Efficient Delegated Private Set Intersection on Outsourced Private Datasets

Aydin Abadi, Sotirios Terzis, Roberto Metere, Changyu Dong

Abstract—Private set intersection (PSI) is an essential cryptographic protocol that has many real world applications. As cloud computing power and popularity have been swiftly growing, it is now desirable to leverage the cloud to store private datasets and delegate PSI computation to it. Although a set of efficient PSI protocols have been designed, none support outsourcing of the datasets and the computation. In this paper, we propose two protocols for delegated PSI computation on outsourced private datasets. Our protocols have a unique combination of properties that make them particularly appealing for a cloud computing setting. Our first protocol, O-PSI, satisfies these properties by using additive homomorphic encryption and point-value polynomial representation of a set. Our second protocol, EO-PSI, is mainly based on a hash table and point-value polynomial representation and it does not require public key encryption; meanwhile, it retains all the desirable properties and is much more efficient than the first one. We also provide a formal security analysis of the two protocols in the semi-honest model and we analyze their performance utilizing prototype implementations we have developed. Our performance analysis shows that EO-PSI scales well and is also more efficient than similar state-of-the-art protocols for large set sizes.

Index Terms—Private Set Intersection, Secure Computation, Cloud Computing

1 Introduction

Private set intersection (PSI) is a cryptographic protocol that allows parties to compute the intersection of their datasets without revealing anything about the datasets beyond the intersection [2]. PSI has a range of real-world applications including privacy-preserving data mining [3], like scenarios where mutually distrusting companies can find out common customers for joint offers without sharing their whole customer data, or ones where social welfare organizations can identify common benefits recipients while protecting the privacy of their beneficiaries; or even homeland security [4], allowing security agencies to find airline passengers in no-fly lists without having access to the whole passenger list or revealing their no-fly list. Also, PSI can be utilized as a sub-routine in larger privacy-preserving computations such as relationship path discovery in social networks [5], botnet detection [6], etc. Due to the importance of PSI, researchers have designed numerous PSI protocols (see section 2). Traditionally, PSI protocols are designed for scenarios in which data owners interact directly with each other using locally stored datasets and jointly compute the set intersection. However, the emergence of cloud computing calls for a change.

Cloud computing offers flexible and cost effective storage and computation resources to clients and has been attracting the attention of individuals and businesses as a vital enabling technology [7]. A report by the IBM Institute for Business Value in 2012 [1] found that cloud computing is driving business innovation along a number of dimensions, with its ability to enable increased collaboration with external partners and its cost advantages as the most important objectives for business adoption. Organizations have been keen to adopt cloud computing in order to reap the benefits it promises. A 2016 RightScale report [2] found that 95% of organizations surveyed are running applications or experimenting with the cloud. In general, “surveys show that more than half of all enterprises consider the cloud to be an essential part of their business models and are willing to devote 50% or more of their IT budget to the cloud” [8], while IDC says that two-thirds of enterprise IT spending will be cloud based by 2020 [3].

Interestingly public cloud adoption rates range between 85% to 90% depending on the survey [8], while according to the RightScale report, use of public clouds has increased with 17% of enterprises surveyed now having more than 1,000 VMs, up from 13% in 2015. At the same time, Forrester analyst Dave Bartoletti has found that enterprises are now looking at cloud as a viable place to run core business applications, with several companies having become more comfortable hosting critical software in the public cloud, a trend he expects to continue with a heavier reliance on public cloud providers [4].

Although certain benefits have proved harder to realize, like reduction of IT costs and IT complexity, improvement of IT team efficiency, and to a lesser extent increase in business agility [9], enterprises report as positive outcomes of cloud adoption amongst others enhancement of the general business model and increased

O-PSI was introduced in a paper that appears in the Proceedings of the 30th International Conference on ICT Systems Security and Privacy Protections (SEC 2015), pp. 3 – 17 [1].

- *Aydin Abadi and Sotirios Terzis are with the Department of Computer and Information Sciences, University of Strathclyde, Glasgow, UK.*
  Email: {aydin.abadi, sotirios.terzis}@strath.ac.uk
- *Roberto Metere and Changyu Dong are with the School of Computing Science, Newcastle University, Newcastle Upon Tyne, UK. The work was done when the authors were at the University of Strathclyde.*
  Email: {R.Metere2, changyu.dong}@newcastle.ac.uk.

[1] http://www-935.ibm.com/services/us/gbs/thoughtleadership/ibv-power-of-cloud.html

[2] http://assets.rightscale.com/uploads/pdfs/RightScale-2016-State-of-the-Cloud-Report.pdf

[3] http://talkincloud.com/cloud-computing-research/doyle-report-idc-says-two-thirds-enterprise-it-spending-will-be-cloud-based

[4] http://www.cio.com/article/3137946/cloud-computing/6-trends-that-will-shape-cloud-computing-in-2017.html

productivity from application users and business user groups [8]. Moreover, in the RightScale report participants identified faster access to infrastructure (62%), greater scalability (58%), higher availability (52%), and faster time to market (52%) as significant cloud benefits that grow with cloud adoption maturity. These advantages mean that according to the report, participants' cloud initiatives involve moving more workloads to the cloud (57% and 35% for enterprises and small and medium-size businesses respectively), expanding public cloud use (46% and 38%), and implementing a cloud first strategy (44% and 29%). In addition to this, public cloud growth is expected to outpace private cloud growth [5], with reports showing strong growth in public cloud workloads while on-premise ones fall, with both business-critical and non-critical workloads in the public cloud doubling over the next two years.

In a context where organizations embrace public clouds for core and mission-critical functions and reap clear business benefits beyond cost reductions from the exploitation not only of their storage but also their computation capabilities, the interest in taking advantage of these capabilities securely has been growing (e.g. [10], [11], [12], [13], [14]). Several research efforts by industry and academia have been directed towards delegated PSI protocols to realize organizational objectives [15], [16], [17], [18], [19], [20]. However, designing a PSI protocol that allows delegation of storage and computation to the cloud is not an easy task. There are several significant differences between this delegated PSI scenario and the traditional PSI case.

The first major difference is in the security model. In traditional PSI, two parties run an interactive protocol. Although they do not trust each other, they fully trust their local computational resources e.g. data storage, hardware and software. In delegated PSI, data storage and computation are now outsourced to a cloud server. The server is run and managed by an external party whose interests may not fully align with those of its clients and it may violate, intentionally or accidentally, data privacy agreements. So, it is difficult for the clients to fully trust the cloud with their sensitive data. Ideally, the untrusted cloud server should be able to carry out computation over outsourced datasets that belong to the clients, but should not learn anything about the stored datasets or the computation result. Designing a protocol for this model is more challenging because security has to be guaranteed not only against the other parties, as in the original PSI model, but also against the additional untrusted cloud server. As we show in section 2, quite a few protocols designed for the delegated PSI scenario to date actually have security problems and leak information to the server. Also, PSI computation involves datasets belonging to different parties. In traditional PSI, each party has full control over their own datasets. In delegated PSI, datasets are outsourced, and clients have to delegate their control to the cloud server. So, it is necessary to have an enforceable authorization mechanism such that the computation can only take place if all data owners agree.

The second major difference is in the computation model. At the center of cloud computing is the concept of outsourcing, as a result of which several new requirements arise. One requirement is that clients should not have to maintain a local copy after outsourcing their datasets. Otherwise, the clients will lose out on some of the cost benefits that use of cloud resources enables. Also, in order to facilitate collaboration with others, clients should be able to outsource their datasets once and use them for many PSI computations, rather than downloading and re-encoding the datasets for each computation. Otherwise, it would be better for clients to just keep a local copy of the datasets. In reality a cloud provider may serve many clients who may not know each other. Thus each client should be able to outsource its datasets independently without knowing anything about other parties' data or having to negotiate, for example, a shared key, with other parties. This requirement seems trivial but turns out to be quite challenging when the server is not fully trusted. To prevent the server from knowing the outsourced datasets, they have to be encrypted, and requiring clients to outsource their data independently, the datasets would be encrypted under independent keys. The server, then, needs to use ciphertexts encrypted under independent keys when computing PSI, which is a highly non-trivial task.

In this paper, we present two protocols for delegated PSI on outsourced private datasets. Our first protocol, O-PSI, is based on additive homomorphic encryption and point-value set representation. The protocol lets clients independently outsource their datasets by representing them as blinded polynomials. To achieve delegated PSI computation, homomorphic encryption is used to "switch" blinding factors so that the outsourced datasets blinded under different blinding keys can now be combined in the computation process. The protocol ensures that intersections can only be computed with the permission of all the clients and that the result (i.e. the intersection and its cardinality) will be protected from the cloud. The protocol also allows the datasets to be used securely an unlimited number of times without the need to secure them again. Although O-PSI has all the desirable properties, it is somewhat inefficient, as it requires costly homomorphic encryption (operations) which has a major impact on its performance. To mitigate this problem, we propose a more efficient protocol, EO-PSI, that preserves all O-PSI's desirable characteristics, while requires no public key encryption or exponentiation operations. The protocol also lets clients outsource their datasets by representing them as blinded polynomials. However, by changing the way the blinding is done and the interaction between the clients, the protocol no longer needs to "switch" blinding factors in order to combine the outsourced datasets in the computation process. We further improve the protocol performance by leveraging hash tables. As a result, EO-PSI is 1 - 2 orders of magnitude faster than O-PSI. We also provide a formal security analysis of the two protocols, and analyze their performance based on prototype implementations we have developed. Our performance analysis shows that EO-PSI scales well and it performs better than not only O-PSI but also other similar state-of-the-art protocols when the dataset is large.

2 Related Work

Private Set Intersection (PSI) was initially introduced in [2]. Following that many protocols such as [4], [21], [22], [23], [24], [25], [26], [27] were proposed. Among them, [21] provides a number of protocols supporting further private set operations based on additive homomorphic encryption and polynomial representation of sets. In [4], [22], the first PSI protocols with linear complexity (in the semi-honest and malicious models respectively) were proposed. In addition, [23], [24] proposed PSI protocols that allow result recipients to hide their set size from the other party during the computation of the intersection, while [24] also proposed protocols that output only the cardinality of the intersection. More recently, some efficient protocols, like [25], [26], [27], have

[5] http://talkincloud.com/cloud-computing-research/public-cloud-growth-outpace-private-cloud-next-12-months-report

been proposed. The protocols in [25] use Bloom filters, secret sharing and oblivious transfer to offer efficient PSI. Later on, [26] extended [25] by using hash tables and a more efficient oblivious transfer extension protocol for better efficiency. Recently, [27] further improved the efficiency of [25] by utilizing permutation-based hashing. Nevertheless, all these regular PSI protocols are interactive, which means clients jointly compute the intersection using locally available datasets. In general, they do not support outsourcing of the data and the computation to a third party (e.g. the cloud) without non-trivial modifications. For example in [23], both parties can send encrypted sets to the cloud and let the cloud compute the intersection. However, by doing so the cloud learns the cardinality of the intersection. Also, the parties must re-encrypt their data if they want to compute another intersection, otherwise the cloud can learn even more information about the parties' sets.

On the other hand, a number of PSI protocols that let clients delegate the computation to a server have been proposed in [15], [16], [17], [18], [19], [20]. The protocols in [17] allow clients to outsource their sets to a server by hashing each element and then adding a random value to it. In this protocol, each time the computation is delegated, every client needs to download an encrypted vector whose size is equal to the client's set size. Also, each element in the vector has the same size as the elements of the outsourced set. This is equivalent to the case where every client first downloads its outsourced set, prepares and uploads it before the cloud computes the result. Moreover, this protocol leaks to the server the cardinality of the intersection. Intersection cardinality is a widely used feature in data mining and could enable the server to infer a lot of things without knowing the content of the intersection. Thus from a privacy point of view, it should not be leaked. Additionally, due to 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 permission. Furthermore, in the protocol, value r is used as a one-time pad multiple times. However, according to its definition it must be used only once [28]. This approach is not secure and allows the cloud to figure out the hash values of each client's set elements. So, the protocol is not fully private. In [19], [20] clients also can delegate the computation to a server. In these protocols, a client encrypts his data and outsources them to the server. Both protocols require a trusted third party to initialize the public and private keys for the clients. Moreover, the schemes in [19], [20] suffer from the aforementioned problems (i.e. the cloud can learn whether two sets have common elements without the clients consent and leaks the intersection cardinality) thus both are not fully private. The protocol proposed in [16] 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. But, 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 [18] 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. 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, the protocol does not support data outsourcing. Another protocol that delegates computation to a server is proposed in [15]. The protocol is efficient, and is based on a pseudorandom permutation (PRP) whose key is generated jointly by the clients at setup. Nonetheless, the protocol requires the clients to interact with each other before delegating the computation and also the delegation is one-off.

To sum up, none of the above protocols allows clients to fully delegate PSI computation to the cloud without the need to either maintain the sets locally or re-encode and re-upload the sets for each set intersection computation while protecting the privacy of both the sets and the intersection. In other words, neither of them supports secure *delegated PSI on outsourced private datasets*. As a result, none of them is particularly suitable for a cloud computing setting. A comparison of our protocols with existing protocols is provided in section 7.

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 at a time and follows the protocol specification exactly. But, it may try to learn more information about the other party's input. The definitions and model are according to [28].

In a delegated PSI protocol, three parties are involved: a cloud C, and two clients A and B. We assume the cloud does not collude with A or B. The non-colluding assumption is widely used in the literature [15], [29], [30]. 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 $\Lambda, S^{(A)}$ and $S^{(B)}$ belonging to C, A and B respectively, the function outputs nothing to C 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 j^{th} message that it received.

Definition. *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_C, 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:*

$$\mathsf{Sim}_C(\Lambda, \Lambda) \overset{c}{\equiv} \mathsf{View}_C^{\pi}(\Lambda, S^{(A)}, S^{(B)})$$

$$\mathsf{Sim}_A(S^{(A)}, \Lambda) \overset{c}{\equiv} \mathsf{View}_A^{\pi}(\Lambda, S^{(A)}, S^{(B)})$$

$$\mathsf{Sim}_B(S^{(B)}, f_{\cap}(S^{(A)}, S^{(B)})) \overset{c}{\equiv} \mathsf{View}_B^{\pi}(\Lambda, S^{(A)}, S^{(B)})$$

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 [31]. It works as follows:

Key Generation: Choose two random large primes q_1 and q_2 according to a given security parameter, and set $N = q_1 \cdot q_2$. Let u be the Carmichael value of N, i.e. $u = lcm(q_1 - 1, q_2 - 1)$ 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 Representing Sets by Polynomials

Polynomial representation of sets was introduced in [2] and is widely used [21], [32]. In this representation, set elements are represented as elements in a finite field $\mathbb{F}_p$ and sets are represented as polynomials over the field. For the universe of set elements, $\mathcal{U}$, we define a public finite field $\mathbb{F}_p$ that is big enough to encode all elements in $\mathcal{U}$. For every $u_i \in \mathcal{U}$, we encode it as $s_i = u_i || \mathtt{G}(u_i)$, where $\mathtt{G}$ is a cryptographic hash function, so that given $s_j \in \mathbb{F}_p$ and $\mathtt{G}$'s output size, one can parse s_j into a and b, and check $b \overset{?}{=} \mathtt{G}(a)$. If $b = \mathtt{G}(a)$ then we say s_j is valid, otherwise, it is invalid. From now on we will use "set element" or simply "element" to refer to the encoded form of the element and "dummy element" to refer to a uniformly random element in $\mathbb{F}_p$. A set element and a dummy element can be easily distinguished because the probability that a random element in $\mathbb{F}_p$ has the correct structure and can pass the above check is negligible if $\mathtt{G}$ is a secure cryptographic hash function. A set S can be represented by a polynomial over $\mathbb{F}_p$: $\rho(x) = \prod_{i=1}^{|S|}(x - s_i)$, where s_i is a set element in S.

For two sets $S^{(A)}$ and $S^{(B)}$ represented by polynomials $\rho^{(A)}$ and $\rho^{(B)}$ respectively, polynomial $\rho^{(A)} \cdot \rho^{(B)}$ represents the set union, $S^{(A)} \cup S^{(B)}$, and $gcd(\rho^{(A)}, \rho^{(B)})$ represents the set intersection, $S^{(A)} \cap S^{(B)}$, where gcd stands for the greatest common divisor. For two degree d polynomials $\rho^{(A)}$ and $\rho^{(B)}$, and two degree d random polynomials $\gamma^{(A)}$ and $\gamma^{(B)}$ whose coefficients are picked uniformly at random from $\mathbb{F}_p$, it is proven in [21] that $\gamma^{(A)} \cdot \rho^{(A)} + \gamma^{(B)} \cdot \rho^{(B)} = \mu \cdot gcd(\rho^{(A)}, \rho^{(B)})$ where μ is a uniformly random polynomial. This means that if $\rho^{(A)}$ and $\rho^{(B)}$ are polynomials representing sets $S^{(A)}$ and $S^{(B)}$, then the polynomial $\beta = \gamma^{(A)} \cdot \rho^{(A)} + \gamma^{(B)} \cdot \rho^{(B)}$ contains only information about $S^{(A)} \cap S^{(B)}$ and no information about other elements in $S^{(A)}$ or $S^{(B)}$. Given polynomial β, to find the intersection, one can extract the polynomial's roots [6], and then consider the set of valid roots as the intersection. Since the computation which we use to obtain the intersection could introduce random roots, we need to encode the elements. In particular, the roots of the polynomial $\rho_C = \mu \cdot gcd(\rho^{(A)}, \rho^{(B)})$ come from both $gcd(\rho^{(A)}, \rho^{(B)})$ and μ. While the roots of $gcd(\rho^{(A)}, \rho^{(B)})$ are the intersection we want, the roots of μ are random elements that should be discarded. Since μ is a uniformly random polynomial, its roots should be uniformly random elements in $\mathbb{F}_p$, i.e. dummy elements. Thus, the encoding allows us to effectively eliminate the invalid roots.

3.4 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. Previous PSI protocols (e.g. [2], [21], [32]) using polynomial representation of sets, represent a polynomial as a vector of the polynomial's coefficients, i.e. they represent a degree d polynomial $\rho(x) = \sum_{i=0}^{d} a_i x^i$ as a vector $\vec{\boldsymbol{a}} = [a_0, ..., a_d]$. 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 this form is $O(d^2)$. In PSI protocols, this leads to significant computational overheads, especially when one polynomial needs to be encrypted and the polynomial multiplication has to be done homomorphically. Homomorphic multiplication operations are computationally expensive. Thus, those protocols using the coefficient-based polynomial representation are not scalable.

We solve this problem by representing the polynomials in another well-known form, point-value. A degree d polynomial $\rho(x)$ can be represented as a set of n ($n > d$) point-value pairs $\{(x_1, y_1), ..., (x_n, y_n)\}$ such that all x_i are distinct and $y_i = \rho(x_i)$ for $1 \leq i \leq n$. If x_i are fixed, we can omit them and represent polynomials as a vector $\vec{\boldsymbol{y}} = [y_1, ..., y_n]$. A polynomial in point-value form can be converted into coefficient form by polynomial interpolation [34], [35]. Given n pairs of (x_i, y_i), we can interpolate a regular coefficient-based polynomial $\zeta(x)$ of degree at most $n - 1$. To this end, we can use the modified (or improved) Lagrange formula:

$$\zeta(x) = \eta(x) \sum_{i=1}^{n} \frac{\psi_i}{x - x_i} \cdot y_i$$

where $\eta(x) = \prod_{i=1}^{n}(x - x_i)$ and $\psi_i = \frac{1}{\prod_{\substack{i=1 \\ i \neq k}}^{n}(x_i - x_k)}$.

We can add or multiply two polynomials by adding or multiplying their corresponding y-coordinates; for two degree d polynomials $\rho^{(A)}$ and $\rho^{(B)}$ represented in point-value form by two vectors $\vec{\boldsymbol{y}}^{(A)}$ and $\vec{\boldsymbol{y}}^{(B)}$, the polynomial $\rho^{(A)} + \rho^{(B)}$ can be computed as $(y_1^{(A)} + y_1^{(B)}, y_2^{(A)} + y_2^{(B)}, ..., y_n^{(A)} + y_n^{(B)})$, and the polynomial $\rho^{(A)} \cdot \rho^{(B)}$ can be computed as $(y_1^{(A)} \cdot y_1^{(B)}, y_2^{(A)} \cdot y_2^{(B)}, ..., y_n^{(A)} \cdot y_n^{(B)})$. Note, because the product of $\rho^{(A)} \cdot \rho^{(B)}$ is a polynomial of degree $2d$, $\rho^{(A)}$ and $\rho^{(B)}$ 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 $O(d)$ and this makes our protocols much more scalable.

3.5 Hash Tables

In our protocols, polynomial factorization is needed in order for the result recipient to obtain the intersection at the end of the computation. The complexity of polynomial factorization is quadratic in the degree of the polynomial being factorized. To improve performance, in EO-PSI we use hash tables to divide a large set into small subsets and represent each subset as a polynomial. This is a technique that has been used in several regular PSI protocols, e.g. [2], [27]. Intuitively, for a c-element set that is represented by a degree-c polynomial, the factorization cost is $O(c^2)$. If we break down the set into h roughly equal-sized

[6] To find the roots of a polynomial over a finite field, we can first factorize it to get a set of monic polynomials (see [33] for some algorithms), then find the monic degree-1 polynomials' roots.

subsets, then we will need to factor h polynomials of degree $\frac{c}{h}$. So, the total cost is reduced to $O(\frac{c^2}{h})$.

In general, hash tables in PSI protocols can be used as follows. First, the public parameters including a random hash function $\mathtt{H}$, the number of bins in the hash table and the bin's maximum size are picked. The number of bins in the hash table should be set such that given the maximum set cardinality, with a high probability each bin receives at most a specific number of elements (we will explain shortly how this can be done).

For the parties to compute the set intersection, each of them maps each set element s_i to the table by computing an address $j = \mathtt{H}(s_i)$, using the hash function whose output is modeled as a uniform random number. Then, it inserts s_i into the corresponding bin $\mathtt{HT}_j$. Because the hash function is deterministic if an element is in the intersection, both parties map it to the same bin. Therefore, a large set of elements can be broken down into a collection of smaller sets (the bins) and a PSI protocol can operate on each bin separately.

As mentioned above, we need to set parameters appropriately to ensure that the number of elements in each bin does not exceed a predefined upper bound. Given the maximum number of elements c and the bin's maximum size d, we can determine the number of bins by analyzing hash tables under the balls into bins model which has been extensively studied in the literature [36], [37].

Theorem* 1.** ***(Upper Tail in Chernoff Bounds) *Let X_i be a random variable defined as $X_i = \sum_{i=1}^{c} Y_i$, where $Pr[Y_i = 1] = p_i$, $Pr[Y_i = 0] = 1 - p_i$, and all Y_i are independent. Let μ be the expectation $\mathrm{E}[X_i] = \sum_{i=1}^{h} p_i$. Then:*

$$Pr[X_i > d = (1+\sigma)\cdot\mu] < \Big(\frac{e^{\sigma}}{(1+\sigma)^{(1+\sigma)}}\Big)^{\mu}, \forall\sigma > 0 \quad (1)$$

Note that in the balls and bins model the expectation is $\mu = \frac{c}{h}$. Inequality 1 provides a bound for the probability that bin i is overloaded. Since there are h bins, the probability that at least one of them is overloaded is bounded by the *union bound.*

$$\begin{aligned} Pr[\exists i, 1 \leq i \leq h : X_i > d] &\leq \sum_{i=1}^{h} Pr[X_i > d] \\ &\leq h\cdot\Big(\frac{e^{\sigma}}{(1+\sigma)^{(1+\sigma)}}\Big)^{\frac{c}{h}} \end{aligned} \quad (2)$$

Thus, when the probability and bin's maximum load are fixed, for any c number of elements (as the maximal number of elements that may be inserted into the hash table) we can set the number of bins using inequality 2. In Section 8, some concrete parameters are calculated for our experiments and are shown in Table 3.

3.6 Notation

We summarize our notation in Table 1.

4 O-PSI: Our First Protocol

In this section, we present O-PSI our first protocol for delegated private set intersection on outsourced private datasets.

4.1 An Overview of O-PSI

The interaction between parties in O-PSI is depicted in Fig. 1. At a high level, the protocol works as follows. First, each client independently prepares its dataset and then stores it as a blinded vector in the cloud. Since the vector is blinded the cloud cannot figure out the client's set. Later on, when client B gets interested in the intersection of its own outsourced dataset and client A's outsourced dataset, it obtains client A's permission by sending a message to it. If A agrees, then they jointly compute a set of encrypted values and send them to the cloud. The encrypted values are used by the cloud to "switch" the blinding factors of A's dataset, which then allows the set intersection to be computed correctly. After that, the cloud uses client A's message and the clients' outsourced datasets to generate an encrypted polynomial encoding the intersection; and then it sends the encrypted polynomial to client B. At the end of the protocol, when client B receives the polynomial, it decrypts it and extracts the polynomial's roots that are the set intersection. The protocol is described below. We will explain the rationale behind the protocol design after the protocol description.

TABLE 1
Table of notation.

Notation		Description
Generic	$\mathcal{U}$	The universe of set elements.
	$\vec{v}$	Vector v.
	$\lvert\vec{v}\rvert = c$	Vector of size c.
	$\mathbb{F}_p$	Finite field of order p.
	$a\|\|b$	a is concatenated with b.
	$(v_i)^{-1}$ and $-v_i$	The multiplicative and additive inverse of value v_i respectively.
	$e^{(I)}$	Value e belongs to client I.
	$\mathtt{PRF}(.)$	Pseudorandom function $\mathtt{PRF}$: $\{0,1\}^m \times \{0,1\}^l \rightarrow \mathbb{F}_p$.
	l, m	The key bit-length (i.e. security parameter) and message bit-length respectively.
	$\vec{o}^{(I)}$	A vector containing client I's outsourced blinded data.
	$\tau^{(I)}(x)$	The polynomial representing client I's set.
	$\omega^{(I)}(x)$	(Pseudo)random polynomial for client I.
In O-PSI	$\vec{e}^{(B)}$	The vector of encrypted pseudorandom values sent by client B to A.
	$\vec{e}^{(A)}$	The vector of encrypted pseudorandom values sent by client A to the cloud.
	$\vec{t}$	Vector of blinded y-coordinates (i.e. the result) sent by the cloud to client B.
	$E_{pk_I}(v_i)$	Value v_i is encrypted using client I public key.
	$D_{sk_I}(v_i)$	Value v_i is decrypted using client I secret key.
In EO-PSI	tk	Temporary key.
	$mk^{(I)}$	Master key for client I.
	$\vec{t}_i$	The vector of blinded y-coordinates (i.e. the result) sent by the cloud to client B.
	$\mathtt{H}(.)$	Hash function whose output ranges over $[1, h]$.
	h	Hash table size or the number of bins.
	$\mathtt{HT}^{(I)}_j$	The j^{th} bin in hash table $\mathtt{HT}^{(I)}$.
	$s^{(I)}_i \rightarrow \mathtt{HT}^{(I)}_j$	element s_i is mapped to bin $\mathtt{HT}^{(I)}_j$.

4.2 O-PSI Protocol

Without loss of generality, first, we consider the two client case, where client A, client B and a cloud engage in the protocol.

a. **Cloud-Side Setup.** The cloud picks a public parameter c that is an upper bound of the set cardinality. The cloud constructs a finite field $\mathbb{F}_p$, where p is a large prime number. It also

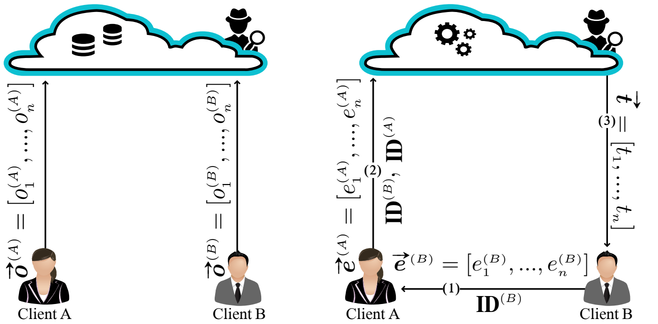
$\vec{o}^{(A)} = [o_1^{(A)}, ..., o_n^{(A)}]$
$\vec{o}^{(B)} = [o_1^{(B)}, ..., o_n^{(B)}]$
Client A
Client B
$\vec{e}^{(A)} = [e_1^{(A)}, ..., e_n^{(A)}]$
(2)
$\mathbf{ID}^{(B)}, \mathbf{ID}^{(A)}$
(3)
$\vec{t} = [t_1, ..., t_n]$
$\vec{e}^{(B)} = [e_1^{(B)}, ..., e_n^{(B)}]$
(1)
$\mathbf{ID}^{(B)}$
Client A
Client B

Fig. 1. The left-hand side figure: party interaction at data outsourcing phase in O-PSI; the right-hand side figure: party interaction at the computation delegation phase in O-PSI.

constructs a vector $\vec{x}$ containing $n = 2c+1$ distinct non-zero x_i values randomly picked from $\mathbb{F}_p$. It picks a pseudorandom function PRF: $\{0,1\}^m \times \{0,1\}^l \to \mathbb{F}_p$, which takes an l-bit key and m-bit message, and maps the message to an element in the field pseudorandomly. The cloud publishes the description of the field, the value n, the vector $\vec{x}$ and the pseudorandom function PRF.

b. **Client-Side Setup and Data Outsourcing.** Let client $I \in \{A, B\}$ have a set $S^{(I)}$, where $S^{(I)} \subset \mathcal{U}$ for some set universe $\mathcal{U}$ and $|S^{(I)}| \leq c$. Each client I performs the following:
   1) 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 PRF. All the keys are generated according to a given security parameter.
   2) Constructs a polynomial $\tau^{(I)}(x) = \prod_{i=1}^{|S^{(I)}|}(x - s_i^{(I)})$ that represents its set $S^{(I)}$. Represent $\tau^{(I)}(x)$ as point-value form, by evaluating it at every element x_i in $\vec{x}$. This yields a vector containing values $\tau^{(I)}(x_i)$, $1 \leq i \leq n$.
   3) Blinds every value $\tau^{(I)}(x_i)$. To do that, it generates a set of pseudorandom values (or blinding factors) $z_i^{(I)} = \mathtt{PRF}(k^{(I)}, i)$; next, computes $o_i^{(I)}$ as follows:

   $$1 \leq i \leq n\text{: } o_i^{(I)} = \tau^{(I)}(x_i) \cdot z_i^{(I)}$$

   At the end of this step, the set elements are represented as vector $\vec{o}^{(I)} = [o_1^{(I)}, ..., o_n^{(I)}]$.
   4) Sends vector $\vec{o}^{(I)}$ to the cloud.

c. **Set Intersection: Computation Delegation.** This phase starts when client B becomes interested in the intersection of its set and client A's set.
   1) Client B sends a message to client A. The message contains client B's ID, $\mathbf{ID}^{(B)}$, and a vector $\vec{e}^{(B)}$, whose elements are computed as follows:

   $$\forall i, 1 \leq i \leq n\text{: } e_i^{(B)} = E_{pk_B}(z_i^{(B)})$$

   where $z_i^{(B)} = \mathtt{PRF}(k^{(B)}, i)$ are the values used by client B to blind its polynomial in step b.3 above.
   2) Given client B's message, client A computes vector $\vec{e}^{(A)}$.
   $\forall i, 1 \leq i \leq n$:

   $$e_i^{(A)} = (e_i^{(B)})^{(z_i^{(A)})^{-1}} = E_{pk_B}(z_i^{(B)} \cdot (z_i^{(A)})^{-1})$$

   where $z_i^{(I)} = \mathtt{PRF}(k^{(I)}, i)$ for $I \in \{A, B\}$ are the values from step b.3.
   3) Client A sends $\vec{e}^{(A)}$, $\mathbf{ID}^{(A)}$, $\mathbf{ID}^{(B)}$, and **Compute** message to the cloud.

d. **Set Intersection: Cloud-Side Result Computation.**
   1) After receiving the **Compute** message from A, the cloud picks two degree c random polynomials $\omega^{(A)}(x)$ and $\omega^{(B)}(x)$ (whose coefficients are chosen from $\mathbb{F}_p$).
   2) The cloud fetches the clients outsourced datasets $\vec{o}^{(A)}$ and $\vec{o}^{(B)}$ and then computes vector $\vec{t}$ as below.
   $\forall i, 1 \leq i \leq n$:

   $$t_i = (e_i^{(A)})^{o_i^{(A)} \cdot \omega^{(A)}(x_i)} \cdot E_{pk_B}(\omega^{(B)}(x_i) \cdot o_i^{(B)})$$
   $$= E_{pk_B}(z_i^{(B)} \cdot (\omega^{(A)}(x_i) \cdot \tau^{(A)}(x_i) + \omega^{(B)}(x_i) \cdot \tau^{(B)}(x_i)))$$

   3) The cloud sends $\vec{t}$ to client B.

e. **Set Intersection: Client-Side Result Retrieval**
   1) Client B decrypts the elements in $\vec{t}$ and then removes the blinding factors. This yields vector $\vec{g}$ computed as follows:
   $1 \leq i \leq n$:

   $$g_i = D_{sk_B}(t_i) \cdot (z_i^{(B)})^{-1} = z_i^{(B)} \cdot (\omega^{(A)}(x_i) \cdot \tau^{(A)}(x_i) + \omega^{(B)}(x_i) \cdot \tau^{(B)}(x_i)) \cdot (z_i^{(B)})^{-1} = \omega^{(A)}(x_i) \cdot \tau^{(A)}(x_i) + \omega^{(B)}(x_i) \cdot \tau^{(B)}(x_i)$$

   2) It then interpolates the polynomial $\phi(x)$ using the point-value pairs (x_i, g_i) and considers the valid roots of $\phi(x)$ as the elements in the set intersection (see section 3.3).

Remark 1: In step a, the cloud publishes a vector $\vec{x}$ that has $2c+1$ elements, because the polynomial $\phi(x)$ in step e.2 is of degree $2c$ and at least $2c+1$ points are needed to interpolate it. Note that the elements in $\vec{x}$ are picked at random from $\mathbb{F}_p$ so the probability of x_i being a root of a client's polynomial is negligible.

Remark 2: In step b.3, if the client does not blind the evaluated polynomial and stores the values $\tau^{(I)}(x_i)$ directly on the cloud, then the cloud could use n pairs of $(x_i, \tau^{(I)}(x_i))$ to interpolate the client's polynomial. As a result, the client's set would be revealed to the cloud. Whereas, when they are blinded the cloud cannot learn anything about the client's set unless it knows the pseudorandom function key used by the client. The client blinds the values by multiplication; while multiplication cannot blind $\tau^{(I)}(x_i) = 0$. This is why we require the probability of $x_i \in \vec{x}$ being a root of a client's polynomial to be negligible.

Remark 3: The data stored in the cloud are independently blinded by its owner. Also, to compute the set intersection correctly, the blinding factors ($z_i^{(I)}$ in the protocol) must be eliminated at the end of the protocol. In step c.2, client A and B jointly compute the vector $\vec{e}^{(A)}$ that allows the cloud to obliviously "switch" A's blinding factors to B's. Accordingly, in step d.2, the cloud uses $\vec{e}^{(A)}$ to eliminate $z_i^{(A)}$ and replace it with $z_i^{(B)}$. The blinding factors $z_i^{(B)}$, later on in step e.1, can be eliminated by client B. What is more, since the values in $\vec{e}^{(A)}$ are encrypted and only client B knows the secret key, the cloud learns nothing in this process.

Remark 4: The client's original blinded dataset remains unchanged in the cloud. In fact, in step d.2, the cloud multiplies *a copy* of the client's blinded dataset by the vector of $\omega^{(I)}(x_i)$.

Remark 5: The only information that the cloud learns about the clients' datasets is the upper bound on the datasets cardinality (i.e. value c) that was initially set by itself. Thus, the cloud learns nothing about the exact number of the set elements and the intersection cardinality.

4.3 Multiple Clients O-PSI

With minor modifications, two-client O-PSI can be turned into m-client O-PSI, where $m > 2$. Below we outline how this can be done. In this case, the client interested in the intersection, client B, sends the same request (see step c.1 of the protocol) to all other clients, A_z, where $1 \leq z \leq y$ and $y = m - 1$. The protocol for each client A_z remains unchanged. For each client A_z, the cloud carries out step d.2 and it computes the result vector $\vec{\boldsymbol{t}}$ as follows:

$$\forall i, 1 \leq i \leq n\text{:}$$

$$t_i = E_{pk_B}(\omega^{(B)}(x_i) \cdot o_i^{(B)}) \cdot \prod_{z=1}^{y} (e_i^{(A_z)})^{o_i^{(A_z)} \cdot \omega^{(A_z)}(x_i)}$$

$$= E_{pk_B}(z_i^{(B)} \cdot (\omega^{(B)}(x_i) \cdot \tau^{(B)}(x_i) + \sum_{z=1}^{y} \omega^{(A_z)}(x_i) \cdot \tau^{(A_z)}(x_i)))$$

Then, the cloud sends $\vec{\boldsymbol{t}}$ to client B. Note that in this case, even if client B colludes with $y-1$ clients, it could not infer the set elements of the non-colluding client, as the random polynomials $\omega^{(A_z)}$ and $\omega^{(B)}$ are picked by the cloud, and are unknown to the clients.

5 A More Efficient Protocol, EO-PSI: Our Second Protocol

In this section, we introduce EO-PSI that preserves all O-PSI's desirable properties and is more efficient. EO-PSI improves O-PSI from two perspectives. First, unlike O-PSI, EO-PSI does not use any public key encryption that is computationally expensive. In O-PSI, the public key encryption is mainly used to prevent the cloud from eventually learning any information about the blinding factors (and set elements) during the cloud-side "switching" of the blinding factors, especially when the computation is delegated multiple times. Recall in O-PSI given the vector $\vec{\boldsymbol{e}}$ the cloud can "switch" one client's blinding factors to another's. In contrast, in EO-PSI no such "switching" is required. Therefore, no public key encryption is needed. In order to achieve this, we slightly change the way each client blinds its polynomial. In EO-PSI, instead of multiplying value $\tau(x_i)$ by a pseudorandom value, the client sums the value and the pseudorandom value. Moreover, the interaction between the clients is changed, in the sense that client A sends a message to both the cloud and client B when it authorizes the computation.

Second, EO-PSI allows each client to break down its original polynomial into smaller degree polynomials. This allows the result recipient to factorize a set of smaller degree polynomials rather than one of very large degree. As a result, it can find the roots of the polynomials (i.e. the set intersection) faster than in O-PSI. To achieve this, the protocol lets each client insert its elements into the bins of a (fixed-size) hash table.

5.1 An Overview of EO-PSI

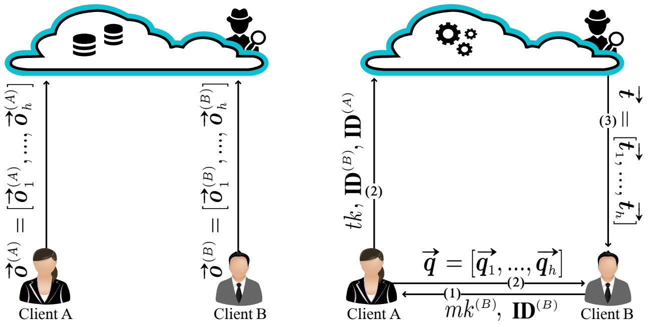
$\vec{\boldsymbol{o}}^{(A)} = [\vec{\boldsymbol{o}}_1^{(A)}, \ldots, \vec{\boldsymbol{o}}_h^{(A)}]$
$\vec{\boldsymbol{o}}^{(B)} = [\vec{\boldsymbol{o}}_1^{(B)}, \ldots, \vec{\boldsymbol{o}}_h^{(B)}]$
Client A
Client B
tk, $\mathbf{ID}^{(B)}$, $\mathbf{ID}^{(A)}$
(2)
(3)
$\vec{\boldsymbol{t}} = [\vec{\boldsymbol{t}}_1, \ldots, \vec{\boldsymbol{t}}_h]$
$\vec{\boldsymbol{q}} = [\vec{\boldsymbol{q}}_1, \ldots, \vec{\boldsymbol{q}}_h]$
(2)
(1)
$mk^{(B)}$, $\mathbf{ID}^{(B)}$
Client A
Client B

Fig. 2. The left-hand side figure: party interaction at data outsourcing phase in EO-PSI; the right-hand side figure: party interaction at the computation delegation phase in EO-PSI.

The interaction between parties in EO-PSI is depicted in Fig. 2. What follows is a high-level description of the protocol. First, each client inserts its set elements into the hash table. Then, it represents the set of elements in each bin of the hash table as a blinded point-value polynomial and sends the polynomials to the cloud. When client B becomes interested in the intersection of its own set and client A's set, it obtains the client's permission by sending a message to it. If client A agrees, it generates a set of vectors and sends them to client B. The vectors help client B to unblind the cloud's response. Client A also sends a key for a pseudorandom function to the cloud. The key is generated on the fly and is used only in this execution of the protocol. The cloud uses the key and the outsourced datasets to compute a set of blinded polynomials, and sends them to client B. Given these polynomials and client A's message, client B unblinds them and retrieves the intersection of the sets.

5.2 EO-PSI Protocol

Similarly, here first we consider the two client case, where client A, client B and a cloud engage in the protocol.

a. **Cloud-Side Setup.** The cloud sets the parameters for a hash table. It sets c as the upper bound of the set cardinality, d as the maximum load that a bin in the hash table can have, and h as the total number of bins in the hash table. Moreover, it chooses a cryptographic hash function, $\mathtt{H}$. Then, the cloud constructs a finite field $\mathbb{F}_p$, where p is a large prime number. It also constructs a vector $\vec{\boldsymbol{x}}$ containing $n = 2d + 1$ distinct non-zero x_i values randomly picked from $\mathbb{F}_p$. It picks a pseudorandom function $\mathtt{PRF} : \{0, 1\}^m \times \{0, 1\}^l \rightarrow \mathbb{F}_p$, which takes an *l*-bit key and *m*-bit message, and maps the message to an element in the field pseudo-randomly. The cloud publishes the hash table parameters, the description of the field, the value n, the vector $\vec{\boldsymbol{x}}$, the pseudorandom function $\mathtt{PRF}$ along with the hash function $\mathtt{H}$.
b. **Client-Side Setup and Data Outsourcing.** Let client $I \in \{A, B\}$ have a set $S^{(I)}$, where $S^{(I)} \subset \mathcal{U}$ and $|S^{(I)}| \leq c$. Each client I performs the following:
   1) Given the hash table parameters, generates a hash table and inserts its set elements into it, as below.

      $$\forall s_i^{(I)} \in S^{(I)}\text{: } \mathtt{H}(s_i^{(I)}) = j\text{, then } s_i^{(I)} \rightarrow \mathtt{HT}_j^{(I)}$$

      where $1 \leq j \leq h$.
   2) Assigns a key (for the pseudorandom function) to each bin in the hash table by picking a master key $mk^{(I)}$, and generating h pseudorandom values (or keys):

      $$\forall j, 1 \leq j \leq h\text{: } k_j^{(I)} = \mathtt{PRF}(mk^{(I)}, j).$$

   3) For every bin $\mathtt{HT}_j^{(I)}$, if it has less than d set elements, pads it with dummy (or random) elements, $r_{j,i}$, to d elements. Then, encodes the bin elements as below.
      a) Constructs a polynomial representing the elements in the bin.

$$\tau_j^{(I)}(x) = \prod_{i=1}^{d}(x - e_i^{(I)})$$

where $e_i^{(I)} \in \mathtt{HT}_j^{(I)}$, $e_i^{(I)} = s_i^{(I)}$ or $e_i^{(I)} = r_{j,i}$.

b) Represents $\tau_j^{(I)}(x)$ in point-value form, by evaluating it at every element $x_i \in \vec{\boldsymbol{x}}$. This yields a vector containing values $\tau_j^{(I)}(x_i)$, $1 \leq i \leq n$.

c) Blinds every value $\tau_j^{(I)}(x_i)$. To do so first generates a pseudorandom value $z_{j,i}^{(I)} = \mathtt{PRF}(k_j^{(I)}, i)$, where key $k_j^{(I)}$ was generated in step b.2. After that, computes $o_{j,i}^{(I)}$ as follows.

$\forall j, 1 \leq j \leq h$ and $\forall i, 1 \leq i \leq n$:

$$o_{j,i}^{(I)} = \tau_j^{(I)}(x_i) + z_{j,i}^{(I)}$$

At the end of this step, the elements in bin $\mathtt{HT}_j$ are represented as the vector $\vec{\boldsymbol{o}}_j^{(I)} = [o_{j,1}^{(I)}, ..., o_{j,n}^{(I)}]$.

4) Sends $\vec{\boldsymbol{o}}^{(I)} = [\vec{\boldsymbol{o}}_1^{(I)}, ..., \vec{\boldsymbol{o}}_h^{(I)}]$ to the cloud.

c. **Set Intersection: Computation Delegation.** This phase starts when client B wants the intersection of its set and client A's set.

1) Client B sends $mk^{(B)}$ and its id, $\mathbf{ID}^{(B)}$, to client A.

2) Given $mk^{(B)}$ and $mk^{(A)}$ client A regenerates $k_j^{(I)}$ (see step b.2) where $1 \leq j \leq h$ and $\forall I, I \in \{A, B\}$.

3) Client A assigns three fresh keys to each bin $\mathtt{HT}_j$. To do that, first, it picks a temporary key tk and then carries out the following.

a) It uses the key, tk, to generate three pseudorandom values k_t.

$$\forall t, 1 \leq t \leq 3\text{: } k_t = \mathtt{PRF}(tk, t).$$

b) It uses each k_t to compute h pseudorandom values.

$\forall j, 1 \leq j \leq h$: $k_{1,j} = \mathtt{PRF}(k_1, j)$, $k_{2,j} = \mathtt{PRF}(k_2, j)$, $k_{3,j} = \mathtt{PRF}(k_3, j)$.

4) For each bin $\mathtt{HT}_j$ client A uses key $k_{1,j}$ to generate a set of pseudorandom values $a_{j,i}$.

$$\forall i, 1 \leq i \leq n\text{: } a_{j,i} = \mathtt{PRF}(k_{1,j}, i).$$

Also, it uses key $k_{2,j}$ and $k_{3,j}$ to generate two degree d pseudorandom polynomials $\omega_j^{(A)}(x)$ and $\omega_j^{(B)}(x)$ for that bin.

5) For each bin $\mathtt{HT}_j$ client A regenerates the pseudorandom values $z_{j,i}^{(A)} = \mathtt{PRF}(k_j^{(A)}, i)$ and $z_{j,i}^{(B)} = \mathtt{PRF}(k_j^{(B)}, i)$ using the keys it derived in step c.2. Then, it computes vector $\vec{\boldsymbol{q}}_j$ as follows. $\forall i, 1 \leq i \leq n$:

$$q_{j,i} = z_{j,i}^{(A)} \cdot \omega_j^{(A)}(x_i) + z_{j,i}^{(B)} \cdot \omega_j^{(B)}(x_i) + a_{j,i}$$

Vectors $\vec{\boldsymbol{q}}_j$ allow client B to remove the blinding factors from the cloud's response without learning the pseudorandom polynomials.

6) Client A sends $\vec{\boldsymbol{q}} = [\vec{\boldsymbol{q}}_1, ..., \vec{\boldsymbol{q}}_h]$ to client B. Also, client A sends the key tk (generated in step c.3), $\mathbf{ID}^{(A)}, \mathbf{ID}^{(B)}$, and **Compute** message to the cloud.

d. **Set Intersection: Cloud-Side Result Computation.**

1) Given the key tk, the cloud derives the three keys $k_{1,j}$, $k_{2,j}$ and $k_{3,j}$ for each bin $\mathtt{HT}_j$, where $1 \leq j \leq h$ (see steps c.3a and c.3b)

2) Using the keys generated in the previous step, the cloud regenerates the set of pseudorandom values $a_{j,i}$ ($\forall i, 1 \leq i \leq n$) and the two pseudorandom polynomials $\omega_j^{(A)}(x)$ and $\omega_j^{(B)}(x)$ for each bin $\mathtt{HT}_j$, where $1 \leq j \leq h$ (see step c.4).

3) The cloud computes the result as follows. First, it fetches the clients' outsourced datasets $\vec{\boldsymbol{o}}_j^{(A)}$ and $\vec{\boldsymbol{o}}_j^{(B)}$ in each bin $\mathtt{HT}_j$. Next, it computes the result vector $\vec{\boldsymbol{t}}_j$ for that bin as below.

$\forall j, 1 \leq j \leq h$ and $\forall i, 1 \leq i \leq n$:

$$t_{j,i} = o_{j,i}^{(A)} \cdot \omega_j^{(A)}(x_i) + o_{j,i}^{(B)} \cdot \omega_j^{(B)}(x_i) + a_{j,i}$$

4) The cloud sends $\vec{\boldsymbol{t}} = [\vec{\boldsymbol{t}}_1, ..., \vec{\boldsymbol{t}}_h]$ to client B.

e. **Set Intersection: Client-Side Result Retrieval**

1) Client B removes the blinding factors from each vector $\vec{\boldsymbol{t}}_j$ ($\forall j, 1 \leq j \leq h$) using the corresponding vector $\vec{\boldsymbol{q}}_j$ (provided by client A in step c.6). The result is the vector $\vec{\boldsymbol{g}}_j$ computed as follows.

$\forall j, 1 \leq j \leq h$ and $\forall i, 1 \leq i \leq n$:

$$g_{j,i} = t_{j,i} - q_{j,i} = \omega_j^{(A)}(x_i) \cdot \tau_j^{(A)}(x_i) + \omega_j^{(B)}(x_i) \cdot \tau_j^{(B)}(x_i)$$

2) Given each vector $\vec{\boldsymbol{g}}_j$ and $\vec{\boldsymbol{x}}$ it interpolates the polynomial $\phi_j(x)$ ($\forall j, 1 \leq j \leq h$).

3) It extracts the roots of each polynomial. It considers the union of the valid roots as the intersection of the sets.

Remark 1: Client I can always update (or replace) the blinding factors of its outsourced dataset in the cloud without leaking any information to it. To do so, it picks a fresh master key $mk'^{(I)}$, and derives h keys $k_j'^{(I)}$ from the master key:

$$\forall j, 1 \leq j \leq h : k_j'^{(I)} = \mathtt{PRF}(mk'^{(I)}, j)$$

Next, it uses each key $k_j'^{(I)}$ to generate n pseudorandom values $z_{j,i}'^{(I)}$ for each bin:

$$\forall j, 1 \leq j \leq h, \forall i, 1 \leq i \leq n : z_{j,i}'^{(I)} = \mathtt{PRF}(k_j'^{(I)}, i)$$

Also, it uses its old master key to regenerate the blinding factors $z_{j,i}^{(I)}$ used to blind the outsourced dataset. Then, for every bin it computes the following values:

$$\forall j, 1 \leq j \leq h, \forall i, 1 \leq i \leq n : u_{j,i}^{(I)} = -z_{j,i}^{(I)} + z_{j,i}'^{(I)}$$

It sends all $u_{j,i}^{(I)}$ to the cloud and asks it to sum them with the corresponding blinded values $o_{j,i}^{(I)} = \tau_j^{(I)}(x_i) + z_{j,i}^{(I)}$. After the cloud follows its instruction it would get the following blinded values:

$\forall j, 1 \leq j \leq h, \forall i, 1 \leq i \leq n :$

$$o_{j,i}'^{(I)} = o_{j,i}^{(I)} + u_{j,i}^{(I)} = \tau_j^{(I)}(x_i) + z_{j,i}'^{(I)}$$

Now, the client can discard its old master key and only needs to keep $mk'^{(I)}$ locally.

Remark 2: If the number of elements mapped to a bin exceed its capacity, the polynomial cannot be interpolated correctly after the computation. To avoid this, the server sets the parameters including the number of bins, the maximum load of each bin and the maximum set cardinality in such a way that the probability of any bin exceeding its capacity is negligible. The parameters can be derived by the cloud using inequality 2 (provided in section 3.5 with example values shown in Table 3).

Remark 3: In EO-PSI, the client needs to find the roots of h polynomials of degree $2d$, where d is a fixed value picked by the cloud and it is much smaller than the maximum number of elements, c. In contrast, in O-PSI the client receives only one polynomial of degree $2c$. Clearly, finding roots of h polynomials of small degree $2d$ is much faster than finding the roots of one polynomial of very large degree $2c$ and our performance evaluation in section 8 also supports this (see Fig. 4).

Remark 4: In both EO-PSI and O-PSI, the cloud-side setup is performed only once, when the cloud comes online. Afterward, it does not need to do any computation in this step. Furthermore, none of our protocols requires the participation of a trusted third party.

Remark 5: Bloom filters can be used in PSI Protocols to improve their efficiency [25]. A Bloom filter encodes a set and allows membership queries. In traditional PSI, the parties have a local copy of their own sets, so they can query the filters using the sets to get the intersection. In delegated PSI, clients outsource their data and do not keep a local copy. A delegated PSI protocol based on Bloom filters would require clients to enumerate the universe of the set elements in order to get the intersection. For this reason, we do not use Bloom filters in our protocol.

Remark 6: Public key cryptography preserves certain algebraic properties, therefore protocols based on it can be simpler and more intuitive than those based on symmetric key cryptography. For this reason, we first design the O-PSI protocol to show feasibility, then the EO-PSI protocol to improve efficiency.

5.3 Multiple Clients EO-PSI

With minor adjustments, the protocol can support $m > 2$ number of clients. Here, we denote the result recipient by client B and the other clients by $A_z, \forall z, 1 \leq z \leq y$ and $y = m - 1$.

Similar to the two clients case, here each client A_z sends to the cloud a temporary key $tk^{(A_z)}$ that lets the cloud generate for each bin $\texttt{HT}_j$ a set of pseudorandom values $a_{j,i}^{(A_z)}$ and two pseudorandom polynomials $\omega_j^{(A_z)}(x)$ and $\omega_j^{(B_z)}(x)$. However, as it is shown below, the cloud-side computation in step d.3 is slightly changed. $\forall j, 1 \leq j \leq h$ and $\forall i, 1 \leq i \leq n$:

$$t_{j,i} = o_{j,i}^{(B)} \cdot \omega_j^{(B)}(x_i) + \sum_{z=1}^{y} a_{j,i}^{(A_z)} + \sum_{z=1}^{y} o_{j,i}^{(A_z)} \cdot \omega_j^{(A_z)}(x_i)$$

where $\omega_j^{(B)}(x) = \sum_{z=1}^{y} \omega_j^{(B_z)}(x)$

Note, in the above step the cloud first adds all the polynomials $\omega_j^{(B_z)}(x)$ together, then it evaluates the result polynomial at every element in $\vec{\boldsymbol{x}}$, and next multiplies the result by client B's blinded values for that bin (i.e. bin $\texttt{HT}_j$).

Consequently, client B in step e.1 removes the blinding factors from vector $\vec{\boldsymbol{t}}_j$ as follows:

$$\begin{aligned} g_{j,i} &= t_{j,i} - \sum_{z=1}^{y} q_{j,i}^{(A_z)} \\ &= \omega_j^{(B)}(x_i) \cdot \tau_j^{(B)}(x_i) + \sum_{z=1}^{y} \omega_j^{(A_z)}(x_i) \cdot \tau_j^{(A_z)}(x_i) \end{aligned}$$

In multiple clients EO-PSI, even if client B colludes with $y-1$ clients, it cannot learn any information about the non-colluding client's set elements. The reason is that, as it is shown in [21], the polynomial $\omega_j^{(B)}(x)$ is always a uniformly random polynomial even if only one of the polynomials $\omega_j^{(B_z)}(x)$ is uniformly random and unknown to client B.

Remark 1: In multiple client EO-PSI, the communication and computation complexities for those clients who authorize the computation (i.e. clients A_z) are independent of the number of clients participating in the protocol. In other words, the computation and communication complexities for client A in the two client case are similar to client A_j's in the multiple clients case. Note that the same holds for multiple client O-PSI.

Remark 2: In multiple client EO-PSI, each client A_j independently authorizes the computation, without the need to interact with the other authorizing clients. The same is true for multiple client O-PSI.

6 Proof of Security

Now we present the proof of EO-PSI security in the semi-honest model. The security proof of O-PSI can be found in [1]. O-PSI and EO-PSI are both proved using the ideal-real paradigm. However, there are some differences between the proofs: (1) the security relies on different assumptions, in O-PSI it relies on the assumption of the existence of a semantically secure additive homomorphic encryption scheme, while in EO-PSI it relies on the assumption of the existence of a secure pseudorandom function; (2) in O-PSI the clients' blinded input sets are represented as a single polynomial, while in EO-PSI, the sets are split into bins and each bin is represented as a polynomial.

***Theorem* 2.** *If* $\texttt{PRF}$ *is a pseudorandom function, then EO-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 a 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 with the protocol, the views can be simulated in the same way up to the point where the execution stops.

Case 1: Corrupted Cloud. In this case, we show that we can construct a simulator Sim_C that can produce a computationally indistinguishable view. In the real execution, the cloud's view, $\mathsf{View}_C^{\pi}(\Lambda, S^{(A)}, S^{(B)})$, is as follows:

$$\{\Lambda, r_C, \vec{\boldsymbol{o}}^{(A)}, \vec{\boldsymbol{o}}^{(B)}, tk, \mathbf{ID}^{(A)}, \mathbf{ID}^{(B)}, \mathbf{Compute}, \Lambda\}$$

In the above view, r_C is the outcome of internal random coins of the cloud, $\vec{\boldsymbol{o}}^{(A)}$, $\vec{\boldsymbol{o}}^{(B)}$ are the hash tables each containing the blinded set representations of A's and B's sets, and tk is an l-bit random key used in the protocol to generate the pseudorandom polynomials and the blinding factors to mask the result generated by the cloud.

To simulate this view, Sim_C does the following: it creates an empty view and appends to it Λ and uniformly at random chosen coins r'_C. It uses the public parameters and the hash function to construct two hash tables $\texttt{HT}'^{(A)}$ and $\texttt{HT}'^{(B)}$. Then, it fills each bin of the hash tables with n uniformly random values picked from the same field $\mathbb{F}_p$; so each bin $\texttt{HT}'^{(I)}_j$ $(\forall I, I \in \{A, B\})$ contains the vector $\vec{\boldsymbol{o}}'^{(I)}_j$ of n random values. It also chooses a key tk'. Afterward, it appends $\vec{\boldsymbol{o}}'^{(A)} = [\vec{\boldsymbol{o}}'^{(A)}_1, ..., \vec{\boldsymbol{o}}'^{(A)}_h]$, $\vec{\boldsymbol{o}}'^{(B)} = [\vec{\boldsymbol{o}}'^{(B)}_1, ..., \vec{\boldsymbol{o}}'^{(B)}_h]$ and tk' to the view. Finally, the simulator appends $\mathbf{ID}^{(A)}, \mathbf{ID}^{(B)}$, **Compute** and 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 (i.e. both are Λ), the random coins are both uniformly random, and so they are indistinguishable. In the real model, the elements in $\vec{\boldsymbol{o}}^{(I)}$ ($\forall I, I \in \{A, B\}$) are blinded with the outputs of a pseudorandom function. Also, the elements in $\vec{\boldsymbol{o}}'^{(I)}$

are random elements of the field. As the blinded values and random value are indistinguishable, the vectors $\vec{\boldsymbol{o}}_i^{(I)}$ and $\vec{\boldsymbol{o}}_i'^{(I)}$ are indistinguishable; thus, the vectors $\vec{\boldsymbol{o}}^{(I)}$ and $\vec{\boldsymbol{o}}'^{(I)}$ are also indistinguishable. As the keys tk and tk' are picked uniformly at random, they are computationally indistinguishable as well. What is more, $\mathbf{ID}^{(A)}, \mathbf{ID}^{(B)}$ and **Compute** in both models are identical. Moreover, the output parts in both views are identical (i.e. both are Λ). So, we conclude that the views are indistinguishable.

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

$$\mathsf{View}_A^{\pi}(\Lambda, S^{(A)}, S^{(B)}) = \{S^{(A)}, r_A, mk^{(B)}, \mathbf{ID}^{(B)}, \Lambda\}$$

The simulator Sim_A does the following: it creates an empty view. It receives the party's input $S^{(A)}$ and appends it to the view. Then, it inserts uniformly at random chosen coins r'_A to it. Next, it picks an l-bit key $mk'^{(B)}$ uniformly at random and appends it to the view. After that, it inserts $\mathbf{ID}^{(B)}$ and Λ into the view. In both models $S^{(A)}$ is identical. Moreover, both r_A and r'_A are picked uniformly at random so they are indistinguishable. Since both keys $mk^{(B)}$ and $mk'^{(B)}$ are chosen uniformly at random they are computationally indistinguishable, too. Moreover, $\mathbf{ID}^{(B)}$ and Λ are identical in both models. So, the two views are indistinguishable.

Case 3: Corrupted client B**.** In the real execution, client B's view is as follows:

$$\mathsf{View}_B^{\pi}(\Lambda, S^{(A)}, S^{(B)}) = \{S^{(B)}, r_B, \vec{\boldsymbol{g}}, \vec{\boldsymbol{q}}, f_{\cap}(S^{(A)}, S^{(B)})\}$$

The simulator Sim_B receives the party's input ($S^{(B)}$) and output ($f_{\cap}(S^{(A)}, S^{(B)})$), and does the following:

1) Creates an empty view, then appends $S^{(B)}$ and uniformly at random chosen coins r'_B to it.
2) Picks two sets $S'^{(A)}$ and $S'^{(B)}$ such that $S'^{(A)} \cap S'^{(B)} = f_{\cap}(S^{(A)}, S^{(B)})$ and $|S'^{(A)}|, |S'^{(B)}| \leq c$.
3) Constructs the hash tables $\mathtt{HT}'^{(A)}$ and $\mathtt{HT}'^{(B)}$ using the public parameters. Next, maps the elements in $S'^{(A)}$ and $S'^{(B)}$ to the bins of $\mathtt{HT}'^{(A)}$ and $\mathtt{HT}'^{(B)}$, respectively. $\forall I, I \in \{A, B\}$ and $\forall s_i'^{(I)} \in S'^{(I)}$: $\mathtt{H}(s_i'^{(I)}) = j$, then $s_i'^{(I)} \rightarrow \mathtt{HT}_j'^{(I)}$, where $1 \leq j \leq h$.
4) For each bin constructs a polynomial representing its elements. If a bin contains less that d elements first it is padded with dummy values, $r_{j,i}'^{(I)}$, to d elements. $\forall I, I \in \{A, B\}$ and $\forall j, 1 \leq j \leq h$: $\tau_j'^{(I)}(x) = \prod_{i=1}^{d}(x - e_i^{(I)})$, where $e_i^{(I)} \in \mathtt{HT}_j'^{(I)}$, $e_i^{(I)} = s_i'^{(I)}$ or $e_i^{(I)} = r_{j,i}'^{(I)}$.
5) Assigns a random polynomial $\omega_j'^{(I)}$ of degree d to each bin $\mathtt{HT}_j'^{(I)}$ ($\forall j, 1 \leq j \leq h$) of the hash table $\mathtt{HT}'^{(I)}$ ($\forall I, I \in \{A, B\}$).
6) Constructs the vectors $\vec{\boldsymbol{g}}'_j$ whose elements are computed as follows. $\forall j, 1 \leq j \leq h$ and $\forall i, 1 \leq i \leq n$:
$$g'_{j,i} = \tau_j'^{(A)}(x_i) \cdot \omega_j'^{(A)}(x_i) + \tau_j'^{(B)}(x_i) \cdot \omega_j'^{(B)}(x_i)$$
where $\tau_j'^{(I)}(x)$ is the polynomial representing the set of elements contained in bin $\mathtt{HT}_j^{(I)}$.
7) Picks a key mk' and derives h keys, k'_j, from it as below.
$$\forall j, 1 \leq j \leq h : k'_j = \mathtt{PRF}(mk', j)$$
8) Uses each key k'_j to generate $\vec{\boldsymbol{q}}'_j$ whose elements are computed as follows.
$$\forall j, 1 \leq j \leq h \text{ and } \forall i, 1 \leq i \leq n\text{: } q'_{j,i} = \mathtt{PRF}(k'_j, i)$$
9) Adds $\vec{\boldsymbol{g}}' = [\vec{\boldsymbol{g}}'_1, ..., \vec{\boldsymbol{g}}'_h]$ and $\vec{\boldsymbol{q}}' = [\vec{\boldsymbol{q}}'_1, ..., \vec{\boldsymbol{q}}'_h]$ to the view.
10) Finally, inserts, $f_{\cap}(S^{(A)}, S^{(B)})$ to the view.

Now we show that the two views are computationally indistinguishable. In both models $S^{(B)}$ is identical. As r_B and r'_B are chosen uniformly at random, they are indistinguishable.

In the real model, the elements in $\vec{\boldsymbol{q}}_j$ are blinded by the outputs of a pseudorandom function. So the blinded elements are uniformly random values. On the other hand, in the ideal model the elements in $\vec{\boldsymbol{q}}'_j$ are the outputs of a pseudorandom function. Hence, the elements in both vectors $\vec{\boldsymbol{q}}$ and $\vec{\boldsymbol{q}}'$ are computationally indistinguishable.

Furthermore, in the real model, given each unblinded vector $\vec{\boldsymbol{g}}_j$, the adversary interpolates a polynomial of the form $\phi(x)_j = \omega_j^{(A)}(x) \cdot \tau_j^{(A)}(x) + \omega_j^{(B)}(x) \cdot \tau_j^{(B)}(x) = \mu_j \cdot gcd(\tau_j^{(A)}(x), \tau_j^{(B)}(x))$, where μ_j is a uniformly random polynomial and $gcd(\tau_j^{(A)}(x), \tau_j^{(B)}(x))$ represents the intersection of the set elements in the corresponding bin. Similarly, in the ideal model, each polynomial $\phi'_j(x)$ interpolated from vector $\vec{\boldsymbol{g}}'_j$ has the form $\phi'_j(x) = \omega_j'^{(A)}(x) \cdot \tau_j'^{(A)}(x) + \omega_j'^{(B)}(x) \cdot \tau_j'^{(B)}(x) = \mu'_j \cdot gcd(\tau_j'^{(A)}(x), \tau_j'^{(B)}(x))$, where μ'_j is a uniformly random polynomial. As mentioned in section 3.3, it has been shown in [21] that the polynomials $\phi_j(x)$ and $\phi'_j(x)$ (for each bin) only contain information about the intersections of the corresponding sets and have the same distribution in both models. Finally, in both views the output part (i.e. $f_{\cap}(S^{(A)}, S^{(B)})$) is identical. Hence, the two views are computationally indistinguishable.

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

Thus, both the O-PSI and EO-PSI protocols are secure in the semi-honest model and we have proven their security using the real-ideal paradigm. In the proof, we used standard assumptions and did not rely on non-standard ones (e.g. random oracle model).

7 Comparison

We first evaluate EO-PSI and O-PSI by comparing their properties to those provided by other protocols that delegate PSI computation to a cloud. We also compare these protocols in terms of communication and computation complexity. Table 2 summarizes the results.

Properties. When PSI computation is delegated to a server who is not fully trusted, protecting the privacy of the computation input and output from the server is crucial. However, as discussed in section 2 the protocols in [17], [19], [20] do not fully preserve data privacy and leak some information to the cloud server. Protocols like the size-hiding variation of [15], those in [16], [18], O-PSI and EO-PSI offer this protection.

Another desirable security property is that PSI computation is only possible with the explicit authorization from all the clients. In [19], the server can use the outsourced data to compute PSI without client permission. The server cannot decrypt the result but can learn information about the intersection e.g. the size. In [17], the protocol includes an authorization step. However, this step is not very effective. 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 permission.

In O-PSI, EO-PSI, [17], [19], [20], datasets are outsourced and stored on the cloud server. The server then uses the stored encrypted datasets for computation. This is not the case for

TABLE 2
Comparison of different delegated PSI protocols. Set cardinality and intersection cardinality are denoted by c and k, respectively.

Property	**EO-PSI**	**O-PSI**	**[15]**	**[16]**	**[17]**	**[18]**	**[19]**	**[20]**
Private against the Cloud	✓	✓	✓	✓	×	✓	×	×
PSI Computation Authorization	✓	✓	✓	✓	×	✓	×	✓
Data Storage Outsourcing	✓	✓	×	×	✓	×	✓	✓
Non-interactive Client-side Setup	✓	✓	×	×	✓	✓	✓	✓
Many Private Set Intersections without Re-preparation	✓	✓	×	×	×	×	×	×
Multiple Clients	✓	✓	✓	✓	✓	×	✓	✓
Communication Complexity	$O(c)$	$O(c)$	$O(c)$	$O(c^2)$	$O(c)$	$O(c^2)$	$O(k)$	$O(k)$
Computation Complexity	$O(c)$	$O(c^2)$	$O(c)$	$O(c^2)$	$O(c^2)$	$O(c^2)$	$O(c)$	$O(c)$

protocols in [15], [16], [18]. Those protocols are for one-off computation and the clients have to upload the encrypted datasets to the server before each computation.

The protocols in [15], [16] require clients to interact with each other at setup. In [15] clients need to jointly compute the key for the pseudorandom permutation used to encode the datasets, while in [16] they need to jointly compute the parameters used to encrypt their datasets. In contrast to these protocols, in [17], [18], [19], [20], O-PSI and EO-PSI the clients can independently prepare and outsource their private datasets. This is particularly desirable in the context of cloud computing as organizations and individuals should be able to outsource their datasets at different points in time and without prior knowledge of those they will engage in computation in the future. Among the five protocols that support non-interactive client setup, the protocols in [19], [20] require additionally a trusted third party to initialize some protocol parameters on behalf of the clients, whereas the rest of the protocols do not need such assistance.

Note that O-PSI and EO-PSI are the only PSI protocols that allow clients to delegate the computation an unlimited number of times without the need to prepare their datasets for each computation. Also, the computation does not reveal any information to the cloud even if it is delegated multiple times. This is vital in the case 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. Nevertheless, this is not the case for any of the other aforementioned protocols. In the other protocols, the clients need to re-encode the datasets locally for each time the computation is delegated in order to prevent the cloud from inferring information about the set elements and the intersection over time.

As we illustrated in sections 4.3 and 5.3, O-PSI and EO-PSI can be easily extended to support multiple clients. The same holds for [15], [16], [17], [19], [20]. In contrast, [18] does not support multiple clients, as it requires an additional logical operation that is not supported by the homomorphic encryption scheme it uses.

Communication Complexity. The communication complexity of O-PSI for the client who receives the result, client B, is $O(c)$, where c is the dataset size. Because client B sends client A the $n = 2c+1$ encrypted random values $E_{pk_B}(r_i^{(B)})$ for $1 \leq i \leq n$, in step c.1. The communication complexity for client A, who authorizes the operation on its dataset, is also $O(c)$, as it sends n values of the form $E_{pk_B}(r_i^{(B)} \cdot (r_i^{(A)})^{-1})$, $1 \leq i \leq n$ to the cloud, in step c.3. The communication complexity for the cloud is $O(c)$ too. Because it sends to client B the result vector $\overrightarrow{\boldsymbol{t}}$ of size n, in step d.3. Thus, the overall communication complexity of our protocol is $3n$ which is linear, $O(c)$, to the dataset size.

Note that, in EO-PSI, for a fixed probability and fixed bin's maximum load, the hash table length h is linear to the set cardinality c. This is clearer, if we rewrite $d = (1+\sigma) \cdot \frac{c}{h}$ (presented in subsection 3.5) as $h = (1+\sigma) \cdot \frac{c}{d}$. During the protocol, client B sends a single value to client A so its communication cost is constant, in step c.1. Client A sends a single value to the cloud and sends h bins each containing n elements to client B, in step c.6. Therefore, client A's communication complexity is $O(c)$. The cloud's communication complexity is $O(c)$ as well, because it sends h bins each containing n elements to client B, in step d.4. Therefore, the overall communication complexity is linear to the dataset size.

In [16] 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 cloud with $O(c)$ communication complexity, and another to interactively compute the private set intersection with the cloud with $O(c^2)$ communication complexity. For the protocol in [18], the communication complexity is also quadratic $O(sc^2)$, 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 [15], [17] have $O(c)$ communication complexity with messages containing symmetric key encryption ciphertexts. Finally, the protocols in [19], [20] have $O(k)$ complexity, where k is the intersection size.

Computation Complexity. We evaluate the computation cost of O-PSI by counting the number of exponentiation and factorization operations, as their cost dominates that of other operations in the protocol. More specifically, client B performs n exponentiations to encrypt the random values in step c.1, and needs another n exponentiations to decrypt the polynomial sent by the cloud in step e.1. Also, it needs to factorize the result polynomial that costs $O(c^2)$ in step e.2. So, in total it carries out $2n$ exponentiations and factorizes a polynomial with the cost of $O(c^2)$. Client A performs n exponentiations to authorize the set intersection in step c.2, while the cloud carries out n exponentiations to encrypt client B's dataset and n exponentiations to transform client A's dataset in step d.2, a total of $2n$ exponentiations. It is interesting to note that using the point-value representation increases the overall storage costs at the cloud side. But, the modest increase in storage brings a significant decrease in the computational costs, from $O(c^2)$ (when using encrypted coefficients such as in [21]) to $O(c)$.

Now we analyze the computation complexity of EO-PSI. In our analysis, we do not consider the pseudorandom function invocation cost as it is a fast operation and dominated by the other operations (e.g. modular arithmetic, interpolation and factorization) in our protocol. Client A, performs $2hn$ modular multiplication and $2hn$ modular addition operations to blind the values, in step

TABLE 3
EO-PSI Hash Table Parameters (bin size d = 100, overloading probability $< 2^{-40}$).

Set size (c)	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
Number of Bins (h)	26	53	106	211	432	863	1772	3543	7282	14564	29128
σ	1.5	1.56	1.56	1.56	1.63	1.63	1.70	1.70	1.78	1.78	1.78

c.5. Also, in order for it to evaluate the two polynomials allocated to every bin, it carries out $2hnd$ modular multiplication and $2hnd$ modular addition operations, in step c.5. So the computation complexity of client A is $O(c)$. Also, the cloud carries out $2hn$ modular multiplication and $2hn$ modular addition operations to blind the values in step d.3. Moreover, for the cloud to evaluate the two polynomials assigned to every bin, it performs $2hnd$ modular multiplication and $2hnd$ modular addition operations, in step d.3. Therefore, the computation complexity of the cloud is $O(c)$, too. Client B executes hn modular addition operations to unblind the cloud's response, in step e.1. Furthermore, in step e.2, it interpolates h polynomials where each polynomial interpolation costs $O(d)$ and d is a constant value. In step e.3, it factorizes h polynomials where each polynomial factorization costs $O(m^2)$ and $m = 2d$ is a constant value. Hence, in total client B's computation cost is $O(c)$.

The semi-honest variant of the protocol in [15] also has linear complexity $O(c)$, as the client computing the result and the cloud invoke the pseudorandom permutation (PRP) c times, while the other client invokes the PRP, $2c$ times. On the other hand, the computational overhead in [16] is quadratic $O(c^2)$, as it involves a joint PSI protocol (plus public key encryption of the dataset elements). The protocol in [18] also has quadratic complexity, as it involves $O(c^2)$ BGN public key encryption operations. In [17] the client performs $O(c)$ modular additions, while the cloud carries out $O(c^2)$ operations to compare the expanded sets of the users. The protocol in [19] is based on bilinear maps and requires $6c$ pairings at the cloud side and $2k$ exponentiations at the client side, resulting in $O(c)$ and $O(k)$ computation complexity at the cloud and client side respectively. The protocol in [20] is also based on bilinear maps, it requires 6 exponentiation operations at the client-side, k pairings for decryption of the result at the client-side, and $8c$ pairing at the server-side. So the overall computation cost of the protocol is $O(c)$.

Storage Complexity. In our protocols, storage complexity at the client-side is constant. In O-PSI each client needs to keep only two keys (for the pseudorandom function and public key encryption); and in EO-PSI it needs to keep only one key (for the pseudorandom function). The protocols designed in [17], [19], [20] require also constant storage complexity at the client-side. Nevertheless, since the protocols in [15], [16], [18] support one-off delegation, the client needs to have data locally available to re-encode and delegate the computation and this introduces storage complexity linear to the set size. The server side storage complexity is linear in all protocols.

Comparison to regular PSI. The computational and communication complexity of EO-PSI is linear to the dataset size. Currently the most efficient regular PSI protocols also have linear complexity. Although similar in complexity, in practice the regular PSI protocols can be more efficient. In our protocols, storage complexity at client-side is constant. The regular PSI protocols' storage complexity is at least linear to the set size, as each party uses the locally available set elements. We emphasize that delegated PSI and regular PSI protocols are designed for different settings with different requirements and they cannot substitute each other. Thus it is difficult to make a fair comparison. For example, in delegated PSI protocol, the cloud has to compute the intersection without knowing anything about both clients' datasets, unlike regular PSI protocols where a party knows its own set. Also there is no way in regular PSI to reduce the storage complexity, as there is no external server to outsource the storage. Therefore the comparison above is only a presentation of facts and should not be interpreted as regular PSI is better than delegated PSI or the other way round.

8 Performance evaluation

We implemented the O-PSI and EO-PSI protocols in C++ [7]. Both implementations use the NTL library [8] for the polynomial operations. The O-PSI implementation also uses the Paillier library [9] for the Paillier homomorphic encryption. In the experiments, we use 32-bit integers as set elements and 80-bit padding. All experiments were run on an Ubuntu 14.04 desktop PC with an Intel i5-4590@3.3 GHz CPU, 8 GB RAM.

In section 7, we compared our protocols with the state-of-the-art protocols in [15], [16], [17], [18], [19], [20]. Here we compare the performance of our protocols with the protocols in [17], [20]. We chose these two protocols because they are delegated PSI protocols that support both storage and computation outsourcing. Thus they are the closest to our protocols. The protocols in [15], [16] are server-aided PSI protocols and only support one-off computation with the help of a server. We did not choose the protocol in [19] because the protocol in [20] is more efficient than it (see [20]). For the protocol in [17] we implemented it using C++ and measured its performance using the hardware above, while for the protocol in [20] we used the performance data reported in the paper. Although the hardware used in [20] is different, this difference would not affect the comparison results, because as we show later, the performance difference to EO-PSI is significant, almost 2 orders of magnitude. We also implemented the protocol in [18]. However, the performance of this protocol is much worse than the others and cannot scale to sets with more than a few hundred elements. Thus we do not include it in the comparison and provide the performance results for it separately in an appendix.

We first show in Fig. 3 the performance comparison of the four protocols. The figure is plotted using a logarithmic scale. In the figure, the x-axis shows the number of elements in both clients' sets and the y-axis indicates the total running time of the protocols. We range the set size between 2^{10} and 2^{20} elements. For EO-PSI, we also set the bin size of the hash table to 100 and the probability of overloading to less than 2^{-40}. Using inequality 2, we calculate

[7] EO-PSI's source code is available at https://github.com/nitrogl/eo-psi

[8] http://www.shoup.net/ntl/

[9] http://acsc.cs.utexas.edu/libpaillier

the number of bins we need in the hash tables. The results are shown in Table 3. Table 3 also shows σ, which is the ratio of the bin size d and the expected number of elements in a bin. A σ closer to 1 means the bins are better utilized. As we can see, the performance of EO-PSI is much better than O-PSI and the one in [20]. In fact, for O-PSI and [20] we skipped tests with set size over 2^{15} as the running time would be too long. The performance of protocol in [17] is better than EO-PSI when the set cardinality is less than 2^{18}, while for larger sets EO-PSI is better.

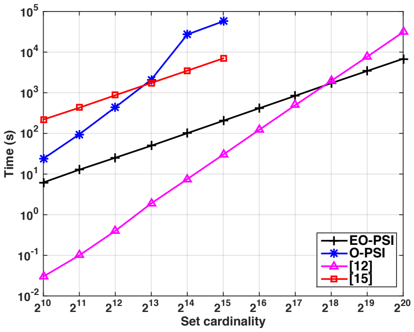

Fig. 3. Performance comparison of protocols (total running time).

We further show the time breakdown by step of all four protocols in Table 4. As we can see, for EO-PSI, the running time for each step increases almost linearly to the set size. The data outsourcing, the computation delegation, and cloud-side computation steps are fast. The protocol running time is dominated by the last step which is the result retrieval step. In the result retrieval step, client B unblinds the data received, interpolates the polynomials, and extracts roots from the polynomials by factorizing them. We found out that factorization takes most of the time in this step. In the experiments with set sizes $2^{10}, 2^{15}$ and 2^{20}, the factorization times are 5.32, 175.35, 5918.93 seconds, respectively. In our implementation, we factorize polynomials by using the NTL library. This step could be improved by improving the underlying factorization algorithm. For O-PSI, the running time is also dominated by the last step. It is worse than EO-PSI because the polynomial to be factored is larger. For the protocol in [17], the running time is dominated by the cloud-side computation. The complexity of this step is quadratic thus it is less scalable than EO-PSI. For the protocol in [20], outsourcing and cloud-side computation steps dominate the running time. Although the complexity of the protocol is linear, it uses heavy public key operations which make it less efficient than EO-PSI.

For EO-PSI, the hash table parameters also affect the performance. When the set size is fixed, if we increase the bin size, then we have fewer bins to process, but the time for processing (i.e. polynomial evaluation, blinding and factorization) each bin will increase. To investigate the best trade-off between the number of bins and their size, we ran experiments with different bin sizes. The results are illustrated in Fig. 4. In the figure, the x-axis is the bin size and the y-axis is the total running time. We range the set size between $2^{10} - 2^{17}$ elements, and the number of bins are calculated accordingly using inequality 2 so that the probability of overloading is less than 2^{-40}. In theory, we should see a (not so sharp) "V" shape such that the running time decreases along with decreasing bin size until some point where the running time starts to increase when the bin size decreases. However, the results from our experiment are not exactly as expected. In the figure, we can see that for all the tested cases, there does exist a turning point when the bin size is ~ 120. Nevertheless, there is an unexpected sharp increase in running time when the bin size is larger than 120.

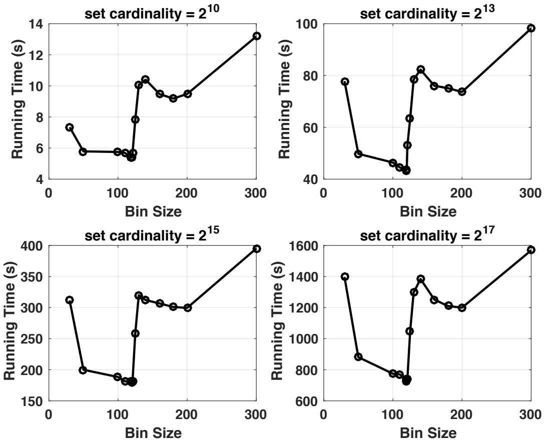

Fig. 4. The performance of EO-PSI with different bin sizes.

Since the running time of EO-PSI implementation is dominated by polynomial factorization which is done by NTL, we suspect that this increase is due to some unknown implementation choices of the NTL library. To confirm this, we measured the time for factorizing random polynomials with degree corresponding to the bin size in NTL. The results are shown in Fig. 5. From the figure, we can see also a sharp increase near bin size 120. Without a further investigation, a safer choice of bin size would be 100, which is a bit away from the sharp increase while the running time is not much higher.

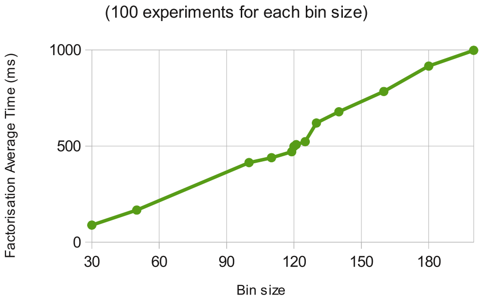

Fig. 5. Time taken to factorize random polynomials using NTL library.

9 Conclusions

Cloud computing is rapidly gaining in popularity among individuals and businesses, mainly due to the innovation it enables and the opportunities it offers. With its importance increasing, outsourcing datasets and computation to the cloud becomes an appealing approach. Nevertheless, as the cloud cannot be fully trusted the privacy of the outsourced data is a major concern for clients. So, the need arises for protocols that can carry out private set operations on outsourced private data without revealing anything about the data and the computation results to the cloud.

In this paper, we presented two such protocols for private set intersection, O-PSI and EO-PSI. The protocols let clients independently prepare and outsource their private datasets to the

TABLE 4
Protocols' performance comparison: breakdown by step (time in seconds).

Steps	Protocols	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
Data Outsourcing	EO-PSI:	0.1	0.2	0.4	0.8	1.8	3.7	7.3	14.6	30.7	59.8	126.3
	O-PSI:	0.2	1	4	16.2	124.2	301.4	–	–	–	–	–
	[17]:	0.007	0.01	0.02	0.05	0.1	0.2	0.4	0.8	1.7	3.5	6.7
	[20]:	117.3	229.2	458.4	915.3	1842.3	3679.9	–	–	–	–	–
Computation Delegation	EO-PSI:	0.1	0.3	0.8	1.5	3.2	6.5	13	25.6	54.3	105.2	218
	O-PSI:	02.4	4.8	9.6	19.2	73.9	86.1	–	–	–	–	–
	[17]:	0.007	0.01	0.02	0.05	0.1	0.2	0.4	0.8	1.6	3.4	6.3
	[20]:	0.089	0.089	0.089	0.089	0.089	0.089	–	–	–	–	–
Cloud-side Computation	EO-PSI:	0.2	0.4	0.8	1.6	3.3	6.8	13.5	26.9	57.2	109.7	226.7
	O-PSI:	2.9	7	18.5	54.9	336.1	697.4	–	–	–	–	–
	[17]:	0.02	0.1	0.4	1.8	7.3	29.6	121.8	493.7	1953.8	7710.2	31173.3
	[20]:	90.6	178.9	368.4	739.6	1468.7	2984.5	–	–	–	–	–
Result Retrieval	EO-PSI:	5.7	11.8	23.2	45.7	92.8	187.5	381.8	773.3	1576.8	3164.2	6293.1
	O-PSI:	20.6	86.8	418.1	2036.9	26685.3	56354.8	–	–	–	–	–
	[17]:	0.0005	0.0006	0.0007	0.00078	0.0008	0.001	0.0015	0.002	0.003	0.006	0.008
	[20]:	10.9	21.1	42.5	90.2	179.1	364.2	–	–	–	–	–
Total	EO-PSI:	6.2	12.8	25.2	49.8	101.2	204.6	415.8	840.6	1719.1	3438.9	6864.2
	O-PSI:	23.9	95.2	441.2	2109.1	27148.8	57357.5	–	–	–	–	–
	[17]:	00.03	0.1	0.4	1.9	7.5	30.1	122.6	495.4	1957.1	7717.3	31186.4
	[20]:	218.9	429.3	869.3	1745.2	3490.1	7028.6	–	–	–	–	–

cloud. At any point later in time, they can ask the cloud to run PSI on their private datasets. In this process, the cloud learns nothing about the dataset elements, the intersection, and the intersection cardinality. Furthermore, the protocols ensure that the cloud can compute the intersection only when all the clients agree and the clients can securely delegate PSI computation on the outsourced datasets an unlimited number of times with no need to download and re-prepare the datasets. These properties make the protocols particularly suitable for a cloud computing setting, allowing clients to fully benefit from the increased collaboration the cloud enables and the cost-efficient resources it provides without sacrificing their privacy.

Although both our protocols satisfy the properties outlined, EO-PSI is much more efficient than O-PSI for two reasons. First, it does not use any public key encryption, which is computationally expensive, and second it lets clients retrieve the result faster by utilizing a hash table. A performance study of prototype implementations of the two protocols clearly demonstrates this. Moreover, a comparison to the performance of other similar state-of-the-art protocols also shows that EO-PSI scales well and is faster than them for large set sizes.

We have shown that our protocols are secure in the presence of semi-honest parties. Even though the semi-honest model is widely used in the literature, we would have a stronger protocol if we relax this assumption. Therefore, in the future, we would like to improve EO-PSI so that it relies on a weaker assumption (e.g. malicious cloud). Finally, we would like to design protocols that support more delegated private set operations (e.g. set different, set union).

Acknowledgements

We would like to thank the anonymous reviewers. This work was partially supported by an EPSRC Doctoral Training Grant studentship and an EPSRC research grant (EP/M013561/2).

References

[1] A. Abadi, S. Terzis, and C. Dong, "O-PSI: delegated private set intersection on outsourced datasets," in *30th IFIP TC11 International Conference on ICT Systems Security and Privacy Protection (SEC)*, 2015, pp. 3–17.

[2] M. J. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *EUROCRYPT*, 2004, pp. 1–19.

[3] C. C. Aggarwal and P. S. Yu, Eds., *Privacy-Preserving Data Mining - Models and Algorithms*, ser. Advances in Database Systems. Springer, 2008, vol. 34.

[4] E. D. Cristofaro, J. Kim, and G. Tsudik, "Linear-complexity private set intersection protocols secure in malicious model," in *ASIACRYPT*, 2010, pp. 213–231.

[5] G. Mezzour, A. Perrig, V. D. Gligor, and P. Papadimitratos, "Privacy-preserving relationship path discovery in social networks," in *CANS*, 2009, pp. 189–208.

[6] S. Nagaraja, P. Mittal, C. Hong, M. Caesar, and N. Borisov, "BotGrep: Finding P2P bots with structured graph analysis," in *USENIX Security Symposium*, 2010, pp. 95–110.

[7] S. Marston, Z. Li, S. Bandyopadhyay, and A. Ghalsasi, "Cloud computing - the business perspective," in *HICSS-44*, 2011, pp. 1–11.

[8] J. Webster, "An enterprise cloud 'survey of surveys'," *Forbes*, 2016. [Online]. Available: http://www.forbes.com/sites/johnwebster/2016/05/03/an-enterprise-cloud-survey-of-surveys/

[9] O. Annenko, "Enterprise cloud adoption: It's high time we run the first reality check," *elastic.io*, 2016. [Online]. Available: https://www.elastic.io/enterprise-cloud-adoption-reality-check/

[10] X. Chen, J. Li, X. Huang, J. Ma, and W. Lou, "New publicly verifiable databases with efficient updates," *IEEE Trans. Dependable Sec. Comput.*, vol. 12, no. 5, pp. 546–556, 2015.

[11] X. Chen, J. Li, J. Weng, J. Ma, and W. Lou, "Verifiable computation over large database with incremental updates," *IEEE Trans. Computers*, vol. 65, no. 10, pp. 3184–3195, 2016.

[12] Y. Wang, Q. Wu, D. S. Wong, B. Qin, S. S. M. Chow, Z. Liu, and X. Tan, "Securely outsourcing exponentiations with single untrusted program for cloud storage," in *ESORICS*, 2014, pp. 326–343.

[13] X. Chen, J. Li, J. Ma, Q. Tang, and W. Lou, "New algorithms for secure outsourcing of modular exponentiations," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 9, pp. 2386–2396, 2014.

[14] C. Wang, K. Ren, and J. Wang, "Secure and practical outsourcing of linear programming in cloud computing," in *INFOCOM*, 2011, pp. 820–828.

[15] S. Kamara, P. Mohassel, M. Raykova, and S. Sadeghian, “Scaling private set intersection to billion-element sets,” in *FC*, 2014, pp. 863–874.

[16] F. Kerschbaum, “Collusion-resistant outsourcing of private set intersection,” in *SAC*, 2012, pp. 1451–1456.

[17] F. Liu, W. K. Ng, W. Zhang, D. H. Giang, and S. Han, “Encrypted set intersection protocol for outsourced datasets,” in *IC2E*, 2014, pp. 135–140.

[18] F. Kerschbaum, “Outsourced private set intersection using homomorphic encryption,” in *ASIACCS*, 2012.

[19] Q. Zheng and S. Xu, “Verifiable delegated set intersection operations on outsourced encrypted data.” *IACR Cryptology ePrint Archive*, p. 178, 2014.

[20] S. Qiu, J. Liu, Y. Shi, M. Li, and W. Wang, “Identity-based private matching over outsourced encrypted datasets,” *Cloud Computing, IEEE Transactions on*, vol. PP, no. 99, 2015.

[21] L. Kissner and D. X. Song, “Privacy-preserving set operations,” in *CRYPTO*, 2005, pp. 241–257.

[22] E. D. Cristofaro and G. Tsudik, “Practical private set intersection protocols with linear complexity,” in *FC*, 2010, pp. 143–159.

[23] G. Ateniese, E. D. Cristofaro, and G. Tsudik, “(if) size matters: Size-hiding private set intersection,” in *PKC*, 2011, pp. 156–173.

[24] S. K. Debnath and R. Dutta, “Secure and efficient private set intersection cardinality using Bloom filter,” in *ISC*, 2015, pp. 209–226.

[25] C. Dong, L. Chen, and Z. Wen, “When private set intersection meets big data: an efficient and scalable protocol,” in *ACM CCS*, 2013, pp. 789–800.

[26] B. Pinkas, T. Schneider, and M. Zohner, “Faster private set intersection based on OT extension,” in *USENIX Security Symposium*. USENIX, 2014.

[27] B. Pinkas, T. Schneider, G. Segev, and M. Zohner, “Phasing: Private set intersection using permutation-based hashing,” in *USENIX Security Symposium*, 2015, pp. 515–530.

[28] O. Goldreich, *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.

[29] E. Stefanov and E. Shi, “Multi-cloud oblivious storage,” in *ACM CCS*, 2013, pp. 247–258.

[30] M. Raykova, B. Vo, S. M. Bellovin, and T. Malkin, “Secure anonymous database search,” in *First ACM Cloud Computing Security Workshop*, 2009, pp. 115–126.

[31] P. Paillier, “Public-key cryptosystems based on composite degree residuosity classes,” in *EUROCRYPT*, 1999, pp. 223–238.

[32] C. Dong, L. Chen, J. Camenisch, and G. Russello, “Fair private set intersection with a semi-trusted arbiter,” in *DBSec*, 2013, pp. 128–144.

[33] K. S. Kedlaya and C. Umans, “Fast polynomial factorization and modular composition,” *SIAM J. Comput.*, vol. 40, no. 6, pp. 1767–1802, 2011.

[34] A. V. Aho and J. E. Hopcroft, *The Design and Analysis of Computer Algorithms*, 1st ed. Addison-Wesley Longman Publishing Co., Inc., 1974.

[35] J. Berrut and L. N. Trefethen, “Barycentric Lagrange interpolation,” *SIAM Review*, vol. 46, no. 3, pp. 501–517, 2004.

[36] M. Raab and A. Steger, “Balls into bins - A simple and tight analysis,” in *Second International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, 1998, pp. 159–170.

[37] P. Berenbrink, A. Czumaj, A. Steger, and B. Vöcking, “Balanced allocations: the heavily loaded case,” in *STOC*, 2000, pp. 745–754.

[38] D. Boneh, E. Goh, and K. Nissim, “Evaluating 2-DNF formulas on ciphertexts,” in *TCC*, 2005, pp. 325–341.

[39] T. Sander, A. L. Young, and M. Yung, “Non-Interactive CryptoComputing for NC^1,” in *FOCS*, 1999, pp. 554–567.

APPENDIX

We also implemented the outsourced PSI protocol in [18] in C++. We use the Pairing-Based Cryptography Library [10] for the underlying bilinear pairing operation. We measured the running time with sets whose cardinalities range from $2^4 = 16$ to $2^7 = 128$. The performance figures can be found in Table 5.

TABLE 5
Running time (in seconds) of the protocol in [18]

Steps	2^4	2^5	2^6	2^7
Data Outsourcing	8.14	16.19	32.22	64.07
Computation Delegation	130.33	517.98	2066.56	8208.07
Cloud-side Computation	191.53	767.37	3046.90	11896.68
Result Retrieval	6.73	27.85	116.67	498.94
Total	336.72	1329.39	5262.35	20667.76

As we can see in the table, the performance of this protocol is much worse than the other protocols we considered in Section 8. There are several reasons for this: (1) the complexity of the protocol is quadratic; (2) the protocol requires costly pairing-based BGN homomorphic encryption [38]; (3) the protocol uses Bloom filters and the Sander Young Yung Technique [39] which together add a large hidden constant factor to the asymptotic complexity and impact significantly the performance.

Aydin Abadi received a PhD from the Department of Computer and Information Sciences, University of Strathclyde, in 2017. His main interests are in the field of cryptography, with a focus on designing secure and practical cryptographic protocols, delegated secure multi-party computation, delegated Private Set Intersection and verifiable computation. He obtained a BSc and MSc from University of Lahijan, Iran and Leeds, UK respectively.

Sotirios Terzis holds a BSc(Hons) and MSc from the University of Crete and a PhD from Trinity College Dublin. He is currently a lecturer at the Department of Computer and Information Sciences, University of Strathclyde. His research interests are in the socio-technical engineering of computer security with a focus on usable authentication and nature and socially inspired mechanisms. He has over 20 publications in international journals and conferences. He is a member of the IEEE Computer Society, the ACM and the British Computer Society.

Roberto Metere is a research student in Secure Computation at the School of Computing, Newcastle University. Previously, he worked as IT consultant, developer and web developer for small and medium size companies in many projects and different computer areas, with a focus on information security and cybersecurity. His research focuses on conducting automatic proofs in the computational model for cryptographic protocol design.

Changyu Dong holds a PhD from Imperial College London. He is currently a senior lecturer at the School of Computing, Newcastle University. His research interests include applied cryptography, trust management, data privacy and security policies. His recent work focuses mostly on designing practical secure computation protocols. The application domains include for example privacy preserving medical data analytics, secure cloud computing, privacy preserving data mining and private data processing in mobile and wireless networks. He has over 30 publications in international journals and conferences.

[10] https://crypto.stanford.edu/pbc/