How to Meet Big Data When Private Set Intersection Realizes Constant Communication Complexity

Sumit Kumar Debnath(✉) and Ratna Dutta

Department of Mathematics, Indian Institute of Technology Kharagpur, Kharagpur 721302, India
sd.iitkgp@gmail.com, ratna@maths.iitkgp.ernet.in

Abstract. This paper presents the *first* PSI protocol that achieves constant ($O(1)$) communication complexity with linear computation overhead and is fast even for the case of large input sets. The scheme is proven to be provably secure in the standard model against semi-honest parties. We combine *somewhere statistically binding (SSB) hash function* with *indistinguishability obfuscation (iO)* and *Bloom filter* to construct our PSI protocol.

Keywords: PSI · SSB hash · iO · Bloom filter

1 Introduction

Electronic information is increasingly often shared among unreliable entities. In this context, one interesting problem involves two parties that secretly want to determine intersection of their respective private data sets while none of them wish to disclose the whole set to other. One can adopt Private Set Intersection (PSI) protocol to address this problem preserving the associated security and privacy issues. It is a two-party cryptographic protocol where each party engages with their private sets. On completion of the protocol either only one of the participants learns the intersection and other learns nothing, yielding one-way PSI or both of them learn the intersection, yielding mutual PSI (mPSI). PSI has emerged a great attention in the recent research community due to its numerous applications in real-life such as privately comparing equal-size low-entropy vectors, collaborative botnet detection, testing of fully sequenced human genomes, affiliation-hiding authentication, social networks, location-based services, privacy preserving data mining, social networks, online gaming etc.

Our Contribution: Our goal is to construct a PSI whose communication cost is optimal while the computation cost is comparable with the existing schemes.

In this paper, we design a new PSI protocol based on *Bloom filter* [2] that is significantly more efficient than all the existing PSI protocols. We adopt a novel two-party computation technique and make use of *somewhere statistically binding (SSB) hash function* [11,13] along with *indistinguishability obfuscation*

K.-Y. Lam et al. (Eds.): ICICS 2016, LNCS 9977, pp. 445–454, 2016.
DOI: 10.1007/978-3-319-50011-9_34

(iO) [1,7]. Starting point of our construction is the approach of [13] of secure function evaluation (SFE) for "multi-decryption".

In our protocol, the client B sends SSB hash value of its private input set to the server A who in turn transmits to B an SSB hash key, obfuscated version of a hard-coded circuit and a Bloom filter. The use of SSB hash reduces the communication complexity of our protocol to constant ($O(1)$) which is due to only three bit-strings of length m (size of a Bloom filter), $p(\kappa)|C|$ (size of an obfuscated circuit) and $w\lfloor\log_2 n\rfloor$ (size of an element of $\mathbb{Z}_{n^w}$), where $p(\kappa)$ is a polynomial in security parameter κ, w is a positive integer, n is the product of two large primes and $\lfloor n\rfloor$ stands for the largest integer less than or equal n. On the other hand, the computation overhead of our protocol is $O(v+L)$ which depends on an SSB hash key computation, a circuit obfuscation, v many Pseudorandom function (PRF) evaluations by the server A and L many circuit evaluation by the client B. Our protocol is secure against semi-honest adversaries in the standard model. For simplicity, we employ the SSB hash [13] based on the Damgård-Jurik cryptosystem [4] secure under the Decisional Composite Residuosity (DCR) assumption. However, any SSB hash can be integrated to construct our PSI protocol.

Constructing PSI for big data sets is a challenging task while efficiency and scalability need to be preserved. Our PSI can easily be adopted to solve this big data issue. To the best of our knowledge, [5,6,16–18] are the most efficient PSI protocols, among which only [5,17] solve the big data issue. All of these protocols attain linear computation complexity while none of them achieve constant communication complexity. On a more positive note, our PSI is the *first* to achieve *constant* communication complexity.

2 Preliminaries

Throughout the paper, the notations κ, $a \leftarrow A$, $x \hookleftarrow X$ and $\{\mathcal{X}_t\}_{t\in\mathcal{N}} \overset{c}{\equiv} \{\mathcal{Y}_t\}_{t\in\mathcal{N}}$ are respectively used to represent "security parameter", "a is output of the procedure A", "variable x is chosen uniformly at random from set X" and "the distribution ensemble $\{\mathcal{X}_t\}_{t\in\mathcal{N}}$ is computationally indistinguishable from the distribution ensemble $\{\mathcal{Y}_t\}_{t\in\mathcal{N}}$". Formally, $\{\mathcal{X}_t\}_{t\in\mathcal{N}} \overset{c}{\equiv} \{\mathcal{Y}_t\}_{t\in\mathcal{N}}$ means for all probabilistic polynomial time (PPT) distinguisher $\mathcal{Z}$, there exists a negligible function $\epsilon(t)$ such that $|Prob_{x\leftarrow\mathcal{X}_t}[\mathcal{Z}(x)=1] - Prob_{x\leftarrow\mathcal{Y}_t}[\mathcal{Z}(x)=1]| \leq \epsilon(t)$. A function $\epsilon : \mathbb{N} \rightarrow \mathbb{R}$ is said to be negligible function of κ if for each constant $c > 0$, we have $\epsilon(\kappa) = o(\kappa^{-c})$ for all sufficiently large κ.

Definition 1. Pseudorandom Function [10]**:** *A random instance $f_k(\cdot)$ is said to be Pseudorandom Function (PRF) for a randomly chosen key k, if the value of the function cannot be distinguished from a random function $\hat{f} : D \rightarrow E$ by any PPT distinguisher $\mathcal{Z}$ i.e., $|Prob[\mathcal{Z}^{f_k}(1^\kappa)=1] - Prob[\mathcal{Z}^{\hat{f}}(1^\kappa)=1]|$ is negligible function of κ.*

A PRF $f_k(\cdot)$ is an efficiently computable function i.e., one can compute $f_k(x)$ using a PPT algorithm for any given $x \in D$. For example, the PRF of [12]:

$f_k(x) = g^{1/(k+x)}$ if $gcd(k+x, n)$ is 1, 1 otherwise. The pseudorandom function is secure under the Decisional Q-Diffie-Hellman Inversion (DHI) assumption [12].

2.1 Damgård-Jurik Cryptosystem [4]

The Damgård-Jurik cryptosystem DJ is a generalization of the Paillier cryptosystem [14] and consists of algorithms $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ which work as follows:

- $\mathsf{DJ.KGen}(1^\kappa) \rightarrow (\mathsf{pk}, \mathsf{sk})$: On input 1^κ, a user does the following:
  – selects two large primes p, q independently of each other;
  – sets $n = pq$ and $\gamma = lcm(p-1, q-1)$;
  – chooses an element $g \in \mathbb{Z}^*_{n^{w+1}}$ for some $w \in \mathbb{N}$ such that $g = (1+n)^j x \bmod n^{w+1}$ for a known j relatively prime to n and $x \in \widetilde{G}$, where $\mathbb{Z}^*_{n^{w+1}} = G \times \widetilde{G}$, G being a cyclic group of order n^w and $\widetilde{G}$ is isomorphic to $\mathbb{Z}^*_n$;
  – computes d using the Chinese Remainder Theorem satisfying that $d \bmod n \in \mathbb{Z}^*_n$ and $d = 0 \bmod \gamma$;
  – sets the public key $\mathsf{pk} = (n, g, w)$ and the secret key $\mathsf{sk} = d$;
  – publishes pk and keeps sk secret to itself.
- $\mathsf{DJ.Enc}(\mathsf{pk}, m) \rightarrow (c)$: Using the public key pk of a decryptor, an Encryptor encrypts a message $m \in \mathbb{Z}_{n^w}$ by selecting $r \longleftarrow \mathbb{Z}^*_{n^{w+1}}$ and computing ciphertext $c = g^m r^{n^w} \bmod n^{w+1}$.
- $\mathsf{DJ.Dec}(\mathsf{sk}, c) \rightarrow (m)$: On receiving the ciphertext c from the encryptor, the decryptor uses its decryption key $\mathsf{sk} = d$ to compute $c^d \bmod n^{w+1}$. The decryptor then applies recursive version of the Paillier decryption mechanism [4] to obtain $(j \cdot m \cdot d) \bmod n^w$ and $(j \cdot d) \bmod n^w$ respectively from $a = c^d = (1+n)^{(j \cdot m \cdot d)} \bmod n^{w+1}$ and $a = g^d = (1+n)^{(j \cdot d)} x^d = (1+n)^{(j \cdot d)} \bmod n^{w+1}$. As $(j \cdot d)$ and $(j \cdot m \cdot d)$ are known to the decryptor, it can compute $m = (j \cdot m \cdot d)(j \cdot d)^{-1} \bmod n^w$.

The scheme is additively homomorphic as there exists an operation $\oplus$ over $\mathbb{Z}_{n^{w+1}}$ $\mathsf{DJ.Enc}(\mathsf{pk}, m_1; r_1) \oplus \mathsf{DJ.Enc}(\mathsf{pk}, m_2; r_2) = \mathsf{DJ.Enc}(\mathsf{pk}, m_1 + m_2; r_3)$ for randomness $r_3 = r_1 r_2$, where $+$ is over $\mathbb{Z}_{n^w}$ and $\mathbb{Z}_{n^{w+1}}$ respectively. We can define homomorphic subtraction $\ominus$ over $\mathbb{Z}_{n^{w+1}}$ as $\mathsf{DJ.Enc}_{\mathsf{pk}}(m_1; r_1) \ominus \mathsf{DJ.Enc}_{\mathsf{pk}}(m_2; r_2) = \mathsf{DJ.Enc}_{\mathsf{pk}}(m_1 - m_2; r_4)$ for randomness $r_4 = \frac{r_1}{r_2}$, where the operations $-$ is over $\mathbb{Z}_{n^w}$. Furthermore, by performing repeated $\oplus$ operation, we can implement an operation $\otimes$ over $\mathbb{Z}_{n^{w+1}}$ as $\mathsf{DJ.Enc}_{\mathsf{pk}}(m_1; r_1) \otimes m_2 = \oplus^{m_2}(\mathsf{DJ.Enc}_{\mathsf{pk}}(m_1; r_1)) = \mathsf{DJ.Enc}_{\mathsf{pk}}(m_1 \cdot m_2; r_5)$ for randomness $r_5 = r_1^{m_2}$, where $\cdot$ is over $\mathbb{Z}_{n^w}$ and $\mathbb{Z}_{n^{w+1}}$ respectively. The semantic security of the cryptosystem DJ holds under the DCR [14] assumption defined below:

Definition 2. Decisional Composite Residuosity (DCR) Assumption [14]**:** *On input 1^κ, let RGen be an algorithm that generates an RSA modulus $n = pq$, where p and q are distinct large primes. The DCR assumption states that given an RSA modulus n (without its factorization) and an integer z, it is computationally hard to decide whether z is an n-th residue modulo n^2, i.e., whether there exists $y \in \mathbb{Z}^*_{n^2}$ such that $z \equiv y^n \pmod{n^2}$.*

2.2 SSB Hash [11,13]

A somewhere statistically binding (SSB) hash $\mathsf{SSBHash}$ consists of PPT algorithms $(\mathsf{Gen}, \mathcal{H}, \mathsf{Open}, \mathsf{Verify})$ along with a finite block alphabet $\Sigma = \{0,1\}^{l_{\mathsf{blk}}}$, output size l_{hash} and opening size l_{opn}, where $l_{\mathsf{blk}}(\kappa), l_{\mathsf{hash}}(\kappa), l_{\mathsf{opn}}(\kappa)$ are fixed polynomials in the security parameter κ. An SSB hash satisfies the following three properties – Correctness, Index Hiding and Somewhere Statistically Binding. For more details see [11,13]. We describe below the DCR based SSB hash of [13] which uses Damgård-Jurik cryptosystem as described in Sect. 2.1 and considers $\Sigma = \mathbb{Z}_{n^w}$ i.e., $l_{\mathsf{blk}} = \lfloor w \log_2 n \rfloor$, output domain as $\mathbb{Z}_{n^w}$ i.e., $l_{\mathsf{hash}} = \lfloor w \log_2 n \rfloor$ and opening domain as $\times^{\alpha}(\mathbb{Z}_{n^w}) = \mathbb{Z}_{n^w} \times \ldots \times \mathbb{Z}_{n^w}$ (α times) i.e., $l_{\mathsf{opn}} = \alpha \lfloor w \log_2 n \rfloor$

- $\mathsf{SSBHash.Gen}(1^{\kappa}, 1^{l_{\mathsf{blk}}}, L, i) \rightarrow (hk)$: Without any loss of generality, we assume that $L = 2^{\alpha}$ is an integer with $L \leq 2^{\kappa}$. A setup authority runs the key generation algorithm for Damgård-Jurik cryptosystem DJ on input 1^{κ} to receive $(\mathsf{pk} = (n, g, w), \mathsf{sk} = d) \leftarrow \mathsf{DJ.KGen}(1^{\kappa})$. Let $(b_{\alpha}, \ldots, b_1)$ be the binary representation of the index $i \in \{0, \ldots, L-1\}$. For $l = 1, \ldots, \alpha$, the setup authority computes $g^{b_l}\gamma_l^{n^w} = c_l = \mathsf{DJ.Enc}(\mathsf{pk}, b_l; \gamma_l)$, $gR_l^{n^w} = 1_{\mathsf{ch}_l} = \mathsf{DJ.Enc}(\mathsf{pk}, 1; R_l)$ and sets $\mathsf{hk} = (\mathsf{pk}, h, 1_{\mathsf{ch}_1}, \ldots, 1_{\mathsf{ch}_{\alpha}}, c_1, \ldots, c_{\alpha})$ as public SSB hash key, where $h : \mathbb{Z}^*_{n^{w+1}} \rightarrow \mathbb{Z}_{n^w}$ is a collision resistant hash function.
- $\mathsf{SSBHash}.\mathcal{H}(\mathsf{hk}, s) \rightarrow (z = H_{\mathsf{hk}}(s))$: Let $s = (s[0], \ldots, s[L-1]) \in \Sigma^L$, where $\Sigma = \mathbb{Z}_{n^w}$. Let T be a binary tree of height α with L leaves. A user considers the leaves as being at level 0 and the root of the tree at level α. The user inductively and deterministically associates a value ct_v at each vertex $v \in T$ in bottom-up fashion as follows:
  - If $v \in T$ is the j-th leaf node (at level 0), $j \in \{0, \ldots, L-1\}$, then the user associates v the value $\mathsf{ct}_v = s[j] \in \mathbb{Z}_{n^w}$.
  - If $v \in T$ is a non-leaf node at level $l \in \{1, \ldots, \alpha\}$ with children v_0, v_1 having associated values $\mathsf{ct}_0, \mathsf{ct}_1$ respectively then the user associates v the value $\mathsf{ct}_v = h(c_v^*)$, where $c_v^* = [\mathsf{ct}_1 \otimes c_l] \oplus [\mathsf{ct}_0 \otimes (1_{\mathsf{ch}_l} \ominus c_l)] \in \mathbb{Z}_{n^w}$, $c_l, 1_{\mathsf{ch}_l}$ being the ciphertexts and h being the hash function extracted from $\mathsf{hk} = (\mathsf{pk}, h, 1_{\mathsf{ch}_1}, \ldots, 1_{\mathsf{ch}_{\alpha}}, c_1, \ldots, c_{\alpha})$ and $\otimes, \oplus, \ominus$ are operations as described in the Sect. 2.1. Note that c_v^* is the encryption of ct_{b_l}. The associated value at the root of T is the final output $z = H_{\mathsf{hk}}(s) \in \mathbb{Z}_{n^w}$.
- $\mathsf{SSBHash.Open}(\mathsf{hk}, s, j) \rightarrow (\pi)$: The user outputs ct_v values associated to siblings v of the nodes along the path form the root to the j-th leaf in T. In other words, if $\mathsf{PathNode}(j)$ denotes the set of nodes on the path from the root to the j-th leaf in T and $\mathsf{HangNode}(j)$ is the set of sibling nodes of all $v \in \mathsf{PathNode}(j)$, then $\pi = \{\mathsf{ct}_v | v \in \mathsf{HangNode}(j)\}$.
- $\mathsf{SSBHash.Verify}(\mathsf{hk}, z, j, u, \pi) \rightarrow (\mathsf{accept}, \mathsf{reject})$: A verifier can recompute the associated values of all the nodes in the tree T that lie on the path from the root to the j-th leaf by utilizing the value u as associated to the j-th leaf node together with the values in π as the associated values of all the sibling nodes along the path. The verifier checks whether the recomputed value at the root is indeed z. If it is z then the verifier outputs accept; otherwise, outputs reject.

2.3 Bloom Filter [2]

Bloom filter (BF) is a data structure that represents a set $X = \{x_1, \ldots, x_v\}$ of v elements by an array of m bits and uses k independent hash functions $H_{\mathsf{Bloom}} = \{h_0, \ldots, h_{k-1}\}$ with $h_i : \{0,1\}^* \to \{0, \ldots, m-1\}$ for $i = 0, \ldots, k-1$ to insert elements or check the presence of an element in that array. Let $\mathsf{BF}_X \in \{0,1\}^m$ represent a Bloom filter for the set X and $\mathsf{BF}_X[i]$ denotes its i-th bit, $i = 0, \ldots, m-1$. We describe below three operations that can be performed using Bloom filter:

- *Initialization:* Set 0 to all the bits of an m-bit array, which is an empty Bloom filter with no elements in it.
- *Add(x):* To add an element $x \in X \subseteq \{0,1\}^*$ into a Bloom filter, x is hashed with the k hash functions in $H_{\mathsf{Bloom}} = \{h_0, \ldots, h_{k-1}\}$ to get k indices $h_0(x), \ldots, h_{k-1}(x)$. Set 1 to the bit position of the Bloom filter having indices $h_0(x), \ldots, h_{k-1}(x)$. Repeat the process for each $x \in X$ to get $\mathsf{BF}_X \in \{0,1\}^m$ – the Bloom filter for the set X.
- *Check($\hat{x}$):* Given BF_X, to check whether an element $\hat{x}$ belongs to X without knowing X, $\hat{x}$ is hashed with the k hash functions in $H_{\mathsf{Bloom}} = \{h_0, \ldots, h_{k-1}\}$ to get k indices $h_0(\hat{x}), \ldots, h_{k-1}(\hat{x})$. Now if atleast one of $\mathsf{BF}_X[h_0(\hat{x})], \ldots, \mathsf{BF}_X[h_{k-1}(\hat{x})]$ is 0, then $\hat{x}$ is not in X, otherwise $\hat{x}$ is *probably* in X.

Bloom filter allows *false positive* whereby an element that has not been inserted in the filter can mistakenly pass the set membership test. This happens when an element $\hat{x}$ does not belong to X but $\mathsf{BF}_X[h_i(\hat{x})] = 1$ for all $i = 0, \ldots, k-1$. On the contrary, Bloom filter never yields a false negative i.e., an element that has been inserted in the filter will always pass the test. This is because if $\hat{x}$ belongs to X then each of $\mathsf{BF}_X[h_0(\hat{x})], \ldots, \mathsf{BF}_X[h_{k-1}(\hat{x})]$ is 1. Given the number v of elements to be added and a desired maximum false positive rate $\frac{1}{2^k}$, the optimal size m of the Bloom filter is $m = \frac{vk}{\ln 2}$.

2.4 Indistinguishability Obfuscation [1,7]

Definition 3. Indistinguishability Obfuscation (iO): *An indistinguishability obfuscator $\mathcal{O}$ for a circuit class $\mathcal{C}_\kappa$ is a PPT uniform algorithm satisfying the following requirements:*

- *(Correctness): For any circuit $C \in \mathcal{C}_\kappa$, if we compute $\overline{C} \leftarrow \mathcal{O}(1^\kappa, C)$ then $\overline{C}(x) = C(x)$ for all inputs x i.e., $Prob[\overline{C} \leftarrow \mathcal{O}(1^\kappa, C) : \overline{C}(x) = C(x)] = 1$ for all inputs x.*
- *(Indistinguishability): For any κ and any two circuits $C_0, C_1 \in \mathcal{C}_\kappa$, if $C_0(x) = C_1(x)$ for all inputs x then the circuits $\mathcal{O}(1^\kappa, C_0)$ and $\mathcal{O}(1^\kappa, C_1)$ are indistinguishable i.e., for all PPT adversaries $\mathcal{Z}$, $\big|Prob[\mathcal{Z}(\mathcal{O}(1^\kappa, C_0)) = 1] - Prob[\mathcal{Z}(\mathcal{O}(1^\kappa, C_1)) = 1]\big| \leq \epsilon(\kappa)$, where $\epsilon(\kappa)$ is negligible function of κ.*

We consider only polynomial-size circuits i.e., the circuit class C_κ consists of circuits of size at most κ. This circuit class is denoted by P/poly and the first candidate iO for this circuit class was introduced by Garg et al. [7]. Their construction is secure in generic matrix model. Following this, a single instance-independent assumption based iO for P/poly were proposed by [8,15].

3 Protocol

Protocol Requirements: The protocol computes the intersection of the server A's private input set $Y = \{y_1, \ldots, y_v\}$ and the client B's private input set $X = \{x_0, \ldots, x_{L-1}\}$. Without any loss of generality we may assume that $X, Y \subseteq \mathbb{Z}_{n^w}$. If not, we can choose a collision resistant hash function $\mathsf{ha} : \{0,1\}^* \rightarrow \mathbb{Z}_{n^w}$ to make the elements of X, Y as members of $\mathbb{Z}_{n^w}$. Auxiliary input includes the size L of B's input set, the security parameter κ, the Bloom filter parameters $(m, H_{\mathsf{Bloom}} = \{h_0, \ldots, h_{k-1}\})$. Without any loss of generality we can assume that $L = 2^\alpha$ for some integer $\alpha \leq \kappa$. If not, we can add 0's as the members of the set X to make its cardinality of the form 2^α. We integrate Bloom filter presented in Sect. 2.3, indistinguishability obfuscation (iO) scheme $\mathcal{O}$ described in Sect. 2.4 together with an SSB hash function $\mathsf{SSBHash}$ with alphabet $\Sigma = \mathbb{Z}_{n^w}$ i.e., $l_{\mathsf{blk}} = \lfloor w \log_2 n \rfloor$, output domain $\mathbb{Z}_{n^w}$ i.e., $l_{\mathsf{hash}} = \lfloor w \log_2 n \rfloor$ and opening domain $\times^\alpha(\mathbb{Z}_{n^w})$ i.e., $l_{\mathsf{opn}} = \alpha \lfloor w \log_2 n \rfloor$, where $n = pq$ is the product of two large primes p and q, and w is a positive integer. We require a circuit $C = C[\mathsf{hk}, z, \mathsf{ke}]$ as defined in Fig. 1. We also assume that C includes some polynomial-size padding to make it sufficiently large. Furthermore, we define an augmented circuit $C^{\mathsf{aug}} = C^{\mathsf{aug}}[\mathsf{hk}, z, \mathsf{ke}, \bar{k}, i^*]$ as in Fig. 2 which will be used in Sect. 3.1 for the security proof of our scheme. We need the padding in C to match its size with C^{aug}.

Construction: The protocol completes in two phases: off-line phase and online phase. In the off-line phase, the server A generates a SSB hash key hk and makes hk public. On the other hand, online phase consists of three algorithms: PSI.Request, PSI.Response and PSI.Complete. The client B runs PSI.Request algorithm to generate a SSB hash value of its input set X with the SSB hash key hk and sends it to A who in turn runs PSI.Response algorithm to generate an obfuscated circuit $\overline{C}$, a Bloom filter $BF_{\overline{Y}}$ and sends these to B. The client B then runs the algorithm PSI.Complete to get the intersection of X and Y.

Constraints: Hash key hk, hash value z, PRF key ke.
Input: $i \in \{0, ..., L-1\}, x \in \mathbb{Z}_{n^w}, \pi \in \times^\alpha(\mathbb{Z}_{n^w})$.
Output: 0 or PRF $f_{\mathsf{ke}}(x)$.
1. Check whether $\mathsf{SSBHash.Verify}(\mathsf{hk}, z, i, x, \pi)$ is accept or reject. If reject, then output 0.
2. Otherwise, output $f_{\mathsf{ke}}(x)$.

Fig. 1. Description of circuit $C[\mathsf{hk}, z, \mathsf{ke}](i, x, \pi)$

Constraints: Old values (hash key hk, hash value z, PRF key ke), New values (PRF key $\bar{k}, i^* \in \{0, ..., L-1\}$)
Input: $i \in \{0, ..., L-1\}, x \in \mathbb{Z}_{n^w}, \pi \in \times^{\alpha}(\mathbb{Z}_{n^w})$.
Output: 0 or PRF $f_{\mathsf{ke}}(x)$ or PRF $f_{\bar{k}}(x)$.
1. Check whether $\mathsf{SSBHash.Verify}(\mathsf{hk}, z, i, x, \pi)$ is accept or reject. If reject, then output 0.
2. Otherwise, if $i \geq i^*$, then output $f_{\mathsf{ke}}(x)$, else if $i < i^*$ output $f_{\bar{k}}(x)$.

Fig. 2. Description of circuit $C^{\mathsf{aug}}[\mathsf{hk}, z, \mathsf{ke}, \bar{k}, i^*](i, x, \pi)$

A high level description of our PSI protocol is presented in Fig. 3. We now describe below the off-line and online phases of our protocol.
Off-line Phase: On input 1^{κ}, the server A does the following:

(i) Runs the algorithm $\mathsf{SSBHash.Gen}$ on input $1^{\kappa}, 1^{l_{\mathsf{blk}}}, L = 2^{\alpha}, 0$ to generate a SSB hash key $\mathsf{hk} \leftarrow \mathsf{SSBHash.Gen}(1^{\kappa}, 1^{l_{\mathsf{blk}}}, L, 0)$, where $l_{\mathsf{blk}} = \lfloor w \log_2 n \rfloor$, where $\mathsf{hk} = (\mathsf{pk}, h, 1_{\mathsf{ch}_1}, \ldots, 1_{\mathsf{ch}_{\alpha}}, c_1, \ldots, c_{\alpha})$, $pk = (n, g, w)$ $h : \mathbb{Z}^*_{n^{w+1}} \rightarrow \mathbb{Z}_{n^w}$ is a collision resistant hash function, $1_{\mathsf{ch}_l} = \mathsf{DJ.Enc}(\mathsf{pk}, 1; R_l)$ and $c_l = \mathsf{DJ.Enc}(\mathsf{pk}, 0; \gamma_l)$.
(ii) Makes hk public.

Online Phase: It consists of the following three algorithms:

- $\underline{\mathsf{PSI.Request}(\mathsf{hk}) \rightarrow z}$: The client B proceeds as follows:
  (i) Sets $s[i] = x_i \in \Sigma = \mathbb{Z}_{n^w}$, for $i = 0, .., L-1$, where $X = \{x_0, \ldots, x_{L-1}\} \subseteq \mathbb{Z}_{n^w}$ is B's private input set.
  (ii) Computes $z = H_{\mathsf{hk}}(s) \leftarrow \mathsf{SSBHash}.\mathcal{H}(\mathsf{hk}, s)$. Note that $z \in \mathbb{Z}_{n^w}$.
  (iii) Finally, sends z to A.
- $\underline{\mathsf{PSI.Response}(z) \rightarrow (\overline{C}, \mathsf{BF}_{\overline{Y}})}$: The server A, on receiving the request z from B, does the following:
  (i) Chooses a PRF key $\mathsf{ke} \hookleftarrow \mathbb{Z}^*_n$ for PRF $f_{\mathsf{ke}}(x) = g^{1/(\mathsf{ke}+x)}$ if $gcd(\mathsf{ke}+x, n)$ is 1, 1 otherwise, where $x \in \{0,1\}^Q$, $\mathsf{ke} \in \mathbb{Z}^*_n$ and $Q = \lfloor w \log_2 n \rfloor$.
  (ii) Designs a circuit as described in Fig. 1.
  (iii) Constructs an obfuscated circuit $\overline{C} \leftarrow \mathcal{O}(1^{\kappa}, C)$ of C.
  (iv) Generates a Bloom filter $\mathsf{BF}_{\overline{Y}}$ of the set $\overline{Y} = \{f_{\mathsf{ke}}(y_1), \ldots, f_{\mathsf{ke}}(y_v)\}$, where $f_{\mathsf{ke}}(y_j) = g^{1/(\mathsf{ke}+y_j)}$ for $j = 1, \ldots, v$ and $Y = \{y_1, \ldots, y_v\} \subseteq \mathbb{Z}_{n^w}$ is A's private input set.
  (v) Sends the obfuscated circuit $\overline{C}$ together with $\mathsf{BF}_{\overline{Y}}$ to B.
- $\underline{\mathsf{PSI.Complete}(\overline{C}, \mathsf{BF}_{\overline{Y}}) \rightarrow (\overline{X} = X \cap Y)}$: On receiving $(\overline{C}, \mathsf{BF}_{\overline{Y}})$ from A, the client B starts with an empty set $\overline{X}$ and does the following
  (i) For each $i = 0, \ldots, L-1$
    – generates opening $\pi_i = \mathsf{SSBHash.Open}(\mathsf{hk}, s, i) \in \times^{\alpha}(\mathbb{Z}_{n^w})$ using the already computed values ct_v's during the calculation of $z \leftarrow \mathsf{SSBHash}.\mathcal{H}(\mathsf{hk}, s)$ and computes PRF values $f_{\mathsf{ke}}(x_i) \leftarrow \overline{C}(i, x_i, \pi_i)$. Note that s, x_i are known to B.

Common input: $\mathsf{hk} = (\mathsf{pk}, h, 1_{\mathsf{ch}_1}, ..., 1_{\mathsf{ch}_\alpha}, c_1, ..., c_\alpha)$
Auxiliary input: $L = 2^\alpha, \kappa, m, H_{\mathsf{Bloom}} = \{h_0, ..., h_{k-1}\}$

A's private input:		B's private input:
$Y = \{y_1, ..., y_w\}$		$X = \{x_0, ..., x_{L-1}\}$
		$z = H_{\mathsf{hk}}(s) \leftarrow \mathsf{PSI.Request}(\mathsf{hk})$, where $s = (s[0], ..., s[L-1])$, $s[i] = x_i$, for $i = 0, ..., L-1$
	$\xleftarrow{z}$	
$(\overline{C}, \mathsf{BF}_{\overline{Y}}) \leftarrow \mathsf{PSI.Response}(z)$, where $\overline{Y} = \{f_{\mathsf{ke}}(y_1), ..., f_{\mathsf{ke}}(y_v)\}$, $\mathsf{ke} \hookleftarrow \mathbb{Z}_n^*$, $\overline{C} \leftarrow \mathcal{O}(1^\kappa, C)$, $C = C[\mathsf{hk}, z, \mathsf{ke}](i, x, \pi)$		
	$\xrightarrow{\overline{C}, \mathsf{BF}_{\overline{Y}}}$	
		output $\overline{X} = X \cap Y \leftarrow \mathsf{PSI.Complete}(\overline{C}, \mathsf{BF}_{\overline{Y}})$, where $\overline{X} = \{x_i \in X \,\vert\, f_{\mathsf{ke}}(x_i) \in \mathsf{BF}_{\overline{Y}}\}$ with $f_{\mathsf{ke}}(x_i) \leftarrow \overline{C}(i, x_i, \pi_i)$, $\pi_i = \mathsf{SSBHash.Open}(\mathsf{hk}, s, i)$, for $i = 0, ..., L-1$

Fig. 3. Communication flow of our PSI

– checks whether $f_{\mathsf{ke}}(x_i)$ is in the set $\overline{Y}$ corresponding to the Bloom filter $\mathsf{BF}_{\overline{Y}}$. If yes, then x_i is included in $\overline{X}$.

(ii) Outputs the final $\overline{X}$ as the intersection of the sets X and Y.

Correctness: The correctness of our protocol follows from the following fact in the $\mathsf{PSI.Complete}$ phase executed by B:
$f_{\mathsf{ke}}(x_i)$ passes the check step of $\mathsf{BF}_{\overline{Y}} \Leftrightarrow f_{\mathsf{ke}}(x_i) \in \overline{Y}$ except with negligible probability $\frac{1}{2^k} \Leftrightarrow$ there exists $y_j \in Y$ such that $f_{\mathsf{ke}}(x_i) = f_{\mathsf{ke}}(y_j) \Leftrightarrow x_i = y_j$ as $f_{\mathsf{ke}}(\cdot)$ is a one-to-one $\Leftrightarrow x_i \in X \cap Y \Leftrightarrow \overline{X} = X \cap Y($ by the construction of $\overline{X})$.

Complexity: In our construction, size of the public parameter hk is $(2\alpha + 1)\lfloor w \log_2 n \rfloor + \log_2 n + |h|$ bit and 2α exponentiations are required to generate hk. The communication complexity includes three bit-strings of length m, $\lfloor w \log_2 n \rfloor$ and $m + poly(\kappa)(|C|)$, where $|h|$=length of the hash function $h : \mathbb{Z}_{n^{w+1}} \rightarrow \mathbb{Z}_{n^w}$ and $m = \frac{kv}{\ln 2}$, $|C|$=length of the circuit $C = C[\mathsf{hk}, z, \mathsf{ke}](i, x, \pi)$. The computation complexity of our PSI is displayed in Table 1.

Table 1. Computation complexity of our PSI protocol

		Exp	Inv	H_{BF}	H_{SSB}	EC	$\text{FHE}_{\mathsf{Enc}}$	$\text{FHE}_{\mathsf{Dec}}$	$\text{FHE}_{\mathsf{Eval}}$
A	$\mathsf{PSI.Response}$	v	v	kv		$2\eta(2M+5)^2 + 4(2M+5)$			
B	$\mathsf{PSI.Request}$	$3(L-1)$	$L-1$		$L-1$				
B	$\mathsf{PSI.Complete}$			kL			$2L$	L	$2L$

$\alpha = \log_2 L$, $M = 2\eta + 5$, η= length of oblivious matrix branching program, $\eta \leq 4^d$, d= depth of the circuit C, Exp= number of exponentiations, Inv= number of inversions, H_{BF}= number of hash operations for Bloom filter, H_{SSB}= number of hash operations for SSB hash

3.1 Security

Theorem 1. *If H is an SSB hash based on* DJ *encryption, $\mathcal{O}$ is an iO scheme and the associated PRF $f_{\mathsf{ke}}(\cdot)$ is secure then the protocol presented in Sect.* 3 *between a server A and a client B is a secure computation protocol in the semi-honest adversarial model* [5,9] *except with negligible probability $\frac{1}{2^k}$.*

Proof. Due to limited space, proof will appear in the full version.

4 Conclusion

In this work, we introduce the idea of constructing PSI utilizing *SSB hash*, *Bloom filter* and *iO*. Compared to the existing PSI schemes, our PSI is the most efficient PSI scheme. More significantly, it is the *first* to achieve *constant* communication complexity with linear computation cost. Our protocol works fast even for big data sets. Security of our scheme is analyzed in the semi-honest setting without any random oracles.

References

1. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K.: On the (im)possibility of obfuscating programs. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 1–18. Springer, Heidelberg (2001). doi:10.1007/3-540-44647-8_1
2. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7), 422–426 (1970)
3. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: Proceedings of 3rd Innovations in Theoretical Computer Science Conference, pp. 309–325. ACM (2012)
4. Damgård, I., Jurik, M.: A generalisation, a simpli.cation and some applications of Paillier's probabilistic public-key system. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 119–136. Springer, Heidelberg (2001). doi:10.1007/3-540-44586-2_9
5. Dong, C., Chen, L., Wen, Z.: When private set intersection meets big data: an efficient and scalable protocol. In: Proceedings of 2013 ACM SIGSAC Conference on Computer & Communications Security, pp. 789–800. ACM (2013)
6. Freedman, M.J., Hazay, C., Nissim, K., Pinkas, B.: Efficient set intersection with simulation-based security. J. Cryptol. **29**(1), 115–155 (2016)
7. Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate indistinguishability obfuscation and functional encryption for all circuits. In: 2013 IEEE 54th Annual Symposium on Foundations of Computer Science (FOCS), pp. 40–49. IEEE (2013)
8. Gentry, C., Lewko, A.B., Sahai, A., Waters, B.: Indistinguishability obfuscation from the multilinear subgroup elimination assumption. In: 2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS), pp. 151–170. IEEE (2015)
9. Goldreich, O.: Foundations of Cryptography: Volume 2, Basic Applications, vol. 2. Cambridge University Press, Cambridge (2009)

10. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. J. ACM (JACM) **33**(4), 792–807 (1986)
11. Hubacek, P., Wichs, D.: On the communication complexity of secure function evaluation with long output. In: Proceedings of 2015 Conference on Innovations in Theoretical Computer Science, pp. 163–172. ACM (2015)
12. Jarecki, S., Liu, X.: Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 577–594. Springer, Heidelberg (2009). doi:10.1007/978-3-642-00457-5_34
13. Okamoto, T., Pietrzak, K., Waters, B., Wichs, D.: New realizations of somewhere statistically binding hashing and positional accumulators. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015. LNCS, vol. 9452, pp. 121–145. Springer, Heidelberg (2015). doi:10.1007/978-3-662-48797-6_6
14. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999). doi:10.1007/3-540-48910-X_16
15. Pass, R., Seth, K., Telang, S.: Indistinguishability obfuscation from semantically-secure multilinear encodings. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8616, pp. 500–517. Springer, Heidelberg (2014). doi:10.1007/978-3-662-44371-2_28
16. 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)
17. Pinkas, B., Schneider, T., Zohner, M.: Faster private set intersection based on OT extension. In: USENIX Security, vol. 14, pp. 797–812 (2014)
18. Shi, R.-H., Mu, Y., Zhong, H., Cui, J., Zhang, S.: An efficient quantum scheme for private set intersection. Quantum Inf. Process. **15**(1), 363–371 (2016)