u, k_u]$ for $u \in [t]$.
9. $\pi \rightarrow \mathcal{V}$: $\pi[\cdot, k_u]$, $u \in [t]$.
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. 7. GrapheneR1CS Protocol

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.*

To prove of Lemma B.4, we can construct an extractor $\mathcal{E}$ using the extractors $\mathcal{E}_{lc}$ and $\mathcal{E}_{qd}$ of LinearCheck and QuadraticCheck described in B.2 and B.3 respectively. $\mathcal{E}$ plays the role of the verifier and $\mathcal{E}_{lc}$, $\mathcal{E}_{qd}$ use the queries generated by $\mathcal{E}$.

B.3 Zero-knowledge

Lemma B.5 (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 $t \leq b$.*

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: $\{\gamma_x, \gamma_y, \gamma_z, r_{lc}, r_{qd}, \rho, Q = \{(j_u, k_u)\}_{u\in[t]}, \delta_{lc}, \delta_{qd}, \beta_{lc}, \beta_x, \beta_y, \beta_z, \beta_{qd}, \tau_{qd}, \gamma_{qd}\}$.
- *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 $u \in [t]$:

$$\tilde{\mathsf{U}}[\cdot, k_u] = \sum_{i\in[p]} \rho_i \mathsf{U}[i,\cdot,k_u] + \rho_{p+i}\mathsf{U}_x[i,\cdot,k_u] + \rho_{2p+i}\mathsf{U}_y[i,\cdot,k_u] + \rho_{3p+i}\mathsf{U}_z[i,\cdot,k_u]$$

$$\tilde{O}[k_u] = \sum_{i\in[p]} \rho_i O[i,k_u] + \rho_{p+i}O_x[i,k_u] + \rho_{2p+i}O_y[i,k_u] + \rho_{3p+i}O_z[i,k_u]$$

$$\tilde{c}_{k_u} \leftarrow \mathsf{Com}(\tilde{\mathsf{U}}'[\cdot,k_u], \tilde{O}[k_u])$$

$$\pi[i,k_u] \leftarrow \mathsf{Com}(\mathsf{U}'[i,\cdot,k_u], O[k_u])$$

$$\pi_a[i,k_u] \leftarrow \mathsf{Com}(\mathsf{U}'_a[i,\cdot,k_u], O[k_u])\ \forall a \in \{x,y,z\}$$

$\mathcal{S}$ 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:

$$c_{k_u} = \prod_{a\in[s+\ell-1]} (c_a)^{\Lambda^T_{n,s+\ell-1}[a,k_u]}\ \forall u \in [t]$$

$$\mathsf{cm}_{lc} = c_0^{\beta_{lc}} \cdot \prod_{a\in[s+\ell-1]} (c_a)^{\varphi_{lc_a}},$$

$$d_{k_u} = \prod_{a\in[2\ell-1]} (d_a)^{\Lambda^T_{n,2\ell-1}[a,k_u]}\ \forall u \in [t]$$

$$\mathsf{cm}_{qd} = d_0^{\beta_{qd}} \cdot \prod_{a\in[2\ell-1]} (c_a)^{\varphi_{qd_a}}$$

$\mathcal{S}$ 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 D-Graphene and Security Proofs

C.1 D-Graphene: Distributed Prover Variant

We now describe the distributed protocol to produce a Graphene proof for a statement, when the witness is shared between D provers $\mathcal{P}_1, \ldots, \mathcal{P}_\mathsf{D}$. For $\xi \in [\mathsf{D}]$, 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, $\sum_{\xi\in[\mathsf{D}]} \langle \mathsf{w} \rangle^\xi = \mathsf{w}$. Recall from Section 4, that there is an algorithm A which aggregates the messages received from provers $\mathcal{P}_1, \ldots, \mathcal{P}_\mathsf{D}$ 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).

C.2 Distributed Oracle Setup

In the 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 $\mathsf{Combine}$ simply multiplies the corresponding commitments. The homomorphism and the fact that the witnesses are additively shared ensure that π contains commitment to the witness.

C.3 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 matrix P of order $h \times n$.
- The vectors $X_u = \mathsf{U}[\cdot, j_u, k_u]$ for $u \in [t]$, for verifier's query $Q = \{(j_u, k_u) : u \in [t]\}$.

Given the verifier's challenges, we see that each of the messages is a 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 knowl-

edge 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 8.

C.4 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 $\langle \mathsf{U}_x[i,j,k].\mathsf{U}_y[i,j,k]\rangle^{\xi}$, where $\mathcal{F}_{\mathsf{Mult}}$ is the functionality that takes linear sharings two vectors as input and outputs a linear sharing of the component-wise multiplication of these vectors. 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 similar to the the distributed linear check protocol. The complete protocol for distributed quadratic check appears in Figure 9, additionally, we discuss how to optimize the MPC overhead when the size of the shared circuit (see Appendix E) is small.

C.5 Security Proofs

As discussed in the definition of DPZK given in Section 4,depending on the corruption scenario there are 4 possible cases:

- 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 state that it has the *Soundness with Witness Extraction (SoWE)* property.
- The verifier is corrupt and tries to learn about the provers' secrets. A protocol secure under this corruption scenario has the *Zero-Knowledge (ZK)* property.
- Among all the provers, t are corrupt and try to learn about the honest provers' secrets. A DPZK protocol is said to have *Witness Confidentiality (WC)* property if it is secure against this corruption scenario.
- A corrupt verifier colludes with t corrupt provers and tries to learn honest provers' secrets. A DPZK protocol is said to have *Witness Confidentiality with Collusion (WCwC)* property if it is secure against this corruption scenario.

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 on behalf of the verifier. If the verifier accepts the proof, $\mathcal{E}$ outputs a valid witness corresponding to the statement, with a very high probability.

Lemma C.1. *For the protocol D-Graphene given in Figure 10, 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. From Lemma B.4, we know that Graphene has proof of knowledge property. Therefore by Lemma 5.2, we know that D-Graphene has the soundness with witness extraction property. This proves the above lemma. □

Zero Knowledge: If a verifier in a DPZK protocol learns nothing more that the assertion of the statement, then it has the zero-knowledge 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. Note that τ consists of the verifier's challenges and the provers' messages.

Lemma C.2. *There exists a simulator $\mathcal{S}_{DP}$ that outputs a perfectly indistinguishable extended view of the verifier in an honest execution of the protocol D-Graphene for $t \leq b$.*

Proof. From the Lemma B.5, we know that Graphene has the zero-knowledge property. Coupled with

Lemma 5.2, this ensures the zero-knowledge property of D-Graphene. Thus there exists a polynomial-time simulator that generates a transcript which is indistinguishable from the real transcript. □

Witness Confidentiality and Witness Confidentiality with Collusion: Our work is mainly concerned with building a public coin proof system, specifically NIZK so that anyone can verify the proof, and any prover can play a verifier's role yet learn nothing. Therefore witness confidentiality and witness confidentiality with collusion are the same in this setting.

Though the proof of the witness confidentiality property is similar to the witness confidentiality proof of Lemma 5.2, we provide detailed proof for our protocol here.

We will separate the provers into two classes: honest(H) and corrupt(C). Note that $|H|+|C| = \mathsf{D}$ and $|C| < t$, for some $t < \mathsf{D}$. 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.3. *For the Distributed Linear Check protocol (Figure 8), there exists a polynomial-time simulator $\mathcal{S}$ corresponding to a subset of t provers out of D ($t < \mathsf{D}$) 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".$\mathcal{S}$ picks all the components of the view of corrupt parties uniformly at random from the appropriate domains such that it 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: $\{\rho, r, Q, \delta, \beta\}$, where $Q = \{(j_u, k_u) : u \in [t]\}$

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 a 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 8 has witness confidentiality property. □

The distributed quadratic check protocol described in Figure 9 has witness confidentiality property if the multiplication of the secrets held by the provers is performed

securely. Suppose Π_{Mult} be a protocol that securely realizes $\mathcal{F}_{\mathsf{Mult}}$ functionality, which can withstand t corrupt parties. Distributed quadratic check has witness confidentiality property against t corrupt provers. The following lemma ensures the claim.

Lemma C.4. *Let Π_{Mult} be a D-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 $\langle (\sum_{\xi \in [\mathsf{D}]} \langle \mathbf{a} \rangle^{\xi}) \cdot (\sum_{\xi \in [\mathsf{D}]} \langle \mathbf{b} \rangle^{\xi}) \rangle^{\xi}$ 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 a 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 outputs a view which is indistinguishable from the view of $\mathcal{A}$ in the Distributed Quadratic Check protocol (Figure 9).*

Proof. Since Π_{Mult} is a secure protocol, there exists 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.3, we consider C as the set of corrupt parties and H as the set of honest parties. We 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 $\langle \tilde{c_{k_u}} \rangle^H$, 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 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 $\langle P \rangle^H[\cdot, k_u]$.
- $\mathcal{S}$ 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.
- $\mathcal{S}$ 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 D-Graphene is the combination of linear check and quadratic check, simulator for the complete protocol follows the same strategies as the simulators described in Lemma C.3 and C.4. Hence, the D-Graphene (Figure 10) has the witness confidentiality property.

Lemma C.5. *Let Π_{Mult} be a secure protocol described in Lemma C.4 which is used in D-Graphene (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 the simulators for the linear check (Lemma C.3) and the quadratic check (Lemma C.4). 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 [10], [13], 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 [13], 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 $\langle \boldsymbol{a}_L \rangle^\xi, \langle \boldsymbol{a}_R \rangle^\xi, \langle \boldsymbol{a}_O \rangle^\xi \; \forall \xi \in [\mathsf{D}]$ such that $\sum_{\xi \in [\mathsf{D}]} \langle \boldsymbol{a}_A \rangle^\xi = \boldsymbol{a}_A \; \forall A \in \{L, R, O\}$. 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 $l(X) = \sum_{\xi \in [\mathsf{D}]} \langle l \rangle^\xi(X)$ and $r(X) = \sum_{\xi \in [\mathsf{D}]} \langle r \rangle^\xi(X)$, i.e., output for $\mathcal{P}_\xi$ is $\langle t \rangle^\xi(X)$ such that $\sum_{\xi \in [\mathsf{D}]} \langle t \rangle^\xi(X) = t(X)$. $\mathcal{P}_\xi$ 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 $\boldsymbol{l} = \sum_{\xi \in [\mathsf{D}]} \langle \boldsymbol{l} \rangle^\xi$ and $\boldsymbol{r} = \sum_{\xi \in [\mathsf{D}]} \langle \boldsymbol{r} \rangle^\xi$. 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 [42]. 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[42]. 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[42]. 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 [42].

In the DPZK version of Spartan [42] 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 [42] 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. Note that, a similar approach can be used to restrict the MPC only to the shared columns if the shared circuit is sufficiently small. This gives an MPC of depth 1 on circuit of size $O(\max(N_s, m))$. Towards improving the efficiency in the distributed setting, usage of secure multiplication can be reduced. Suppose for a gate g among all the provers, $\mathcal{P}_1, \ldots, \mathcal{P}_{\mathsf{D}}$, only P_1 and P_2 has input to g. Thus the shares of the remaining provers corresponding to the gate g is 0. Hence running an MPC between P_1 and P_2 maybe sufficient. Then resharing the outcome among all the provers is required. However, this approach requires end-to-end formal analysis. Moreover, this is circuit specific and maybe hard to implement.

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

G Performance Evaluation

In Section B.2.3, 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 $\mathsf{Graphene}$ 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*: $t_{\mathcal{P}} = 4p((m + n)\mathsf{C}_{\mathsf{FFT}}(m) + m\mathsf{C}_{\mathsf{FFT}}(l) + n\mathsf{C}_{\mathsf{FFT}}(h) + 4p\ell\mathsf{C}_{\mathsf{MXP}}(m) + (n - \ell)\mathsf{C}_{\mathsf{MXP}}(\min(\ell, m)) + (s + 3\ell)\mathsf{C}_{\mathsf{MXP}}(2m) + (48tm + 32m)\mathsf{C}_{\mathsf{EXP}}$

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

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 B.2.1.
7. $\mathcal{P}_\xi$ computes: Matrix $\langle P \rangle^\xi$ from R and $\langle \mathsf{U} \rangle^\xi$ as described in Section B.2.1. 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. $\mathsf{A} \to \mathcal{V}$: $X_u = \sum_{\xi \in [\mathsf{D}]} \langle X_u \rangle^\xi, P_u = \sum_{\xi \in [\mathsf{D}]} \langle P_u \rangle^\xi$ 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 $z = \sum_{\xi \in [\mathsf{D}]} \langle z \rangle^\xi$, $V_u = \sum_{\xi \in [\mathsf{D}]} \langle V_u \rangle^\xi$.
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. 8. Distributed Linear Check Protocol

- 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.

In Figure 2, we compare Graphene with Ligero [1] and Bulletproofs [13] 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 D-Graphene. D-Grapheneachieves the following efficiency, where the notations are the same as above and $\mathsf{MPC}(x, y, z, w)$ denotes the cost of running a secure evaluation among z parties with w corruption for x many multiplication gates with depth y.

- *Number of rounds*: $r_{\mathsf{zk}} = O(\log N)$.
- *Argument size*: $c_{\mathsf{zk}} = 4pt\mathsf{B}_\mathbb{F} + (4pt + 8t\log m + 4\ell + s + 8t + 4)\mathsf{B}_\mathbb{G}$.
- *Provers' complexity*: $t_\mathcal{P} = \mathsf{D}(4p((m+n)\mathsf{C}_{\mathsf{FFT}}(m) + m\mathsf{C}_{\mathsf{FFT}}(l) + n\mathsf{C}_{\mathsf{FFT}}(h) + 4p\ell\mathsf{C}_{\mathsf{MXP}}(m) + (n -$

$\mathsf{DistQuadraticCheck}(\mathsf{pp}, [\pi_x], [\pi_y], [\pi_z]; \langle \mathsf{U}_x\rangle^\xi, \langle \mathsf{U}_y\rangle^\xi, \langle \mathsf{U}_z\rangle^\xi \langle 0^{2m}\rangle^\xi)$:
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 $\xi \in [\mathsf{D}]$ and $\sum_{\xi\in[\mathsf{D}]} \langle \mathsf{w}_a\rangle^\xi = \mathsf{w}_a \ \forall a \in \{x, y, z\}$.

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 $\langle \tilde{c}\rangle^\xi_k = \prod_{i=1}^{p} (\langle \mathcal{O}_x\rangle^\xi[i,k])^{\rho_i} \cdot (\langle \mathcal{O}_y\rangle^\xi[i,k])^{\rho_{p+i}} \cdot (\langle \mathcal{O}_z\rangle^\xi[i,k])^{\rho_{2p+i}} \ \forall k \in [\ell]$.
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 B.2.2. 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. $\mathsf{A} \to \mathcal{V}$: $A_u = \sum_{\xi\in[\mathsf{D}]} \langle A_u\rangle^\xi$ 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: $\langle V_u\rangle^\xi = \sum_{i=1}^{p} \delta_i \big(\beta_x \langle \mathsf{U}_x\rangle^\xi[i,\cdot,k_u] + \beta_y \langle \mathsf{U}_y\rangle^\xi[i,\cdot,k_u] + \beta_z \langle \mathsf{U}\rangle^\xi[i,\cdot,k_u]\big)$ 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 $z = \sum_{\xi\in[\mathsf{D}]} \langle z\rangle^\xi$ and $V_u = \sum_{\xi\in[\mathsf{D}]} \langle V_u\rangle^\xi$.
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) $\mathsf{InnerProduct}(\mathsf{pp}, \mathbf{1}^T_{j_u} \Lambda_{h,2m-1}, \mathsf{cm}_{k_u}, v_u; \overline{P}[\cdot, k_u])$ 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) $\mathsf{InnerProduct}(\mathsf{pp}, \mathbf{1}^T_{j_u} \Lambda_{h,m}, T_u, w_u; \overline{V}_u)$, 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 (3).

Fig. 9. Distributed Quadratic Check Protocol

$\ell)\mathsf{C}_{\mathsf{MXP}}(\min(\ell, m)) + (s + 3\ell)\mathsf{C}_{\mathsf{MXP}}(2m) + (48tm + 32m)\mathsf{C}_{\mathsf{EXP}}) + 8\mathsf{MPC}(N, 1, \mathsf{D}, \mathsf{D} - 1)$

- 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 $\kappa_{\mathrm{gr}} = (1 - e/n)^t + 2\big(2m/h + (1 - 2m/h)(2\ell + e)/n\big)^t$.

We now illustrate D-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 [5, 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

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

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

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 [32] 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.

$DP - \mathsf{GrapheneR1CS}(\mathsf{pp}, A, B, C, [\pi]; \mathsf{w}_x, \mathsf{w}_y, \mathsf{w}_z, \mathsf{w})$:
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 $\tilde{c}_k = \prod_{\xi \in [\mathsf{D}]} \langle \tilde{c}_k \rangle^\xi$ 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 $c_k = \prod_{\xi \in [\mathsf{D}]} \langle c_k \rangle^\xi \ \forall k \in \{0, \ldots, s + \ell - 1\}$ and $d_k = \prod_{\xi \in [\mathsf{D}]} \langle d_k \rangle^\xi \ \forall k \in \{0, \ldots, 2\ell - 1\}$ 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 $S_u = \sum_{\xi \in [\mathsf{D}]} \langle S_u \rangle^\xi$, $Plc_u = \sum_{\xi \in [\mathsf{D}]} \langle Plc_u \rangle^\xi$ and $Pqd_u = \sum_{\xi \in [\mathsf{D}]} \langle Pqd_u \rangle^\xi$ 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 $Vlc_u = \sum_{\xi \in [\mathsf{D}]} Vlc_u$ and $Vqd_u = \sum_{\xi \in [\mathsf{D}]} Vqd_u$.
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. 10. Distributed GrapheneR1CS Protocol