Efficient Unbalanced Private Set Intersection Protocol over Large-Scale Datasets Based on Bloom Filter

Ou Ruan(✉), Chaohao Ai, and Changwang Yan

School of Computer Science, Hubei University of Technology, Wuhan 430068, China
ruanou@hbut.edu.cn

Abstract. Private set intersection (PSI) is a special case of secure multi-party computation where participants can securely get their intersection without leaking additional information. Although many protocols have been proposed in recent years, they are still slightly inefficient when facing large-scale datasets in an unbalanced scenario. Client's running time of most PSI protocols is related to Server's set size, and it's very large when Client has a small set but Server has a large-scale set. In the paper, we propose an efficient unbalanced PSI protocol over large-scale datasets in the semi-honest model. By properly using the encrypted Bloom filter, randomized technique and multiply homomorphism of ElGamal cryptography, Client can get the intersection correctly and his running time is only linear with his set size and is independent of Server's set size. Thus, our protocol has a lightweight Client and performs better than other protocols when Server has a large-scale dataset. We also give a detailed experimental analysis with other related protocols, which shows our protocol is more efficient than others.

Keywords: private set intersection · unbalanced scenario · large-scale datasets · Bloom filter · ElGamal cryptography

1 Introduction

Private set intersection (PSI) allows two parties to learn the same items between them and not reveal any other information, which has many practical applications such as discovering relationship paths in social networks [24], jointly learning a neural network model [34], linear classifier through logistic regression [7], privacy-preserving running the standard face recognition algorithm [17], lightweight contact tracing [36], and so on.

The oblivious transfer (OT) and oblivious pseudo-random function (OPRF) are the main techniques to improve the efficiency of PSI protocols(e.g. [6,20]), even in the face of malicious adversaries (e.g. [26,32,33]). In terms of the balance between computation and communication, some protocols(e.g. [1,25]) using

Supported by Enterprise Technology Innovation Development Project of Hubei Province of China (No. 2021BAB009).

H. Duan et al. (Eds.): SecureComm 2023, LNICST 568, pp. 287–301, 2025.
https://doi.org/10.1007/978-3-031-64954-7_15

polynomial interpolation techniques were proposed. In many of these PSI protocols, there is an issue that the set size of the two parties should be the same. But in practice, two parties usually have a different set size, which is called the unbalanced scenario.

In an unbalanced scenario, the owner of the large set is called Server and the owner of the small set is Client. Generally speaking, the computing power of Client is far lower than that of Server. In 2017, Chen *et al.* [9] first proposed an unbalanced PSI protocol when Server's set is much larger than Client's, and one year later, Resende *et al.* [31] defined the terminology of unbalanced PSI protocol. Then, some unbalanced PSI protocols(e.g. [8,12,21,22]) were presented by using Bloom filter or homomorphic encryption technology to reduce the running time of Client. However, Client's calculation amount of these protocols becomes heavy when Server has a large amount of data because his calculation amount is related to the set size of Server, thus these unbalanced PSI protocols are not suitable for large-scale data scenarios.

Three main practical scenarios require large-scale PSI implementations, which are measuring ad conversion rates, security incident information sharing, and private contact discovery. It becomes a basic requirement that a PSI protocol can handle large-scale data. Many unbalanced PSI protocols(e.g. [4,13,19] [27–29]) for large-scale datasets were proposed and were committed to reducing the amount of computation of Client. However, these protocols still have some issues: (1) Client's calculation amount of some protocols have a small relationship with Server's data rather than that it is completely unrelated, then there has a large calculation amount on Client when the set size of Server is large; (2) The running time of these protocols is very large. To solve these problems, we propose a new unbalanced PSI protocol based on Bloom filter and ElGamal cryptography.

1.1 Our Contributions

In this paper, we consider an unbalanced scenario where the set size of Server is far larger than the set size of Client, and propose an effective PSI protocol. In our protocol, firstly, we use an efficient data structure - Bloom filter to reduce the amount of calculation of Client, and the running time of Client is almost constant when his set size is fixed. The operation of generating and encrypting the Bloom filter only needs to be done on Server, and Client just needs to hash its elements. Secondly, the Bloom filter is encrypted by the efficient encryption algorithm - ElGamal cryptosystem. In fact, the ElGamal cryptosystem cannot encrypt 0. But after our random processing, there is no 0 in the Bloom filter. Moreover, the ElGamal cryptosystem has homomorphic properties which can satisfy the multiplication of ciphertexts and the decryption result is 1 if all plaintexts are 1. Thirdly, we add a pre-processing stage to the protocol, where time-consuming operations can be put into. After pre-processing, the online running time of Server and Client is further reduced and mainly depended on Client's set size.

The contributions of this paper are as follows:

1) We propose an efficient unbalanced PSI protocol based on Bloom filter and ElGamal cryptography. In our PSI protocol, the running time of Client and

the online running time of Server are almost constant if the set size of Client is fixed. Thus our protocol is very suitable for the unbalanced large-scale datasets scenario.

2) We give a detailed experimental analysis with other protocols, which shows our protocol is more efficient than other related protocols: (a) Client in our protocol is lightweight, and is the most efficient especially when the set size of Server exceeds 2^{22}; (b) After pre-processing, Server's online running time of our protocol is decided only by the smaller Client's set size and is less than others when the size of Server exceeds 2^{22}.

1.2 Organizational Structure

The remaining part of this paper is organized as follows: in Sect. 2, we introduce related works; next, we present the used technologies in Sect. 3; in Sect. 4, we propose our protocol in detail and give its security analysis; then we describe our experimental results in Sect. 5; in the end, we make a summary of this paper in Sect. 6.

2 Related Works

As a special case of secure multi-party computation [10,18,30], PSI allows two parties to know their intersection without revealing any other information. In 1986, Meadows [23] proposed the earliest PSI scheme based on public key encryption, which is the main technique for PSI. And, many efficient PSI protocols(e.g. [2,11,14,37]) were presented based on homomorphic encryption. To improve its efficiency, other methods were introduced such as symmetric-key primitives [32], oblivious pseudorandom function (OPRF) [20], point-value polynomial [1], OT extension [25], and so on.

Most of these protocols are mainly designed for the balanced setting where the set size of two parties needs to be the same. But in practice, we often encounter such a situation where the data of Server is much more than that of Client, which is called an unbalanced environment. In 2017, Chen *et al.* [9] gave an unbalanced PSI protocol based on homomorphic encryption. Their protocol had communication complexity linear in Client's set size and logarithmic in Server's. Furthermore, an important feature of their protocol was that the workload of Client is much less than that of Server. One year later, Chen *et al.* [8] improved it based on OPRF and extended it to malicious security. The protocol [8] used a pre-processing phase to improve its performance and added flexibility to the parametrization.

Resende *et al.* [31] further improved the efficiency of Chen *et al.* [9] based on Cuckoo filters. They used a Cuckoo filter to optimize the protocol of Baldi *et al.* [3] and reduced the amount of data transmitted. Compared with the protocol [9], it was 59× less in data transmission and 76× faster with 10Gbps.

Later, Resende *et al.* [12] optimized the protocol [31] to reduce data exchange and the data stored by Client by using an efficient Rank based Select Filter(RSQF) instead of the Cuckoo filter. Comparing with the optimized protocol

presented by Chen *et al.* [8], it was faster in almost all scenarios except when the set size of Server is larger than 2^{16}. In 2020, Chase *et al.* [6] proposed a lightweight PSI protocol based on OT and OPRF in the unbalanced scenario. From the experimental results, their protocol was faster than protocols [8,12,31] in medium bandwidth.

Although these protocols are highly efficient when Server's set size is not large, the online operating efficiency of Client is significantly reduced when Server faces a large-scale scenario. How to design efficient PSI for large-scale datasets becomes a big challenge. Huang *et al.* [19] showed three types of protocols for unbalanced large-scale datasets and domains by applying garbled circuits. Later, Pinkas *et al.* [28] improved the protocol [19] by using OT extension. One year later, Pinkas *et al.* [27] further optimized them by using Cuckoo hash and OT technology. It was faster and had lower communication than the OT-based PSI protocol [28]. And, Pinkas *et al.* [29] made improvements based on protocols [27,28] and proposed a new version of the OT-based PSI protocol based on an efficient OPRF.

In 2017, an efficient PSI protocol was proposed by Davidson et al. [13], which was based on an encrypted Bloom filter and Paillier encryption method. The protocol only needed one round of communication and Client's running time is almost entirely determined by his set size. After that, Bay et al. [4] improved and proposed a multi-party PSI protocol. Because of using Paillier encryption, their computation time is very large, especially in the face of large-scale datasets.

Even though Client's running time of the above protocols was reduced obviously and had only a small relationship with the set size of Server, when facing large-scale datasets, the running time of these protocols was still large and could not be well applied in practice.

3 Preliminaries

3.1 Notation

We show the notations used in this paper in Table 1.

Table 1. Table of Notation

Notation	Description
p	A large prime
Z_p^*	A multiplication group which has $p-1$ elements
$rand()$	Generate a random number of Z_p^*
BF_X	A Bloom filter of size m on the set X
EBF	Entry-wise encryption of a Bloom filter
$Enc(M)$	Generic public-key encryption of M
$Dec(C)$	Decryption of C

3.2 Bloom Filter

Bloom filter [5] was proposed by Bloom in 1970. It's actually a long binary vector and a series of random mapping functions, which can be used to retrieve whether an element is in a set. For a Bloom filter, there are three main operations: initialization, insertion and query.

Initialization: Prepare a bit array with a length of m and set all entries to 0.

Insertion: When an element is added to the set, k hash functions are used to map an element to k entries of the bit array and set them to 1.

Query: To see if an element is in the set, we just need to see whether all its mapped entries are 1.

From Dong *et al.* [15], there have $m \geq -\frac{n \ln p_{err}}{(\ln 2)^2}$ and $k = -\log_2 p_{err}$, where p_{err} denotes the probability of misjudgment, n is the size of a data set, m is the length of Bloom filter, and k is the optimal number of hash functions.

Definition 1. (Encrypted Bloom Filter)*: For a given Bloom filter BF_X with m entries of a data set X, the corresponding encrypted Bloom filter is denoted by EBF_X, $EBF_{X,j} = Enc_{pk}(BF_{X,j})$ for $j \in \{1, ..., m\}$ where pk is the public key of an encryption algorithm.*

3.3 ElGamal Cryptography

ElGamal cryptosystem [16], proposed by ElGamal in 1985, is an asymmetric encryption algorithm, which is based on the difficulty of the discrete logarithm problem. It has three algorithms: key generation, encryption, and decryption.

Key Generation: Let p be a large prime and g denote a primitive element of Z_p^*, $(g < p)$, Z_p^* is a multiplication group that has $p - 1$ elements. Choose a random secret key α $(\alpha < p)$, and compute the public key $\beta = g^{\alpha} mod\ p$. Here Z_p^*, β, g and p are public information, while α is the private key.

Encryption: Select a random value $r \in Z_p$ and compute $y_1 = g^r mod\ p$ and $y_2 = m\beta^r mod\ p$, where m is plaintext. Ciphertext $C = (y_1, y_2)$.

Decryption: After getting ciphertext C, the plaintext m can be got from the following equation:

$$m = y_2(y_1^{\alpha})^{-1} mod\ p \tag{1}$$

Definition 2. (Multiply homomorphism)*: Give two plaintext m_1 and m_2, $Enc(m_1)$ and $Enc(m_2)$ represent the encrypted ciphertext of m_1 and m_2 respectively. The multiply homomorphism of ElGamal encryption algorithm can be defined by the following equation:*

$$Dec(Enc(m_1) \times Enc(m_2)) = m_1 \times m_2 \tag{2}$$

ElGamal cryptosystem enjoys the above multiply homomorphism.

3.4 Security Model

We use the following security definitions to illustrate the security of our protocol.

Definition 3. (Negligible Function $negl(n)$)*:*

Give a security parameter n, in general, we can think of n as the length of the key used in encryption and decryption. We call a function $negl(n)$ a negligible function if it satisfies the following conditions: for every possible polynomial p and sufficiently large n, it holds that $negl(n) < \frac{1}{p(n)}$.

Definition 4. (Computational Indistinguishability)*:*

Assume $Pr[equa]$ represent the probability that an equation $equa$ is true, and let $X = \{X_\lambda\}_{\lambda \in N}$ and $Y = \{Y_\lambda\}_{\lambda \in N}$ be two probability ensembles indexed by N. We say that the ensembles are computationally indistinguishable if for any probabilistic polynomial time algorithm $\mathcal{A}$ they meet the following condition:

$$|Pr[\mathcal{A}(X_\lambda) \to 1] - Pr[\mathcal{A}(Y_\lambda) \to 1]| \leq negl(n) \tag{3}$$

We call X and Y which satisfy this situation $X \simeq Y$.

Definition 5. (Semi-honest Security)*:*

Let $f(X, Y) = (\perp, X \cap Y)$ be a functionality where a Server with an input X and a Client with an input Y run together and Client learns the intersection $X \cap Y$ but Server gets nothing. Π is a protocol for running the functionality $f(X, Y)$. Define the view of Π for Server to be $View_S^\Pi(X, Y) = (X, r_1, \ell_S, out_S)$ and Client to be $View_C^\Pi(X, Y) = (Y, r_2, \ell_C, out_C)$, where r_1 and r_2 represent internal coin tosses, ℓ_S and ℓ_C are the messages received by Server and Client respectively, out_S and out_C denote the output of Server and Client.

We say that the protocol Π securely computes f if there exists polynomial-time simulators Sim_S and Sim_C where $Sim_S(X, \perp)$ and $View_S^\Pi(X, Y)$, $Sim_C(Y, X \cap Y)$ and $View_C^\Pi(X, Y)$ are computationally indistinguishable.

$$\begin{aligned} \{View_S^\Pi(X, Y)\} &\simeq \{Sim_S(X, \perp)\} \\ \{View_C^\Pi(X, Y)\} &\simeq \{Sim_C(Y, X \cap Y)\} \end{aligned} \tag{4}$$

4 Our Unbalanced PSI Protocol over Large-Scale Datasets

Let Server and Client be the two parties involved in the protocol. Server has a private data set X of n_1 elements and Client has another data set Y of n_2 elements. The purpose of the protocol is to calculate the intersection of Server and Client, but not get any other information. Figure 1 formally describes our unbalanced PSI protocol.

In the protocol, Server generates a Bloom filter for his large-scale datasets X and then encrypts it with the ElGamal encryption algorithm. Because the ElGamal encryption algorithm cannot encrypt 0, Server gets the encrypted Bloom

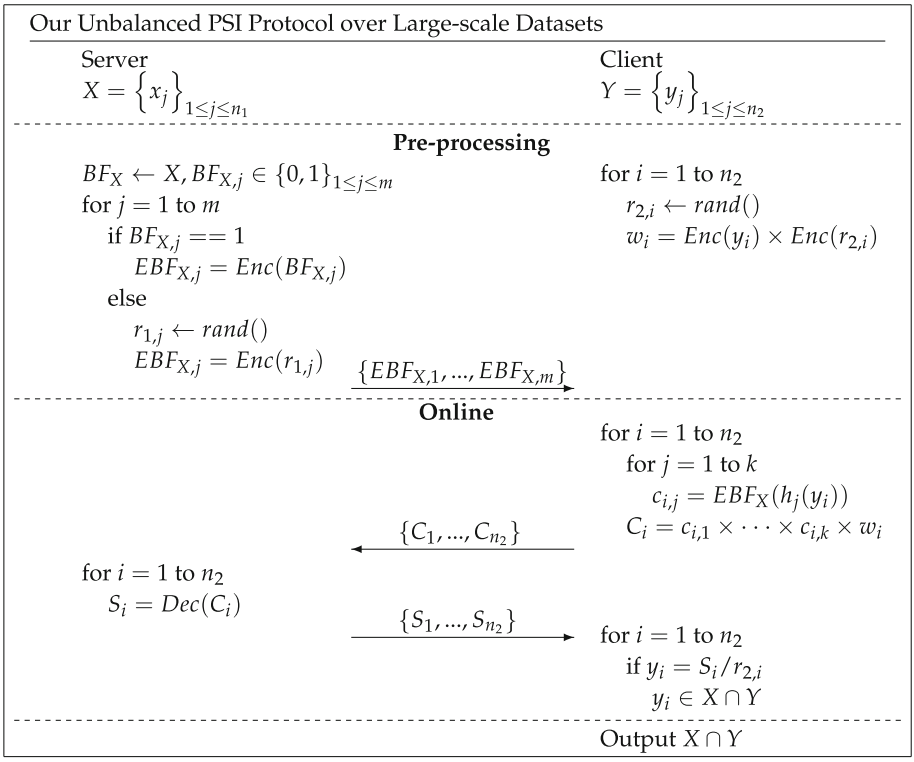


Fig. 1. Our Unbalanced PSI Protocol over Large-scale Datasets

filter using random numbers instead of 0. Client hashes each of his elements and gets its items of encrypted Bloom filter, and computes the multiplication of these items and encryptions of his element and a random number, then he can judge whether the element is in the set X by using the multiply homogenicity of ElGamal cryptosystem. For Server and Client, the time-consuming operations are both ElGamal encryptions and these operations are independent of each other, thus, they can be put into the pre-processing stage.

The detailed steps are as follows:

Input: Server holds a decryption key sk of ElGamal encryption scheme and a data set $X = \{x_j\}$ for $j \in \{1, ..., n_1\}$, and Client has a data set $Y = \{y_j\}$ for $j \in \{1, ..., n_2\}$.

Pre-processing:

1) Server uses the set X to generate Bloom filter BF_X, the size of Bloom filter m and the number of hash functions k are all obtained according to the set size of Server.
2) Server encrypts BF_X with the public key pk and gets the encrypted Bloom filter EBF_X. If $BF_{X,j} = 1$ in the Bloom filter, it is encrypted directly,

$EBF_{X,j} = Enc(BF_{X,j})$; otherwise, a random number $r_{1,j}$ is encrypted and $EBF_{X,j} = Enc(r_{1,j})$ for $j \in \{1, ..., m\}$ and $r_{1,j} \in Z_p^*$.
3) Client generates n_2 random number $r_{2,i}$ and gets $w_i = Enc(y_i) \times Enc(r_{2,i})$ for $i \in \{1, ..., n_2\}$ and $r_{2,i} \in Z_p^*$.
4) Server sends EBF_X and the hash functions $\{ h_1, h_2, ..., h_k\}$ to Client.

Online:

1) Client computes k hash values of each element y_i, $c_{i,j} = EBF_X(h_j(y_i))$ for $i \in \{1, ..., n_2\}$ and $j \in \{1, ..., k\}$, $C_i = c_{i,1} \times c_{i,2} \cdots \times c_{i,k} \times w_i$ for $i \in \{1, ..., n_2\}$, then Client sends $\{C_1, ..., C_{n_2}\}$ to Server.
2) Server decrypts $\{C_1, ..., C_{n_2}\}$ and sends $S_i = Dec(C_i)$ for $i \in \{1, ..., n_2\}$ to Client.
3) Client gets the intersection. For each $i \in \{ 1, ..., n_2\}$, if $y_i = S_i/r_{2,i}$, Client adds y_i to the intersection.

Output: Client outputs the intersection.

4.1 Protocol Correctness

Client can compute the intersection correctly from the following equation:
for $i \in \{ 1, ..., n_2\}$

$$\begin{aligned} y_i &= S_i/r_{2,i} \\ &= Dec(C_i)/r_{2,i} \\ &= Dec\{c_{i,1} \times \cdots \times c_{i,k} \times Enc(y_i) * Enc(r_{2,i})\}/r_{2,i} \\ &= \{BF_X(h_1(y_i)) \times \cdots \times BF_X(h_k(y_i)) \times y_i \times r_{2,i}\}/r_{2,i} \\ &= BF_X(h_1(y_i)) \times \cdots \times BF_X(h_k(y_i)) \times y_i \end{aligned}$$

if $y_i = S_i/r_{2,i}$
$BF_X(h_1(y_i)) \times \cdots \times BF_X(h_k(y_i)) = 1$
then $BF_X(h_1(y_i)) = 1, \ldots, BF_X(h_k(y_i)) = 1$
else
$BF_X(h_1(y_i)) \times \cdots \times BF_X(h_k(y_i)) \neq 1$
then $\exists j \in \{ 1, ..., k\}\ BF_X(h_j(y_i)) \neq 1$

4.2 Security Analysis

Theorem 1. *If EBF is an encrypted Bloom filter with a secure ElGamal scheme, the protocol* Π *shown in Fig. 1 is secure in the semi-honest model when parameters m and k are got as Sect. 3.*

Proof. We prove its security by considering two cases that Server is corrupted or Client is corrupted.

Case 1: Server corrupted.

In the real protocol, $View_S^{\Pi}(X, Y) = (X, EBF_X, \{C_1, ..., C_{n_2}\}, \{S_1, ..., S_{n_2}\})$. A simulator Sim_S is constructed with Server's input and performs the following steps:

1) Create an empty view $Sim_S(X, \perp)$ and then append X to the view.
2) Create a random set Y' with n_2 elements.
3) Using the set X to generate BF_X and encrypt BF_X to get EBF_X.
4) Compute $c_{i,j}^{'} = EBF_X(h_j(y_i^{'}))$ for $i \in \{1, ..., n_2\}$ and $j \in \{1, ..., k\}$.
5) Generate a random value $r_{2,i}^{'}$ and compute $C_i^{'} = c_{i,1}^{'} \times c_{i,2}^{'} \cdots \times c_{i,k}^{'} \times Enc(y_i^{'}) \times Enc(r_{2,i}^{'})$ for $i \in \{1, ..., n_2\}$.
6) Decrypt $C_i^{'}$ and get $S_i^{'} = Dec(C_i^{'}) = BF_X(h_1(y_i^{'})) \times \cdots \times BF_X(h_k(y_i^{'})) \times y_i^{'} \times r_{2,i}^{'}$ for $i \in \{1, ..., n_2\}$.
7) Insert EBF_X, $C_i^{'}$ and $S_i^{'}$ to $Sim_S(X, \perp)$. So, the view of Sim_S is $Sim_S(X, \perp) = (X, EBF_X, \{C_1^{'}, ..., C_{n_2}^{'}\}, \{S_1^{'}, ..., S_{n_2}^{'}\})$.

In $View_S^{\Pi}(X, Y) = (X, EBF_X, \{C_1, ..., C_{n_2}\}, \{S_1, ..., S_{n_2}\})$ and $Sim_S(X, \perp) = (X, EBF_X, \{C_1^{'}, ..., C_{n_2}^{'}\}, \{S_1^{'}, ..., S_{n_2}^{'}\})$, the X and EBF_X are identical. ElGamal cryptosystem and random values are used to encrypt the elements in the simulation. So, $C_i^{'}$ and C_i, $S_i^{'}$ and S_i are computationally indistinguishable for $i \in \{1, ..., n_2\}$.

As described above, $View_S^{\Pi}(X, Y) \simeq Sim_S(X, \perp)$. This means that $View_S^{\Pi}(X, Y)$ and $Sim_S(X, \perp)$ are computationally indistinguishable.

Case 2: Client corrupted.

In the real protocol, $View_C^{\Pi}(X, Y) = (Y, EBF_X, \{C_1, ..., C_{n_2}\}, \{S_1, ..., S_{n_2}\}, X \cap Y)$. A simulator Sim_C is constructed with Client's input and output and performs the following steps:

1) Create an empty view $Sim_C(Y, X \cap Y)$ and then append Y and $X \cap Y$ to the view.
2) Pick a random set X' that satisfies $X^{'} \cap Y = X \cap Y$.
3) Using the set $X^{'}$ to generate $BF_X^{'}$ and encrypt $BF_X^{'}$ to get $EBF_X^{'}$.
4) Compute $c_{i,j}^{'} = EBF_X^{'}(h_j(y_i))$ for $i \in \{1, ..., n_2\}$ and $j \in \{1, ..., k\}$.
5) Generate a random value $r_{2,i}^{'}$ and compute $C_i^{'} = c_{i,1}^{'} \times c_{i,2}^{'} \cdots \times c_{i,k}^{'} \times Enc(y_i) \times Enc(r_{2,i}^{'})$ for $i \in \{1, ..., n_2\}$.
6) Decrypt $C_i^{'}$ and get $S_i^{'} = Dec(C_i^{'}) = BF_X^{'}(h_1(y_i)) \times \cdots \times BF_X^{'}(h_k(y_i)) \times y_i \times r_{2,i}^{'}$ for $i \in \{1, ..., n_2\}$.
7) Insert $EBF_X^{'}$, $C_i^{'}$ and $S_i^{'}$ to $Sim_C(Y, X \cap Y)$. Therefore, the view of Sim_C is $Sim_C(Y, X \cap Y) = (Y, EBF_X^{'}, \{C_1^{'}, ..., C_{n_2}^{'}\}, \{S_1^{'}, ..., S_{n_2}^{'}\}, X \cap Y)$.

In $View_C^{\Pi}(X, Y) = (Y, EBF_X, \{C_1, ..., C_{n_2}\}, \{S_1, ..., S_{n_2}\}, X \cap Y)$ and $Sim_C(Y, X \cap Y) = (Y, EBF_X^{'}, \{C_1^{'}, ..., C_{n_2}^{'}\}, \{S_1^{'}, ..., S_{n_2}^{'}\}, X \cap Y)$, the Y and $X \cap Y$ are identical. Elgamal cryptosystem and random values are used to encrypt the elements in the simulation. So, $EBF_X^{'}$ and EBF_X, $C_i^{'}$ and C_i, $S_i^{'}$ and S_i are computationally indistinguishable for $i \in \{1, ..., n_2\}$. This means that $View_C^{\Pi}(X, Y)$ and $Sim_C(Y, X \cap Y)$ are computationally indistinguishable.

From the above two cases, we can infer the protocol is secure in the semi-honest model. □

5 Implementation and Performance Analysis

In this section, we present the implementations of our protocol and other related protocols [4,6,12,13], and discuss the performance with a detailed analysis of the experimental results.

5.1 Set-Up

We implement our protocol and other protocols [4,6,12,13] in Ubuntu 18.04 with the system of Linux on Inter Core i7-1165G7 with 2.8 GHz. These implementations depend on the Number Theory Library (NTL [35]) and the GNU Multiple Precision Arithmetic Library (GMP). We set the false positive probability p_{err} to 2^{-30} and use the optimal Bloom filter parameters with the size of Bloom filter $m = -\frac{n_1 \ln p_{err}}{(\ln 2)^2}$ and the number of hash functions $k = -\log_2 p_{err}$ where n_1 is the set size of Server. For ElGamal cryptosystem, we run with moduli p with bit-lengths 1024. In the experiments, the set size of Client is fixed to 2^{12} and the set size of Server is from 2^{12} to 2^{24}.

5.2 Experimental Results

For each experiment, we execute every protocol 10 times and then take the average value as the running time.

Table 2 is a comparison of the full running time of our protocol and other related protocols [4,6,12,13], which shows the full running time of Server and Client separately in different server set size.

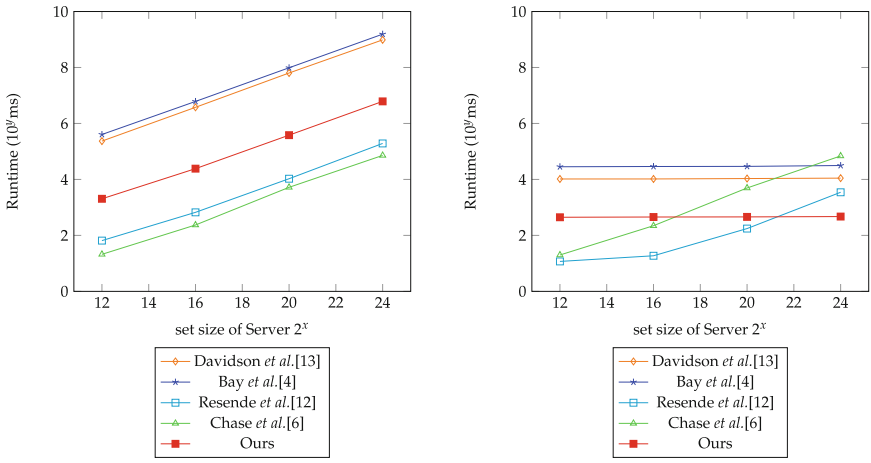
Then, we analyze all these protocols to distinguish the pre-process operations and the online operations and show them in Table 3 which describes the pre-processing time and the online running time of these protocols in detail, where Server-Pre and Client-Pre represent the pre-processing time of Server and Client respectively, Server-Online and Client-online denote the online running time of Server and Client respectively. For a more intuitive comparison, we present two line charts in Fig. 2 and Fig. 3, where the X-axis is the set size of Server and the

Table 2. Full running time in *ms* of our Protocol and related Others

Set size of server		Davidson [13]	Bay [4]	Resende [12]	Chase [6]	Ours
2^{12}	Server	234440.1	404270.5	65.1	21.1	2031.3
	Client	21462.2	16335.6	1.4	24.6	140.1
2^{16}	Server	3778788.3	6134676.9	666.1	235.3	24292.1
	Client	21580.9	17087.6	16.9	26.4	164.8
2^{20}	Server	63608536.8	97825481.2	10599.9	5224.7	381997.5
	Client	**22468.8**	**18772.2**	**326.3**	**66.7**	**183.9**
2^{24}	Server	971338552.1	1553296207.9	192997.5	72178.3	6158104.9
	Client	**23118.8**	**20532.1**	**6776.2**	**667.5**	**206.4**

Table 3. Detail running time in *ms* of our Protocol and related Others

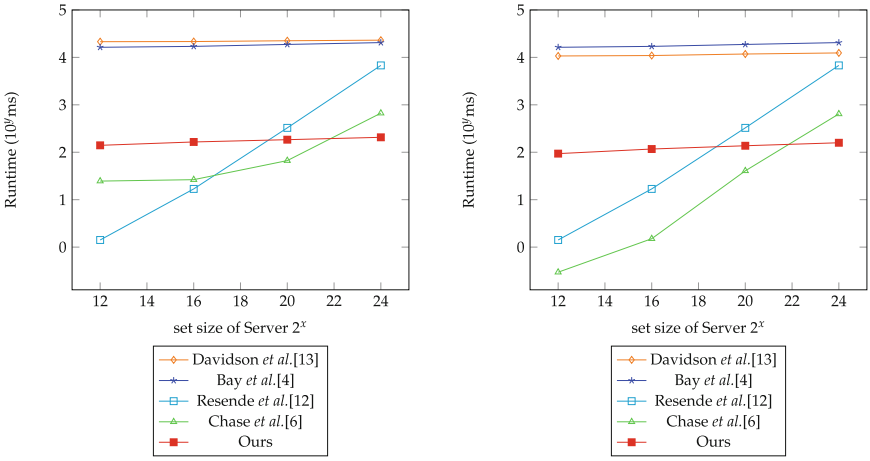
Set Size		Davidson [13]	Bay [4]	Resende [12]	Chase [6]	Ours
2^{12}	Server-Pre	224047.9	375865.8	53.3	1.1	1586.1
	Client-Pre	10740.8	0.0	0.0	24.4	46.6
	Server-Online	10392.2	28404.8	11.8	19.9	445.1
	Client-Online	10721.4	16335.6	1.4	0.3	93.5
2^{16}	Server-Pre	3768372.8	6105637.5	647.4	13.8	23838.6
	Client-Pre	10621.6	0.0	0.0	24.9	47.8
	Server-Online	10415.6	29039.4	18.7	221.6	453.5
	Client-Online	10959.3	17087.6	16.9	1.5	116.9
2^{20}	Server-Pre	63597748.0	97796238.0	10424.4	281.4	381539.3
	Client-Pre	10671.5	0.0	0.0	26.1	46.9
	Server-Online	**10788.8**	**29243.2**	**175.5**	**4943.3**	**458.2**
	Client-Online	**11797.4**	**18772.2**	**326.4**	**40.6**	**137.0**
2^{24}	Server-Pre	971327428.0	1553264820.0	189522.8	3095.7	6157634.0
	Client-Pre	10713.9	0.0	0.0	24.2	47.9
	Server-Online	**11124.2**	**31387.9**	**3474.7**	**69082.7**	**470.9**
	Client-Online	**12404.9**	**20532.1**	**6776.2**	**643.3**	**158.5**



(a) Full running time (b) Online running time

Fig. 2. The running time of Server

Y-axis indicates the running time of Server or Client. Figure 2 represents the full running time and online running time of Server respectively, and Fig. 3 denotes the full running time and online running time of Client respectively.



(a) Full running time (b) Online running time

Fig. 3. The running time of Client

5.3 Result Analysis

The main consideration in this paper is time overhead rather than communication overhead. In Table 2, Fig. 2-a and Fig. 3-a, we show the full running time of Server and Client of different protocols. From Table 2 and Fig. 2-a, we can see that although Client's set size is fixed, the full running time of Server for all protocols increases as the Server's set size increases. And, from Table 2 and Fig. 3-a, the Client's running time of our protocol and protocols [4,13] is mainly determined by Client's own set size, and as the set size of Server gradually becomes larger, the advantages of our protocol become more and more obvious. When the set size of Server is greater than 2^{22}, Client's full running time of our protocol is minimal.

In Table 3, we give the pre-processing time and online running time of different protocols. Our main concern is the online running time of Server and Client, which is shown in Fig. 2-b and Fig. 3-b. From them, we can find that the online running time of Server and Client in Resende *et al.* [12] and Chase *et al.* [6] protocols increase significantly as the set size of Server increases, but the online running time of our protocol and protocols [4,13] is independent of the set size of Server and is almost a constant value. Since many computations are done in the pre-processing phase, our protocol is more efficient than protocols [4,13], and the online running time of our protocol is the least when Server's set size exceeds 2^{22}.

The advantages of our protocol can be summarized as follows:

1) In our protocol, the full running time, online running time of Client and the online running time of Server after pre-processing depend mainly on Client's

set size, and there is basically no effect as Server's size increases. Thus, our protocol is very suitable for the unbalanced large-scale datasets scenario.
2) Client in our protocol is lightweight. Client's full running time and online time of our protocol is more efficient than others' when the size of Server exceeds 2^{22}.
3) Server's online running time of our protocol after pre-processing is more efficient than others when the size of Server exceeds 2^{22}.

6 Conclusion

We proposed an efficient unbalanced PSI protocol based on Bloom filter and ElGamal cryptography. In the PSI protocol, there is a lightweight Client, and Client's running time is almost constant if its set size is fixed, thus it is very suitable for the unbalanced large-scale datasets scenario. We also gave a detailed experimental analysis which showed our protocol was more efficient than other works. Extending our protocol to multiple parties' environments and malicious security are our future works.

References

1. Abadi, A., Terzis, S., Dong, C.: O-PSI: delegated private set intersection on outsourced datasets. In: Federrath, H., Gollmann, D. (eds.) SEC 2015. IAICT, vol. 455, pp. 3–17. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18467-8_1
2. Abadi, A., Terzis, S., Metere, R., Dong, C.: Efficient delegated private set intersection on outsourced private datasets. IEEE Trans. Dependable Secure Comput. **16**(4), 608–624 (2017)
3. Baldi, P., Baronio, R., De Cristofaro, E., Gasti, P., Tsudik, G.: Countering gattaca: efficient and secure testing of fully-sequenced human genomes. In: Proceedings of the 18th ACM Conference on Computer and Communications Security, pp. 691–702 (2011)
4. Bay, A., Erkin, Z., Hoepman, J.H., Samardjiska, S., Vos, J.: Practical multi-party private set intersection protocols. IEEE Trans. Inf. Forensics Secur. **17**, 1–15 (2021)
5. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7), 422–426 (1970)
6. Chase, M., Miao, P.: Private set intersection in the internet setting from lightweight oblivious PRF. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. LNCS, vol. 12172, pp. 34–63. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56877-1_2
7. Chaudhuri, K., Monteleoni, C.: Privacy-preserving logistic regression. In: Proceedings of NIPS, pp. 289–296 (2008)
8. Chen, H., Huang, Z., Laine, K., Rindal, P.: Labeled PSI from fully homomorphic encryption with malicious security. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 1223–1237 (2018)
9. Chen, H., Laine, K., Rindal, P.: Fast private set intersection from homomorphic encryption. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 1243–1255 (2017)

10. Cohen, R., Haitner, I., Omri, E., Rotem, L.: From fairness to full security in multiparty computation. J. Cryptol. **35**(1), 1–70 (2022)
11. Cong, K., et al.: Labeled PSI from homomorphic encryption with reduced computation and communication. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, pp. 1135–1150 (2021)
12. Davi Resende, A.C., de Freitas Aranha, D.: Faster unbalanced private set intersection in the semi-honest setting. J. Cryptogr. Eng. **11**(1), 21–38 (2021)
13. Davidson, A., Cid, C.: An efficient toolkit for computing private set operations. In: Pieprzyk, J., Suriadi, S. (eds.) ACISP 2017. LNCS, vol. 10343, pp. 261–278. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59870-3_15
14. Debnath, S.K., Stanica, P., Kundu, N., Choudhury, T.: Secure and efficient multiparty private set intersection cardinality. Adv. Math. Commun. **15**(2), 365 (2021)
15. Dong, C., Chen, L., Wen, Z.: When private set intersection meets big data: an efficient and scalable protocol. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, pp. 789–800 (2013)
16. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. Inf. Theory **31**(4), 469–472 (1985)
17. Erkin, Z., Franz, M., Guajardo, J., Katzenbeisser, S., Lagendijk, I., Toft, T.: Privacy-preserving face recognition. In: Goldberg, I., Atallah, M.J. (eds.) PETS 2009. LNCS, vol. 5672, pp. 235–253. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03168-7_14
18. Halevi, S., Hazay, C., Polychroniadou, A., Venkitasubramaniam, M.: Round-optimal secure multi-party computation. J. Cryptol. **34**(3), 1–63 (2021)
19. Huang, Y., Evans, D., Katz, J.: Private set intersection: are garbled circuits better than custom protocols? In: Network and Distributed Security Symposium (NDSS 2012) (2012)
20. Kolesnikov, V., Kumaresan, R., Rosulek, M., Trieu, N.: Efficient batched oblivious PRF with applications to private set intersection. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 818–829 (2016)
21. Lv, S., et al.: Unbalanced private set intersection cardinality protocol with low communication cost. Futur. Gener. Comput. Syst. **102**, 1054–1061 (2020)
22. Ma, J.P., Chow, S.S.: Secure-computation-friendly private set intersection from oblivious compact graph evaluation. In: Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security, pp. 1086–1097 (2022)
23. Meadows, C.: A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In: 1986 IEEE Symposium on Security and Privacy, p. 134. IEEE (1986)
24. Mezzour, G., Perrig, A., Gligor, V., Papadimitratos, P.: Privacy-preserving relationship path discovery in social networks. In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) CANS 2009. LNCS, vol. 5888, pp. 189–208. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10433-6_13
25. Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: SpOT-light: lightweight private set intersection from sparse OT extension. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11694, pp. 401–431. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26954-8_13
26. Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: PSI from PaXoS: fast, malicious private set intersection. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12106, pp. 739–767. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45724-2_25

27. Pinkas, B., Schneider, T., Segev, G., Zohner, M.: Phasing: private set intersection using permutation-based hashing. In: 24th USENIX Security Symposium (USENIX Security 2015), pp. 515–530 (2015)
28. Pinkas, B., Schneider, T., Zohner, M.: Faster private set intersection based on {OT} extension. In: 23rd USENIX Security Symposium (USENIX Security 2014), pp. 797–812 (2014)
29. Pinkas, B., Schneider, T., Zohner, M.: Scalable private set intersection based on OT extension. ACM Trans. Privacy Secur. (TOPS) **21**(2), 1–35 (2018)
30. Resende, A., Railsback, D., Dowsley, R., Nascimento, A.C., Aranha, D.F.: Fast privacy-preserving text classification based on secure multiparty computation. IEEE Trans. Inf. Forensics Secur. **17**, 428–442 (2022)
31. Resende, A.C.D., Aranha, D.F.: Faster unbalanced private set intersection. In: Meiklejohn, S., Sako, K. (eds.) FC 2018. LNCS, vol. 10957, pp. 203–221. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-662-58387-6_11
32. Rindal, P., Rosulek, M.: Malicious-secure private set intersection via dual execution. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 1229–1242 (2017)
33. Rosulek, M., Trieu, N.: Compact and malicious private set intersection for small sets. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, pp. 1166–1181 (2021)
34. Shokri, R., Shmatikov, V.: Privacy-preserving deep learning. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 1310–1321 (2015)
35. Shoup, V.: NTL: a library for doing number theory,version 5.3.2 (2003). http://www.shoup.net/ntl/
36. Trieu, N., Shehata, K., Saxena, P., Shokri, R., Song, D.: Epione: lightweight contact tracing with strong privacy. IEEE Data Eng. Bull. **43**, 95–107 (2020)
37. Ying, J.H., Cao, S., Poh, G.S., Xu, J., Lim, H.W.: Psi-stats: private set intersection protocols supporting secure statistical functions. In: Ateniese, G., Venturi, D. (eds.) ACNS 2022. LNCS, vol. 13269, pp. 585–604. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-09234-3_29