A Scalable Multiparty Private Set Intersection

Atsuko Miyaji[1,2,3](✉) and Shohei Nishida[1]

[1] Japan Advanced Institute of Science and Technology, Nomi, Japan
{miyaji,shohei-n}@jaist.ac.jp
[2] Japan Science and Technology Agency (JST) CREST, Tokyo, Japan
[3] Graduate School of Engineering, Osaka University, Suita, Japan

Abstract. Both scalability and flexibility become crucial for privacy preserving protocols in the age of Big Data. Private Set Intersection (PSI) is one of important privacy preserving protocols. Usually, PSI is executed by 2-parties, a client and a server, where both a client and a server compute jointly the intersection of their private sets and at the end only the client learns the intersection and the server learns nothing. From the scalable point of view, however, the number of parties are not limited to two. In this paper, we propose a scalable and flexible multiparty PSI (MPSI) for the first time: the data size of each party is independent to each other and the computational complexity is independent to the number of parties. We also propose d-and-over MPSI for the first time.

1 Introduction

Both scalability and flexibility become crucial for privacy preserving protocols in the age of Big Data. Private Set Intersection (PSI) is one of important privacy preserving protocols. PSI is executed by 2 parties, a client and a server, where both compute jointly the intersection of their private sets and, at the end, only the client learns the intersection and the server learns nothing. From the scalable point of view, however, the number of parties are not limited to two. This is why a multiparty PSI (MPSI) [8,14] becomes important. However, both are far from scalability: the computational complexity depends on the number of parties, and the data size of each party is equal to each other in [14] and [8] computes only the approximate number of intersection.

In this paper, we propose a scalable and flexible MPSI: the data size of each party is independent to each other and the computational complexity is independent to the number of parties. Furthermore we also propose a new notion of d-and-over multiparty PSI (d-and-over MPSI) for $d \leq n$. A d-and-over MPSI means to compute securely $\bigcap^{\geq d} S_j = \bigcup_{n \geq \ell \geq d} (S_{j_1} \cap \cdots \cap S_{j_\ell})$, where S_i is a set of $\mathbf{P}_\mathsf{i}$. Let us think the following scenario: There are n shops $\mathbf{P}_\mathsf{i}$ in a shopping mall whose customers' list is S_i. Shops think to promote number of customers each other and plan to have a promotion campaign. In the promotion campaign, a

This study is partly supported by Grant-in-Aid for Scientific Research (C) (15K00183) and (15K00189).

shop $\mathbf{P}_\mathsf{i}$ wants to know customers who joins an intersection of 3-and-over shops including $\mathbf{P}_\mathsf{i}$ without learning any information about customers that are not in the intersection. Such a scalable MPSI has, however, not proposed yet as far as authors know.

This paper is organized as follows. Section 2 summarises security assumption and building blocks used in our proposal. Section 3 explains the previous results. Then, after investigating set operations required in the case of n parties in Section 4, we propose concrete schemes of MPSI and d-and-over MPSI in Section 5. Comparison with the previous MPSI [14] is shown in Section 6.

2 Preliminary

This section summarises security assumption and building blocks used in our proposal.

2.1 Security Assumption

We describe two standard adversary models [10]: semi-honest adversaries and malicious adversaries. In semi-honest adversaries model, all players act according to their prescribed actions in the protocol. If a protocol is secure in a semi-honest model, then no player gains information about other player's private input sets, other than what can be deduced from the result of the protocol. On the other hand, in malicious adversaries model, an adversary player can behave arbitrarily. In particular, we cannot hope to prevent a malicious player from refusing to participate in the protocol, substituting an input with an arbitrary value, and aborting the protocol prematurely.

The security assumptions used in our protocol are defined as follows.

Definition 1 (DDH Assumption). *Let $\mathbb{F}_p$ be a finite field, $g \in \mathbb{F}_p$ with prime order q and size of q is ℓ. The DDH(Decisional Diffie-Hellman) problem is hard in $\mathbb{G}$ if, for any efficient algorithm $\mathcal{A}$, there exists $\epsilon > 0$ and the following probability is satisfied:* $|\mathsf{Pr}[x, y \leftarrow \{0,1\}^\ell : \mathcal{A}(g, g^x, g^y, g^{xy}) = 1] - \mathsf{Pr}[x, y, z \leftarrow \{0,1\}^\ell : \mathcal{A}(g, g^x, g^y, g^z) = 1]| < \epsilon$.

2.2 Bloom Filter

A Bloom filter [2], denoted by BF, is a space-efficient probabilistic data structure, that is used to test whether an element x is included in a set S. False positive matches are possible, but false negatives are not, thus a Bloom filter has a 100% recall rate. Elements can be added to the set, but not removed. A Bloom filter is an array of m bits that can represent a set S with at most w elements. A Bloom filter uses a set of k independent uniform hash functions $\mathcal{H} = \{H_0, ..., H_{k-1}\}$, where $H_i : \{0,1\}^* \longrightarrow \{0, 1, \cdots, m-1\} (0 \leq \forall i \leq k-1)$. Here after, we denote a Bloom filter parametrised (m, k) by $\mathsf{BF}_{m,k}(S)$ that encodes a set S. Let us explain how BF is constructed, which is given by $\mathsf{const.BF}$ (see Algorithm 1):

output $\mathsf{BF}_{m,k}(S)$ for input of a set S. Initially, all bits in the array are set to 0. To insert an element $x \in S$ into the filter, the element is hashed using k hash functions to get k index numbers. Bits at these indexes are set to 1, i.e. set $\mathsf{BF}_{m,k}[H_i(x)] = 1$ for $0 \leq i \leq k-1$. To check if an item y is in S, we execute check.BF (see Algorithm 2): y is hashed by k hash functions, and all locations where y is hashed are checked. If any bit at the locations is 0, y is not in S, otherwise y is probably in S. However, a false positive is possible, i.e. it is possible that y is not in the set S, but all $\mathsf{BF}[H_i(y)]$ are set to 1. The false positive probability p is [3]: $p = \left\{1 - \left(1 - \frac{1}{m}\right)^{kw}\right\}^k \approx \left\{1 - e^{-kw/m}\right\}^k$. For a given m and w, the value of k that minimizes the false positive probability is: $k = \frac{m}{w}\ln 2$. When $e^{-kw/m} = 1/2$, the false positive probability $p = (1/2)^k \approx (0.6185)^{m/w}$. The number z of 0 bits in a Bloom filter for a set S is strongly concentrated around its expectation $m(1-1/m)^{k|S|}$ [3]. Therefore, given z, m and k, the size of S is given approximately to $|S| = \frac{\ln(z/m)}{k\ln(1-1/m)}$.

Algorithm 1. const.BF(S)

```
Require: A set $S$
Ensure: A Bloom filter $\mathsf{BF}_{m,k}(S)$
 1: for $i = 0$ to $m-1$ do
 2:     $\mathsf{BF}_{m,k}[i] \leftarrow 0$
 3: end for
 4: for all $x \in S$ do
 5:     for $i = 0$ to $k-1$ do
 6:         $j = H_i(x)$
 7:         if $\mathsf{BF}_{m,k}[j] = 0$ then
 8:             $\mathsf{BF}_{m,k}[j] \leftarrow 1$
 9:         end if
10:     end for
11: end for
```

Algorithm 2. check.BF(BF, S_q)

```
Require: A Bloom filter $\mathsf{BF}_{m,k}(S)$, a set $S_q$
Ensure: A set $S_\cap (= S \cap S_q)$
1: generates the empty set $S_\cap = \{\}$
2: for all $x \in S_q$ do
3:     for $i = 0$ to $k-1$ do
4:         $j = H_i(x)$
5:     end for
6:     if all $\mathsf{BF}_{m,k}[j] = 1$ then
7:         add $x$ to the set $S_\cap$
8:     end if
9: end for
```

2.3 Additive Homomorphic Encryption

An additive homomorphic encryption is important tool to deal with encrypted data. One of typical additive homomorphic encryption is Paillier encryption[16]. In our scheme, an additive homomorphic encryption is used for matching, and, thus exponential ElGamal encryption [4] is enough and more efficient than Paillier encryption. In fact, results of decryption in ex-ElGamal can distinguish whether two message m_1 and m_2 are equal although it can not decrypt a message itself. Furthermore, ex-ElGamal can be extended to decrypt a ciphertext distributedly, where n parties $\mathbf{P}_i(1 \leq i \leq n)$ jointly decrypt, which consists of three functions:

Key Generation:
Let $\mathbb{F}_p$ be a finite field, $g \in \mathbb{F}_p$ with prime order q. Each party chooses $x_i \in \mathbb{Z}_q$

randomly and computes $y_i = g^{x_i} \pmod p$, then $y = \prod_{i=1}^{n} y_i \pmod p$ is a public key and each x_i is a share for a party to decrypt a ciphertext.

Encryption: $\mathsf{Enc}[m] \to (u, v)$
For a message $m \in \mathbb{Z}_q$ with a public key y, choose $r \in \mathbb{Z}_q$ randomly, compute both $u = g^r \pmod p$ and $v = g^m y^r \pmod p$, then output (u, v) as a ciphertext of m.

Decryption: $\mathsf{dis.Dec}[(u, v)] \to g^m$
Each party computes $z_i = u^{x_i} \pmod p$ and $z = \prod_{i=1}^{n} z_i \pmod p$ jointly and decrypt the ciphertext as $g^m = v/z \pmod p$.

Both ex-ElGamal encryption and the above distributed version have the following features: (1)an additive homomorphism for messages $m_1, m_2 \in \mathbb{Z}_p$: $\mathsf{Enc}(m_1)\mathsf{Enc}(m_2) = \mathsf{Enc}(m_1 + m_2)$. (2) a scalar homomorphism for message m and $k \in \mathbb{Z}_q$: $\mathsf{Enc}(m)^k = \mathsf{Enc}(km)$.

3 Previous Works

This section overviews prior works on PSI between a server and a client and MPSI among n parties. In PSI, let server and client data sets be $S = \{s_1, ..., s_v\}$ and $C = \{c_1, ..., c_w\}$, where $|S| = v$ and $|C| = w$. In MPSI, we assume that the number of each party's set is equal to each other for simplicity.

PSI Protocol Based on Polynomial Evaluation: Main idea is to represent elements in C as roots of a polynomial, and send its encrypted polynomial to a server; evaluate it on elements in S, which introduced by Freedman [9] for the first time. This is secure against semi-honest adversaries under a public key encryption. The computational complexity is $O(vw)$ exponentiations, and communicational complexity is $O(v + w)$. The computational complexity is reduced to $O(v \log\log w)$ exponentiations by using balanced allocations technique [1]. Kissner and Song extended protocols to MPSI [14]. The computational complexity is $O(nw^2)$ exponentiations and communicational complexity is $O(nw)$ and it is secure against semi-honest and malicious adversaries (in the random oracle model) using generic zero-knowledge proofs.

PSI Protocol Based on DDH: Main idea is to apply DDH assumption [6]: after presenting each data by hash value $\{h(s_i)\}$ and $\{h(c_i)\}$, the client sends a set of $\{h(c_i)^{r_i}\}$ encrypted by a random number r_i; the server sends back $\{h(c_i)^{rr_i}\}$ and $\{h(s_i)^r\}$ for a random number r, finally the client evaluate $S \cap C$ by decrypting to $\{h(c_i)^r\}$. This is secure against semi-honest adversaries under DDH assumption. The total computational complexity is $O(v + w)$ exponentiations and the total communicational complexity is $O(v + w)$. The security is enhanced to against malicious adversaries in the random oracle model in [5] by using blind signature. Any extension to MPSI based on DDH, however, has not been proposed.

PSI Protocol Based on Bloom Filter: PSI based on Bloom filter is proposed in [15] for the first time by just executing AND of Bloom filters of server and client. This protocol, however, is not secure because Bloom filter itself leaks information about other party's set. In [13], the security is enhanced by combining Bloom filters with the Goldwasser Micali encryption [11]. In a semi-honest version, the computational complexity is both $O(kw)$ hash operations and $O(m)$ public key operations and the communicational complexity is $O(m)$, where (m, k) is a parameter of Bloom filter. Another protocol combined Bloom filter, Oblivious transfer extension [12,17], and garbled Bloom filter constructed newly is proposed [7]. The main difference between Bloom filter and garbled Bloom filter is that a Bloom filter is 1-bit array while a garbled Bloom filter is a λ-bit array. To add an element $x \in S$ to a garbled Bloom filter, x is split into k shares with λ bits using the XOR-based secret sharing ($x = x_1 \bigoplus ... \bigoplus x_k$) and mapped x_i into an index of $H_i(x)$. To query an element y, all bit strings at $H_i(y)$ is XOR them together. If the result is y, then y is in S, otherwise y is not in S. The client uses Bloom filter $\mathsf{BF}(C)$ and the server uses garbled Bloom filter $\mathsf{GBF}(S)$. Then, if an element x is in $C \cap S$, then for every position i it hashes to, $\mathsf{BF}(C)[i]$ must be 1 and $\mathsf{GBF}(S)[i]$ must be x_i. Thus, the client evaluates $C \cap S$. The computational complexity is $O(kw)$ hash operations and $O(m)$ public key operations and communicational complexity is $O(m)$, where the number of public key operations can be changed to $O(\lambda)$ by using Oblivious transfer extension. This is secure against semi-honest adversaries under secure Oblivious transfer protocol. Another research computes the approximate number of multiparty set union in [8]. However, MPSI based on Bloom filter has been proposed.

4 Multiparty Set Intersection

We investigate what set operations are required in the case of n parties. Let us investigate the following scenarios: There are n medical institutions $\mathbf{P}_{\mathsf{i}}$ whose patient list is S_i. Patients often use several medical institutions. Each medical institution $\mathbf{P}_{\mathsf{i}}$ wants to find common patients without learning any information about patients that are not in S_i. That is, $\mathbf{P}_1$ wants to know patients who uses 2-and-over medical institutions including $\mathbf{P}_{\mathsf{i}}$ without learning any information about patients that are not in $\mathbf{P}_{\mathsf{i}}$, which is denoted by $\cap^{\geq 2} S_j[1]$.

Let us formalize intersections of n parties. As we have seen the above scenario, intersections of all parities and d-and-over parties for $\forall d (\leq n)$ are necessary, which are called MPSI and *d-and-over* MPSI, respectively. Here, d-and-over MPSI is denoted by $\bigcap^{\geq d} S_j = \bigcup_{n \geq \ell \geq d} (S_{j_1} \cap \cdots \cap S_{j_\ell})$. For example, given 4 party-set $S_1 = \{g, h, i, j, f, n, o, k\}$, $S_2 = \{a, h, n, m, b, i, o, \ell\}$, $S_3 = \{f, n, o, k, e, m, l, d\}$, and $S_4 = \{b, i, o, l, c, j, k, d\}$. Then MPSI is $\{o\}$; and 3-and-over MPSI is given by $\bigcap^3 S_i = \{i, k, \ell, n, o\}$. Then, intersection of 3-and-over MPSI given to each $\mathbf{P}_{\mathsf{i}}$ is $\cap^{\geq 3} S_j[1] = \{i, k, n, o\}$, $\cap^{\geq 3} S_j[2] = \{i, \ell, n, o\}$, $\cap^{\geq 3} S_j[3] = \{k, \ell, n, o\}$, and $\cap^{\geq 3} S_j[4] = \{i, k, \ell, o\}$.

Let us discuss how to achieve MPSI and d-and-over MPSI. If we apply PSI to achieve MPSI, the computation and communication complexity seems to depend to the number of parties, which exactly happens to [14]. On the other hand, if we apply MPSI to achieve a d-and-over MPSI, we would need to execute MPSI in

${}_n\mathrm{C}_d$ times, which is rather wastefulness. This is why it is necessary to construct MPSI and d-and over MPSI directly. On the other hand, privacy issues on MPSI and d-and-over MPSI are informally give as follows.
MPSI Privacy: An MPSI scheme is party-private if any party $\mathbf{P}_\mathsf{i}$ learns no information about elements of other parties' set except elements in $\cap S_j$.
d-and-over MPSI Privacy: A d-and-over MPSI scheme is party-private if any party $\mathbf{P}_\mathsf{i}$ learns no information about elements of other parties' set except elements in $\cap^{\geq d} S_j[i]$.

5 Scalable Multiparty PSI

Our schemes of MPSI and d-and-over MPSI will be presented after describing protocol intuition briefly.

5.1 Protocol Intuition

The following notations are used in our two protocols.

- $\mathbf{P}_\mathsf{i}$: i-th party, where the number of parties is n
- $\mathbf{D}$: dealer who does not know anything about inputs or outputs
- $S_i = \{s_{i,1}, s_{i,2}, \cdots, s_{i,w_i}\}$: a set of $\mathbf{P}_\mathsf{i}$, where $|S_i| = \omega_i$
- $\cap S_j$ or $\cap^{\geq d} S_j$: intersection of all or d-and-over parties out of n
- $\cap^{\geq d} S[i]$: intersection of d-and-over parties possessed by $\mathbf{P}_\mathsf{i}$, $\cap^{\geq d} S \subset S_i$
- $\mathsf{Enc}/\mathsf{dis.Dec}$: distributed ex-ElGamal encryption/decryption by all $\mathbf{P}_\mathsf{i}$
- m: size of Bloom filter
- $\mathcal{H} = \{H_0, ..., H_{k-1}\}$: set of hashes used in Bloom filter, where k is $\#\mathcal{H}$.
- $\boldsymbol{\ell} = [\ell, \cdots, \ell]$ $(1 \leq \ell \leq n)$: an m-dimension array, where all strings in the array are set to ℓ
- $\mathsf{BF}_{m,k}(S_i) = [\mathsf{BF}_i[0], \cdots, \mathsf{BF}_i[m-1]]$: Bloom filter on a set S_i
- $\mathsf{IBF}_{m,k}(\cup S_i) = [\sum_{i=1}^{n} \mathsf{BF}_i[0], \cdots, \sum_{i=1}^{n} \mathsf{BF}_i[m-1]]$: integrated Bloom filter of n sets $\{S_i\}$, where $\sum_{i=1}^{n} \mathsf{BF}_i[j]$ presents the sum of all parties' array.
- $\mathsf{IBF}_{m,k}(\cup S_i) \setminus \boldsymbol{\ell} = [\sum_{i=1}^{\ell} \mathsf{BF}_i[0] - \ell, \cdots, \sum_{i=1}^{n} \mathsf{BF}_i[m-1] - \ell](1 \leq \ell \leq n)$: $\boldsymbol{\ell}$-subtraction from $\mathsf{IBF}_{m,k}(\cup S_i)$.

Our scheme is flexible for the data size of party, and, thus, the data size of each party is independent to each other. We introduce a dealer $\mathbf{D}$ to reduce the computational complexity of parties. $\mathbf{D}$ can be outsourced since it does not know anything about S_i or $|S_i|$. A distributed ex-ElGamal encryption among n-party is used to achieve all computation without knowing S_i themselves and at the end decryption is jointly done. In both protocols, each $\mathbf{P}_\mathsf{i}$ constructs $\mathsf{BF}_{m,k}(S_i)$ for a set S_i and encrypts each array by Enc. All encrypted Bloom filters are securely added by a dealer $\mathbf{D}$ without decrypting. These procedures are executed in both MPSI and d-and-over MPSI. In MPSI, $\mathbf{D}$ encrypts a randomized *n-subtraction* of $\mathsf{IBF}_{m,k}(\cup S_i)$, $\boldsymbol{r}(\mathsf{IBF}_{m,k}(\cup S_i) \setminus \boldsymbol{n})$. If $x \in \cap S_i$, the corresponding array locations in an encrypted array where x is mapped by k hashes is an encryption of 0; an encryption of randomized value otherwise. In d-and-over MPSI, $\mathbf{D}$ computes a randomized encryption of *ℓ-subtraction* of $\boldsymbol{r}(\mathsf{IBF}_{m,k}(\cup S_i) \setminus \ell)$ for $d \leq \ell \leq n$. If $x \in \cap^\ell S$ for $d \leq \exists \ell \leq n$, the corresponding array locations in $\mathsf{IBF}_{m,k}(\cup S_i) \setminus \boldsymbol{j}$ for $\ell \leq \exists j \leq n$ where x is mapped by k hashes is an encryption of 0; an encryption of randomized value otherwise. An difference from MPSI is that the corresponding array locations in $\mathsf{IBF}_{m,k}(\cup S_i) \setminus \ell$ is not a necessary encryption of 0 even if $x \in \cap^\ell S$.

5.2 MPSI and d-and-over MPSI

First, we present MPSI, MPSI consists of 4 phases: initialization, $\mathbf{P}_\mathsf{i}$'s Bloom filter construction, $\mathbf{D}$'s Encryption of $\boldsymbol{n}$-subtraction of $\mathsf{IBF}_{m,k}(\cup S_i)$, and, finally, $\mathbf{P}_\mathsf{i}$'s MPSI computation. As system parameters, a finite field $\mathbb{F}_p$ and a base-point $g \in \mathbb{F}_p$ with order q for a distributed ex-ElGamal encryption (Enc, $\mathsf{dis.Dec}$), given to $\mathbf{P}_\mathsf{i}$ and $\mathbf{D}$, but both $\mathsf{const.BF}(S)$ and $\mathsf{check.BF}(\mathsf{BF}, S_q)$ are given to only $\mathbf{P}_\mathsf{i}$. When we encrypt or randomize a vector such as a Bloom filter $\mathsf{BF}_{m,k} = [a_0, \cdots, a_{m-1}]$, each location is encrypted or randomized independently: $\mathsf{Enc}(\mathsf{BF}_{m,k}) = [\mathsf{Enc}(a_0), \cdots, \mathsf{Enc}(a_{m-1})]$ or $\boldsymbol{r}\mathsf{BF}_{m,k} = [r_0 a_0, \cdots, r_{m-1} a_{m-1}]$ by $\boldsymbol{r} = [r_0, \cdots, r_{m-1}] \in \mathbb{Z}_q^m$, respectively.

Initialization: $\mathbf{P}_\mathsf{i}$ executes the following:

1. Generate a secret key $x_i \in \mathbb{Z}_q$ and compute a public key $y_i = g^{x_i} \in \mathbb{Z}_q$ and broadcast y_i to other parties.
2. Compute an n-party public key $y = \prod_i y_i$ whose secret key is $x = \sum x_i$.

$\mathsf{BF}_{m,k}(S_i)$ **construction:** $\mathbf{P}_\mathsf{i}$ executes the following:

1. Do $\mathsf{const.BF}_{m,k}(S_i) \longrightarrow \mathsf{BF}_{m,k}(S_i) = [\mathsf{BF}_i[0], \cdots, \mathsf{BF}_i[m-1]]$ (Algorithm 1).
2. Encrypt each array of $\mathsf{BF}_{m,k}(S_i)$ by using Enc_y with a public key y: $\mathsf{Enc}_y(\mathsf{BF}_{m,k}(S_i)) = [\mathsf{Enc}_y(\mathsf{BF}_i[0]), \cdots, \mathsf{Enc}_y(\mathsf{BF}_i[m-1])]$.
3. Send $\mathsf{Enc}_y(\mathsf{BF}_{m,k}(S_i))$ to $\mathbf{D}$.

Encryption of $\boldsymbol{n}$-subtraction of $\mathsf{IBF}_{m,k}(\cup S_i)$: $\mathbf{D}$ executes the following:

1. Encrypt $\mathsf{IBF}_{m,k}(\cup S_i)$ by Enc_y without knowing $\mathsf{IBF}_{m,k}(\cup S_i)$ as follows: $\mathsf{Enc}_y(\mathsf{IBF}_{m,k}(\cup S_i)) = \prod_{i=1}^n \mathsf{Enc}_y(\mathsf{BF}_{m,k}(S_i))$.
2. Encrypt $\mathsf{IBF}_{m,k}(\cup S_i) \setminus \boldsymbol{n}$ randomized by $\boldsymbol{r} = [r_0, \cdots, r_{m-1}] \in \mathbb{Z}_q^m$: $\mathsf{Enc}_y(\boldsymbol{r}(\mathsf{IBF}_{m,k}(\cup S_i) \setminus \boldsymbol{n})) = (\mathsf{Enc}_y(\mathsf{IBF}_{m,k}(\cup S_i)) \cdot \mathsf{Enc}_y(-\boldsymbol{n}))^{\boldsymbol{r}}$, where $\mathsf{Enc}_y(-\boldsymbol{n}) = [\mathsf{Enc}_y(-n), \cdots, \mathsf{Enc}_y(-n)]$.
3. Broadcast $\mathsf{Enc}_y(\boldsymbol{r}(\mathsf{IBF}_{m,k}(\cup S_i) \setminus \boldsymbol{n}))$ to $\mathbf{P}_\mathsf{i}$.

MPSI computation: $\mathbf{P}_\mathsf{i}$ executes the following:

1. All $\mathbf{P}_\mathsf{i}$ jointly decrypt $\mathsf{Enc}_y(\boldsymbol{r}(\mathsf{IBF}_{m,k}(\cup S_i) \setminus \boldsymbol{n}))$.
2. Execute $\mathsf{check.BF}_{m,k}(\boldsymbol{r}(\mathsf{IBF}_{m,k}(\cup S_i) \setminus \boldsymbol{n}), S_i) \longrightarrow \cap S_i$ and output $\cap S_i$.

Correctness of MPSI follows from the fact that if an element x is included in $\cap S_i$, the corresponding array locations in $\mathsf{Enc}_y(\boldsymbol{r}(\mathsf{IBF}_{m,k}(\cup S_i) \setminus \boldsymbol{n}))$ where x is mapped by k hashes is an encryption of 0, which are decrypted to 1;an encryption of randomized value otherwise.

Next, we present d-and-over MPSI. Procedures of d-and-over MPSI is the same as that of MPSI until $\mathbf{D}$ computes $\mathsf{Enc}_y(\mathsf{IBF}_{m,k}(\cup S_i))$. So, we describe after $\mathbf{D}$ computes $\mathsf{Enc}_y(\mathsf{IBF}_{m,k}(\cup S_i))$.

Encryption of ℓ-subtraction of $\mathsf{IBF}_{m,k}(\cup S_i)$: D executes the following:

1. Encrypt $\mathsf{IBF}_{m,k}(\cup S_i)\setminus\boldsymbol{\ell}$ randomized by $\boldsymbol{r} = [r_0, \cdots, r_{m-1}] \in \mathbb{Z}_q^m (d \le \ell \le n)$: $\mathsf{Enc}_y(\boldsymbol{r}(\mathsf{IBF}_{m,k}(\cup S_i) \setminus \boldsymbol{\ell})) = (\mathsf{Enc}_y(\mathsf{IBF}_{m,k}(\cup S_i)) \cdot \mathsf{Enc}_y(-\boldsymbol{\ell}))^{\boldsymbol{r}}$.
2. Broadcast $\{\mathsf{Enc}_y(\boldsymbol{r}(\mathsf{IBF}_{m,k}(\cup S_i) \setminus \boldsymbol{\ell}))\}_\ell$ $(d \le \ell \le n)$ to $\mathbf{P}_\mathsf{i}$.

d-and-over MPSI computation: $\mathbf{P}_\mathsf{i}$ executes the following:

1. All $\mathbf{P}_\mathsf{i}$ jointly decrypt $\{\mathsf{Enc}_y(\boldsymbol{r}(\mathsf{IBF}_{m,k}(\cup S_i) \setminus \boldsymbol{\ell}))\}_\ell$.
2. Let CBF_ℓ be an m-array for $d \le \ell \le n$, where an array is set to 1 if and only if the corresponding array of $\boldsymbol{r}\mathsf{IBF}_{m,k}(\cup S_i) \setminus \boldsymbol{\ell}$ is 1, and others are set to 0.
3. Set $\mathsf{CBF} = \mathsf{CBF}_\ell \vee \cdots \vee \mathsf{CBF}_n$.
4. Execute $\mathsf{check.BF}_{m,k}(\mathsf{CBF}, S_i) \longrightarrow \cap^{\ge d} S[i]$ and output $\cap^{\ge d} S[i]$.

Correctness of d-and-over MPSI follows from the fact that if an element $x \in \cap^\ell S$ for $d \le \exists \ell \le n$, the corresponding array locations in $\mathsf{IBF}_{m,k}(\cup S_i)\setminus\boldsymbol{j}$ for $\ell \le \exists j \le n$ where x is mapped by k hashes is an encryption of 0, which are decrypted to 1; an encryption of randomized value otherwise.

The security of both protocols is given as follows, whose proof will be presented in the final paper.

Theorem 1. *If the Decisional Diffie-Hellman assumption holds, then both MPSI and d-and-over are secure against semi-honest adversary.*

6 Comparison

Table 1 compares the computational and communicational complexity of our protocol with [14]. Each protocol is secure against semi-honest adversaries without the trusted third party under each security assumption of employed public key encryption: [14] uses Paillier encryption (Decisional Composite Residue (DCR)) and our protocols use ex-ElGamal encryption (DDH). Bloom filter parameters (m, k) used in our protocol are set as follows: $k = 80$ and $m = 80\omega/\ln 2$, where ω is the maximum $|S_i| = \omega_i$. Then, the false positives probability is given by $p = 2^{-80}$. $\mathbf{P}_\mathsf{i}$'s dominant computational complexity is Bloom filter construction and MPSI or d-and-over MPSI computation, which is $O(\omega_i)$ and doesn't depend on the number of parties unlike [14]. $\mathbf{D}$'s dominant computational complexity is $\boldsymbol{n}$-subtraction of $\mathsf{IBF}_{m,k}(\cup S_i)$, which is $O(n\omega)$ in both MPSI and d-and over MPSI. Our scheme is flexible for the data size of party, and, thus, the data size of each party is independent to each other. An approach to compute approximate number of MPSI is proposed in [8] by using features of Bloom Filter. Our protocol can be converted to compute easily approximate number of $|\cap S_j|$ or $|\cap^{\ge d} S[i]|$.

Table 1. Comparison of MPSI

Protocol	[14]	Our protocol
Computational complexity	$O(n\omega^2)$	$\mathbf{P}_\mathsf{i} : O(\omega_i)$, $\mathbf{D} : O(n\omega)$
Communicational complexity	$O(n\omega)$	$O(n\omega)$
Number of input data	$\|S_1\| = ... = \|S_n\|$	any
Privacy	$S_1, ..., S_n$	$S_1, ..., S_n, \|S_1\|, ..., \|S_n\|$

7 Conclusion

In this paper, we have proposed a scalable and flexible multiparty PSI (MPSI). We have also proposed a new notion of d-and-over MPSI and presented a concrete protocol. Our schemes are flexible: data size of each party is independent to each other. We also introduce a dealer **D** to reduce the computational complexity of parties, who acts as opposed to the trusted third party and does not know anything about inputs or outputs (including its size), and, it thus can be outsourced. Thanks to **D**, $\mathbf{P}_\mathsf{i}$'s, computational complexity is $O(\omega_i)$, which doesn't depend on the number of parties unlike [14].

References

1. Azar, Y., et al.: Balanced allocations. SIAM journal on computing **29**(1), 180–200 (1999)
2. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM **13**(7), 422–426 (1970)
3. Broder, A., Mitzenmacher, M.: Network applications of bloom filters: A survey. Internet mathematics **1**(4), 485–509 (2004)
4. Cramer, R., et al.: A secure and optimally efficient multi-authority election scheme. European transactions on Telecommunications **8**(5), 481–490 (1997)
5. De Cristofaro, E., Kim, J., Tsudik, G.: Linear-complexity private set intersection protocols secure in malicious model. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 213–231. Springer, Heidelberg (2010)
6. De Cristofaro, E., Tsudik, G.: Practical private set intersection protocols with linear complexity. In: Sion, R. (ed.) FC 2010. LNCS, vol. 6052, pp. 143–159. Springer, Heidelberg (2010)
7. Dong, C., et al.: When private set intersection meets big data: an efficient and scalable protocol. In: ACMCCS 2013, pp. 789–800. ACM (2013)
8. Egert, R., Fischlin, M., Gens, D., Jacob, S., Senker, M., Tillmanns, J.: Privately computing set-union and set-intersection cardinality via bloom filters. In: Foo, E., Stebila, D. (eds.) ACISP 2015. LNCS, vol. 9144, pp. 413–430. Springer, Heidelberg (2015)
9. Freedman, M.J., Nissim, K., Pinkas, B.: Efficient private matching and set intersection. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 1–19. Springer, Heidelberg (2004)
10. Goldreich, O.: Secure multi-party computation. Manuscript, Preliminary version (1998)

11. Goldwasser, S., Micali, S.: Probabilistic encryption. Journal of computer and system sciences **28**(2), 270–299 (1984)
12. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (2003)
13. Kerschbaum, F.: Outsourced private set intersection using homomorphic encryption. In: ACMCCS 2012, pp. 85–86. ACM (2012)
14. Kissner, L., Song, D.: Privacy-preserving set operations. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 241–257. Springer, Heidelberg (2005)
15. Many, D., et al.: Fast private set operations with sepia. Technical Report 345 (2012)
16. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, p. 223. Springer, Heidelberg (1999)
17. Rabin, M.O.: How to exchange secrets with oblivious transfer. Tech. Memo, TR-81 (1981)