PDA: Semantically Secure Time-Series Data Analytics with Dynamic Subgroups

Taeho Jung\(^1\), Junze Han\(^1\), Xiang-Yang Li\(^{1,2}\)

\(^1\)Department of Computer Science, Illinois Institute of Technology, Chicago, IL
\(^2\)Department of Computer Science and Technology, TNLIST, Tsinghua University, Beijing

Abstract—Third-party analysis on private records is becoming increasingly important due to the widespread data collection for various analysis purposes. However, the data in its original form often contains sensitive information about individuals, and its publication will severely breach their privacy. In this paper, we present a novel Privacy-preserving Data Analytics framework PDA, which allows a third-party aggregator to obliviously conduct many different types of polynomial-based analysis on private data records provided by a dynamic sub-group of users. Notably, every user needs to keep only \(O(n)\) keys to join data analysis among \(O(2^n)\) different groups of users, and any data analysis that is represented by polynomials is supported by our framework. Besides, a real implementation shows the performance of our framework is comparable to the peer works who present ad-hoc solutions for specific data analysis applications. Despite such nice properties of PDA, it is provably secure against a very powerful attacker (chosen-plaintext attack) even in the Dolev-Yao network model where all communication channels are insecure.

Index Terms—Privacy-preserving Computation, Multi-party Computation, Secure data analysis

1 INTRODUCTION

Great amount of data is collected for the information-based decision making in our real life these days (Smart Grid [25], Social Network Services [19], Location Based Services [36] etc.). This trend originated from the business needs. Enterprises have been increasingly looking to discover the answers to specific business questions from the data they possess, and various data mining techniques are applied to derive the answers (e.g., marketing [8] and health care [39]). Driven by such needs, publication, sharing and analysis of user-generated data has become common. For example, President Information Technology Advisory Committee (PITAC) released a report to establish a nation-wide electronic medical records sharing system to share the medical knowledge for various clinic decision making [20]; AOL published its anonymized query logs for research purpose [3]; Netflix released anonymized dataset containing 500,000 subscribers’ movie ratings to organize a public contest for the best recommendation algorithm [7]. However, user-generated data in its original form contains much sensitive information about individuals (2, 24), and the publication of such data has brought active discussions in both industry and academia [15, 18, 45]. Current primary practice is to rely on 1) sanitization to clean up the sensitive information in the publication [33], or 2) regulation enforcement to restrict the usage of the published data [34, 37]. The former approach may bring distortion to the data during the sanitization which leads to accuracy loss, and the latter one solely relies on the legal or voluntary agreements that the data should be used only for agreed purposes. The drawback of those protections is two folds. 1) A good trade-off between the utility and the privacy is hard to find; 2) Even if data recipients do not misbehave on the data, such protection does not guarantee the protection, in the sense that a profitable sensitive data may become the target of various types of attacks, and the contracts or agreements cannot guarantee any protection in this case.

Various previous works focus on the data anonymization (46, 40) to protect the identities of the data owners, but the de-anonymization (26, 29) is its powerful countermeasure, and it cannot provably guarantee individual privacy. In this paper, we focus on a more substantial aspect of privacy implications in the third-party data analysis – the confidentiality of the data itself. We investigate the privacy preserving data analysis problem in which a third-party aggregator wants to perform data analysis on a private dataset generated by any subgroup of total users in the system, where the total users in the system may change dynamically. To enable this without privacy concerns, it is desirable to have a data analysis tool which can provably guarantee the data privacy without bringing distortion to the analysis result such that the aggregator will get the most utility out of it. Various types of solutions based on different techniques are proposed to support such privacy preserving computation (secure multi-party computation, functional encryption, perturbation etc., all reviewed in Section 8), but we have the following design goals to make our scheme applicable in a realistic environment, and most of existing solutions fail to achieve one or more goals.

Design Goals

Varieties: Two varieties need to be considered. In reality, the actual group of users generating data changes rapidly along the time, and the type of analysis itself also changes depending on the characteristics of the dataset. Therefore, a scheme needs to adapt itself to such two varieties.
Time-series Data: Users continuously generate data, therefore the data analysis scheme should preserve the data confidentiality even when the data is a time series and may be substantially self-correlated.

Communication & Computation Overhead: Considering the rate at which data is generated and collected these days, the analysis on the published data should be efficient in terms of both computation and communication overhead.

Channel Security: In dynamic environments such as VANET, establishing pair-wise secure channels is almost impossible due to the huge overhead. Therefore a scheme cannot rely on secure channels to achieve privacy.

Privacy requirement Independent Accuracy: Distorted result is not accepted in many cases, therefore we require accurate result with zero or negligible distortion from the published data, which is not a function of the guaranteed privacy level. This property will enable us to guarantee the proposed security (IND-CPA) all the time, hence trade-off privacy level. This property will enable us to guarantee the proposed security (IND-CPA) all the time, hence trade-off privacy level. This property will enable us to guarantee the proposed security (IND-CPA) all the time, hence trade-off privacy level.

In this paper, we design a framework for Privacy-preserving Data analytics (PDA), a cryptographic tool which enables third-party analysis on private data records without privacy leakages. The main contributions of this paper are summarized as follows.

1) Instead of presenting ad-hoc solutions for many applications, PDA enables any type of data analysis that is based on polynomials (e.g., statistics calculation, regression analysis, and support vector machine classification) to be performed on a private dataset distributed among individuals.

2) We propose a provably secure cryptographic framework which guarantees semantic security against chosen-plaintext attack. Specifically, any polynomial adversary’s advantage is polynomially bounded by the advantage of solving a Decional Diffie-Hellman problem.

3) Although we support many different formats of polynomial-based data analysis among any subset of \( n \) users, every user needs to hold only \( O(n) \) keys, and the computation & communication complexities for adding an extra user to existing group of \( n \) users are both \( O(n) \) while removing an existing user does not require extra operation. Besides, on average the communication overhead for an ordinary user is less than 1 KB per product term in the polynomial.

2 Preliminaries

2.1 Notations

We use \( \mathbb{Z}_N \) to denote the additive group \( \{0, 1, \ldots, N - 1\} \), and \( \mathbb{Z}_N^* \) to denote the invertible residues modulo \( N \). Also, \( |G| \) and \( |P| \) are used to denote the order of group \( G \) and the size of the set \( P \) respectively. Recall that, given the conventional notational notation of Euler’s totient function \( \phi(N) \), \( |\mathbb{Z}_N^*| = \phi(N) \). When demonstrating the reducibility between the computation problems \( X, Y \), we use \( X \leq_p Y \) to denote that \( X \) is polynomially reducible to \( Y \), and \( X \equiv Y \) to denote that \( X \) is polynomially equivalent to \( Y \).

**Definition 1.** \( k \)-th Decisinal Diffie-Hellman (\( k \)-DDH) problem in the group \( \mathbb{Z}_N^* \) with generator \( g \) is to decide whether \((g^k)^{\omega} = (g^a)^b \) given a triple \((g^a)^k, (g^b)^k, (g^c)^k\), and any algorithm \( \mathcal{A} \)'s advantage in solving the \( k \)-DDH problem is denoted by \( \text{adv}_{k, \mathcal{A}}^{\text{DDH}} \).

**Definition 2.** A number \( w \) is called \( N \)-th residue modulo \( N^2 \) if there exists \( y \in \mathbb{Z}_{N^2}^* \) such that \( w = y^N \mod N^2 \).

Recall that the set of \( N \)-th residues modulo \( N^2 \) is cyclic multiplicative subgroup of \( \mathbb{Z}_{N^2}^* \) of order \( \phi(N) \). Every \( N \)-th residue has \( N \) roots, and specifically, the set of all \( N \)-th roots of the identity element forms a cyclic multiplicative group \( \{1 + N\}^2 = 1 + xN \mod N^2 | x \in \mathbb{Z}_N \} \) of order \( N \).

**Lemma 1.** There exists a function \( D : \{(1 + N)^x \mod N^2 | x \in \mathbb{Z}_N \} \rightarrow \mathbb{Z}_N \) which can efficiently solve the discrete logarithm within the subgroup of \( \mathbb{Z}_{N^2}^* \) generated by \( g = 1 + N \).

**Proof.** For any \( y = (1 + N)x \mod N^2 \). Then, the function \( D(y) = (y - 1)/N \mod x \) efficiently solves the discrete logarithm within the subgroup, where \((y - 1)/N\mod x\) is the quotient of the integer division \((y - 1)/N\mod x\) rather than \( y - 1 \) times the inverse of \( N \).

**Definition 3.** Decisional Composite Residuosity (DCR) problem in the group of \( N \)-th residues modulo \( N^2 \) with generator \( g \) is to decide whether a given element \( x \in \mathbb{N}N_{N^2} \) is an \( N \)-th residue modulo \( N^2 \).

2.2 Computational Assumptions

\( N \) is an RSA modulus of two safe prime numbers \( p, q \) (\( N = pq \) and \( p, q \) satisfy \( p = 2p' + 1, q = 2q' + 1 \) for some large prime numbers \( p', q' \) [14]). Then, the DDH assumption states that if the factorization of \( N \) is unknown, the DDH problem in \( \mathbb{Z}_N^* \) is intractable [10, 35]. That is, for any probabilistic polynomial time algorithm (PPTA) \( \mathcal{A} \), \( \text{adv}_{\mathcal{A}}^{\text{DDH}} \) is a negligible function of the security parameter \( \kappa \).

**Lemma 2.** When \( N \) is a semiprime of two safe prime numbers \( p, q \) and the factorization \( N = pq \) remains unknown, we have \( \text{DDH} \leq_p \text{k-DDH} \).

**Proof.** Given an oracle solves \( k \)-DDH problem, if one knows \( k \), he can submit \(( ((a^k)^k, (g^b)^k, (g^c)^k) \) to this oracle. \( g^c = g^b \mod N \) if and only if the oracle outputs 1.

Under the same condition, the DCR assumption states that the DCR problem in \( \mathbb{Z}_{N^2}^* \) is believed to be intractable if the factorization of \( N = pq \) is unknown [52]. That is, for any PPTA \( \mathcal{A} \), \( \text{adv}_{\mathcal{A}}^{\text{DCR}} \) is a negligible function of the security parameter. In this paper, we rely on the above assumptions to define and prove the security.

3 Problem Definitions

3.1 Problem Modelling

W.l.o.g., we assume there are total \( n \) users, and each of them is represented by an integer ID \( \{1, \ldots, n\} \). Each user generates time-series data records \((x_i, t) \) at time \( t \), but we use \( x_i \) to denote his data value for visual comfort. A third-party aggregator wishes to choose an arbitrary subset of users...
\[ P \subseteq \{1, \ldots, n\} \] to conduct certain data analysis on their private data records by evaluating:

\[ f(x_P) = \sum_k c_k \left( \prod_{i \in P} x_{ik}^{e_{ik}} \right) \tag{1} \]

where \( x_{ik} \) is user \( i \)'s private value for the \( k \)-th product term \( \prod x_{ik}^{e_{ik}} \), and we call \( P \) as the participants of the polynomial evaluation. Note that Eq. (1) can be used to represent any general multivariate polynomial because the coefficients \( c_k \)'s and the exponents \( e_{ik} \)'s are tunable parameters.

To model the time-variation, we use a discrete time domain \( T = \{t_1, t_2, \ldots\} \) of time slots, and when the aggregator submits the query polynomial to users, he will also declare a sequential time window \( T_f \subset T \) associated with the polynomial in order to request users’ data generated during the time window \( T_f \).

### 3.2 Informal Threat Assumptions

Due to privacy concerns, the analytic result should be only given to the aggregator, and any user’s data should be kept secret to anyone else but the owner unless it is deducible from the polynomial value. Besides, both users and the aggregator are assumed to be passive and adaptive adversaries. They are passive in that they will not tamper the correct computation of the protocol. If users tamper the protocol, it is highly likely that the aggregator will detect it since the outcome of the protocol will not be in a valid range (e.g., mean age = 3,671.9) due to the cryptographic operations on large integers, and the aggregator is interested in the correct analytic result, so they will not tamper the protocol itself. Furthermore, users may report a value with small deviation such that the analytic result still 'looks' good due to many reasons (moral hazards, unintentional mistake, etc.), but evaluating the reliability of the reported value is out of this paper’s scope. On the other hand, they are adaptive in that they may produce their public values adaptively after seeing others’ public values (i.e., passive rushing attack [30]). Besides, we also assume the users will not collude with each other or the aggregator in our basic construction, and this assumption will be relaxed in the advanced version in the later section.

As aforementioned, we assume the Dolev-Yao network model. That is, all the communication channels are open to anyone, and anyone can overhear and synthesize any message. This is a reasonable assumption since in a dynamic network environment in which actual group of available users may change (e.g., sensor networks, crowdsourcing environment), establishing pairwise secure communication channels is costly and may not be possible due to limited resources.

### 3.3 Privacy-preserving Data Analytics: PDA

To formally define the correctness and the security of our framework, we first present a precise definition of PDA.

**Definition 4.** Our Privacy-preserving Data Analytics (PDA) is the collection of the following four polynomial time algorithms: Setup, KeyGen, Encode, and Aggregate.

\[ \text{Setup}(\lambda^*) \rightarrow \text{params} \]

\[ \{ \text{EK}_i \}_{i=1}^n \rightarrow \text{KeyGen}(\text{params}) \]

\[ \forall k, \forall i : \text{C}(x_{ik}) \leftarrow \text{Encode}(x_{ik}, f(x_P), \text{EK}_i, T_f) \]

\[ \text{Aggregate}(\{ \text{C}(x_{ik}) \mid \forall k, \forall i \in P \}) \rightarrow f(x_P) = \sum(c_k \prod x_{ik}^{e_{ik}}) \]

where the probabilities are taken over all random bits consumed by the probabilistic algorithms.

The security of PDA is formally defined via following data publishing game.

**Data Analysis Game:**

**Setup:** 3 disjoint time domains are chosen: \( T_1 \) for phase 1, \( T_2 \) for phase 2, and \( T_r \) for the challenge phase.

**Init:** The adversary declares his role in the PDA scheme (i.e., aggregator or user), and the challenger controls the remaining participants. Both of them engage themselves in the Setup and KeyGen algorithms.

**Phase 1 in \( T_1 \):** The adversary submits polynomially many calls to the encode oracle \( f(x_P) \) (based on arbitrary query \( f(x), T_f \subset T_1, \{x_{ik}\}_{i \in P, k} \)). If the declared time windows do not overlap with each other, he receives all corresponding communication strings from the oracle. Otherwise, he receives nothing.

**Challenge in \( T_r \):** The adversary declares the target polynomial \( f^*(x_P) \) to be challenged as well as its time window \( T_{f'} \subset T_r \). Then, he submits two sets of values \( x_{P_1}, x_{P_2} \), such that \( f^*(x_{P_1}) = f^*(x_{P_2}) \), to the challenger. The challenger flips a fair binary coin \( b \) and generate the corresponding communication strings based on \( x_{P,b} \), which are given to the adversary.

**Phase 2 in \( T_2 \):** Phase 1 is repeated adaptively, but the time window \( T_f \) should be chosen from \( T_2 \).

**Guess:** The adversary gives a guess \( b' \) on \( b \).

The advantage of an algorithm \( A \) in this game is defined as

\[ \text{adv}_{\text{PDA}} = | \text{Pr}[b' = b] - \frac{1}{2} | \]

1. The oracle generates and gives the communication strings at the adversary’s request based on the queried polynomial \( f(x_P) \), its associated time window \( T_f \), and the input values \( \{x_{ik}\}_{i \in P, k} \). Challengers can act as such encode oracle in this game for the users that are not under adversary’s control.
only, and the choice will be monitored by the users since the coefficients and powers are public.

**Definition 6.** Our PDA is indistinguishable against chosen-plaintext attack (IND-CPA) if all polynomial time adversaries’ advantages in the above game are of a negligible function w.r.t. the security parameter $\kappa$ when $T_1, T_2,$ and $T_e$ are three disjoint time domains.

Note that our security definition does not guarantee that the private values cannot be deduced from the output values. Even a perfectly secure black-box algorithm cannot guarantee it if the same set of $k$ values are fed as inputs for $k$ different polynomials’ evaluation (a simple linear system). Rather, we use IND-CPA to define the security which is shown to be equivalent to the semantic security [36]. In other words, our framework guarantees that any adversary’s knowledge (either adversarial aggregator or user) on the probability distribution of the private values remains unchanged even if he has access to all the communication strings generated by our PDA. The indistinguishability becomes particularly crucial when the data is generated continuously along the time, and the same user’s data may be temporally correlated. Unless guaranteed by the semantic security, the data privacy is hardly preservable.

## 4 Achieving Correct & Secure PDA

In this section, we first present a basic construction without colluding users for simplicity, and further presents the countermeasure to collusion attacks in Section 7.

### 4.1 Intuitions

The following two steps show the big sketch of our design on the oblivious polynomial evaluation.

**Hiding each value:** To hide every private value $x_{ik}$ in the polynomial $f(x^P)$ (Eq. 4), we let each user $i$ publish the perturbed data $x_{ik}r_{ik}$, where $r_{ik}$ is an independently chosen random noise, but it satisfies that $\prod_{i\in P} r_{ik} = 1$ with modulo operation using our novel secret sharing technique. It follows that the product of all perturbed data is equal to $\prod_i x_{ik}$ for all $k$.

**Hiding each product term:** To hide the product terms in Eq. (4), we employ two users with special roles. The first special user $i_1$ will publish homomorphically encrypted perturbed data $E(x_{1k}r_{1k})$ instead of his perturbed data using the aggregator’s key. Since the plaintext of this ciphertext is a perturbed data, the aggregator does not learn $x_{1k}$. On the other hand, because only the aggregator has the private key of the ciphertext, no one else learns $x_{1k}$ or $\prod_i x_{ik}$ either.

The second user $i_2$ does not publish his perturbed data either. Instead, he conducts homomorphic operations:

$$E(x_{1k}r_{1k})\prod_i x_{ik} = E(\prod_i x_{ik})$$

for all $k$, where $E(m)$ denotes the ciphertext of $m$ encrypted with a homomorphic encryption scheme. Then, he locally picks random numbers $K_k$ such that $\prod_k K_k = 1$ (modulo operation), and publishes perturbed ciphertexts $K_k E(\prod_i x_{ik})$ to the aggregator. The product of them is equal to $E(\sum_k \prod_i x_{ik})$, and only the aggregator can decrypt it because other users do not have the private key. On the other hand, the aggregator cannot obtain values of any single product term because the ciphertexts are perturbed.

Letting the users independently obtaining random noises $r_{ik}$ such that $\prod_{i\in P} r_{ik} = 1$ is the most challenging part. To do so, we exploit the sequential time window $T_f$ associated with the polynomial $f(\cdot)$. All users will pick the $k$-th time slot $t_k$ in $T_f$ to generate $r_{ik}$ for his $x_{ik}$. Since the random noise is based on $t_k$, we need different time slots for all product terms. This implies we have to consume $T_f$ of size $m$ for a polynomial $f$ with $m$ product terms. Also, all sequential time windows for the evaluation must be disjoint to each other to guarantee that no time slots are used for twice.

### 4.2 Constructions

Firstly, we define the Lagrange coefficient for $i \in \mathbb{Z}_N$ and a set, $P_i$, of integers in $\mathbb{Z}_N$, as $L_{i,P}(x) = \prod_{j\in P} \frac{x-j}{i-j}$, and use the simplified Lagrange coefficient $L_{i,P}(0)$ for a special case $L_{i,P}(0)$ for the visual comfort.

Then, our provably secure framework PDA works as follows. Note that modular operations are omitted unless necessary for the sake of simplicity.

**Setup**

A crypto server picks two safe semiprimes $N = pq$, $\tilde{N} = \tilde{p}\tilde{q}$ such that $p, q$ are of same length, $\tilde{p}, \tilde{q}$ are of same length $\kappa$ bits, and $\phi(N)/\tilde{N} = k$ is an integer. Then, he randomly picks a generator $g$ of the group $\mathbb{Z}_N^\ast$, a generator $\tilde{g}$ of $\mathbb{Z}_{\tilde{N}}^\ast$, and sets $h = g^k$. Subsequently, he randomly chooses a hash function $H : \mathbb{Z} \rightarrow (h)$, which is modelled as a random oracle. Finally, he publishes $\text{params} = (g, \tilde{g}, N, \tilde{N}, H)$ and destroys $p, q, \tilde{p}, \tilde{q}$ such that they are no longer available.

**Lemma 3.** $h^{\tilde{N}} = 1 \mod N$.

**Proof.** $h^{\tilde{N}} = g^{N\tilde{N}} = g^{\phi(N)} = 1 \mod N$.

Any party independent from aggregator or users is qualified as the crypto server, for example, Internet Service Providers or governments.

**Key Generation**

**Aggregator’s Key Generation:** The aggregator generates a pair of additive homomorphic encryption keys $PK, SK$, and publishes $PK$ only. This additive homomorphic encryption must have the following properties:

$$E_{PK}(m_1) \cdot E_{PK}(m_2) = E_{PK}(m_1 + m_2)$$

$$E_{PK}(m_1)^{m_2} = E_{PK}(m_1m_2)$$

where $E_{PK}(m)$ denotes the ciphertext of $m$ encrypted with $PK$. Any additive homomorphic encryption with message space $\mathbb{Z}_N$ is accepted (e.g., Paillier’s cryptosystem [52]).

**Users’ Key Generation:** All users participate in the following secret sharing.

1. Every user $i$ ($i \in \{1, \cdots, n\}$) randomly picks $r_i \in \mathbb{Z}_{\tilde{N}}^\ast$, and broadcasts $y_i = g^{r_i} \in \mathbb{Z}_N^\ast$ via insecure channel.

2. Then, every user $i$ uses $y_{i+1}$ and $y_{i-1}$ to locally calculate $Y_i = (y_{i+1} \cdot y_{i-1}^{-1})^{r_i} \in \mathbb{Z}_N^\ast$, where the inverse is over modulo $\tilde{N}$. Specially, user 1 uses $y_2$ and $y_{n-1}$, and user $n$ uses $y_1$ and $y_{n-1}$.
Lemma 4. \( \prod_{i=1}^{n} Y_i = 1 \mod \tilde{N}, \prod_{i=1}^{n} Y_i^N = 1 \mod \tilde{N}^2 \)

**Proof.**
\[
\prod_{i=1}^{n} Y_i = g^{(r_2-r_n)r_1} \cdots g^{(r_1-r_{n-1})r_n} = g^0 = 1 \mod \tilde{N}
\]

Then, \( \prod_{i=1}^{n} Y_i = 1 + m\tilde{N} \) for some integer \( m \), and \( \prod_{i=1}^{n} Y_i^N = (1 + m\tilde{N})^N = 1 \mod \tilde{N}^2 \) as in Section 2.1.

Subsequently, every user \( j \) randomly defines a polynomial \( q_j^{(d)}(x) \) over \( \mathbb{Z}_r^* \), whose degree is \( d \) and the constant term is 0 (i.e., \( q_j^{(d)}(0) = 0 \)). That is, every user randomly picks the coefficients for its own polynomial from \( \mathbb{Z}_r^* \). This is repeated for all \( d = 2, \cdots, n - 1 \). Then, each user \( i \) executes the encoding key query (Sub-Algorith 1), in which all other users jointly interact with him to respond to the query. The query outputs the specific data points on the secret global polynomials \( q^{(2)}(x), \cdots, q^{(n-1)}(x) \) for each user, where the global polynomial is the sum of all individual polynomials.

**Sub-Algorith 1 EK, Query**

1. Every user \( j \neq i \) computes and publishes:
   \[
   Q_{ij} = Y_j^{\tilde{N}} (1 + \tilde{N}) q_j^{(d)}(i) \mod \tilde{N}^2
   \]

2. After seeing all the broadcasts, user \( i \) calculates:
   \[
   Y_i^{\tilde{N}} (1 + \tilde{N}) q_i^{(d)}(i) \prod_{j \neq i} Q_{ij} = (1 + \tilde{N})^{\sum_{j=1}^{n} q_j^{(d)}(i)} \mod \tilde{N}^2
   \]

Subsequently, he uses aforementioned discrete logarithm solver \( D(\cdot) \) to solve \( q^{(d)}(i) = \sum_{j} q_j^{(d)}(i) \mod \tilde{N} \) from it, which is set as one of his encoding keys. Later, \( q^{(d)}(i) \) will be used to generate the data masker when \( d + 1 \) users’ data is to be analyzed.

Step 1 & 2 are repeated for \( d = 2, \cdots, n - 1 \).

**Output:** \( \text{EK}(q^{(2)}(i), \cdots, q^{(n-1)}(i)) \)

Essentially, in the user’s key generation, each user \( j \) randomly generates polynomials \( q_j^{(2)}(x), \cdots, q_j^{(n-1)}(x) \) to respond others’ key queries, and each key recipient \( i \) receives one data point for each secret polynomial \( q_j^{(d)}(i) \), where the polynomial \( q^{(d)}(x) \) remains unknown to everyone because it is merely the sum of all users’ random polynomials. However, specific point on this unknown polynomial can be jointly computed as shown in the query.

Lemma 5. For any set of users \( \mathcal{P} \), we have:
\[
\sum_{i \in \mathcal{P}} q_i^{(|\mathcal{P}|-1)(i)} L_{i, \mathcal{P}} = 0 \mod \tilde{N}
\]

**Proof.** Recall that the constant terms in all polynomials are 0. Then, due to the polynomial interpolation, the above sum equals \( q_i^{(|\mathcal{P}|-1)(0)} = 0 \mod \tilde{N} \).

**Input Encoding**

Recall that we employ two users with special roles. For the simplicity of the presentation, we assume that the first two users (user 1 & 2) have to participate in the polynomial evaluation and act as the special users, but any other ordinary user can replace them.

After the aggregator declares the target polynomial \( f(x_P) = \sum_k c_k \left( \prod_i x_{ik}^{e_{ik}} \right) \) as well as the corresponding time window \( T_f \), all users examine the time window. If \( T_f \) overlaps with any of the consumed time windows, all users abort. Otherwise, every user \( i \in \mathcal{P} \) except user 1 & 2 computes and publishes his encoded value \( C(x_{ik}) \) for all \( k \):
\[
\forall k : C(x_{ik}) = x_{ik}^{e_{ik}} H(t_k)^{(\prod_{i \in \mathcal{P}})^{-1}}(i) L_{i, \mathcal{P}} \in \mathbb{Z}\tilde{N}^3
\]

where \( t_k \) is the \( k \)-th time slot in \( T_f \). At the same time, user 2 computes and publishes:
\[
\forall k : V_k = E_{PK}(C(x_{2k}^{e_{2k}})) \prod_{i \in \mathcal{P}, i \neq 2} C(x_{ik}^{e_{ik}}) = E_{PK}\left( \prod_{i \in \mathcal{P}} x_{ik}^{e_{ik}} \right)
\]

Subsequently, he randomly selects \( K_k \in \mathbb{Z}_{\tilde{N}^*}^* \) such that \( \prod_k K_k = 1 \mod N^2 \), and he publishes \( K_k V_k^{e_{2k}} \) for all \( k \).

Even if a user \( i \)’s \( x_{ik} \) does not appear in a product term \( \prod x_{ik}^{e_{ik}} \), he still sets \( x_{ik} = 1 \) and participates in the analysis by following the algorithm to guarantee the correctness of the final aggregation.

**Aggregation**

When all users finish their encoding and broadcasting, the aggregator conducts the following homomorphic operations:
\[
\prod_k K_k V_k^{e_{2k}} = E_{PK}\left( \sum_k c_k \left( \prod_{i \in \mathcal{P}} x_{ik}^{e_{ik}} \right) \right) \mod N^2
\]

Then, he can use \( SK \) to decrypt the polynomial value.

**4.3 Supporting Dynamic User Group**

Up to this point, our design is able to support data analysis in dynamic subgroup of a fixed group of \( n \) users. A more substantial ability is to support a dynamic group of users, i.e., to enable new users to join the system and old users to leave the system.

**Adding a new user \( n + 1 \)**

When a new user whose ID is \( n + 1 \) needs to join the group of users \( \{1, \cdots, n\} \), the number of participants in one round of data analysis can be as large as \( n + 1 \). The first step to do is to update the key set of every user \( i \) in the old group \( \{1, \cdots, n\} \): each user \( i \) needs to acquire and add one extra

3. \( C(x_{ik}) \) may not be in \( \mathbb{Z}_{\tilde{N}} \) when \( x_{ik} \) is not invertible under modulo \( N \), but this occurs only when \( x_{ik} \) is not co-prime to \( n, i.e., x_{ik} \) is equal to \( p \) or \( q \), two large safe prime numbers. The chance this happens is \( \frac{1}{\Phi(n)} \), and this is also the case where the large RSA number \( n \) is factorized. We simply ignored this chance because it is a negligible function of the security parameter, and it is also commonly believed that the factorization is intractable.
item $q^{(n)}(i)$ into his key set $E_K$ for the worst case where all of $n + 1$ users participate in the analysis (so that their $q^{(n)}(i)$’s can be used when such case occurs). The second step is to issue the new user $n + 1$ his full key set $E_{K_{n+1}} = \{q^{(2)}(n + 1), \ldots, q^{(n-1)}(n + 1), q^{(n)}(n + 1)\}$ so that he can participate in any data analysis consisting of $2, 3, \ldots, n + 1$ participants.

For the first step, all users in the new group perform the aforementioned secret sharing in the Key Generation, and every user $j \in \{1, \ldots, n + 1\}$ randomly defines $q^{(n)}(x)$ whose degree is $n$ and the constant term is 0. Then, for each user $i \in \{1, \ldots, n + 1\}$, every user $j \neq i$ computes and publishes $Q_{ij}$ of degree $n$ as in Sub-algorithm 1 and $i$ calculates $q^{(n)}(i)$ in the same way. When this is repeated for all users, every user $i$ in the old group $\{1, \ldots, n\}$ has a complete key set which contains key items $q^{(2)}(i), \ldots, q^{(n)}(i)$, and the new user $n + 1$ only has $q^{(n)}(n + 1)$.

For the second step, every user $j \neq i$ performs the following process for $d = 2, \ldots, n - 1$. User $j$ re-uses his previously defined individual polynomial $q^{(d)}(x)$ to calculate $q^{(d)}(n + 1)$ and publishes $Q_{n+1,j}$ of degree $d$ as in Sub-algorithm 1. Subsequently, user $n + 1$ calculates the product and acquires the $q^{(d)}(n + 1)$ in the same way. When this is repeated for $d = 2, \ldots, n - 1$, the new user acquires $q^{(2)}(n + 1), \ldots, q^{(n-1)}(n + 1)$.

Finally, $2n + 1$ new items are shared among $n + 1$ users, and every user $i$ in the new group has his own $n + 1$ items in the key set $E_K$, which is the full key set for the new group of users $\{1, \ldots, n + 1\}$.

**Adding $n'$ new users**

Batch arrival can be handled in a similar way. When $n'$ new users arrive at the same time, one round of secret sharing occurs in the new group of $n + n'$ users. As the first step, $n'$ extra key items $q^{(n)}(i), \ldots, q^{(n')}(n + n' - 1)(i)$ are delivered to every $i \in \{1, \ldots, n + n'\}$. Then, as the second step, every new user $i \in \{n + 1, \ldots, n + n'\}$ acquires $q^{(2)}(i), \ldots, q^{(n-1)}(i)$ as aforementioned.

**Removing one or more existing users**

When one or more users are removed or leave the group, the system can simply ignore the removed users, and keys of the rest users are not affected by the removal.

### 4.4 Representing Real Numbers

Due to the cryptographic operations, all the computations and operations in our scheme are closed in integer groups, and thus we need to use integers to represent real numbers.

We exploit the homomorphism in PDA to represent real numbers via integers using the fixed point representation $[51]$. Given a real number $x$, its fixed point representation is given by $[x] = [x \cdot 2^e]$ for a fixed $e$, and all the arithmetic operations reduce to the integer versions as follows:

- **Addition/Subtraction**: $[x \pm y] = [x] \pm [y]$
- **Multiplication/Division**: $[x \cdot y^\pm 1] = [x] \cdot [y]^\pm 1 \cdot 2^{\mp e}$ where $x \cdot 2^{-1}$ stands for $x$ divided by 2. Then, we have the following conversions for the arithmetic operations used in our schemes if using fixed point representations.
- $C([x])C([y]) \Rightarrow C([x])C([y]) \cdot 2^{-e}$.

$x \cdot [2^{-1}] = [x] \cdot 2^{-e}$.

$C([x])C([y]) \Rightarrow C([x])C([y]) \cdot 2^{-e}$.

- $E_{PK}(|[x]|)^y \Rightarrow E_{PK}(|[x]|)^y \cdot 2^{-e}$

where $C(m)$ is the encoded communication string of $[m]$ (Encode algorithm), and $E_{PK}(m)$ denotes the ciphertext of $m$ encrypted with $PK$ (Key Generation algorithm). We do not use floating point representation because it is hard to implement the arithmetic operations on floating point numbers. Although this conversion gives practically high precision, one may consider employing $[5, 52]$ for much higher precisions.

### 4.5 Data Analysis via PDA: Examples

We only present some examples in this section since infinitely many different analysis are supported and it is not possible to enumerate all of them here, but any data analysis based on polynomials can be conducted via our framework.

**Statistics Calculation**: Various statistical values including, but not limited to mean, variance, sample skewness, mean square weighted deviation are all polynomials of the data values, and all of them can be privately evaluated by a third-party aggregator without knowing individual data by leveraging our PDA.

**Regression Analysis**: Regression analysis is a statistical process for estimating the relationship among the variables. Widely used polynomial regression and ridge regression both belong to it. In both, each user $i$’s data record is described as a feature vector $x$ and a dependent variable $y_i$, and training a regression model is to find $p$ which minimizes $MSE(p) = \sum_y (y_i - p(x_i))^2$, i.e., the linear predictor who predicts users’ dependent variable vector $y$ using their feature matrix $X$ with minimum mean squared error. Since $MSE(p)$ is convex, it is minimized if and only if $Ap = b$, where $A = X^T X$ and $b = X^T y$. Further, $A = \sum_i x_i x_i^T$, and $b = \sum_i y_i x_i$. Therefore, the aggregator can obliviously evaluate them with several calls to our PDA. Note that any polynomial regression model or ridge regression model can be trained in the same way.

**Support Vector Machine**: Support Vector Machine (SVM) is a supervised learning model which is widely used to classify the data records. Similar to the regression analysis, a user $i$’s data record is described as a feature vector $x_i$ and its class $y_i \in \{-1, 1\}$, and training a SVM is to find a vector $w$ and a constant $b$ such that the hyperplane $wx - b = 0$ separates the records having $y_i = -1$ from those having $y_i = 1$ with the maximum margin $\frac{2}{\|w\|}$. The dual form of this optimization problem is represented as maximizing $L(a)$ in $a = \{a_1, \ldots, a_n\}$.

$$L(a) = \sum_{i=1}^n a_i - \frac{1}{2} \sum_{i,j} a_i a_j y_i y_j x_i^T x_j$$

subject to $\sum_{i=1}^n a_i y_i = 0$ and $0 \leq a_i \leq C$. This is solved by the Sequential Minimum Optimization (SMO) algorithm $[53]$ with user-defined tolerance, which greedily seeks for the local maximum iteratively until convergence with certain error. Due to the equality constraint, $L(a)$ becomes a one-dimensional concave quadratic function in each iteration, which can be trivially optimized. The coefficients of the quadratic function are polynomials of all users’ feature vectors and class values, and the aggregator can obliviously calculate them using our framework and optimize $L(a)$ at each iteration locally.
5 Correctness & Security Proofs

5.1 Correctness Proof

Theorem 1. Our PDA is correct.

Proof. We show by steps that each algorithm correctly calculates the output, which guarantees the correctness of our scheme.

Key Generation: Correctness of the homomorphic encryption is already shown [52], therefore we focus on the user’s key generation. Due to Lemma 4 we have:

\[ Y_i \tilde{N} (1 + \tilde{N}) q_i^{(d)}(i) = (1 + \tilde{N}) \sum_{j=1}^{n} Y_j \tilde{N} (1 + \tilde{N}) q_j^{(d)}(i) \mod \tilde{N}^2 \]

Further, the existence of the discrete logarithm solver (Lemma 1) guarantees that the discrete logarithm will be correctly calculated.

Input Encoding

User 1 conducts the following computation in the input encoding:

\[ \forall j : V_j = E_{PK} (C(x_{i,j}^{e_j})) = E_{PK} \left( \prod_{i \in P} C(x_i^{e_{ij}}) \right) \]

The first two equations are true because of the additive homomorphism of the encryption \( E_{PK}(\cdot) \), and the third equation is true because:

\[ \sum_{i \in P} q_i^{(l)}(i) L_{i,p}(0) = 0 \mod \tilde{N} \quad \text{(Lemma 5)} \]

\[ \Rightarrow H(t_i)^0 = (h_i)^0 \mod N \quad \text{(Lemma 3)} \]

Correctness of the aggregation algorithm is obvious. In conclusion, after all Setup, KeyGen, Encode are executed, the output of Aggregate is \( f(x_F) \) with probability 1.

5.2 Security Proof

To prove the indistinguishability (IND-CPA) of our PDA, we first prove the following lemmas.

Lemma 6. In the KeyGen algorithm, \( Y_i \) is statistically indistinguishable from a uniformly random element chosen from \( \tilde{Z}_N \).

Proof. In the first secret-sharing phase, every user publishes \( y_i = g^{r_i} \) where \( r_i \) is randomly chosen, but since DDH problem is believed to be intractable in \( \tilde{Z}_N \) when the factorization of \( N \) is unknown, \( g^{r_i} \) is indistinguishable from a uniformly random element chosen from \( \tilde{Z}_N \) as shown in [10].

Lemma 7. In the KeyGen algorithm, any secret data point \( q_i^{(d)}(i) \) remains statistically indistinguishable from a uniformly random chosen element, and all the encoding keys are indistinguishable from uniform randomly chosen elements.

Proof. In the EK, Query (Sub-AlGORITHM 4) phase, every user except user \( j \) publishes \( Q_{ij} = Y_j \tilde{N} (1 + \tilde{N}) q_j^{(d)}(i) \). We show that \( Q_{ij} \) leaks no statistical information about the hidden random polynomial \( q_j^{(d)}(i) \) as follows.

We first prove that, given \( m \) and \( y = Y_j \tilde{N} (1 + \tilde{N})^m \) only (except \( Y_j \)), deciding whether \( m \) is the corresponding hidden value in \( y \) (denoted as D-Hidden problem) is equivalent to the DCR problem. We suppose there is an oracle \( O_{D-Hidden} \) which can solve the D-Hidden problem, and another oracle \( O_{DCR} \) who can solve the DCR problem.

D-Hidden \( \leq_P \) DCR: Since \( (1 + \tilde{N}) \) is an \( N \)-th non-residue, \( y \) is an \( N \)-th residue if and only if \( m = 0 \ mod N \).

\[ \text{Therefore, deciding whether } m \text{ is a hidden value in } y \text{ is exactly as hard as the DCR problem.} \]

Due to the semantic security of \( Q_{ij} \) against chosen-plaintext attack.

With the above lemmas, we are ready to present the proof for the following theorem.

Theorem 2. With the DDH assumption and DCR assumption, our PDA scheme is indistinguishable against chosen-plaintext attack (IND-CPA) under the random oracle model. Namely, for any PPTA, its advantage \( adv_{PDA} \) in the Data Analysis Game is bounded as follows:

\[ adv_{PDA} \leq \epsilon q_c + \frac{1}{q_c} \cdot adv_{DDH} \]

where \( \epsilon \) is the base of the natural logarithm and \( q_c \) is the number of queries to the encode oracle.

Proof. To prove the theorem, we present three relevant games Game 1, Game 2, and Game 3, in which we use \( A \) and \( B \) to denote the adversary and the challenger. For each \( l \in \{1, 2, 3\} \), we denote \( E_l \) as the event that \( B \) outputs 1 in the Game \( l \), and we define \( adv_l = |Pr[E_l] - \frac{1}{2}| \).

Game 1: This is the game exactly identical to the Data Publishing Game in Section 5.3. \( A \)'s encode queries \( f(x) \), \( T_f \), \{\( x_{ik} \)\}_{i,k} are answered by returning the encode values \( \{C(x_{ik})\}_{i,k} \). In the challenge phase, the adversary \( A \) selects a target polynomial \( f^*(x) \), the associated time window \( T^*_f \), and two set of values \( x_{p,1}, x_{p,2} \) which satisfy \( f^*(x_{p,1}) = f^*(x_{p,2}) \). Then, the challenger \( B \) returns the corresponding encoded values to the adversary \( A \). When the game terminates, \( B \) outputs 1 if \( b^* = b \) and 0 otherwise.

By the definition, \( adv_1 = |Pr[E_1] - \frac{1}{2}| = adv_{PDA} \).

Game 2: This game occurs after the Game 1 terminates. In Game 2, the adversary \( A \) and the challenger \( B \) repeat the
same operations as in Game 1 using the same time windows of those operations. However, for each encode query in Game 1 at time $t \in T_f \cup T_2$, the challenger $B$ flips a biased binary coin $\mu_{T_f}$ for the entire time window $T_f$ which takes 1 with probability $\frac{1}{Q - 1}_t$ and 0 with probability $\frac{q}{N}$. When the Game 2 terminates, $B$ checks whether any $\mu_{T_f} = 1$. If there is any, $B$ outputs a random bit. Otherwise, $B$ outputs 1 if $b' = b$ and 0 if $b' \neq b$. If we denote $F$ as the event that $\mu_{T_f} = 1$ for any $T_f$, the same analysis as in Section 2 shows that $\Pr[F] = \frac{1}{q + t}$. According to Lemma 4, Game 1 to Game 2 is a transition based on a failure event of large probability, and therefore we have $adv_2 = adv_2 \cdot \Pr[F] = \frac{adv_2}{q + t}$.

**Game 3:** In this game, the adversary $A$ and the challenger $B$ repeat the same operations as in Game 1 using the same time windows of those operations. However, there is a change in the answers to the encode queries $(f(x), T_f, \{x_{ik}\}, i, k)$. The oracle will respond to the query with the following encoded values:

$$\forall i, \forall k : C(x_{ik}) = \begin{cases} x_{ik}^{e_k} H(t_k) g^{(|\mathbb{P}|-1)(i) L_i, \sigma(0)} & \mu_{T_f} = 1 \\ x_{ik}^{e_k} (H(t_k)^s) & \mu_{T_f} = 0 \end{cases}$$

where $s$ is a uniform randomly chosen element from $\mathbb{Z}$. $\text{fix}_t$ for each polynomial $f(x)$, i.e., the same polynomial has the same $s$. When Game 3 terminates, $B$ outputs 1 if $b' = b$ and 0 otherwise.

Due to the Lemma 8 below, distinguishing Game 3 from Game 2 is at least as hard as a DDH problem for any adversary $A$ in Game 2 or Game 3. It follows then:

$$\Pr[E_2] - \Pr[E_3] = \Pr[E_2] - \Pr[E_3] = \left| \Pr[E_2] - \Pr[E_3] \right| = \left| \Pr[E_2] - \Pr[E_3] \right| = \left( \Pr[E_2] - 1/2 \right) \cdot \frac{q_c}{q_c + 1} = adv_2 \cdot \frac{q_c}{q_c + 1} \leq adv_2^{DDH}$$

Combining the above inequality with the advantages deduced from Game 1 and Game 2, we finally have:

$$adv_2 \cdot \frac{q_c}{q_c + 1} \leq adv_2^{DDH}$$

**Lemma 8.** Distinguishing Game 3 from Game 2 is at least as hard as solving a DDH problem for $A$.

**Proof.** The only difference between Game 2 and Game 3 is the oracle's answers to the encode queries. When $\mu_{T_f} = 1$, the answers in Game 3 are exactly same as the ones in Game 2. However, when the failure event $F$ occurs (i.e., $\mu_{T_f} = 0$), $B$ answers using $H(t)^s$ instead of $H(t)$. Since the values $x_{ik}$ is submitted as a query by $A$, $A$ knows the value of $x_{ik}$. Then, distinguishing Game 2 and Game 3 is equivalent to distinguishing the following two terms:

$$H(t)^s g^{(|\mathbb{P}|-1)(i) L_i, \sigma(0)} \text{ vs. } H(t)^{s} g^{(|\mathbb{P}|-1)(i) L_i, \sigma(0)}$$

Since $H(t)$ maps $t \in \mathbb{Z}$ to a cyclic multiplicative group $(h)$, we let $H(t) = h^z$ for some integer $x$. Let $a = x$ and $b = q^{(|\mathbb{P}|-1)(i) L_i, \sigma(0)}$. Given a triple $(h^a, h^b, h^c)$, deciding whether $h^c$ is $h^{ab}$ or a random element in $\mathbb{Z}_N$ is exactly the $k$-DDH problem. Since $s$ is a uniform random number, distinguishing between the above two terms is equivalent to solving this $k$-DDH problem. However, in Game 3, the adversary is only given $h^a$ except $h^b$, which means distinguishing Game 3 from Game 2 is at least as hard as the $k$-DDH problem.

Due to Lemma 8 the $k$-DDH problem is at least as hard as the DDH problem when $k$ is known. However, $k = \phi(N) / N$ is unknown to $A$, and therefore the $k$-DDH in our group is also at least as hard as the DDH problem for $A$.

In conclusion, distinguishing Game 3 from Game 2 is at least as hard a DDH problem for $A$.  

## 6 PERFORMANCE EVALUATION

### 6.1 Theoretic Performance: Storage & Communication

**Key Storage Complexity:** We only require every user to keep a set of encoding keys $EK_i = \{q_2(i), \ldots, q_{n-1}(i)\}$. Lagrangian coefficients $L_i, \sigma$ needed in Encode can be locally computed when $P$ of the analysis is declared. Therefore, the storage complexity is $O(n)$, where $n$ is the total number of users. To the best of our knowledge, there is no sub-linear solution under the same threat model and performance requirement (Section 5).

**Communication Overhead:** The communication overhead for evaluating a certain polynomial $f(x)$ in terms of transmitted bits is summarized in the Table 1. $\kappa$ is the bit-length of the safe prime numbers used in our scheme, $n'$ is the number of new users added into the system, $m$ is the total number of product terms in the polynomial (Eq. (1)), and $\theta_{min}$ is the minimum threshold of the number of participants in the evaluation.

| Algorithm | Send (bits) | Receive (bits) |
|-----------|-------------|----------------|
| Authority | 0           | $O(mn\kappa)$ |
| Special User 1 | $O(kn_1\kappa)$ | $O(kn_1\kappa)$ |
| KeyGen when $k$ users collide Encode | $O(mn\kappa)$ | $O(mn\kappa)$ |
| Adding $n'$ users | $O(n'\kappa)$ | $O(n'\kappa)$ |
| Other participants including Special User 2 | $O(kn_2\kappa)$ | $O(kn_2\kappa)$ |
| KeyGen when $k$ users collide Encode | $O(mn\kappa)$ | $O(mn\kappa)$ |
| Adding $n'$ users | $O(n'\kappa)$ | $O(n'\kappa)$ |

User 1 with special role has to receive all other users' communication strings to generate the homomorphic ciphertexts, and his communication overhead is $\theta_{min}$ times greater. Besides, we only need two communication rounds for each polynomial evaluation, which is a constant irrelevant to the number of participants. Specifically, everyone except user 1 sends out his data in the first round, and user 1 broadcasts the homomorphic ciphertext in the second round. Finally, when $\kappa = 512$, the actual size of an encoded value $C(x_{ik})$ is 256B, and a homomorphic ciphertext $E(C(x_{2k}))$ is 512B.
6.2 Practical Performance: Computation

To evaluate the computation overhead, we implemented our scheme in Ubuntu 12.04 using the GMP library (gmplib.org) based on C in a computer with Intel i3-2365M CPU @ 2.80 GHz x2, Memory 3GB. To exclude the communication overhead from the measurement, we generated all the communication strings as files in one computer and conducted all the computation locally (I/O overhead excluded in the measurement). Security parameter $\kappa$ is set as 512 bits to achieve strong prime numbers (the moduli $N, \tilde{N}$ are around 1024 bits). Paillier’s cryptosystem is employed as the homomorphic encryption in Encode, which is also implemented in C.

Microbenchmark

We first perform a series of microbenchmarks to measure the basic cost units of our scheme as in Table 2. Overhead of the Encode and Decode algorithms depend on the polynomial to be evaluated, so we randomly chose a polynomial with single product term having 10,000 participants to measure the exact run time of each algorithm (Table 2).

| Algorithm       | Min   | Max   | Mean   | Std. Dev. |
|-----------------|-------|-------|--------|-----------|
| Authority       | 0.25ms| 0.31ms| 0.28ms | 0.01ms    |
| Aggregate       | 9.7ms | 10.1ms| 9.846ms| 0.057ms   |
| Encode          | 0.115ms| 0.145ms| 0.129ms| 0.0186ms  |

User 1’s and 2’s overhead is more expensive than other users because they conduct operations in both $\mathbb{Z}_N$ (1024 bits) and $\mathbb{Z}_{N^2}$ (2048 bits) while other users’ operations are closed only in $\mathbb{Z}_N$. Besides, the aggregator’s overhead is cheaper than user 1’s or 2’s because he only performs multiplications which are much cheaper than exponentiations.

As discussed in Section 7.4, User 1’s overhead is affected by the number of actual participants in each product. Also, it is clear that the overall overhead is related to the number of products in the polynomial. To evaluate the scalability w.r.t. those parameters, we execute the Encode algorithm and Aggregate algorithm for randomly chosen polynomials with different parameters (Figure 1).

In Figure 1(a), x-axis is the minimum threshold, y-axis is the number of product terms in a polynomial, and z-axis is the run time. In Figure 1(b) and 1(c), x-axis is the number of product terms in a polynomial, and y-axis is the run time. Clearly, the run time of Encode grows bilinearly w.r.t. the parameters for the special-role user 1, therefore, when either parameter is large, one should employ a cloud server to act as the special user 1 since an ordinary user may not be able to handle such huge computation. The run time of Encode grows linearly w.r.t. the number of product terms for other users (Figure 1(b)), and the same applies for Aggregate for the aggregator. Note that the performance is more sensitive to the number of products because it is equal to the number of exponentiations in $\mathbb{Z}_{N^2}$, which is much more expensive than the multiplications.

Data Analysis via PDA

In this section, we obtain various datasets available online (UCI machine learning datasets [4], SNAP Amazon Movie Review [49], SNAP Amazon Product Review [49]), and conduct various analysis on them as described in Section 4.5 to show the generality of our framework as well as its performance. Note that the run times shown below are the overall run time when all the computations are conducted in a laptop computer, and they are not the run times per user. The actual run time for non-special user is almost negligible because the special users’ overhead contribute to the majority of the run time as analyzed above.

Statistics Calculation: As aforementioned (Section 4.5), various statistics analysis are represented as polynomials, which can be directly evaluated by our scheme. To compare the performance with peer works, we implemented three schemes Shi [54], Joye [40], and Jung [41] in the same environment, whose purpose is to calculate a product or a sum on a private dataset. Due to this limitation, these schemes can be used to calculate simple statistic values only (e.g., mean, standard deviation), and we compared the overall run time of the mean calculation on random values as most of the statistics analysis supported by [40], [41], [54] are based on mean calculation. The entire calculation is conducted on a laptop computer to measure the pure computation overhead only, and the security parameter of each scheme is chosen such that all schemes have the similar security level ($\approx$80-bit security level).

Fig. 2. Comparisons of mean calculation

As shown in Figure 2, the overhead of our scheme is comparable to the one of Joye [40], and is twice of the ones of Shi [54] and Jung [41]. The main factor of this gap is the bit-length of the moduli. When bit-security levels are comparable to each other, the bit-lengths of the largest moduli in Shi [54] and Jung [41] are as twice large as the ones in Joye [40]. Although our performance is not the best among these four schemes, other schemes can only let an aggregator obliviously evaluate the product or the sum over a fixed dataset. Considering our scheme’s generality which enables an aggregator obliviously evaluate general multivariate polynomials over arbitrary subset of datasets (with the support of new user arrival and old user removal), we claim that this performance gap is acceptable.

Regression Analysis: We trained a linear regression model over the datasets in a privacy-preserving manner using our scheme (Section 4.5), and measured the communication overhead for each ordinary user as well as the overall run
time of the entire training in a local computer (Table 3). Specifically, in SNAP datasets, we counted the number of ‘good’ strings (e.g., ‘I love’, ‘I like’, ‘best’) and ‘bad’ strings (e.g., ‘I hate’, ‘terrible’) in the review, and treated those numbers as well as the price as the features. We further treated the score rating as the dependent variable.

Moreover, we compare our performance with a similar work [51] which designed a privacy-preserving ridge regression over millions of private data records (Table 4). As we failed to obtain the source codes of their scheme, we perform the comparison based on the statistics provided in their paper. This is not a completely fair comparison of the different computing environment. However, PDA supports any polynomial-based data analysis while [51] only supports the ridge regression, therefore we claim that our solution is more preferable if the performance of our general solution is comparable to the ad hoc solution.

SVM Analysis: We implemented the SMO algorithm [53] in a privacy preserving manner (Section 4.5) to train a linear support vector machine over the same datasets (part of UCI datasets) as those in [53]. Please refer to the paper for the detailed description of the dataset.

Note that the run time to train a SVM is linear to the number of iterations until convergence in the SMO algorithm, which heavily depends on the characteristics of the dataset. Our experiment shows that the SVM training via PDA has a non-negligible overhead (Table 5), but this is inevitable because 1) the run time of plain SMO algorithm on the same dataset is also of several seconds, and 2) the inherent complexity of the multiplications and exponentiations over k-bit integers are $O(k^2)$ and $O(k^3)$ respectively, and the plain SVM without privacy consideration performs arithmetic operations in double type numbers (8B) while our framework (with 80-bit security level) performs arithmetic operations as well as exponentiations on 256B numbers. To the best of our knowledge, [44] is the first and only work who presents a privacy-preserving method to train a SVM over a large number of individual records (most of similar works study how to train SVM on a small number of vertically or horizontally partitioned database), but their work does not evaluate the performance in an real implementation. Although we were not able to acquire the implementation, we could indirectly compare our performance with this only similar work. Their theoretic performance complexities (computation & communication) in theory are exponentially greater than ours. Such a gap exists because they relied on the secure multi-party computation. The performance gap between the solutions based on secure multi-party computation and our work discussed in Section 8.1 also tells the performance gap in real implementations is huge. Therefore, ours can be considered as a development to their solution which improved the performance aspect by getting rid of the secure multi-party computation.

7 DISCUSSIONS & ANALYSIS

7.1 Disjoint Time Domains

To be indistinguishable against CPA, our Encode must be randomized. However, it is not easy to let users achieve independent but coordinated random masks such that their aggregation yields a constant if we do not rely on a trusted center. We use the time slot as the ‘seed’ of our hash function $H(\cdot)$ to address this challenge.

In the Encode algorithm, $H(t_i)^{q(\prod_{j=1}^{p-1}i)} \in \mathbb{Z}_N^*$ is used to mask the private value $x_{ik}$. As analyzed in the appendix, such random masks are seemingly random under the random oracle model. Since the random mask is based on the hash function $H(\cdot)$ which is deterministic, we require
a disjoint time domain and forbid the re-use of any time slot \( t_k \). Same time slots yield same randomizers, and this will allow some polynomial adversaries to have non-negligible advantages in our game.

7.2 Minimum Threshold for \(|P|\)
Since the security of PDA relies on the random masks whose product or sum is equal to an identity element, we require the minimum number of users in any polynomial evaluation should be \( \theta_{min} = 3 \). Otherwise, a user can trivially deduce another user’s secret random mask \( r \) by inverting his own random mask \( r^{-1} \) (in multiplicative group). That is, besides the special user 1&2, we also need one ordinary user participating in the evaluation. This is why we calculate the secret key \( q^{(i)}(i) \) in KeyGen algorithm only for \( d = 2, 3, \cdots, n - 1 \) (recall that \( q^{(k)}(i) \) is used when \( k + 1 \) users participate in the evaluation).

7.3 Passive Rushing & Collusion Attacks
In previous sections, we intentionally simplified the adversarial model as well as the construction for the readability. Consequently, the basic construction allows certain attacks. In this section, we present a countermeasure to achieve a more complete construction.

In a passive rushing attack, attackers access the messages sent from others before they choose their random numbers \([30]\). Specifically, during the exchanges of \( Y_j \) in the KeyGen algorithm, an adversarial user \( i \) can send out \( y_{i-1} = y_{i+1}^{-1} \) to user \( i \) \((a \in \mathbb{Z}_q^*)\) after receiving \( y_{i+1} \). Then, \( P_i \)'s random mask is equal to \( y_{i-1} y_{i-1}^{-1} = \tilde{g}^{r_{i-1} (r_{i-1} - a) r_{i-1}} = \tilde{g}^{r_{i-1}} \), which can be efficiently calculated with \( i \)'s public parameter \( y_i = \tilde{g}^r \). This attack does not tamper the final computation and cannot be detected. With exactly same theory, two colluding users \( i + 1 \) and \( i - 1 \) can also easily calculate \( i \)'s secret parameter \( Y_i = \tilde{g}^{r_{i+1} - r_{i-1}} r_{i-1} \).

However, we can adopt the ideas from \([41]\). With one extra round of exchange, we can let each user have \( Y_i = (y_{i+2} y_{i+1}^{-1})^{r_{i+1} r_{i+1}} \), and a passive rushing attacker or two colluding users will fail to infer a user’s \( Y_i \). In general, to defend against \( k \) colluding passive rushing attackers or \( k + 1 \) colluding users, we can have \( k \) more extra rounds of exchanges to let each user \( i \) have \( Y_i = (y_{r_{i+2} r_{i+1}^{-1} r_{r_{i+1}}})^{r_{i+2} r_{i+1}^{-1} r_{r_{i+1}}} \).

Recall that we initially assumed non-cooperative users in this paper. But with the above prevention, we can relax the assumption and make our PDA resilient to a collusion attack of up to \( k - 1 \) passive rushing attackers or \( k + 1 \) normal attackers. As a result, all encoding keys \( q^{(2)}(i), \cdots, q^{(k)}(i) \) become unusable. This is because any user’s encoding key \( EK \) is composed of several data points in hidden polynomials, and \( k \) colluding users can deduce the \( k \)-degree hidden polynomial \( q^{(k)}(x) \) of 0 constant term. Then, these colluding users can compute any user \( i \)'s encoding key \( q^{(k)}(i) \). Thus, the lowest degree of the usable polynomial is \( k + 1 \), and the minimum threshold \( \theta_{min} \) for \(|P|\) in the PDA becomes \( k + 2 \) (each user uses \( q^{(k+1)}(i) \) as the encoding key) to guarantee the semantic security of PDA. Employment of this countermeasure translates to multiple extra communication rounds in KeyGen, but this is a one-time operation at the beginning of the system initialization.

7.4 Performance Optimization
In the algorithm Encode, we require that everyone should submit his encoded value \( C(x_{ik}) \) even though his \( x_{ik} \) does not appear in the product term \( \prod x_{ik} \) (in which case \( x_{ik} \) is set as 1). This is because the number of actual participating users should be at least a certain threshold \( \theta_{min} \) to guarantee the semantic security (reviewed in details in Section 7.5). However, in some polynomials, single product terms involve very few users regardless of the total number of participants \(|P|\) of the polynomial, such as \( f(x_P) = \sum_{x \in P} x_1 x_j \). If strictly following Encode, one needs to have \(|P| - 1 \) (encoded) 1’s multiplied for each product term \( x_1 x_j \). However, to preserve the semantic security of the scheme, we only need \( \theta_{min} \) users to participate in each single product term, and this is an unnecessary overhead. This performance degradation becomes huge when \( \theta_{min} \ll |P| \), but we can simply get rid of it as follows. Whenever the number of participants involved in a product term is less than \( \theta_{min} \), randomly find dummy users in \( P \) only until we have \( \theta_{min} \) users instead of having everyone in \( P \) participating in the analysis. Then, dummy users participate in the evaluation by providing 1 as their private values, and by doing so, we only have minimum number of participants for every product term, which greatly cut off the unnecessary overhead.

7.5 Limitations & Future Works
Performance Limit: It is shown in Section 6 that user 1’s communication and computation overhead grows linearly w.r.t. the actual participants in each product term, which may become huge in some big data analysis. In such case, we may add a cloud server to take this role, and let him provide 1 during the evaluation. Besides, each person needs to send/receive \( O(n^2 \kappa) \) bits (Section 6,1) with small constant during the one-time operation KeyGen, which may be large when \( n \) is large. This can be reduced with several naive ideas, for examples, dividing \( n \) users into several subgroups and forcing the analysis within the subgroups only. But such ideas disable some types of analysis, and handling this limit is one of our future works.

Polynomial Parameters: This paper focuses more on user’s data privacy, and pays less attention to the aggregator’s privacy. This is why we require that the polynomials should be public, otherwise aggregators may adaptively design the polynomial \( (e.g., f(x_P) = x_1) \) to directly infer a user’s data. However, in some cases, the polynomial parameters may be proprietary property which also need to be protected \( (e.g., \text{data mining techniques}) \), but then corresponding verification mechanisms should be introduced as well to enable users to verify whether the polynomial is maliciously designed, which is one of our future works.

Finite Real Numbers: Although our design supports polynomials over real numbers, the real numbers are represented via fixed-point representation, which constructs one-to-one mapping between the real numbers and the integers in the cyclic group we use as a part of crypto protocols. Therefore, the real numbers that can be expressed in our system is a finite set of real numbers whose size is equal to the order of the group in our algorithms \( (i.e., N) \).

Data Publishing: Our PDA only supports pull-based data
analysis in which the aggregator requests the data analysis with the declaration of the polynomial, and then users publish their data according to it. One thought is to achieve a push-based analysis, where users publish their data whenever it is generated, and the aggregator chooses desired subset of data and periods to perform the data analysis. However, once the data is published, it will not be under the user’s control, and it becomes much more challenging to protect the privacy. Designing a push-based data analysis which can prevent malicious analysis on the already published data is a very challenging task, and we leave this as our future work too.

8 RELATED WORK

The purpose of our scheme is to protect the data privacy rather than the identity privacy. Therefore, we focus more on reviewing peer works studying the privacy-preserving data analysis whose purpose is to protect the data confidentiality. In general, related approaches can be categorized into the following families: approaches based on secure multi-party computation, perturbation, segmentation, functional encryption and secret-shared keys.

8.1 Secure Multi-Party Computation

| TABLE 7 | Comparison of the fastest SMC implementation and ours |
|---------|-----------------------------------------------|
| 5,550 AND via GMW | 134 additions via ours |
| 15-20 seconds | 0.06-0.08 seconds |

Secure Multi-party Computation (SMC) [57] enables $n$ parties to jointly and privately compute a function $f_i(x)$ where the result $f_i$ is returned to the participant $i$ only. One can trivially use SMC to enable data analysis on published sanitized data, but SMC is subject to extraordinarily high delay in real life implementations. Among various real implementations on multi-party computation schemes (FairplayMP [6], VIFF [23], SEPIA [13], and GMW [17]), even the fastest GMW [17] does not scale well, whose communication rounds is linear to the number of users with huge constant factors, and it takes 15-20 seconds to conduct 5,500 AND gates evaluation. Similar-size computation in our scheme only incurs subsecond-level computation time and single-round communication of several milliseconds (Table 7). Such huge performance gap is due to the different purposes. SMC targets at implementing arbitrary computation while ours target at polynomial-based data analysis only. Considering the number of users and the volume of data, it is more practical to rely on schemes specially designed for general analysis to enable lightweight protocols.

8.2 Perturbation

The perturbation is often leveraged to achieve the differential or membership privacy (22, 37). In such approaches, a random noise drawn from a known distribution is added to mask the original data, and some statistics about the aggregate result can be deduced based on the distribution of the random noises. These approaches enable certain analysis on published (sanitized) data (normally without requiring a trusted key dealer), but the accuracy is dependent on the guaranteed security (i.e., accuracy increases when security is sacrificed), and the temporal correlation between the data leads to a limited utility when the data is time-series. Ács et al proposed a privacy-preserving smart metering system which support various aggregated statistics analysis without leaking household activities. [1] presented how to perform Fan et al. [28] presented an adaptive approach to time-series data releasing with differential privacy, but their relative error is greater than $10^{-1}$ with certain privacy sacrifice. Recently, Chen et al. [16] presented a scheme based on perturbation to mine frequent patterns in the form of substring, but the utility for prefixes is poor due to the noises making the scheme differentially private. Supported by the recent demand in the accuracy [38], we claim that, for some applications in the future, it is not desirable to rely on the approaches whose accuracy depends on the privacy requirements of the applications.

8.3 Functional Encryption

Functional encryption [11] is a type of recently proposed public key encryption in which a certain function of the plaintext can be derived from the ciphertext if one possesses a valid private key corresponding to the function. It is possible to leverage FE to design a solution to our problem, but the one key is associated with a pre-defined function in FE, and this key can be used to derive only one function of the plaintext. In many data mining algorithms, the actual target analysis functions (classifiers or predictive models) are trained iteratively, in which each training at each iteration is another type of function to be evaluated over the private dataset. Since all such analysis are data-dependent, issued keys can hardly be re-used, and keys should be issued to all participants in advance of every data analysis, which is a communicationally expensive interaction. Furthermore, the key distribution must be performed via secured channels.

In contrast, our keys can be re-used arbitrarily many times as long as the time windows for every polynomial are disjoint with each other.

8.4 Secret-shared Keys

Several schemes distribute certain keys to the data users such that the sum or product of the keys are constants. Then, these keys are used to mask the original data. These schemes are very close to ours.

Shi [34] and Joye [40] let the aggregator compute the sum of $n$ participants’ data $x_i$’s without learning anything else. They both let a trusted third party generate random keys $s_i$ for the $i$-th participant such that $\sum s_i = 0$. Then, it follows that $\prod H(t)^{s_i} = 1$, where $H(\cdot)$ is a hash function employed as a random oracle and $t$ is the time slot when the calculation is carried out. Then, this property is used by both schemes to compute the sum $\sum x_i$, and both of them are proved to be semantically secure against chosen-plaintext attack. However, both of the schemes rely on a trusted key issuer, and the data is not confidential against him. Furthermore, only the product or sum calculation is enabled, which makes it less suitable for a general data analysis. Jung [41], [42] removed the necessity of a trusted key dealer in their settings, and their scheme enables both product and sum
calculations. They also mention the possibility of evaluating some multivariate polynomials, but all the product terms in the polynomials are revealed to the aggregator. More importantly, the hardness of breaking their scheme reduces to computational diffie-hellman (CDH) problem only, which makes their scheme one-way but not computationally secure.

These four schemes based on secret-shared keys are close to our solution, but all the schemes need a fixed group and assign a set of secret keys, which means for a set of $n$ users, one needs to let all users hold $O(2^n)$ keys to conduct data analysis among any subgroup of them, which violates the flexibility requirement.

As a summary, the comparisons of all aforementioned approaches and ours are shown in the Table 6.

### 8.5 Secure Multivariate Polynomial Evaluation

To the best of our knowledge, [9], [22], [31], [42] are the only four works who study the secure multi-party computation on multivariate polynomial evaluation. First of all, the solution in [42] reveals all single product terms in the polynomial, and it leaks much information about the private values. [31] focused on a polynomial of degree 3, and points out that this can be generalized to a higher degree. However, in higher degrees, the communication overhead is not optimal. Subsequently, [22] presented a more efficient scheme with the unicast complexity $O(km^2n \log d)$ and broadcast complexity $O(km^3(\log d)^2)$ (untight bound for simplicity), where $k$ is the security parameter, $m$ is the number of product terms in the polynomial, $n$ is the number of participants, and $d$ is the highest degree of the polynomial. Also, their communication rounds is a constant in terms of the ‘round table’ rounds, which is linear to the number of users. In our framework, the unicast complexity for each evaluation is only $O(\theta_{\text{min,mc}})$ for the special user ($\theta_{\text{min}}$ being the minimum threshold for the number of participants) and $O(mk)$ for other participants, and there is no broadcast overhead. Also, the communication round is a constant irrelevant to the number of participants in the evaluation. Besides, [9] also discusses how to compute limited families of polynomials, but the rounds complexity is $O(1\log l)$ where $l$ is the bit-length of the inputs.

### 9 Conclusion

In this paper, we successfully designed PDA, a general framework which enables a third-party aggregator to perform many types of polynomial-based data analysis on private data records, where the data records are possessed by millions of users. Namely, our framework enables the aggregator to evaluate various multivariate polynomials over any subgroup of users’ private data without learning individuals’ input values, and it also supports new users to join the system and old users to leave the system. Our formal proof shows the semantic security of PDA against chosen-plaintext attacks, and the extensive evaluations show that PDA incurs constant overhead regardless of the total number of users or the complexity of the polynomial. Besides, PDA also exhibits various nice properties (design goals in Section 1) which make it suitable for real life applications.

### REFERENCES

[1] G. Ács and C. Castelluccia. I have a dream! (differentially private smart metering). In Information Hiding, pages 118–132. Springer, 2011.
[2] C. G. Akcora, B. Carminati, and E. Ferrari. Privacy in social networks: How risky is your social graph? In ICDE, pages 9–19. IEEE, 2012.
[3] M. Aliasgari, M. Blanton, Y. Zhang, and A. Steele. Secure computation on floating point numbers. In NDSS, 2013.
[4] K. Bache and M. Lichman. UCI machine learning repository, 2013.
[5] M. Barbaro, T. Zeller, and S. Hansell. A face is exposed for aol searcher no. 4417749. New York Times, 9(2008):8For, 2006.
[6] A. Ben-David, N. Nisan, and B. Pinkas. Fairplaymp: a system for secure multi-party computation. In CCS, pages 257–266. ACM, 2008.
[7] J. Bennett and S. Lanning. The netflix prize. In Proceedings of KDD cup and workshop, volume 2007, page 35, 2007.
[8] M. J. Berry and G. Linoff. Data mining techniques: for marketing, sales, and customer support. John Wiley & Sons, Inc., 1997.
[9] M. Blanton. Achieving full security in privacy-preserving data mining. In PASSAT and SocialCom, pages 925–934. IEEE, 2011.
[10] D. Boneh. The decision diffie-hellman problem. In Algorithmic number theory, pages 48–63. Springer, 1998.
[11] D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In TCC, pages 253–273. Springer, 2011.
[12] J. S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In UAI, pages 43–52. Morgan Kaufmann Publishers, 1998.
[13] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos. Sepia: Privacy-preserving aggregation of multi-domain network events and statistics. In Security. USENIX, 2010.
[14] J. Camenisch and M. Michels. Proving in zero-knowledge that a number is the product of two safe primes. In EUROCRYPT, pages 107–122. Springer, 1999.
[15] B. Carminati, E. Ferrari, and M. Guglielmi. Share: Secure information sharing framework for emergency management. In ICDE, pages 1336–1339. IEEE, 2013.
[16] R. Chen, G. Acs, and C. Castelluccia. Differentially private sequential data publication via variable-length n-grams. In CCS, pages 638–649. ACM, 2012.
[17] S. G. Choi, K.-W. Hwang, J. Katz, T. Malkin, and D. Rubenstein. Secure multi-party computation of boolean circuits with applications to privacy in on-line marketplaces. In CT-RSA, pages 416–432. Springer, 2012.
[18] S. S. Chow, J.-H. Lee, and L. Subramanian. Two-party computation model for privacy-preserving queries over distributed databases. In NDSS, 2009.
[19] E. K. Clemons, S. Barnett, and A. Appadurai. The future of advertising and the value of social network websites: some preliminary examinations. In ICEC, pages 267–276. ACM, 2007.

### TABLE 6

| Approach based on          | Storage Complexity | Communication Rounds | Accurate result? | Rely on secure channel? |
|----------------------------|--------------------|----------------------|------------------|------------------------|
| Multi-party Computation    | High               | $O(n)^*$             | Yes              | No                     |
| Perturbation               | $O(1)$             | $O(1)$               | No               | No                     |
| Functional Encryption      | High               | $O(1)$               | Yes              | Yes                    |
| Secret-shared Keys         | $O(2^n)$           | $O(1)$               | Yes, except [11], [42] |
| Ours                       | $O(n)$             | $O(1)$               | Yes              | No                     |

*This is the best known complexity, and every gate in the circuit incurs one round of oblivious transfer.
