O-PSI: Delegated Private Set Intersection on Outsourced Datasets

Aydin Abadi, Sotirios Terzis, and Changyu Dong(✉)

Department of Computer and Information Sciences,
University of Strathclyde, Glasgow, UK
{aydin.abadi,sotirios.terzis,changyu.dong}@strath.ac.uk

Abstract. Private set intersection (PSI) has a wide range of applications such as privacy-preserving data mining. With the advent of cloud computing it is now desirable to take advantage of the storage and computation capabilities of the cloud to outsource datasets and delegate PSI computation. In this paper we design O-PSI, a protocol for delegated private set intersection on outsourced datasets based on a novel point-value polynomial representation. Our protocol allows multiple clients to independently prepare and upload their private datasets to a server, and then ask the server to calculate their intersection. The protocol ensures that intersections can only be calculated with the permission of all clients and that datasets and results remain completely confidential from the server. Once datasets are outsourced, the protocol supports an unlimited number of intersections with no need to download them or prepare them again for computation. Our protocol is efficient and has computation and communication costs linear to the cardinality of the datasets. We also provide a formal security analysis of the protocol.

1 Introduction

Cloud computing allows clients with limited computation and storage capabilities to outsource their private data and at a later time, ask the cloud to perform computation on them. Delegation of data storage and computation to the cloud has become common practice for individuals and big enterprises alike [1,2]. As a result, often the need arises for clients to perform computation on their outsourced private data jointly, ideally without the need to download the data.

In this paper, we consider a particular such scenario, in which the private data take the form of sets and the computation of interest is set intersection, i.e. private set intersection (PSI).

In PSI, two parties want to find out the intersection of their sets and also want to prevent the other party from finding out anything more about their own set than the elements of the intersection. In general, PSI captures a wide range of real-world applications such as privacy preserving data mining [3], homeland security [4] and so on. For example, consider a case where a law enforcement agency has a list of suspects and wants to compare it against flight passenger lists. Here the names of the suspects should be kept hidden from the airlines

H. Federrath and D. Gollmann (Eds.): SEC 2015, IFIP AICT 455, pp. 3–17, 2015.
DOI: 10.1007/978-3-319-18467-8_1

while the agency should not be able to find out about other passengers in order to protect their privacy. As another example, consider the situation where a social welfare organization wants to know whether any of its members receives income from another organization, but neither organization can reveal their list of members.

Although a number of protocols have been proposed for PSI (see section 2 for a survey), cloud computing introduces additional challenges as the private datasets are outsourced and the private set intersection is delegated to cloud servers. In addition to keeping their sets confidential, clients are also interested in preventing cloud servers from finding out anything about their sets and the intersection. In other words, clients are interested in *delegated private set intersection on outsourced data*. To allow for more flexibility it is desirable that clients should be able to engage in PSI computation with any other client of the cloud provider. However, they should remain in charge of deciding which clients are allowed to use their sets. To fully take advantage of the cloud capabilities and minimize costs, clients should not have to keep locally or download their datasets every time an intersection needs to be computed, while their involvement to the computation should be limited.

We propose O-PSI, a PSI protocol that addresses these requirements. Our protocol uses homomorphic encryption and a novel point-value polynomial representation for datasets that allows clients to independently secure their sets and outsource them to the cloud, while cloud servers are able to calculate their intersection. The protocol ensures that intersections can only be computed with the permission of the clients and that the result will remain secret from the server. The protocol also allows outsourced sets to be used an unlimited number of times securely without the need to secure them again. More interestingly, the novel set representation means that computation and communication costs are linear to the size of the sets.

The paper starts with a survey of related work in section 2, followed by a brief overview of our security model and key concepts we rely on in section 3. Section 4 presents the design of our protocol, while section 5 proves its security. Section 6 proposes extensions to support data integrity verification and multiple clients, while section 7 presents an analysis of its computation and communication complexity, and a comparison to work that is closest to our aims. Section 8 concludes the paper and identifies directions for future work.

2 Related Work

Private set intersection (PSI) was introduced in [5]. Following that [6] proposed a number of protocols supporting further set operations and multiple clients based on additive homomorphic encryption and polynomial representation of sets. More recently, several efficient protocols have been proposed. For example, [4,7] use blind signatures and hash functions to provide efficient PSI in the semi-honest and the malicious security models respectively, [8] uses Bloom filters, secret sharing and oblivious transfer to offer even more efficient protocols,

and [9] extends [8] and uses hash tables and a more efficient oblivious transfer extension protocol for better efficiency. However, all these regular PSI protocols are interactive, in the sense that clients jointly compute the intersection. They are not designed with the capability to outsource any data or delegate any of the computation to a third party.

In another line of research, in [10,11] the protocols proposed for outsourced verifiable dynamic set operations, including set intersection. These protocols make use of bilinear map accumulators and authenticated hash tables (i.e. accumulator trees) to verify the correctness of operations carried out by a server on outsourced sets. However, these protocols are designed for a single client to outsource a collection of sets to a server and later to compute the intersections of its own sets. The protocols are designed to provide verifiability of computation, not data privacy. Data are outsourced in plaintext and the protocols do not work if data are encrypted.

More interestingly, a number of PSI protocols have been proposed in which clients delegate computation to a server [12–16]. A protocol proposed in [14] allows clients to outsource their sets to a server by hashing each element and adding a random value. They then delegate the computation of the intersection to the server. However, this protocol is not fully private, as it reveals to the server the cardinality of the intersection. In addition to the above issue, because of the way the sets are encoded if the intersection between the sets of client A and B is computed, followed by that between the sets of client A and C, then the server will also find out whether some elements are common in the sets of client B and C without their consent. In [16] clients also delegate the computation to a server. Clients encrypt their sets and outsource them. The server also provides a proof that allows the clients to verify the correctness of the result. However, the protocol is not fully private and suffers from the same issues described above. Another protocol that delegates computation to a server is proposed in [12]. The protocol is based on a pseudorandom permutation (PRP) of the set elements with the key for the PRP generated jointly by the clients at setup. One variant of the protocol can hide the cardinality of the intersection. However, in this variant computation is delegated to one of the clients rather than the server. The server's role is limited to re-encoding one client's set to maintain the privacy of the computation. In the protocol, clients can detect if the server provided incorrect results at the cost of replicating a number of times all elements of the sets.

In a similar line of research, a protocol proposed in [13] allows one client, say client A, to encrypt and outsource its set, and delegate computation to a server. The server can then engage in a PSI protocol on this client's behalf with another client, say client B. However, this delegation is one-off: if A wants to compute set intersection with C, then A must encrypt its set with a new key and re-delegate to the server. In addition to this protocol, in [15] two clients can delegate the PSI computation to a server. In this protocol rather than encrypting and outsourcing their sets, the clients encrypt and outsource bloom filters of their sets that are then used by the server to privately compute their intersection. However, in this case in order for the clients to get the result of the intersection they need to keep a local copy of their sets. So, this protocol does not really allow outsourcing the sets.

From the above discussion, it should be clear that none of the protocols above allows clients to fully delegate PSI computation to the server without the need to either maintain the sets locally or having to re-encode and re-upload the sets for each intersection computation, namely none support *delegated private set intersection on outsourced sets*. As a result, none of them are particularly suited for a cloud computing setting.

3 Preliminaries

3.1 Security Model

We consider a setting in which *static semi-honest* adversaries are present. In this setting, the adversary controls one of the parties and follows the protocol specification exactly. However, it may try to learn more information about the other party's input. The definitions and model are according to [17].

In a delegated PSI protocol, three parties are involved: a server P, and two clients A and B. We assume the server does not collude with A or B. As the server (or cloud provider) is often a well established IT company, it is reasonable to assume it will not collude with the clients because collusion will seriously damage its reputation and decrease its revenue. This non-colluding assumption is widely used in the literature [12,18,19]. The three-party protocol π computes a function that maps the inputs to some outputs. We define this function as follows: $F : \Lambda \times 2^{\mathcal{U}} \times 2^{\mathcal{U}} \rightarrow \Lambda \times \Lambda \times f_{\cap}$, where Λ denotes the empty string, $2^{\mathcal{U}}$ denotes the powerset of the set universe and $f_{\cap}$ denotes the set intersection function. For every tuple of inputs Λ, S_A and S_B belong to P, A and B respectively, the function outputs nothing to P and A, and outputs $f_{\cap}(S_A, S_B) = S_A \cap S_B$ to B.

In the semi-honest model, a protocol π is secure if whatever can be computed by a party in the protocol can be obtained from its input and output only. This is formalized by the simulation paradigm. We require a party's *view* in a protocol execution to be simulatable given only its input and output. The view of the party i during an execution of π on input tuple (x, y, z) is denoted by $\mathsf{view}_i^{\pi}(x, y, z)$ and equals $(w, r^i, m_1^i, ..., m_t^i)$ where $w \in (x, y, z)$ is the input of i, r^i is the outcome of i's internal random coin tosses and m_j^i represents the jth message that it received.

Definition 1. *Let F be a deterministic function as defined above. We say that the protocol π securely computes F in the presence of static semi-honest adversaries if there exist probabilistic polynomial-time algorithms Sim_P, Sim_A and Sim_B that given the input and output of a party, can simulate a view that is computationally indistinguishable from the party's view in the protocol:*

$$Sim_P(\Lambda, \Lambda) \stackrel{c}{\equiv} \mathsf{view}_P^{\pi}(\Lambda, S_A, S_B)$$

$$Sim_A(S_A, \Lambda) \stackrel{c}{\equiv} \mathsf{view}_A^{\pi}(\Lambda, S_A, S_B)$$

$$Sim_B(S_B, f_{\cap}(S_A, S_B)) \stackrel{c}{\equiv} \mathsf{view}_B^{\pi}(\Lambda, S_A, S_B)$$

3.2 Homomorphic Encryption

A semantically secure additively homomorphic public key encryption scheme has the following properties:

1. Given two ciphertexts $E_{pk}(a), E_{pk}(b)$, $E_{pk}(a) \cdot E_{pk}(b) = E_{pk}(a+b)$.
2. Given a ciphertext $E_{pk}(a)$ and a constant b, $E_{pk}(a)^b = E_{pk}(a \cdot b)$.

One such scheme is the Paillier public key cryptosystem [20]. It works as follows:

Key Generation: Choose two random large primes p and q according to a given security parameter, and set $N = pq$. Let u be the Carmichael value of N, i.e. $u = lcm(p-1, q-1)$ where lcm stands for the least common multiple. Choose a random $g \in \mathbb{Z}^*_{N^2}$, and ensure that $s = (L(g^u \bmod N^2))^{-1} \bmod N$ exists where $L(x) = \frac{(x-1)}{N}$. The public key is $pk = (N, g)$ and the secret key is $sk = (u, s)$.

Encryption: To encrypt a plaintext $m \in \mathbb{Z}_N$, pick a random value $r \in \mathbb{Z}^*_N$, and compute the ciphertext: $C = E_{pk}(m) = g^m \cdot r^N \bmod N^2$.

Decryption: To decrypt a ciphertext C, $D_{sk}(C) = L(C^u mod\ N^2) \cdot s \bmod N = m$.

3.3 Polynomial Representation of Sets

Many PSI protocols e.g. [5,6], use a polynomial representation of sets. Let R be a field, then we denote a polynomial ring as $R[x]$. The polynomial ring $R[x]$ consists of all polynomials with coefficients from R. Given a set S of size d, $|S| = d$, we can map each element in S to an element in a sufficiently large field R. Then we can represent this set as a polynomial in the polynomial ring $R[x]$. The polynomial is defined as $\rho(x) = \prod_{s_i \in S}(x - s_i)$ and has the property that every element $s_i \in S$ is a root of it.

For two sets S_A and S_B represented by polynomials ρ_A and ρ_B respectively, then $gcd(\rho_A, \rho_B)$ represents the set intersection $S_A \cap S_B$, where gcd stands for the greatest common divisor. For polynomials ρ_A and ρ_B of degree d and γ_A and γ_B that are degree d polynomials chosen uniformly at random from $R[x]$, it is proved in [6] that $\gamma_A \cdot \rho_A + \gamma_B \cdot \rho_B = \mu \cdot gcd(\rho_A, \rho_B)$ such that μ is a uniformly random polynomial. This means that if ρ_A and ρ_B are polynomials representing sets S_A and S_B, then the polynomial $\gamma_A \cdot \rho_A + \gamma_B \cdot \rho_B$ contains only information about $S_A \cap S_B$ and no information about other elements in S_A or S_B. This forms the basis of their PSI protocol in which a party obtains $\gamma_A \cdot \rho_A + \gamma_B \cdot \rho_B$ to find the set intersection but learns nothing more about elements in the other party's set.

4 O-PSI: Delegated Private Set Intersection on Outsourced Datasets

4.1 Polynomials in Point-value Form

In section 3.3 we showed that a set can be represented as a polynomial and set intersection can be computed by polynomial arithmetic. All previous PSI protocols using polynomial representation of sets, represent a polynomial as a vector

of polynomial's coefficients. They represent a degree d polynomial $\rho = \sum_{i=0}^{d} a_i x^i$ as a vector $\mathbf{a} = (a_0, a_1, ..., a_d)$. This representation, while it allows the protocols to correctly compute the result, has a major disadvantage. The complexity of multiplying two polynomials of degree d in co-efficient representation is $O(d^2)$. In PSI protocols, this leads to significant computational overheads. Usually in such protocols, one polynomial needs to be encrypted and the polynomial multiplication has to be done homomorphically. Homomorphic multiplication operations are computationally expensive. Thus using a co-efficient representation means that the protocols are not scalable.

In O-PSI, we solve this problem by representing the polynomials in another well-known form, point-value. A degree d polynomial ρ can be represented as a set of n $(n > d)$ point-value pairs $\{(x_0, y_0), ..., (x_{n-1}, y_{n-1})\}$ such that all x_i are distinct and $y_i = \rho(x_i)$ for $0 \leq i \leq n-1$. If the x values are fixed, we can omit them and represent polynomials as vectors $\mathbf{y} = (y_0, y_1, ..., y_{n-1})$. A polynomial in point-value form can be translated into co-efficient form by polynomial interpolation [21]. Polynomial arithmetic in point-value representation can be done by point-wise addition or multiplication. For two degree d polynomials ρ_A and ρ_B represented in point-value form by two vectors $\mathbf{y}^{(A)}$ and $\mathbf{y}^{(B)}$, $\rho_A + \rho_B$ can be computed as $(y_1^{(A)} + y_1^{(B)}, y_2^{(A)} + y_2^{(B)}, ..., y_{n-1}^{(A)} + y_{n-1}^{(B)})$, and $\rho_A \cdot \rho_B$ can be computed as $(y_1^{(A)} \cdot y_1^{(B)}, y_2^{(A)} \cdot y_2^{(B)}, ..., y_{n-1}^{(A)} \cdot y_{n-1}^{(B)})$. Note because the product of $\rho_A \cdot \rho_B$ is a polynomial of degree $2d$, ρ_A and ρ_B must be represented by at least $2d + 1$ points to accommodate the result. The key benefit of point-value representation is that multiplication complexity is reduced to $O(d)$. This makes O-PSI much more scalable.

4.2 O-PSI Protocol

The interaction between parties in O-PSI is depicted in Fig. 1. At a high level, the protocol works as follows. Each client first outsources its set to the server. To do so, the client uploads a vector that encodes its set to the server. The vector is blinded so that the server cannot figure out the client's set, and the other client cannot figure out any element outside the intersection. If a client,

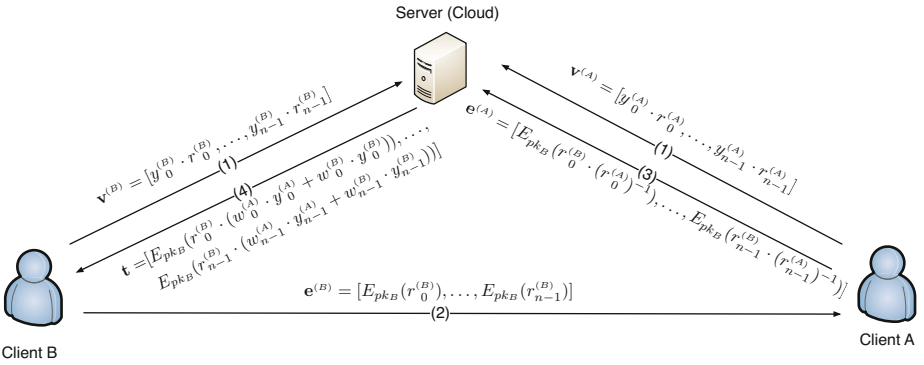

Fig. 1. Interaction between parties in O-PSI

client B, wants to compute the intersection of its own set and another client's set, say client A's set, it must obtain permission from A. If A agrees, A can compute jointly with B some encrypted values. The encrypted values will be used by the server to remove part of the blinding factors from A's data, and this then allows the set intersection to be computed. At the end of the protocol client B receives an encrypted vector which it can decrypt and use the decrypted values to interpolate a polynomial that encodes the intersection. The protocol is described below. We will explain the rationale behind the protocol design after the protocol description.

1. **Setup** Let $\mathcal{U}$ be the universe of set elements. There is a public finite field R that is big enough to encode all elements in $\mathcal{U}$ and also when an element is picked uniformly at random from R has only negligible probability of representing an element of a set. Client A has a set $S_A \subset \mathcal{U}$ and client B has a set $S_B \subset \mathcal{U}$. Without loss of generality, we let $|S_A| = |S_B| = d$. The server publishes a vector $\mathbf{x}$ containing $n = 2d + 1$ random distinct values from R. The server also publishes a pseudorandom function $f : \{0,1\}^l \times \mathbb{Z} \rightarrow R$, which maps an l-bit string to an element in R pseudorandomly.
2. **Outsource** This step is the same at both clients. Let $I \in \{A, B\}$, then the client I does the following:
   (a) Generates a Paillier key pair (pk_I, sk_I) (see section 3.2) and publishes the public key. It also chooses a random private key k_I for the pseudorandom function f. All keys are generated according to a given security parameter.
   (b) Constructs a polynomial $\tau_I = \prod_{s_i^{(I)} \in S_I}(x - s_i^{(I)})$ that represents its set S_I. Evaluates τ_I at every value in the $\mathbf{x}$ published by the server producing $\mathbf{y}^{(I)}$ such that $y_i^{(I)} = \tau_I(x_i)$ for $0 \leq i \leq n-1$.
   (c) Sends $\mathbf{v}^{(I)}$ to the server, where $\forall v_i^{(I)} \in \mathbf{v}^{(I)}, v_i^{(I)} = y_i^{(I)} \cdot r_i^{(I)}$, $y_i^{(I)}$ is the ith element in $\mathbf{y}^{(I)}$, $r_i^{(I)} = f(k_I, i)$. Here, $\mathbf{v}^{(I)}$ is a blinded version of its set polynomial.
3. **Set Intersection** In this step, client B wants to know the intersection of its set and client A's set.
   (a) Client B sends a request to client A. Along with the request, client B also sends its ID and a vector $\mathbf{e}^{(B)}$, such that $e_i^{(B)} = E_{pk_B}(r_i^{(B)})$ where $r_i^{(B)} = f(k_B, i)$ for $0 \leq i \leq n-1$ are the values used to blind its set polynomial.
   (b) Client A can send a **Deny** message to end the protocol here, or if it agrees to engage in the computation of the set intersection, it sends a **Permit** message to client B. It also sends a **Compute** message that contains its own and B's IDs, and a vector $\mathbf{e}^{(A)}$ to the server. The vector $\mathbf{e}^{(A)}$ is computed as follows: for $0 \leq i \leq n-1$, $e_i^{(A)} = (e_i^{(B)})^{(r_i^{(A)})^{-1}} = E_{pk_B}(r_i^{(B)} \cdot (r_i^{(A)})^{-1})$ where $r_i^{(I)} = f(k_I, i)$ for $I \in \{A, B\}$ are the values from step 2c above.
   (c) After receiving the **Compute** message from A, the server extracts $\mathbf{e}^{(A)}$ and retrieves the data $\mathbf{v}^{(A)}$ and $\mathbf{v}^{(B)}$ from its storage. The server then chooses two degree d polynomials ω_I randomly from $R[x]$ and computes

two vectors $\mathbf{w}^{(I)}$ ($I \in \{A, B\}$) such that $w_i^{(I)} = \omega_I(x_i)$ for $0 \leq i \leq n-1$ where x_i is the ith element in the public vector $\mathbf{x}$.

(d) The server computes a result vector $\mathbf{t}$ such that for $0 \leq i \leq n-1$:

$$\begin{aligned} t_i &= (e_i^{(A)})^{v_i^{(A)} \cdot w_i^{(A)}} \cdot E_{pk_B}(w_i^{(B)} \cdot v_i^{(B)}) \\ &= E_{pk_B}(r_i^{(B)} \cdot (r_i^{(A)})^{-1} \cdot y_i^{(A)} \cdot r_i^{(A)} \cdot w_i^{(A)}) \cdot E_{pk_B}(w_i^{(B)} \cdot y_i^{(B)} \cdot r_i^{(B)}) \\ &= E_{pk_B}(r_i^{(B)} \cdot (w_i^{(A)} \cdot y_i^{(A)} + w_i^{(B)} \cdot y_i^{(B)})) \end{aligned}$$

The server sends $\mathbf{t}$ to client B.

(e) After receiving $\mathbf{t}$, client B computes a vector $\mathbf{z}$ such that for $0 \leq i \leq n-1$:

$$\begin{aligned} z_i &= D_{sk_B}(t_i) \cdot (r_i^{(B)})^{-1} \\ &= r_i^{(B)} \cdot (w_i^{(A)} \cdot y_i^{(A)} + w_i^{(B)} \cdot y_i^{(B)}) \cdot (r_i^{(B)})^{-1} \\ &= w_i^{(A)} \cdot y_i^{(A)} + w_i^{(B)} \cdot y_i^{(B)} \end{aligned}$$

It then interpolates the polynomial ζ using point-value pairs (x_i, z_i). The roots of ζ are the elements in the set intersection.

Remark 1: In the Setup step, the server needs to publish a vector $\mathbf{x}$ that has $2d+1$ elements, because the polynomial ζ in step 3e is of degree $2d$ and at least $2d+1$ points are needed to interpolate it. The elements in $\mathbf{x}$ are picked at random from R so that the probability of x_i being a root of a client's polynomial is negligible.

Remark 2: In step 2c, the client blinds its vector. If the client stores $\mathbf{y}$ directly on the server without blinding, then the server can use $\mathbf{y}$ and $\mathbf{x}$ to interpolate the client's polynomial, thus revealing the client's set. With blinding this is not possible unless the server knows the pseudorandom function key used by the client. The protocol blinds values by multiplication. However, multiplication cannot blind a value if the value is 0. This is why we require the probability of x_i in $\mathbf{x}$ being a root of a client's polynomial to be negligible. If x_i is a root then y_i is 0 and cannot be blinded.

Remark 3: The data values stored on the server are blinded by their owner. To compute the set intersection those blinding factors ($r_i^{(I)}$ in the protocol) must be eliminated. In step 3b, client A and B jointly compute the vector $\mathbf{e}^{(A)}$ to "switch" A's blinding factors to B's blinding factors. In step 3d, $\mathbf{e}^{(A)}$ is used to eliminate $r_i^{(A)}$ and replace it with $r_i^{(B)}$. This factor switching makes it possible later to eliminate $r_i^{(B)}$ in step 3e. The values in $\mathbf{e}^{(A)}$ are encrypted with B's public key, so the server learns nothing in this process.

Remark 4: The client's original blinded dataset remains unchanged in the server. In fact in step 3c, the server multiplies *a copy* of the client's blinded dataset by the vector $\mathbf{w}^{(I)}$.

5 Proof of Security

Now we sketch the security proof of O-PSI in the semi-honest model (see section 3.1). We conduct the security analysis for the three cases where one of the parties is corrupted.

Theorem 1. *If the homomorphic encryption scheme is semantically secure, the O-PSI protocol is secure in the presence of a semi-honest adversary.*

Proof. We will prove the theorem by considering in turn the case where each of the parties has been corrupted. In each case we invoke the simulator with the corresponding party's input and output. Our focus is in the case where party A wants to engage in the computation of the intersection. If party A does not want to proceed in the protocol, the views can be simulated in the same way up to the point where the execution stops.

Case 1: Corrupted server In this case, we show that we can construct a simulator Sim_P that can produce a computationally indistinguishable view. In the real execution, the server's view is as follows:

$$\mathsf{view}_P^{\pi}(\Lambda, S_A, S_B) = \{\Lambda, r_P, \mathbf{v}^{(A)}, \mathbf{v}^{(B)}, \mathbf{Compute}, \mathbf{e}^{(A)}, \Lambda\}$$

where r_P are the random coins of the server, $\mathbf{v}^{(A)}, \mathbf{v}^{(B)}$ are the blinded set representations of A's and B's sets, **Compute** is the command to proceed from A, and $\mathbf{e}^{(A)}$ is the encrypted vector that is used in the protocol to switch blinding factors.

To simulate the view, Sim_P does the following: it creates an empty view, then appends Λ and uniformly at random chosen coins r'_P to the view. It then randomly generates two d-element sets S'_A and S'_B. It also chooses two random keys k'_A and k'_B for a pseudorandom function f. It encodes S'_A into its polynomial representation, evaluates the polynomial with the public values $\mathbf{x}$, and blinds the evaluation results with $r_i^{(A)'} = f(k'_A, i)$ for $0 \leq i \leq n-1$. The result is $\mathbf{v}^{(A)'}$. Similarly it can generate $\mathbf{v}^{(B)'}$. Then $\mathbf{v}^{(A)'}$ and $\mathbf{v}^{(B)'}$ are appended to the view. Following that, the simulator generates the **Compute** command string with the correct format and appends it to the view. It then computes $r_i^{(B)'} \cdot (r_i^{(A)'})^{-1}$ and encrypts the results with B's public key. This produces $\mathbf{e}^{(A)'}$ that is appended to the view. Finally, the simulator appends Λ to the view and outputs the view.

We argue that the simulated view is computationally indistinguishable from the real view. In both views, the input parts are identical, the random coins are both uniformly random, and so they are indistinguishable. In the real view $\mathbf{v}^{(A)}, \mathbf{v}^{(B)}$ are blinded with the outputs of a pseudorandom function, so do the vectors in the simulated view. Since the outputs of the pseudorandom function are computationally indistinguishable, the distributions of $\mathbf{v}^{(A)}, \mathbf{v}^{(B)}, \mathbf{v}^{(A)'}, \mathbf{v}^{(B)'}$ are therefore computationally indistinguishable. If the homomorphic encryption is semantically secure, then $\mathbf{e}^{(A)}$ and $\mathbf{e}^{(A)'}$ are also computationally indistinguishable. The output parts in both views are identical. So, we conclude that the views are indistinguishable.

Case 2: Corrupted client A In the real execution, the A's view is as follows:

$$\mathsf{view}_A^{\pi}(\Lambda, S_A, S_B) = \{S_A, r_A, \mathbf{e}^{(B)}, \Lambda\}$$

The simulator Sim_A does the following: it creates an empty view, then appends Λ and uniformly at random chosen coins r'_A to the view. It then chooses n random values r_i and encrypts each r_i with B's public key. The result is $\mathbf{e}^{(B)'}$ and it is appended to the view. The simulator then appends Λ to the view. It is easy to see that If the homomorphic encryption is semantically secure, then $\mathbf{e}^{(B)}$ and $\mathbf{e}^{(B)'}$ are computationally indistinguishable. So, the two views are indistinguishable.
Case 3: Corrupted client B In the real execution, the B's view is as follows:

$$\mathsf{view}_B^{\pi}(\Lambda, S_A, S_B) = \{S_B, r_B, \mathbf{Permit}, \mathbf{t}, f_{\cap}(S_A, S_B)\}$$

The simulator Sim_B does the following: it creates an empty view, and appends Λ and uniformly at random chosen coins r'_B to the view. Then it generates the **Permit** command string with the correct format and appends it to the view. Following that, it creates two d-element sets S'_A and S'_B such that $S'_A \cap S'_B = f_{\cap}(S_A, S_B)$, converts S'_A to its polynomial representation, evaluates the polynomial using the public values $\mathbf{x}$ and obtains $\mathbf{y}^{(A)'}$. Similarly the simulator can obtain $\mathbf{y}^{(B)'}$. The simulator chooses randomly two degree d polynomials ω'_A and ω'_B, evaluates them using the public values $\mathbf{x}$ and obtains $\mathbf{w}^{(A)'}$ and $\mathbf{w}^{(B)'}$. It also chooses a random key k'_B for a pseudorandom function f and computes $r_i^{(B)'} = f(k'_B, i)$ for $0 \leq i \leq n-1$. Then the simulator computes for each i, $E_{pk_B}(r_i^{(B)'} \cdot (w_i^{(A)'} \cdot y_i^{(A)'} + w_i^{(B)'} \cdot y_i^{(B)'}))$. The result is $\mathbf{t}'$. The simulator appends $\mathbf{t}'$ to the view and then appends $f_{\cap}(S_A, S_B)$. It is easy to see that the distributions of $\mathbf{t}$ and $\mathbf{t}'$ are computationally indistinguishable. So, the two views are indistinguishable.

Combining the above, we conclude the protocol is secure and complete our proof.

6 Extensions

In this section we extend O-PSI to support dataset integrity verification and multiple clients. These extensions require no major modification of the protocol.

6.1 Dataset Integrity Verification

To add data integrity verification to O-PSI we can use the verification mechanism of any provable data possession protocol that does not reveal any information about the confidential data to the server. For this purpose, we can adopt the homomorphic verification tags proposed in [22]. These tags are homomorphic in the sense that given two tags T_a and T_b for elements a and b one can combine them $T_a \cdot T_b$ which is equal to the tag Tag_{a+b} of the sum $a+b$ of the two elements.

In O-PSI, client $I \in \{A, B\}$ defines a tag for each element $v_i^{(I)}$ of the blinded dataset as: $T_{v_i^{(I)}} = (h(k_I||i) \cdot g^{v_i^{(I)}})^{d_I} \bmod N$, where h is a secure deterministic hash-and-encode function that maps strings uniformly to a unique cyclic subgroup of $\mathbb{Z}_N^*$, QR_N, k_I is a random value used for all elements in the set, $g = a^2, a \xleftarrow{R} Z_N^*$, and $N = p'q'$ is a RSA modulus, $p' = 2p'' + 1, q' = 2q'' + 1$ and $d_I \cdot e_I = 1 \; mod \; p''q''$, where q'' and p'' are prime numbers. The hash value $h(k_I||i)$ binds the tag $T_{v_i^{(I)}}$ to the value $v_i^{(I)}$ and prevents the server from using the tag to compute a proof for a different value. Note, $v_i^{(I)} = y_i^{(I)} \cdot r_i^{(I)}$ is a uniformly random value. Consequently, each tag $T_{v_i^{(I)}}$ does not leak any information about the private value $y_i^{(I)}$ to the server. In this protocol client I, along with its blinded dataset, outsources a vector $\mathbf{tag}^{(I)}$ comprising values $T_{v_i^{(I)}}$ $(0 \leq i \leq n-1)$ to the server. The challenge, proof generation and verification phases of the protocol remain unchanged to those described in [22].

6.2 Multiple Clients

O-PSI can be used to compute the intersection of the outsourced datasets of multiple clients. In this case, the client interested in the intersection, client B, sends the same request (see step 3a of the protocol) to all other clients, A_j $(1 \leq j \leq m)$. The protocol for each client A_j remains unchanged (see step 3b). For each client A_j, the server carries out step 3c, and computes the result vector $\mathbf{t}$ such that for $0 \leq i \leq n-1$:

$$\begin{aligned} t_i &= E_{pk_B}(w_i^{(B)} \cdot v_i^{(B)}) \cdot \prod_{1 \leq j \leq m} (e_i^{(A_j)})^{v_i^{(A_j)} \cdot w_i^{(A_j)}} \\ &= E_{pk_B}(r_i^{(B)} \cdot (w_i^{(B)} \cdot y_i^{(B)} + \sum_{1 \leq j \leq m} w_i^{(A_j)} \cdot y_i^{(A_j)})) \end{aligned}$$

Then the server sends $\mathbf{t}$ to client B, that carries out the final step, step 3e, unchanged. Note that in this protocol, even if $m-1$ clients collude, none can infer the set elements of the non-corrupted client, as the random polynomials $\omega_I^{(A_j)}$, picked by the server, are unknown to the clients.

7 Evaluation

We evaluate O-PSI by comparing its properties to those provided by other protocols that delegate PSI computation to a server. We also compare these protocols in terms of communication and computation complexity. Table 1 summarises the results.

Properties. The protocols in [12,13] require clients to interact with each other at setup. In [12] clients need to generate jointly the key of the pseudorandom permutation used to encode the datasets, while in [13] they need to jointly compute some parameters that are used in the encryption of their datasets. In contrast to

Table 1. Comparison of different delegated PSI protocols. Set cardinality and intersection cardinality are denoted by d and k respectively.

Property	**O-PSI**	**[12]**	**[13]**	**[14]**	**[15]**	**[16]**
Non-interactive setup	✓	×	×	✓	✓	✓
Hiding the intersection size from the server	✓	✓	✓	×	✓	×
Many set intersections without re-preparation	✓	×	×	×	×	×
Multiple clients	✓	✓	✓	✓	×	✓
Computation integrity verification	×	✓	×	×	×	✓
Communication complexity	$O(d)$	$O(d)$	$O(d^2)$	$O(d)$	$O(d^2)$	$O(k)$
Computation complexity	$O(d)$	$O(d)$	$O(d^2)$	$O(d^2)$	$O(d^2)$	$O(d)$

these protocols, in [14–16] and O-PSI the clients can independently prepare and outsource their private datasets. This is desirable in a cloud computing context as organizations and individuals can take advantage of the storage capabilities of the cloud and outsource their data at different points in time and without prior consideration of who is going to use them.

In a delegated PSI protocol, privacy should be maintained and the server should not learn anything about the intersection during the computation, including its cardinality. This is the case for the size-hiding variation of [12], protocols in [13,15], and O-PSI. However, as discussed in section 2 this is not the case for [14,16].

More interestingly, O-PSI is the only protocol in which clients can reuse their outsourced datasets on the server in multiple delegated PSI computations without the need to prepare their datasets for each computation, and computing PSI on the outsourced dataset multiple times does not reveal any information to the server. This is an important advantage in scenarios where outsourced datasets are expected to be used a lot of times, as it significantly reduces the overall communication and storage cost for the clients. This is not the case for any of the other protocols, because the clients either do not outsource their datasets, or need to re-encode them locally for each operation in order to prevent the server from inferring information about the intersection over time.

As we showed in section 6.2, O-PSI can be easily extended to support multiple clients. This is also the case for [12–14,16]. However, this is not possible for [15], as this requires an additional logical operation that is not supported by the homomorphic encryption scheme used.

O-PSI has been designed for the semi-honest security model and as a result does not consider the case where the server maliciously deviates from the protocol and computes the wrong result. This is a reasonable assumption in a cloud computing context where cloud providers are keen to preserve their reputation and this assumption is widely considered in the literature [13–15,23,24]. However [16] allows the client to verify the correctness of the results, while as we have seen in section 2, [12] can detect server misbehavior at an additional cost.

In conclusion, in contrast to other protocols, O-PSI has a unique combination of properties that make it particularly appealing for a cloud computing setting.

Communication Complexity. The communication complexity of O-PSI for the client who receives the result, client B, is $O(d)$, where d is the dataset size. This is because, client B sends to client A the $n = 2d + 1$ encrypted random values $E_{pk_B}(r_i^{(B)})$ for $0 \leq i \leq n-1$ (see step 3a). The communication complexity for client A, who authorizes the operation on its dataset, is $O(d)$, as for each of the n values it receives from client B it sends to the server $E_{pk_B}(r_i^{(B)} \cdot (r_i^{(A)})^{-1})$ (see step 3b). The communication complexity for the server is $O(d)$, as it sends to client B the result vector $\mathbf{t}$ of size n (see step 3d). Thus, the overall communication complexity of our protocol is $3n$ which is linear, $O(d)$, to the dataset size.

In [13] for each set intersection, the client engages in a two-round protocol, one round to upload its elements in the form of RSA ciphertexts to the server with $O(d)$ communication complexity, and another to interactively compute private set intersection with the server with $O(d^2)$ communication complexity. For the protocol in [15], the communication complexity is also quadratic $O(sd^2)$, where s is the number of hash functions used for the bloom filter, and the messages contain BGN encryption ciphertexts. On the other hand, the protocols in [12,14] have $O(d)$ communication complexity with messages containing symmetric key encryption ciphertexts, while the protocol in [16] has $O(k)$ complexity, where k is the intersection size.

In conclusion, similar to the most efficient protocols, O-PSI has linear communication complexity, however at an increased message size, which results from the additional dataset outsourcing properties and privacy guarantees that it provides.

Computation Complexity. We evaluate the computational cost of O-PSI by counting the number of exponentiation operations, as their cost dominates that of other operations. More specifically, client B performs n exponentiations to encrypt the random values in step 3.a, and needs another n exponentiations to decrypt the polynomial sent by the server in step 3.e. So, in total it carries out $2n$ exponentiations. Client A performs n exponentiations to enable the set intersection in step 3.b, while the server carries out n exponentiations to encrypt client B's dataset and n exponentiations to transform client A's dataset in step 3.d, a total of $2n$ exponentiations. It is interesting to note that using the point-value representation increases the overall storage costs at the server. However, the modest increase in storage brings a significant decrease in the computational costs, from $O(d^2)$ (when using encrypted coefficients such as in [6]) to $O(d)$. In total O-PSI involves $5n$ exponentiations. Hence, its computation complexity is linear to the size of the dataset, $O(d)$.

The semi-honest variant of the protocol in [12] also has linear complexity $O(d)$, as the client computing the result and the server invoke the pseudorandom permutation (PRP) d times, while the other client invokes the PRP, $2d$ times. On the other hand, the computational overhead in [13] is quadratic $O(d^2)$, as it involves a joint PSI protocol (plus public key encryption of the dataset elements). The protocol in [15] also has quadratic complexity, as it involves $O(d^2)$ BGN public key encryption operations. In [14] the client performs $O(d)$ modular additions, while the server carries out $O(d^2)$ operations to compare the

expanded sets of the users. Finally, the protocol in [16] is based on bilinear maps and requires $6d$ pairings at the server side and $2k$ exponentiations at the client side, resulting in $O(d)$ and $O(k)$ computation complexity at the server and client side respectively.

In conclusion, similar to the most efficient protocols, due to the use of polynomials in point-value form, O-PSI incurs only linear computational costs. However, the additional properties it provides come at the cost of more costly exponentiation operations.

8 Conclusions and Future Work

In this paper we have presented O-PSI, a protocol that allows clients to outsource their private datasets and delegate PSI computation to a server. A key building block of O-PSI is a novel representation of sets as polynomials in point-value form. The protocol allows clients to independently prepare and outsource their private datasets, while allowing, with the clients' permission, the server to compute multiple set intersections without revealing any information about the result or the sets, and no need for re-preparation of the sets. O-PSI has been shown to be secure in the semi-honest model, and has linear communication and computation complexity, with respect to the size of the datasets. O-PSI can be easily extended to support multiple clients and dataset integrity verification. As a result, O-PSI is a scalable protocol particularly suited for cloud computing environments. In the future, we plan to investigate how O-PSI can be extended to support additional set operations like set union or subset. We also plan to explore how clients can update their sets without the need to fully re-encode them, and verify the integrity of any computation.

Acknowledgments. We would like to thank the anonymous reviewers. Aydin Abadi is supported by a EPSRC Doctoral Training Grant studentship.

References

1. Fiore, D., Gennaro, R., Pastro, V.: Efficiently verifiable computation on encrypted data. In: 21st ACM Conference on Computer and Communications Security, Scottsdale, AZ, USA, pp. 844–855 (2014)
2. Backes, M., Fiore, D., Reischuk, R.M.: Verifiable delegation of computation on outsourced data. In: 20th ACM Conference on Computer and Communications Security, Berlin, Germany, pp. 863–874 (2013)
3. Agrawal, R., Srikant, R.: Privacy-preserving data mining. ACM Sigmod. Record **29**(2), 439–450 (2000)
4. Cristofaro, E.D., Tsudik, G.: Practical private set intersection protocols with linear complexity. In: 14th International Conference on Financial Cryptography and Data Security, pp. 143–159 (2010)
5. Freedman, M.J., Nissim, K., Pinkas, B.: Efficient Private Matching and Set Intersection. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 1–19. Springer, Heidelberg (2004)

6. Kissner, L., Song, D.: Privacy-Preserving Set Operations. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 241–257. Springer, Heidelberg (2005)
7. De Cristofaro, E., Kim, J., Tsudik, G.: Linear-Complexity Private Set Intersection Protocols Secure in Malicious Model. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 213–231. Springer, Heidelberg (2010)
8. Dong, C., Chen, L., Wen, Z.: When private set intersection meets big data: an efficient and scalable protocol. In: 20th ACM Conference on Computer and Communications Security, pp. 789–800 (2013)
9. Pinkas, B., Schneider, T., Zohner, M.: Faster private set intersection based on OT extension. In: 23rd USENIX Security Symposium, San Diego, CA, USA, USENIX (2014)
10. Papamanthou, C., Tamassia, R., Triandopoulos, N.: Optimal Verification of Operations on Dynamic Sets. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 91–110. Springer, Heidelberg (2011)
11. Canetti, R., Paneth, O., Papadopoulos, D., Triandopoulos, N.: Verifiable set operations over outsourced databases. In: 17th IACR International Conference on Theory and Practice of Public-Key Cryptography, pp. 113–130 (2014)
12. Kamara, S., Mohassel, P., Raykova, M., Sadeghian, S.: Scaling Private Set Intersection to Billion-Element Sets. In: Christin, N., Safavi-Naini, R. (eds.) FC 2014. LNCS, vol. 8437, pp. 193–213. Springer, Heidelberg (2014)
13. Kerschbaum, F.: Collusion-resistant outsourcing of private set intersection. In: 27th ACM Symposium on Applied Computing, Riva, Trento, Italy, pp. 1451–1456 (2012)
14. Liu, F., Ng, W.K., Zhang, W., Giang, D.H., Han, S.: Encrypted set intersection protocol for outsourced datasets. In: IEEE International Conference on Cloud Engineering, IC2E 2014, pp. 135–140. IEEE Computer Society, Washington, DC (2014)
15. Kerschbaum, F.: Outsourced private set intersection using homomorphic encryption. In: 7th ACM Symposium on Information, Compuer and Communications Security, ASIACCS 2012, Seoul, Korea, May 2–4, pp. 85–86 2012 (2012)
16. Zheng, Q., Xu, S.: Verifiable delegated set intersection operations on outsourced encrypted data. IACR Cryptology ePrint Archive, 178 (2014)
17. Goldreich, O.: The Foundations of Cryptography, vol. 2. Basic Applications. Cambridge University Press (2004)
18. Stefanov, E., Shi, E.: Multi-cloud oblivious storage. In: 20th ACM Conference on Computer and Communications Security, Berlin, Germany, pp. 247–258 (2013)
19. Raykova, M., Vo, B., Bellovin, S.M., Malkin, T.: Secure anonymous database search. In: First ACM Cloud Computing Security Workshop, Chicago, IL, USA, pp. 115–126 (2009)
20. Paillier, P.: Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, p. 223. Springer, Heidelberg (1999)
21. Aho, A.V., Hopcroft, J.E.: The Design and Analysis of Computer Algorithms, 1st edn. Addison-Wesley Longman Publishing Co., Inc., Boston (1974)
22. Ateniese, G., Burns, R.C., Curtmola, R., Herring, J., Kissner, L., Peterson, Z.N.J., Song, D.X.: Provable data possession at untrusted stores. In: 14th ACM Conference on Computer and Communications Security, pp. 598–609 (2007)
23. Wang, C., Ren, K., Wang, J.: Secure and practical outsourcing of linear programming in cloud computing. In: 30th IEEE International Conference on Computer Communications, Shanghai, China, pp. 820–828 (2011)
24. Hahn, F., Kerschbaum, F.: Searchable encryption with secure and efficient updates. In: 21st ACM Conference on Computer and Communications Security, Scottsdale, AZ, USA, pp. 310–320 (2014)