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

Sumit Kumar Debnath, Ratna Dutta
Department of Mathematics
Indian Institute of Technology Kharagpur
Kharagpur -721302, India
E-mails: {skdebnath, ratna}@maths.iitkgp.ernet.in

Abstract

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. This paper presents the *first* PSI protocol that achieves *constant* ($p(\kappa)$) communication complexity with linear computation overhead and is fast even for the case of large input sets, where $p(\kappa)$ is a polynomial in security parameter κ. 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 · semi-honest adversary · big data · SSB hash · iO · $Bloomfilter$

1 Introduction

In our everyday life, shairing of electronic information among mutually distrustful parties increases rapidly. Naturally, this raises important privacy concerns with respect to the disclosure and long-term safety of sensitive content. In this area, an interesting problem is to compute intersection of private data sets of two mutually dishonest entities. A few relevant applications are presented below:

1. Government tax authority wants to detect whether any suspected tax evaders have accounts with a certain foreign bank and, if so, obtain their account records. The bank's domicile forbids wholesale disclosure of account holders while the tax authority can not disclose its list of suspects.

A preliminary version of this paper appears in ICICS 2016.

2. Two real estate companies would like to identify customers (e.g., homeowners) who are double-dealing, i.e., have signed exclusive contracts with both companies to assist them in selling their properties.

3. A detective agency may want to verify whether a given biometric appears on a government watch-list. Note that privacy of biometric owner has to be preserved if no matches found, while at the same time, unrestricted access to the watch-list cannot be approved.

Private set intersection (PSI) can be utilized to solve the aforementioned problems. 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). On the other hand, if the participants wish to determine the cardinality rather than the intersection then that variant of PSI is known as Private set intersection cardinality (PSI-CA). Similar to PSI, it can be divided into two types: one-way PSI-CA and mutual PSI-CA (mPSI-CA). The first PSI-CA dates back to the work of Agrawal et al. [1]. Following it a number of PSI-CA were proposed in [8, 13, 23].

Related Works: Agrawal et al. [1] introduced the concept of PSI relying on commutative encryption and attains linear complexity. In [14], an *oblivious polynomial evaluations* (OPE) based PSI protocol was proposed by Freedman et al. They represents a set as polynomial in their construction. Later, Hazay and Nissim [22] improved the work of [14] in malicious model without random oracles. In malicious model, adversaries can run any efficient strategy in order to carry out their attack and can deviate at will from the prescribed protocol. On the other hand, in semi-honest model, adversaries follow the prescribed protocol, but adversaries try to gain more information than allowed from the protocol transcript. None of these constructions [14, 22] achieve linear computation complexity.

To improve the efficiency, Hazay and Lindell [21] employed the idea of *oblivious pseudo-random function* (OPRF) in the construction of PSI. Later, Jarecki and Liu [26] extended the work of [21] utilizing *additively homomorphic encryption* (AHE) [7]. Their scheme is secure in the standard model against malicious adversaries. All these constructions [21, 26] attain linear complexity.

De Cristofaro et al. presented a sequence of PSI protocols [9–11] retaining linear com-

Table 1 : Comparison of PSI protocols in semi-honest model

Protocol	Security model	Security assumption	Comm. cost	Comp. cost	Based on
[14]	Std	AHE	$O(L+v)$	$O(v \log\log L)$	OPE
Scheme 1 [12]	ROM	CDH	$O(L+v)$	$O(L+v)$	BF
[32]	ROM		$O(L+v)$	$O(L+v)$	BF
[31]	ROM		$O(L+v)$	$O(L+v)$	PH
[33]			$O(L)$	$O(L)$	QC
Our PSI	Std	DCR	$O(1)$	$O(L+v)$	SSB hash, BF, iO

BF= Bloom Filter, Std= Standard, DCR=Decisional Composite Residuosity, AHE= Additively Homomorphic Encryption, OPE= Oblivious Polynomial Evaluations, CDH=Computational Diffie-Hellman, iO=Indistinguishability Obfuscation, ROM= Random Oracle Model, PH= Permutation-based Hashing, QC= Quantum Computation, v, L are sizes of input sets.

plexity. The work of Huang et al. [24] showed how to employ garble circuit (GC) in designing

Table 2 : Comparison of PSI protocols in malicious model

Protocol	Security model	Security assumption	Comm. cost	Comp. cost	Based on
[26]	Std	Dq-DHI	$O(L+v)$	$O(L+v)$	OPRF
Scheme 2 [12]	ROM	CDH	$O(L+v)$	$O(L+v)$	BF
[20]	Std	d-strong DDH	$O(L+v)$	$O(L+v)$	
[13]	ROM	DDH,DCR	$O(L+v)$	$O(L+v)$	OPE

BF= Bloom Filter, OPRF= Oblivious Pseudorandom Function, DCR=Decisional Composite Residuosity, CDH=Computational Diffie-Hellman, Dq-DHI=Decisional q-Diffie-Hellman Inversion, ROM= Random Oracle Model, DDH=Decisional Diffie-Hellman, v, L are sizes of input sets.

PSI protocol. The scheme is secure under the Decisional Diffie-Hellman (DDH) assumption in the ROM against semi-honest adversaries and achieves linear communication and $\Theta(v \log v)$ as computational complexity. Here v is the cardinality of the private sets of both the parties. Dong et al. [12] came up with Bloom filter [4] based PSI protocols. A Bloom filter is a data structure that represents a set by an array with entries 0 or 1 and exhibits itself as an useful tool to scale large data sets. One of the constructions of [12] is secure against semi-honest adversaries, while the other one is secure against malicious adversaries under the Computational Diffie-Hellman (CDH) assumption. Following this work, Pinkas et al. [32] overviewed the existing solutions for set intersection in the semi-honest setting and presented a comparative summary of their efficiency. Later, Hazay [20] constructed an efficient PSI relying on algebraic PRF. This scheme is secure in the standard model against malicious parties. Recently, Freedman et al. [13] improved their earlier work of [14] and constructed two PSI protocols, one of which is secure in the semi-honest environment and the other one withstands attacks against malicious adversaries. More recently, Pinkas et al. [31] used permutation-based hashing technique to build very efficient PSI protocols secure in the ROM against semi-honest adversaries. See Table 1 and Table 2 for a comparative summary of PSI protocols in semi-honest model and malicious model respectively. Lastly, Shi et al. [33] employed quantum computation to construct a PSI which is proven to be secure in semi-honest model and achieves linear complexity.

Our Contribution: 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 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* [4] 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* [25, 28] along with *indistinguishability obfuscation (iO)* [2, 16]. Starting point of our construction is the approach of [28] 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 $p(\kappa)$ (polynomial in security parameter κ) i.e., to $O(1)$ which, unlike the existing PSI constructions, is independent of private input set sizes v and L of A and B respectively. 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 under the cryptographic assumption on which the corresponding SSB hash is secure. For simplicity, we employ the SSB hash [28] based on the Damgård-Jurik cryptosystem [7] 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, [12, 13, 31–33] are the most efficient PSI protocols, among which only [12, 32] 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, $a \leftarrow A$, $x \leftarrowtail X$, $[n]$ and $\{\mathcal{X}_t\}_{t\in\mathcal{N}} \stackrel{c}{\equiv} \{\mathcal{Y}_t\}_{t\in\mathcal{N}}$ are respectively used to represent "security parameter", "a is output of the procedure A", "variable x is chosen uniformly at random from set X", the set "{1,...,n}" and "the distribution ensemble $\{\mathcal{X}_t\}_{t\in\mathcal{N}}$ is computationally indistinguishable from the distribution ensemble $\{\mathcal{Y}_t\}_{t\in\mathcal{N}}$". Informally, $\{\mathcal{X}_t\}_{t\in\mathcal{N}} \stackrel{c}{\equiv} \{\mathcal{Y}_t\}_{t\in\mathcal{N}}$ 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 2.1. Pseudorandom Function [19]: *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 [26]:

$$f_k(x) = \begin{cases} g^{1/(k+x)} \text{ if } gcd(k+x, n) = 1 \\ 1 \qquad\quad \text{otherwise,} \end{cases}$$

where $x \in \{0,1\}^Q, k \in \mathbb{Z}_n^*$, where $Q = \lfloor w \log_2 n \rfloor$ for some positive integer w and $n = pq$ with p, q as primes. Here $\lfloor m \rfloor$ stands for the largest integer less than or equal m. The pseudorandom function is proven to be secure under the Decisional Q-Diffie-Hellman Inversion (DHI) Assumption [26].

Definition 2.2. Functionality: *A* functionality $\mathcal{F}_\Pi$*, computed by two parties A and B with inputs X_A and X_B respectively by running a protocol Π, is denoted by $\mathcal{F}_\Pi : (X_A, X_B) \rightarrow (Y_A, Y_B)$, where Y_A and Y_B are the outputs of A and B respectively on completion of the protocol Π between A and B.*

The functionality for PSI is denoted as $\mathcal{F}_{PSI} : (Y, X) \rightarrow (\perp, X \cap Y)$, where $\perp$ stands for "nothing". This essentially indicates that parties A and B with private input sets Y, X respectively engage in the protocol PSI. On completion of the protocol PSI, B receives $X \cap Y$ while A receives nothing.

2.1 Security Model for Semi-honest Adversary [18]

A two-party protocol, Π is a random process that computes a function $f = (f_1, f_2)$ from pair of inputs (x, y) (one per party) to a pair of outputs $(f_1(x, y), f_2(x, y))$. A protocol Π is said to be secure in semi-honest model if whatever cloud be computed by a party after participating in the protocol, it could be obtained from its input and output only. In other words, the parties follow the protocol honestly while the adversaries try to extract more information than allowed from the protocol transcript. This is formalized using the simulation paradigm. On the input pair (x, y), the view of the party P_i during an execution of Π, denoted by $\mathsf{View}_i^{\Pi}(x, y)$, is defined as $\mathsf{View}_i^{\Pi}(x, y) = (w, r^{(i)}, m_1^{(i)}, ..., m_t^{(i)})$, where $w \in \{x, y\}$ represents P_i's input, $r^{(i)}$ is the outcome of P_i's internal coin tosses, and $m_j^{(i)}$ $(j = 1, 2, ..., t)$ represents the j-th message received by P_i during the execution of Π.

Definition 2.3. Security in Semi-honest Model: *Let $f = (f_1, f_2) : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^*$ be a deterministic function. We say that the protocol Π securely computes f in semi-honest model if there exists PPT adversaries, denoted by S_1 and S_2, controlling parties P_1 and P_2 respectively, such that*

$$\{S_1(x, f_1(x, y))\}_{x,y\in\{0,1\}^*} \stackrel{c}{\equiv} \mathsf{View}_1^{\Pi}(x, y)_{x,y\in\{0,1\}^*},$$

$$\{S_2(y, f_2(x, y))\}_{x,y\in\{0,1\}^*} \stackrel{c}{\equiv} \mathsf{View}_2^{\Pi}(x, y)_{x,y\in\{0,1\}^*},$$

where $\{S_1(x, f_1(x, y))\}_{x,y\in\{0,1\}^*}$ and $\{S_2(y, f_2(x, y))\}_{x,y\in\{0,1\}^*}$ respectively denote the simulated views of P_1 and P_2 which contain input of the corresponding party, simulated random coins and simulated protocol messages received by the corresponding party.

2.2 Damgård-Jurik Cryptosystem [7]

The Damgård-Jurik cryptosystem DJ is a generalization of the Paillier cryptosystem [29] 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 $r \leftarrow\!\!\!- \mathbb{Z}^*_{n^{w+1}}$ 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 $c^d \bmod n^{w+1}$. If c is valid then

$$c^d = (g^m r^{n^w})^d = ((1+n)^{j \cdot m} x^m r^{n^w})^d = (1+n)^{(j \cdot m \cdot d) \bmod n^w} (x^m r^{n^w})^{d \bmod \gamma}$$
$$= (1+n)^{(j \cdot m \cdot d) \bmod n^w}$$

The decryptor then applies Algorithm 1 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 $m = (j \cdot m \cdot d)(j \cdot d)^{-1} \bmod n^w$.

Algorithm 1 Recursive version of the Paillier decryption mechanism

Input: $a = (1+n)^R \bmod n^{w+1}$.
Output: $R \bmod n^w$.
$i := 0$;
for $j := 1$ to w **do**
    $t_1 := L(a \bmod n^{j+1}) = \frac{a-1}{n} \bmod n^j$;
    $t_2 := i$;
    for $k := 2$ to j **do**
        $i := i - 1$;
        $t_2 := t_2 \cdot i \bmod n^j$;
        $t_1 := t_1 - \frac{t_2 \cdot n^{k-1}}{k!} \bmod n^j$;
    end do
    $i := t_1$;
end do
output i.

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 $\mathsf{DJ.Enc_{pk}}(m_1; r_1) \ominus \mathsf{DJ.Enc_{pk}}(m_2; r_2) = \mathsf{DJ.Enc_{pk}}(m_1 - m_2; r_4)$ 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 $\mathsf{DJ.Enc_{pk}}(m_1; r_1) \otimes m_2 = \oplus^{m_2} \mathsf{DJ.Enc_{pk}}(m_1; r_1) = \mathsf{DJ.Enc_{pk}}(m_1 \cdot m_2; r_5)$ 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 [29] assumption defined below:

Definition 2.4. Decisional Composite Residuosity (DCR) Assumption [29]: *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 $z \equiv y^n \pmod{n^2}$.

Correctness: The correctness of the Algorithm 1 follows from the following fact:

(i) For $j = 1$, $t_1 = R \bmod n$ after execution of line 5, $t_2 = 0$ after execution of line 6 and $i = R \bmod n$ after execution of line 12.

(ii) After execution of line 5, $t_1 = \binom{R}{1} + \binom{R}{2}n + ... + \binom{R}{j}n^{j-1} \bmod n^j$, $j \in \{2, ..., w\}$.

(iii) After execution of line 6, $t_2 = R \bmod n^{j-1}$, $j \in \{2, ..., w\}$.

(iv) After execution of lines 7–11,

$$\begin{aligned} t_1 &= \binom{R}{1} + \binom{R}{2}n + ... + \binom{R}{j}n^{j-1} - \sum_{k=2}^{j} \frac{R(R-1)...(R-k+1)n^{k-1}}{k!} \bmod n^j \\ &= \binom{R}{1} + \binom{R}{2}n + ... + \binom{R}{j}n^{j-1} - \sum_{k=2}^{j} \binom{R}{k}n^{k-1} \bmod n^j \\ &= R \bmod n^j,\ j \in \{2, ..., w\}. \end{aligned}$$

(v) After execution of line 12, $i = R \bmod n^j,\ j \in \{2, ..., w\}$.

(vi) Finally, after execution of line 13, i.e. for $j = w$, the algorithm returns $i = R \bmod n^w$.

2.3 SSB Hash [25, 28]

Definition 2.5. SSB Hash: *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*. From next, we will use* $l_{\mathsf{blk}}, l_{\mathsf{hash}}, l_{\mathsf{opn}}$ *instead of* $l_{\mathsf{blk}}(\kappa), l_{\mathsf{hash}}(\kappa), l_{\mathsf{opn}}(\kappa)$*. The algorithms work as follows:*

- $\mathsf{SSBHash.Gen}(1^\kappa, 1^{l_{\mathsf{blk}}}, L, i) \rightarrow (hk)$*. Setup authority runs this algorithm which takes as input a security parameter* κ*, a block length* l_{blk}*, an input length* $L \leq 2^\kappa$ *and an index* $i \in \{0, ..., L-1\}$*, and outputs a public hashing key* hk*.*
- $\mathsf{SSBHash}.\mathcal{H}(\mathsf{hk}, s) \rightarrow (H_{\mathsf{hk}}(s))$*. It is a deterministic polynomial time algorithm run by a user. This algorithm takes as input* $s = (s[0], ..., s[L-1]) \in \Sigma^L$ *and a hash key* hk*, and outputs* $H_{\mathsf{hk}}(s) \in \{0,1\}^{l_{\mathsf{hash}}(\kappa)}$*, where* $l_{\mathsf{hash}}(\kappa)$ *is independent of input length* L*.*
- $\mathsf{SSBHash.Open}(\mathsf{hk}, s, j) \rightarrow (\pi)$*. This algorithm is run by a user. It takes as input a hash key* hk*,* $s = (s[0], ..., s[L-1]) \in \Sigma^L$ *and an index* $j \in \{0, ..., L-1\}$ *and returns an opening* $\pi \in \{0,1\}^{l_{\mathsf{opn}}}$*.*

- $\mathsf{SSBHash.Verify}(\mathsf{hk}, z, j, u, \pi) \rightarrow (\mathsf{accept}, \mathsf{reject})$. *Verifier runs this algorithm on input a hash key hk, hash value $z \in \{0,1\}^{l_{\mathsf{hash}}}$, an integer $j \in \{0, ..., L-1\}$, a value $u \in \Sigma$ and an opening $\pi \in \{0,1\}^{l_{\mathsf{opn}}}$, and outputs a decision $\in \{\mathsf{accept}, \mathsf{reject}\}$. In other words, this verification determines whether a pre-image s of $z = H_{\mathsf{hk}}(s)$ has $s[j] = u$.*

An SSB hash satisfies the following three properties:

– *Correctness*: For any integer $L \leq 2^\kappa$ and $i, j \in \{0, ..., L-1\}$, any $hk \leftarrow \mathsf{SSBHash.Gen}(1^\kappa, 1^{l_{\mathsf{blk}}}, L, i)$, $s = (s[0], ..., s[L-1]) \in \Sigma^L, \pi \leftarrow \mathsf{SSBHash.Open}(\mathsf{hk}, s, j)$, corresponding Verify algorithm should return $accept$ as output i.e., $\mathsf{SSBHash.Verify}(\mathsf{hk}, H_{\mathsf{hk}}(s), j, s[j], \pi) = accept$.

– *Index Hiding*: Given 1^κ, an integer $L \leq 2^\kappa$, two indices $i_0, i_1 \in \{0, ..., L-1\}$ and a hash key $hk \leftarrow \mathsf{SSBHash.Gen}(1^\kappa, 1^{l_{\mathsf{blk}}}, L, i_b)$, the probability of generating correct index i_b from any PPT attacker $\mathcal{A}$ is negligible i.e., if b' is the output for any PPT attacker $\mathcal{A}$ then we have $|Prob[b = b']| \leq \epsilon(\kappa)$, where $\epsilon(\kappa)$ is negligible function of κ.

– *Somewhere Statistically Binding*: We say that hk is statistically binding for an index $j \in \{0, ..., L-1\}$, if there do not exist any values $z \in \{0,1\}^{l_{\mathsf{hash}}}, s, s' \in \Sigma^L, \pi \leftarrow \mathsf{SSBHash.Open}(\mathsf{hk}, s, j)$, $\pi' \leftarrow \mathsf{SSBHash.Open}(\mathsf{hk}, s', j)$ with $s[j] \neq s'[j]$ such that $\mathsf{SSBHash.Verify}(\mathsf{hk}, z, j, s[j], \pi) = \mathsf{SSBHash.Verify}(\mathsf{hk}, z, j, s'[j], \pi')$, where $L \leq 2^\kappa$ is an integer, $hk \leftarrow \mathsf{SSBHash.Gen}(1^\kappa, 1^{l_{\mathsf{blk}}}, L, i)$ and $s[j], s'[j]$ are the j-th blocks of s, s' respectively.

Example: We describe below the DCR based SSB hash of [28] which uses Damgård-Jurik cryptosystem as described in section 2.2 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 $\times^\alpha \mathbb{Z}_{n^w}$ 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 $(b_\alpha, ..., b_1)$ be the binary representation of the index $i \in \{0, ..., L-1\}$. For $l = 1, ..., \alpha$, the setup authority computes $g^{b_l} \gamma_l^{n^w} = c_l = \mathsf{DJ.Enc}(\mathsf{pk}, b_l; \gamma_l)$, $g R_l^{n^w} = 1_{\mathsf{ch}_l} = \mathsf{DJ.Enc}(\mathsf{pk}, 1; R_l)$ and sets $\mathsf{hk} = (\mathsf{pk}, h, 1_{\mathsf{ch}_1}, ..., 1_{\mathsf{ch}_\alpha}, c_1, ..., c_\alpha)$ 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 $s = (s[0], ..., s[L-1]) \in \Sigma^L$, 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), $j \in \{0, ..., L-1\}$, 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 $l \in \{1, ..., \alpha\}$ 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 $\mathsf{hk} = (\mathsf{pk}, h, 1_{\mathsf{ch}_1}, ..., 1_{\mathsf{ch}_\alpha}, c_1, ..., c_\alpha)$ and $\otimes, \oplus, \ominus$

are operations as described in the section 2.2. Note that c_v^* is the encryption of ct_{b_l} as

$$\begin{aligned}
c_v^* &= [\mathsf{ct}_1 \otimes c_l] \oplus [\mathsf{ct}_0 \otimes (1_{\mathsf{ch}_l} \ominus c_l)] \\
&= [\mathsf{ct}_1 \otimes \mathsf{DJ.Enc}_(\mathsf{pk}, b_l; \gamma_l)] \oplus [\mathsf{ct}_0 \otimes \left\{\mathsf{DJ.Enc}_(\mathsf{pk}, 1; R_l) \ominus \mathsf{DJ.Enc}(\mathsf{pk}, b_l; \gamma_l)\right\}] \\
&= \mathsf{DJ.Enc}(\mathsf{pk}, b_l\mathsf{ct}_1; (\gamma_l)^{\mathsf{ct}_1}) \oplus [\mathsf{ct}_0 \otimes \mathsf{DJ.Enc}(\mathsf{pk}, 1 - b_l; \frac{R_l}{\gamma_l})] \\
&= \mathsf{DJ.Enc}(\mathsf{pk}, b_l\mathsf{ct}_1; (\gamma_l)^{\mathsf{ct}_1}) \oplus \mathsf{DJ.Enc}(\mathsf{pk}, (1 - b_l)\mathsf{ct}_0; (\frac{R_l}{\gamma_l})^{\mathsf{ct}_0}) \\
&= \mathsf{DJ.Enc}(\mathsf{pk}, b_l\mathsf{ct}_1 + (1 - b_l)\mathsf{ct}_0; (\gamma_l)^{\mathsf{ct}_1}(\frac{R_l}{\gamma_l})^{\mathsf{ct}_0}) \\
&= \begin{cases} \mathsf{DJ.Enc}(\mathsf{pk}, \mathsf{ct}_0; (\gamma_l)^{\mathsf{ct}_1}(\frac{R_l}{\gamma_l})^{\mathsf{ct}_0}) \text{ if } b_l = 0 \\ \mathsf{DJ.Enc}(\mathsf{pk}, \mathsf{ct}_1; (\gamma_l)^{\mathsf{ct}_1}(\frac{R_l}{\gamma_l})^{\mathsf{ct}_0}) \text{ if } b_l = 1 \end{cases} \\
&= \mathsf{DJ.Enc}(\mathsf{pk}, \mathsf{ct}_l)
\end{aligned}$$

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 $\pi = \left\{\mathsf{ct}_v \middle| v \in \mathsf{HangNode}(j)\right\}$.
- $\mathsf{SSBHash.Verify}(\mathsf{hk}, z, j, u, \pi) \rightarrow (\mathsf{accept}, \mathsf{reject})$: 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.

Complexity: Complexity: The algorithm $\mathsf{SSBHash.Gen}$ requires 2α exponentiations (Exp) under modulo n^{w+1} to generate the ciphertexts $\{c_1, ..., c_\alpha\}$, $\{1_{\mathsf{ch}_1}, ..., 1_{\mathsf{ch}_\alpha}\}$, whereas the algorithm $\mathsf{SSBHash}.\mathcal{H}$ incurs $3(2^\alpha - 1)$ Exp and $2^\alpha - 1$ inversions (Inv) under modulo n^{w+1} together with $2^\alpha - 1$ hash operations. Finally, the algorithm $\mathsf{SSBHash.Verify}$ requires 3α Exp and α under modulo n^{w+1} along with α hash operations.

2.4 Bloom Filter [4]

Bloom filter (BF) is a data structure that represents a set $X = \{x_1, ..., x_v\}$ of v elements by an array of m bits and uses k independent hash functions $H_{\mathsf{Bloom}} = \{h_0, ..., h_{k-1}\}$ with $h_i : \{0,1\}^* \rightarrow \{0, ..., m-1\}$ for $i = 0, ..., k-1$ to insert elements or check the presence of elements in that array. Let $\mathsf{BF}_X \in \{0,1\}^m$ represents a Bloom filter for the set X and $\mathsf{BF}_X[i]$ denotes its i-th bit, $i = 0, ..., m-1$. Three operations that can be performed using Bloom filter are – Initialization, Add and Check. We describe below how these operations are implemented using Bloom filter [4].

– *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 $H_{\mathsf{Bloom}} = \{h_0, ..., h_{k-1}\}$ to get k indices $h_0(x), ..., h_{k-1}(x)$. Set 1 to the bit position of the Bloom filter having indices $h_0(x), ..., h_{k-1}(x)$. Repeat the process for each $x \in X$ to get $\mathsf{BF}_X \in \{0,1\}^m$ – the Bloom filter for the set X. See Algorithm 2.

Algorithm 2 Construction of Bloom filter

Input: X, an m-bit empty array BF_X, $H_{\mathsf{Bloom}} = \{h_0, ..., h_{k-1}\}$.
Output: Bloom filter $\mathsf{BF}_X \in \{0,1\}^m$ of X.
for $x \in X$ **do**
  for $i = 0$ to $k-1$ **do**
    $\mathsf{BF}_X[h_i(x)] \leftarrow 1$;
  end do
end do

– *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 $H_{\mathsf{Bloom}} = \{h_0, ..., h_{k-1}\}$ to get k indices $h_0(\hat{x}), ..., h_{k-1}(\hat{x})$. Now if atleast one of $\mathsf{BF}_X[h_0(\hat{x})]$, ..., $\mathsf{BF}_X[h_{k-1}(\hat{x})]$ is 0, then $\hat{x}$ is not in X, otherwise $\hat{x}$ is *probably* in X. See Algorithm 3.

Algorithm 3 Membership test

Input: $\hat{x}$, BF_X, $H_{\mathsf{Bloom}} = \{h_0, ..., h_{k-1}\}$.
Output: YES/NO.
for $i = 0$ to $k-1$ **do**
  if $\mathsf{BF}_X[h_i(\hat{x})] = 0$;
    output NO// $\hat{x}$ does not belong to the set X corresponding to BF_X;
  end if
end do
output YES// $\hat{x}$ probably belongs to the set X corresponding to BF_X;

Bloom filter allows *false positive* whereby an element that has not been inserted in the filter can mistakenly pass the set membership test described in Algorithm 3. This happens when an element $\hat{x}$ does not belong to X but $\mathsf{BF}_X[h_i(\hat{x})] = 1$ for all $i = 0, ..., k-1$. On the contrary, Bloom filter never yields 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 $\mathsf{BF}_X[h_0(\hat{x})], ..., \mathsf{BF}_X[h_{k-1}(\hat{x})]$ is 1.

Theorem 2.6. *[12] 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 $m = \frac{vk}{\ln 2}$.*

Complexity: Algorithms 2 and 3 incurs respectively $k|X|$ and k many hash evaluations.

2.5 Indistinguishability Obfuscation [2, 16]

Definition 2.7. 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.,*

$$Prob[\overline{C} \leftarrow \mathcal{O}(1^\kappa, C) : \overline{C}(x) = C(x)] = 1 \text{ for all inputs } x.$$

– *(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 $\mathcal{Z}$, $\left|Prob[\mathcal{Z}(\mathcal{O}(1^\kappa, C_0)) = 1] - Prob[\mathcal{Z}(\mathcal{O}(1^\kappa, C_1)) = 1]\right| \leq \epsilon(\kappa)$, 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. [16]. Their construction is secure in generic matrix model. Following this, a single instance-independent assumption based iO for P/poly were proposed by [17, 30].

Example(The Candidate iO Construction of [16]): An iO for P/poly is constructed by designing an iO construction for a restricted circuit class, namely NC^1 which is then coupled with (leveled) fully homomorphic encryption (FHE) [5] with decryption in NC^1 utilizing "two-key" encryption technique of [27]. NC^1 is the family of polynomial size circuits with logarithmic depth and bounded fan-in.

An NC^1 circuit C with input length τ and depth d is first transformed to oblivious matrix branching program $\mathsf{MBP} = \{(\mathsf{input}(i), P_{i,0}, P_{i,1})\}_{i=1}^{\eta}$. This transformation is possible due to Barrington's theorem [3]. Here the function $\mathsf{input} : [\eta] \to [\tau]$ describes the input bit examined in the i-th step and $P_{i,b}$'s are permutation matrices of order 5 for $b \in \{0, 1\}$. To evaluate MBP, on any particular τ-bit input $\bar{\alpha} = (\alpha_1\alpha_2...\alpha_\tau)$, compute the product matrix $P = \prod_{i=1}^{\eta} P_{i,\bar{\alpha}_{\mathsf{input}(i)}}$ and output 1 if P is the identity matrix and 0 otherwise. Let $M = 2\eta + 5$. The obfuscation procedure attempts to garble the branching program MBP of the circuit C using multilinear jigsaw puzzles as introduced in [15] which is restricted version of multilinear maps [6, 16] as follows:

1. Generate a ring $\mathbb{Z}_\rho$ with prime $\rho \geq 2^\kappa$, the public parameters par, and a secret state ϕ to pass to the encoding algorithm of the generator by running the instance-generator InstGen of the multilinear jigsaw generator. Note that the multilinearity level of the jigsaw puzzle is considered as $[\eta + 2]$.

2. Sample independent scalars $\{\mu_{i,0}, \mu_{i,1}, \mu'_{i,0}, \mu'_{i,1}\}$ randomly from the ring $\mathbb{Z}_\rho$, subject to the constraint that $\prod_{i \in I_j} \mu_{i,0} = \prod_{i \in I_j} \mu'_{i,0}$ and $\prod_{i \in I_j} \mu_{i,1} = \prod_{i \in I_j} \mu'_{i,1}$ for all $j \in [\tau]$, where $I_j = \{i \in [\eta] : \mathsf{input}(i) = j\}$.

3. Compute two pairs block-diagonal matrices $\{\mathsf{BD}_{i,b}\}_{b\in\{0,1\}}, \{\mathsf{BD}'_{i,b}\}_{b\in\{0,1\}}$ of order $2M+5$ each for every $i \in [\eta]$, where the diagonal entries $1, ..., 2M$ are chosen at random and the bottom-right 5×5 are scaled $P_{i,b}$'s and identity matrix respectively:

$$\mathsf{BD}_{i,b} \sim \begin{bmatrix} \$ & & & & & \\ & . & & & & \\ & & . & & & \\ & & & . & & \\ & & & & \$ & \\ & & & & & \mu_{i,b}A_{i,b} \end{bmatrix}, \mathsf{BD}'_{i,b} \sim \begin{bmatrix} \$ & & & & & \\ & . & & & & \\ & & . & & & \\ & & & . & & \\ & & & & \$ & \\ & & & & & \mu'_{i,b}I \end{bmatrix},$$

where $b \in \{0, 1\}$, the \$'s on main diagonal are random coefficients that are unique to each matrix and the unspecified entries are 0.

4. Choose vectors $\mathbf{s_1}$ and $\mathbf{t_1}$, and $\mathbf{s_2}$ and $\mathbf{t_2}$ of dimension $2M + 5$ as follows:
   - The entries $1, ..., M$ in $\mathbf{s_1}$ and $\mathbf{s_2}$ vectors are set to 0 and the entries $M + 1, ..., 2M$ are randomly chosen. While the entries $1, ..., M$ in $\mathbf{t_1}$ and and $\mathbf{t_2}$ are selected at random and $M + 1, ..., 2M$ are set to 0.
   - Choose two pairs of random vectors $\{\mathbf{s}^*_\mathbf{1}, \mathbf{t}^*_\mathbf{1}\}$ and $\{\mathbf{s}^*_\mathbf{2}, \mathbf{t}^*_\mathbf{2}\}$ of length 5 such that $\langle \mathbf{s}^*_\mathbf{1}, \mathbf{t}^*_\mathbf{1}\rangle$=$\langle \mathbf{s}^*_\mathbf{2}, \mathbf{t}^*_\mathbf{2}\rangle$, where the notation $\langle , \rangle$ stands for inner product. The last 5 entries in $\mathbf{s_1}, \mathbf{t_1}, \mathbf{s_2}, \mathbf{t_2}$ are set to the entries of $\mathbf{s}^*_\mathbf{1}, \mathbf{t}^*_\mathbf{1}, \mathbf{s}^*_\mathbf{2}, \mathbf{t}^*_\mathbf{2}$ respectively.

5. Sample $2(\eta+1)$ random full-rank matrices $R_0, R_1, ...R_\eta$ and $R'_0, R'_1, ...R'_\eta$ of order $2M+5$ each over the ring $\mathbb{Z}_\rho$ and compute their inverses.

6. Randomize the branching program MBP over the ring $\mathbb{Z}_\rho$ as follows:

   $\mathsf{RND}_\rho(\mathsf{MBP}) = (\mathsf{PROG}_1, \mathsf{PROG}_2)$, where

$$\mathsf{PROG}_1 = \begin{bmatrix} \tilde{\mathbf{s_1}} = \mathbf{s_1}R_0^{-1}, \tilde{\mathbf{t_1}} = R_\eta \mathbf{t_1}^T, \\ \{\widetilde{\mathsf{BD}}_{i,b} = R_{i-1}\mathsf{BD}_{i,b}R_i^{-1}\}_{i\in[\eta],b\in\{0,1\}} \end{bmatrix}$$

$$\text{and } \mathsf{PROG}_2 = \begin{bmatrix} \tilde{\mathbf{s_2}} = \mathbf{s_2}R_0^{-1}, \tilde{\mathbf{t_2}} = R_\eta \mathbf{t_2}^T, \\ \{\widetilde{\mathsf{BD}}'_{i,b} = R'_{i-1}\mathsf{BD}'_{i,b}R'^{-1}_i : i \in [\eta], b \in \{0, 1\}\} \end{bmatrix}$$

   Note that this randomize program $\mathsf{RND}_\rho(\mathsf{MBP})$ consists of two parallel programs – PROG_1 and PROG_2. The original branching program MBP is embedded with all $P_{i,b}$'s in PROG_1, while a "dummy program" of the same length is embedded in PROG_2 consisting only of identity matrices, thereby computing the constant function 1. Here $\mathbf{v^T}$ stands for transpose of the vector $\mathbf{v}$. The dummy program is used to test equality: the original program outputs 1 on a given input only when it agrees with the dummy program on that input.

7. Encode each element of the step-i matrices $\widetilde{\mathsf{BD}}_{\mathbf{i,b}}, \widetilde{\mathsf{BD}}'_{\mathbf{i,b}}$ relative to the singleton index-set $\{i+1\}$, each element of the vectors $\tilde{\mathbf{s_1}}, \tilde{\mathbf{s_2}}$ relative to singleton index-set $\{1\}$, and each element of the vectors $\tilde{\mathbf{t_1}}, \tilde{\mathbf{t_1}}$ relative to singleton index-set $\{\eta + 2\}$ using the encoding algorithm of the multilinear jigsaw generator.

8. The public parameters par of the jigsaw generator together with all the encoded matrices and vectors constitutes the obfuscated program.

For any input $\bar{\alpha} = (\alpha_1\alpha_2...\alpha_\tau)$ to the original program, the evaluator of the corresponding obfuscated program computes encoding of $\tilde{\mathbf{s}}_\mathbf{1} \prod_{i\in[\eta]} \widetilde{\mathsf{BD}}_{i,\alpha_{\mathsf{input}(i)}} \tilde{\mathbf{t}}_\mathbf{1} - \tilde{\mathbf{s}}_\mathbf{2} \prod_{i\in[\eta]} \widetilde{\mathsf{BD}}'_{i,\alpha_{\mathsf{input}(i)}} \tilde{\mathbf{t}}_\mathbf{2}$ relative to the index-set $[\eta+2]$ using only the allowed multilinear operations and runs the zero-testing algorithm of the jigsaw puzzle. If the zero-test passes, then the evaluator outputs 1, otherwise, it outputs 0.

This construction is proven to be an iO for NC^1 circuits under a new complexity assumption which holds in a generic matrix model in [16]. This iO for NC^1 circuits together with FHE is used to obfuscate a circuit $C \in P/\mathsf{poly}$ as follows:

(a) Generate two public key/secret key pairs $(\mathsf{pk}^{(1)}_{\mathsf{FHE}}, \mathsf{sk}^{(1)}_{\mathsf{FHE}})$ and $(\mathsf{pk}^{(2)}_{\mathsf{FHE}}, \mathsf{sk}^{(2)}_{\mathsf{FHE}})$ of the FHE scheme and publish the public keys $\mathsf{pk}^{(1)}_{\mathsf{FHE}}$ and $\mathsf{pk}^{(2)}_{\mathsf{FHE}}$.

(b) Generate the ciphertexts $\mathsf{chi}^{(1)}$ and $\mathsf{chi}^{(2)}$ by encrypting the circuit $C \in P/\mathsf{poly}$ under $\mathsf{pk}^{(1)}_{\mathsf{FHE}}$ and $\mathsf{pk}^{(2)}_{\mathsf{FHE}}$ respectively.

(c) Obfuscate NC^1 circuit C_{Dec_0} for verifying the proof generated by the obfuscated circuit evaluator.

The obfuscated circuit evaluator holding an input $\bar{\alpha}$ does the following:

– Generates ciphertexts $c^{(1)}_{\bar{\alpha}}$ and $c^{(2)}_{\bar{\alpha}}$ by encrypting $\bar{\alpha}$ using $\mathsf{pk}^{(1)}_{\mathsf{FHE}}$ and $\mathsf{pk}^{(2)}_{\mathsf{FHE}}$ respectively.

– Produces encryptions e_1 and e_2 of the output $C(\bar{\alpha})$ under $\mathsf{pk}^{(1)}_{\mathsf{FHE}}$ and $\mathsf{pk}^{(2)}_{\mathsf{FHE}}$ respectively using the FHE evaluation algorithm with $c^{(1)}_{\bar{\alpha}}$, $c^{(2)}_{\bar{\alpha}}$, $\mathsf{chi}^{(1)}$ and $\mathsf{chi}^{(2)}$.

– Keeps track of all the intermediate bit values encountered during evaluation of e_1 and e_2 as a "proof" π of the fact that $\bar{\alpha}$ is used to perform the evaluation correctly on both $\mathsf{chi}^{(1)}$ and $\mathsf{chi}^{(2)}$.

– Feeds $(e_1, e_2, \bar{\alpha}, \pi)$ into the obfuscated form of the circuit C_{Dec_0} which in turns checks the proof π to make sure that e_1 and e_2 were correctly computed. If the proof checks out, then the circuit decrypts e_1 using secret key $\mathsf{sk}^{(1)}_{\mathsf{FHE}}$ and outputs this decrypted value which should be $C(\bar{\alpha})$.

There is another NC^1 circuit C_{Dec_1} that is equivalent to C_{Dec_0}, which directly decrypts e_2 using $\mathsf{sk}^{(2)}_{\mathsf{FHE}}$ instead. Both the circuits behave identically on all inputs due to the proof π that must be provided. As $\mathsf{sk}^{(2)}_{\mathsf{FHE}}$ is never used anywhere when using C_{Dec_0}, the semantic security of the FHE scheme using $\mathsf{pk}^{(2)}_{\mathsf{FHE}}$ is maintained even given C_{Dec_0}. Alternatively applying the semantic security of the FHE scheme and switching back and forth between C_{Dec_0} and C_{Dec_1} using the iO property, the above obfuscator is proven to be an iO for P/poly in [16].

Complexity: To obfuscate a circuit total number of required encodings is $2(2M+5)^2+4(2M+5)$. On the other hand, the circuit evaluation cost for an input $\bar{\alpha}$ incurs 2 FHE encryptions, 2 FHE evaluations and 1 FHE decryption.

3 Protocol

Protocol Requirements: The protocol computes the intersection of the server A's private input set $Y = \{y_1, ..., y_v\}$ and the client B's private input set $X = \{x_0, ..., x_{L-1}\}$. 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 $\mathsf{ha} : \{0,1\}^* \to \mathbb{Z}_{n^w}$ 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 $(m, H_{\mathsf{Bloom}} = \{h_0, ..., h_{k-1}\})$. 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 section 2.4, indistinguishability obfuscation

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 $f_{\mathsf{ke}}(x)$.

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

(iO) scheme $\mathcal{O}$ described in section 2.5 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 Figure 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 Figure 2

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 $f_{\bar{k}}(i)$.
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 $f_{\bar{k}}(i)$.

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

which we will use in section 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: $\mathsf{PSI.Request}$, $\mathsf{PSI.Response}$ and $\mathsf{PSI.Complete}$. The client B runs $\mathsf{PSI.Request}$ 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 $\mathsf{PSI.Response}$ algorithm to generate an obfuscated circuit $\bar{C}$, a Bloom filter BF_Y and sends these to B. The client B then runs the

algorithm PSI.Complete to get the intersection of X and Y. A high level description of the functionality $\mathcal{F}_{PSI}$ of our PSI protocol is presented in Figure 3.

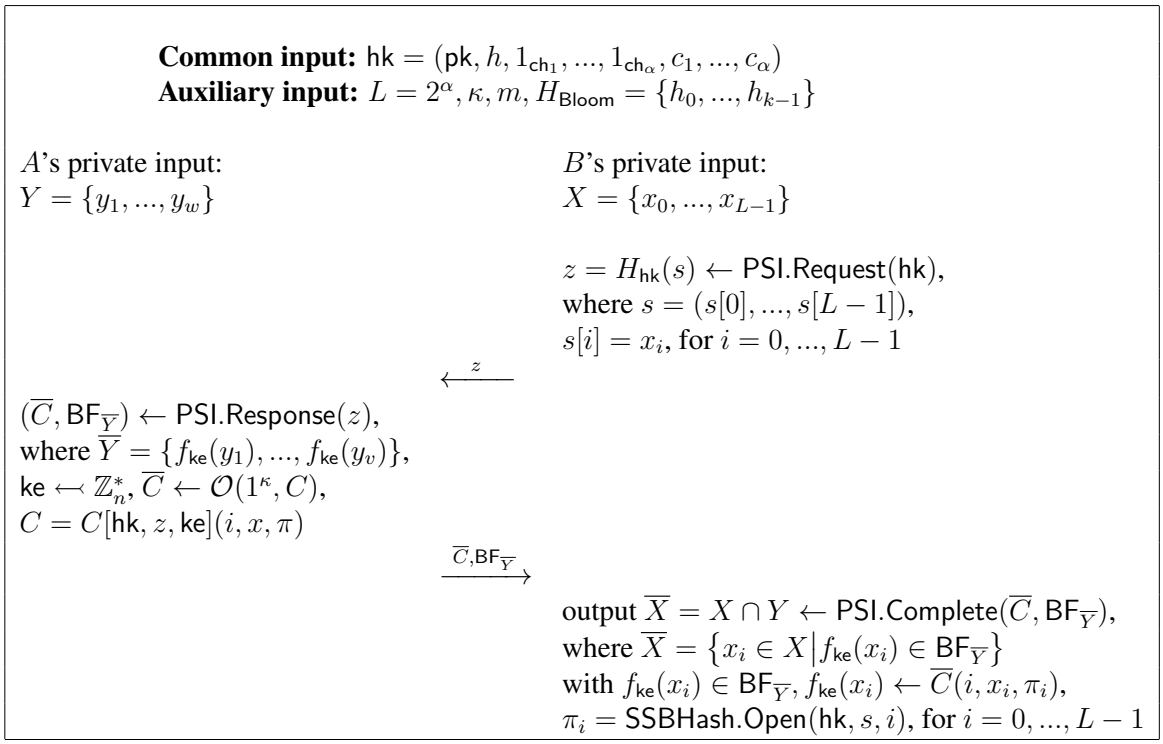

Figure 3 : Communication flow of our PSI

We now describe below the off-line and online phases of our protocol.

Off-line Phase: On input security parameter 1^κ, the server A does the following:

(i) Runs the algorithm 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 $\mathsf{hk} = (\mathsf{pk}, h, 1_{\mathsf{ch}_1}, ..., 1_{\mathsf{ch}_\alpha}, c_1, ..., c_\alpha)$, $pk = (n, g, w)$ $h : \mathbb{Z}^*_{n^{w+1}} \rightarrow \mathbb{Z}_{n^w}$ 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 consists of 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 $X = \{x_0, ..., x_{L-1}\} \subseteq \mathbb{Z}_{n^w}$ 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 $\mathsf{ke} \leftarrowtail \mathbb{Z}_n^*$ for PRF

  $$f_{\mathsf{ke}}(x) = \begin{cases} g^{1/(\mathsf{ke}+x)} \text{ if } gcd(\mathsf{ke}+x, n) = 1 \\ 1 \qquad\quad \text{otherwise,} \end{cases}$$

  where $x \in \{0,1\}^Q, \mathsf{ke} \in \mathbb{Z}_n^*$, where $Q = \lfloor w \log_2 n \rfloor$.

  (ii) Designs a circuit as described in Figure 1.

  (iii) Constructs an obfuscated circuit $\overline{C} \leftarrow \mathcal{O}(1^\kappa, C)$ of C.

  (iv) Runs Algorithm 2 to generate a Bloom filter $\mathsf{BF}_{\overline{Y}}$ of the set $\overline{Y} = \{f_{\mathsf{ke}}(y_1), ..., f_{\mathsf{ke}}(y_v)\}$, where $f_{\mathsf{ke}}(y_j) = g^{1/(\mathsf{ke}+y_j)}$ for $j = 1, ..., v$ and $Y = \{y_1, ..., y_v\} \subseteq \mathbb{Z}_{n^w}$ 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 $i = 0, ..., L-1$

  – 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.

  – runs Algorithm 3 on inputs $\mathsf{BF}_{\overline{Y}}$ and $f_{\mathsf{ke}}(x_i)$ to check 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 PSI.Complete phase executed by B:

$$\begin{aligned} & f_{\mathsf{ke}}(x_i) \text{ passes the check step of } \mathsf{BF}_{\overline{Y}} in Algorithm 3 \\ \Leftrightarrow\ & f_{\mathsf{ke}}(x_i) \in \overline{Y} \text{ except with negligible probability } \tfrac{1}{2^k} \\ \Leftrightarrow\ & \text{there exists } y_j \in Y \text{ such that } f_{\mathsf{ke}}(x_i) = f_{\mathsf{ke}}(y_j) \\ \Leftrightarrow\ & x_i = y_j \text{ as } f_{\mathsf{ke}}(\cdot) \text{ is a PRF function} \\ \Leftrightarrow\ & x_i \in X \cap Y \\ \Leftrightarrow\ & \overline{X} = X \cap Y (\text{ by the construction of } \overline{X}) \end{aligned}$$

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 $m = \frac{kv}{\ln 2}$, $|C|$=lenght of the circuit $C = C[\mathsf{hk}, z, \mathsf{ke}](i, x, \pi)$. The computation complexity of our PSI is displayed in Table 3.

Table 3 : 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	PSI.Response	v	v	kv		$2\eta(2M+5)^2+4(2M+5)$			
B	PSI.Request	$3(L-1)$	$L-1$		$L-1$				
B	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 3.1. *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 section 3 between a server A and a client B is a secure computation protocol for functionality $\mathcal{F}_{PSI}$ in the security model described in section 2.1 except with negligible probability $\frac{1}{2^k}$, under DCR assumption.*

Proof. We consider two cases to prove the security of our scheme: (i) when the client B is corrupted and (ii) when the server A is corrupted. In both the cases, our aim is to construct a simulator who given the input and output of the corrupted party, can extract a view which is indistinguishable from the real view of the corrupted party. Here view of an entity consists of input message, the outcome of the entity's internal coin tosses and the messages received by the entity during the protocol execution.

(i) **Case I** (B is corrupted). Let us construct a simulator S_B that has access to B's private input X and output $X \cap Y$. We will prove that S_B's simulated view $\mathsf{sim}_B = \{X, \mathsf{hk}_{L-1}, \widetilde{C}, \mathsf{BF}_{\widetilde{Y}}\}$ is indistinguishable from B's real-world view $\mathsf{View}_B = \{X, \mathsf{hk}, \overline{C}, \mathsf{BF}_{\overline{Y}}\}$, where $\mathsf{hk}_{L-1}, \widetilde{C}, BF_{\widetilde{Y}}$ are simulated protocol messages from the server A. We prove this via a sequence of hybrid arguments **Hybrid 0**, **Hybrid 1**, **Hybrid 2**, $\{\textbf{Hybrid } (\mathbf{2}; \mu, \nu), \mu \in \{0, ..., L\}, \nu \in \{0, ..., L-1\}\}$, **Hybrid 3**, where distributions of two successive hybrids differ only by small modifications and finally we arrive at the hybrid **Hybrid 4** which has the distribution of the simulated view sim_B. where we make small modifications and finally arrive at the distribution of the simulated view sim_B.

Hybrid 0 : This is analogous to the real world where B's view is $\mathsf{View}_B = \{X, \mathsf{hk}, \overline{C}, \mathsf{BF}_{\overline{Y}}\}$.

Hybrid 1 : This argument is same as **Hybrid 0** except that the simulator S_B selects $\{r'_i\}_{i=0}^{L-1}$ uniformly as B's random coins. Consequently, **Hybrid 0** and **Hybrid 1** are indistinguishable.

Hybrid 2 : This argument is similar to **Hybrid 1** except that the simulator S_B selects a PRF key $\bar{k}$ and obfuscates $C_0^{\mathsf{aug}} = C^{\mathsf{aug}}[\mathsf{hk}, z, \mathsf{ke}, \bar{k}, i^* = 0]$ given in Figure 2, instead of $C = C[\mathsf{hk}, z, \mathsf{ke}]$ on behalf of the server A by setting $\overline{C} \leftarrow \mathcal{O}(1^\kappa, C_0^{\mathsf{aug}})$. As $i^* = 0$ in $C_0^{\mathsf{aug}} = C^{\mathsf{aug}}[\mathsf{hk}, z, \mathsf{ke}, \bar{k}, i^* = 0]$ and $i \in \{0, ..., L-1\}$, we have $i \geq i^*$ in step 2 of Figure 2. Therefore, the outputs produced by both circuits C and C_0^{aug} on the same input (i, x, π) are identical to $f_{\mathsf{ke}}(x)$ if SSB.Verify succeeds or 0 otherwise. Hence by the iO security of the obfuscator $\mathcal{O}$, **Hybrid 1** and **Hybrid 2** are indistinguishable.

$\{$**Hybrid** $(\mathbf{2}; \mu, \nu), \mu \in \{0, ..., L\}, \nu \in \{0, ..., L-1\}\}$: This argument is same as **Hybrid 2** except the followings:

– Instead of obfuscating the circuit C_0^{aug}, the simulator S_B obfuscates the circuit $C_\mu^{\mathsf{aug}} = C^{\mathsf{aug}}[\mathsf{hk}, z, \mathsf{ke}, \bar{k}, i^* = \mu]$ by setting $\overline{C} \leftarrow \mathcal{O}(1^\kappa, C_\mu^{\mathsf{aug}})$, where C_μ^{aug} is as in Figure 2 with $i^* = \mu$.

– Instead of choosing $\mathsf{hk} \leftarrow \mathsf{SSBHash.Gen}(1^\kappa, 1^{l_{\mathsf{blk}}}, L, 0)$, the simulator S_B chooses $\mathsf{hk}_\nu \leftarrow \mathsf{SSBHash.Gen}(1^\kappa, 1^{l_{\mathsf{blk}}}, L, \nu)$ which is statistically binded at the index ν.

Clearly, **Hybrid** $(\mathbf{2}; 0, 0)$ is identical to **Hybrid 2**. We will now prove the following lemma:

Lemma 3.1. *(a) For $\mu \in \{0, ..., L-1\}$, the argument* **Hybrid** $(\mathbf{2}; \mu, \mu)$ *is indistinguishable from* **Hybrid** $(\mathbf{2}; \mu, \mu)$ *i.e.,*

$$\textbf{Hybrid } (\mathbf{2}; \mu, \mu) \stackrel{c}{\equiv} \textbf{Hybrid } (\mathbf{2}; \mu + 1, \mu), \mu \in \{0, ..., L-1\}.$$

(b) For $\mu \in \{0, ..., L-2\}$, the argument **Hybrid** $(\mathbf{2}; \mu + 1, \mu)$ *is indistinguishable from* **Hybrid** $(\mathbf{2}; \mu + 1, \mu + 1)$ *i.e.,*

$$\textbf{Hybrid } (\mathbf{2}; \mu + 1, \mu) \stackrel{c}{\equiv} \textbf{Hybrid } (\mathbf{2}; \mu + 1, \mu + 1), \mu \in \{0, ..., L-2\}.$$

Proof of Lemma 3.1: Note that in **Hybrid** $(\mathbf{2}; \mu, \mu)$ and **Hybrid** $(\mathbf{2}; \mu + 1, \mu)$, the circuits to be obfuscated by S_B are respectively C_μ^{aug} and $C_{\mu+1}^{\mathsf{aug}}$, and the hash keys are same which is $\mathsf{hk}_\mu \leftarrow \mathsf{SSBHash.Gen}(1^\kappa, 1^{l_{\mathsf{blk}}}, L, \mu)$. To prove part (a) of Lemma 3.1, we have to show that the outputs of $C_\mu^{\mathsf{aug}} = C^{\mathsf{aug}}[\mathsf{hk}_\mu, z, \mathsf{ke}, \bar{k}, i^* = \mu]$ and $C_{\mu+1}^{\mathsf{aug}} = C^{\mathsf{aug}}[\mathsf{hk}_\mu, z, \mathsf{ke}, \bar{k}, i^* = \mu + 1]$ are indistinguishable. Notice that for all inputs of the form (i, x, π) with $i < \mu$, both C_μ^{aug} and $C_{\mu+1}^{\mathsf{aug}}$ output 0 when $\mathsf{SSB.Verify}$ does not succeed or $f_{\bar{k}}(i)$ otherwise as we have $i < i^*$ in step 2 of Figure 2 for $i^* = \mu$ as well as $i^* = \mu + 1$. On the other hand, for all inputs of the form (i, x, π) with $i \geq \mu + 1$, both C_μ^{aug} and $C_{\mu+1}^{\mathsf{aug}}$ output either 0 or $f_{\mathsf{ke}}(x)$ according as $\mathsf{SSB.Verify}$ fails or succeeds. Thus the outputs of C_μ^{aug} and $C_{\mu+1}^{\mathsf{aug}}$ only differ for the inputs of the form (μ, x, π) when $\mathsf{SSBHash.Verify}(\mathsf{hk}_\mu, z, \mu, x, \pi) = \mathsf{accept}$. By the "somewhere statistically binding" property of the SSB hash which is selected to be binding at index μ, the inputs of the form (μ, x, π) with $\mathsf{SSBHash.Verify}(\mathsf{hk}_\mu, z, \mu, x, \pi) = \mathsf{accept}$ must satisfy $x = x_\mu$, where x_μ is B's μ-th input. In this case, C_μ^{aug} outputs $f_{\mathsf{ke}}(x)$, whereas $C_{\mu+1}^{\mathsf{aug}}$ outputs $f_{\bar{k}}(\mu)$. Note that $f_{\mathsf{ke}}(x)$ and $f_{\bar{k}}(\mu)$ are computationally indistinguishable by the security of the PRF f. Hence part (a) of Lemma 3.1 is established from by the security of f and iO security of the obfuscator $\mathcal{O}$. Part (b) of Lemma 3.1 immediately follows from the "index hiding" property of the SSB hash function.

Hence we can conclude by Lemma 3.1 that **Hybrid** $(\mathbf{2}; L, L-1)$ and **Hybrid** $(\mathbf{2}; 0, 0)$ are indistinguishable. As **Hybrid** $(\mathbf{2}; 0, 0)$ is identical to **Hybrid 2**, we can say that **Hybrid** $(\mathbf{2}; L, L-1)$ is indistinguishable from **Hybrid 2**.

Hybrid 3 : This argument is similar to **Hybrid** $(\mathbf{2}; L, L-1)$ except that the simulator S_B obfuscates $C' = C^{\mathsf{aug}}[\mathsf{hk}_{L-1}, z, \bot, \bar{k}, i^* = L]$ instead of $C_L^{\mathsf{aug}} = C^{\mathsf{aug}}[\mathsf{hk}_{L-1}, z, \mathsf{ke}, \bar{k}, i^* = L]$, where ke is replaced $\bot$. As ke is never used in C', output of C' is 0 for $\mathsf{SSB.Verify} =$

reject or $f_{\bar{k}}(i)$ otherwise. Also note that in step 2 of Figure 2, $i \geq i^* = L$ is never satisfied as $i \in \{0, ..., L-1\}$ for C_L^{aug} and output is 0 when SSB.Verify does not succeed, or $f_{\bar{k}}(i)$ otherwise. Therefore, behavior of C_L^{aug}, C' are identical and indistinguishability of $\mathbf{Hybrid}\ (\mathbf{2}; L, L-1)$ and $\mathbf{Hybrid\ 3}$ follows from the iO security of the obfuscator $\mathcal{O}$.

$\mathbf{Hybrid\ 4}$: Note that S_B has knowledge of PRF key $\bar{k}$, private input set X and output $X \cap Y$ of the corrupted party B. This hybrid is analogous to $\mathbf{Hybrid\ 3}$ except that the simulator S_B constructs a set $Y' = \{y'_1, ..., y'_v\}$ by including all the elements of $X \cap Y$ together with $v - |X \cap Y|$ many random elements and generates the Bloom filter $\mathsf{BF}_{\widetilde{Y}}$ of the set $\widetilde{Y} = \{f_{\bar{k}}(y'_1), ..., f_{\bar{k}}(y'_v)\}$ instead of $\mathsf{BF}_{\overline{Y}}$ on behalf of the server A following Algorithm 2. The Bloom filter $\mathsf{BF}_{\widetilde{Y}}$ is indistinguishable from $\mathsf{BF}_{\overline{Y}}$ in the sense that to construct $X \cap Y$, the client B checks the elements of the set $\{f_{\mathsf{ke}}(x_0), ..., f_{\mathsf{ke}}(x_{L-1})\}$ against $\mathsf{BF}_{\overline{Y}}$ in $\mathbf{Hybrid\ 3}$ whilst in $\mathbf{Hybrid\ 4}$, the client B checks the elements of the set $\{f_{\bar{k}}(x_0), ..., f_{\bar{k}}(x_{L-1})\}$ against $BF_{\widetilde{Y}}$. Thus the indistinguishability of $\mathbf{Hybrid\ 3}$ and $\mathbf{Hybrid\ 4}$ holds from the security of the PRF f.

(ii) **Case II** (A is corrupted). We construct a simulator S_A that has access to A's private input set Y and output $\perp$. We will show that S_A's simulated view $\mathsf{sim}_A = \{Y, \{\bar{R}_i, \bar{\gamma}_i\}_{i=1}^{\log_2 L}, \bar{z}\}$ is indistinguishable from A's real-world view $\mathsf{View}_A = \{Y, \{R_i, \gamma_i\}_{i=1}^{\log_2 L}, z\}$, where $\{\bar{\gamma}_i\}_{i=1}^{\log_2 L}$'s are A's simulated random coins which are required during generation of SSB hash key $\mathsf{hk} \leftarrow \mathsf{SSBHash.Gen}(1^\kappa, 1^{l_{\mathsf{blk}}}, L, 0)$ in the off-line phase and $\bar{z}$ is simulated protocol message from the client B. To prove this, we construct the following sequence of hybrid arguments $\mathbf{Hybrid\ 0}$, $\mathbf{Hybrid\ 1}$, where consecutive hybrids differ by small modifications and finally arrive at a hybrid $\mathbf{Hybrid\ 2}$ with the distribution of the simulated view sim_A.

$\mathbf{Hybrid\ 0}$: This is exactly the real world, where A's view is $\mathsf{View}_A = \{Y, \{\gamma_i\}_{i=1}^{\log_2 L}, z\}$.

$\mathbf{Hybrid\ 1}$: This argument is similar to $\mathbf{Hybrid\ 0}$ except that the simulator S_A selects $\{\bar{\gamma}_i\}_{i=1}^{\log_2 L}$ uniformly as A's random coins. As a consequence, $\mathbf{Hybrid\ 0}$ and $\mathbf{Hybrid\ 1}$ are indistinguishable.

$\mathbf{Hybrid\ 2}$: This argument is identical to $\mathbf{Hybrid\ 1}$ except that the simulator S_A chooses $\bar{z} \leftarrow\!\!\!\!\rightarrow \mathbb{Z}_{n^w}$ instead of computing $z = H_{\mathsf{hk}}(s)$ on behalf of B. By the security of randomization, $\mathbf{Hybrid\ 1}$ and $\mathbf{Hybrid\ 2}$ are indistinguishable.

4 Conclusion and Future Work

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. Extending our work to achieve security in the malicious environment is an interesting direction of future work.

References

[1] R. Agrawal, A. Evfimievski, and R. Srikant. Information sharing across private databases. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 86–97. ACM, 2003.

[2] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im) possibility of obfuscating programs. In *Annual International Cryptology Conference*, pages 1–18. Springer, 2001.

[3] D. A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc 1. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 1–5. ACM, 1986.

[4] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[5] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 309–325. ACM, 2012.

[6] J.-S. Coron, T. Lepoint, and M. Tibouchi. New multilinear maps over the integers. In *Annual Cryptology Conference*, pages 267–286. Springer, 2015.

[7] I. Damgård and M. Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *Public Key Cryptography*, pages 119–136. Springer, 2001.

[8] E. De Cristofaro, P. Gasti, and G. Tsudik. Fast and private computation of cardinality of set intersection and union. In *Cryptology and Network Security*, pages 218–231. Springer, 2012.

[9] E. De Cristofaro, J. Kim, and G. Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In *Advances in Cryptology-ASIACRYPT 2010*, pages 213–231. Springer, 2010.

[10] E. De Cristofaro and G. Tsudik. Practical private set intersection protocols with linear complexity. In *Financial Cryptography and Data Security*, pages 143–159. Springer, 2010.

[11] E. De Cristofaro and G. Tsudik. Experimenting with fast private set intersection. In *Trust and Trustworthy Computing*, pages 55–73. Springer, 2012.

[12] C. Dong, L. Chen, and Z. Wen. When private set intersection meets big data: An efficient and scalable protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 789–800. ACM, 2013.

[13] M. J. Freedman, C. Hazay, K. Nissim, and B. Pinkas. Efficient set intersection with simulation-based security. *Journal of Cryptology*, 29(1):115–155, 2016.

[14] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology-EUROCRYPT 2004*, pages 1–19. Springer, 2004.

[15] S. Garg, C. Gentry, and S. Halevi. Candidate multilinear maps from ideal lattices. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 1–17. Springer, 2013.

[16] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 40–49. IEEE, 2013.

[17] C. Gentry, A. B. Lewko, A. Sahai, and B. Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 151–170. IEEE, 2015.

[18] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*, volume 2. Cambridge university press, 2009.

[19] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM (JACM)*, 33(4):792–807, 1986.

[20] C. Hazay. Oblivious polynomial evaluation and secure set-intersection from algebraic prfs. *IACR Cryptology ePrint Archive*, page 4, 2015.

[21] C. Hazay and Y. Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *Theory of Cryptography*, pages 155–175. Springer, 2008.

[22] C. Hazay and K. Nissim. Efficient set operations in the presence of malicious adversaries. In *Public Key Cryptography–PKC 2010*, pages 312–331. Springer, 2010.

[23] S. Hohenberger and S. A. Weis. Honest-verifier private disjointness testing without random oracles. In *Privacy Enhancing Technologies*, pages 277–294. Springer, 2006.

[24] Y. Huang, D. Evans, and J. Katz. Private set intersection: Are garbled circuits better than custom protocols. In *Network and Distributed System Security Symposium (NDSS). The Internet Society*, 2012.

[25] P. Hubacek and D. Wichs. On the communication complexity of secure function evaluation with long output. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, pages 163–172. ACM, 2015.

[26] S. Jarecki and X. Liu. Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection. In *Theory of Cryptography*, pages 577–594. Springer, 2009.

[27] M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 427–437. ACM, 1990.

[28] T. Okamoto, K. Pietrzak, B. Waters, and D. Wichs. New realizations of somewhere statistically binding hashing and positional accumulators. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 121–145. Springer, 2015.

[29] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 223–238. Springer, 1999.

[30] R. Pass, K. Seth, and S. Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In *International Cryptology Conference*, pages 500–517. Springer, 2014.

[31] B. Pinkas, T. Schneider, G. Segev, and M. Zohner. Phasing: Private set intersection using permutation-based hashing. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 515–530, 2015.

[32] B. Pinkas, T. Schneider, and M. Zohner. Faster private set intersection based on ot extension. In *USENIX Security*, volume 14, pages 797–812, 2014.

[33] R.-h. Shi, Y. Mu, H. Zhong, J. Cui, and S. Zhang. An efficient quantum scheme for private set intersection. *Quantum Information Processing*, 15(1):363–371, 2016.