J Med Syst (2017) 41: 37
DOI 10.1007/s10916-016-0657-4

Privacy-Preserving Integration of Medical Data

A Practical Multiparty Private Set Intersection

Atsuko Miyaji[1] · Kazuhisa Nakasho[2] · Shohei Nishida[3]

Received: 30 June 2016 / Accepted: 1 November 2016 / Published online: 16 January 2017

Abstract Medical data are often maintained by different organizations. However, detailed analyses sometimes require these datasets to be integrated without violating patient or commercial privacy. Multiparty Private Set Intersection (MPSI), which is an important privacy-preserving protocol, computes an intersection of multiple private datasets. This approach ensures that only designated parties can identify the intersection. In this paper, we propose a practical MPSI that satisfies the following requirements: The size of the datasets maintained by the different parties is independent of the others, and the computational complexity of the dataset held by each party is independent of the number of parties. Our MPSI is based on the use of an outsourcing provider, who has no knowledge of the data inputs or outputs. This reduces the computational complexity. The performance of the proposed MPSI is evaluated by implementing a prototype on a virtual private network to enable parallel computation in multiple threads. Our protocol is confirmed to be more efficient than comparable existing approaches.

Keywords Medical data · Privacy-preserving data integration · Private set intersection

This article is part of the Topical Collection on *Transactional Processing Systems*

✉ Atsuko Miyaji
miyaji@comm.eng.osaka-u.ac.jp

[1] Graduate School of Engineering, Osaka University, 2-1 Yamadaoka Suita, Osaka, Japan

[2] Department of Machine Intelligence and Systems Engineering, Akita Prefectural University, 84-4 Ebinokuchi, Tsuchiya, Yurihonjo, Akita, Japan

[3] Japan Advanced Institute of Science and Technology, Asahidai 1-1, Nomi-shi, Ishikawa, Japan

Introduction

Medical organizations often store the data accumulated through medical analyses. However, detailed data analysis sometimes requires separate datasets to be integrated without violating patient or commercial privacy. Consider the scenario in which the occurrence of similar accidents can be attributed to a particular defective product. Such defective products should be identified as quickly as possible. However, the databases related to accidents are maintained separately by different organizations. Thus, investigating the causes of accidents is often time-consuming. For example, suppose child A has broken her/his leg at school, but it is not clear whether the accident was caused by defective equipment. In this case, information relating to A's injury, such as the patient's name and type of injury, are stored in hospital database S_1. Information pertaining to A's accident, such as their name and the location of the swing at the school, are stored in database S_2, which is held by the fire department. Finally, information relating to the insurance claim following A's accident, such as the name and medical costs, is maintained in the insurance company's database, S_3. Computing the intersection of these databases, $S_1 \cap S_2 \cap S_3$, without compromising privacy would enable us to combine the separate sets of information, which may allow the cause of the accident to be identified. Let us consider another situation. Several clinics, denoted as P_i, maintain separate databases, represented as S_i. The clinics

wish to know the patients they have in common to enable them to share treatment details; however, P_i should not be able to access any information about patients not stored in their own dataset. In this case, the intersection of the set must not reveal private information.

These examples illustrate the need for the Multiparty Private Set Intersection (MPSI) protocol [11, 17, 18, 21]. MPSI is executed by multiple parties who jointly compute the intersection of their private datasets. Ultimately, only designated parties can access the intersection. Previous protocols are impractical, because the bulk of the computation is a function of the number of players. One previous study required the size of the datasets maintained by the different players to be equal [17, 21]. Another study [11] computed only the approximate number of intersections, whereas other researchers [18] required more than two trusted third-parties.

In this paper, we propose a practical MPSI with the following features:

1. The size of the datasets maintained by each party is independent of those maintained by the other parties.
2. The computational complexity for each party is independent of the number of parties. This is accomplished by introducing an outsourcing provider, $\mathcal{O}$. In fact, all computations related to the number of parties are carried out by $\mathcal{O}$. Thus, the number of parties is irrelevant.

The remainder of this paper is organized as follows. Previous results that are used to develop the proposed protocol are summarized in "Preliminaries". "Previous work" then introduces some related studies. We propose the new MPSI in "Practical MPSI", and present the results of its implementation in "Implementation results".

Preliminaries

In this section, we summarize the DDH assumption, Bloom filter, and ElGamal encryption. We consider security according to the honest-but-curious model [13]: all players act according to their prescribed actions in the protocol. A protocol that is secure in an honest-but-curious model does not allow any player to gain information about other players' private input sets, besides that which can be deduced from the result of the protocol. Note that the term *adversary* here refers to insiders, i.e., protocol participants. Outsider adversaries are not considered. In fact, behavior by outsider adversaries can be mitigated via standard network security techniques.

Our protocol is based on the following security assumption.

Definition 1 (DDH Assumption) Let t be a security parameter. A decisional Diffie–Hellman (DDH) parameter generator $\mathcal{IG}$ is a probabilistic polynomial time (PPT) algorithm that takes input 1^k and outputs a description of a finite field $\mathbb{F}_p$ and a basepoint $g \in \mathbb{F}_p$ with prime order q. We say that $\mathcal{IG}$ satisfies the *DDH assumption* if $|p_1 - p_2|$ is negligible (in K) for all PPT algorithms A, where $p_1 = \Pr[(\mathbb{F}_p, g) \leftarrow \mathcal{IG}(1^K); y_1 = g^{x_1}, y_2 = g^{x_2} \leftarrow \mathbb{F}_p : A(\mathbb{F}_p, g, y_1, y_2, g^{x_1 x_2}) = 0]$ and $p_2 = \Pr[(\mathbb{F}_p, g) \leftarrow \mathcal{IG}(1^K); y_1 = g^{x_1}, y_2 = g^{x_2}, z \leftarrow \mathbb{F}_p : A(\mathbb{F}_p, g, y_1, y_2, z) = 0]$.

A Bloom filter [3], denoted by BF, consists of m arrays and has a space-efficient probabilistic data structure. The BF can check whether an element x is included in a set S by encoding S with at most w elements. The encoded Bloom filter of S is denoted by $\mathsf{BF}(S)$.

The BF uses a set of k independent uniform hash functions $\mathcal{H} = \{H_0, ..., H_{k-1}\}$, where $H_i : \{0,1\}^* \longrightarrow \{0, 1, \cdots, m-1\}$ for $0 \leq \forall i \leq k-1$. The BF consists of two functions: Const embeds a given set S into $\mathsf{BF}(S)$, and ElementCheck checks whether an element x is included in S. SetCheck, an extension of ElementCheck, checks whether an element x in S' is in $S' \cap S$ (see Algorithm 3). In Const (see Algorithm 1), $\mathsf{BF}(S)$ is constructed for a given set S by first setting all bits in the array to 0. To embed an element $x \in S$ into the filter, the element is hashed using k hash functions to obtain k index numbers, and the bits at these indexes are set to 1, i.e., set $\mathsf{BF}[H_i(x)] = 1$ for $0 \leq i \leq k-1$. In ElementCheck (see Algorithm 2), we check all locations where x is hashed; x is considered to be not in S if any bit at these locations is 0; otherwise, x is probably in S.

Some false positive matches may occur, i.e., it is possible that all $\mathsf{BF}[H_i(y)]$ are set to 1, but y is not in S. The false positive rate FPR is given by $\mathsf{FPR} = \left\{1 - \left(1 - \frac{1}{m}\right)^{kw}\right\}^k \approx \left\{1 - e^{-kw/m}\right\}^k$ [4]. However, false negatives are not possible, and so Bloom filters have a 100 % recall rate.

Algorithm 1 Const(S)

Input: A set S
Output: A Bloom filter $\mathsf{BF}(S)$
1. **for** $i = 0$ to $k-1$ **do**
2. $\quad \mathsf{BF}(S)[j] \leftarrow 1$
3. **end for**
4. **for all** $x \in S$ **do**
5. $\quad$ **for** $i = 0$ to $k-1$
6. $\quad\quad j = H_i(x)$
7. $\quad\quad$ **if** $\mathsf{BF}(S)[j] = 0$
8. $\quad\quad\quad \mathsf{BF}(S)[j] \leftarrow 1$
9. $\quad\quad$ **end if**
10. $\quad$ **end for**
11. **end for**
12. output $\mathsf{BF}(S)$. stop.

Algorithm 2 ElementCheck(BF, x)

Input: A Bloom filter BF(S), an element x
Output: 1 if $x \in S$ and 0 if $x \notin S$
1. **for** $i = 0$ to $k - 1$ **do**
2. $j = H_i(x)$
3. **if** BF(S)[j] $= 0$
4. output 0. stop.
5. **end if**
6. **end for**
7. output 1. stop.

Algorithm 3 SetCheck(BF, S')

Input: A Bloom filter BF(S), a set S'
Output: A set $S_\cap (= S \cap S')$
1. $S_\cap \leftarrow \{\}$
2. **for** all $x \in S'$ **do**
3. **for** $i = 0$ to $k - 1$
4. $j = H_i(x)$
5. **if** BF[j] $= 0$ **then**
6. go to next x
7. **end if**
8. **end for**
9. add x to the set $S_\cap$
10. **end for**
11. output $S_\cap$. stop.

Homomorphic encryption under addition is useful for processing encrypted data. A typical homomorphic encryption under addition was proposed by Paillier [19]. However, because Paillier encryption cannot reduce the order of a composite group, it is computationally expensive compared with the following ElGamal encryption. Our protocol requires matching without revealing the original messages, for which exponential ElGamal encryption (exElGamal) is sufficient [5]. In fact, the decrypted results of exElGamal encryption can distinguish whether two messages m_1 and m_2 are equal, although the exElGamal scheme cannot decrypt messages itself. Furthermore, exElGamal can be used in (n, n)-threshold distributed decryption [9], where decryption must be performed by *all players acting together*. An exElGamal encryption with (n, n)-threshold distributed decryption consists of three functions:

Key generation
Let $\mathbb{F}_p$ be a finite field, $g \in \mathbb{F}_p$, with prime order q. Each player P_i chooses $x_i \in \mathbb{Z}_q$ at random and computes $y_i = g^{x_i} (\mathrm{mod}\ p)$. Then, $y = \prod_{i=1}^{n} y_i (\mathrm{mod}\ p)$ is a public key and each x_i is a share for each player to decrypt a ciphertext.

Encryption thrEnc[m] $\rightarrow (u, v)$
Choose $r \in \mathbb{Z}_q$ at random, and compute both $u = g^r (\mathrm{mod} p)$ and $v = g^m y^r (\mathrm{mod} p)$ for the input message $m \in \mathbb{Z}_q$ and a public key y. Output (u, v) as a ciphertext of m.

Decryption thrDec[(u, v)] $\rightarrow g^m$
Each player P_i computes $z_i = u^{x_i} (\mathrm{mod} p)$. All players then compute $z = \prod_{i=1}^{n} z_i (\mathrm{mod} p)$ jointly.[1] Finally, each player can decrypt the ciphertext as $g^m = v/z (\mathrm{mod} p)$.

ExElGamal encryption with (n, n)-threshold decryption has the following features:

(1) homomorphic under addition:
Enc(m_1) Enc(m_2)= Enc($m_1 + m_2$) for messages $m_1, m_2 \in \mathbb{Z}_p$.

(2) homomorphic under scalar operations:
Enc(m)^k = Enc(km) for a message m and $k \in \mathbb{Z}_q$.

Previous work

This section summarizes prior works on PSI between a server and a client and MPSI among n players. In PSI, let $S = \{s_1, ..., s_v\}$ and $C = \{c_1, ..., c_w\}$ be server and client datasets, where $|S| = v$ and $|C| = w$. In MPSI [17], we assume that each player holds the same number of datasets.

PSI protocol based on polynomial representation The main idea is to represent the elements in C as the roots of a polynomial. The encrypted polynomial is sent to the server, where it is evaluated on the elements in S, as originally proposed by Freedman [12]. This is secure against honest-but-curious adversaries under secure public key encryption. The computational complexity is $O(vw)$ exponentiations, and the communication overhead is $O(v + w)$. The computational complexity can be reduced to $O(v \log\log w)$ exponentiations using the balanced allocation technique [1]. Kissner and Song extended this protocol to MPSI [17], which requires $O(nw^2)$ exponentiations and $O(nw)$ communication overhead. The MPSI version is secure against honest-but-curious and malicious adversaries (in the random oracle model) using generic zero-knowledge proofs.

PSI protocol based on DH-key agreement The main objective here is to apply the DH-key agreement protocol [7]: after representing the server and client datasets as hash values $\{h(s_i)\}$ and $\{h(c_i)\}$, respectively, the client encrypts the dataset as $\{h(c_i)^{r_i}\}$ using a random number r_i and sends

[1] The computational complexity of z for each player can be made independent of the number of players in various ways. For example, set $z = 1$. P_1 computes $z = z \cdot z_1$ and sends z to P_2, P_2 computes $z = z \cdot z_2$ and sends z to P_3, and, finally, P_n computes $z = z \cdot z_n$ and shares z among all players. If we place all players in a binary tree, the communication complexity can be reduced, but each player's computational complexity is still independent of the number of players.

the encrypted set to the server. The server encrypts the client set $\{h(c_i)^{r_i}\}$ and the server set $\{h(s_i)\}$ using a random number r, which gives $\{h(c_i)^{rr_i}\}$ and $\{h(s_i)^r\}$, respectively, and returns these sets to the client. Finally, the client evaluates $S \cap C$ by decrypting to $\{h(c_i)^r\}$. This is secure against honest-but-curious adversaries under the DDH assumption. The total computational complexity is $O(v + w)$ exponentiations and the total communication overhead is $O(v + w)$. The security of this approach can be enhanced against malicious adversaries in the random oracle model [6] by using a blind signature. However, no extensions to MPSI based on the DH-key agreement protocol have been proposed.

PSI protocol based on BF This protocol was originally proposed in [18]. As the Bloom filter itself reveals information about the other player's dataset, the set of players is separated into two groups: input players who have datasets and privacy players who perform private computations under shared secret information. In [16], the privacy of each player's dataset is protected by encrypting each array of the Bloom filter using Goldwasser–Micali encryption [14]. In an honest-but-curious version, the computational complexity is $O(kw)$ hash operations and $O(m)$ public key operations, and the communication overhead is $O(m)$, where m and k are the number of arrays and hash functions, respectively, used in the Bloom filter. The Bloom filter is used in the Oblivious transfer extension [15, 20] and the newly constructed garbled Bloom filter [10]. The main novelty in the garbled Bloom filter is that each array requires λ bits, rather than the single bit needed for the conventional Bloom filter. To embed an element $x \in S$ to a garbled Bloom filter, x is split into k shares with λ bits using XOR-based secret sharing ($x = x_1 \bigoplus ... \bigoplus x_k$). The x_i are then mapped to an index of $H_i(x)$. An element y is queried by subjecting all bit strings at $H_i(y)$ to an XOR operation. If the result is y, then y is in S; otherwise, y is not in S. The client uses a Bloom filter $\mathsf{BF}(C)$ and the server uses a garbled Bloom filter $\mathsf{GBF}(S)$. If 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 can compute $C \cap S$. The computational complexity of this method is $O(kw)$ hash operations and $O(m)$ public key operations, and the communication overhead is $O(m)$. The number of public key operations can be changed to $O(\lambda)$ using the Oblivious transfer extension. This is secure against honest-but-curious adversaries if the Oblivious transfer protocol is secure. Finally, some researchers have computed the approximate number of multiparty set unions [11].

Practical MPSI

This section presents a practical MPSI that is secure under the honest-but-curious model.

Notation and privacy definition

In the remainder of this paper, the following notation is used.

- P_i: i-th player, $i = 1, \cdots, n$
- $\mathcal{O}$: outsourcing provider with no knowledge of the inputs or outputs
- $S_i = \{s_{i,1}, s_{i,2}, \cdots, s_{i,w_i}\}$: dataset held by P_i, where $|S_i| = \omega_i$
- $\cap S_j$: intersection of all n players
- thrEnc and thrDec: (n, n)-threshold exElGamal encryption and decryption, respectively
- m and k: number of arrays and hashes used in BF
- $\boldsymbol{\ell} = [\ell, \cdots, \ell]$ $(1 \leq \ell \leq n)$: an m-dimensional array, where all strings in the array are set to ℓ
- $\mathsf{BF}(S_i) = [\mathsf{BF}_i[0], \cdots, \mathsf{BF}_i[m-1]]$: Bloom filter applied to a set S_i
- $\mathsf{IBF}(\cup S_i) = [\sum_{i=1}^{n} \mathsf{BF}_i[0], \cdots, \sum_{i=1}^{n} \mathsf{BF}_i[m-1]]$: integrated Bloom filter of n sets $\{S_i\}$, where $\sum_{i=1}^{n} \mathsf{BF}_i[j]$ is the sum of all players' arrays.

We introduce an outsourcing provider $\mathcal{O}$ to reduce the computational burden on all players. The dealer has no information about the elements of any player's set. The privacy issues faced by MPSI with an outsourcing provider can be informally written as follows.

Definition 2 (MPSI privacy) An MPSI scheme with an outsourcing provider $\mathcal{O}$ is player-private if the following two conditions hold:

- P_i does not learn anything about the elements of other players' datasets except for the elements in $\cap S_j$.
- the outsourcing provider $\mathcal{O}$ does not learn anything about the elements of any player's set.

Proposed MPSI

Our MPSI consists of four phases: i) initialization, ii) Bloom filter construction and the encryption of P_i data, iii) the $\mathcal{O}$'s randomization of $\mathsf{thrEnc}(\mathsf{IBF}(\cup S_i) - \mathbf{n})$, and iv) the computation of $\cap \mathsf{P}_i$. The computation of $\cap \mathsf{P}_i$ consists of three steps: a) joint decryption of an (n, n)-threshold exElGamal among n players, b) Bloom filter check, and c) output intersection. Figure 1 shows an overview of our protocol after the initialization phase. The system parameters of a finite field $\mathbb{F}_p$ and a basepoint $g \in \mathbb{F}_p$ with order q for an (n, n)-threshold exElGamal encryption (thrEnc, thrDec) are provided to both P_i and $\mathcal{O}$. For the Bloom filter, $\mathsf{Const}(S)$ and $\mathsf{SetCheck}(\mathsf{BF}, \mathsf{S}')$ are only provided to P_i, where the array size is m and k independent hash functions are used.

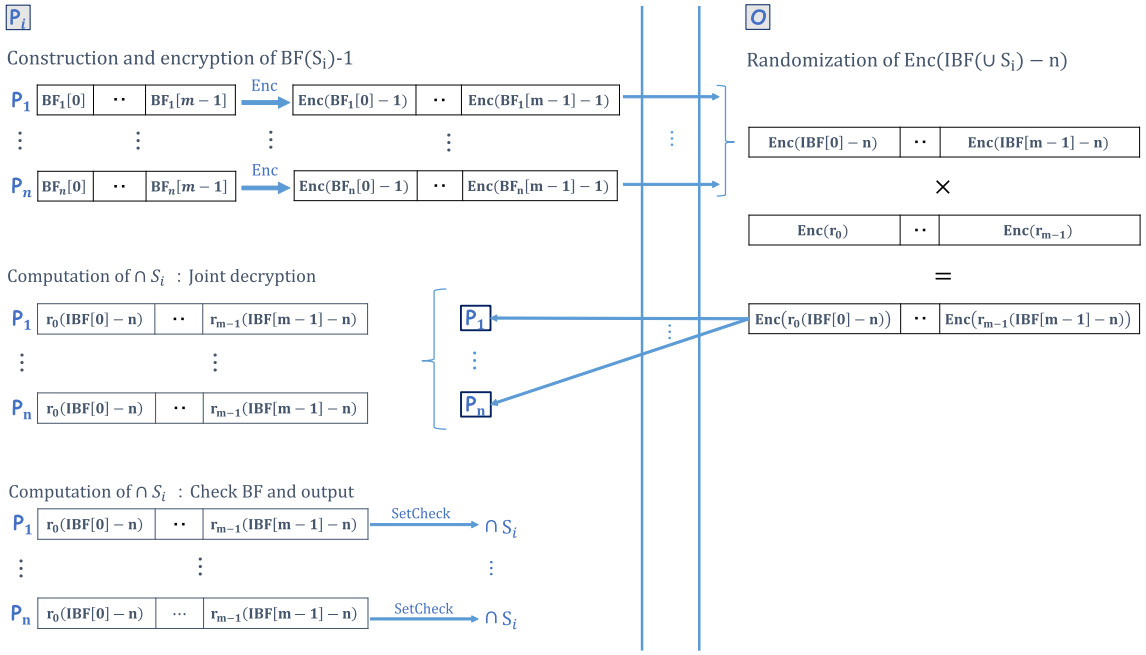

Fig. 1 Overview of our MPSI

To encrypt, randomize, or subtract a vector such as a Bloom filter $\mathsf{BF} = [a_0, \cdots, a_{m-1}]$, each location is encrypted, randomized, or subtracted independently:

$$\begin{aligned} \mathsf{thrEnc}(\mathsf{BF}) &= [\mathsf{thrEnc}(a_0), \cdots, \mathsf{thrEnc}(a_{m-1})], \\ \mathbf{r}\mathsf{BF} &= [r_0 a_0, \cdots, r_{m-1} a_{m-1}], \text{ or} \\ \mathsf{BF} - \mathbf{r} &= [a_0 - r_0, \cdots, a_{m-1} - r_{m-1}] \end{aligned}$$

for $\mathbf{r} = [r_0, \cdots, r_{m-1}] \in \mathbb{Z}_q^m$.

Our protocol proceeds as follows.

Initialization:

1. P_i generates $x_i \in \mathbb{Z}_q$, computes $y_i = g^{x_i} \in \mathbb{Z}_q$, and publishes y_i to the other players as a public key, where the corresponding secret key is x_i.
2. P_i computes $y = \prod_i y_i$, where y is an n-player public key. Note that no player knows the corresponding secret key $x = \sum x_i$ before executing the joint decryption.

Construction and encryption of $\mathsf{BF}(S_i) - \mathbf{1}$:

1. P_i executes $\mathsf{Const}(S_i) \longrightarrow \mathsf{BF}(S_i)$ $= [\mathsf{BF}_i[0], \cdots, \mathsf{BF}_i[m-1]]$ (Algorithm 1).
2. P_i encrypts $\mathsf{BF}(S_i) - \mathbf{1}$ using thrEnc_y:

$$\begin{aligned} &\mathsf{thrEnc}_y(\mathsf{BF}(S_i) - \mathbf{1}) \\ &= [\mathsf{thrEnc}_y(\mathsf{BF}_i[0]-1), \cdots, \mathsf{thrEnc}_y(\mathsf{BF}_i[m-1]-1)], \end{aligned}$$

   where y is an n-player public key.
3. P_i sends $\mathsf{thrEnc}_y(\mathsf{BF}(S_i) - \mathbf{1})$ to $\mathcal{O}$.

Randomization of $\mathsf{thrEnc}(\mathsf{IBF}(\cup S_i) - \mathbf{n})$:

1. $\mathcal{O}$ encrypts $\mathsf{IBF}(\cup S_i) - \mathbf{n}$ without knowing $\mathsf{IBF}(\cup S_i)$ using an additive homomorphic feature and multiplying by $\mathsf{thrEnc}_y(\mathsf{BF}(S_i) - \mathbf{1})$ as follows:

$$\mathsf{thrEnc}_y(\mathsf{IBF}(\cup S_i) - \mathbf{n}) = \prod_{i=1}^{n} \mathsf{thrEnc}_y(\mathsf{BF}(S_i) - \mathbf{1}).$$

2. $\mathcal{O}$ randomizes $\mathsf{thrEnc}_y(\mathsf{IBF}(\cup S_i) - \mathbf{n})$ as $\mathbf{r} = [r_0, \cdots, r_{m-1}] \in \mathbb{Z}_q^m$:

$$\mathsf{thrEnc}_y(\mathbf{r}(\mathsf{IBF}(\cup S_i) - \mathbf{n})) = (\mathsf{thrEnc}_y(\mathsf{IBF}(\cup S_i) - \mathbf{n}))^{\mathbf{r}}.$$

3. $\mathcal{O}$ broadcasts $\mathsf{thrEnc}_y(\mathbf{r}(\mathsf{IBF}(\cup S_i) - \mathbf{n}))$ to P_i.

Computation of $\cap P_i$:

1. All players decrypt $\mathsf{thrEnc}_y(\mathbf{r}(\mathsf{IBF}(\cup S_i) - \mathbf{n}))$ jointly.
2. P_i computes $\mathsf{SetCheck}(\mathbf{r}(\mathsf{IBF}(\cup S_i) - \mathbf{n}), S_i)$ and obtains $\cap S_i$.

The above protocol satisfies the correctness requirement. This is because each array position of $\mathsf{thrEnc}_y(\mathbf{r}(\mathsf{IBF}(\cup S_i) - \mathbf{n}))$ is decrypted to 1, where $x \in \cap S_i$ is embedded by each hash function; however, each array position for which $x \notin \cap S_i$ is embedded by each hash function is decrypted to a random value.

Security Proof

The security of our MPSI protocol is as follows.

Theorem 1 *For any coalition of fewer than n players, MPSI is player-private against an honest-but-curious adversary under the DDH assumption.*

Proof The views of P_i and $\mathcal{O}$, that is,

$$\mathsf{thrEnc}_y(\mathsf{BF}_{m,k}(S_i)) = [\mathsf{thrEnc}_y(\mathsf{BF}_i[0]), \cdots, \mathsf{thrEnc}_y(\mathsf{BF}_i[m-1])],$$

are shown to be indistinguishable from a random vector $\mathbf{r} = [r_0, \cdots, r_{m-1}] \in \mathbb{Z}_q^m$. Assume that a polynomial-time distinguisher $\mathcal{D}$ outputs 0 when the views are presented as a random vector and outputs 1 when they are constructed in MPSI, $\mathsf{thrEnc}(\mathsf{BF}_i[0]), \cdots, \mathsf{thrEnc}(\mathsf{BF}_i[m-1])$. We show that a simulator $\overline{\mathrm{SIM}}$ that solves the DDH assumption can be constructed as follows.

Upon receiving a DDH challenge $(\overline{g}, \overline{g}^{\alpha}, \overline{g}^{\beta}, \overline{g}^{\gamma})$, $\overline{\mathrm{SIM}}$ executes the following:

1. Set n-player public key $y = \overline{g}^{\beta}$ and choose random numbers $d_0, ..., d_{m-1}$ and $r_1, ..., r_{m-1}$ from $\mathbb{Z}_q$.
2. Send
   $$[(\overline{g}^{\alpha}, \overline{g}^{d_0}\overline{g}^{r}), (\overline{g}^{\alpha})^{r_1}, \overline{g}^{d_1} \cdot (\overline{g}^{\gamma})^{r_1}, \cdots, \overline{g}^{d_{m-1}} \cdot (\overline{g}^{\gamma})^{r_{m-1}}]$$
   as $\overline{\mathsf{thrEnc}_y(\mathsf{BF}_{m,k}(S_i))}$ to $\mathcal{D}$.

If $(\overline{g}, \overline{g}^{\alpha}, \overline{g}^{\beta}, \overline{g}^{\gamma})$ is a DH-key-agreement-protocol element, i.e., $\gamma = \alpha\beta$, then $\overline{\mathsf{thrEnc}_y(\mathsf{BF}_{m,k}(S_i))}$ is distributed in the same way as when constructed by the MPSI scheme. Thus, $\mathcal{D}$ must output 1. If $(\overline{g}, \overline{g}^{\alpha}, \overline{g}^{\beta}, \overline{g}^{\gamma})$ is not a DH tuple, then $\overline{\mathsf{thrEnc}_y(\mathsf{BF}_{m,k}(S_i))}$ is randomly distributed, and $\mathcal{D}$ has to output 0. As a result, $\overline{\mathrm{SIM}}$ can use the output of $\mathcal{D}$ to respond to the DDH challenge correctly. Therefore, $\mathcal{D}$ can answer correctly with negligible advantage over random guessing. Furthermore, as all inputs of each player are encrypted until the decryption is performed, and decryption cannot be performed by fewer than n players, nothing can be learned by any player prior to decryption.

As for the views of $\mathsf{thrEnc}_y(\mathbf{r}(\mathsf{IBF}_{m,k}(\cup S_i) \setminus \mathbf{n}))$, the same argument holds. Therefore, for any coalition of fewer than n players, MPSI is player-private under the honest-but-curious model. □

Efficiency

Although many PSI protocols have been proposed, to the best of our knowledge, relatively few have considered the multiparty scenario [11, 17, 18, 21]. Our target is multiparty private set intersection, and the final result must be obtained by *all* players acting together, without a trusted third-party (TTP). Among previous MPSI protocols, the approach in [11] computes only the approximate number of intersections, and that in [18] requires more than two TTPs. In contrast, [21] follows almost the same method as [17] and thus has a similar complexity. The only difference exists in the security model. Hence, we only compare our scheme with that of [17].

The computational and communication efficiency of the proposed protocol and [17] are compared in Table 1. These approaches are secure against honest-but-curious adversaries without a TTP under exElGamal encryption (DDH security) and Paillier encryption (Decisional Composite Residue (DCR) security), respectively.

Our MPSI uses the Bloom filter for the computations performed by P_i and the integrations performed by the $\mathcal{O}$. The use of a Bloom filter eliminates the restriction on set size. Thus, in our MPSI, the set size of each player is flexible. However, P_i's computations consist of Bloom filter construction, joint decryption, and Bloom filter check. Neither the computations related to the Bloom filter nor the joint decryption depends on the number of players, as shown in "Preliminaries". In summary, the computational complexity of operations performed by P_i is $O(\omega_i)$. All player-dependent data are sent to $\mathcal{O}$, who integrates $\prod_{i=1}^{n} \mathsf{thrEnc}_y(\mathsf{IBF}(\cup S_i))$ without decryption. As a result, the computational complexity of operations performed by $\mathcal{O}$ is $O(n\omega)$.

Implementation results

Implementation

To investigate the behavior and performance of our MPSI protocol, we implemented a prototype in C++ using the GNU Multi-Precision (GMP) library (version 5.1.3) and OpenSSL (version 1.0.1f). GMP is used for large-integer arithmetic and random number generation in the exElGamal encryption. To instantiate hash functions for the Bloom filter, we used SHA-1 in OpenSSL: $H_i(x) := \mathrm{sha1}(s_i \parallel x) \bmod m$, where s_i is a unique salt. This truncation of the hash functions is based on the recommendation of the National Institute of Standards and Technology (NIST) [8]. Each executable communicates through TCP. We used Boost.Asio C++ 1.54.0 for the TCP socket.

The C++ prototype has two executables: one for the players and one for the outsourcing provider. The prototype can work in either pipeline or parallel mode. In pipeline mode, the computation and communication threads are

Table 1 Efficiency of [17] and the proposed protocol

	[17]	Ours
Computational complexity	$O(n\omega^2)$	$\mathsf{P}_i : O(\omega_i), O : O(n\omega)$
Communication overhead	$O(n\omega)$	$\mathsf{P}_i : O(\omega + n), O : O(n\omega)$
Restriction on set size	$\lvert S_1\rvert = \ldots = \lvert S_n\rvert$	none
Protected values	$S_i (\forall i \in [1, n])$	$S_i, \lvert S_i\rvert (\forall i \in [1, n])$

separated. Thus, computation and data transmission are processed in parallel when possible. Pipeline mode allows each executable to start immediately without waiting for the completion of all previous computations. Parallel mode extends the pipeline mode by multiplying the number of computation threads in each executable. The most expensive process of our protocol is Bloom filter encryption and decryption. In parallel mode, the encryption and decryption computation is conducted in multiple threads. This significantly improves the performance of our protocol.

Evaluation

All experiments were performed on the Google Compute Engine (GCE). GCE is a cloud computing system that delivers virtual machines running in Google's data centers. In our experiments, each executable was calculated on a single virtual machine. We used the Ubuntu 14.04 LTE operating system with Intel Xeon 2.50 GHz CPUs. Each CPU core was assigned 3.75 GB of memory. Every virtual machine was connected to a virtual private network. The bandwidth between two virtual machines was approximately 2.0 Gbps, although our protocol used less than 10 Mbps.

The time required for Bloom filter construction, encryption, decryption, randomization procedures, and MPSI computation was measured. However, the measurements do not include initialization and finalization, e.g., parsing command lines, reading and writing CSV files, TCP socket setup and shutdown, and public key exchange. Each player input a database set of size 2^6–2^{14}. We measured the performance for $n = 4, 8, 16$ and tested the security parameters for 80-bit, 112-bit, 128-bit, 196-bit, and 256-bit security. Each security parameter is half of the bit size of q. The evaluation of the security parameter is based on the NIST guidelines for key management [2], as summarized in Table 2. We chose a false positive rate $\mathsf{FPR} = 0.65\ \%$, as was adopted in [18].

First, we report the runtimes in pipeline mode. The performance measurements are presented in Tables 3 and 4 (Figs. 2, 3, 4, and 5). To measure each executable time separately, we excluded the wait time for communication. From Table 3, it is clear that the runtime scales almost linearly

Table 2 Security parameter and group size

security parameter	$\lvert p\rvert$	$\lvert q\rvert$
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	512

All numbers shown in the table are in bits

Table 3 Pipeline mode performance (80-bit security)

n	exe	Set size				
		2^6	2^8	2^{10}	2^{12}	2^{14}
4	$\mathcal{O}$	0.65	2.69	10.4	36.7	151
	P	0.82	3.39	13.4	54.1	214
8	$\mathcal{O}$	0.76	2.95	12.4	44.4	178
	P	0.90	3.75	15.7	60.3	241
16	$\mathcal{O}$	0.90	3.64	15.8	56.4	225
	P	1.30	4.71	19.2	76.1	307

All times in the table are in seconds

Table 4 Pipeline mode performance (set size = 2^6)

n	exe	Security parameter (bit)				
		80	112	128	192	256
4	$\mathcal{O}$	0.61	2.74	8.29	57.2	275
	P	0.87	4.28	11.1	85.7	417
8	$\mathcal{O}$	0.72	2.95	7.84	58.1	277
	P	1.43	4.38	10.8	86.9	417
16	$\mathcal{O}$	0.90	3.41	9.09	61.4	284
	P	1.30	5.18	12.0	91.8	433

All times in the table are in seconds

Table 5 Breakdown of runtime (set size = 2^6, $n = 4$)

exe	Process	Security parameter (bit)				
		80	112	128	192	256
$\mathcal{O}$	(A)	0.61	2.74	8.29	57.2	275
P	(B)	0.50	2.67	6.79	55.8	275
	(C)	0.37	1.60	4.35	29.9	142
	(D)	~ 0.01	~ 0.01	~ 0.01	~ 0.01	~ 0.01

All times in the table are in seconds

Table 6 Breakdown of runtime (set size = 2^6, Security parameter = 80)

exe	Process	Number of Players		
		4	8	16
$\mathcal{O}$	(A)	0.55	0.67	0.82
P	(B)	0.45	0.44	0.44
	(C)	0.34	0.43	0.67
	(D)	~ 0.01	~ 0.01	~ 0.01

All times in the table are in seconds

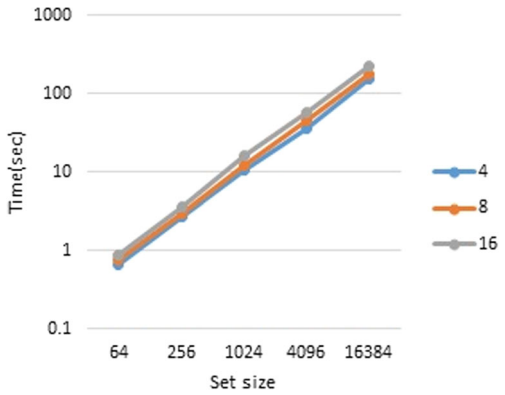

Fig. 2 Outsourcing provider, 80-bit security

with the set size. It is also apparent that the player's runtime increases in accordance with n. This is because, in our implementation, each player performs the joint decryption process independently. However, the joint decryption process can be distributed by the players so that the computational complexity remains constant with respect to n. The outsourcing provider's runtime obeys scales with the computational complexity, namely, $O(n\omega)$. The breakdown of runtimes is presented in Tables 5 and 6.

The processes described in the table are as follows:

- Outsourcing provider
- (A) Randomization of $\mathsf{thrEnc}(\mathsf{IBF}(\cup S_i) - \mathbf{n})$
- Player
- (B) Construction and encryption of $\mathsf{BF}(S_i) - \mathbf{1}$
- (C) Joint decryption of $\mathsf{thrEnc}_y(\mathbf{r}(\mathsf{IBF}_{m,k}(\cup S_i) - \mathbf{n}))$
- (D) $\mathsf{SetCheck}(\mathbf{r}(\mathsf{IBF}(\cup S_i) - \mathbf{n}), S_i)$ and obtains $\cap S_i$

Clearly, the time consumption is dominated by the encryption and decryption of the Bloom filter array.

The performance measurements in parallel mode are presented in Table 7 (Fig. 6). We fixed the security parameter at 80-bit security and measured the total runtime,

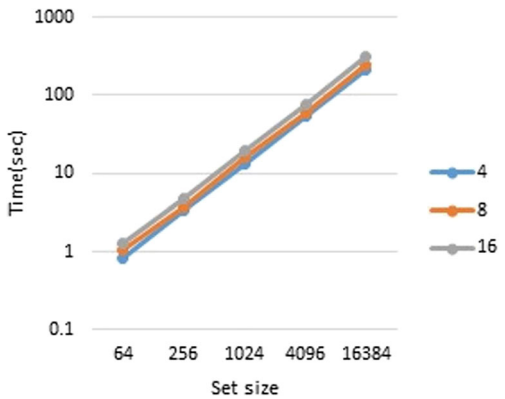

Fig. 3 Player, 80-bit security

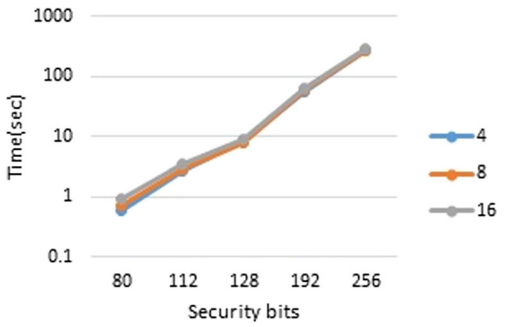

Fig. 4 Outsourcing provider, set size = 2^6

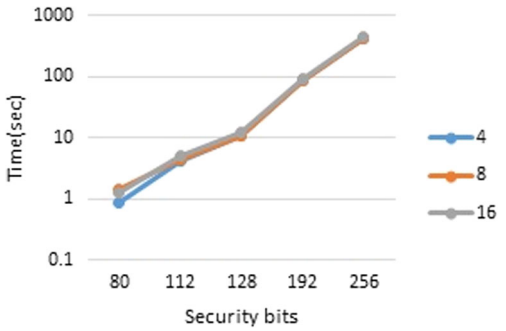

Fig. 5 Player, set size = 2^6

Table 7 Parallel mode performance (80-bit security)

CPU core	Set size				
	2^6	2^8	2^{10}	2^{12}	2^{14}
1	1.02	3.89	15.0	82.9	297
2	1.49	2.83	8.72	33.0	131
4	1.33	2.22	6.14	22.6	87.1

All times in the table are in seconds

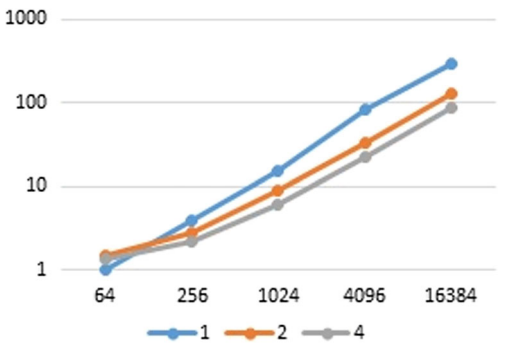

Fig. 6 Parallel mode performance (80-bit security)

Table 8 Performance comparison (80-bit security)

Protocol	Set size				
	2^6	2^8	2^{10}	2^{12}	2^{14}
Kissner and Song's ($n = 4$)	0.50	3.06	50.6	1051	N/A
Our protocol ($n = 4$)	1.02	3.89	15.0	82.9	297
Kissner and Song's ($n = 8$)	0.92	6.41	92.0	1491	N/A
Our protocol ($n = 8$)	1.50	3.05	19.4	83.2	355
Kissner and Song's ($n = 16$)	2.10	13.9	190	3246	N/A
Our protocol ($n = 16$)	1.98	7.29	28.7	112	450

All times in the table are in seconds

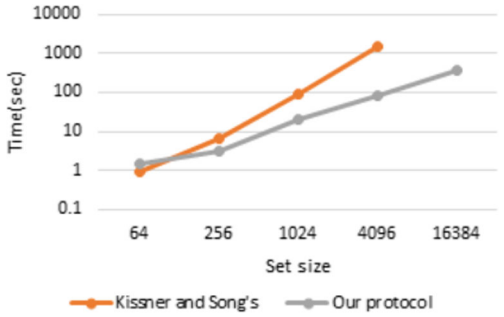

Fig. 8 $n = 8$

including the computation time and the wait time for communication. Although the total runtimes are not exactly proportional to the number of CPU cores, there is a significant improvement in the multi-core environment. As the time consumption of our protocol is dominated by the encryption and decryption of the Bloom filter array, these processes can easily be implemented in parallel. We believe this property is one of the most important advantages of our protocol.

Comparison

We compared our protocol with Kissner and Song's MPSI protocol [17]. We implemented Kissner and Song's MPSI protocol with PARI in C++ for the comparison. All measurements were conducted in pipeline mode. The results are presented in Table 8 (Figs. 7, 8 and 9).

The results show that our protocol is faster than Kissner and Song's MSPI protocol when $n = 4$ and the set size is greater than 2^8, when $n = 8$ and the set size is greater than 2^6, and when $n = 16$ and the set size is greater than 2^4. Furthermore, although Kissner and Song's MSPI protocol crashed with a set size of 2^{14}, these results reveal that the time consumption of their protocol is approximately proportional to the square of the set size. As in our protocol, Kissner and Song's MSPI protocol uses the (n, n)-threshold scheme, so it does not require a conspiracy assumption. However, their protocol is not scalable with respect to either the set size or number of players.

Conclusion

This paper has described a practical MPSI in which some of the computations are outsourced to a third-party. As none of the information of $S_i, |S_i|(\forall i \in [1, n])$ is revealed to the third-party, this function can be safely outsourced. Our scheme satisfies that the following requirements: any restrictions on the sets are eliminated, meaning that the set size of each player can be flexibly chosen; and the computational burden on each player is independent of the number of players.

Importantly, our scheme can be applied to the efficient integration of medical and related data maintained by different organizations without violating any privacy constraints. We confirmed that the computational complexity is independent of the number of organizations from which data are being integrated.

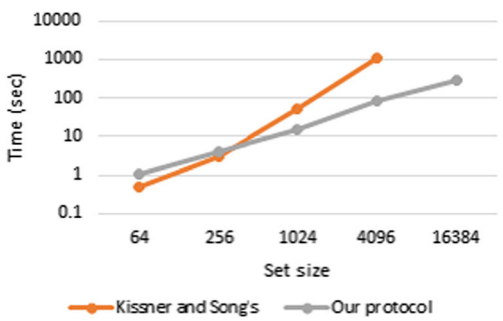

Fig. 7 $n = 4$

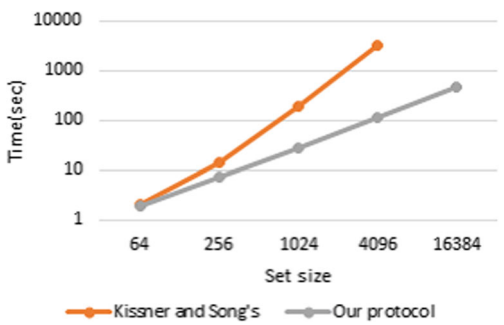

Fig. 9 $n = 16$

Acknowledgments The authors express our gratitude to anonymous referees for invaluable comments. This work is supported in part by a Grant-in-Aid for Scientific Research (C)(15K00183) and (15K00189) and the Japan Science and Technology Agency, CREST, and Infrastructure Development for Promoting International S&T Cooperation.

References

1. Azar, Y., Broder, A. Z., Karlin, A. R., and Upfal, E., Balanced allocations. *SIAM J. Comput.* 29(1):180–200, 1999.
2. Barker, E., Barker, W., Burr, W., Polk, W., and Smid, M.: Nist special publication 800-57: Recommendation for key management – part 1: General(revision 3). Technical report, National Institute of Standards and Technology (NIST), 2012.
3. Bloom, B. H., Space/time trade-offs in hash coding with allowable errors. *Commun. ACM* 13(7):422–426, 1970.
4. Broder, A., and Mitzenmacher, M., Network applications of bloom filters: A survey. *Internet Math.* 1(4):485–509, 2004.
5. Cramer, R., Gennaro, R., and Schoenmakers, B., A secure and optimally efficient multi-authority election scheme. *Eur. Trans. Telecommun.* 8(5):481–490, 1997.
6. De Cristofaro, E., Kim, J., and Tsudik, G., Linear-complexity private set intersection protocols secure in malicious model. In: ASIACRYPT 2010, volume 6477 of LNCS, pages 213–231. Springer, 2010.
7. De Cristofaro, E., and Tsudik, G., Practical private set intersection protocols with linear complexity. In: FC 2010, volume 6052 of LNCS, pages 143–159. Springer, 2010.
8. Dang, Q., Nist special publication 800-107: Recommendation for applications using approved hash algorithms(revision 1). Technical report, National Institute of Standards and Technology (NIST), 2012.
9. Desmedt, Y., and Frankel, Y., Threshold cryptosystems. In: CRYPTO 1989, volume 1462 of LNCS, pages 307–315. Springer, 1989.
10. Dong, C., Chen, L., and Wen, Z., When private set intersection meets big data: An efficient and scalable protocol. In: ACMCCS 2013, pages 789–800. ACM, 2013.
11. Egert, R., Fischlin, M., Gens, D., Jacob, S., Senker, M., and Tillmanns, J., Privately computing set-union and set-intersection cardinality via bloom filters. In: ACISP 2015, volume 9144 of LNCS, pages 413–430. Springer, 2015.
12. Freedman, M. J., Nissim, K., and Pinkas, B., Efficient private matching and set intersection. In: EUROCRYPT 2004, volume 3027 of LNCS, pages 1–19. Springer, 2004.
13. Goldreich, O., Secure multi-party computation. Manuscript. Preliminary version, 1998.
14. Goldwasser, S., and Micali, S., Probabilistic encryption. *J. Comput. Syst. Sci.* 28(2):270–299, 1984.
15. Ishai, Y., Kilian, J., Nissim, K., and Petrank, E., Extending oblivious transfers efficiently. In: CRYPTO 2003, volume 2729 of LNCS, pages 145–161. Springer, 2003.
16. Kerschbaum, F., Outsourced private set intersection using homomorphic encryption. In: ACMCCS 2012, pages 85–86. ACM, 2012.
17. Kissner, L., and Song, D., Privacy-preserving set operations. In: CRYPTO 2005, volume 3621 of LNCS, pages 241–257. Springer, 2005.
18. Many, D., Burkhart, M., and Dimitropoulos, X., Fast private set operations with sepia. *Tech. Rep.* 345, 2012.
19. Paillier, P., Public-key cryptosystems based on composite degree residuosity classes. In: EUROCRYPT 1999, volume 1592 of LNCS, pages 223–238. Springer, 1999.
20. Rabin, M. O., *How to exchange secrets with oblivious transfer*. *Tech. Memo, TR-81*, 1981.
21. Sang, Y., and Shen, H., Efficient and secure protocols for privacy-preserving set operations. *ACM Trans. Inf. Syst. Secur.* 13(1):9:1–9:35, 2009.