Private Polynomial Commitments and Applications to MPC

Rishabh Bhadauria[1], Carmit Hazay[1,3], Muthuramakrishnan Venkitasubramaniam[2,3], Wenxuan Wu[4], and Yupeng Zhang[4]

[1] Bar Ilan University, Israel
[2] Georgetown University, USA
[3] Ligero Inc
[4] Texas A&M University

Abstract. Polynomial commitment schemes allow a prover to commit to a polynomial and later reveal the evaluation of the polynomial on an arbitrary point along with proof of validity. This object is central in the design of many cryptographic schemes such as zero-knowledge proofs and verifiable secret sharing. In the standard definition, the polynomial is known to the prover whereas the evaluation points are not private. In this paper, we put forward the notion of *private polynomial commitments* that capture additional privacy guarantees, where the evaluation points are hidden from the verifier while the polynomial is hidden from both.

We provide concretely efficient constructions that allow simultaneously batch the verification of many evaluations with a small additive overhead. As an application, we design a new concretely efficient multi-party private set-intersection with malicious security and improved asymptotic communication and space complexities.

We demonstrate the concrete efficiency of our construction via an implementation. Our scheme can prove 2^{10} evaluations of a private polynomial of degree 2^{10} in 157s. The proof size is only 169KB and the verification time is 11.8s. Moreover, we also implemented the multi-party private set intersection protocol and scale it to 1000 parties (which has not been shown before). The total running time for 2^{14} elements per party is 2,410 seconds. While existing protocols offer better computational complexity, our scheme offers significantly smaller communication and better scalability (in the number of parties) owing to better memory usage.

1 Introduction

A polynomial commitment is a cryptographic building block that allows a prover to commit to a polynomial, which can later be opened at any evaluation point with proof that the evaluation is correctly computed. Polynomial commitments, which serve as an important building block in constructing cryptographic protocols, were introduced by Kate et al. [49] for the construction of verifiable secret sharing in the synchronous and asynchronous setting [6]. The scheme was generalized to multivariate polynomials by Papamanthou et al. [54], and to zero-knowledge proofs of knowledge by Zhang et al. [72]. In recent years, they are extensively used to build efficient zero-knowledge proof systems [71, 65, 68, 62, 34, 24], where recent new schemes without a trusted setup were proposed in [65, 70, 16, 64, 50]. Subsequent works considered batched openings for multiple evaluations [63] and multiple polynomials [40]. Another application where polynomial commitments are utilized is “Proof of retrievability” [48, 69]. In this problem, the server wishes to prove to a verifier that all of the client’s data is stored correctly. The polynomial commitment allow the prover to prove the integrity of the data storage. Logarithmic and constant size polynomial commitments are also used in constructing vector commitments [19, 18, 23].

To date, all concretely efficient polynomial commitments require the verifier to know the evaluation point and the prover to know the polynomial. While such a notion is sufficient to design succinct zero-knowledge arguments, secure multi-party computation (MPC) requires additional privacy guarantees. In this paper, we consider a different setting where the polynomial is unknown to the prover and is encrypted. Moreover, the evaluation points are committed by the prover and may not be publicly known to the verifier. This setting is very common in MPC where both the polynomial and the evaluation points must remain private as they are defined based on the parties’ inputs. We denote this primitive by *private polynomial commitment* and show that it can be used as a building block in many applications that arise in the secure multi-party setting; see Sections 1.1 and 5. Our scheme is particularly useful in batch scenarios when there are multiple evaluation points. In this case, the proof size and verifier’s complexity grow additively with the number of points.

1.1 Our Contributions

Our contribution is threefold. (1) abstracting the new notion of private polynomial commitments and providing two constructions. (2) demonstrating its applicability for MPC and (3) implementing our commitment schemes and presenting a new multi-party private set-intersection (MPSI) protocol.

Private Polynomial Commitments Our contribution includes two flavours of private polynomial commitments with a hidden (encrypted) polynomial; one where the evaluation points are public and the other where they are private. Our schemes are built on the recent scheme of an inner product argument [17], which generalizes the inner product argument from [15] to bilinear groups. Specifically, we embed the ciphertexts encrypting the coefficients in the base group using an Additively Homomorphic Encryption (AHE) scheme introduced in [13]. Working with bilinear maps allow to publicly verify a single multiplication in the exponent which allows any party to verify the proof. More specifically, for a polynomial of degree d, the overhead is dominated by $O(d)$ bilinear pairings whereas the proof size is $O(\log d)$ and the verifier time is $O(d)$ exponentiations. Our construction supports batched evaluations efficiently. To open at m evaluation points, the proof size is $O(m + \log d)$ and the verifier time is only $O(m + d)$. The polynomial is hidden from all parties and only an encrypted form is available to the prover.

Our constructions rely on two different commitment schemes for committing to the encrypted polynomial (using the pairing-based scheme from [4]) and the evaluation points (using Pedersen commitment [55]). We further rely on the Boneh et al. pairing-based encryption scheme [13] to be compatible with our pairing-based commitment scheme, both of which rely on the Decisional Linear Assumption (DLIN) and Double Pairing Problem (DPP).

Our commitment scheme uses an inner product argument [15] as a building block (denoted by BBB-IPA) and is the first polynomial commitment scheme where the prover does not know the actual polynomial and only has access to its encryption. The main challenge in constructing this commitment scheme was the integration of encrypted polynomials into the polynomial commitment scheme. Secondly, directly constructing a scheme would not provide batching. To ensure batching and overall small proof size, we reduce the proving of the polynomial evaluation to multiple inner products. First, we provide a new inner product argument that allows the prover to verify inner products on encrypted ciphertext with the evaluation vector. Second, we prove the correct structure of multiple evaluation vectors by verifying the linear and quadratic constraints. Both our linear and quadratic tests reduce the multiple constraints on all different evaluation vectors to verify a single inner product argument, thereby ensuring the batching feature is effective. An additional feature is that the proof can be made non-interactive using Fiat-Shamir.

Applications Private polynomial commitment schemes are useful for private computations based on polynomials. We list four such applications that can benefit from the scalability and batching of the evaluations as inherent in our commitment scheme. Firstly, we use our new private polynomial commitment as a building block to present a new scalable multi-party PSI protocol that is secure against malicious adversaries. We also discuss three other applications - Oblivious Polynomial Evaluation, Verifiable Polynomial Evaluation, and Non-Interactive two-party PSI; for more details see Section 5.

Scalable multi-party private set-intersection (MPSI). PSI is a fundamental problem in secure computation that has been widely studied in the past decade. In this problem a set of parties $P_1, \ldots, P_n$, holding input sets $X_1, \ldots, X_n$ of sizes $m_1, \ldots, m_n$, respectively, wish to compute $X_1 \cap X_2 \cap \ldots \cap X_n$. The two-party setting has been studied extensively and continues to be a hot topic of research owing to numerous applications such as contact discovery, dating services, data mining, recommendation systems, and law enforcement. In a long line of works, highly efficient two-party protocols have been designed with almost linear overhead in the set sizes (see some recent works at [58, 56, 57, 22] and references therein). Furthermore, Google has recently leveraged this technology to match login credentials against an encrypted database.

While considerable progress has been made in the two-party setting, very few works have explored the concrete efficiency of PSI in the multi-party setting and the existing works have mostly considered only the semi-honest setting. Furthermore, current approaches fail to achieve overheads as in the two-party setting and do not scale well due to communication and space bottlenecks. Multiparty PSI is a fundamental cryptographic primitive with a richer set of applications beyond the two-party ones such as distributed intrusion detection, identifying the most visited sites or watched movies, contact tracing and more.

Our starting point is the work of Freedman et al. [33] who designed a simple two-party PSI protocol based on polynomials. Roughly speaking, P_1 creates a polynomial $Q(\cdot)$ whose roots correspond to its input data set and sends this polynomial to P_2, encrypted under an additively homomorphic encryption scheme. P_2 homomorphically evaluates a "masked" variant of the encrypted polynomial on its data set. In more detail, for each element x in P_2's input set, P_2 generates fresh randomness r and sends an encryption of $r \cdot Q(x) + x$ to P_1. P_1 decrypts and identifies the elements in the set intersection. Namely, if the decrypted value x is in P_1's set, then x is extracted from the decryption of the ciphertext. Whereas if the item x is not in the intersection, with very low probability, there exists an element z for which $r \cdot Q(z) + z$ is a false positive.

More recently, Hazay and Venkitasubramaniam [46] extended [33] to the multi-party setting by reducing the multi-party PSI (MPSI) task among n parties to n instances of two-party PSI. In this work we explore the practicality of [46] in the malicious setting where up to $n-1$ parties can be corrupted. On a high level, in [46], parties $P_2, \ldots, P_n$ create a polynomial whose roots correspond to their respective inputs and send their encrypted coefficients to P_1. P_1 then aggregates the polynomials and homomorphically evaluates the resulting encrypted polynomial on its input set. To make the protocol secure against malicious adversaries, [46] introduced a simple mechanism for P_1 to prove and the parties to verify that P_1 aggregated the polynomials correctly, and relied on zero-knowledge proofs for the remaining steps.

The protocol presented in [46] implies an overall communication complexity of $O(n^2 + n \cdot m_{\max} + n \cdot m_{\min} \cdot \log m_{\max})$ where $m_{\max}$ (resp. $m_{\min}$) is the size of the largest (resp. smallest) input set. The threshold key generation incurs a communication cost of $O(n^2)$. The central party aggregates the input polynomials of all the parties and returns the encrypted coefficients of the aggregated polynomial. This yields a communication overhead of $O(n \cdot m_{\max})$. The main source of overhead is due to the zero-knowledge proof applied by the central party for proving correct evaluation, which implies an overhead of $O(n \cdot m_{\min} \cdot \log m_{\max})$. This phase is captured in our protocol by private polynomial commitments.

More precisely, in this work, we introduce a variant of [46] where we rely on a new abstraction that is based on private polynomial commitments. By leveraging the efficiency and batching features of our commitment schemes, we manage to improve the communication and computation complexities of [46]. We further provide an implementation of our PSI protocol and explore its concrete efficiency. This is in contrast to [46] which had the potential of being concretely efficient but did not provide an implementation.

The complexity of our protocol. In addition to our new abstraction, we further improve the asymptotic complexity of [46] to $O(n^2 + \sum_{i=1}^{n} m_i + n \cdot (m_{\min} +$

$\log m_{\max}$)). Introducing private polynomial commitments (PPC) as a building block, the central party in our protocol does not send the encrypted aggregated polynomial. Instead, a commitment of encrypted aggregated polynomials is sent to the parties. This allows us to remove the $O(n \cdot m_{\max})$ factor. To further reduce the communication complexity, we leverage the batching feature of PPC which allows the central party to prove the correctness of multiple evaluations on the aggregated polynomial. The proof size, in this case, is $O(m_{\min} + \log m_{\max})$ which contributes an additive factor of $O(n \cdot (m_{\min} + \log m_{\max}))$ to the communication complexity of our MPSI protocol. A detailed analysis is provided in Table 3 where the communication complexity is broken according to the central party overhead and the other parties and is presented for each phase separately.

Comparison with recent work. Three recent works that design PSI protocols with malicious security are [9, 36, 41]. Similarly to our work, these works also achieve linear communication complexity in the number of parties by relying on a star topology. The main advantage of these protocols is that they rely on oblivious transfer (OT), oblivious linear evaluation (OLE) (used in [41]) and symmetric-key primitives for which we have very efficient instantiations. In comparison to previous work [9, 36, 41], our protocol achieves the best communication and space complexities. Specifically, our communication complexity is dominated by the term $O(n^2\kappa + nm\kappa)$ where the gain compared to previous work is due to an aggregation of the encrypted input polynomials and the small batched proof size. We compare the communication complexity in Table 1. In the typical parameter regime, the computational security parameter κ is greater than the statistical parameter λ satisfying the inequality $\lambda + \log m < \kappa$ where m is the input set size. Applying this inequality to the asymptotic communication complexity of [36] yields communication complexity that matches ours.

Most MPSI protocols (including ours) are designed for a star topology, where a central party aggregates the other parties' messages and therefore requires larger space. In prior works, the space complexity of the central party is inflated with a factor that depends both on the input and the number of parties, whereas our space complexity only grows with $O(m\kappa)$. The space complexity of the other, "non-central" parties, is independent of the number of parties. We compare the space complexity in Table 2

Our paper realizes a standard MPSI functionality where a single party (typically the central party) receives the output, but can be extended to guarantee security even when all parties receive the output. Both [41] and our protocol achieve this standard security whereas the works of [9, 36] provide a weaker security guarantee that allows the party that first receives the output (if controlled by the adversary) to unnoticeably remove certain elements from the output when broadcasting it to all parties. Note that these protocols can achieve full security, but this will require applying general-purpose zero-knowledge proofs.

On the other hand, the computational cost of [9, 36, 41] grows with $\Omega(mn\kappa)$ field multiplications, while the dominating cost of our computation is $O(m^2)$ exponentiations. This can be further reduced into $O(m \frac{\log m}{\log \log m})$ using hashing. While for a small number of parties, our protocol is slower, the total running

	P_1	P_i	Total
[9]	$O(nm\kappa^2 + nm\kappa \log m\kappa)$	$O(m\kappa^2 + m\kappa \log m\kappa)$	$O(nm\kappa^2 + nm\kappa \log m\kappa)$
[36]	$O(n\kappa + nm(\kappa + \lambda + \log m))$	$O(n\kappa + m(\kappa + \lambda + \log m))$	$O(n^2\kappa + nm(\kappa + \lambda + \log m))$
[41]	$O(nm\kappa + n\lambda\kappa \log m)$	$O((n + m)\kappa + \lambda\kappa \log m)$	$O(n^2\kappa + nm\kappa + n\lambda\kappa \log m)$
Theorem 2	$O(nm\kappa)$	$O((n + m)\kappa)$	$O(n^2\kappa + nm\kappa)$

Table 1: The communication complexity analysis of MPSI *in bits* where κ is the computational security parameter, λ is the statistical security parameter, n is the number of parties, m is an upper bound on the inputs set sizes and P_1 is the central party.

time essentially remains the same when the number of parties increases. For instance, our experiments show that our scheme takes 9,141 seconds for 1000 parties and 2^{16} elements per party. Prior works cannot run at this scale.

We highlight some applications which require PSI for a large number of parties and large input sizes: (1) Cache-sharing [53] involves multiple network providers who wish to cache common elements with high access frequency in a shared cache and require privacy of their local cache. (2) Another application is to generate statistics over the Tor network. Prior literature e.g., [27, 66] has relied on MPC, secure aggregation and differential privacy to generate statistics on Tor servers in a privacy-preserving manner. Large-scale MPSI can be useful here where common features need to be extracted among the relay servers without compromising the users' privacy. (3) Hospitals and healthcare providers can collaborate to analyze common features between databases which include a large number of medical records. (4) Finally, MPSI can be applied for contact tracing. A large group of patients can execute an MPSI protocol to find common locations they have been to without leaking each individual's travel history. The result can help the actions of testing or quarantine in these areas.

	P_1	P_i
[9]	$O(nm\kappa^2 + m\kappa \log m\kappa)$	$O(m\kappa^2)$
[36]	$O(nm\kappa + m(\kappa + \lambda + \log m))$	$O(m(\kappa + \lambda + \log m))$
[41]	$O(nm\kappa)$	$O(m\kappa)$
Theorem 2	$O(m\kappa)$	$O(m\kappa)$

Table 2: The space complexity analysis of MPSI *in bits* where κ is the computational security parameter, n is the number of parties, m is an upper bound on the inputs set sizes and P_1 is the central party.

Private polynomial commitments are also useful for reusable non-interactive two-party PSI. Non-interactive secure computation introduced in [47], considers a "receiver" that publicly broadcasts a single message and any "sender" can interact in a two-party secure computation protocol with the receiver by sending a single message to the receiver. The receiver only needs to broadcast once and any number of interactions with the receiver can be performed. Specializing the setting to PSI, our protocol enables non-interactive PSI which can be applied to dating services, ride-share matching, and contact tracing. While such a protocol may introduce high computational cost compared to existing works e.g., [60], its communication cost is competitive as it benefits from our batching feature, which is extremely useful in a client-server setting; see more details in Section 5.3.

Oblivious polynomial evaluation. The oblivious polynomial evaluation (OPE) functionality is an important functionality in the field of secure two-party computation. It considers a setting where party P_2 holds a d-degree polynomial $Q(\cdot)$ and party P_1 holds an element t, and the goal is that P_1 obtains $Q(t)$ and nothing else while P_2 learns nothing. OPE has proven to be a useful building block and can be used to solve numerous cryptographic problems; e.g., secure equality of strings, set-intersection, approximation of a Taylor series, RSA key generation, oblivious keyword search, set membership, blacklisting anonymous users, data entanglement and more [33, 32, 52, 8, 43, 38].

In this work, we consider a distributed variant of OPE, where the input polynomial is additively secret-shared amongst the parties, and the goal of the parties is to evaluate (in the exponent) the aggregated polynomial privately and correctly. The scenario where the polynomial is distributed naturally arises in settings where the data cannot be stored on a single memory device due to privacy considerations. Secret-sharing sensitive data protects it against leakage attacks and eliminates the risk of breaching the stored memory. In some cases, the data is distributed to avoid a single point of failure and to ensure continuous access to the data.

Private polynomial commitments are useful in this context and enable secure evaluation of the combined polynomial in the presence of $n-1$ malicious corruptions, similar to our PSI protocol. The incoming communication complexity of P_1 is linear in the size of shares, whereas the outgoing communication only grows logarithmically in the polynomial degree plus P_1's input size (and hence sublinear in d). The bulk of the computational overhead is attributed to P_1, which evaluates the aggregated polynomial on its input. An interesting feature of our protocol is its usage for multi-point evaluations. Here P_1 evaluates $Q(\cdot)$ on multiple points $t_1, t_2, \ldots$ where the accumulated overhead per evaluation point for ensuring malicious security vanishes away due to our batching property.

Verifiable polynomial evaluations. In this setting, computationally weak devices (or clients) wish to outsource their computation and data to an *untrusted* server in the cloud. The ultimate goal in this setting is to design efficient protocols that minimize the computational overhead of the clients and instead rely on the extended resources of the server. Of course, the amount of work invested by the client for verifying the correctness of the computation is *substantially* smaller than running the computation by itself. Another ambitious challenge of verifiable computation is to minimize the *communication* from the cloud.

The problem of delegating a single polynomial was studied by Benabbas et al. [10], who introduced a new cryptographic primitive of algebraic PRFs, which enables the generation of short authentication message to verify the server's reply.

Followup works [29, 7, 20, 21] improved different aspects of [10]. Nevertheless, all prior constructions considered a setting where a single client communicates with the server. Extending these solutions to the multi-client setting is not immediate (even in the non-private setting) since the server needs to aggregate the shares of the polynomials and provide proof for validating the aggregation, which is highly non-trivial. We observe that polynomial commitment schemes directly imply a verifiable evaluation of distributed polynomials where correctness is established via the proof provided by the server.

When considering verifiable computation, one can consider a setting where the function is either public or private. Verifiable computation with function privacy is often harder to achieve. We note that our construction follows even if the polynomials are encrypted while the evaluation points are given in the clear. This can capture scenarios where the polynomial represents a database with secret payloads yet the queries are not private.

Implementation Details To validate the concrete efficiency of our construction, we implemented our private polynomial commitment scheme and multi-party PSI protocol. Our implementation of the private polynomial commitment scheme demonstrates the advantage of batch opening. For a polynomial of degree 2^{16}, the proof size is 18.6KB and the verifier time is 53.7s to open one evaluation, while they are only 6.1MB and 757s for 2^{16} evaluations respectively, which are significantly better than repeating the single opening 2^{16} times. Our multi-party PSI protocol with malicious security can scale to 1000 parties with 2^{16} elements per party. The majority of the time is spent on the computation of the proofs of our private polynomial commitment, which can be further accelerated through multi-threading and hashing. The communication and the memory usage of our protocol is an order of magnitude better than existing schemes, and thus our protocol performs better for a large number of parties and networks with limited bandwidth; see Section 6 for further details. We plan to open-source our implementation and the source code is available at `https://anonymous.4open.science/r/PCOM-CCF4`.

2 Preliminaries

2.1 Basic Notations

We denote the security parameter by κ. We say that a function $\mu : \mathbb{N} \to \mathbb{N}$ is negligible if for every positive polynomial $p(\cdot)$ and all sufficiently large κ it holds that $\mu(\kappa) < \frac{1}{p(\kappa)}$. We use the abbreviation PPT to denote probabilistic polynomial time. We further denote by $a \leftarrow A$ the random sampling of a from a distribution A, by $[d]$ the set of elements $(1, \ldots, d)$ and by $[0, d]$ the set of elements $(0, \ldots, d)$.

We now specify the definition of computationally indistinguishable.

Definition 1 (Computational Indistinguishability) *Let $X = \{X(a, \kappa)\}_{a\in\{0,1\}^*,\kappa\in\mathbb{N}}$ and $Y = \{Y(a, \kappa)\}_{a\in\{0,1\}^*,\kappa\in\mathbb{N}}$ be two distribution ensembles. We say that X and*

Y are computationally indistinguishable, denoted $X \approx Y$, if for every PPT machine D, every $a \in \{0,1\}^*$, every positive polynomial $p(\cdot)$ and all sufficiently large κ:

$$\big| \Pr[D(X(a,\kappa),1^\kappa)=1] - \Pr[D(Y(a,\kappa),1^\kappa)=1] \big| < \frac{1}{p(\kappa)}.$$

We denote $A[i]$ as i^{th} element in vector A. We denote $A[: d']$ as a vector consisting of only the first d' elements of vector A while $A[d' :]$ denotes all the elements in A starting from index d'. For two vectors A and B, we define the point-wise product as $A \odot B$. For a given vector A and a value x, $B = A^x$ is defined as point-wise exponentiation where $B_i = A_i^x$ for all each index $i \in [|B|]$. We denote $\langle A, B \rangle$ for the inner product between two vectors A and B.

2.2 Hardness Assumptions

Let $\mathcal{G}$ be a group generation algorithm, which outputs $(p, \mathbb{G}, \mathbb{G}_1, e, g)$ given 1^κ, where $\mathbb{G}, \mathbb{G}_1$ are the descriptions of groups of prime order p, e is a bilinear mapping (see below) and g is a generator of $\mathbb{G}$.

Definition 2 (DLIN) *We say that the decisional linear problem (DLIN) is hard relative to $\mathcal{G}$, if for any PPT distinguisher D there exists a negligible function* negl *such that*

$$(p, \mathbb{G}, \mathbb{G}_1, e, g, g^x, g^y, g^{xr}, g^{ys}, g^{r+s}) \approx_c$$
$$(p, \mathbb{G}, \mathbb{G}_1, e, g, g^x, g^y, g^{xr}, g^{ys}, g^d),$$

where $(p, \mathbb{G}, \mathbb{G}_1, e, g) \leftarrow \mathcal{G}(1^\kappa)$ and $x, y, r, s, d \leftarrow \mathbb{Z}_p$ where d is the degree of the polynomial.

Definition 3 (Bilinear pairing) *Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be multiplicative cyclic groups of prime order p and let g, v be a generator of $\mathbb{G}_1$ and $\mathbb{G}_2$ repsectively. A map $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map for $\mathbb{G}_1$ and $\mathbb{G}_2$ if it has the following properties: (1) Bi-linearity: $\forall w \in \mathbb{G}_1, g \in \mathbb{G}_2$, $\forall a, b \in \mathbb{Z}_p$, $e(w^a, g^b) = e(w, g)^{ab}$. (2) Non-degeneracy: $e(w, g)$ generates $\mathbb{G}_T$.*

We assume that the D-linear assumption holds in $\mathbb{G}_1$.

Definition 4 (DPP) *We say that the double pairing problem (DPP) is hard relative to $\mathbb{G}_1, \mathbb{G}_2$, if for any PPT adversary $\mathcal{A}$ there exists a negligible function* negl *such that*

$$\Pr\big[(w_r, w_t) \leftarrow \mathbb{G}_1; (r,t) \leftarrow \mathcal{A}(S, w_r, w_t) \mid (r,t) \in \mathbb{G}_2 \times \mathbb{G}_2 \\ \wedge\ e(w_r, r) \cdot e(w_t, t) = 1\big] \leq \mathsf{negl}$$

where $S = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, p, e, w, g) \leftarrow \mathcal{G}(1^\kappa)$

2.3 Public Key Encryption Schemes (PKE)

We specify first the definitions of public key encryption and IND-CPA.

Definition 5 (Public Key Encryption Scheme) *A Public-Key Encryption Scheme is a tuple for four algorithms* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Rerand})$:

- $(PK, SK) \leftarrow \mathsf{KeyGen}(1^\kappa)$*: inputs security parameter* κ *and outputs public key* PK *and secret key* SK*.*
- $c_a \leftarrow \mathsf{Enc}_{PK}(a; r)$*: inputs a message* a*, randomness* r *and public key* PK *and outputs a ciphertext* c_a*.*
- $a \leftarrow \mathsf{Dec}_{SK}(c_a)$ *: inputs a cipertext* c_a*,* a *and secret key* SK *and outputs a plaintext message* a*.*
- $c'_a \leftarrow \mathsf{Rerand}_{PK}(c_a; r)$ *: inputs a ciphertext* c_a*, randomness* r *and public key* PK *and outputs a ciphertext* c'_a *which encrypts the same message but by homomorphically adding randomness* r *to ciphertext* c_a*.*

For a public key encryption scheme $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Rerand})$ and a non-uniform adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, we consider the following IND-CPA game:

$$\begin{aligned}
&(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{KeyGen}(1^\kappa).\\
&(m_0, m_1, \mathit{history}) \leftarrow \mathcal{A}_1(\mathrm{PK}), \text{ s.t. } |m_0| = |m_1|.\\
&c \leftarrow \mathsf{Enc}_{\mathrm{PK}}(m_b), \text{ where } b \leftarrow \{0,1\}.\\
&b' \leftarrow \mathcal{A}_2(c, \mathit{history}).\\
&\mathcal{A} \text{ wins if } b' = b.
\end{aligned}$$

Denote by $\mathcal{A}_{\Pi,\mathcal{A}}(\kappa)$ the probability that $\mathcal{A}$ wins the IND-CPA game.

Definition 6 (IND-CPA) *A public key encryption scheme* $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Rerand})$ *has indistinguishable encryptions under chosen plaintext attacks (IND-CPA), if for every non-uniform adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *there exists a negligible function* negl *such that* $\mathcal{A}_{\Pi,\mathcal{A}}(\kappa) \leq \frac{1}{2} + \mathsf{negl}(\kappa)$*.*

Additionally, we introduce another algorithm Eval which inputs an encrypted polynomial and an evaluation point and outputs a ciphertext which represents the encrypted evaluation.

Additively Homomorphic PKE A public key encryption scheme is additively homomorphic if given two ciphertexts $c_1 = \mathsf{Enc}_{\mathrm{PK}}(m_1; r_1)$ and $c_2 = \mathsf{Enc}_{\mathrm{PK}}(m_2; r_2)$ it is possible to efficiently compute $\mathsf{Enc}_{\mathrm{PK}}(m_1 + m_2; r)$ with independent r, and without the knowledge of the secret key. Clearly, this assumes that the plaintext message space is a group; we actually assume that both the plaintext and ciphertext spaces are groups (with respective group operations $+$ or $\cdot$).

The [13] PKE In this paper, we utilize the additive variant of [13] PKE (denoted by BBS) which is similar to the additive variant of El-Gamal encryption [35]. The public key is a tuple $\mathrm{PK} = (\mathbb{G}, p, g, h, u, v)$ and the corresponding secret key $\mathrm{SK} = (x, y, z)$ s.t. $u^x = g, v^y = g, h = g^z$. To encrypt a message $m \in \mathbb{G}$ we choose $r, s \leftarrow \mathbb{Z}_p$ and let the ciphertext be $\mathsf{BBS.Enc}_{\mathrm{PK}}(m, (r, s)) = (a, b, c) = (u^r, v^s, g^{r+s} \cdot h^m)$. To decrypt a ciphertext $(a, b, c) \in \mathbb{G}^3$ we first compute $h^m = \mathsf{BBS.Dec}_{SK}((a, b, c)) = c/a^x b^y$ and then finding m by running an exhaustive search. This variant is only applicable for small plaintext domains, which is the case in our work. We also require a rerandomization algorithm Rerand which inputs a ciphertext and randomness and outputs a new ciphertext with the same plaintext but new randomness $(a', b', c') = \mathsf{BBS.Rerand}_{\mathrm{PK}}((a, b, c); (r', s'))$. The homomorphic scheme is IND-CPA secure assuming the hardness of the DLIN assumption.

Threshold version. In this version, the parties first agree on a group $\mathbb{G}$ of order p and a generator g. Then each party P_i picks $x_i, y_i, z_i \in \mathbb{Z}_p$ and sends $u_i = g^{x_i}, v_i = g^{y_i}, h_i = g^{z_i}$ it to all other parties. Finally, the parties compute $u = \prod_{i=1}^n u_i$, $v = \prod_{i=1}^n v_i$ and $h = \prod_{i=1}^n h_i$. Clearly the secret key $(x, y, z) = (\sum_{i=1}^n x_i, \sum_{i=1}^n y_i, \sum_{i=1}^n z_i)$ is shared amongst the parties. In order to ensure correct behaviour, the parties must prove knowledge of their secret key (x_i, y_i, z_i) by running a zero-knowledge proof on it. To ensure simulation-based security, each party must commit to its share first and decommit this commitment only after the commit phase is completed. Note that the simulator can enforce the public key outcome by rewinding the corrupted parties after seeing their decommitment information. Furthermore, the threshold decryption can be made non-interactive by posting a decryption share and proof of consistency. For using the encryption scheme in the protocol, we also use two more algorithms Rerand and Eval. Rerand is the algorithm used to rerandomize the ciphertext while Eval is added to evaluate a polynomial encrypted using an encryption scheme at an evaluation point.

The BBS threshold encryption scheme is a tuple of protocols $(\pi_{\mathsf{KeyGen}}, \pi_{\mathrm{DecZero}})$ and a tuple of three algorithm $(\mathsf{Enc}, \mathsf{Rerand}, \mathsf{Eval})$:

- π_{KeyGen} is an interactive protocol among parties $P_1 \dots, P_n$ where each party P_i receives an output $(\mathrm{PK}, \mathrm{SK}_i)$ where PK is the public key used for encryption and SK_i is the share of the secret key given to SK_i which will be used in threshold decryption.
- $c_m \leftarrow \mathsf{Enc}_{\mathrm{PK}}(m; (r, s))$: inputs a message m, randomness r, s and public key PK and outputs a ciphertext c_m.
- $c' \leftarrow \mathsf{Rerand}_{\mathrm{PK}}(c; (r, s))$: inputs a ciphertext c, randomness (r, s) and public key PK and outputs a re-randomized ciphertext c'.
- $c_y \leftarrow \mathsf{Eval}_{\mathrm{PK}}(, \mathbf{C}, t; (r, s))$: inputs the encrypted polynomial $\mathbf{C}$ in form of a vector of ciphertext represented the encrypted coefficient of the polynomial, randomness (r, s), an evaluation point t and public key PK and outputs c_y which is encrypted evaluation.
- π_{DecZero} is an interactive protocol among parties $P_1, \dots, P_n$ where a party inputs a ciphertext c. Additionally, all the parties input their share of secret

key (SK_i) as an input. The protocol outputs 1 if c encrypts a 0-message and 0 otherwise.

Protocol π_{KeyGen}:

- Parties first agree on a group $\mathbb{G}$ of order p and two generator u, v.
- Each party P_i randomly chooses $x_i, y_i \in \mathbb{F}$ such that $u^{x_i} = v^{y_i}$. P_i also computes $g_i = u^{x_i}$.
- P_i broadcast: Each party P_i sends g_i and π_{DL} with inputs $((\mathbb{G}, u, g_i), x_i)$ to prove knowledge of x_i.
- Each party P_i randomly chooses $z_i \in \mathbb{F}$ and computes $g = \prod_{i=1}^{n} g_i$, $h_i = g^{z_i}$.
- P_i broadcast: Each party P_i sends h_i and π_{DL} with inputs $((\mathbb{G}, g, h_i), z_i)$ to prove knowledge of x_i.
- Each party P_i computes $h = \prod_{i=1}^{n} h_i$.
- Upon verifying the zero-knowledge proof received, each party P_i outputs $\mathrm{PK} = (\mathbb{G}, p, g, h, u, v)$ and $\mathrm{SK}_i = (x_i, y_i, z_i)$.

Algorithm $\mathsf{Enc}_{\mathbf{PK}}(m, (r, s))$: Output $c_m = (u^r, v^s, g^{r+s}h^m)$.

Algorithm $\mathsf{Rerand}_{\mathbf{PK}}(c, (r, s))$: Split $(c_1, c_2, c_3) = c$ and output $c' = (c_1 \cdot u^r, c_2 \cdot v^s, c_3 \cdot g^{r+s})$.

Algorithm $\mathsf{Eval}_{\mathbf{PK}}(\mathbf{C}, t; (r, s))$: Split $\mathbf{C} = \{c_0, c_1, \ldots c_d\}$ and compute $c_y = \prod_{i=0}^{d} {c_i}^{t^i}$. Next, split $(x, y, z) = c_y$ and output $c' = (x \cdot u^r, y \cdot v^s, z \cdot g^{r+s})$.

Protocol $\pi_{\mathbf{DecZero}}$:

- P_i broadcast: Each party P_i rerandomizes the ciphertext c as c_i. P_i sends c_i along with proof π_{eq} showing the message encrypted in c and c_i is same.
- Each party compute $c^* = \prod_{i=1}^{n} c_i$.
- P_i broadcast: Set $(d, e, f) = c^*$. Each party P_i sends $d' = d^{x_i}$ and $e' = e^{y_i}$. To ensure consistency with their secret key, π_{pow} is used to show that the exponent is same in d', u^{x_i} as well as e', v^{y_i}.
- Each party verifies if $\frac{f}{d' \cdot e'} = 1$. Output 1 if true else outputs 0.

2.4 Commitment Schemes

A commitment scheme is a cryptographic primitive that allows a commitment to commit to a message by sending a commitment which reveals nothing about the message while later can be opened to a specific message.

Definition 7 (Commitment Scheme) *A commitment scheme is a tuple of three algorithm* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Commit})$ *and defined as follows:*

– *params* $\leftarrow \mathsf{Setup}(1^\kappa)$*: Inputs security parameter κ and outputs parameters params.*
– $\mathsf{ck} \leftarrow \mathsf{KeyGen}(params)$*: Inputs parameter params and outputs commitment key* ck.

– $\mathsf{com} \leftarrow \mathsf{Commit}_{\mathsf{ck}}(m; r)$: *Inputs commitment key* ck*, message* m *and randomness* r *and outputs commitment* com.

A commitment scheme satisfies these security properties:

– *Binding: For all PPT adversaries* $\mathcal{A}$*, there exists a negligible function* $\epsilon(\cdot)$ *such that:*

$$\begin{aligned}\Pr\Big[&params \leftarrow \mathsf{Setup}(1^{\kappa});\\ &\mathsf{ck} \leftarrow \mathsf{KeyGen}(params);\\ &(m_0, r_0, m_1, r_1) \leftarrow \mathcal{A}(1^{\kappa}, params; r_{\mathcal{A}});\\ &\mathsf{com}_0 = \mathsf{Commit}_{\mathsf{ck}}(m_0; r_0)\\ &\mathsf{com}_1 = \mathsf{Commit}_{\mathsf{ck}}(m_1; r_1)\\ &\quad \mathsf{com}_0 = \mathsf{com}_1 \wedge m_0 \neq m_1\Big] \leq \epsilon(\kappa)\end{aligned}$$

– *Hiding: For all PPT adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, there exists a negligible function* $\epsilon(\cdot)$ *such that:*

$$\begin{aligned}\Pr\Big[&params \leftarrow \mathsf{Setup}(1^{\kappa});\\ &\mathsf{ck} \leftarrow \mathsf{KeyGen}(params);\\ &(m_0, m_1, r) \leftarrow \mathcal{A}_1(1^{\kappa}, params; r_{\mathcal{A}});\\ &b \leftarrow_R \{0, 1\}\\ &\mathsf{com} = \mathsf{Commit}_{\mathsf{ck}}(m_b)\\ &\tilde{b} \leftarrow \mathcal{A}_2(\mathsf{com}, r)\\ &\quad b = \tilde{b}\Big] \leq 1/2 + \epsilon(\kappa)\end{aligned}$$

For our polynomial commitment protocol, we require two instances of commitment. One commitment scheme is used for committing to the encrypted polynomials while the other is used for committing the evaluation point. We require an additional property for one of the commitments (commitment used for encrypted polynomials). The commitment needs to be a randomized variant of doubly homomorphic as defined in [17].

Definition 8 (Doubly Homomorphic Commitment Scheme) *A Commitment scheme* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Commit})$ *is randomized doubly homomorphic if:*

– $\mathsf{Commit}_{\mathsf{ck}}(m; r) + \mathsf{Commit}_{\mathsf{ck}}(m'; r') = \mathsf{Commit}_{\mathsf{ck}}(m + m'; r + r')$.
– $\mathsf{Commit}_{\mathsf{ck}_m}(m; r) + \mathsf{Commit}_{\mathsf{ck}'_m}(m; r')$
$= \mathsf{Commit}_{(\mathsf{ck}_m + \mathsf{ck}'_m)}(m; r + r')$.

where $\mathsf{ck}_r, \mathsf{ck}'_r$ *are parts of the commitment key associated with randomness and* $\mathsf{ck}_m, \mathsf{ck}'_m$ *are parts of the commitment key associated with the message committed while* ck *is a commitment key.* m *and* m' *are messages while* r, r' *are the randomness used.*

The Pedersen Commitment Scheme. The Pedersen commitment scheme (denoted by Ped) [55] is defined as follows:

- $(\mathbb{G}, p) \leftarrow \mathsf{Ped.Setup}(1^\kappa)$: Outputs $\mathbb{G}$ with order p.
- $\mathsf{ck} = (g_0, \ldots, g_{d-1}, h) \leftarrow \mathsf{Ped.KeyGen}(\mathbb{G}, p, d)$: Outputs the commitment key ck. Here d is the number of group elements to be committed.
- $\mathsf{com}_{\mathbf{m}} = \mathsf{Ped.Commit}_{\mathsf{ck}}(\mathbf{m}, r) = h^r \prod_{i=0}^{d-1} g^{m_i}$: Outputs the commitment of message $\mathbf{m}$ where $\mathbf{m} = (m_0, \ldots, m_{d-1})$.

The Pedersen commitment scheme is computationally binding under the discrete logarithm assumption, i.e., any two different openings of the same commitment are reduced to computing $\log_g h$. Finally, it is perfectly hiding since a commitment is uniformly distributed in $\mathbb{G}$. The scheme is additively homomorphic.

The Commitment Scheme in [4] . The pairing based commitment in [4] (denoted by AFG) is defined as follows:

- $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, p, e, w, g) \leftarrow \mathsf{AFG.Setup}(1^\kappa)$: Outputs $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_t$ of order p with w and g being generators for $\mathbb{G}_1$ and $\mathbb{G}_2$ while e is the bilinear map.
- $\mathsf{ck} = (w_r, w_0, w_2, \ldots, w_{d-1}) \leftarrow \mathsf{AFG.KeyGen}(\mathbb{G}_1,\ \mathbb{G}_2,\ \mathbb{G}_t,\ p,\ e,\ w,\ g,\ d)$: Outputs the commitment key ck. Here d is the number of group elements to be committed.
- $\mathsf{com}_{\mathbf{m}} = \mathsf{AFG.Commit}_{\mathsf{ck}}(\mathbf{m}, r) = e(w_r, r) \prod_{i=0}^{d-1} e(w_i, m_i)$: Outputs the commitment of message $\mathbf{m}$ where $\mathbf{m} = (m_0, \ldots, m_{d-1})$.

The above commitment scheme is perfectly hiding since a commitment is uniformly distributed in $\mathbb{G}$. The scheme is also computationally binding under the double pairing assumptions stated in Definition 4.

Forking Lemma Consider a public-coin interactive protocol with r rounds. We define $(n_1, \ldots, n_r)$-tree of accepting transcripts for this interactive protocol as follows. The tree is of depth r where the root is labelled with the statement and each node in depth i has n_i children, where each child is associated with the i^{th} challenge. Each edge from parent to child node is associated with a message sent from the prover to the verifier. Each root-to-leaf path corresponds to an accepting transcript. Thus, the tree represents $\prod_{i=1}^r n_i$ different transcripts.

Lemma 1. *Forking Lemma [14] Let (C, R) be an r-round public coin interactive protocol. Let $\mathcal{X}$ be a witness extraction algorithm that succeeds with probability $1 - \mathsf{negl}(\kappa)$ for some negligible function $\mathsf{negl}(\kappa)$ in extracting a witness from an $(n_1, \ldots, n_r)$-tree of accepting transcripts in probabilistic polynomial time. Assume that $\prod_{i=1}^r n_i$ is bounded above by a polynomial in the security parameter κ. Then (C, R) has witness-extended emulation.*

We utilize the forking lemma to reduce the witness-extended emulation property of our polynomial commitment schemes to the existence of a PPT extractor, which given $(n_1, \ldots, n_r)$-tree of accepting transcripts can extract the witness of the polynomial commitment scheme.

2.5 Zero-Knowledge Proofs

Our PSI protocol employs three types of ZK proofs. The following proof π_{DL} is required for proving consistency in our maliciously secure threshold decryption protocol. Namely, π_{DL} is employed for demonstrating the knowledge of a solution x to a discrete logarithm problem [61]. Formally stating, $\mathcal{R}_{DL} = \{((\mathbb{G}, g, h), x) \mid h = g^x\}$.

π_{EXP} is a ZK proof of knowledge for demonstrating the knowledge with respect to an additively homomorphic commitment scheme. This protocol is used in our multi-party PSI protocol for two different purposes. Firstly, P_1 broadcasts its input by sending the commitment of its input and proving knowledge of it. Secondly, it is also used by all other parties while they broadcast a ciphertext generated by evaluating their input polynomial on a common random point to demonstrate the knowledge of plaintext of the ciphertext. As P_1 uses Pedersen commitment to commit to its input, we realize π_{EXP} using a standard Σ-protocol for the following relation: $\mathcal{R}_{\mathrm{EXP}} = \{((\mathbb{G}, g, h, h'), (m, r)) \mid h' = g^m h^r\}$.

The proof π_{pow} is a ZK proof for demonstrating the equality of exponent. This is used for generating the public key for the threshold encryption scheme. We realize π_{pow} using a standard $\Sigma-$protocol for the following relation: $\mathcal{R}_{\mathsf{pow}} = \{((\mathbb{G}, g, h, c_1, c_2), (m)) \mid c_1 = g^m \wedge c_2 = h^m\}$.

Additionally, π_{COIN} is a coin tossing protocol which is executed in the PSI protocol in order to sample a random group element for verifying the correctness of aggregation. Its overhead is $O(n^2)$ for n parties.

2.6 Inner Product Argument [15]

The inner product argument allows a party to provide proof for correct inner product evaluation. More formally, given two vectors a and b (where the vectors can be hidden or known), proves that the inner product is equal to a known (or committed) value c.

[14] introduces an inner product argument with logarithmic proof size which is improved by [15] which relies on Discrete Log (DL) assumption. The variant of the protocol used in our construction requires both vectors and inner products to be private. Therefore, the prover provides commitments to the two vectors and the inner product. The inner product argument utilizes masking to achieve honest-verifier zero-knowledge in addition to completeness and witness extended emulation [15] (Section 4). The prover recursively reduces verifying the inner product of two large vectors into verifying the inner product of small vectors. In the last iteration, This utilizes the doubly homomorphic property of the underlying commitments wherein the commitments used are homomorphic in both key-space and message-space. In Figure 1, we present the inner product argument using our notations and is denoted by BBB-IPA.

3 Private Polynomial Commitment Schemes

In this section, we introduce a new polynomial commitment scheme with privacy features. Loosely speaking, such a protocol is carried out between a committer C

Inner Product Argument (BBB-IPA)

Private Inputs: C : $A \in \mathbb{Z}_p^{d+1}, B \in \mathbb{Z}_p^{d+1}, c \in \mathbb{Z}_p$.
Public Inputs: $\mathsf{ck}_1, \mathsf{ck}_2, \mathsf{ck}_3, \mathsf{com}_A, \mathsf{com}_B, \mathsf{com}_c$.
Protocol:

1. C and R compute the combined commitment $\mathsf{com} = \mathsf{com}_A \cdot \mathsf{com}_B \cdot \mathsf{com}_c$.

For round $rnd = 1$ to $\log d - 1$:

2. Set $d' = (d+1)/2$. C sets $A_L = A[: d']$, $A_R = A[d' :]$, $B_L = F[: d']$ and $B_R = F[d' :]$ while both C and R sets $\mathsf{ck}_{1L} = \mathsf{ck}_1[: d']$, $\mathsf{ck}_{1R} = \mathsf{ck}_1[d' :]$, $\mathsf{ck}_{2L} = \mathsf{ck}_2[: d']$, and $\mathsf{ck}_{2R} = \mathsf{ck}_2[d' :]$.
3. C generates intermediate cross-commitments:

   $\mathsf{com}_{A_L} = \mathsf{Commit}_{\mathsf{ck}_{1R}}(A_L, r_{A_L})$. $\mathsf{com}_{A_R} = \mathsf{Commit}_{\mathsf{ck}_{1L}}(A_R, r_{A_R})$
   $\mathsf{com}_{B_L} = \mathsf{Commit}_{\mathsf{ck}_{2R}}(B_L, r_{B_L})$, $\mathsf{com}_{B_R} = \mathsf{Ped.Commit}_{\mathsf{ck}_{2L}}(B_R, r_{B_R})$,

   where $r_{A_L}, r_{A_R}, r_{B_L}, r_{B_R} \in \mathbb{Z}_p$.
4. $\mathsf{C} \to \mathsf{R}$: C generates L and R and sends L, R to C:
   $c_l = \langle A_R, B_L \rangle$, $c_r = \langle A_L, B_R \rangle$,
   $L = \mathsf{com}_{A_R} \cdot \mathsf{com}_{B_L} \cdot \mathsf{Commit}_{\mathsf{ck}_3}(c_l)$, $R = \mathsf{com}_{A_L} \cdot \mathsf{com}_{B_R} \cdot \mathsf{Commit}_{\mathsf{ck}_3}(c_r)$.
5. $\mathsf{R} \to \mathsf{C}$: R sends a random challenge $x \in \mathbb{Z}_p$.
6. C sets $A' = A_L + x \cdot A_R$ $\quad B' = B_L + x^{-1} \cdot B_R$ while C and R both locally compute the new keys $\mathsf{ck}'_1 = \mathsf{ck}_{1L} \odot \mathsf{ck}_{1R}^{x^{-1}}$ and $\mathsf{ck}'_2 = \mathsf{ck}_{2L} \odot \mathsf{ck}_{2R}^{x}$ where $\odot$ denotes element-wise multiplication of two vectors.
7. R computes the new commitment $\mathsf{com}' = L^x \cdot \mathsf{com} \cdot R^{x^{-1}}$.
8. C and R will update $A = A'$, $B = B'$, $\mathsf{com} = \mathsf{com}'$, and $\mathsf{ck}_i = \mathsf{ck}'_i \forall i \in [2]$.

In round $\log d$:

9. In the last round, C opens com to A', B' and c' and R accepts if $c' = \langle A', B' \rangle$.
10. If all checks pass, R outputs $b = 1$ else output $b = 0$.

Fig. 1: Inner Product Argument (BBB-IPA)

and a receiver R where C commits to an encrypted polynomial $\mathbf{C}$, denoted by a sequence of ciphertexts $\mathbf{C} = (c_0, c_1, \ldots, c_d)$ where c_i is a ciphertext that encrypts the i^{th} coefficient of the underlying plaintext polynomial. In these schemes, upon committing to the encrypted polynomial, C sends $\mathbf{C}$ to R and later evaluates it at an evaluation point t. Following that, C proves that a ciphertext c_y is a correct evaluation of the encrypted polynomial at some private evaluation point t.

3.1 Security Definitions

We continue with the security definition of our new polynomial commitments.

Definition 9 *(Private Polynomial Commitments with Hidden Evaluation Points)*

Let $\mathsf{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval}, \mathsf{Rerand})$ *be an AHE scheme with groups* $\mathcal{M}$ *and* $\mathcal{C}$*.Let* PK *be the public key of the underlying AHE scheme and generated by* $\mathsf{E.KeyGen}$*. A private commitment scheme* PCOM *w.r.t* E *is a tuple of algorithms* $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{CommitPt})$ *and a protocol* (C, R) *defined as follows:*

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa, d)$*: takes an input* κ, d *where* κ *is the security parameter and* d *is the degree of the polynomial, and outputs public parameters* pp*.*
- $\mathsf{com}_{\mathbf{C}} \leftarrow \mathsf{Commit}(\mathsf{pp}, \mathbf{C}; r_{\mathbf{C}})$*: takes as input a public parameters* pp*, a vector of ciphertexts (representing an encrypted polynomial)* $\mathbf{C} = (c_0, c_1, \ldots, c_d)$ *where* $c_i \in \mathcal{C}$ *for all* i *and randomness* $r_{\mathbf{C}}$*, and outputs a commitment* $\mathsf{com}_{\mathbf{C}}$*.*
- $\mathsf{com}_T \leftarrow \mathsf{CommitPt}(\mathsf{pp}, t, d; r_T)$: *takes as input public parameters* pp*, an evaluation point* t*, a randomness* r_T *and* d *is the degree of the polynomial and outputs a commitment* com_T*.*
- (C, R) *is a public-coin interactive protocol between* C *and* R*. Both* C *and* R *have common inputs, public parameters* pp*, a public-key* PK *for the underlying AHE scheme, a commitment* $\mathsf{com}_{\mathbf{C}}$*, another commitment* com_T *and an evaluation ciphertext* $c_y, \in \mathcal{C}$*.* C *additionally receives as input an encrypted polynomial* $\mathbf{C}$*, an evaluation point* t *and randomness* $r_{\mathbf{C}}, r_{c_y}, r_t$*. At the end of the protocol execution,* R *either outputs accept or reject. We denote by* $\big(\mathsf{C}(\mathbf{C}, t, r_{\mathbf{C}}, r_{c_y}, r_T), \mathsf{R}\big)$ $(\mathsf{pp}, PK, \mathsf{ck}, \mathsf{com}_{\mathbf{C}}, \mathsf{com}_T, c_y)$ *the random variable representing an execution and given an instance of the execution* e*, we denote by* $\mathsf{view}_1(e)$ *(resp.* $\mathsf{view}_2(e)$*) the view of the* C *(resp.,* R*) and* $\mathsf{out}_1(e)$ *(resp.,* $\mathsf{out}_2(e)$*) the output of* C *(resp.,* R*).*

We require the following security properties to be satisfied:

Completeness: *For any vector of ciphertexts* $\mathbf{C} = (c_0, c_1, \ldots, c_d)$ *generated using* $PK \leftarrow \mathsf{E.KeyGen}(1^\kappa)$ *and an evaluation point* t*, we have that:*

$$\begin{aligned}\Pr\Big[&\mathsf{pp} \leftarrow \mathsf{PCOM.Setup}(1^\kappa, d);\\ &\mathsf{com}_{\mathbf{C}} \leftarrow \mathsf{PCOM.Commit}(\mathsf{pp}, \mathbf{C}; r_{\mathbf{C}});\\ &\mathsf{com}_T \leftarrow \mathsf{PCOM.CommitPt}(\mathsf{pp}, t, d; r_T);\\ &c_y = \mathsf{Eval}(PK, \mathbf{C}, t; r_{c_y}) :\\ &\quad \mathsf{out}_2(\mathsf{C}(\mathbf{C}, t, r_{\mathbf{C}}, r_{c_y}, r_T), \mathsf{R})\\ &\quad (\mathsf{pp}, PK, \mathsf{com}_{\mathbf{C}}, \mathsf{com}_T, c_y) = 1\Big] = 1\end{aligned}$$

Binding: *For all PPT adversaries* $\mathcal{A}$*, there exists a negligible function* $\epsilon(\cdot)$ *such that:*

$$\begin{aligned}
\Pr\Big[&\mathsf{pp} \leftarrow \mathsf{PCOM.Setup}(1^\kappa, d);\\
&PK \leftarrow \mathsf{E.KeyGen}(1^\kappa);\\
&(\mathbf{C}_0, r_{\mathbf{C}_0}, \mathbf{C}_1, r_{\mathbf{C}_1}, t_0, r_{T_0}, t_1, r_{T_1}) \leftarrow \mathcal{A}(1^\kappa, n, \mathsf{pp}, PK; r_\mathcal{A});\\
&\mathsf{com}_{\mathbf{C}_0} = \mathsf{PCOM.Commit}(\mathsf{pp}, \mathbf{C}_0; r_{\mathbf{C}_0})\\
&\mathsf{com}_{\mathbf{C}_1} = \mathsf{PCOM.Commit}(\mathsf{pp}, \mathbf{C}_1; r_{\mathbf{C}_1})\\
&\mathsf{com}_{T_0} = \mathsf{PCOM.CommitPt}(\mathsf{pp}, t_0, d; r_{T_0})\\
&\mathsf{com}_{T_1} = \mathsf{PCOM.CommitPt}(\mathsf{pp}, t_1, d; r_{T_1})\\
&(\mathsf{com}_{\mathbf{C}_0} = \mathsf{com}_{\mathbf{C}_1} \wedge \mathbf{C}_0 \neq \mathbf{C}_1)\\
&\quad \vee (\mathsf{com}_{T_0} = \mathsf{com}_{T_1} \wedge t_0 \neq t_1)\Big] \leq \epsilon(\kappa)
\end{aligned}$$

Witness-Extended Emulation: *For all PPT adversaries $\mathcal{A}$, there exists an expected polynomial time emulator $\mathcal{E}$ and negligible function $\epsilon(\cdot)$ such that:*

$$\begin{aligned}
\Pr\Big[&pp \leftarrow \mathsf{PCOM.Setup}(1^\kappa, d);\\
&PK \leftarrow \mathsf{E.KeyGen}(1^\kappa);\\
&(\mathsf{com}_\mathbf{C}, \mathsf{com}_T, c_y) \leftarrow \mathcal{A}(1^\kappa, n, \mathsf{pp}, PK; r_\mathcal{A});\\
&e \leftarrow (\mathcal{A}(r_\mathcal{A}), \mathsf{R})(pp, PK, \mathsf{com}_\mathbf{C}, \mathsf{com}_T, c_y);\\
&(\mathbf{C}, t, r_\mathbf{C}, r_{c_y}, r_T) \leftarrow \mathcal{E}^{\mathcal{A}(\mathsf{pp}, PK, \mathsf{com}_\mathbf{C}, \mathsf{com}_T, c_y; r_\mathcal{A})}\\
&\qquad\qquad (\mathsf{pp}, PK, \mathsf{com}_\mathbf{C}, \mathsf{com}_T, c_y, e) :\\
&(\mathsf{out}_2(e) = 1) \Rightarrow\\
&\quad (\mathsf{com}_\mathbf{C} = \mathsf{PCOM.Commit}(\mathsf{pp}, \mathbf{C}; r_\mathbf{C})\\
&\qquad \wedge \mathsf{com}_T = \mathsf{PCOM.CommitPt}(\mathsf{pp}, t, d; r_T)\\
&\qquad\quad \wedge\ c_y = \mathsf{Eval}_{PK}(\mathbf{C}, t; r_{c_y}))\Big] \geq 1 - \epsilon(\kappa)
\end{aligned}$$

Honest Verifier Privacy: *There exists a tuple of expected PPT algorithms $\mathcal{S}$, given any vector of coefficient of polynomial $(p_0, \ldots, p_d)$ and an evaluation point t, such that the following distributions are indistinguishable:*

$$- \left\{\begin{array}{c}
\mathsf{pp} \leftarrow \mathsf{PCOM.Setup}(1^\kappa, d);\\
PK \leftarrow \mathsf{E.KeyGen}(1^\kappa);\\
\mathbf{C} \leftarrow (c_0, \ldots, c_d) = (\mathsf{Enc}_{PK}(p_0; r_0), \ldots, \mathsf{Enc}_{PK}(p_d; r_d)) :\\
\mathsf{com}_\mathbf{C} \leftarrow \mathsf{PCOM.Commit}(\mathsf{pp}, \mathbf{C}; r_\mathbf{C});\\
\mathsf{com}_T \leftarrow \mathsf{PCOM.CommitPt}(\mathsf{pp}, t, d; r_t);\\
c_y \leftarrow \mathsf{Eval}_{PK}(\mathbf{C}, t; r_{c_y});\\
e \leftarrow (\mathsf{C}(\mathbf{C}, t, r_\mathbf{C}, r_{c_y}, r_T, r_\mathcal{A}), \mathsf{R})\\
(\mathsf{pp}, PK, \mathsf{com}_\mathbf{C}, \mathsf{com}_T, c_y) :\\
\mathsf{view}_2(e)
\end{array}\right\}$$

$$- \left\{\begin{array}{c}
pp \leftarrow \mathsf{PCOM.Setup}(1^\kappa, d);\\
PK \leftarrow \mathsf{E.KeyGen}(1^\kappa);\\
\mathcal{S}(\mathsf{pp}, PK, d; r_\mathcal{S})
\end{array}\right\}$$

3.2 Our Protocols

In this section, we present the construction of our private polynomial commitment. Our construction is based on the additive homomorphic encryption (AHE) scheme from [13] and the inner-pairing product argument from [17]. As a warm-up, we start by considering a single point where the idea is that the evaluation of a polynomial $f(x) = \sum_{i=0}^{d} a_i x^i$ at point t can be viewed as the inner product between the coefficients vector $(a_0, a_1, \dots, a_d)$ and the evaluation vector $T = (1, t, t^2, \dots, t^d)$. Therefore, given the ciphertexts encrypting the coefficients and the commitments of the evaluation vector T, the committer proves in Phase 1 that the polynomial evaluation on the ciphertext is indeed the inner product between the two vectors using the techniques in [17]. Next, it remains to show that the committed evaluation vector is well-formed, i.e., it is indeed the powers of the evaluation point t. To prove this property, denoting the i-th element in a vector T as $T[i]$, it suffices to show that (1) the 0-th element $T[0]$ is 1; (2) $T[i+1] = T[i] \cdot T[1]$ for $i = 0, \dots, d-1$. These two conditions can further be translated into two types of constraints: linear constraints and quadratic constraints. The first condition is equivalent to the inner product between T and a public vector $(1, 0, \dots, 0)$ is 1. For the second condition, we define three selector matrices $A, B, C \in \mathbb{F}^{d \times (d+1)}$ such that

$$\begin{aligned} X &= A \times T = (T[0], T[1], \dots, T[d-1]), \\ Y &= B \times T = (T[1], T[1], \dots, T[1]), \\ Z &= C \times T = (T[1], T[2], \dots, T[d]). \end{aligned} \tag{1}$$

Finally, the committer proves that $X \odot Y = Z$, where $\odot$ denotes the Hadamard (element-wise) product. It is not hard to see that T is the correct evaluation vector if and only if it satisfies these constraints.

We use standard techniques such as [15] to reduce the linear constraints and the quadratic constraints to inner product arguments in Phases 2 and 3. Note that the protocols in these two phases are independent of the ciphertexts encrypting the coefficients. The formal protocol of our private polynomial commitment is presented in Figure 2. This protocol uses the encryption scheme from [13], the pairing-based commitment from [4] and the Pedersen commitment [55] (see Appendix 2) as building blocks. The protocol also involves private inner product argument, linear constraints test and quadratic constraints test, as described above in the three phases. We present these protocols later in Figures 4, 5 and 6 together with our scheme for multiple evaluations.

Multiple Evaluations. The major advantage of our construction is that it supports batched evaluations on multiple points efficiently, where the proof size and the receiver's time do not increase by much compared to a single evaluation. We describe our scheme for multiple evaluations in Figures 3. The differences from the single evaluation variant are highlighted in purple. In particular, in Phase 1 (Steps 1 and 2 in Figure 3), C and R check the inner products between the coefficient vector in the ciphertext and all the evaluation vectors in the commitments using a single private inner product argument protocol via a random

$\mathsf{Setup}(1^\kappa, d)$**:** Generate the public parameters of the bilinear map and the commitment scheme Ped and AFG. $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, p, e, w, g) \leftarrow \mathcal{G}(1^\kappa)$, $\mathsf{ck}_1 = (w_r, w_0, \ldots, w_d), a, b) \leftarrow \mathsf{AFG}.\mathit{KeyGen}(S, 3d+8)$.
$\mathsf{ck}_2 = (v_r, v_0, \ldots, v_d) \leftarrow \mathsf{Ped.KeyGen}(1^\kappa, d+2)$, $\mathsf{ck}_3 \leftarrow \mathsf{Ped.KeyGen}(1^\kappa, 2)$, $\mathsf{ck}_4 = (x_r, x_0, \ldots, x_d) \leftarrow \mathsf{Ped.KeyGen}(1^\kappa, d+2)$.
Output $pp = (\mathsf{ck}_1, \mathsf{ck}_2, \mathsf{ck}_3, \mathsf{ck}_4, a, b)$.

$\mathsf{Commit}(pp, \mathbf{C}, r_\mathbf{C})$**:** Given the ciphertext of the coefficients $\mathbf{C} = (c_0, \ldots, c_d)$, output $\mathsf{AFG.Commit}_{\mathsf{ck}_1}(\mathbf{C}, g^{r_\mathbf{C}}) = e(g^{r_\mathbf{C}}, w_r) \cdot \prod_{i=0}^{d} e(c_i, w_i)$, where $r_\mathbf{C} \in \mathbb{Z}_p$.
$\mathsf{CommitPt}(pp, t, r_T, d)$**:** Given an evaluation point t, generate $T = (1, t, \ldots, t^d)$ and output $\mathsf{Ped.Commit}_{\mathsf{ck}_2}(T, r_T)$ where $r_T \in \mathbb{Z}_p$.

Protocol $\mathit{\Pi}_{priv}(\mathsf{C}(\mathbf{C}, r_\mathbf{C}, t, r_T), \mathsf{R})(pp, \mathsf{com}_\mathbf{C}, \mathsf{com}_T, c_y)$**:**

1. C and R execute **Private inner Product Argument** specified in (Figures 4) with common input $pp, \mathsf{com}_\mathbf{C}, \mathsf{com}_T, c_y$ and $\mathbf{C}, T$ as private inputs to C.
2. $\mathsf{C} \rightarrow \mathsf{R}$: Let A, B, C be public selector matrices defined in Equation 1. C computes $X = A \times T = (1, t, \ldots, t^{d-1})$, $Y = B \times T = (t, \ldots, t)$, $Z = C \times T = (t, \ldots, t^d)$. C commits to X, Y, Z by $\mathsf{com}_X = \mathsf{Ped.Commit}_{\mathsf{ck}_2}(X, r_X), \mathsf{com}_Y = \mathsf{Ped.Commit}_{\mathsf{ck}_2}(Y, r_Y), \mathsf{com}_Z = \mathsf{Ped.Commit}_{\mathsf{ck}_2}(Z, r_Z)$, where $r_X, r_Y, r_Z \in \mathbb{Z}_p$. C sends $\mathsf{com}_X, \mathsf{com}_Y, \mathsf{com}_Z$ to R.
3. $\mathsf{C} \leftrightarrow \mathsf{R}$: C and R execute **Linear Constraints Test** specified in Figure 5 with common input $\mathsf{com}_T, \mathsf{com}_X$ and T, X as private inputs to C. Repeat the same for Y and Z. Let D be public selector matrix defined as $D \times T = [1]$, C and R execute **Linear Constraints Test** specified in Figure 5 with common input com_T, D and T as private inputs to C.
4. $\mathsf{C} \leftrightarrow \mathsf{R}$: C and R execute **Quadratic Constraint Test** specified in Figure 6 with common input $\mathsf{com}_X, \mathsf{com}_Y, \mathsf{com}_Z$ and X, Y, Z as private inputs to C.
5. R outputs 1 if all checks pass.

Fig. 2: Private Polynomial Commitments (Single Evaluation).

linear combination. In Phase 2 (Step 4 in Figure 3), the product between a selector matrix (i.e., A, B or C) and all the evaluation vectors can be reduced to a single inner product via two random linear combinations, as shown in Figure 5. In Phase 3 (Step 5 in Figure 3), the protocol of the quadratic constraint test is more complicated. We are not able to reduce the Hadamard product of matrices $X \odot Y = Z$ to a single inner product. Instead, we reduce the Hadamard product to the sum of m inner products via a random linear combination in Step 1 of Figure 6. Then we propose a protocol (Step 3 of Figure 6) to prove the sum of the inner products with a proof size of only $O(\log d)$. The protocol is an extension of the scheme for the Hadamard product in [15] in a non-black-box way.

Protocol $\Pi_{priv}^{batched}(\mathsf{C}(\mathbf{C}, r_{\mathbf{C}}, \{t_i\}_{i\in[m]}, \{r_{T_i}\}_{i\in[m]}), \mathsf{R})(pp, \mathsf{com}_{\mathbf{C}}, \{\mathsf{com}_{T_i}\}_{i\in[m]}, \{c_{y_i}\}_{i\in[m]})$**:**

1. $\mathsf{R} \to \mathsf{C}$: R sends $S = (s_1, s_2, \cdots, s_m) \in \mathbb{Z}_p^m$.
2. $\mathsf{C} \leftrightarrow \mathsf{R}$: Let $F = \sum_{i=1}^m s_i \cdot T_i$, $\mathsf{com}_F = \prod_{i=1}^m \mathsf{com}_{T_i}^{s_i}$ and $c_y = \prod_{i=1}^m c_{y_i}^{s_i}$.
   C and R execute **Private inner Product Argument** specified in (Figures 4) with common input $pp, \mathsf{com}_{\mathbf{C}}, \mathsf{com}_F, c_y$ and $\mathbf{C}, F$ as private inputs to C
3. $\mathsf{C} \to \mathsf{R}$: Let A, B, C be public selector matrices defined in Equation 1. C computes $X_i = (1, t_i, \ldots, t_i^{d-1})$, $Y_i = (t_i, \ldots, t_i)$ and $Z_i = (t_i, \ldots, t_i^d)$. Let $T \in \mathbb{Z}_p^{(d+1)\times m}$ be the matrix with the i-th column as T_i. C commits to each column of X, Y, Z, namely, $\mathsf{com}_{X_i} = \mathsf{Ped.Commit}_{\mathsf{ck}_2}(X_i, r_{X_i}), \mathsf{com}_{Y_i} = \mathsf{Ped.Commit}_{\mathsf{ck}_2}(Y_i, r_{Y_i}), \mathsf{com}_{Z_i} = \mathsf{Ped.Commit}_{\mathsf{ck}_2}(Z_i, r_{Z_i})$ where $r_{X_i}, r_{Y_i}, r_{Z_i} \in \mathbb{Z}_p$ and sends $\{\mathsf{com}_{X_i}, \mathsf{com}_{Y_i}, \mathsf{com}_{Z_i}\}_{i\in[m]}$ to the R.
4. $\mathsf{C} \leftrightarrow \mathsf{R}$: C and R execute **Linear Constraints Test** specified in Figure 5 with common input $\mathsf{pp}, \{\mathsf{com}_{T_i}\}_{i\in[m]}, \{\mathsf{com}_{X_i}\}_{i\in[m]}$ and T, X as private inputs to C. Repeat the same for Y and Z. Let D be public selector matrix defined as $D \times T = [1]^m$, C and R execute **Linear Constraints Test** specified in Figure 5 with common input com_T, D and T as private inputs to C.
5. $\mathsf{C} \leftrightarrow \mathsf{R}$: C and R execute **Quadratic Constraint Test** specified in Figure 6 with common input $\mathsf{pp}, \{\mathsf{com}_{X_i}\}_{i\in[m]}, \{\mathsf{com}_{Y_i}\}_{i\in[m]}, \{\mathsf{com}_{Z_i}\}_{i\in[m]}$ and X, Y, Z as private inputs to C.
6. R outputs 1 if all checks pass.

Fig. 3: Batched proof for Private Polynomial Commitments.

Theorem 1. *Protocol* PCOM *(Figure 3) is a private polynomial commitment scheme as in Definition 9, under the Decisional Linear (DLIN) and the Double Pairing Problem (DPP) hardness assumptions (see Section 2.2).*

Proof Sketch: To show PCOM is a private polynomial commitment scheme (Definition 9), we show that the protocol satisfies completeness, binding, witness-extended emulation and honest verifier privacy.

Completeness: In the private inner product argument test, there are two phases - the masking phase and the inner product phase. In the end, R accepts if the combined commitment of the private polynomial, evaluation vector and evaluation ciphertext is decommitted correctly. This essentially follows from showing that the commitment of the private polynomial, the commitment of the evaluation vector and the evaluation ciphertext are updated correctly in each round. The rest of the protocol involving the linear constraint test, quadratic test and the BBB-IPA follow essentially observing that the corresponding constraints are satisfied.

Private inner Product Argument

Private Inputs: C : $\mathbf{C} = (c_0, \ldots, c_d) \in \mathbb{G}_E^{d+1}, F = (f_0, \ldots, f_d) \in \mathbb{Z}_p^{d+1}$.
Public Inputs: $pp = (\mathsf{ck}_1, \mathsf{ck}_2, \mathsf{ck}_3, a, b, \mathrm{PK}), \mathsf{com}_{\mathbf{C}}, \mathsf{com}_F, c_y$.

1. **Masking Phase:**
   (a) $\mathsf{C} \to \mathsf{R}$: C generates a random encrypted polynomial $\mathbf{E} = (e_0, \ldots, e_d) \in \mathbb{G}_E^{d+1}$ where $e_i = \mathsf{Eval}_{\mathrm{PK}}(r_i)$ and $r_i \in \mathbb{Z}_p$. A random vector $M = (M_0, \ldots, M_d) \in \mathbb{Z}_p^{d+1}$ is also sampled and generates commitment $\mathsf{com}_{\mathbf{E}} = \mathsf{AFG.Commit}_{\mathsf{ck}_1}(\mathbf{E}, r_{\mathbf{E}})$ and $\mathsf{com}_M = \mathsf{Ped.Commit}_{\mathsf{ck}_2}(M, r_M)$ where $r_{\mathbf{E}}, r_M \in \mathbb{Z}_p$.
   C also computes: $c_l = \langle \mathbf{E}, F \rangle$, $c_r = \langle \mathbf{C}, M \rangle$, $c_m = \langle \mathbf{E}, M \rangle$ and sends $\mathsf{com}_{\mathbf{E}}, \mathsf{com}_M, c_l, c_r, c_m$ to R.
   (b) $\mathsf{R} \to \mathsf{C}$: R sends a random challenge $x \in \mathbb{Z}_p$.
   (c) Both parties set com' where: $\mathsf{com} = \mathsf{com}_{\mathbf{C}} \cdot e(\mathsf{com}_F, a) \cdot e(c_y, b)$, $\mathsf{com}' = \mathsf{com} \cdot \mathsf{com}_{\mathbf{E}}^x \cdot e(\mathsf{com}_M, a)^{x^{-1}} \cdot e(c_l^x \cdot c_m \cdot c_r^{x^{-1}}, b)$, and C sets $\mathbf{C}' = \mathbf{C} \odot \mathbf{E}^x$ and $F' = F + x^{-1} \cdot M$ where $\odot$ denotes element-wise multiplication of two vectors.
   (d) Both parties update $\mathsf{com} = \mathsf{com}'$, $\mathbf{C} = \mathbf{C}'$, $F = F'$.
2. **Inner Product Phase:**
   For round $rnd = 1$ to $\log d - 1$:
   (a) Set $d' = (d+1)/2$. C sets $\mathbf{C}_L = \mathbf{C}[: d']$, $\mathbf{C}_R = \mathbf{C}[d' :]$, $F_L = F[: d']$ and $F_R = F[d' :]$ while both C and R sets $\mathsf{ck}_{1L} = \mathsf{ck}_1[: d']$, $\mathsf{ck}_{1R} = \mathsf{ck}_1[d' :]$, $\mathsf{ck}_{2L} = \mathsf{ck}_2[: d']$, and $\mathsf{ck}_{2R} = \mathsf{ck}_2[d' :]$.
   (b) C generates intermediate cross-commitments:
   $\mathsf{com}_{\mathbf{C}_L} = \mathsf{AFG.Commit}_{\mathsf{ck}_{1R}}(\mathbf{C}_L, r_{\mathbf{C}_L})$, $\mathsf{com}_{\mathbf{C}_R} = \mathsf{AFG.Commit}_{\mathsf{ck}_{1L}}(\mathbf{C}_R, r_{\mathbf{C}_R})$, $\mathsf{com}_{F_L} = \mathsf{Ped.Commit}_{\mathsf{ck}_{2R}}(F_L, r_{F_L})$, $\mathsf{com}_{F_R} = \mathsf{Ped.Commit}_{\mathsf{ck}_{2L}}(F_R, r_{F_R})$, where $r_{\mathbf{C}_L}, r_{\mathbf{C}_R}, r_{F_L}, r_{F_R} \in \mathbb{Z}_p$.
   (c) $\mathsf{C} \to \mathsf{R}$: C generated L and R: $c_l = \langle \mathbf{C}_R, F_L \rangle$, $c_r = \langle \mathbf{C}_L, F_R \rangle$
   $L = \mathsf{com}_{\mathbf{C}_R} \cdot e(\mathsf{com}_{F_L}, a) \cdot e(c_l, b)$, $R = \mathsf{com}_{\mathbf{C}_L} \cdot e(\mathsf{com}_{F_R}, a) \cdot e(c_r, b)$, where $a, b \in pp$ and sends L, R to C.
   (d) $\mathsf{R} \to \mathsf{C}$: R sends a random challenge $x \in \mathbb{Z}_p$.
   (e) C sets $\mathbf{C}' = \mathbf{C}_L \odot \mathbf{C}_R^x$ and $F' = F_L + x^{-1} \cdot F_R$ where $\odot$ denotes element-wise multiplication of two vectors while C and R both locally compute the new keys $\mathsf{ck}_1' = \mathsf{ck}_{1L} \odot \mathsf{ck}_{1R}^{x^{-1}}$ and $\mathsf{ck}_2' = ck_{2L} \odot \mathsf{ck}_{2R}^x$
   (f) R computes new commitment $\mathsf{com}' = L^x \cdot \mathsf{com} \cdot R^{x^{-1}}$
   (g) C and R will update $\mathbf{C} = \mathbf{C}'$, $F = F'$, $\mathsf{com} = \mathsf{com}'$, and $\mathsf{ck}_i = \mathsf{ck}_i' \forall i \in [2]$
   In round $\log d$:
   (h) In the last round,C opens com to $\mathbf{C}', F'$ and c_y' and R accepts if $c_y = \langle \mathbf{C}', F \rangle$.
   (i) If all checks pass, R outputs $b = 1$ else output $b = 0$.

Fig. 4: Private Inner Product Argument.

Binding: To argue the binding property of PCOM, it can be trivially reduced to the binding property of the Ped and AFG commitment scheme.
Witness-Extended Emulation: To argue witness-extended emulation of PCOM, as shown in [15], it is enough to show that given $(n_1, \ldots, n_r)$-tree of

Linear constraint Test (Prove $A \times T = X$)

- Private Inputs: C has private inputs: $X \in \mathbb{Z}_p^{d\times m}, T \in \mathbb{Z}_p^{d+1\times m}$.
- Public Inputs: $pp = (\mathsf{ck}_1, \mathsf{ck}_2, \mathsf{ck}_3, a, b, \mathrm{PK})$, $\{\mathsf{com}_{T_i}\}_{i\in[m]}$, $\{\mathsf{com}_{X_i}\}_{i\in[m]}$ where $\mathsf{com}_{T_i}, \mathsf{com}_{X_i} \in \mathbb{G}_1$.
  1. $\mathsf{R} \to \mathsf{C}$: R sends random vectors $S \in \mathbb{Z}_p^d$ and $U \in \mathbb{Z}_p^m$. Let $S_A = S \times A$, $T_U = T \times U$, $X_U = X \times U$. We observe that if $A \times T = X$ then for any $S \in \mathbb{Z}_p^d$ and $U \in \mathbb{Z}_p^m$ we have:
  $$S \times A \times T \times U = S \times X \times U, \text{ i.e, } \langle S_A, T_U\rangle - \langle S, X_U\rangle = 0.$$
  2. $\mathsf{C} \to \mathsf{R}$: C computes two cross terms inner product l and r and sends their respective commitments $\mathsf{com}_l, \mathsf{com}_r$ to R: $l = \langle S_A, -X_U\rangle$, $r = \langle S, T_U\rangle$, $\mathsf{com}_l = \mathsf{Ped.Commit}_{\mathsf{ck}_3}(l, r_l)$, $\mathsf{com}_r = \mathsf{Ped.Commit}_{\mathsf{ck}_3}(r, r_r)$, where $r_l, r_r \in \mathbb{Z}_p$.
  3. $\mathsf{R} \to \mathsf{C}$: R sends a random challenge $x \in \mathbb{Z}_p$.
  4. $\mathsf{R} \leftrightarrow \mathsf{C}$: C computes $L = S_A + x^{-1} \cdot S$ and $R = T_U - x \cdot X_U$. C and R both compute $\mathsf{com}_L = \mathsf{Ped.Commit}_{ck_4}(S_A + x^{-1} \cdot S; 0)$ and $\mathsf{com}_R = \prod_{i=1}^{m} \mathsf{com}_{T_i}^{U[i-1]} \cdot \mathsf{com}_{X_i}^{U[i-1]\cdot x}$ .C and R execute BBB-IPA (Figure 1) on common inputs is $\mathsf{ck}_4, \mathsf{ck}_2, \mathsf{ck}_3, \mathsf{com}_L, \mathsf{com}_R, \mathsf{com}_l^x \cdot \mathsf{com}_r^{x^{-1}}$ and private inputs of C are , $L, R, x \cdot l + x^{-1} \cdot r$.
  5. If all checks pass, R outputs $b = 1$ else output $b = 0$.
- A special case is when $D \times T = X$ where X is a known vector of dimensions $1 \times m$. The above test can be simplified where R sends a random vector $U \in \mathbb{Z}_p^m$ and the check is reduced from $D \times T = X$ to $\langle D, T_U\rangle = d$ where $T_U = T \times U$ and $d = \sum_{i=0}^{m-1} U[i]$. C and R compute $\mathsf{com}_D = \mathsf{Ped.Commit}_{\mathsf{ck}_4}(D, 0)$, $\mathsf{com}_{T_U} = \prod_{i=1}^{m} \mathsf{com}_{T_i}^{U[i-1]}$,$\mathsf{com}_d = \mathsf{Ped.Commit}(d, 0)$ .C and R execute BBB-IPA (Figure 1) on common inputs is $\mathsf{ck}_4, \mathsf{ck}_2, \mathsf{ck}_3, \mathsf{com}_D, \mathsf{com}_{T_U}, \mathsf{com}_d$ and private inputs of C are D, T_U, d . If all checks pass, R outputs $b = 1$ else output $b = 0$.

Fig. 5: Linear Constraint Test.

accepting transcripts, there exist a PPT extractor $\mathcal{X}$ which extracts the witness for PCOM. To construct $\mathcal{X}$, we first construct a witness-extraction algorithm $\mathcal{X}_1$ that succeeds in extracting the witness of Private Inner Product Argument given $(n_1, \ldots, n_r)$-tree of accepting transcripts. Using the rewinding property of the extractor and choosing different randomness in each rewinding, the extractor $\mathcal{X}_1$ can extract the witness. Here, the witness is the encrypted polynomial, evaluation vector, encrypted evaluation and the randomness used to generate the commitments. Next $\mathcal{X}$ extracts the evaluation vector from Linear Test and Quadratic test to verify if the evaluation used in all three tests is the same. We use the witness-extended emulation extractor of BBB-IPA as a subprotocol in extracting the evaluation vector from the Linear and Quadratic tests.

Honest Verifier Privacy: To show honest verifier privacy, we construct a simulator $\mathcal{S}$. Indistinguishability of the simulation essentially follows from semantic security of the underlying encryption scheme, hiding of the commitment

Quadratic Constraint Test (Prove $X \odot Y = Z$)

Private Inputs: $\mathsf{C} : X, Y, Z \in \mathbb{Z}_p^{d \times m}$.
Public Inputs: $pp = (\mathsf{ck}_1, \mathsf{ck}_2, \mathsf{ck}_3, \mathsf{ck4}, a, b, \mathrm{PK}), \{\mathsf{com}_{X_i}\}_{i\in[m]}, \{\mathsf{com}_{Y_i}\}_{i\in[m]}, \{\mathsf{com}_{Z_i}\}_{i\in[m]}$ where $\mathsf{com}_{X_i}, \mathsf{com}_{Y_i}, \mathsf{com}_{Z_i} \in \mathbb{G}_1$.

1. $\mathsf{R} \rightarrow \mathsf{C}$: R sends a random vector $S \in \mathbb{Z}_p^m$ and a random value w. Now if $X \odot Y = Z$, then $\sum_{i\in m} w^i(\langle X_i, Y_i \odot S\rangle - \langle Z_i, S\rangle) = 0$.
2. Let $L_i = w^i \cdot X_i, L_{i+m} = w^i \cdot Z_i$, $R_i = Y_i \odot S, R_{i+m} = -S$
   C and R compute a new key ck_5 where $\mathsf{ck}_5[j] = \mathsf{ck}_2^{S[j]^{-1}}[j]$ for all $j \in [0, d]$ and compute the commitments as follows: $\mathsf{com}_{L_i} = \mathsf{com}_{X_i}^{w^i}, \mathsf{com}_{L_{i+m}} = \mathsf{com}_{Z_i}^{w^i}$, $\mathsf{com}_{R_i} = \mathsf{com}_{Y_i}, \mathsf{com}_{R_{i+m}} = \mathsf{Ped.Commit}_{\mathsf{ck}_5}(-S)$
3. C sets $d = 0$ while R sets $\mathsf{com}_d = 1$. Also set $m' = 2m$.
   For round 1 to $\log m$:
   (a) $\mathsf{C} \rightarrow \mathsf{R}$: Set m' = m'/2. C computes two cross terms inner product $l = \sum_{i=1}^{m'} \langle L_i, R_{i+m'}\rangle$ and $r = \sum_{i=1}^{m'} \langle L_{i+m'}, R_i\rangle$ and sends a Ped commitment of these two (com_l and com_r) to R.
   where $r_l, r_r \in \mathbb{Z}_p$.
   (b) $\mathsf{R} \rightarrow \mathsf{C}$: R sends a random challenge $x \in \mathbb{Z}_p$.
   (c) C computes $\{L'_i = L_i + x^{-1} \cdot L_{i+m}\}_{i\in[m']}$ and $\{R'_i = R_i + x \cdot R_{i+m}\}_{i\in[m']}$ while R updates the commitments $\mathsf{com}_{L'_i} = \mathsf{com}_{L_i} \cdot \mathsf{com}_{L_{i+m}}^{x^{-1}}$ and $\mathsf{com}_{R'_i} = \mathsf{com}_{R_i} \cdot \mathsf{com}_{R_{i+m}}^{x}$.
   (d) C computes $d' = d + x \cdot l + x^{-1} \cdot r$ while R computes $\mathsf{com}_{d'} = \mathsf{com}_d \cdot \mathsf{com}_l^x \cdot \mathsf{com}_r^{x^{-1}}$.
   (e) C updates $L_i = L'_i$, $R_i = R'_i$, $d = d'$ while R updates $\mathsf{com}_d = \mathsf{com}_{d'}$.
   In round $\log m + 1$:
   (f) C sets $L = L_1$ and $R = R_1$ while R sets $\mathsf{com}_L = \mathsf{com}_{L_1}$ and $\mathsf{com}_R = \mathsf{com}_{R_1}$ C and R execute BBB-IPA (Figure 1) on instance with common input $\mathsf{ck}_2, \mathsf{ck}_5, \mathsf{ck}_3, \mathsf{com}_L, \mathsf{com}_R, \mathsf{com}_d$ and L, R, d as private inputs of C.

Fig. 6: Quadratic Constraint Test.

scheme, honest-verifier zero-knowledge property of the underlying BBB-IPA and standard masking techniques.

We provide the full proof in Appendix A.1.

Complexity. The communication complexity of our polynomial commitments is $O(\log d)$ for a single evaluation and $O(m + \log d)$ for m points where d is the degree of the polynomial. Their round complexity is $O(\log m + \log d)$ rounds.

The computational complexity of the committer is $O(m \cdot d)$ modular exponentiations and $O(d)$ bilinear pairings, while the complexity of the receiver is $O(m+d)$ exponentiations. The space complexity of our private polynomial commitment scheme is $O(m+d)$ for the committer as it needs to store the encrypted polynomial and the evaluation points. The space complexity of the receiver is $O(m)$ (resp. $O(m+\log d)$) in the interactive (resp. non-interactive setting). This difference is because, in the non-interactive setting, the entire proof is stored for validation.

3.3 Other variants of the Private Polynomial Commitment

Non-interactive proofs and public verifiability via Fiat-Shamir transform. As the proof systems for the single and batched setting of private polynomial commitment schemes are public-coin (i.e. R only sends random coins during the interaction of the protocol), it can be transformed to a non-interactive proof system via the Fiat-Shamir transform [28]. Furthermore, these proofs will be publicly-verifiable.

Private polynomial commitment with public evaluation points. In the private polynomial commitment protocols from Figures 2 and 3, the evaluation points are known only to the committer and are committed to the receiver. It is not hard to change the protocols to support public evaluation points known both to the committer and the receiver. A naive approach is to execute Phase 1 only. As the receiver knows the evaluation points, it can compute the well-formed evaluation vectors on its own without the checks in Phases 2 and 3. However, in the batched variant in Figure 3, the complexity of the receiver would become $O(dm)$, as computing the commitments of the evaluation vectors takes $O(dm)$ time. Instead, to maintain the same complexity, the committer and the receiver still execute all three phases of the protocol. In Phase 2, the receiver computes the commitment of Y on its own. As $Y_i = (t_i, t_i, \ldots, t_i)$, computing the commitments of all Y_is only takes $O(d + m)$ time.

In our application for multi-party private set intersection (MPSI), we will rely on the non-interactive proof variant of our private polynomial commitment both with hidden and public points. Precisely, we will have a polynomial committed once and then incorporate proofs of evaluations on both types of points.

Multivariate polynomial commitment. Our protocols can also be generalized to support multivariate polynomials. The evaluation of a multivariate polynomial can also be viewed as the inner product between the coefficient vector and the evaluation vector computed by all monomials of the evaluation point. Therefore, Phase 1 of the protocols in Figure 2 and 3 remains the same. In Phases 2 and 3, we instead check the form of the evaluation vectors of the multivariate polynomial. These can be reduced to linear and quadratic constraints with different A, B, C matrices. The techniques to batch multiple evaluations in Figures 3, 6 and 5 remain the same.

4 Scalable Multi-Party PSI

Our first application is a new scalable PSI protocol that follows the blueprint of [46]. This protocol is carried out in a star topology network with P_1 being the central party. In this work, we show that the actions of P_1 can be captured by the abstraction of a private polynomial commitment.

We broadly split our protocol description into four main phases. In the first phase (Key Generation), the parties jointly generate a public key without disclosing their corresponding secret key shares, as well as the public parameters for the two polynomial commitments. The second phase (Commitment Phase) is executed by the central party P_1 that broadcasts commitments of its input together with a proof of knowledge. In the third phase (Aggregation), all parties (except P_1) send it an encrypted polynomial whose roots correspond to their inputs. P_1 combines these polynomials for each party and provides a commitment of the encrypted aggregated polynomial while proving the correctness of aggregation. The last phase (Intersection) concludes the protocol by extracting the intersection, where P_1 evaluates the aggregated polynomial on its input and provides proof of correct evaluation. Once the proof is validated, the parties decrypt each evaluation to get the intersection.

Our polynomial commitments will be useful in [46] for two purposes; proving the correctness of aggregation by evaluating on a public point and proving the correctness of evaluations on P_1's input finally to reveal the intersection.

We use the following primitives in our construction:

- A threshold additively homomorphic encryption scheme with protocols (Π_{GEN} and Π_{DecZero}) to respectively sample a public key together with the secret key shares, and a protocol to determine if a target ciphertext decrypts to 0. We instantiate our scheme with the BBS encryption scheme (Section 2.3) which relies on the DLIN assumption (Definition 2).
- Our polynomial commitment scheme PCOM, (that is compatible with the threshold encryption scheme), and is instantiated with non-interactive publicly verifiable proofs of evaluation of hidden points (in the batched setting) and public points (in the single instance setting). We respectively denote the committer and receiver algorithms for the corresponding (non-interactive) proof systems by $(\mathsf{PCOM.C}_{hid}^{batch}, \mathsf{PCOM.R}_{hid}^{batch})$ and $(\mathsf{PCOM.C}_{pub}, \mathsf{PCOM.R}_{pub})$. To construct PCOM, we require two commitment schemes: Pederson Commitment scheme (Section 2.4) which relies on the DL assumption and the AFG Commitment scheme (Section 2.4) that is based on bilinear pairing and relies on the DPP assumption (Definition 4).
- An n-party protocol Π_{COIN} to sample random coins.
- A simulation extractable non-interactive publicly verifiable proof system Π_{EXP} to prove knowledge of exponent. We instantiate this with the non-interactive variant of the classic protocol due to [61] via the Fiat-Shamir transform. We denote the prover and verifier algorithms by $(\mathsf{DL}.P_{pub}, \mathsf{DL}.V_{pub})$.

The protocol is split into two parts and presented in Figures 7 and 8. The first three phases of the protocol: Key Generation, Commitment Phase and Aggregation are covered in Figure 7 whereas the Intersection is contained in Figure 8.

Theorem 2. *The protocol π_{MPSI} described in Figure 7 and Figure 8 securely realizes $\mathcal{F}_{MPSI}$ (described in Figure 9) in the presence of malicious adversaries and dishonest majority under Decisional Linear (DLIN) and Double Pairing Problem (DPP) hardness assumptions.*

Protocol π_{MPSI} with Malicious Security (Part 1)

Input: Party P_i is given a set $X_i = \{x_i^1, \ldots, x_i^{m_i}\}$ of size m_i for all $i \in [n]$. All parties are given a security parameter 1^κ and a description of a group $\mathbb{G}$.
The protocol:

1. **Key Generation.** The parties mutually generate a public key PK and the corresponding secret key shares $(\mathrm{SK}_1, \ldots, \mathrm{SK}_n)$ by running π_{GEN}. P_1 also runs the setup for the polynomial commitment scheme by running $\mathsf{PCOM}.\mathit{Setup}(1^\kappa, m_{\max})$.
2. **Commitment phase.** P_1 creates commitments to its inputs $\{com_{T_1}, \ldots, \mathsf{com}_{T_n}\}$ where $\mathsf{com}_{T_i} = \mathsf{PCOM.CommitPt}(\mathsf{pp}, x_1^i, r_{T_i}, m_{\max})$ and $r_{T_i} \in \mathbb{Z}_p$ is randomly chosen and generates a proof using $\mathsf{DL}.P$ proving knowledge of the committed message and broadcasts the commitment and proof to all parties.
3. **Aggregation**
   (a) For all $i \in [2, n]$, party P_i computes the coefficients of a polynomial $A_i(\cdot) = (a_0^i, \ldots, a_{m_i}^i)$ of degree m_i, with roots set to the m_i elements of X_i. In addition, P_i chooses a random element $\lambda_i \leftarrow \mathbb{G}$ and computes the product $\lambda_i \cdot a_j^i$ for every coefficient within A_i. P_i sends P_1 the sets of ciphertexts $\mathbf{C}_i = (c_0^i, \ldots, c_{m_i}^i)$, encrypting the coefficients of $\lambda_i \cdot A_i(\cdot)$.
   (b) Upon receiving the ciphertexts from all parties, party P_1 combines the following ciphertexts
   $$c_0 = \prod_{i=2}^{n} c_0^i, \ldots, c_{m_{\max}} = \prod_{i=2}^{n} c_{m_{\max}}^i$$
   where $m_{\max} = \max(m_2, \ldots, m_n)$. Note that P_1 generates the ciphertexts by encrypting the coefficients of the combined polynomial $A(\cdot) = \lambda_2 \cdot A_2(\cdot) + \cdots + \lambda_n \cdot A_n(\cdot)$. P_1 then generates and broadcasts $\mathsf{com}_{\mathbf{C}}$ which is a commitment of the encrypted polynomial $\mathbf{C}(\cdot) = (c_0, \ldots, c_{m_{\max}})$ using $\mathsf{PCOM.Commit}(pp, \mathbf{C}, r_{\mathbf{C}})$ where $r_{\mathbf{C}}$ is generated randomly.
   (c) Next, the parties verify whether the polynomials aggregation was done correctly. Specifically, the parties first agree on a random element u from the appropriate plaintext domain using the coin tossing protocol π_{COIN} (Section 2.5). P_1 broadcasts the encrypted evaluation $\tilde{\lambda} = \mathsf{Eval}(\mathrm{PK}, \mathbf{C}, u)$ along with a proof of correct evaluation by using $\mathsf{PCOM.C}_{pub}$ on public inputs $\mathsf{pp}, \mathsf{com}_C, u, \tilde{\lambda}$ and private inputs $\mathbf{C}, r_{\mathbf{C}}$.
   (d) Then, each party broadcasts the ciphertext $\tilde{\lambda}_i = \mathsf{Eval}(\mathrm{PK}, \mathbf{C}_i, u)$, together with a ZK proof of knowledge generated using $\mathsf{DL}.P$ for proving the knowledge of the plaintext. If all the proofs are verified correctly, then the parties check that $\tilde{\lambda} - \prod_{i=2}^{n} \tilde{\lambda}_i$ encodes a 0-message using π_{DecZero}.

Fig. 7: Multi-party PSI protocol (Part 1).

Proof sketch: We split the analysis into two cases based on whether the set of corrupted parties includes the central party P_1 or not. Consider an adversary $\mathcal{A}$ that corrupts a set of parties that includes P_1. We define a simulator $\mathcal{S}$ and prove that the real and simulated executions are computationally indistinguishable. The indistinguishability between the real and simulated execution is reduced to the privacy property of the encryption scheme, the hiding property of the

Protocol π_{MPSI} with Malicious Security (Part 2)

The protocol (continued):

4. **Intersection.**
   (a) If the above verification is completed correctly, P_1 evaluates the aggregated polynomial that is encrypted within ciphertexts $\mathbf{C} = \big(c_1, \ldots, c_{m_{\max}}\big)$, on its input elements $\{x_1^j\}_{j=1}^{m_1}$, and proves consistency with the commitment $\mathsf{com}_{\mathbf{C}}$. P_1 forwards the encrypted evaluations $c_y = \mathsf{Eval}(PK, \mathbf{C}, t)$ along with a proof generated using $\mathsf{PCOM.C}_{hid}^{batch}$ on public inputs $\mathsf{pp}, \mathsf{com}_{\mathbf{C}}, \{\mathsf{com}_{T_i}\}_{i\in[m]}, \{c_{y_i}\}_{i\in[m_1]}$ and private inputs $\mathbf{C}, r_{\mathbf{C}}, X_1, \{r_{T_i}\}_{i\in[m_1]}$
   (b) All parties verify the evaluations and then decrypt the evaluations using protocol π_{DecZero} to reveal the intersection.

Fig. 8: Multi-party PSI protocol (Part 2).

Functionality $\mathcal{F}_{MPSI}$

$\mathcal{F}_{MPSI}$ communicates with parties $P_1, \ldots, P_n$ with input sets $X_1, \ldots, X_n$ and an adversary $\mathcal{A}$ controlling a subset of parties.

1. Upon receiving a message $(\mathrm{input}, P_i, X_i)$ from party P_i, store the set X_i. Once all inputs $i \in [n]$ are received, set $X = \cap_{i=1}^{n} X_i$ and send (input) to $\mathcal{A}$.
2. Upon receiving $(\mathrm{deliver})$ from $\mathcal{A}$, output (output, X) to P_1. If received (abort) from $\mathcal{A}$, output $\perp$ to P_1.

Fig. 9: Multi-party PSI Functionality.

commitment schemes, and the privacy property of the polynomial commitment. In the first case, the central party P_1 is corrupted, and the input of P_1 can be extracted from P_1's input commitment in the commit phase. The input of other corrupted parties can be extracted by rewinding the aggregation phase. This is achieved by extracting $d+1$ evaluation points of every corrupted party's polynomial as shown in [46]. In the second case, the simulation is the same as the previous case with the exception that it does not need to extract P_1's input.

The complete proof is provided in Section A.2.

Complexity. The communication complexity of our protocol is linear in the input sizes and the number of parties, where the smallest input size can be given to P_1. Naively, the communication complexity of our protocol is $O(n^2 + \sum_{i=1}^{n} m_i + n \cdot m_{\min} \cdot \log m_{\max})$ when the polynomial commitment is separately used for each evaluation point. The batching feature of our scheme reduces the communication cost of our protocol to $O(n^2 + \sum_{i=1}^{n} m_i + n \cdot (m_{\min} + \log m_{\max}))$. For the central party P_1, the communication cost is $O(n(m_{\min} + \log m_{\max})$. P_1 generates a batched evalution proof of size $O(m_{\min} + \log m)$. The dominating cost for P_1 is sending the evaluation proof to all other parties. For all other parties, the

	P_1	P_i	Total
KeyGen	$O(n)$	$O(n)$	$O(n^2)$
Commit	$O(n \cdot m_{\min})$	—	$O(n \cdot m_{\min})$
Aggregate	$O(n \cdot \log m_{\max})$	$O(m_i + n)$	$O(n^2 + \sum_{i=2}^{n} m_i + n \cdot \log m_{\max})$
Intersection	$O(n \cdot (m_{\min} + \log m_{\max}))$	$O(m_{\min})$	$O(n \cdot (m_{\min} + \log m_{\max}))$
MPSI	$O(n \cdot (m_{\min} + \log m_{\max}))$	$O(n + m_{\min} + m_i)$	$O(n^2 + \sum_{i=1}^{n} m_i + n \cdot (m_{\min} + \log m_{\max}))$

Table 3: MPSI Communication Complexity.

communication cost is $O(n + m_{\min} + m_i)$ where $O(n)$ is sent during the Key Generation phase as well as verifying the aggregation. Additionally, the communication cost in sending the encrypted polynomial to P_1 and generating the intersection is $O(m_i)$ and $O(m_{\min})$ respectively. We provide a detailed analysis in Table 3, providing the communication complexity of the parties individually as well as together along every phase of the MPSI protocol. The round complexity of our protocol is dominated by the round complexity of the underlying polynomial commitments. In the random oracle model, the round complexity is 4.

Computationally, the dominating part of the protocol is evaluating the aggregated polynomial and executing the private polynomial commitment from Section 3. The complexity of our protocol is $O(m_{\max} \cdot m_{\min})$ exponentiations. We further reduce the polynomial degrees and the overall workload using hashing techniques; see below for more details. The space complexity of our protocol in the interactive setting is $O(m_{\max})$ for P_1 and $O(m_i)$ for every other party P_i, while in the non-interactive setting the complexity is $O(m_{\max})$ for P_1 and $O(m_i + \log m_{\max})$ for party P_i. We note that the space complexity of P_1 is independent of the number of parties. In particular, the polynomials received by the parties can be aggregated on-the-fly and do not require any extra space. Regarding the polynomial commitments, the non-interactive variant requires P_i to store the entire proof in the memory which increases the space complexity by an additive factor of $O(\log m_{\max})$.

Hashing. A notable optimization in PSI protocols is using simple hashing to map the input into smaller sets (buckets) and running a different instance per bucket. In our context, this enables us to reduce the workload of P_1 from quadratic to quasilinear. The idea behind simple hashing lies in splitting the input set into bins where based on a hash function, each element is assigned to a bin. Next, the parties sort their input into bins and run an MPSI protocol separately on each bin. Splitting the input into bins reduces the size of the degree of the polynomials and improves the computation cost of the parties for the computationally heavy tasks of polynomials interpolations and evaluations.

Simple hashing can be directly used in the malicious setting where each bin induces a separated polynomial. Note that the adversary can only attempt to put an item in the wrong bin but this item can be ignored by the simulator. Let h be a hash function, $m_{\max}$ be the maximum number of items in an input set, $\mathcal{B}$ be the number of bins and M is the maximum of items in a bin. It is known that if a hash function maps $m_{\max}$ items into $\mathcal{B}$ bins and $m_{\max} \geq \mathcal{B} \log \mathcal{B}$ then with very high probability, $M = \frac{m_{\max}}{\mathcal{B}} + \sqrt{\frac{m_{\max} \log \mathcal{B}}{\mathcal{B}}}$ [59, 67]. Setting $\mathcal{B} = \frac{m_{\max} \log \log m_{\max}}{\log m_{\max}}$ and applying the Chernoff bound implies that $M = O(\frac{\log m_{\max}}{\log \log m_{\max}})$ with negligible error in $m_{\max}$. Simple hashing can be used to reduce the number of exponentiations, thereby reducing the computational cost. Namely, for each bin, the number of required exponentiations is $O(M^2)$ and the overall number of exponentiations will be $O(\mathcal{B}M^2)$. Substituting the values of $\mathcal{B}$ and M using the above analysis will result in $O(m_{\max} \frac{\log m}{\log \log m})$ exponentiations. We refer to Section 6 for more details regarding the concrete improvement.

The hashing techniques are not useful for improving [9] as they cannot be broken into small instances. While the improvement for [36] will potentially be smaller since its computational complexity is quasilinear in the input size.

5 Other Applications

In this section, we consider a list of distributed tasks in different settings, whose realization can make use of private polynomial commitments. All applications can benefit from the batching of our scheme while achieving malicious security.

5.1 Oblivious Polynomial Evaluation

Following the discussion from Section 1, in this work, we consider a distributed variant of the oblivious polynomial evaluation functionality denoted by DOPE, where the polynomial $Q_i(\cdot)$ is linearly shared amongst a set of $n-1$ parties. More formally, we define the DOPE functionality as follows. The input of party P_i for $i \in [2, n]$ is a polynomial $Q_i(\cdot)$ of degree at most d whereas the input of P_1 is an element t, and the goal is that P_1 learns $\sum_{i \in [2,n]} Q_i(t)$.

	P_1 comm	P_i comm	Total comm	P_1 comp	P_i comp
[45]	$O(n(d\kappa) + n\lambda)$	$O(d\lambda\kappa)$	$O((n+\lambda)d\kappa)$	$O(nd\lambda)$	$O(d\lambda)$
[43]	$O(n\kappa \log d)$	$O(d\kappa)$	$O(nd\kappa)$	$O(nd)$	$O(d)$
Our Work	$O(n\kappa \log d)$	$O(d\kappa)$	$O(nd\kappa)$	$O(d)$	$O(d)$

Table 4: Comparison between different DOPE protocols where comm refers to the communication complexity and comp refers to the computational complexity (stated as the number of exponentiations), κ is the computational security parameter, λ is the statistical security parameter, n is the number of parties and d is the degree of the polynomial.

We can realize our DOPE functionality (in Figure 10)in the presence of $n-1$ malicious corruptions based on our polynomial commitment scheme following

Functionality $\mathcal{F}_{DOPE}$

$\mathcal{F}_{DOPE}$ communicates with parties $P_2, \ldots, P_n$ with input polynomial $Q_2(\cdot), \ldots, Q_n(\cdot)$, party P_1 with input point t and an adversary $\mathcal{A}$ controlling the subset of parties.

1. Upon receiving a message (input, P_i, Q_i) from party P_i for $i \in [2, n]$, store the polynomial Q_i and send a message (input) to $\mathcal{A}$.
2. Upon receiving a message (input, P_1, t) from party P_1, store the value t and send a message (input) to $\mathcal{A}$.
3. Upon receiving a message (deliver) from $\mathcal{A}$, set $x = \sum_{i=2}^{n} Q_i(t)$ and output (output, x) to P_1. If received the message (abort) from $\mathcal{A}$, output $\perp$ to P_1.

Fig. 10: DOPE Functionality.

the blueprint of our PSI protocol. Namely, the parties send their encrypted coefficients to P_1 that aggregates the ciphertexts and evaluates $Q(\cdot)$ on its input t. P_1 further attaches proofs of correct aggregation and evaluation. Finally, the parties run a distributed decryption protocol for P_1 to learn $Q(t)$. Note that, while in PSI the inputs of the parties are extracted from the polynomials' roots, where the inputs are the polynomial's shares that form $Q(\cdot)$.

Our scheme is further flexible regarding the level of threshold introduced by the underlying secret sharing scheme. In particular, one may use any threshold linear secret sharing for splitting the polynomial into shares (rather than simple additive sharing), where the threshold parameter can be smaller than $n-1$. We also have a simple aggregation mechanism which allows the DOPE to be reduced to a single OPE execution where $n-1$ parties play the role of P_2.

Two prior OPE constructions with malicious security [45, 43] can be extended to the distributed setting, where each party P_i for $i \in [2, n]$ carries out an individual OPE with P_1. Compared to previous work; see Table 4, our construction achieves better computational complexity for the central party P_1 due to the fact that the aggregation mechanism allows P_1 to combine the polynomials cheaply and then run the protocol with almost the same cost as running a two-party instance of OPE. The overall communication complexity of our protocol is similar to [43] and is better than [45].

Finally, we note that we can further extend our protocol to support multivariate polynomials to cover a broader class of functionalities.

5.2 Verifiable Polynomial Evaluations

In this setting, we focus on verifying the evaluations of a polynomial $Q(\cdot)$, linearly shared across a set of $n-1$ clients, that are aggregated and stored by a cloud server. Specifically, a set of clients outsource their shares of a d-degree polynomial (potentially in the clear), to an untrusted server while storing a short state. The server stores the aggregated polynomials and prepares proof for this computation. Next, whenever the clients provide an input x, the server computes $Q(x)$ and a short proof that allows the clients to verify this computation in sub-linear time in d. We require the verification process to be *public*. Finally, the client's output $Q(x)$.

Employing our polynomial commitment by the server, the clients can non-interactively verify the proofs it provides. Furthermore, our solution supports the feature that the polynomial may also be kept private since the shares can be stored on the server while encrypted, where only the evaluation points are public. In more detail, each party P_i sends the server its polynomial share $Q_i(\cdot)$. The server aggregates the shares and computes a proof of correct aggregation (that can be made non-interactive by using the random oracle to choose the random evaluated point for this test). Upon receiving an input x, the parties forward it to the server that computes (the encryption of) $Q(x)$ together with a proof of correctness. Our protocol is secure in the presence of $n-2$ corrupted clients, and a colluding server. Note that the degree of $Q(\cdot)$ may be huge, yet uploading it is a one-time phase whose complexity amortizes away over multiple evaluation points. Moreover, the proofs of correct evaluations can be batched.

Related modelling is multi-clients verifiable computation where a set of clients wish to compute some function f on their joint inputs while non-interactively communicating only with the server over a sequence of evaluations [25, 42, 11]. Such constructions have only been demonstrated in a setting where the clients and the server do not collude [42]. Our protocol achieves full security but requires an additional round of communication at the end due to decryption.

Verifiable polynomial evaluations on encrypted data. The second application in this area is verifiable computation on encrypted data. The notion was proposed by Gennaro et al. in [37] and follow-up works [39, 30, 31, 12] proposed constructions for computations such as linear functions and polynomial evaluations. These schemes provide both privacy of the outsourced data to the untrusted server and the integrity of the results computed by the server. However, these constructions rely on fully or somewhat homomorphic encryptions based on lattice and zero-knowledge proofs over polynomial rings, thus their overhead is high and they have not been realized in practice. Also, these protocols cannot be directly extended to multi-clients.

Our scheme yields a more efficient verifiable computation on encrypted data for polynomial evaluations. The prover's computation only involves operations on bilinear maps, making it one step closer to being practical. In the amortized setting, the verifier's time is faster than evaluating the polynomial locally for multiple evaluations. In particular, to compute m evaluations on a degree-d polynomial, the proof size is $O(m + \log d)$ and the verifier's time is $O(d + m)$.

Our model requires a setup phase for the clients prior to communicating with the server. This setup phase is independent of the input and is only carried out once, regardless of the number of polynomial evaluations computed later. The clients store a short state upon concluding this phase, which is later used to extract $Q(x)$. In our protocol, the parties run the key generation protocol for the underlying threshold encryption scheme, store the secret key share, and use it to partially decrypt the ciphertext returned from the server.

5.3 Non-interactive Two-party PSI (NISI)

Ishai et al. [47] introduced the Non-interactive Secure computation (NISC) model where a Receiver first posts an "encryption" of its input publicly and then a Sender can compute a function over the encrypted input along with its input and obtain an "encryption" of the output that the Receiver can decrypt. The classic Yao's garbled circuit-based two-party protocol in the semi-honest setting when combined with a 2-round OT is an example of such a protocol. Several works have explored the feasibility and concrete efficiency of such protocols in the malicious Boolean setting [47, 5, 51, 44, 3]. Private polynomial commitments can be used directly to implement a non-interactive secure private set-intersection protocol by relying on a variant of the [33] protocol. Such a scheme will additionally have the feature of reusability where the receiver only needs to post its encrypted input once and any number of senders can transmit the result of the set intersection to the receiver. An important application of reusable NISI is applicable is contact discovery in messaging services such as Signal and Telegram.

Concretely to PSI in the malicious setting, Cristofaro et al. [26] design a two-round PSI protocol with linear communication complexity. More recently, the work by Rosulek and Trieu [60] showed how to obtain a 2-round PSI by relying on a variant of the Diffie-Hellman Key Agreement and an ideal permutation oracle. This work has highly competitive communication and computation costs for small set sizes (between 2^7 and 2^{16} elements). We provide a comparison of the communication costs in Table 5. We can see that our work is competitive in communication because the proofs are succinct in the batch setting. Additionally, we rely on more standard assumptions. Even though our computation costs are higher our protocol could be useful in a client-server setting where the receiver is a lightweight client device and the sender is the server with significantly bigger computational resources. We further point out that the reported computational costs could be improved by further parallelizing our implementation. We leave this as future work to explore.

6 Implementation

We implemented our encrypted polynomial commitment scheme and the multi-party PSI scheme, and we present the experimental results in this section.

n	2^8	2^{16}	2^{20}
[26]	62.74 (KB)	13.33 (MB)	213 (MB)
[60]	16.38 (KB)	4.19 (MB)	67.11 (MB)
Here (est.)	49.7 (KB)	5.86 (MB)	68 (MB)

Table 5: Communication cost of two-party PSI with set size m.

Software and hardware. The system is implemented in C++. We use the ate-pairing library [1] for bilinear maps and the GMP library [2] for field arithmetic. Our experiments are executed on a BN-curve over a 254-bit prime, which offers 128-bit of security. There are 3200 lines of code for the encrypted polynomial commitment and 1000 lines for the other building blocks in the MPSI protocol. We ran all experiments on an AWS c5.9xlarge instance with an Intel Xeon Platinum 8000 processor and 72GB of RAM. We report the average running time over 5 executions, except for the largest instances due to the long running time.

6.1 Private Polynomial Commitments

Single evaluation. We first present the performance of our encrypted polynomial commitment scheme as a stand-alone primitive. Figure 11 shows the prover and verifier times (left y-axis) and proof size (right $y - axis$) of one evaluation of the variant with committed points (Section 3.2). We vary the degree of the polynomial from 2^4 to 2^{16}. As shown in the figure, the prover time grows linearly with the polynomial degree. It takes 11s to generate the proof for $d = 2^{10}$ and 701s to generate the proof for $d = 2^{16}$. The verifier time also grows linearly with the degree, as it has to update the commitment key together with the prover in our scheme. It takes 0.93s to verify the proof for $d = 2^{10}$ and 53.7s for $d = 2^{16}$, which roughly matches the time on reducing the commitment key in the prover's time. The proof size is only logarithmic on the degree of the polynomial and is very small in practice. It is 11.9KB for $d = 2^{10}$ and 18.6KB for $d = 2^{16}$.

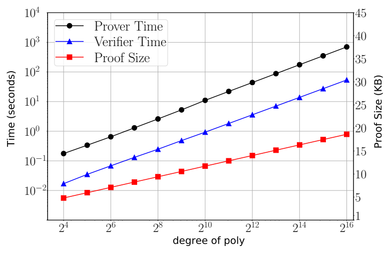

Fig. 11: Performance of single evaluation of our encrypted polynomial commitment with point hiding.

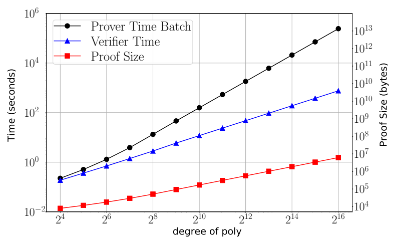

Fig. 12: Performance of multiple evaluations of our encrypted polynomial commitment with point hiding. $m = d$.

Multiple evaluations. The major advantage of our scheme is the batched proofs for multiple evaluations and we further present the performance of evaluating multiple points in Figure 12. In the figure, we set the number of evaluations the same as the degree of the polynomial, but our implementation supports

# of elements m	2^8	2^{10}	2^{12}	2^{14}	2^{16}
Size of bin M	2^8	2^6	2^6	2^6	2^6
# of bins $\mathcal{B}$	1	81	334	1,366	5,487
$n = 2$	13.94	130.01	536.1	2,192	8,264
$n = 8$	13.96	130.1	536.66	2,194	8,270
$n = 32$	13.97	130.4	538.4	2,199	8,292
$n = 128$	14.02	131.7	545.56	2,220	8,376
$n = 500$	14.26	136.4	562.76	2,301	8,712
$n = 1000$	14.58	142.9	589.5	2,410	9,141

Table 6: Total running time of our multiparty PSI scheme in seconds.

both a larger degree and a larger number of evaluations. As shown in the figure, the prover time grows quadratically. It takes 0.225s to generate a proof for $m = d = 2^4$ and 242,395s for $m = d = 2^{16}$.

The proof size and the verifier time are particularly good for multiple evaluations. The proof size is only 7.9KB for $m = d = 2^4$ evaluations and 6.1MB for $m = d = 2^{16}$ evaluations, which is significantly smaller than repeating the single evaluation protocol the same number of times. The experimental result matches the logarithmic complexity in d and the linear complexity in m.

The verifier time only grows quasi-linearly now. It only takes 757s to verify 2^{16} proofs of evaluations of a degree-2^{16} polynomial, which is merely $14\times$ larger than verifying a single proof. The experimental result justifies that the verifier time is amortized to $O(\log d)$ for multiple evaluations and is particularly efficient in our application of multiparty PSI.

6.2 Performance of Multi-Party PSI

In this section, we report the performance of our multiparty PSI protocol with malicious security. We executed all parties on the single AWS instance and we simulated a network connection using the Linux `tc` command, communicating via a localhost network. We simulated a LAN setting with 10 Gbps network bandwidth. We executed P_2 to P_n on the same machine but only count the running time of one of them in the total time. This is to better simulate the scheme in practice where all the parties can run the computation simultaneously.

We tested our MPSI protocol for 2–1000 parties and 2^8–2^{16} elements per party (here we set $m_{\max} = m_{\min}$) and the total running time are shown in Table 6. We applied the hashing technique described in Section 4 and the parameters achieving 40-bit of statistical security are included in the table.

As shown in the table, our protocol is slow for a small number of parties where it takes 13.94s to compute a two-party intersection with 2^8 elements per party. This is $55\times$ slower than the malicious MPSI scheme based on symmetric key primitives from [9, Table 5]. The gap is even larger on larger sets, which is expected as our protocol relies on public-key primitives. However, our running time hardly grew with the number of parties where it still takes 14.02s for 128 parties with 2^8 elements each, and 14.58s for 1000 parties. This is because most of the running time is due to evaluating the aggregated polynomial and generating

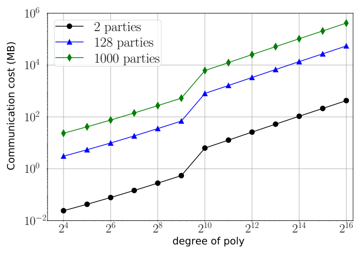

Fig. 13: Communication of our multiparty PSI protocol.

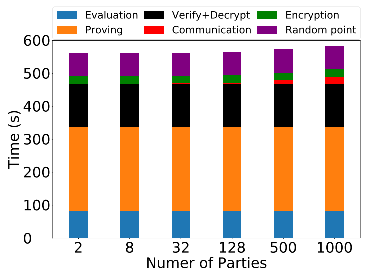

Fig. 14: Breakdown of the running time in our multiparty PSI protocol. $m = 2^{12}$ elements per party.

the proofs using our commitment scheme, which only depends on the maximum size of the set $m_{\max}$ and the size of P_1's set $m_{\min}$. In contrast, the running time of PSimple [9] grows linearly with the number of parties and is 0.8s for 32 parties with 2^8 elements each, which is 17× faster than ours. We expect that our protocol is faster than PSimple for 500 parties with 2^8 elements per party.

Our protocol is also efficient in communication. The total communication is shown in Figure 13. As shown in the figure, the communication size for 2 parties with 2^8 elements per party is 279 KB, whereas the total communication for 1000 parties with 2^8 elements per party is 278MB, which is not the bottleneck of our protocol. Compared with [9], the communication size is 7.5MB for 2 parties and 7.5GB for 1000 parties respectively, which is around 27× larger than ours. The jump in Figure 13 for $m = 2^{10}$ is due to using the hashing technique for $m \geq 2^{10}$.

We further show the breakdown of our total running time in Figure 14. We fix the size of the set per party at 2^{12} and vary the number of parties from 2 to 1000. As shown in the figure, our protocol is clearly computation-heavy and most of the time is on the evaluations of the aggregated polynomial, the proof generation and the verification of our private polynomial commitment. Even with 1000 parties, they contribute to 97.5% of the total running time. Due of this observation, we could improve the total running time significantly through parallelization. Both the polynomial evaluations and the private polynomial commitment are trivially parallelizable. Moreover, the total running time of our scheme is not sensitive to the bandwidth of the network. On a WAN network with 100Mbps bandwidth, our scheme would become around two times slower for 1000 parties. By contrast, the performance of symmetric-key-based schemes such as PSimple is limited by the communication overhead. It cannot be improved through parallelization and will become worse on a network with lower bandwidth.

Finally, another major advantage of our protocol is memory usage and scalability. As the memory usage of P_1 is only $O(m_{\max})$, we are able to scale up to 1000 parties and 2^{16} elements per party. The memory usage of P_1 on this largest instance is only 1GB. We did not test more elements per party due to the long running time, but not have high memory usage. To compare, the PSimple scheme [9] runs out of memory for 12 parties and 2^{20} elements per party.

This is because P_1 has to store random OTs for the garbled bloom filter with each party, which leads to a high overhead on the memory.

Overall, the experimental results show that our scheme has good scalability and communication in practice, and is particularly efficient for applications with a large number of parties or limited bandwidth networks.

7 Acknowledgements

We thank the anonymous PKC'23 reviewers for their helpful comments. The first and second authors are supported by ISF grant No. 1316/18. The second, third and fifth authors are supported by DARPA under Contract No. HR001120C0087. The third author was supported by Technology and Humanity Fund from Georgetown University's McCourt School of Public Policy. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA. Distribution Statement A. Approved for public release. Distribution Unlimited.

References

1. Ate pairing. `https://github.com/herumi/ate-pairing`
2. The GNU multiple precision arithmetic library. `https://gmplib.org/`
3. Abascal, J., Sereshgi, M.H.F., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Is the classical GMW paradigm practical? the case of non-interactive actively secure 2pc. In: CCS. pp. 1591–1605 (2020)
4. Abe, M., Fuchsbauer, G., Groth, J., Haralambiev, K., Ohkubo, M.: Structure-preserving signatures and commitments to group elements. J. Cryptol. pp. 363–421 (2016)
5. Afshar, A., Mohassel, P., Pinkas, B., Riva, B.: Non-interactive secure computation based on cut-and-choose. In: EUROCRYPT. pp. 387–404 (2014)
6. Backes, M., Datta, A., Kate, A.: Asynchronous computational VSS with reduced communication complexity. In: CT-RSA. vol. 7779, pp. 259–276 (2013)
7. Backes, M., Fiore, D., Reischuk, R.M.: Verifiable delegation of computation on outsourced data. In: CCS. pp. 863–874 (2013)
8. Bayer, S., Groth, J.: Zero-knowledge argument for polynomial evaluation with application to blacklists. In: EUROCRYPT. pp. 646–663 (2013)
9. Ben-Efraim, A., Nissenbaum, O., Omri, E., Paskin-Cherniavsky, A.: Psimple: Practical multiparty maliciously-secure private set intersection. In: ASIA CCS. pp. 1098–1112 (2022)
10. Benabbas, S., Gennaro, R., Vahlis, Y.: Verifiable delegation of computation over large datasets. In: CRYPTO. pp. 111–131 (2011)
11. Bhadauria, R., Hazay, C.: Multi-clients verifiable computation via conditional disclosure of secrets. In: SCN. pp. 150–171 (2020)
12. Bois, A., Cascudo, I., Fiore, D., Kim, D.: Flexible and efficient verifiable computation on encrypted data. In: Garay, J.A. (ed.) Public-Key Cryptography – PKC 2021 (2021)
13. Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In: CRYPTO. pp. 41–55 (2004)
14. Bootle, J., Cerulli, A., Chaidos, P., Groth, J., Petit, C.: Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In: EUROCRYPT. pp. 327–357 (2016)
15. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: IEEE S&P. pp. 315–334 (2018)

16. Bünz, B., Fisch, B., Szepieniec, A.: Transparent snarks from dark compilers. In: EUROCRYPT. pp. 677–706 (2020)
17. Bünz, B., Maller, M., Mishra, P., Tyagi, N., Vesely, P.: Proofs for inner pairing products and applications. In: ASIACRYPT. pp. 65–97 (2021)
18. Camenisch, J., Dubovitskaya, M., Haralambiev, K., Kohlweiss, M.: Composable and modular anonymous credentials: Definitions and practical constructions. In: ASIACRYPT. vol. 9453, pp. 262–288 (2015)
19. Catalano, D., Fiore, D.: Vector commitments and their applications. In: PKC. vol. 7778, pp. 55–72 (2013)
20. Catalano, D., Fiore, D., Gennaro, R., Vamvourellis, K.: Algebraic (trapdoor) one-way functions and their applications. In: TCC. pp. 680–699 (2013)
21. Catalano, D., Fiore, D., Warinschi, B.: Homomorphic signatures with efficient verification for polynomial functions. In: CRYPTO. pp. 371–389 (2014)
22. Chase, M., Miao, P.: Private set intersection in the internet setting from lightweight oblivious PRF. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO. pp. 34–63 (2020)
23. Chepurnoy, A., Papamanthou, C., Zhang, Y.: Edrax: A cryptocurrency with stateless transaction validation. IACR Cryptol. ePrint Arch. p. 968 (2018)
24. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.: Marlin: Preprocessing zksnarks with universal and updatable srs. In: EUROCRYPT. pp. 738–768 (2020)
25. Choi, S.G., Katz, J., Kumaresan, R., Cid, C.: Multi-client non-interactive verifiable computation. In: TCC. pp. 499–518 (2013)
26. Cristofaro, E.D., Kim, J., Tsudik, G.: Linear-complexity private set intersection protocols secure in malicious model. In: Abe, M. (ed.) ASIACRYPT. pp. 213–231 (2010)
27. Fenske, E., Mani, A., Johnson, A., Sherr, M.: Distributed measurement with private set-union cardinality. In: CCS. pp. 2295–2312 (2017)
28. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: CRYPTO. pp. 186–194 (1986)
29. Fiore, D., Gennaro, R.: Publicly verifiable delegation of large polynomials and matrix computations, with applications. In: CCS. pp. 501–512 (2012)
30. Fiore, D., Gennaro, R., Pastro, V.: Efficiently encrypted data. In: ACM SIGSAC. pp. 844–855 (2014)
31. Fiore, D., Nitulescu, A., Pointcheval, D.: Boosting verifiable computation on encrypted data. In: PKC (2020)
32. Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O.: Keyword search and oblivious pseudorandom functions. In: Kilian, J. (ed.) TCC. pp. 303–324 (2005)
33. Freedman, M.J., Nissim, K., Pinkas, B.: Efficient private matching and set intersection. In: EUROCRYPT. pp. 1–19 (2004)
34. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. IACR Cryptol. ePrint Arch. **2019**, 953 (2019)
35. Gamal, T.E.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. Inf. Theory pp. 469–472 (1985)
36. Garimella, G., Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: Oblivious key-value stores and amplification for private set intersection. In: CRYPTO. pp. 395–425 (2021)
37. Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In: CRYPTO

38. Ghosh, S., Nielsen, J.B., Nilges, T.: Maliciously secure oblivious linear function evaluation with constant overhead. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT. pp. 629–659 (2017)
39. Goldwasser, S., Kalai, Y.T., Popa, R.A., Vaikuntanathan, V., Zeldovich, N.: How to run turing machines on encrypted data. In: CRYPTO. pp. 536–553. Springer (2013)
40. Gorbunov, S., Reyzin, L., Wee, H., Zhang, Z.: Pointproofs: Aggregating proofs for multiple vector commitments. In: ACM SIGSAC. pp. 2007–2023 (2020)
41. Gordon, S.D., Hazay, C., Le, P.H.: Fully secure PSI via mpc-in-the-head. PoPETS **2022**(3), 291–313 (2022)
42. Gordon, S.D., Katz, J., Liu, F., Shi, E., Zhou, H.: Multi-client verifiable computation with stronger security guarantees. In: TCC. pp. 144–168 (2015)
43. Hazay, C.: Oblivious polynomial evaluation and secure set-intersection from algebraic prfs. In: TCC. pp. 90–120 (2015)
44. Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Actively secure garbled circuits with constant communication overhead in the plain model. In: TCC. pp. 3–39 (2017)
45. Hazay, C., Lindell, Y.: Efficient oblivious polynomial evaluation with simulation-based security. IACR Cryptol. ePrint Arch. p. 459 (2009)
46. Hazay, C., Venkitasubramaniam, M.: Scalable multi-party private set-intersection. In: PKC. pp. 175–203 (2017)
47. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Prabhakaran, M., Sahai, A.: Efficient non-interactive secure computation. In: EUROCRYPT. pp. 406–425 (2011)
48. Juels, A., Jr., B.S.K.: Pors: proofs of retrievability for large files. In: CCS. pp. 584–597 (2007)
49. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Abe, M. (ed.) ASIACRYPT. pp. 177–194 (2010)
50. Lee, J.: Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. IACR Cryptol. ePrint Arch. **2020**, 1274 (2020)
51. Mohassel, P., Rosulek, M.: Non-interactive secure 2pc in the offline/online and batch settings. In: EUROCRYPT. pp. 425–455 (2017)
52. Naor, M., Pinkas, B.: Oblivious polynomial evaluation. SIAM J. Comput. pp. 1254–1281 (2006)
53. Nguyen, D.T., Trieu, N.: Mpccache: Privacy-preserving multi-party cooperative cache sharing at the edge. IACR Cryptol. ePrint Arch. (2021), `https://eprint.iacr.org/2021/317`
54. Papamanthou, C., Shi, E., Tamassia, R.: Signatures of correct computation. In: TCC. pp. 222–242. Springer (2013)
55. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: CRYPTO. pp. 129–140 (1991)
56. Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: Spot-light: Lightweight private set intersection from sparse OT extension. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO. pp. 401–431 (2019)
57. Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: PSI from paxos: Fast, malicious private set intersection. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT. pp. 739–767 (2020)
58. Pinkas, B., Schneider, T., Tkachenko, O., Yanai, A.: Efficient circuit-based PSI with linear communication. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT. pp. 122–153. Springer (2019)
59. Raab, M., Steger, A.: "balls into bins" - A simple and tight analysis. In: Randomization and Approximation Techniques in Computer Science. pp. 159–170 (1998)

60. Rosulek, M., Trieu, N.: Compact and malicious private set intersection for small sets. IACR Cryptol. ePrint Arch. p. 1159 (2021)
61. Schnorr, C.: Efficient signature generation by smart cards. J. Cryptol. pp. 161–174 (1991)
62. Setty, S.T.V.: Spartan: Efficient and general-purpose zksnarks without trusted setup. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO. pp. 704–737 (2020)
63. Tomescu, A., Chen, R., Zheng, Y., Abraham, I., Pinkas, B., Gueta, G.G., Devadas, S.: Towards scalable threshold cryptosystems. In: IEEE S&P. pp. 877–893 (2020)
64. Vlasov, A., Panarin, K.: Transparent polynomial commitment scheme with polylogarithmic communication complexity. IACR Cryptol. ePrint Arch. **2019**, 1020 (2019)
65. Wahby, R.S., Tzialla, I., Shelat, A., Thaler, J., Walfish, M.: Doubly-efficient zkSNARKs without trusted setup. In: IEEE S&P. pp. 926–943 (2018)
66. Wails, R., Johnson, A., Starin, D., Yerukhimovich, A., Gordon, S.D.: Stormy: Statistics in tor by measuring securely. In: CCS. pp. 615–632 (2019)
67. Wieder, U.: Balanced allocations with heterogenous bins. In: SPAA. pp. 188–193 (2007)
68. Xie, T., Zhang, J., Zhang, Y., Papamanthou, C., Song, D.: Libra: Succinct zero-knowledge proofs with optimal prover computation. In: CRYPTO (2019)
69. Yuan, J., Yu, S.: Proofs of retrievability with public verifiability and constant communication cost in cloud. In: SCC@ASIACCS. pp. 19–26. ACM (2013)
70. Zhang, J., Xie, T., Zhang, Y., Song, D.: Transparent polynomial delegation and its applications to zero knowledge proof. In: IEEE S&P (2020)
71. Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: vsql: Verifying arbitrary SQL queries over dynamic outsourced databases. In: IEEE S&P. pp. 863–880 (2017)
72. Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: A zero-knowledge version of vsql. IACR Cryptol. ePrint Arch. **2017**, 1146 (2017)

Supplementary Material

A Full proofs

A.1 Proof of Theorem 1

Claim. Protocol PCOM (Figures 3) satisfies completeness.

To argue completeness of PCOM, it suffices to prove that R accepts in each of the three tests: (1) Private Inner Product Argument (2) Linear Constraints Test, and (3) Testing Quadratic Test. We argue this for each test below:

Private Inner Product Argument.

This test (Figure 4) has two phases, the Masking Phase (Step 1) and the Inner Product Phase (Step 2). At the end of the Product Phase, R accepts if $\mathsf{com} = \mathsf{com}_{\mathbf{C}'} \cdot e(\mathsf{com}_{F'}, a) \cdot e(c'_y, b)$ and C opens the com to $\mathbf{C}', F'$ and c'_y in the last round. We will argue that $\mathsf{com}' = \mathsf{com}_{\mathbf{C}'} \cdot e(\mathsf{com}_{F'}, a) \cdot e(c'_y, b)$ is where $\mathsf{com}_{\mathbf{C}'}$ and com'_F are commitments to $\mathbf{C}'$ and F' respectively and $c'_y = \langle \mathbf{C}', F' \rangle$. We show that the invariant is satisfied after every round. First, we will argue that the invariant holds at the end of the masking phase. The inner product phase proceeds in rounds and we will argue the invariant holds at the end of each iteration and this will conclude the proof of completeness of the inner product phase.

For the Masking Phase, $\mathbf{E}$ and M are the masking encrypted polynomial and vector respectively. To argue that the invariant holds, we show the com' generated in Step 1(c) is a combined commitment of $\mathbf{C}' = \mathbf{C} \odot \mathbf{E}^x$, $F' = F + x^{-1} \cdot M$ and c'_y as the inner product of the two. From Step 1(c) we have that com' is :

$$\begin{aligned}
&\mathsf{com} \cdot \mathsf{com}_{\mathbf{E}}^{x} \cdot e(\mathsf{com}_M, a)^{x^{-1}} \cdot e(c_l^x \cdot c_m \cdot c_r^{x^{-1}}, b) \\
&= \big(\mathsf{com}_{\mathbf{C}} \cdot e(\mathsf{com}_B, a) \cdot e(c_y, b)\big) \cdot \big(\mathsf{com}_{\mathbf{E}}^{x} \cdot e(\mathsf{com}_M, a)^{x^{-1}}\big) \\
&\qquad \cdot e(c_l^x \cdot c_m \cdot c_r^{x^{-1}}, b) \\
&= \big(\mathsf{com}_{\mathbf{C}} \cdot \mathsf{com}_{\mathbf{E}}^{x}\big) \cdot \big(e(\mathsf{com}_B, a) \cdot e(\mathsf{com}_M, a)^{x^{-1}}\big) \\
&\qquad \cdot \big(e(c_y, b) \cdot e(c_l^x \cdot c_m \cdot c_r^{x^{-1}}, b)\big) \\
&= \big(\mathsf{com}_{\mathbf{C}} \cdot \mathsf{com}_{\mathbf{E}}^{x}\big) \cdot \big(e(\mathsf{com}_B, a) \cdot e(\mathsf{com}_M^{x^{-1}}, a)\big) \\
&\qquad \cdot \big(e(c_y, b) \cdot e(c_l^x \cdot c_m \cdot c_r^{x^{-1}}, b)\big) \\
&= \big(\mathsf{com}_{\mathbf{C}} \cdot \mathsf{com}_{\mathbf{E}}^{x}\big) \cdot e(\mathsf{com}_B \cdot \mathsf{com}_M^{x^{-1}}, a) \\
&\qquad \cdot e(c_y \cdot c_l^x \cdot c_m \cdot c_r^{x^{-1}}, b) \\
&= P_1 \cdot e(P_2, a) \cdot e(P_3, b)
\end{aligned} \tag{2}$$

where $P_1 = \mathsf{com}_{\mathbf{C}} \cdot \mathsf{com}_{\mathbf{E}}^{x}$, $P_2 = \mathsf{com}_B \cdot \mathsf{com}_M^{x^{-1}}$ and $P_3 = c_y \cdot c_l^x \cdot c_m \cdot c_r^{x^{-1}}$.

In order to prove the invariant holds, we need to show that P_1 is a commitment of $\mathbf{C}' = \mathbf{C} \odot \mathbf{E}^x$, P_2 is a commitment of $B' = B + x^{-1} \cdot M$ and P_3 is c'_y where $c'_y = \langle \mathbf{C}', B' \rangle$.

We denote $\mathsf{AFG.Commit}_{\mathsf{ck}_1}$ as $\mathsf{Commit}_{\mathsf{ck}_1}$. We have

$$\begin{aligned}
P_1 &= \mathsf{com}_{\mathbf{C}} \cdot \mathsf{com}_{\mathbf{E}}^{x} \\
&= \mathsf{Commit}_{\mathsf{ck}_1}(\mathbf{C}; r_{\mathbf{C}}) \cdot \mathsf{Commit}_{\mathsf{ck}_1}(\mathbf{E}; r_{\mathbf{E}})^{x} \\
&= \mathsf{Commit}_{\mathsf{ck}_1}(\mathbf{C}; r_{\mathbf{C}}) \cdot \mathsf{Commit}_{\mathsf{ck}_1}(\mathbf{E}^x; r_{\mathbf{E}}) \\
&= \mathsf{Commit}_{\mathsf{ck}_1}(\mathbf{C} \odot \mathbf{E}^x; r_{\mathbf{C}} + x \cdot r_{\mathbf{E}}) \\
&= \mathsf{Commit}_{\mathsf{ck}_1}(\mathbf{C}'; r_{\mathbf{C}} + x \cdot r_{\mathbf{E}}) \\
&= \mathsf{com}_{\mathbf{C}'}
\end{aligned}$$

We denote $\mathsf{Ped.Commit}_{\mathsf{ck}_2}$ as $\mathsf{Commit}_{\mathsf{ck}_2}$ and show that P_2 is a commitment of B'

$$\begin{aligned}
P_2 &= \mathsf{com}_B \cdot \mathsf{com}_M^{x^{-1}} \\
&= \mathsf{Commit}_{\mathsf{ck}_2}(B, r_B) \cdot \mathsf{Commit}_{\mathsf{ck}_2}(M, r_M)^{x^{-1}} \\
&= \mathsf{Commit}_{\mathsf{ck}_2}(B, r_B) \cdot \mathsf{Commit}_{\mathsf{ck}_2}(x^{-1} \cdot M, r_M) \\
&= \mathsf{Commit}_{\mathsf{ck}_2}(B + x^{-1} \cdot M, r_B + x^{-1} \cdot r_M) \\
&= \mathsf{Commit}_{\mathsf{ck}_2}(B', r_B + x^{-1} \cdot r_M) \\
&= \mathsf{com}'_B
\end{aligned}$$

Lastly, we show that $P_3 = c'_y$ where $c'_y = \langle \mathbf{C}', B' \rangle$:

$$\begin{aligned}
P_3 &= c_y \cdot c_l^x \cdot c_m \cdot c_r^{x^{-1}} \\
&= \langle \mathbf{C}, B \rangle \cdot \langle \mathbf{E}, B \rangle^x \cdot \langle \mathbf{E}, M \rangle \cdot \langle \mathbf{C}, M \rangle^{x^{-1}} \\
&= \langle \mathbf{C}, B \rangle \cdot \langle \mathbf{E}^x, B \rangle \cdot \langle \mathbf{E}, M \rangle \cdot \langle \mathbf{C}, M^{x^{-1}} \rangle \\
&= \langle \mathbf{C}, B \rangle \cdot \langle \mathbf{E}^x, B \rangle \cdot \langle \mathbf{E}^x, M^{x^{-1}} \rangle \cdot \langle \mathbf{C}, x^{-1} \cdot M \rangle \\
&= \langle \mathbf{C} \odot \mathbf{E}^x, B \rangle \cdot \langle \mathbf{C} \odot \mathbf{E}^x, x^{-1} \cdot M \rangle \\
&= \langle \mathbf{C} \odot \mathbf{E}^x, B + x^{-1} \cdot M \rangle \\
&= \langle \mathbf{C}', B' \rangle \\
&= c'_y
\end{aligned}$$

Substituting the values of P_1, P_2 and P_3 in Equation 2:

$$\begin{aligned}
\mathsf{com}_0 &= P_1 \cdot e(P_2, a) \cdot e(P_3, b) \\
&= \mathsf{com}_{\mathbf{C}'} \cdot e(\mathsf{com}'_B, a) \cdot e(c'_y, b)
\end{aligned}$$

This shows that the invariant is satisfied at the end of the masking phase.

For the Inner Product Phase (Step 2), we argue that $\mathsf{com}' = \mathsf{com}_{\mathbf{C}'} \cdot e(\mathsf{com}_{F'}, a) \cdot e(c'_y, b)$ where $\mathbf{C}' = \mathbf{C}_L \odot \mathbf{C}_R^x$, $B' = B_L + x^{-1} \cdot B_R$ and $c'_y = \langle \mathbf{C}', B' \rangle$ and $\mathsf{com}_{\mathbf{C}'}$ and $\mathsf{com}_{B'}$ are commitments to $\mathbf{C}'$ and B' respectively. Let the com' generated in step 2(f) be denoted as com_i and expanded below:

$$\begin{aligned}
\mathsf{com}_i &= L^x \cdot \mathsf{com} \cdot R^x \\
&= (\mathsf{com}_{\mathbf{C}_R} \cdot e(\mathsf{com}_{B_L}, a) \cdot e(c_l, b))^x \\
&\qquad \cdot (\mathsf{com}_{\mathbf{C}} \cdot e(\mathsf{com}_{B_L}, a) \cdot e(c_y, b)) \\
&\qquad \cdot (\mathsf{com}_{\mathbf{C}_L} \cdot e(\mathsf{com}_{B_R}, a) \cdot e(c_r, b))^{x^{-1}} \\
&= (\mathsf{com}_{\mathbf{C}_R}^x \cdot \mathsf{com}_{\mathbf{C}} \cdot \mathsf{com}_{\mathbf{C}_L}^{x^{-1}}) \\
&\qquad \cdot (e(\mathsf{com}_{B_L}, a)^x \cdot e(\mathsf{com}_B, a) \cdot e(\mathsf{com}_{B_R}, a)^{x^{-1}}) \\
&\qquad \cdot (e(c_l, b)^x \cdot e(c_y, b) \cdot e(c_r, b)^{x^{-1}}) \\
&= (\mathsf{com}_{\mathbf{C}_R}^x \cdot \mathsf{com}_{\mathbf{C}} \cdot \mathsf{com}_{\mathbf{C}_L}^{x^{-1}}) \\
&\qquad \cdot (e(\mathsf{com}_{B_L}^x, a) \cdot e(\mathsf{com}_B, a) \cdot e(\mathsf{com}_{B_R}^{x^{-1}}, a)) \\
&\qquad \cdot (e(c_l^x, b) \cdot e(c_y, b) \cdot e(c_r^{x^{-1}}, b)) \\
&= (\mathsf{com}_{\mathbf{C}_R}^x \cdot \mathsf{com}_{\mathbf{C}} \cdot \mathsf{com}_{\mathbf{C}_L}^{x^{-1}}) \\
&\qquad \cdot e(\mathsf{com}_{B_L}^x \cdot \mathsf{com}_B \cdot \mathsf{com}_{B_R}^{x^{-1}}, a) \\
&\qquad \cdot e(c_l^x \cdot c_y \cdot c_r^{x^{-1}}, b) \\
&= P_1 \cdot e(P_2, a) \cdot e(P_3, b) \qquad (3)
\end{aligned}$$

where $P_1 = \left(\mathsf{com}_{\mathbf{C}_R}^x \cdot \mathsf{com}_{\mathbf{C}} \cdot \mathsf{com}_{\mathbf{C}_L}^{x^{-1}}\right)$, $P_2 = \mathsf{com}_{B_L}^x \cdot \mathsf{com}_B \cdot \mathsf{com}_{B_R}^{x^{-1}}$ and $P_3 = c_l^x \cdot c_y \cdot c_r^{x^{-1}}$.

In order to prove the invariant, we show that P_1 is a commitment of $\mathbf{C}' = \mathbf{C}_L \odot \mathbf{C}_R^x$, P_2 is a valid commitment of $B' = B_L + x^{-1} \cdot B_R$ and P_3 is c'_y where $c'_y = \langle \mathbf{C}', B' \rangle$.

We denote $\mathsf{AFG.Commit}_{\mathsf{ck}_1}$ as $\mathsf{Commit}_{\mathsf{ck}_1}$ and we show that P_1 is a commitment of $\mathbf{C}'$:

$$\begin{aligned}
P_1 &= \mathsf{com}_{\mathbf{C}_R}^x \cdot \mathsf{com}_{\mathbf{C}} \cdot \mathsf{com}_{\mathbf{C}_L}^{x^{-1}} \\
&= \mathsf{Commit}_{\mathsf{ck}_{1L}}(\mathbf{C}_R, r_{\mathbf{C}_R})^x \cdot \mathsf{Commit}_{\mathsf{ck}_1}(\mathbf{C}, r_{\mathbf{C}}) \\
&\qquad \cdot \mathsf{Commit}_{\mathsf{ck}_{1R}}(\mathbf{C}_L, r_{\mathbf{C}_L})^{x^{-1}} \\
&= \mathsf{Commit}_{\mathsf{ck}_{1L}}(\mathbf{C}_R^x, x \cdot r_{\mathbf{C}_R}) \cdot \mathsf{Commit}_{\mathsf{ck}_1}(\mathbf{C}, r_{\mathbf{C}}) \\
&\qquad \cdot \mathsf{Commit}_{\mathsf{ck}_{1R}^{x^{-1}}}(\mathbf{C}_L, x^{-1} \cdot r_{\mathbf{C}_L}) \\
&= \mathsf{Commit}_{\mathsf{ck}_{1L}}(\mathbf{C}_R^x, x \cdot r_{\mathbf{C}_R}) \cdot \mathsf{Commit}_{\mathsf{ck}_{1L}}(\mathbf{C}_L, r_{\mathbf{C}_L}) \\
&\qquad \cdot \mathsf{Commit}_{\mathsf{ck}_{1R}}(\mathbf{C}_R, 0) \cdot \mathsf{Commit}_{\mathsf{ck}_{1R}^{x^{-1}}}(\mathbf{C}_L, x^{-1} \cdot r_{\mathbf{C}_L}) \\
&= \mathsf{Commit}_{\mathsf{ck}_{1L}}(\mathbf{C}_R^x, x \cdot r_{\mathbf{C}_R}) \cdot \mathsf{Commit}_{\mathsf{ck}_{1L}}(\mathbf{C}_L, r_{\mathbf{C}} \\
&\qquad \cdot \mathsf{Commit}_{\mathsf{ck}_{1R}^{x^{-1}}}(\mathbf{C}_R^x, 0) \cdot \mathsf{Commit}_{\mathsf{ck}_{1R}^{x^{-1}}}(\mathbf{C}_L, x^{-1} \cdot r_{\mathbf{C}_L}) \\
&= \mathsf{Commit}_{\mathsf{ck}_{1L}}(\mathbf{C}_L \odot \mathbf{C}_R^x, x \cdot r_{\mathbf{C}_R} + r_{\mathbf{C}}) \\
&\qquad \cdot \mathsf{Commit}_{\mathsf{ck}_{1R}^{x^{-1}}}(\mathbf{C}_L \odot \mathbf{C}_R^x, x^{-1} \cdot r_{\mathbf{C}_L}) \\
&= \mathsf{Commit}_{\mathsf{ck}_{1L} \odot \mathsf{ck}_{1R}^{x^{-1}}}(\mathbf{C}_L \odot \mathbf{C}_R^x, x \cdot r_{\mathbf{C}_R} + r_{\mathbf{C}} + x^{-1} \cdot r_{\mathbf{C}_L}) \\
&= \mathsf{Commit}_{\mathsf{ck}_{1L} \odot \mathsf{ck}_{1R}^{x^{-1}}}(\mathbf{C}', x \cdot r_{\mathbf{C}_R} + r_{\mathbf{C}} + x^{-1} \cdot r_{\mathbf{C}_L}) \\
&= \mathsf{com}_{\mathbf{C}'}
\end{aligned}$$

We denote $\mathsf{Ped.Commit}_{\mathsf{ck}_2}$ as $\mathsf{Commit}_{\mathsf{ck}_2}$ and we show that P_2 is a valid commitment of B':

$$\begin{aligned}
P_2 &= \mathsf{com}_{B_L}^{x} \cdot \mathsf{com}_{B} \cdot \mathsf{com}_{B_R}^{x^{-1}} \\
&= \mathsf{Commit}_{\mathsf{ck}_{2R}}(B_L, r_{B_L})^{x} \cdot \mathsf{Commit}_{\mathsf{ck}_2}(B, r_B) \cdot \\
&\qquad \mathsf{Commit}_{\mathsf{ck}_{2L}}(B_R, r_{B_R})^{x^{-1}} \\
&= \mathsf{Commit}_{\mathsf{ck}_{2R}}(B_L, r_{B_L})^{x} \cdot \mathsf{Commit}_{\mathsf{ck}_{2R}}(B_R, 0) \\
&\qquad \cdot \mathsf{Commit}_{\mathsf{ck}_{2L}}(B_L, r_B) \cdot \mathsf{Commit}_{\mathsf{ck}_{2L}}(B_R, r_{B_R})^{x^{-1}} \\
&= \mathsf{Commit}_{\mathsf{ck}_{2R}^{x}}(B_L, x \cdot r_{B_L}) \cdot \mathsf{Commit}_{\mathsf{ck}_{2R}}(B_R, 0) \\
&\qquad \cdot \mathsf{Commit}_{\mathsf{ck}_{2L}}(B_L, r_B) \cdot \mathsf{Commit}_{\mathsf{ck}_{2L}}(x^{-1} \cdot B_R, x^{-1} \cdot r_{B_R}) \\
&= \mathsf{Commit}_{\mathsf{ck}_{2R}^{x}}(B_L, x \cdot r_{B_L}) \cdot \mathsf{Commit}_{\mathsf{ck}_{2R}^{x}}(x^{-1} \cdot B_R, 0) \\
&\qquad \cdot \mathsf{Commit}_{\mathsf{ck}_{2L}}(B_L, r_B) \cdot \mathsf{Commit}_{\mathsf{ck}_{2L}}(x^{-1} \cdot B_R, x^{-1} \cdot r_{B_R}) \\
&= \mathsf{Commit}_{\mathsf{ck}_{2R}^{x}}(B_L + x^{-1} \cdot B_R, x \cdot r_{B_L}) \\
&\qquad \cdot \mathsf{Commit}_{\mathsf{ck}_{2L}}(B_L + x^{-1} \cdot B_R, r_B + x^{-1} \cdot r_{B_R}) \\
&= \mathsf{Commit}_{\mathsf{ck}_{2L} \odot \mathsf{ck}_{2R}^{x}}(B_L + x^{-1} \cdot B_R, x \cdot r_{B_L} + r_B + x^{-1} \cdot r_{B_R}) \\
&= \mathsf{Commit}_{\mathsf{ck}_{2L} \odot \mathsf{ck}_{2R}^{x}}(B', x \cdot r_{B_L} + r_B + x^{-1} \cdot r_{B_R}) \\
&= \mathsf{com}_{B'}
\end{aligned}$$

Lastly, we show that $P_3 = c'_y$ where $c'_y = \langle \mathbf{C}', B' \rangle$:

$$\begin{aligned}
P_3 &= c_l^{x} \cdot c_y \cdot c_r^{x^{-1}} \\
&= \langle \mathbf{C}_R, B_L \rangle^{x} \cdot \langle \mathbf{C}, B \rangle \cdot \langle \mathbf{C}_L, B_R \rangle^{x^{-1}} \\
&= \langle \mathbf{C}_R^{x}, B_L \rangle \cdot \langle \mathbf{C}, B \rangle \cdot \langle \mathbf{C}_L, B_R^{x^{-1}} \rangle \\
&= \langle \mathbf{C}_R^{x}, B_L \rangle \cdot \langle \mathbf{C}_L, B_L \rangle \cdot \langle \mathbf{C}_R, B_R \rangle \cdot \langle \mathbf{C}_L, B_R^{x^{-1}} \rangle \\
&= \langle \mathbf{C}_R^{x}, B_L \rangle \cdot \langle \mathbf{C}_L, B_L \rangle \cdot \langle \mathbf{C}_R^{x}, B_R \rangle \cdot \langle \mathbf{C}_L, x^{-1} \cdot B_R \rangle \\
&= \langle \mathbf{C}_L \odot \mathbf{C}_R^{x}, B_L \rangle \cdot \langle \mathbf{C}_L \odot \mathbf{C}_R^{x}, x^{-1} \cdot B_R \rangle \\
&= \langle \mathbf{C}_L \odot \mathbf{C}_R^{x}, B_L + x^{-1} \cdot B_R \rangle \\
&= \langle \mathbf{C}', B' \rangle \\
&= c'_y
\end{aligned}$$

Substituting the values of P_1, P_2 and P_3 in Equation 3:

$$\begin{aligned}
\mathsf{com}_i &= P_1 \cdot e(P_2, a) \cdot e(P_3, b) \\
&= \mathsf{com}_{\mathbf{C}'} \cdot e(\mathsf{com}'_B, a) \cdot e(c'_y, b)
\end{aligned}$$

This shows that the invariant is satisfied. In the last round, C opens up all the commitment and is trivially complete.

Linear Constraint Test: In the “Linear Constraint Test", the protocol reduces checking $A \times T = X$ to checking $S \times A \times T \times U = S \times X \times U$ where S and U are generated by R. As S, U are random linear combiners, this allows the reduction of the problem to go through. Assuming that BBA-IPA satisfies completeness, in the “Linear Constraint Test", we reduce the test from checking $\langle S_A, T_U\rangle = \langle S, X_U\rangle$ to $\langle L, R\rangle = x\cdot l + x^{-1}r$ where $l = \langle S_A, -X_U\rangle$ and $r = \langle S, T_U\rangle$. We show that $\langle L, R\rangle = x\cdot l + x^{-1}\cdot r$ below:

$$\begin{aligned}
\langle L, R\rangle &= x\cdot l + x^{-1}\cdot r\\
&= x\cdot\langle S_A, -X_U\rangle + x^{-1}\cdot\langle S, T_U\rangle\\
&= \langle S_A, -x\cdot X_U\rangle + \langle x^{-1}\cdot S, T_U\rangle\\
&= \langle S_A, -x\cdot X_U\rangle + \langle S_A, T_U\rangle +\\
&\quad \langle S, -X_U\rangle + \langle x^{-1}\cdot S, T_U\rangle\\
&= \langle S_A, -x\cdot X_U\rangle + \langle S_A, T_U\rangle +\\
&\quad \langle x^{-1}\cdot S, -x\cdot X_U\rangle + \langle x^{-1}\cdot S, T_U\rangle\\
&= \langle S_A, T_U - x\cdot X_U\rangle + \langle x^{-1}\cdot S, T_U - x\cdot X_U\rangle\\
&= \langle S_A + x^{-1}\cdot S, T_U - x\cdot X_U\rangle
\end{aligned}$$

At the end, the completeness property of BBB-IPA ensures the completeness of this phase.

Quadratic Constraint Test: In the “Quadratic Constraint Test", the protocol reduces checking $\forall i\in[m], X_i\odot Y_i = Z_i$ is modified to a single check for each i by using a random linear combiner and is modified as $X_i\odot Y_i\odot R = Z_i\odot R$. This can furthermore be written as in inner product $\forall i\in[m]\langle X_i, Y_i\odot R\rangle - \langle Z_i\odot R\rangle = 0$. This, in turn, can be reduced to a single inner product using another random linear combiner $\sum_{i=1}^{m} w^i(\langle X_i, Y_i\odot R\rangle - \langle Z_i\odot R\rangle)$. Using the similar technique in phase 2 (or "Linear Constraint Test"), the summation formerly introduces can be reduced to a single inner product (at every iteration, the number of inner products is reduced by a factor of 2). The analysis used to show the completeness of phase 2 can also be applied here to show completeness after every iteration. In the end, the completeness property of BBB-IPA ensures the completeness of this phase. This concludes the proof of completeness.

Claim. Protocol PCOM (Figures 3) satisfies Binding

To argue the binding property of PCOM, it can be trivially reduced to the binding property of the Ped and AFG commitment scheme.

Claim. Protocol PCOM (Figures 3) satisfies Witness-Extended Emulation

Consider a public-coin interactive protocol with r rounds. We define $(n_1,\ldots,n_r)$-tree of accepting transcripts for this interactive protocol as follows. The tree is of depth r where the root is labelled with the statement and each node in depth i has n_i children, where each child is associated with the i^{th} challenge. Each edge from parent to child node is associated with a message sent from the prover to the verifier. Each root-to-leaf path corresponds to an accepting transcript.

Using the Forking Lemma (Lemma 1), we can reduce the witness-extended emulation property to the existence of a PPT extractor $\mathcal{X}$, which given $(n_1, \dots, n_r)$-tree of accepting transcripts can extract the witness of the polynomial commitment scheme. In the proof discussed below, we will assume we have these $(n_1, \dots, n_r)$-tree of accepting transcripts where each $n_i = 3$ except $n_1 = m$. This ensures $\prod_{r}^{i=1} n_i = m \cdot 3^{\lceil \log_2(d)+1 \rceil} \leq 3 \cdot md^{3/2}$ which is bounded by a polynomial in d and m and in turn λ.

First, we construct a witness-extraction algorithm $\mathcal{X}_1$ that succeeds in extracting the witness of the "Private Inner Product Argument" given $(n_1, \dots, n_r)$-tree of accepting transcripts.

Denote the challenge message send by C in Round i as $x^{(i)}$. We also represent variables in each round with superscript (i).

In round a, C opens the commitment com. This allows $\mathcal{X}_1$ to get $\mathbf{C}^{(a)}$ and $B^{(a)}$ for round a. The extractor $\mathcal{X}_1$ rewinds C three times. Except with negligible probability, it receives three different values of $x^{(i)}$ denoted by x_1, x_2, x_3 such that $x_i \neq x_j$ for $1 \leq i < j \leq 3$. Let us denote these transcripts as T_1, T_2, T_3.

Using T_1, T_2, T_3, $\mathcal{X}_1$ obtains B_L and B_R by solving the linear equation below:

$$B_L + x_i^{-1} \cdot B_R = B_i^{(a)} \tag{4}$$

where $B_j^{(a)}$ is $B^{(a)}$ extracted in transcript T_j.

$\mathcal{X}_1$ extracts $B^{(a-1)} = B_L || B_R$.

Using T_1, T_2, T_3, $\mathcal{X}_1$ obtains $\mathbf{C}_L$ and $\mathbf{C}_R$ by solving the linear equation below:

$$\mathbf{C}_L \odot \mathbf{C}_R^{x_i} = \mathbf{C}_j^{(a)}$$

where $\mathbf{C}_j^{(i+1)}$ is $\mathbf{C}^{(i+1)}$ extracted in transcript T_j.

$\mathcal{X}_1$ extracts $\mathbf{C}^{(a-1)} = \mathbf{C}_L || \mathbf{C}_R$.

We can repeat the above steps recursively for every round. We also use this same technique for extracting the unmasked version of $\mathbf{C}$ and B.

Now to extract all the evaluation vectors $\mathcal{X}$ extracts different B using $\mathcal{X}_1$ based on the random linear combiner $(r_1, \dots, r_m)$. Now $\mathcal{X}$ rewinds the C $m-$ times where it receives, except with negligible probability, $m-$ transcript $T_1, \dots T_m$ with $m-$ different set of linear combiners where $(r_1^i, \dots r_m^i)$ is set as the random linear combiners for each transcript T_i.

Using $T_1, \dots, T_m$, $\mathcal{X}$ obtains t_i by solving the linear equation below:

$$B_j = \sum_{i=1}^{m} r_i^j t_i$$

where B_j is B extracted in transcript T_j.

Once B is extracted, $\mathcal{X}$ can rewind the protocol to Figure 3 (Step 1) and repeat it till there are $m+1$ set of different vectors S. Using this and solving linear equations, $\mathcal{X}$ can extract each evaluation vector $\{T_i\}_{i \in [m]}$ and can extract $\{t_i\}_{i \in [m]}$ trivially.

Claim. Protocol PCOM (Figures 3) satisfies Honest Verifier Privacy

We describe the Simulator $\mathcal{S}$ in Figure 15. Indistinguishability of the simulation essentially follows from semantic security of the underlying encryption scheme, hiding of the commitment scheme and masking techniques. More formally, we consider a sequence of intermediate hybrid experiments and argue indistinguishability via a standard hybrid argument.

Hybrid$_0$: This hybrid experiment proceeds as in the real world, i.e. the output of the experiment is the output of the malicious receiver when interacting with the honest committer.

Hybrid$_1$: This hybrid is identical to **Hybrid**$_0$ with the exception that we consider a simulator that proceeds as an honest committer against the malicious receiver but replaces the protocol of BBB-IPA in “Linear Test Constraints" and “Quadratic Test Constraints" with its simulation $\mathcal{S}_{BBB-IPA}$. The indistinguishability of **Hybrid**$_0$ and **Hybrid**$_1$ follows from the Zero-Knowledge property of BBB-IPA.

Hybrid$_2$: This hybrid is identical to **Hybrid**$_1$ with the exception that the simulator replaces all the l_i and r_i vector in “Quadratic Constraint Test" with random vectors. This is possible due to the fact that any checks related to l_i and r_i were done in BBB-IPA which is replaced by its simulator $\mathcal{S}_{BBB-IPA}$ and will generate a transcript that is always accepted by R. This is the same as replacing the “Quadratic Constraint Test" with $\mathcal{S}_{p3}$.

Hybrid$_3$: This hybrid is identical to **Hybrid**$_2$ with the exception that the simulator chooses a random vector v in “Linear Constraint Test" with a random vector as well as chooses l, r such that $l^x + r^{x^{-1}} =< s_A - x^{-1} \cdot s, v >$. This is possible due to the fact that any checks related to v, l and r were done in BBB-IPA which is replaced by its simulator $\mathcal{S}_{BBB-IPA}$ and will generate a transcript that is always accepted by R. This is the same as replacing the “Linear Constraint Test" with $\mathcal{S}_{p2}$.

Hybrid$_4$: This hybrid is identical to **Hybrid**$_3$ with the exception that the simulator chooses random encrypted polynomial $\mathbf{C}'$ and a random vector B' and set $\mathsf{com}_{\mathbf{E}}$ such that $\mathsf{com}_{\mathbf{C}} \cdot \mathbf{E}^{x_m} = \mathsf{com}_{\mathbf{C}'}$, com_M such that $\mathsf{com}_B \cdot \mathsf{com}_M^{x_m^{-1}} = \mathsf{com}_{B'}$ and c_l, c_r such that $c_l^{x_m} \cdot c_y \cdot c_r^{x_m^{-1}} = c'_y$. This is possible due to the fact the inner product part of the "Private Inner Product Argument" will accept this proof. This is the same as replacing “Private Inner Product Argument" with $\mathcal{S}_{p1}$.

Hybrid$_5$: This hybrid is identical to **Hybrid**$_4$ with the exception that the simulator replaces c_y in Eval algorithm with $\mathcal{S}_3$. This involves outputting a random ciphertext. The indistinguishability of **Hybrid**$_4$ and **Hybrid**$_5$ follows from the IND-CPA security of the encryption scheme.

Hybrid$_6$: This hybrid is identical to **Hybrid**$_5$ with the exception that the simulator replaces $\mathsf{com}_{\mathbf{C}}$ generated using Commit algorithm with $\mathcal{S}_1$. This involves committing to a random encrypted polynomial. The indistinguishability of **Hybrid**$_5$ and **Hybrid**$_6$ follows from the hiding property of the commitment scheme.

Hybrid$_7$: This hybrid is identical to **Hybrid**$_6$ with the exception that the simulator replaces com_T generated using $\mathsf{CommitPt}$ algorithm with $\mathcal{S}_2$. This involves committing to a random point. The indistinguishability of $\mathbf{Hybrid}_6$ and $\mathbf{Hybrid}_7$ follows from the hiding property of the commitment scheme.

$\mathbf{Hybrid}_8$: This hybrid is identical to $\mathbf{Hybrid}_7$ with the exception that the simulator replaces protocol (C, R) with $\mathcal{S}$. We argue that $\mathbf{Hybrid}_7$ and $\mathbf{Hybrid}_8$ are perfectly indistinguishable. This is due to the fact that protocol (C, R) in $\mathbf{Hybrid}_7$ does not use the real inputs in the simulation.

$\mathbf{Hybrid}_9$: This hybrid is identical to $\mathbf{Hybrid}_8$ with the exception that simulator does not get the real inputs. We argue that $\mathbf{Hybrid}_8$ and $\mathbf{Hybrid}_9$ are perfectly indistinguishable. This is due to the fact that $\mathbf{Hybrid}_8$ does not use the real inputs in the simulation.

A.2 Proof of Theorem 2

We split the analysis into two cases based on whether the set of corrupted parties includes the central party P_1 or not.

Case 1: the corrupted set includes the central party P_1 : Consider an adversary $\mathcal{A}$ that corrupts a set of parties that includes P_1. We define the simulator $\mathcal{S}$ in Figure 16. We prove that the real and simulated executions are computationally indistinguishable. The differences in both executions reduce to the privacy property of the encryption scheme and the hiding property of the commitment scheme. Our proof follows via a sequence of hybrids between $\mathbf{Hybrid}_0$ to $\mathbf{Hybrid}_4$ where $\mathbf{Hybrid}_0$ is identical to the real execution while $\mathbf{Hybrid}_4$ is identical to the simulated execution.

$\mathbf{Hybrid}_0$: The first game is the real execution.

$\mathbf{Hybrid}_1$: This hybrid is identical to $\mathbf{Hybrid}_0$ with the exception that we define a simulator that extracts the input of corrupted parties as done in the simulation. More specifically, let X_i denote the input of party P_i. Recall that the simulator knows the input of the honest parties as well, it then checks whether the final output is correct with respect to all inputs and aborts otherwise. Finally, based on the witness-extended emulation property of PCOM, the correctness of π_{DecZero} and the binding property of the commitment generated by P_1, the extracted values will be consistent. Therefore the simulator will only abort with negligible probability. This implies that the output distributions of the two executions are statistically close.

$\mathbf{Hybrid}_2$: This hybrid is identical to $\mathbf{Hybrid}_1$ with the exception that the simulator replaces π_{GEN} and π_{DecZero} with $\mathcal{S}_{\text{GEN}}$ and $\mathcal{S}_{\text{DecZero}}$, respectively. Note that, this allows the simulator to simulate π_{DecZero} without knowing the actual secret key. Let Z denote the intersection as computed in the previous hybrid. In this hybrid, the simulator enforces the output of π_{DecZero} according to whether the evaluated point is in Z or not. Namely, for every $z \in X_1$, if $z \in Z$ then $\mathcal{S}_{\text{DecZero}}$ enforces the output to be zero, else a random element. This is done by invoking $\mathcal{S}_{\text{DecZero}}$ on the public key, the secret key shares of the corrupted parties and the resultant plaintext. Indistinguishability follows from the security of the threshold encryption scheme. This implies that $\mathbf{Hybrid}_1$ and $\mathbf{Hybrid}_2$ are indistinguishable.

$\mathbf{Hybrid}_3$: This hybrid is identical to $\mathbf{Hybrid}_2$ with the exception that the simulator replaces π_{EXP} with its simulator $\mathcal{S}_{\mathrm{EXP}}$ respectively. The indistinguishability of $\mathbf{Hybrid}_2$ and $\mathbf{Hybrid}_3$ follows from the zero-Knowledge property of π_{EXP}.

$\mathbf{Hybrid}_4$: This hybrid is identical to $\mathbf{Hybrid}_3$ with the exception that the simulator replaces the real inputs of honest parties with random inputs. Namely, the simulator sends ciphertexts encrypting random polynomials on behalf of the honest parties. The indistinguishability of $\mathbf{Hybrid}_3$ and $\mathbf{Hybrid}_4$ follows from the IND-CPA security of the encryption scheme. As the simulator does not need to know the secret key, the ciphertexts obtained from the IND-CPA security game can directly be plugged into the protocol and can be reduced to the IND-CPA security game.

Finally, we note that the above hybrid is identical to the simulation. This concludes the proof of the first case.

Case 2: the corrupted set excludes the central party P_1: The proof for this corruption case is very similar to the previous case with the exception that the simulator does not need to extract the input of P_1.

Consider an adversary $\mathcal{A}$ that corrupts a set of parties that excludes P_1. We define the simulator $\mathcal{S}$ in Figure 17. We prove that the ensemble of real execution and simulated execution are computationally indistinguishable. The difference in both the ensemble reduces to the privacy property of the encryption scheme and hiding property of perfectly hiding commitments. Our proof follows a sequence of hybrid proofs. Our proof consists of multiple hybrids from $\mathbf{Hybrid}_0$ to $\mathbf{Hybrid}_5$. $\mathbf{Hybrid}_0$ is identical to the real execution while $\mathbf{Hybrid}_5$ is identical to the simulated execution.

$\mathbf{Hybrid}_0$: The first game is the real execution.

$\mathbf{Hybrid}_1$: This hybrid is identical to $\mathbf{Hybrid}_0$ with the exception that the simulator extracts the input of corrupted parties. The simulator extracts the inputs of all corrupted parties from π_{EXP}, and aborts if it fails to extract. Let X_i be the inputs for each corrupt party P_i. It checks if the final output is correct with respect to X_i. This is possible as the simulator in this hybrid knows the real input of honest parties. For extracting the inputs of corrupt parties, the simulator rewinds to the point where all parties generate a random value u. The simulator rewinds till $d+1$ evaluations are recorded where π_{EXP} provides a valid proof and the message can be extracted. For each corrupt party P_i, the simulator receives $d+1$ evaluation on its polynomial and then can be interpolated. The roots of the interpolated polynomial are the inputs of the corrupt party. Taking the intersection of these inputs generates X^*. Finally using the witness-extended emulation of PCOM, soundness of π_{DecZero} and the binding property of the commitment generated by P_1, the extracted values will be consistent. Therefore the simulator will abort with negligible probability if extracted inputs is not consistent and will deviate from $\mathbf{Hybrid}_0$ otherwise $\mathbf{Hybrid}_0$ will be perfectly indistinguishable from $\mathbf{Hybrid}_1$. This shows that $\mathbf{Hybrid}_0$ and $\mathbf{Hybrid}_1$ are indistinguishable.

Hybrid$_2$: This hybrid is identical to **Hybrid**$_1$ with the exception that the simulator replaces π_{GEN} and π_{DecZero} with $\mathcal{S}_{\text{GEN}}$ and $\mathcal{S}_{\text{DecZero}}$ respectively. The set intersection X can also be evaluated based on the extracted inputs of corrupt parties and inputs of the honest parties. $\mathcal{S}_{\text{DecZero}}$ is given a message and it needs to bias the output of decryption according to the message. For every $z \in X_1$, if $z \in X$ then $\mathcal{S}_{\text{DecZero}}$ will bias the output to be zero else a random element. The indistinguishability follows from the property of threshold decryption. This shows that **Hybrid**$_1$ and **Hybrid**$_2$ are indistinguishable.

Hybrid$_3$: This hybrid is identical to **Hybrid**$_2$ with the exception that the simulator replaces PCOM and π_{EXP} with their respective simulators $\mathcal{S}_{\mathsf{PCOM}}$ and $\mathcal{S}_{\text{EXP}}$ respectively. The indistinguishability of **Hybrid**$_2$ and **Hybrid**$_3$ follows from the Zero-Knowledge property of PCOM.

Hybrid$_4$: This hybrid is identical to **Hybrid**$_3$ with the exception that the simulator replaces the commitment generated by P_1 with a commitment to 0-message. The indistinguishability of **Hybrid**$_3$ and **Hybrid**$_4$ follows from the hiding property of the commitment scheme. We reduce the indistinguishability of **Hybrid**$_3$ and **Hybrid**$_4$ to the hiding game of commitment.

Hybrid$_5$: This hybrid is identical to **Hybrid**$_4$ with the exception that the simulator replaces the real inputs of honest parties with random inputs. The simulator sends the encryption evaluation of random polynomials on behalf of the honest parties. The indistinguishability of **Hybrid**$_4$ and **Hybrid**$_5$ follows from the IND-CPA security of the encryption scheme. As the simulator does not need to know the secret key, the ciphertext obtained from the IND-CPA security game can directly be plugged into the protocol and can be reduced to the IND-CPA security game.

$\mathcal{S}$ for PCOM

$\mathcal{S}(\mathsf{pp}, \mathrm{PK}, d; r_s)$:

1. Invoke Simulator S_1, S_2 and S_3 to generate $\mathsf{com}_{\mathbf{C}} \leftarrow \mathcal{S}_1(\mathsf{pp}, \mathrm{PK}; r_s)$, $\mathsf{com}_T \leftarrow \mathcal{S}_2(\mathsf{pp}, d; r_s)$ and $c_y \leftarrow \mathcal{S}_3(\mathrm{PK}; r_s)$.
2. The simulators $\mathcal{S}_{p1}$, $\mathcal{S}_{p2}$ and $\mathcal{S}_{p3}$ are simulators for "Private Inner Product Argument", "Linear Constraint Test" and "Quadratic Constraint Test". $\mathcal{S}_4$ invokes the simulator $\mathcal{S}_{p1}(\mathsf{pp}, \mathrm{PK}, \mathsf{com}_{\mathbf{C}}, \mathsf{com}_T, c_y; r_s)$, $\mathcal{S}_{p2}(\mathsf{pp}, \mathsf{com}_T; r_s)$ and $\mathcal{S}_{p3}(\mathsf{pp}, \mathsf{com}_T; r_s)$

$\mathsf{com}_{\mathbf{C}} \leftarrow \mathcal{S}_1(\mathsf{pp}, \mathrm{PK}; r_s)$:

1. Generate a random encrypted polynomial $\mathbf{C}^*$ and compute com_C^* as follows:

$$\mathsf{com}_{\mathbf{C}} = \mathsf{Commit}(\mathsf{pp}, \mathbf{C}^*; r^*_{\mathbf{C}})$$

where $r^*_{\mathbf{C}}$ is randomly chosen. We use randomness r_s to generate $\mathbf{C}^*$ and $r^*_{\mathbf{C}}$.

$\mathsf{com}_T \leftarrow \mathcal{S}_2(\mathsf{pp}, d; r_s)$:

1. Generate a random point t^* and compute com_T as follows:

$$\mathsf{com}_T = \mathsf{CommitPt}(\mathsf{pp}, t^*, d, r^*_t)$$

where r^*_t is randomly chosen. We use randomness r_s to generate t^* and r^*_t.

$c_y \leftarrow \mathcal{S}_3(\mathrm{PK}; r_s)$:

1. Generate a random value m and r and compute c_y as follows:

$$c_y = \mathsf{BBS.Enc}_{\mathrm{PK}}(m; r)$$

where PK is the public key.We use randomness r_s to generate m and r.

$\mathcal{S}_{p1}(\mathsf{pp}, \mathrm{PK}, \mathsf{com}_{\mathbf{C}}, \mathsf{com}_T, c_y; r_s)$:

1. Generate a random encrypted polynomial $\mathbf{C}'$ and a random vector B' using randomness r_s. Also generate their respective commitments $\mathsf{com}_{\mathbf{C}'}$ and $\mathsf{com}_{B'}$ honestly. Also set $c'_y =< \mathbf{C}', B' >$.
2. Based on the challenge x_m in the masking phase, set:
   - $\mathsf{com}_{\mathbf{E}}$ such that $\mathsf{com}_{\mathbf{C}} \cdot \mathsf{com}_{\mathbf{E}}^{x_m} = \mathsf{com}_{\mathbf{C}'}$ is satisfied.
   - com_M such that $\mathsf{com}_B \cdot \mathsf{com}_M^{x_m{}^{-1}} = \mathsf{com}_{B'}$ is satisfied.
   - c_l, c_r such that $c_l^{x_m} \cdot c_y \cdot c_r^{x_m{}^{-1}} = c'_y$ is satisfied.
3. Run the test similar to "Private Inner Product Argument" with the changes above in the masking part.

$\mathcal{S}_{p2}(\mathsf{pp}, \mathsf{com}_T; r_s)$:

1. Choose a random vector v using randomness r_s and set l, r such that $l \cdot x + x^{-1} \cdot r =< s_A - x^{-1} \cdot s, v >$.
2. Run the test similar to "Linear Constraint Test" with the changes above in Step 3 in the protocol as well as replace BBB-IPA and instead execute its simulator $\mathcal{S}_{BBB-IPA}$ on the inner product $< s_A - x^{-1} \cdot s, v >= x \cdot l + x^{-1} \cdot r$.

$\mathcal{S}_{p3}(\mathsf{pp}, \{\mathsf{com}_{t_1}, \ldots, \mathsf{com}_{t_m}\}; r_s)$:

1. Choose 4m random vector l_i and r_i for $i \in [2m]$ using randomness r_s.
2. Run the test similar to the " Quadratic Constraint Test" with the changes above in Step 3 in the protocol as well as replace BBB-IPA and instead execute its simulator $\mathcal{S}_{BBB-IPA}$.

Fig. 15: Simulator $\mathcal{S}$ for PCOM

Simulator $\mathcal{S}$

1. **Key Generation**: $\mathcal{S}$ generates $(\text{PK}, \text{SK}) \leftarrow \mathsf{KeyGen}$ and invokes the simulator $\mathcal{S}_{\mathsf{Gen}}(\text{PK})$ for π_{Gen}.
2. **Commitment Phase**: Upon receiving the commitments on P_1 inputs' as well as a proof of knowledge π_{EXP}, $\mathcal{S}$ extracts the input X_1 of P_1 by invoking the extractor E^{DL}.
3. **Aggregation**:
   (a) For every party $i \in \mathcal{H}$ where $\mathcal{H}$ is set of all honest parties, $\mathcal{S}$ generates a random polynomial $C_i(\cdot)$ of degree m_i, encrypts its coefficients and sends the ciphertexts to P_1 on behalf of P_i.
   (b) Upon receiving the commitment of the encrypted polynomial $\mathbf{C}$ denoted by $\mathsf{com}_{\mathbf{C}}$, $\mathcal{S}$ participates honestly in π_{COIN} on behalf of the honest parties.
   (c) Let u denote the output of π_{COIN}. $\mathcal{S}$ then participates uses $\mathsf{PCOM}.V_{pub}$ and verifies whether $\tilde{\lambda} = \mathsf{Eval}(\text{PK}, \mathbf{C}, u)$ and $\mathsf{com}_{\mathbf{C}} = \mathsf{PCOM}.\mathsf{Commit}(\mathsf{pp}, \mathbf{C})$.
   (d) $\mathcal{S}$ honestly evaluates its input polynomial on u and invokes the simulator for π_{EXP} on behalf of the honest parties.
   (e) Excluding the central party P_1, $\mathcal{S}$ extracts the corrupt parties' inputs. This is achieved by extracting $d+1$ evaluation points of every corrupted party's polynomial. In more details, each corrupt party sends a ciphertext together with a proof of knowledge π_{EXP}, where these ciphertexts are generated by evaluating the input polynomials on the evaluation point u. Upon receiving the ciphertexts and the proofs, $\mathcal{S}$ extracts the plaintexts by running the simulation-extractor E^{DL}. This process is repeated for every corrupted party multiple times. Namely, to extract the input polynomial of each corrupted party, $\mathcal{S}$ rewinds the adversary to the beginning of π_{COIN} in order to generate a new random element v. $\mathcal{S}$ records $d+1$ such evaluations for each corrupted party and use them to interpolate each input polynomial and extract its roots. Let the input set extracted for a corrupt party P_i denoted by X_i. $\mathcal{S}$ also invokes the simulator of π_{DecZero} on behalf of the honest parties.
4. **Intersection**:
   (a) $\mathcal{S}$ sends X_i as an input of corrupt party P_i to the trusted party and receives set intersection Z from the trusted party.
   (b) $\mathcal{S}$ receives c_{y_i} for each $x_i \in X_1$. $\mathcal{S}$ furthermore participates honestly in PCOM on behalf of other honest parties to ensure $c_{y_i} = \mathsf{Eval}(\text{PK}, \mathbf{C}, x_i)$ and com_{x_i} is a commitment to x_i.
   (c) For every $x_1^j \in Z$ where x_1^j is j^{th} element in set X_1, $\mathcal{S}$ enforces the decryption to be zero and a random element otherwise. This is achieved by invoking the simulator $\mathcal{S}_{\text{DecZero}}$ on the appropriate plaintext (zero if element in Z or a random element otherwise), PK and corrupted parties' share of secret key.
5. $\mathcal{S}$ outputs the same as $\mathcal{A}$.

Fig. 16: Simulator for MPSI protocol (Case 1)

Simulator $\mathcal{S}$

1. **Key Generation**: $\mathcal{S}$ generates $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{KeyGen}$ and invokes the simulator $\mathcal{S}_{\mathsf{Gen}}(\mathrm{PK})$ for π_{Gen}.
2. **Commitment Phase**: $\mathcal{S}$ commits to all 0 for P_1 commitment and executes a fake proof of knowledge by invoking the simulator $\mathcal{S}_{\mathrm{EXP}}$.
3. **Aggregation**:
   (a) For every honest party $i \in \mathcal{H}$ where $\mathcal{H}$ is set of all honest parties, $\mathcal{S}$ generates a random polynomial $C_i(\cdot)$ of degree m_i, encrypts this random polynomial by encrypting the coefficients on behalf of P_i.
   (b) $\mathcal{S}$ generates the commitment $\mathsf{com}_{\mathbf{C}}$ by invoking the simulator $\mathcal{S}_{\mathsf{PCOM}}$ and sends to the adversary on behalf of P_1.
   (c) $\mathcal{S}$ participates honestly in π_{COIN} on behalf of the honest parties.
   (d) Upon receiving u as the output of π_{COIN}, $\mathcal{S}$ invoke the simulator $\mathcal{S}_{\mathsf{PCOM}}$ to pass the verification.
   (e) $\mathcal{S}$ sends a ciphertext and invokes π_{EXP} honestly on behalf of the honest parties where the input polynomial used is the same polynomial which was encrypted.
   (f) $\mathcal{S}$ extracts the corrupt parties' inputs. This is achieved by extracting $d+1$ evaluation points of every corrupted party's polynomial. In more detail, each corrupt party sends a ciphertext together with a proof of knowledge π_{EXP}, where these ciphertexts are generated by evaluating the input polynomials on the evaluation point u. Upon receiving the ciphertexts and the proofs, $\mathcal{S}$ extracts the plaintexts by running the simulation-extractor E^{DL}. This process is repeated for every corrupted party multiple times. Namely, to extract the input polynomial of each corrupted party, $\mathcal{S}$ rewinds the adversary to the beginning of π_{COIN} in order to generate a new random element v. $\mathcal{S}$ records $d+1$ such evaluations for each corrupted party and uses them to interpolate each input polynomial and extract its roots. Let the input set extracted for a corrupt party P_i denoted by X_i. $\mathcal{S}$ also invokes the simulator of π_{DecZero} on behalf of the honest parties.
4. Concluding the intersection:
   (a) $\mathcal{S}$ sends X_i as an input of corrupt party P_i to the trusted party and receives set intersection Z from the trusted party.
   (b) For every $x_1^j \in Z$ where x_1^j is j^{th} element in set X_1, $\mathcal{S}$ biased the decryption to be zero and random element otherwise. This is achieved by invoking the simulator $\mathcal{S}_{\mathrm{DecZero}}$ on the appropriate plaintext (zero if the element in Z), PK and corrupted parties' share of the secret key.
5. $\mathcal{S}$ outputs the same as $\mathcal{A}$.

Fig. 17: Simulator for MPSI protocol (Case 2)