Privacy-Preserving Distributed Set Operations

Qingsong Ye and Huaxiong Wang

Department of Computing, Macquarie University, Australia
{qingsong,hwang}@ics.mq.edu.au

Abstract. Motivated by the demand of databases outsourcing and its security concerns, we investigate privacy-preserving set operations in a distributed scenario. Combining Shamir secret sharing scheme and homomorphic encryption, we propose a one-round protocol for privacy-preserving set intersection. We then show that, with an additional round of interaction, the cardinality of set intersections can be computed efficiently. Moreover, using oblivious polynomial evaluation techniques, we provide a solution for the subset relation problem. All protocols constructed in this paper are provably secure against a semi-honest adversary under the Decisional Diffie-Hellman assumption.

Keywords: privacy-preserving set-operation, homomorphic encryption

1 Introduction

Privacy-Preserving Set Operations (PPSO) [9] are cryptographic techniques allowing two or more parties, each holding a set of inputs, to jointly calculate set operations of their inputs without leaking any information. Imagine that two companies want to discover the consumption pattern of their shared customers. That is, they want to determine the likelihood that a customer buying the product P_1 from the company C_1 is also buying the product P_2 from the company C_2. To obtain this information, they would like to perform a set intersection operation on their private datasets. In order to preserve confidentiality of the companies business and to protect the customers privacy, the customers details must not be revealed. Another example is when a research institute and a group of hospitals are conducting a study in which they analyze patients records anonymously.

Motivated by the demand of databases outsourcing and security requirements imposed on its applications, we investigate PPSO in a distributed environment. We call this privacy-preserving distributed set operations. To illustrate the security problem we consider the following scenario. Assume a provider owning a dataset wishes to make his dataset available to clients. He distributes the dataset to w servers using a threshold secret sharing (with a threshold t). Now a client holding her private dataset, wishes to compute a specific set operation for the two sets held by the client and the provider. In order to do this successfully, the client interacts with t or more servers. In addition we assume that the interaction between the client and servers is done with minimum possible disclosure of information. In other words, the client learns nothing except the final result of the set operation.

Our Contribution. In this paper, we propose an efficient technique for PPSO in the distributed setting. Our approach is based on homomorphic encryption and oblivious polynomial evaluation. If these underlying techniques are semantically secure, then so are the protocols we construct. We design an efficient method to enable privacy-preserving computation of *distributed set intersection* based on Lagrange's interpolation. We then apply the oblivious polynomial evaluation technique to construct a protocol for testing whether a dataset held by the client is a subset of a dataset held jointly by the servers. The protocol is efficient and only requires another round of communication between the client and a single one of the w servers. To reduce the computational cost of oblivious polynomial evaluation, we employ random shuffling and provide efficient solutions for the *cardinality of set-intersection* problem.

Our protocols are secure against a honest-but-curious (semi-honest) adversary. Such an adversary follows the steps of the execution of the protocol but tries to learn extra information from the messages receives during its execution. Our homomorphic encryption is based on the ElGamal cryptosystem [5], which is semantically secure provided the Decisional Diffie-Hellman (DDH) assumption holds [17]. Note that, as in [4, 8], our protocols reveal the size of the datasets of both the client and the provider. As suggested in [8], "dummy" elements can be used for dataset padding in order to hide the size of the original dataset. But in this case the protocol reveals the upper bound on the number of elements in the sets.

Because of space limitation all the proofs in this paper are only sketched.

Related Work. In general, PPSO can be implemented using secure multiparty computation (MPC) protocols [2, 18]. However, such solutions generally are inefficient. Kessner and Song [9] proposed a solution to various privacy-preserving set operations in the context of private computation on multisets for multiple players. Based on a threshold homomorphic cryptosystem, Sang et al. in [15] gave protocols for the set intersection and set matching problems with an improved computational and communication complexity.

Protocols for finding the intersection of two private datasets were studied in [12]. Recently, Freedman et. al. [4] proposed efficient constructions for the private set intersection using the representation of datasets as roots of a polynomial. Protocols for testing the set disjointness are discussed in [8, 7]. Protocols for equality tests are a special case of the private disjointness problem, where each party has a single element in the database. These were considered in [3, 12, 10].

Our paper is organized as follows. In Sect. 2, we introduce the cryptographic primitives used in our protocols, describe the distributed environment in which our protocols are run, and give the adversary model. In Sect. 3, we present a protocol and its security analysis for the set intersection. Sect. 4 gives a protocol and its security analysis for the subset relation. In Sect. 5, we use the constructions for set intersection and subset relation to construct a new algorithm finding the cardinality of set intersection. Finally, in Sect. 6, we give concluding remarks and discuss possible future work.

2 Preliminaries

2.1 Additively Homomorphic Encryption

We will utilize an additively homomorphic public-key cryptosystem. Let $E_{\text{pk}}(\cdot)$ denote the homomorphic encryption function with a public key pk. It supports the following operations, which can be performed without knowing the private key.

- Given $E_{\text{pk}}(a)$ and $E_{\text{pk}}(b)$, we can efficiently compute $E_{\text{pk}}(a+b)$.
- Given a constant c and $E_{\text{pk}}(a)$, we can efficiently compute $E_{\text{pk}}(ca)$.

In our protocols, the computations are carried out over $\mathbb{Z}_{\text{p}}$ where p is prime. We note that all of our protocols can be based on the standard variant of the ElGamal encryption scheme, which recently was used for constructing protocols for PPSO (see, for example [8, 1, 10]). Let the triple (K, E, D) be the variant of ElGamal where

- K is the key-generation algorithm defined as follows. Given a security parameter $l = \lceil \log_2 p \rceil$, $(pk, sk) \leftarrow K(1^l)$ where the public-key is $pk := \langle p, g, h, f \rangle$ and the corresponding secret-key is $x := \log_{\text{g}} h$. More precisely, we assume that $q = (p-1)/2$ is also a prime number, g is an element of order q in $\mathbb{Z}^*_{\text{p}}$ and $h, f \in \langle g \rangle$;
- E is the encryption algorithm defined as follows. Given the public-key pk and a plaintext $m \in \mathbb{Z}_{\text{q}}$, one encrypts m as $E_{\text{pk}}(r, m) = (g^r, h^r f^m)$ where $r \overset{\text{R}}{\leftarrow} \mathbb{Z}_{\text{q}}$;
- D is the decryption algorithm defined as follows. Given the secret-key x and a ciphertext (a, b) the decryption algorithm returns $a^{-x}b \mod p$. Notice that this will only return f^m rather than m, however this suffices for our purposes. Indeed, in our protocols, we are only interested in testing whether $m = 0$ or not, which is equivalent to testing if $a^{-x}b \equiv 1 \mod p$.

To demonstrate that the above encryption scheme is indeed homomorphic, we define a operation $\odot$ as multiplication over $\mathbb{Z}_{\text{p}} \times \mathbb{Z}_{\text{p}}$. Let $m_1, m_2, m \in \mathbb{Z}_{\text{q}}$ and corresponding $r_1, r_2, r \overset{\text{R}}{\leftarrow} \mathbb{Z}_{\text{q}}$, then

$$E_{\text{pk}}(r_1, m_1) \odot E_{\text{pk}}(r_2, m_2) := \left(g^{r_1+r_2}, h^{r_1+r_2} f^{m_1+m_2}\right) = E_{\text{pk}}(r_1 + r_2, m_1 + m_2).$$

If we repeat this operation c times for a single encryption, then we have

$$E_{\text{pk}}(r, m)^c = \underbrace{E_{\text{pk}}(r, m) \odot E_{\text{pk}}(r, m) \odot \ldots \odot E_{\text{pk}}(r, m)}_{\text{c times}} := E_{\text{pk}}(cr, cm).$$

For simplicity, we use $E_{\text{pk}}(m)$ to represent $E_{\text{pk}}(r, m)$ in the rest of the presentation as we assume that there is always a corresponding $r \overset{\text{R}}{\leftarrow} \mathbb{Z}_{\text{q}}$.

2.2 Oblivious Polynomial Evaluation

Given two parties, $\mathcal{A}$ and $\mathcal{B}$. $\mathcal{A}$ owns a function $\mathcal{F}$ and would like to let $\mathcal{B}$ compute the value $\mathcal{F}(b)$ for an input b owned by $\mathcal{B}$. The computation should be done in such a way that

– $\mathcal{B}$ does not learn any information about the function $\mathcal{F}$.
– $\mathcal{A}$ learns the result $\mathcal{F}(b)$ but does not learn anything about the private input b of $\mathcal{B}$.

We use oblivious polynomial evaluation given in [12, 4] to perform the set operations. $\mathcal{A}$ defines a polynomial $\mathcal{W}$ whose roots are her inputs $\mathcal{D}_{\mathrm{A}} = \{x_1, \ldots, x_n\}$:

$$\mathcal{W}(y) := (y - x_1)(y - x_2)\ldots(y - x_n) = \sum_{u=0}^{n} \alpha_u y^u.$$

$\mathcal{A}$ encrypts the coefficients α_u of this polynomial with the homomorphic public key cryptosystem and sends all the encrypted coefficients $\{E_{\mathrm{pk}}(\alpha_0), \ldots, E_{\mathrm{pk}}(\alpha_n)\}$ to $\mathcal{B}$, where $E_{\mathrm{pk}}(\alpha_i) = (g^{r_i}, h^{r_i} f^{\alpha_i})$. Because of the homomorphic property, $\mathcal{B}$ then evaluates the polynomial for the input β as follows:

$$E_{\mathrm{pk}}(\mathcal{W}(\beta)) = \sum_{i=0}^{n} E_{\mathrm{pk}}(\mathcal{W}(\alpha_i))^{\beta^i} = (g^{\sum_{i=0}^{n} r_i\beta^i}, h^{\sum_{i=0}^{n} r_i\beta^i} f^{\sum_{i=0}^{n} \alpha_i\beta^i}),$$

and sends it to $\mathcal{A}$. When $\mathcal{A}$ receives $E_{\mathrm{pk}}(\mathcal{W}(\beta))$, she decrypts it and obtains $f^{\mathcal{W}(\beta)}$. She then concludes if $\mathcal{W}(\beta) = 0$ if and only if $f^{\mathcal{W}(\beta)} = 1$.

2.3 Distributed Environment

The main players are a client $\mathcal{C}$, a provider $\mathcal{P}$, and w servers $S_1, S_2, \ldots, S_w$. We assume that the provider holds a set of secrets $\mathcal{D}_{\mathrm{P}} = \{\mu_0, \mu_1, \ldots, \mu_{n-1}\}$, which is distributed to w servers using (t, w)-Shamir's secret sharing scheme. In addition, $\mathcal{P}$ sends $\lambda \xleftarrow{\mathrm{R}} \mathbb{Z}_{\mathrm{q}} - \{0\}$ to all w servers. Suppose that there exits a public pseudo-random number generator G, whose outputs $G(\lambda)$ is uniformly distributed in $\mathbb{Z}_{\mathrm{q}} - \{0\}$. The notation $G(\lambda)$ means that the pseudo-random number generator G takes the random value λ as its seed.

The provider $\mathcal{P}$ does not directly interact with $\mathcal{C}$ for the set operations, instead $\mathcal{C}$ contacts at least t servers to perform a requested set operation. Note, this distributed setting was first proposed by Naor and Pinkas [13].

Our homomorphic encryption system is based on a variant of ElGamal cryptosystem in which the message space is over $\mathbb{Z}_{\mathrm{q}}$ where $q \geq n$. For simplicity, we omit modulus q within the computation of shares construction in this section.

Initialization and Share Distribution Phase. First, $\mathcal{P}$ constructs a polynomial $F(y)$ whose coefficients are his inputs (n secrets) as

$$F(y) = \sum_{i=0}^{n-1} \mu_i y^i.$$

Then $\mathcal{P}$ generates a random masking bivariate polynomial $H(x, y)$ of the following form:

$$H(x, y) = \sum_{j=1}^{t-1} \sum_{i=0}^{n-1} a_{j,i} x^j y^i \quad \text{where } a_{j,i} \in \mathbb{Z}_{\mathrm{q}}.$$

Note that we have $H(0, y) = 0$ for any y. Using the polynomial H, $\mathcal{P}$ defines another bivariate polynomial $Q(x, y) = F(y) + H(x, y)$. Note that we get $\forall y\ Q(0, y) = F(y)$. For $1 \leq \ell \leq w$, $\mathcal{P}$ computes and sends $Q(\ell, y)$ to server S_ℓ where

$$Q(\ell, y) = F(y) + H(\ell, y) = \sum_{i=0}^{n-1} \mu_i y^i + \sum_{j=1}^{t-1} \sum_{i=0}^{n-1} a_{j,i} \ell^j y^i = \sum_{i=0}^{n-1} (\mu_i + b_{i,\ell}) y^i.$$

Denote $\mu_{i,\ell} = \mu_i + b_{i,\ell}$. The server S_ℓ receives a set of shared coefficients $\{\mu_{0,\ell}, \ldots, \mu_{n-1,\ell}\}$ of the polynomial F (see [11]).

Secret Reconstruction Phase. By the Lagrange interpolation formula, we know that the coalition of t or more servers can reconstruct the original polynomial F. The t polynomials $Q(\ell_m, y)$ for $m \in [1, \ldots, t]$ are:

$$\begin{aligned} Q(\ell_1, y) &= F(y) + \sum_{j=1}^{t-1} \sum_{i=0}^{n-1} a_{j,i} {\ell_1}^j y^i \\ &\vdots \qquad\qquad \vdots \\ Q(\ell_t, y) &= F(y) + \sum_{j=1}^{t-1} \sum_{i=0}^{n-1} a_{j,i} {\ell_t}^j y^i \end{aligned}$$

This can be re-written in the matrix form as follows:

$$\begin{pmatrix} Q(\ell_1, y) \\ Q(\ell_2, y) \\ \vdots \\ Q(\ell_t, y) \end{pmatrix} = V \begin{pmatrix} F(y) \\ \sum_{j=1}^{t-1} a_{j,0}\, y^i \\ \vdots \\ \sum_{j=1}^{t-1} a_{j,n-1}\, y^i \end{pmatrix},$$

where the coefficient matrix

$$V = \begin{pmatrix} 1 & \ell_1 & {\ell_1}^2 & \cdots & {\ell_1}^{t-1} \\ 1 & \ell_2 & {\ell_2}^2 & \cdots & {\ell_2}^{t-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \ell_t & {\ell_t}^2 & \cdots & {\ell_t}^{t-1} \end{pmatrix}$$

is a Vandermonde matrix. As in the original (t, w)-Shamir scheme, the ℓ_m's are all distinct, so the matrix V is invertible [16]. Thus we obtain

$$V^{-1}\begin{pmatrix} Q(\ell_1, y) \\ Q(\ell_2, y) \\ \vdots \\ Q(\ell_t, y) \end{pmatrix} = \begin{pmatrix} F(y) \\ \sum_{j=1}^{t-1} a_{j,0}\, y^i \\ \vdots \\ \sum_{j=1}^{t-1} a_{j,n-1}\, y^i \end{pmatrix}. \tag{1}$$

Since we are only interested in the reconstruction of the function F, we simply need to know the first row of V^{-1}. Let $(v_{1,1},\ v_{1,2},\ \cdots,\ v_{1,t})$ be the first row of V^{-1}, (1) can be simplified as:

$$\begin{pmatrix} v_{1,1}\ v_{1,2} \cdots\ v_{1,t} \end{pmatrix} \begin{pmatrix} Q(\ell_1, y) \\ \vdots \\ Q(\ell_t, y) \end{pmatrix} = F(y).$$

Therefore

$$\begin{pmatrix} v_{1,1}\ v_{1,2} \cdots\ v_{1,t} \end{pmatrix} \begin{pmatrix} \mu_{0,\ell_1}\ \mu_{1,\ell_1}\ \cdots\ \mu_{n-1,\ell_1} \\ \vdots \quad \vdots \quad \ddots \quad \vdots \\ \mu_{0,\ell_t}\ \mu_{1,\ell_t}\ \cdots\ \mu_{n-1,\ell_t} \end{pmatrix} = \begin{pmatrix} \mu_0\ \mu_1 \cdots\ \mu_{n-1} \end{pmatrix}. \tag{2}$$

From (2) we conclude that the original set of secrets can be reconstructed as $\mu_i = \sum_{j=1}^{t} v_{1,j}\, \mu_{i,\ell_j}$ for $i \in [0, \ldots, n-1]$.

Lemma 1. *Equation* (2) *for reconstructing the original secret is the same as the Lagrange interpolation formula and* $v_{1,j} = \prod_{\substack{1 \le k \le t \\ k \neq j}} \frac{\ell_k}{\ell_k - \ell_j}$ *for* $1 \le j \le t$.

Lemma 2. *In Lagrange's interpolation formula, we have*

$$\sum_{j=1}^{t} \left(\prod_{\substack{1 \le k \le t \\ k \neq j}} \frac{\ell_k}{\ell_k - \ell_j} \right) = 1.$$

We sketch the proofs of Lemmas 1 and 2 in Appendix A.

Lemma 3. *Let* $b_j = \prod_{\substack{1 \le k \le t \\ k \neq j}} \frac{\ell_k}{\ell_k - \ell_j}$ *in the interpolation formula for* $j \in [1, \ldots, t]$. *Then* $\sum_{j=1}^{t} b_j(\mu_{i,\ell_j} - \mu_i) = 0$.

Proof. From Lemma 2, we know that $\sum_{j=1}^{t} b_j = 1$. We also know that $\sum_{j=1}^{t} \mu_{i,\ell_j}\, b_j = \mu_i$ by the interpolation formula. The conclusion is straightforward.

Note that we use the variant of ElGamal that is defined over $\mathbb{Z}_p$. Further in our paper, the computations are being done modulo p and we simplify the notation by skipping the modulus in the congruences. If we use different modulus, the congruence will be written in full to avoid confusion.

2.4 Adversary Model

We consider a semi-honest adversary model. Due to space constraints, we only provide the intuition and informal definitions of this model. The reader is referred to [6] for a more complete discussion.

In this model, there is no direct interaction between $\mathcal{C}$ and $\mathcal{P}$. Instead the client $\mathcal{C}$ and w servers are assumed to follow the steps defined in the protocol. The security definition is straightforward that only the client $\mathcal{C}$ learns the result of the protocol.

Definition 1. (t-secure). *A set operation protocol is said to be* t-secure *if the client $\mathcal{C}$, colluding with at most $t-1$ out of w servers learns no information about the provider's dataset and the set operation result.*

Following [12, 13] our model should meet the following requirements:

Correctness. A protocol is correct if the client $\mathcal{C}$ is able to compute the valid result from shares obtained from t servers (and additional information from a single server in the second round if this is applicable) assuming that each server and the client honestly follow the protocol.

***Client*'s security.** The protocol should guarantee the client privacy, i.e. the servers learn nothing about either the client inputs or its corresponding computed output. In other words, a server is not able to distinguish the client inputs from uniform random variables.

***Provider*'s security.** The protocol should not give out to the client any information about the function held by the provider apart from the output of the function assuming that no server colludes with the client. Also any $t-1$ or less servers should not able to find out any information about the function. We say that the provider privacy is t-secure.

3 Privacy-Preserving Set Intersection

We first define privacy-preserving set intersection. We then formulate a protocol allowing its computation and analyze its security.

Problem Definition: Assume that the provider $\mathcal{P}$ holds his dataset $\mathcal{D}_\mathrm{P} = \{\mu_0, \ldots, \mu_{n-1}\}$ and constructs and distributes the shares of the dataset as well as the random value λ in the way described in Sect. 2.3. Suppose further that the client $\mathcal{C}$ holds a dataset $\mathcal{D}_\mathrm{C} = \{c_0, \ldots, c_{m-1}\}$. Our protocol to compute the intersection of two sets $\mathcal{D}_\mathrm{P}$ and $\mathcal{D}_\mathrm{C}$ is defined as follows. The client $\mathcal{C}$ contacts t servers and fetches the information from them. Except $|\mathcal{D}_\mathrm{C}|$ and $|\mathcal{D}_\mathrm{P}|$ are publicly known, the information

Input: The client $\mathcal{C}$ has a set of data $\mathcal{D}_{\mathrm{C}}$. Each server $S_\ell (1 \leq \ell \leq w)$ knows the random value λ and the shared coefficients $\{\mu_{0,\ell}, \ldots, \mu_{n-1,\ell}\}$ of the function F (whose coefficients are the elements of the provider's dataset $\mathcal{D}_{\mathrm{P}}$).
Output: The client $\mathcal{C}$ learns $\mathcal{D}_{\mathrm{C}} \cap \mathcal{D}_{\mathrm{P}}$.

1. $\mathcal{C}$ generates a new key pair $(pk, sk) \leftarrow K(1^l), r \xleftarrow{\mathrm{R}} \mathbb{Z}_{\mathrm{q}}$, then
   (a) sets $\tau_\upsilon \leftarrow E_{\mathrm{pk}}(c_\upsilon) = (g^r, h^r f^{-c_\upsilon})$ for $\upsilon \in [0, \ldots, m-1]$.
   (b) broadcasts $\{ pk, \tau_0, \ldots, \tau_{m-1} \}$ to t servers $S_{\ell_1}, \ldots, S_{\ell_t}$.
2. For $j = 1, \ldots, t$ each contacted server S_{ℓ_j}
   (a) computes $\tau_{\upsilon,i,j} \leftarrow (g^{\lambda r}, h^{\lambda r} f^{\lambda(\mu_{i,\ell_j} - c_\upsilon)})$ for $\upsilon \in [0, \ldots, m-1]$ and $i \in [0, \ldots, n-1]$.
   (b) sends $\{\tau_{0,0,j}, \ldots, \tau_{0,n-1,j}, \tau_{1,0,j}, \ldots, \tau_{m-1,n-1,j} \}$ to $\mathcal{C}$.
3. For $\upsilon = 0, \ldots, m-1$, the client $\mathcal{C}$
   (a) computes $d_{\upsilon,i,j} \leftarrow D_{\mathrm{sk}}(\tau_{\upsilon,i,j})$ for $i \in [0, \ldots, n-1]$, $j \in [1, \ldots, t]$.
   (b) computes $d_{\upsilon,i} \leftarrow \prod_{j=1}^{t} (d_{\upsilon,i,j})^{b_j}$ for $i = 0, \ldots, n-1$, where $b_j = \prod_{\substack{1 \leq k \leq t \\ k \neq j}} \frac{\ell_k}{\ell_k - \ell_j}$ in the Lagrange interpolation formula.
   (c) concludes $c_\upsilon \in \mathcal{D}_{\mathrm{P}}$, if $d_{\upsilon,i} = 1$ for $i \in [0, \ldots, n-1]$; otherwise $d_{\upsilon,i}$ is a random integer.
4. When this process concludes, $\mathcal{C}$ learns $\mathcal{D}_{\mathrm{C}} \cap \mathcal{D}_{\mathrm{P}}$.

Fig. 1. Privacy-preserving set intersection protocol

allows the client to compute the intersection in such a way that the client learns nothing about the set $\mathcal{D}_{\mathrm{P}}$ and the servers do not learn anything about $\mathcal{D}_{\mathrm{C}}$.

We use Lemma 3 to construct a scheme allowing to compute the intersection of two private data sets held by $\mathcal{P}$ and $\mathcal{C}$. It is described in Fig. 1.

Correctness. In the above protocol, the client $\mathcal{C}$ first encrypts each element c_υ of her dataset by using her public key as $E_{\mathrm{pk}}(f^{-c_\upsilon})$ for $\upsilon \in [0, \ldots, m-1]$ and broadcasts all these encrypted elements to t servers. For each encrypted element $E_{\mathrm{pk}}(f^{-c_\upsilon})$, the servers S_{ℓ_j} $(1 \leq j \leq t)$ compute $E_{\mathrm{pk}}(f^{\lambda(\mu_{i,\ell_j} - c_\upsilon)})$ for $i \in [0, \ldots, n-1]$, and send all the $E_{\mathrm{pk}}(f^{\lambda(\mu_{i,\ell_j} - c_\upsilon)})$'s back to $\mathcal{C}$. The client $\mathcal{C}$ then decrypts those $E_{\mathrm{pk}}(f^{\lambda(\mu_{i,\ell_j} - c_\upsilon)})$'s and computes $f^{\lambda \sum_{j=1}^{t} b_j (\mu_{i,\ell_j} - c_\upsilon)}$ for each $i = 0, \ldots, n-1$. Note that if $c_\upsilon = \mu_i$ then $\sum_{j=1}^{t} b_j (\mu_{i,\ell_j} - c_\upsilon) = 0$. Therefore the client $\mathcal{C}$ learns that $c_\upsilon \in \mathcal{D}_{\mathrm{P}}$ if there exists $i \in [0, \ldots, n-1]$ such that $f^{\lambda \sum_{j=1}^{t} b_j (\mu_{i,\ell_j} - c_\upsilon)} = 1$. When all the steps are finished $\mathcal{C}$ learns $\mathcal{D}_{\mathrm{C}} \cap \mathcal{D}_{\mathrm{P}}$.

Efficiency. As far as the communication efficiency is concerned, the client $\mathcal{C}$ broadcasts a set of encrypted m secrets to t servers. Each contacted server S_{ℓ_j} responds with mn messages. So the communication complexity of this protocol is $O(tmn \times \log_2 p)$ bits.

As to the computation efficiency, the client $\mathcal{C}$ needs $m+2$ modular exponentiations and m modular multiplications to encrypt her dataset, tmn decryptions, tmn modular

exponentiations and $mn(t-1)$ modular multiplications for the Lagrange interpolation. Each contacted server S_{ℓ_j} needs to perform mn modular exponentiations and multiplications for processing its shares. So the computation complexity of this protocol is $O(tmn)$ modular multiplications considering that on single modular exponentiation takes at most $\lfloor \log_2(p-1) \rfloor$ modular multiplications using Fast Exponentiation Algorithm [14].

Security. The two theorems given below characterize the security of the set intersection protocol.

Theorem 1. *Given the set intersection protocol described in Fig. 1 and assuming that the underlying homomorphic encryption is semantically secure, then each of the contacted servers cannot distinguish inputs generated by the client $\mathcal{C}$ from random integers with a non-negligible probability.*

Proof (Sketch). For any $\upsilon \in [0, \ldots, m-1]$, the cryptogram $E_{\text{pk}}(c_\upsilon)$ is indistinguishable from a random integer chosen uniformly from the $\mathbb{Z}_{\text{q}}$ assuming intractability of the DDH problem. Therefore no server learns anything about the client input.

Theorem 2. *Assuming that the discrete logarithm problem is hard, the client $\mathcal{C}$ cannot compute any information about shared coefficients $\{\mu_{0,\ell}, \ldots, \mu_{n-1,\ell}\}$ $(1 \leq \ell \leq w)$ distributed by the provider $\mathcal{P}$. In addition $\mathcal{P}$'s privacy is* t-secure.

Proof (Sketch). In the system setting, λ is selected from the $\mathbb{Z}_{\text{q}}$ randomly and uniformly. When the client $\mathcal{C}$ decrypts the ciphertext from each of t servers, the decrypted $f^{\lambda(\mu_{i,\ell_j} - c_\upsilon)}$ is indistinguishable from a random integer chosen uniformly from the $\mathbb{Z}_{\text{q}}$ assuming the discrete logarithm problem is hard. Furthermore, $\mathcal{C}$ cannot compute anything from less than t servers information. This is guaranteed by the perfectness of Shamir secret sharing.

4 Subset Relation

In this section, we define privacy-preserving subset relation and then we apply the oblivious polynomial evaluation technique to formulate the protocol that allows to compute it and analyze its security.

Problem Definition: Assume that the provider $\mathcal{P}$ holds his dataset $\mathcal{D}_{\text{P}} = \{\mu_0, \ldots, \mu_{n-1}\}$ and constructs and distributes the shares of the dataset as well as the random value λ in the way described in Sect. 2.3. Suppose further that the client $\mathcal{C}$ holds a dataset $\mathcal{D}_{\text{C}} = \{c_0, \ldots, c_{m-1}\}$. The protocol to compute the subset relation of two sets $\mathcal{D}_{\text{P}}$ and $\mathcal{D}_{\text{C}}$ is defined as follows. The client $\mathcal{C}$ contacts t servers and fetches the information from them. Except $|\mathcal{D}_{\text{C}}|$ and $|\mathcal{D}_{\text{P}}|$ are publicly known, the information allows the client to compute if $\mathcal{D}_{\text{C}} \subseteq \mathcal{D}_{\text{P}}$ in such a way that the client learns nothing about the set $\mathcal{D}_{\text{P}}$ and the servers do not learn anything about $\mathcal{D}_{\text{C}}$.

Correctness. In the protocol given in Fig. 2, the client $\mathcal{C}$ first encrypts each element c_υ of her dataset by using her public key as $E_{\text{pk}}(f^{-c_\upsilon})$ for $\upsilon \in [0, \ldots, m-1]$,

Input: The client $\mathcal{C}$ has a set of data $\mathcal{D}_{\mathrm{C}}$. Each server $S_\ell (1 \le \ell \le w)$ knows the random value λ and the shared coefficients $\{\mu_{0,\ell}, \ldots, \mu_{n-1,\ell}\}$ of the function F (whose coefficients are the elements of the provider's dataset $\mathcal{D}_{\mathrm{P}}$).
Output: The client $\mathcal{C}$ obtains 1 if $\mathcal{D}_{\mathrm{C}} \subseteq \mathcal{D}_{\mathrm{P}}$; otherwise a random integer.

1. Steps 1 - 3(b) are same as the distributed set-intersection protocol (Fig. 1), except one more step is added before step 2(a), and $\tau_{\upsilon,i,j}$ should be $(g^r, h^r f^{\theta_\upsilon + (\mu_{i,\ell_j} - c_\upsilon)})$ in step 2(a) accordingly.
2. (a_0) generates θ_υ for $0 \le \upsilon \le m-1$ from $G(\lambda)$.
   (a) computes $\tau_{\upsilon,i,j} \leftarrow (g^r, h^r f^{\theta_\upsilon + (\mu_{i,\ell_j} - c_\upsilon)})$ for $\upsilon \in [0, \ldots, m-1]$ and $i \in [0, \ldots, n-1]$.
4. (a) $\mathcal{C}$ defines a polynomial W whose roots are $d_{\upsilon,i}$'s

$$\begin{aligned} W(y) &= (y - d_{0,0}) \ldots (y - d_{0,n-1})(y - d_{1,0}) \ldots (y - d_{m-1,n-1}) \\ &= \sum_{u=0}^{mn} a_u y^u. \end{aligned}$$

   (b) $\mathcal{C}$ randomly picks e from w and sends $\{ E_{\mathrm{pk}}(a_0), \ldots, E_{\mathrm{pk}}(a_{mn}) \}$ to the server e, where $E_{\mathrm{pk}}(a_u) = (g^{r'}, h^{r'} f^{a_u})$.
5. Server e
   (a) computes $R_\upsilon = h^{\sum_{j=0}^{mn} r' f^{j\theta_\upsilon}} f^{\sum_{j=0}^{mn} a_j f^{j\theta_\upsilon}}$ for $0 \le \upsilon \le m-1$.
   (b) computes $\hat{E} = \prod_{\upsilon=0}^{m-1} E_{\mathrm{pk}}(W(f^{\theta_\upsilon})) = \left(g^{r' \sum_{\upsilon=0}^{m-1} \sum_{j=0}^{mn} f^{j\theta_\upsilon}}, \prod_{\upsilon=0}^{m-1} R_\upsilon \right)$.
   (c) sends $\hat{E}$ back to $\mathcal{C}$.
6. $\mathcal{C}$ decrypts $E_{\mathrm{pk}}(\hat{E})$, and checks whether it is 1.

Fig. 2. Privacy-preserving subset relation protocol

and broadcasts all these encrypted elements to t servers. For every encrypted element $E_{\mathrm{pk}}(f^{-c_\upsilon})$, each contacted server $S_{\ell_j} (1 \le j \le t)$ computes $E_{\mathrm{pk}}(f^{\theta_\upsilon + (\mu_{i,\ell_j} - c_\upsilon)})$ for $0 \le i \le n-1$, and sends all the$E_{\mathrm{pk}}(f^{\theta_\upsilon + (\mu_{i,\ell_j} - c_\upsilon)})$'s back to $\mathcal{C}$. The client $\mathcal{C}$ decrypts those $E_{\mathrm{pk}}(f^{\theta_\upsilon + (\mu_{i,\ell_j} - c_\upsilon)})$'s, and computes $d_{\upsilon,i} = f^{\theta_\upsilon \sum_{j=1}^{t} b_j + \sum_{j=1}^{t} b_j (\mu_{i,\ell_j} - c_\upsilon)}$ for each $i = 0, \ldots, n-1$. By Lemma 2, we have $d_{\upsilon,i} = f^{\theta_\upsilon + \sum_{j=1}^{t} b_j (\mu_{i,\ell_j} - c_\upsilon)}$. Then $\mathcal{C}$ defines a polynomial W by using all the values $d_{\upsilon,i}$ as its roots, and sends the coefficients of the polynomial W to a single one of the w servers. Observe that if $c_\upsilon = \mu_i$ for $\upsilon \in [0, \ldots, m-1]$ and $i \in [0, \ldots, n-1]$, then $d_{\upsilon,i} = f^{\theta_\upsilon + \sum_{j=1}^{t} b_j (\mu_{i,\ell_j} - c_\upsilon)} = f^{\theta_\upsilon}$. This means that $W(f^{\theta_\upsilon}) = 0$. By the homomorphic property, the only contacted server e in the second round is able to compute $E_{\mathrm{pk}}(W(f^{\theta_\upsilon}))$. To save some computations, we only compute R_υ, the second part of $E_{\mathrm{pk}}(W(f^{\theta_\upsilon}))$ for $\upsilon \in [0, \ldots, m-1]$. Using all the R_υ's, server e then computes $\prod_{\upsilon=0}^{m-1} E_{\mathrm{pk}}(W(f^{\theta_\upsilon}))$. When $\mathcal{C}$ receives and decrypts $\prod_{\upsilon=0}^{m-1} E_{\mathrm{pk}}(W(f^{\theta_\upsilon}))$, it concludes $\mathcal{D}_{\mathrm{C}} \subseteq \mathcal{D}_{\mathrm{P}}$ if it is 1; otherwise $\mathcal{D}_{\mathrm{C}}$ is not a subset of $\mathcal{D}_{\mathrm{P}}$.

Efficiency. The protocol requires a two-round communication. First, the client $\mathcal{C}$ broadcasts a set of encrypted m secrets to t servers. Each contacted server S_{ℓ_j} responds with mn messages. In the second round, the client $\mathcal{C}$ sends $mn+1$ encrypted coefficients to a single server and receives only one message. The communication complexity of this protocol is still $O(tmn \times \log_2 p)$ bits.

As to the computation efficiency, the client $\mathcal{C}$ needs $m+2$ modular exponentiations and m modular multiplications to encrypt her own dataset, $tmn+1$ decryptions, tmn modular exponentiations and $(t-1)mn$ modular multiplications for the Lagrange interpolation, $2^{mn}-mn$ modular additions and $(mn-1)2^{mn}-mn+1$ modular multiplications for the polynomial expansion. Each contacted server S_{ℓ_j}, in the first round, needs to perform mn modular exponentiations and modular multiplications respectively for processing its shares. In the second round, the only contacted server e needs m^2n-m modular multiplications to precompute $f^{2\theta_\upsilon}, \ldots, f^{mn\theta_\upsilon}$ for $1 \leq \upsilon \leq m$. Server e also needs to perfomr m^2n modular multiplications and exponentiations respectively for computing all R_υ's. Denote $\hat{E}$ as $(\hat{L}, \hat{R})$, it needs $m(n+1)-1$ modular additions and one modular exponentiation to compute $\hat{L}$ and m modular multiplications to get $\hat{R}$. So the computation complexity of this protocol is $O(2^{mn})$ modular additions and multiplications respectively using the Fast Exponentiation Algorithm.

Security. The security model and its proof for this protocol is similar to that of the set-intersection protocol.

5 Cardinality of Set-Intersection

Using similar techniques as previous protocols, we develop an algorithm computing the cardinality of set intersection.

Problem Definition: Assume that the provider $\mathcal{P}$ holds his dataset $\mathcal{D}_\mathrm{P} = \{\mu_0, \ldots, \mu_{n-1}\}$ and constructs and distributes the shares of the dataset as well as the random value λ in the way described in Sect. 2.3. Suppose further that the client $\mathcal{C}$ holds a dataset $\mathcal{D}_\mathrm{C} = \{c_0, \ldots, c_{m-1}\}$. The protocol to compute the cardinality of two intersection sets $\mathcal{D}_\mathrm{P}$ and $\mathcal{D}_\mathrm{C}$ is defined as follows. The client $\mathcal{C}$ contacts t servers and fetches the information from them. Except $|\mathcal{D}_\mathrm{C}|$ and $|\mathcal{D}_\mathrm{P}|$ are public known, the information allows the client to compute $|\mathcal{D}_\mathrm{C} \cap \mathcal{D}_\mathrm{P}|$ in such a way that the client learns nothing about the set $\mathcal{D}_\mathrm{P}$ and the servers do not learn anything about $\mathcal{D}_\mathrm{C}$.

By using oblivious polynomial evaluation, we need to obtain the coefficients $a_0, \ldots, a_n$ of $W(y) = \sum_{i=0}^{n} a_i\, y^i$ from the factorization $\prod_{i=0}^{n-1}(y - c_i)$. To avoid this cost, we simply use the technique of random shuffling to permute the ordered encrypted results in the second round by the only contacted server.

The protocol, given in Fig. 3, works in the same way as the distributed subset relation protocol until Step 3(b). The client $\mathcal{C}$ then encrypts each of the values $d_{\upsilon,i}$ and sends all these encrypted elements $E_{\mathrm{pk}}(d_{\upsilon,i})$'s to a single one of the w servers. The only contacted server multiplies $f^{-\theta_\upsilon}$ to the second part of $E_{\mathrm{pk}}(d_{\upsilon,i})$, then raises the power of θ_υ to both components of $E_{\mathrm{pk}}(d_{\upsilon,i})$. When this process concludes, the server randomly permutes all the ciphertexts and returns them to $\mathcal{C}$. The client $\mathcal{C}$ then counts

Input: The client $\mathcal{C}$ has a set of data $\mathcal{D}_{\mathrm{C}}$. Each server $S_\ell (1 \leq \ell \leq w)$ knows the random value λ and the shared coefficients $\{\mu_{0,\ell}, \ldots, \mu_{n-1,\ell}\}$ of the function F (whose coefficients are the elements of the provider's dataset $\mathcal{D}_{\mathrm{P}}$).
Output: The client $\mathcal{C}$ learns $|\mathcal{D}_{\mathrm{C}} \cap \mathcal{D}_{\mathrm{P}}|$.

1. Steps 1 - 3(b) are same as the distributed subset relation protocol (Fig. 2).
3. (c) $\mathcal{C}$ randomly picks e from w and sends $\{ E_{\mathrm{pk}}(d_{0,0}), \ldots, E_{\mathrm{pk}}(d_{m-1,n-1}) \}$ to the server e, where $E_{\mathrm{pk}}(d_{\upsilon,i}) = (g^{r'}, h^{r'} d_{\upsilon,i})$.
4. Server e
   (a) computes
   $$E_{\mathrm{pk}}(d'_{\upsilon,i}) = \left(g^{\theta_\upsilon r'}, \left(h^{r'} d_{\upsilon,i} f^{-\theta_\upsilon}\right)^{\theta_\upsilon}\right) = \left(g^{\theta_\upsilon r'}, h^{\theta_\upsilon r'} f^{\theta_\upsilon \sum_{j=1}^{t} b_j (\mu_{i,\ell_j} - c_\upsilon)}\right)$$
   for $\upsilon \in [0, \ldots, m-1]$ and $i \in [0, \ldots, n-1]$.
   (b) shuffles all the $E_{\mathrm{pk}}(d'_{\upsilon,i})$ and sends the shuffled $\{\widehat{d'_1}, \ldots, \widehat{d'_{mn}}\}$ back to $\mathcal{C}$.
5. (a) $\mathcal{C}$ decrypts $D_{\mathrm{sk}}(\widehat{d'_\iota})$ and obtains 1 if one element of $\mathcal{D}_{\mathrm{C}}$, which is carried in the computation of $\widehat{d'_\iota}$, is in $\mathcal{D}_{\mathrm{P}}$; otherwise obtains a random integer.
   (b) $\mathcal{C}$ counts the number of ciphertexts received that decrypt to 1.

Fig. 3. Privacy-preserving cardinality of set-intersection protocol

the number of ciphertexts received that decrypt to 1. The security proof for this protocol trivially follows that of the subset relation protocol.

The efficiency is the same as the subset relation protocol for the first round. In the second round of this protocol, the client $\mathcal{C}$ sends mn messages to the server e and the server e returns mn messages back. As to the computation efficiency in the second round, the client $\mathcal{C}$ performs mn encryptions and decryptions respectively. Accordingly, the server e needs to perform mn modular multiplications and mn modular exponentiations to complete the cardinality computation.

6 Conclusion and Future Work

In this paper, we have proposed a protocol for the privacy-preserving set intersection computation in a distributed environment by combining Shamir's secret sharing scheme and homomorphic encryption scheme. Moreover, we have shown that with a second round of interaction and a random shuffling the cardinality of set-intersection could be computed efficiently. By using oblivious polynomial evaluation techniques, we have also constructed an two-round privacy-preserving subset relation protocol.

The further research will be to provide a solution of above distributed set operations against active adversary.

References

[1] B. Aiello, Y. Ishai, and O. Reingold. Priced oblivious transfer: How to sell digital goods. In B. Pfitzmann, editor, *Advances in Cryptology - Eurocrypt '01*, volume 2045 of *LNCS*, pages 119–135. Springer-Verlag Berlin Heidelberg, 2001.

[2] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *20th Annual ACM Symposium on Theory of Computing*, pages 1–10. ACM Press, 1988.

[3] R. Fagin, M. Naor, and P. Winkler. Comparing information without leaking it. *Communications of the ACM*, 39(5):77–85, 1996.

[4] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology - Eurocrypt '04*, volume 3024 of *LNCS*, pages 1–9. Springer-Verlag Berlin Heidelberg, 2004.

[5] T. E. Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and D. Chaum, editors, *Advances in Cryptology - Crypto '84*, volume 196, pages 19–22. Springer-Verlag, August 1984.

[6] O. Goldreich. *The Foundations of Cryptography*, volume 2. Cambridge University Press, 2004.

[7] S. Hohenberger and S. A. Weis. Honest-verifier private disjointness testing without random oracles. In *6th Workshop on Privacy Enhancing Technologies (PET'06)*, volume 4258 of *LNCS*, pages 277–294. Springer-Verlag Berlin Heidelberg, 2006.

[8] A. Kiayias and A. Mitrofanova. Testing disjointness and private datasets. In A. S. Patrick and M. Yung, editors, *Finanicial Cryptography (FC'05)*, volume 3570, pages 109–124. Springer-Verlag Berlin Heidelberg, 2005.

[9] L. Kissner and D. Song. Privacy-preserving set operaitons. In V. Shoup, editor, *Advances in Cryptology - Crypto '05*, volume 3621 of *LNCS*, pages 241–257. Springer-Verlag Berlin Heidelberg, 2005.

[10] H. Lipmaa. Verifiable homomorphic oblivious transfer and private equality test. In C. S. Laih, editor, *Advances in Cryptology - Asiacrypt '03*, volume 2894, pages 416–433. Springer-Verlag Berlin Heidelberg, 2003.

[11] P. Mohassel and M. Franklin. Efficient polynomial operations in the shared-coefficients setting. In M. Yung, editor, *Public Key Cryptography (PKC'06)*, volume 3958 of *LNCS*, pages 44–57. Springer-Verlag, 2006.

[12] M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. In *31st annual ACM Symposium on Theory of Computing (STOC '99)*, pages 245–254, Atlanta, Georgia, May 1999.

[13] M. Naor and B. Pinkas. Distributed oblivious transfer. In T. Okanoto, editor, *Advances in Cryptology - Asiacrypt '00*, volume 1976 of *LNCS*, pages 205–219. Springer-Verlag, 2000.

[14] J. Pieprzyk, T. Hardjono, and J. Seberry. *Fundamentals of Computer Security*. Springer, 2003.

[15] Y. Sang, H. Shen, Y. Tan, and N. Xiong. Efficient protocols for privacy preserving matching against distributed datasets. In *8th International Conference of Information and Communications Security (ICICS'06)*, volume 4307 of *LNCS*, pages 210–227. Springer - Verlag, 2006.

[16] D. R. Stinson. An explication of secret sharing scheme. *Designs, Codes and Cryptography*, 2:357–390, 1992.

[17] Y. Tsiounis and M. Yung. On the security of elgamal based encryption. In *Public Key Cryptography (PKC'98)*, volume 1431 of *LNCS*, pages 117–134. Springer-Verlag, 1998.

[18] A. C. Yao. Protocols for secure computations. In *23rd Symposium on Foundations of Computer Science (FOCS)*, pages 160–164. IEEE, 1982.

A Proof of Lemma 1 and 2

We have t participants and each of them has a share. Corresponding to the t points, the Vandermonde matrix V is constructed as follows:

$$V = \begin{pmatrix} 1 & x_{i_1} & \dots & x_{i_1}^{t-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{i_t} & \dots & x_{i_t}^{t-1} \end{pmatrix}.$$

Since the points are pairwise distinct, V is invertible. Let $V^{-1} = (v_{i,j})_{1 \le i,j \le t}$. By taking first row of V^{-1} and first column of V, we obtain $\sum_{j=1}^{t} v_{1,j} = 1$.

The t polynomials $P_1(x), \dots, P_t(x)$ are defined as $P_j(x) := \prod_{\substack{1 \le k \le t \\ k \ne j}} \frac{x - x_{i_k}}{x_{i_k} - x_{i_j}}$ for any $1 \le j \le t$. Note that these polynomials have a nice property, namely

$$\forall j \in [1, \dots, t], \;\; P_j(x_{i_e}) = \begin{cases} 1 & \text{if } j = e \\ 0 & \text{otherwise .} \end{cases}$$

Those polynomials also can be rewritten as: $\forall j \in [1, \dots, t] \;\; P_j(x) = \sum_{k=1}^{t} a_{j,k}\, x^{k-1}$ where each $a_{j,k} \in \mathbb{Z}_{\text{p}}$.

We now build a $t \times t$ matrix:

$$D = \begin{pmatrix} a_{1,1} & a_{2,1} & \cdots & a_{t,1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1,t} & a_{2,t} & \cdots & a_{t,t} \end{pmatrix}.$$

The j^{th} column of D represents the coefficients of $P_j(x)$. We claim that: $V^{-1} = D$. It is sufficient to prove that $V \times D$ is a identity matrix.

Let $V \times D = \mathcal{W} = (w_{\varsigma,\eta})_{\substack{1 \le \varsigma \le t \\ 1 \le \eta \le t}}$. We fix $\varsigma, \eta \in [1, \dots, t]$ the coefficient $w_{\varsigma,\eta}$ is obtained by using the ς^{th} row of V along with the η^{th} column of D as $w_{\varsigma,\eta} = \sum_{m=1}^{t} x_{i_\varsigma}^{m-1}\, a_{\eta,m}$. Notice that $w_{\varsigma,\eta} = P_\eta(x_{i_\varsigma})$. Using the previous property of the polynomial, we obtain

$$w_{\varsigma,\eta} = \begin{cases} 1 & \text{if } \eta = \varsigma \\ 0 & \text{otherwise .} \end{cases}$$

This property demonstrates that $\mathcal{W}$ is a identity matrix, which proves that $V^{-1} = D$.

Since the sum of the coefficients of the first row of V^{-1} is 1, we get $\sum_{j=1}^{t} v_{1,j} = \sum_{j=1}^{t} a_{j,1} = 1$. Notice that $a_{j,1}$ is the constant coefficient of $P_j(x)$, so $\forall j \in [1, \dots, t], \; a_{j,1} = P_j(0) = \prod_{\substack{1 \le k \le t \\ k \ne j}} \frac{x_{i_k}}{x_{i_k} - x_{i_j}}$. Combining the previous two findings, we can conclude that:

$$\sum_{j=1}^{t} \left(\prod_{\substack{1 \le k \le t \\ k \ne j}} \frac{x_{i_k}}{x_{i_k} - x_{i_j}} \right) = 1.$$