Abadi A, Terzis S, Dong C.

VD-PSI: Verifiable Delegated Private Set Intersection on Outsourced Private Datasets.

***In: Financial Cryptography and Data Security 2016 (FC 2016)*. 2016, Barbados: International Financial Cryptography Association**

VD-PSI: Verifiable Delegated Private Set Intersection on Outsourced Private Datasets

Aydin Abadi, Sotirios Terzis, and Changyu Dong

Department of Computer and Information Sciences, University of Strathclyde, Glasgow, UK
firstname.surname@strath.ac.uk

Abstract. Private set intersection (PSI) protocols have many real world applications. With the emergence of cloud computing the need arises to carry out PSI on outsourced datasets where the computation is delegated to the cloud. However, due to the possibility of cloud misbehaviors, it is essential to verify the integrity of any outsourced datasets, and result of delegated computation. Verifiable Computation on private datasets that does not leak any information about the data is very challenging, especially when the datasets are outsourced independently by different clients. In this paper we present VD-PSI, a protocol that allows multiple clients to outsource their private datasets and delegate computation of set intersection to the cloud, while being able to verify the correctness of the result. Clients can independently prepare and upload their datasets, and with their agreement can verifiably delegate the computation of set intersection an unlimited number of times, without the need to download or maintain a local copy of their data. The protocol ensures that the cloud learns nothing about the datasets and the intersection. VD-PSI is efficient as its verification cost is linear to the intersection cardinality, and its computation and communication costs are linear to the dataset cardinality. Also, we provide a formal security analysis in the standard model.

1 Introduction

Private set intersection (PSI) allows parties to compute the intersection of their datasets without revealing anything about the data beyond the intersection [11]. PSI has a variety of real world applications like privacy preserving data mining [2], homeland security [7], etc. Over the years, due to its importance researchers have designed a number of PSI protocols [22, 9, 28, 18, 20, 23, 21, 29]. However, these typically require parties to jointly compute the intersection with locally available datasets. With the growing impact of cloud computing on businesses, due to its economic benefits [25], the need arises to delegate PSI computation on outsourced datasets to the cloud [1, 18].

Cloud computing offers flexible and cost effective storage and computation resources to clients. Nevertheless, past real-world incidents [5] and recent research (see [17, 4] for surveys) have shown that the cloud cannot be fully trusted and it may misbehave by exposing or tampering with clients' sensitive data, or fiddling with computation results. These misbehaviors can have a serious impact on businesses. So, it is essential for clients not only to protect the confidentiality of their outsourced data, but also to verify the integrity of the data and the result of the computation delegated to the cloud. In other words, there is a pressing need for *verifiable delegated PSI on outsourced private datasets*.

Verifying the integrity of computation while preserving the confidentiality of the data is particularly challenging. This is even more the case when the data are outsourced independently by different owners, and the data integrity is also a concern. Beyond protecting the confidentiality, and verifying the integrity of any outsourced data and delegated computation, there is a need for a mechanism that ensures outsourced data cannot be used without the owner's permission. But, this mechanism should not restrict the particular clients that may come together to compute the intersection of their datasets, by requiring that they are specified in advance; also it should not constraint the number of clients that come together or the number of intersections that may be performed.

In this paper we present VD-PSI, a protocol that supports efficient verifiable delegated PSI on outsourced private datasets and satisfies the above requirements. VD-PSI allows the result recipient to check whether the computation was performed correctly on the requested intact datasets, without having to keep a local copy of the data, or having any knowledge of the other clients' data. The protocol imposes minimal computational overheads to the verifier (i.e. at most linear to the intersection cardinality). It allows clients to independently prepare and upload their private datasets. It ensures that outsourced datasets can only be used with the owner's permission. It supports multiple clients who can verifiably delegate PSI computation an unlimited number of times with no need to download or re-encode their outsourced data. Moreover, it achieves all that while ensuring that the result recipient learns nothing beyond the intersection about the other clients' datasets, and the cloud learns nothing about the datasets and the intersection. The computation and communication costs of VD-PSI are linear to the set cardinality.

The rest of the paper starts with a survey of related work in section 2, while section 3 defines our security model and key concepts we rely on. Section 4 presents the design of our protocol and its extension, while section 5 proves its security. Section 6 presents an analysis of the protocol's overheads and a comparison to work closest to it. Finally, section 7 concludes the paper and identifies directions for future work.

2 Related Work

Verifiable delegated PSI can be built either on top of generic verifiable computation protocols or as a standalone operation specific verifiable protocol. Verifiable computation (VC) protocols allow a computationally weak client to securely delegate computation of a function to a powerful but untrusted server. In this setting, the client can detect if the server provides an incorrect result.

We first consider generic VC protocols. A number of protocols such as [15, 14] are designed to address the VC problem, where [15] is based on the concept of ringers, and [14] uses the argument system and relies on private information retrieval. However, they do not preserve the privacy of client data. As a result, the server may misbehave and expose sensitive client data, i.e. the input and output of the computation. Other protocols like [12] which uses fully homomorphic encryption (FHE) and Yao's garbled circuits, and [10] based on FHE and a homomorphic hashing technique, preserve the privacy of client data; but, they have been designed for single-client scenarios.

There are practical scenarios in which multiple clients, who mutually distrust each other, want to verifiably delegate computation to a server. In this case, the computation should be verifiable and a client's data should be protected from the other clients who receive the result. These requirements integrate secure multi-party computation (MPC) with VC. In secure MPC protocols, clients (jointly) perform some computation without revealing to each other their private inputs. There are two protocols for such scenarios, [6] based on Yao's garbled circuits, FHE and non-interactive proxy oblivious transfer, and [16] based on the same building blocks plus multi-sender attribute-based encryption. These protocols do not require clients to directly interact with each other at setup. Nonetheless, they do not support outsourced data as they require each client to know the public keys of all other clients when preparing its private data. Consequently, changing the participating clients requires outsourced data to be downloaded and re-prepared.

The protocol in [24] also supports verifiable multi-client delegated generic computation, and allows clients to independently outsource their private data. In this protocol, clients do not need to interact during setup, but they have to interactively decrypt the result when receiving it. Although this protocol satisfies all the requirements set out in section 1, it is not computationally efficient as it is based on (multi-key) fully homomorphic encryption and expensive generic zero-knowledge proofs. Moreover, the protocol is not suitable when the number of inputs is large, as in order to verify the result correctness a client needs to access all the (hash values of) encrypted inputs.

We now turn our attention to protocols designed specifically for PSI. PSI was first introduced in [11] based on additive homomorphic encryption and polynomial representation of sets. Over the years, some efficient protocols, like [9, 28, 27], have been proposed. However, they are interactive in the sense that the clients jointly compute the intersection of their sets. Thus, they need to have a local copy of their sets to compute the set intersection. A number of protocols that support delegation of PSI to a server are proposed in [18, 20, 23, 21, 29, 1]. Among them only [1], which is mainly based on point-value polynomial representation of sets and additive homomorphic encryption, supports delegation of both storage and the computation to the cloud, and is fully private. However, it only considers a semi-honest adversary. As a result, the clients cannot verify the correctness of the computation. The only protocols that support verifiable delegated PSI are in [29, 18]. In the former, a client encrypts his data and uses tags based on bilinear maps for verification. However, as it is shown in [1], it is not fully private and leaks information about the intersection to the server. The latter is based on a pseudo-random function, whose key is generated jointly by the clients prior to encoding the data and detects server misbehavior at the cost of replicating a number of times all elements of the sets. Although the protocol is efficient, it does not support outsourced data.

From the above discussion, it should be clear that neither PSI protocols nor most of generic computation protocols can meet all our requirements. Only [24] meets all the requirements, but it is not efficient. So, the quest for *efficient verifiable delegated PSI on outsourced private datasets* remains open.

3 Preliminaries

3.1 Security Model

We consider a static adversary who controls one of the parties at a time. The definition and model are according to [13]. Without loss of generality we consider the three party case where a cloud C, and two clients, A and B are involved. We allow an adversary who corrupts C to be malicious. So, it may arbitrarily deviate from the prescribed protocol, but it does not collude with the clients. This is a reasonable assumption as it is usually a well-established company and does not want to jeopardize its reputation by colluding with others. The non-colluding assumption is well-accepted and widely used in the literature [18, 16]. Moreover, we allow an adversary who corrupts a client to be semi-honest.

We define a three-party protocol π computing function F where $F : \Lambda \times 2^{\mathcal{U}} \times 2^{\mathcal{U}} \rightarrow \Lambda \times \Lambda \times f_{\cap}$. Here, Λ denotes the empty string, $2^{\mathcal{U}}$ denotes the powerset of the set universe and $f_{\cap}$ denotes the set intersection function. For every tuple of inputs Λ, S_A and S_B 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. To show the protocol is secure, we define an ideal model, which satisfies all the security needs. In the ideal model, there is an incorruptible trusted third party (TTP) which helps with the functionality. The protocol is said to be secure if for every adversary in the real model there is an adversary in the ideal model that can simulate the real adversary.

Real Model: Here, protocol π is executed between parties A, B, C and an adversary denoted by R that is allowed to corrupt one party. In the beginning of the protocol, each party $I \in \{A, B\}$ receives its private input S_I, the protocol's public parameters, random coins r, and an auxiliary input z, while the cloud C receives the public parameters, a set of random coins r, and an auxiliary input z. At the end of the execution, an honest party outputs whatever is prescribed by the protocol and the adversary outputs its view. The joint output of the real model execution of π between the parties in the presence of the adversary R is defined as $\mathsf{REAL}_{\pi,R(z)}(\Lambda, S_A, S_B)$.

Ideal Model: The ideal model takes place between parties A, B, C and a simulator SIM that is allowed to corrupt at most one party at a time. Each party receives the same input as the corresponding party in the real model. An honest party always sends its input to the TTP. The corrupted party may abort or send arbitrary input. The cloud, C, receives d (i.e. $d \geq |S_I|, I \in \{A, B\}$) from the TTP. The TTP computes the set intersection and sends the result to B. If the TTP receives an abort message as an input, it sends B the special symbol $\perp$. The joint output of the parties in the ideal model in the presence of SIM is defined as $\mathsf{IDEAL}_{F,SIM(z)}(\Lambda, S_A, S_B)$.

Definition 1. *Let π be a protocol and F a deterministic function defined as above. Protocol π is said to securely compute F in the presence of static adversaries if for every probabilistic polynomial time (PPT) adversary R in the real model, there exists a PPT adversary SIM in the ideal model such that $\forall I, I \in \{A, B, C\}$:*

$$\mathsf{IDEAL}_{F,SIM_I(z)}(\Lambda, S_A, S_B) \stackrel{c}{\equiv} \mathsf{REAL}_{\pi,R_I(z)}(\Lambda, S_A, S_B)$$

3.2 Additively Homomorphic Encryption

A semantically secure additively homomorphic encryption has the following properties:

(a) Given two ciphertexts $E_{pk}(a), E_{pk}(b)$, $E_{pk}(a) \cdot E_{pk}(b) = E_{pk}(a+b)$.
(b) 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 [26]. 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 [11] and is widely used [22, 8, 1]. For the universe of set elements, $\mathcal{U}$, we can define a public finite field $R = \mathbb{F}_p$ that is big enough to encode all elements in $\mathcal{U}$. Also, the field is big enough that when an element is picked uniformly at random from it, the probability that the value belongs to the universe is negligible. Then, we can define a polynomial ring $R[x]$, which consists of all polynomials with coefficients from R. We can represent a set, S, by polynomial $\rho(x) = \prod_{i=1}^{d}(x - s_i)$, where $s_i \in S, |S| = d$ and $\rho(x) \in R[x]$. The polynomial has the property that every element $s_i \in S$ is a root of it. Furthermore, we can always encode every s_i as $s'_i = s_i||h(s_i)$, where h is a cryptographic hash function, so that given $s'_j = a||b$ and h's output size, one can parse s'_j into a and b, and check $b \stackrel{?}{=} h(a)$.

For two sets S_A and S_B represented by polynomials ρ_A and ρ_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 ρ_A and ρ_B, and two degree d polynomials γ_A and γ_B picked uniformly at random from $R[x]$, it is proven in [22] 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 ρ_A and ρ_B are polynomials representing sets S_A and S_B, then the polynomial $\rho_C = \gamma_A \cdot \rho_A + \gamma_B \cdot \rho_B$ contains only information about $S_A \cap S_B$ and no information about other elements in S_A or S_B. Given polynomial ρ_C, to find the intersection, one can extract the polynomial's roots[1] and consider the roots that have the above structure $(s_i||h(s_i))$ as the intersection.

Based on the theorem of the interpolating polynomial, for a set $\{(x_1, y_1), ..., (x_n, y_n)\}$ with x_i distinct, there exists a unique polynomial $\rho(x)$ of degree at most $n-1$ such that $\forall i, 1 \leq i \leq n : y_i = \rho(x_i)$. Therefore, $\rho(x)$ can be represented by the n pairs (x_i, y_i). If the x_i values are fixed and public, we can represent the polynomial as a vector of y-coordinates, $\vec{\boldsymbol{y}} = [y_1, ..., y_n]$.

[1] To find the roots of a polynomial over a finite field, we can first factorize it to get a set of monic polynomials (see [19] for some algorithms), then find the monic polynomials' roots.

Polynomial arithmetic in point-value representation can be done by adding or multiplying the corresponding y-coordinates. The key benefit of point-value representation is that multiplication complexity is $O(d)$; whereas, multiplying polynomials in coefficient form has $O(d^2)$ complexity. We can convert a polynomial in point-value form to regular coefficient form, using polynomial interpolation [3].

4 The Proposed Scheme: VD-PSI

4.1 An Overview of VD-PSI

We give an overview of the protocol and depict the interaction between parties in Fig 1. Without loss of generality, we consider two clients A and B. At setup, each client independently encodes, blinds, and stores his dataset in the cloud. Later on, when client B becomes interested in the intersection of his dataset and client A's, he obtains her permission by sending a message to her. Client A authorizes the computation by using the message sent by client B to compute a new message that she sends to the cloud. The cloud uses the message and the clients' outsourced datasets to compute the intersection, and sends the result to client B. Client B, decodes the cloud's response, retrieves the intersection and checks its correctness. If the result was correct the client accepts it.

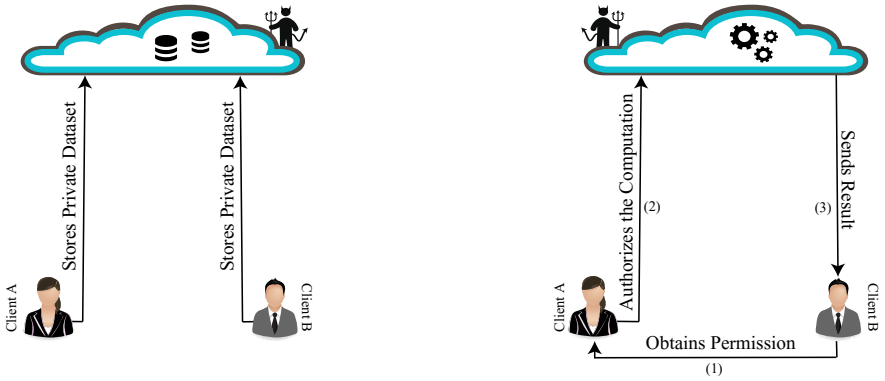

Fig. 1: The left-hand side figure: parties interaction at data outsourcing phase; the right-hand side figure: parties interaction at computation delegation phase.

The main novelty of VD-PSI is a lightweight verification mechanism that allows a client to efficiently verify the correctness of the result without having access to his own outsourced dataset and having any knowledge of the other client's dataset. To achieve this, when the clients decide to delegate the computation of the set intersection, they agree on a secret value β and encode β in a way that reveals nothing to the cloud, then send the encoded β to the cloud. When computing the intersection, the cloud uses the clients' outsourced datasets and the encoded β. If the cloud computes honestly, value β will be inserted into the intersection. Since the outsourced data is blinded, when client

B receives the result from the cloud, he needs to unblind it to get the intersection. If the cloud misbehaves or tampers with the result, then after unblinding client B gets a random set, which will not contain β. Thus, by checking whether β is included in the result set, client B knows whether the result set is correct. The verification is very lightweight because the only overhead is to check whether β is included in the result set.

4.2 VD-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. We denote the multiplicative inverse and additive inverse of value h_i, by $(h_i)^{-1}$and $(-h_i)$, respectively. We use $E_{pk_I}(h_i)$ and $D_{sk_I}(h_i)$ to say that value h_i is encrypted using client I's public key, and decrypted using his secret key, respectively.

1. **Cloud-Side Setup.** The cloud picks a public parameter d that is an upper bound of the set cardinality. It constructs a finite field $\mathbb{F}_p$, where p is a large prime number. It also constructs a vector $\vec{\boldsymbol{x}}$ containing $n = 2d + 3$ distinct non-zero x_i values randomly picked from $\mathbb{F}_p$. It picks a pseudo-random function $f : \{0,1\}^m \times \{0,1\}^l \rightarrow \mathbb{F}_p$, which takes an l-bit key (i.e. where l is the security parameter) and an m-bit message, and maps the message to an element in the field pseudo-randomly. The cloud publishes the description of the field, the value n, the vector $\vec{\boldsymbol{x}}$ along with the pseudo-random function f.
2. **Client-Side Setup and Data Outsourcing.** Let client $I \in \{A, B\}$ have a set S_I, where $S_I \subset \mathcal{U}$ and $|S_I| \leq d$. Both clients do the following tasks:
   (a) Compute a key pair (pk_I, sk_I) for Paillier encryption and publish the public key pk_I. Pick two random private keys, $k_r^{(I)}$ and $k_z^{(I)}$ for the pseudo-random function f. All keys are generated according to given security parameters.
   (b) Generate a polynomial representation of the set. $\forall s_i^{(I)} \in S_I$: $\tau_I(x) = \prod_{i=1}^{|S_I|}(x - s_i^{(I)})$.
   (c) Convert the polynomial into point-value form by evaluating $\tau_I(x)$ at every element x_i in vector $\vec{\boldsymbol{x}}$. This yields values $\tau_I(x_i)$, where $1 \leq i \leq n$.
   (d) Blind every $\tau_I(x_i)$ by first computing pseudo-random values $r_i^{(I)} = f(k_r^{(I)}, i)$ and $z_i^{(I)} = f(k_z^{(I)}, i)$, and then computing $o_i^{(I)} = r_i^{(I)} \cdot (\tau_I(x_i) + z_i^{(I)})$.
   (e) Send the blinded dataset $\vec{\boldsymbol{o}}^{(I)} = [o_1^{(I)}, ..., o_n^{(I)}]$ to the cloud.
3. **Set Intersection: Computation Delegation.** In this phase, client B is interested in the intersection of his set and client A's.
   (a) Client B picks a uniformly random value $\beta \xleftarrow{R} \mathbb{F}_p^*$ that will be inserted into the two datasets and chooses three fresh keys $k_a^{(B)}$, $k_b^{(B)}$ and $k_{r'}^{(B)}$ that are used to blind the messages sent by client A to the cloud.
   (b) Client B constructs a vector $\vec{\boldsymbol{e}}^{(B)}$ that will be used by client A to ask the cloud to insert β to her dataset and switch her blinding factors. $\forall i, 1 \leq i \leq n$: $e_i^{(B)} = E_{pk_B}(\sigma(x_i) \cdot r_i'^{(B)} \cdot r_i^{(B)})$, where $\sigma(x_i) = (x_i - \beta)$, values $r_i^{(B)}$ are the blinding factors used by client B in step 2d and $r_i'^{(B)} = f(k_{r'}^{(B)}, i)$.
   (c) Client B sends to client A: $\vec{\boldsymbol{e}}^{(B)}$, β, $k_a^{(B)}$, $k_b^{(B)}$, $k_{r'}^{(B)}$, $k_z^{(B)}$, and his ID, $\mathbf{ID}^{(B)}$.
   (d) Client A generates $\vec{\boldsymbol{v}}^{(A)}$ and $\vec{\boldsymbol{v}}^{(B)}$ that allow the cloud to multiply each client dataset by a random polynomial and insert β to it. Also, $\vec{\boldsymbol{v}}^{(A)}$ allows the cloud

to switch the blinding factors of client A's dataset. $\forall i, 1 \leq i \leq n$:

$$\begin{aligned} v_i^{(A)} &= (e_i^{(B)})^{\omega_A(x_i)\cdot(r_i^{(A)})^{-1}} \\ &= E_{pk_B}(r_i^{(B)} \cdot r_i'^{(B)} \cdot \omega_A(x_i) \cdot \sigma(x_i) \cdot (r_i^{(A)})^{-1}) \\ v_i^{(B)} &= \omega_B(x_i) \cdot \sigma(x_i) \cdot r_i'^{(B)} \end{aligned}$$

where $r_i'^{(B)} = f(k_{r'}^{(B)}, i)$, key $k_{r'}^{(B)}$ was sent by client B in step 3c, $r_i^{(A)}$ are the blinding values used by client A in step 2d, $\omega_A(x)$ and $\omega_B(x)$ are two random polynomials of degree $d+1$ and $\sigma(x) = (x - \beta)$.

(e) Client A generates $\vec{\boldsymbol{v}}'^{(A)}$ and $\vec{\boldsymbol{v}}'^{(B)}$ to allow the cloud to preserve the correctness of the result. $\forall i, 1 \leq i \leq n$:

$$\begin{aligned} v_i'^{(A)} &= (e_i^{(B)})^{\omega_A(x_i)\cdot(-z_i^{(A)})+a_i} \\ &= E_{pk_B}((-z_i^{(A)}) \cdot r_i^{(B)} \cdot r_i'^{(B)} \cdot \omega_A(x_i) \cdot \sigma(x_i) + c_i) \\ v_i'^{(B)} &= (e_i^{(B)})^{\omega_B(x_i)\cdot(-z_i^{(B)})+b_i} \\ &= E_{pk_B}((-z_i^{(B)}) \cdot r_i^{(B)} \cdot r_i'^{(B)} \cdot \omega_B(x_i) \cdot \sigma(x_i) + d_i) \end{aligned}$$

where $c_i = a_i \cdot r_i^{(B)} \cdot r_i'^{(B)} \cdot \sigma(x_i)$, $d_i = b_i \cdot r_i^{(B)} \cdot r_i'^{(B)} \cdot \sigma(x_i)$, $a_i = f(k_a^{(B)}, i)$, $b_i = f(k_b^{(B)}, i)$, the keys $k_a^{(B)}$ and $k_b^{(B)}$ sent by client B in step 3c, and $z_i^{(I)}$ are the values used by client $I \in \{A, B\}$ in step 2d.

(f) Client A sends to the cloud: $\vec{\boldsymbol{v}}^{(A)}$, $\vec{\boldsymbol{v}}'^{(A)}$, $\vec{\boldsymbol{v}}^{(B)}$, $\vec{\boldsymbol{v}}'^{(B)}$, $\mathbf{ID}^{(B)}$, $\mathbf{ID}^{(A)}$, and a request message **Compute**.

4. **Set Intersection: Cloud-Side Computation.**

(a) When the cloud receives client A's message, it uses $\vec{\boldsymbol{v}}^{(A)}$, $\vec{\boldsymbol{v}}'^{(A)}$ and client A's outsourced dataset $\vec{\boldsymbol{o}}^{(A)}$ to switch the dataset blinding factors, insert β to the dataset, and multiply it by a random polynomial; this results $\vec{\boldsymbol{t}}^{(C_1)}$.
$\forall i, 1 \leq i \leq n$: $t_i^{(C_1)} = (v_i^{(A)})^{o_i^{(A)}} \cdot v_i'^{(A)} =$
$E_{pk_B}(r_i^{(B)} \cdot r_i'^{(B)} \cdot \omega_A(x_i) \cdot \sigma(x_i) \cdot \tau_A(x_i) + c_i)$.

(b) The cloud uses $\vec{\boldsymbol{o}}^{(B)}$, $\vec{\boldsymbol{v}}^{(B)}$, $\vec{\boldsymbol{v}}'^{(B)}$ to insert β into client B's dataset, and multiply it by a random polynomial. This yields $\vec{\boldsymbol{t}}^{(C_2)}$.
$\forall i, 1 \leq i \leq n$: $t_i^{(C_2)} = v_i'^{(B)} \cdot E_{pk_B}(v_i^{(B)} \cdot o_i^{(B)}) =$
$E_{pk_B}(r_i^{(B)} \cdot r_i'^{(B)} \cdot \omega_B(x_i) \cdot \sigma(x_i) \cdot \tau_B(x_i) + d_i)$.

(c) The cloud combines the values computed in steps 4b and 4a to produce the final result $\vec{\boldsymbol{t}}^{(C_3)}$. $\forall i, 1 \leq i \leq n$: $t_i^{(C_3)} = t_i^{(C_1)} \cdot t_i^{(C_2)} =$
$E_{pk_B}(r_i^{(B)} \cdot r_i'^{(B)} \cdot (\omega_B(x_i) \cdot \sigma(x_i) \cdot \tau_B(x_i) + \omega_A(x_i) \cdot \sigma(x_i) \cdot \tau_A(x_i)) + c_i + d_i)$.

(d) The cloud sends to client B vector $\vec{\boldsymbol{t}}^{(C_3)}$.

5. **Set Intersection: Client-Side Result Verification and Retrieval.**

(a) Client B obtains $\vec{\boldsymbol{g}}$ by decrypting the cloud's response $\vec{\boldsymbol{t}}^{(C_3)}$, and unblinding the decrypted values using his knowledge of $(-c_i)$, $(-d_i)$, $(r_i^{(B)})^{-1}$ and $(r_i'^{(B)})^{-1}$. $\forall i, 1 \leq i \leq n$:

$$\begin{aligned} g_i &= (D_{sk_B}(t_i^{(C_3)}) + (-c_i) + (-d_i)) \cdot (r_i^{(B)})^{-1} \cdot (r_i'^{(B)})^{-1} \\ &= \omega_B(x_i) \cdot \sigma(x_i) \cdot \tau_B(x_i) + \omega_A(x_i) \cdot \sigma(x_i) \cdot \tau_A(x_i). \end{aligned}$$

(b) Client B interpolates a polynomial, $\phi(x)$, using the n point-value pairs (x_i, g_i), extracts its roots, and checks whether β is among them. If it is, he considers the rest of the roots as elements of the intersection; otherwise, he aborts.

Remark 1. In step 2d, client $I \in \{A, B\}$ blinds his private data $\tau_I(x_i)$ as $o_i^{(I)} = r_i^{(I)} \cdot (\tau_I(x_i) + z_i^{(I)})$ to preserve their privacy and to detect unauthorized modifica-

tions. If the client does not blind $\tau_I(x_i)$, the cloud can interpolate the polynomial $\tau_I(x)$ and find the client's set elements. After blinding, every $o_i^{(I)}$ is a uniformly random value and does not leak any information about $\tau_I(x_i)$. If the cloud changes a subset of elements in $\vec{\boldsymbol{o}}^{(I)}$, in step 5b the corresponding values in $\vec{\boldsymbol{g}}$ become uniformly random. As a result, the polynomial interpolated using the n pairs of (x_i, g_i) will not have root β (with a high probability), and the client will detect the misbehavior. The same also occurs if the cloud deviates from the protocol.

Remark 2. We set $n = 2d + 3$, because in step 5b, polynomial $\phi(x)$ is of degree $2d + 2$ and at least $2d + 3$ pairs of (x_i, y_i) are required to interpolate it. Therefore, given n pairs of (x_i, y_i), computed correctly, client B can always interpolate $\phi(x)$.

Remark 3. In section 3.3, we saw that the set of all roots of polynomial $\omega_B(x) \cdot \tau_B(x) + \omega_A(x) \cdot \tau_A(x)$ is $S_A \cap S_B$. Note that β is also a root of the polynomial $\phi(x) = \sigma(x) \cdot (\omega_B(x) \cdot \tau_B(x) + \omega_A(x) \cdot \tau_A(x))$, where $\sigma(x) = x - \beta$. Hence, in the protocol, a correctly computed result always contains value β.

Remark 4. Since for each computation, the fresh random polynomials $\omega_A(x)$ and $\omega_B(x)$ are used, the result recipient cannot find out anything beyond the intersection about the other client's set. Also, the cloud cannot learn the exact number of elements in the set; it only knows the upper bound of the set cardinality (i.e. d).

Remark 5. Every client I, after outsourcing his private dataset needs to keep locally only two secret keys, $k_r^{(I)}$ and $k_z^{(I)}$. Moreover, client B who is interested in the result generates keys $k_a^{(B)}$, $k_b^{(B)}$, $k_{r'}^{(B)}$ and value β on the fly for each run of the protocol and he can discard them after it ends. Furthermore, given p each client I can always generate his own public key N_I independently, such that $N_I > p + 2p^2 + p^3$ to preserve the computation correctness (i.e. to prevent any overflow during homomorphic operations). To determine the lower bound of N_I, we can calculate the maximal value that message m_i in $E(m_i)$ may have as a result of homomorphic operations in the protocol (here, by $E(m_i)$ we mean encryption of message m_i). To do so, we start from step 3b and calculate the upper bound of value m_i in each step (note that $o_i^{(I)} \in \mathbb{F}_p$). We continue this up to step 4d and then we set the lower bound of N_I to the maximal upper bound of m_i, that is $p + 2p^2 + p^3$.

Remark 6. We stress that the clients' outsourced datasets remain unchanged. Also, at the end of protocol all parties can discard all intermediate messages received.

4.3 Multiple Clients

With minor modifications two-client VD-PSI can be turned into q-client VD-PSI, where $q > 2$. Below we outline how this can be done. We denote the result recipient by client B and the other clients by A_j $(\forall j, 1 \leq j \leq m)$, where $m = q - 1$.

More specifically, in step 3c, client B sends to every client A_j the same message. Each client A_j takes the same steps described above, except step 3d, where she replaces $\vec{\boldsymbol{v}}^{(B)}$ with $\vec{\boldsymbol{v}}_j^{(B)}$, $v_{j,i}^{(B)} = E_{pk_B}(\omega_B^j(x_i) \cdot \sigma(x_i) \cdot r_i'^{(B)})$, where $\omega_B^j(x)$ is the random polynomial picked by client A_j.

In step 4b, the cloud computes $\vec{\boldsymbol{t}}^{(C_2)}$, by selecting one of the clients, say client A_k, and only using her vectors $\vec{\boldsymbol{v}}_k^{(B)}$, $\vec{\boldsymbol{v}}_k'^{(B)}$, discarding $\vec{\boldsymbol{v}}_j^{(B)}$, $\vec{\boldsymbol{v}}_j'^{(B)}$ generated by the other clients $A_j, \forall j, 1 \leq j \leq m, j \neq k$. As a result $\forall i, 1 \leq i \leq n$: $t_i^{(C_2)} = v_{k,i}'^{(B)} \cdot (v_{k,i}^{(B)})^{o_i^{(B)}} = E_{pk_B}(r_i^{(B)} \cdot r_i'^{(B)} \cdot \omega_B^k(x_i) \cdot \sigma(x_i) \cdot \tau_B(x_i) + d_i)$.

In step 4c, the cloud computes $\vec{\boldsymbol{t}}^{(C_3)} = \vec{\boldsymbol{t}}^{(C_2)} \cdot \prod_{1 \le j \le m} \vec{\boldsymbol{t}}_j^{(C_1)}$, $\forall i, 1 \le i \le n$: $t_i^{(C_3)} = E_{pk_B}(m \cdot c_i + d_i + (r_i^{(B)} \cdot r_i'^{(B)} \cdot (\omega_B^k(x_i) \cdot \sigma(x_i) \cdot \tau_B(x_i) + \sum_{1 \le j \le m} \omega_A^j(x_i) \cdot \sigma(x_i) \cdot \tau_A^j(x_i)))$ and sends it to client B.
Finally, in step 5a, client B computes $\vec{\boldsymbol{g}}$ as follows. $\forall i, 1 \le i \le n$:
$g_i = (D_{sk_B}(t_i^{(C_3)}) + m \cdot (-c_i) + (-d_i)) \cdot (r_i^{(B)})^{-1} \cdot (r_i'^{(B)})^{-1}$
The rest of the steps remain unchanged.
Remark 1: In the multi-client case, each client encrypts elements of vector $\vec{\boldsymbol{v}}_j^{(B)}$; whereas, in the two client case it does not need to do that. Nonetheless, regardless of the number of clients, every client's computation complexity is linear to the set cardinality.
Remark 2: Verification complexity at the verifier side is independent of the number of clients. Also, the number of messages every client, except the client who is interested in result, sends and receives is independent of the number of clients, too. The client who is interested in the result sends the same message to all other clients.
Remark 3: The security model we consider in this paper can be easily extended to the multi-client case, security analysis of the multi-client case remains the same as the two client case and we do not include it for the sake of brevity.

5 Proof of Security

In this section we sketch the security proof of the protocol. To this end, first we show that the cloud's misbehaviors can be detected with high probability, then we provide the main theorem.
Recall, the client encodes his set as blinded y-coordinates having the following form: $o_i = r_i \cdot (\tau(x_i) + z_i)$, where $o_i \neq 0$, $r_i = f(k_r, i)$ and $z_i = f(k_z, i)$. If $o_i = 0$ the client replaces k_r and k_z with new random keys and encodes $\tau(x_i)$ again until $\forall i, 1 \le i \le n : o_i \neq 0$. Note, o_i is uniformly distributed in $\mathbb{F}_p^*$. We can show that if the cloud applies any change to o_i, this will make the y-coordinate a uniformly random value.

Lemma 1. *Given $o_i = r_i \cdot (\tau(x_i) + z_i)$, where r_i and z_i are two independent pseudo-random values that are unknown to the cloud, if the cloud changes o_i to o_i', then $\tau'(x_i) = r_i^{-1} \cdot o_i' - z_i$ becomes a uniformly random value.*

Proof. When $o_i \neq o_i'$, $\tau'(x_i)$ is a uniformly random value in $\mathbb{F}_p$, because r_i and z_i are picked uniformly at random and independently of each other. □

In step 5a of the protocol, client B after decrypting the server's response obtains blinded values of the form $p_i = e_i \cdot g_i + z_i$, where e_i and z_i are pseudo-random values. If the cloud misbehaves (e.g. deviates from the protocol, modifies the outsourced client datasets, etc), some p_i are changed to p_i', and Lemma 1 implies that $g_i' = e_i^{-1} \cdot (p_i' - z_i)$ will be a uniformly random value. So, any cloud misbehavior turns some of the values g_i into uniformly random values.
Now, we show that given a set of y-coordinates some of which are uniformly random values, the polynomial that client B interpolates from them (see step 5b in the protocol), will not contain the specific root β with a high probability.

Lemma 2. *Let polynomial $\tau(x)$ be interpolated from $S = \{(x_1, y_1), ..., (x_n, y_n)\}$, and have a root β such that $\forall i, 1 \leq i \leq n : \beta \neq x_i$. Let $S' = \{(x_1, y'_1), ..., (x_n, y'_n)\}$, where at least one of y'_i is a uniformly random value and the rest of them are equal to the y-coordinates in S (i.e. $y'_j = y_j$). Let polynomial $\tau'(x)$ be interpolated from S'. The probability that $\tau'(x)$ has the root β is negligible.*

Proof. Given S', we interpolate a unique polynomial $\tau'(x)$ of degree at most $n-1$. According to the Lagrange interpolation, the polynomial is

$$\tau'(x) = \sum_{1 \leq j \leq n} y'_i \cdot \prod_{\substack{1 \leq k \leq n \\ j \neq k}} \frac{x - x_k}{x_j - x_k}$$

We evaluate $\tau'(x)$ at β:

$$\tau'(\beta) = \sum_{1 \leq j \leq n} y'_i \cdot \prod_{\substack{1 \leq k \leq n \\ j \neq k}} \frac{\beta - x_k}{x_j - x_k}$$

As $\forall i, 1 \leq i \leq n : \beta \neq x_i$, we would have $\prod_{\substack{1 \leq k \leq n \\ j \neq k}} \frac{\beta - x_k}{x_j - x_k} \neq 0$. Since, at least one of y'_i is uniformly random, value $y'_i \cdot \prod_{\substack{1 \leq k \leq n \\ j \neq k}} \frac{\beta - x_k}{x_j - x_k}$ is uniformly random. Therefore, $\tau'(\beta)$ is uniformly random. Thus, $Pr[\tau'(\beta) = 0] = \frac{1}{p}$ which is negligible. □

Now we are ready to prove that the client can detect cloud misbehavior with high probability.

Theorem 1. *Let clients A and B have sets $S^{(A)}$ and $S^{(B)}$ respectively; also let $S_\cap = S^{(A)} \cap S^{(B)}$. In the protocol if the cloud sends S' (where $S' \neq S_\cap$) to the client, the client can detect it with high probability.*

Proof. Due to Lemma 1, server misbehavior turns some of the y-coordinates (representing $S_\cap$) into uniformly random values. Also, in the protocol, β is chosen uniformly at random from $\mathbb{F}_p^*$, so the probability that $\beta = x_k$ for some $k, 1 \leq k \leq n$, is negligible; due to Lemma 2, if the client interpolates a polynomial by using a set of y-coordinates where at least one of them is a uniformly random value, the probability that the polynomial would have β as a root is negligible. Thus, if the server computes an incorrect intersection the client can detect this with high probability through the absence of β from the intersection. □

Finally, we prove our main theorem.

Theorem 2. *If the homomorphic encryption scheme is semantically secure, then the protocol is secure in the presence of (1) a malicious cloud and honest clients, (2) a semi-honest client and honest cloud.*

Proof. We consider three cases where each party is corrupted at a time.

Case 1: Cloud is corrupted. We construct a simulator SIM_C in the ideal model that uses the adversary R_C as a subroutine. Simulator SIM_C executes the following tasks.

(a) Pick two random sets S_E and S_D and choose the keys $k_r^{(E)}$, $k_z^{(E)}$, $k_r^{(D)}$, $k_z^{(D)}$, $k_b^{(E)}$, $k_a^{(E)}$, $k_{r'}^{(E)}$.

(b) Generate polynomials $\tau_E(x)$ and $\tau_D(x)$ representing the sets. Then, evaluate the polynomials at every element in $\vec{\boldsymbol{x}}$ and blind the evaluated values. This results in two vectors, $\vec{\boldsymbol{o}}^{(E)}$ and $\vec{\boldsymbol{o}}^{(D)}$. $\forall o_i^{(I)} \in \vec{\boldsymbol{o}}^{(I)}, o_i^{(I)} = r_i^{(I)} \cdot (\tau_I(x_i) + z_i^{(I)})$, $r_i^{(I)} = f(k_r^{(I)}, i), z_i^{(I)} = f(k_z^{(I)}, i)$, where $I \in \{D, E\}$.

(c) Pick a random value β', and construct polynomial $\sigma'(x) = (x - \beta')$. Then, pick two random polynomials, ω_E and ω_D, of degree $d+1$. Then, compute $\vec{\boldsymbol{v}}^{(E)}$ and $\vec{\boldsymbol{v}}^{(D)}$ as follows. $\forall i, 1 \leq i \leq n$:

$$v_i^{(D)} = E_{pk_E}(r_i^{(E)} \cdot r_i'^{(E)} \cdot \omega_D(x_i) \cdot \sigma'(x_i) \cdot (r_i^{(D)})^{-1})$$
$$v_i^{(E)} = \omega_E(x_i) \cdot r_i'^{(E)} \cdot \sigma'(x_i)$$

where $r_i'^{(E)} = f(k_{r'}^{(E)}, i)$.

(d) Compute $\vec{\boldsymbol{v}}'^{(E)}$ and $\vec{\boldsymbol{v}}'^{(D)}$; $\forall i, 1 \leq i \leq n, a_i^{(E)} = f(k_a^{(E)}, i), b_i^{(E)} = f(k_b^{(E)}, i)$:

$$v_i'^{(D)} = E_{pk_E}((-z_i^{(D)}) \cdot r_i^{(E)} \cdot r_i'^{(E)} \cdot \omega_D(x_i) \cdot \sigma'(x_i) + c_i^{(E)})$$
$$v_i'^{(E)} = E_{pk_E}((-z_i^{(E)}) \cdot r_i^{(E)} \cdot r_i'^{(E)} \cdot \omega_E(x_i) \cdot \sigma'(x_i) + d_i^{(E)})$$

where $I \in \{D, E\}, z_i^{(I)} = f(k_z^{(I)}, i)$, $c_i^{(E)} = a_i^{(E)} \cdot r_i^{(E)} \cdot r_i'^{(E)} \cdot \sigma'(x_i)$, $d_i^{(E)} = b_i^{(E)} \cdot r_i^{(E)} \cdot r_i'^{(E)} \cdot \sigma'(x_i)$.

(e) Invoke R_C and feed it with $\vec{\boldsymbol{o}}^{(D)}$, $\vec{\boldsymbol{o}}^{(E)}$, $\vec{\boldsymbol{v}}^{(D)}$, $\vec{\boldsymbol{v}}^{(E)}$, $\vec{\boldsymbol{v}}'^{(D)}$, $\vec{\boldsymbol{v}}'^{(E)}$, $\mathbf{ID}^{(D)}$, $\mathbf{ID}^{(E)}$, and message **Compute**. Then, receive $\vec{\boldsymbol{t}}^{(C)}$ from R_C and decrypt the elements. Next, remove the blinding factors. This yields $\vec{\boldsymbol{g}}'$; $1 \leq i \leq n, g_i' \in \vec{\boldsymbol{g}}'$: $g_i' = \tau_E(x_i) \cdot \sigma'(x_i) \cdot \omega_E(x_i) + \tau_D(x_i) \cdot \sigma'(x_i) \cdot \omega_D(x_i)$.

(f) Interpolate a polynomial using the n point-value pairs (x_i, g_i'). Extract the roots of the polynomial. Check whether β' is among the roots. If it is not, abort and instruct the TTP to send abort message $\perp$ to client B. Otherwise, ask TTP to send the result to the client.

(g) Output whatever the adversary outputs and terminate.

First, we consider the adversary's output. In the real model the elements in $\vec{\boldsymbol{o}}^{(A)}$, $\vec{\boldsymbol{o}}^{(B)}$, $\vec{\boldsymbol{v}}^{(B)}$ are blinded by the outputs of a pseudo-random function using random secret keys of length l. The same is true in the ideal model for the elements in $\vec{\boldsymbol{o}}^{(E)}$, $\vec{\boldsymbol{o}}^{(D)}$, $\vec{\boldsymbol{v}}^{(E)}$. Since the outputs of the pseudo-random function are computationally indistinguishable, the distributions of $\vec{\boldsymbol{o}}^{(A)}$, $\vec{\boldsymbol{o}}^{(B)}$, $\vec{\boldsymbol{v}}^{(B)}$ and $\vec{\boldsymbol{o}}^{(E)}$, $\vec{\boldsymbol{o}}^{(D)}$, $\vec{\boldsymbol{v}}^{(E)}$ are computationally indistinguishable, too. If the homomorphic encryption is semantically secure then $\vec{\boldsymbol{v}}^{(A)}$,$\vec{\boldsymbol{v}}'^{(A)}$,$\vec{\boldsymbol{v}}'^{(B)}$ and $\vec{\boldsymbol{v}}^{(D)}$,$\vec{\boldsymbol{v}}'^{(D)}$,$\vec{\boldsymbol{v}}'^{(E)}$ are computationally indistinguishable. Moreover, in both models, the protocol outputs Λ (i.e. empty) to the adversary. Therefore, we conclude that the adversary's outputs in both models are computationally indistinguishable.

Now we consider client B's output. We show the honest client B aborts with the same probability in both models. In the ideal model, if the cloud misbehaves, SIM_C would detect it with a high probability according to Theorem 1. In this case it will send $\perp$ to the client and accordingly the client will abort. Note that in this case SIM_C has not found the value β' in the intersection. In the real model, since the client knows the value β he can do the same checks that SIM_C does. So, in both models the client aborts with the same probability if the cloud misbehaves. Finally, since client A has no output, her output is identical in both models.

From the above we conclude that:

$$\mathsf{IDEAL}_{F,SIM_{C(z)}}(\Lambda, S_A, S_B) \stackrel{c}{\equiv} \mathsf{REAL}_{\pi,R_{C(z)}}(\Lambda, S_A, S_B).$$

Case 2: Client B is corrupted. In this case we consider a semi-honest adversary that controls client B. In the real execution, the joint outputs of the parties include only client B's view containing vector $\vec{\boldsymbol{t}}^{(C_3)}$, where the vector comprises the set intersection. Now we construct a simulator, SIM_B in the ideal model. The simulator executes the following tasks.

(a) Invoke adversary R_B, and receive $\vec{\boldsymbol{e}}^{(B)}, S_B, \beta, k_a^{(B)}, k_b^{(B)}, k_{r'}^{(B)}, k_z^{(B)}$ from it.
(b) Send S_B to TTP and receive the result $f_\cap(S_A, S_B)$. Pick two random sets S_E and S_D, where $S_E \cap S_D = f_\cap(S_A, S_B)$. Construct two polynomials $\tau_E(x)$ and $\tau_D(x)$ representing set S_E and S_D, respectively.
(c) Pick two uniformly random polynomials, $\omega_E(x)$ and $\omega_D(x)$ of degree $d+1$.
(d) Generate $\vec{\boldsymbol{t}}$ containing t_i such that $t_i = (e_i^{(B)})^{\omega_E(x_i)\cdot\tau_E(x_i)+\omega_D(x_i)\cdot\tau_D(x_i)+a_i^{(B)}+b_i^{(B)}}$ where $a_i^{(B)} = f(k_a^{(B)}, i)$, $b_i^{(B)} = f(k_b^{(B)}, i)$.
(e) Feed $\vec{\boldsymbol{t}}$ to R_B. Output whatever the adversary outputs.

Since the other parties have output Λ (i.e. no output), we only need to consider the adversary's view. Given the output vector, the adversary decrypts and unblinds the elements. Next it interpolates a polynomial which is of the form $\omega_E \cdot \tau_E + \omega_D \cdot \tau_D = \mu \cdot gcd(\tau_E, \tau_D)$, where μ is a uniformly random polynomial, and $gcd(\tau_E, \tau_D)$ represents the intersection of the sets (see subsection 3.3). Therefore, the result polynomial only contains the information of the set intersection and has the same distribution in both models, as the uniformly random polynomials ω_A and ω_B are chosen by an honest party. Also, the value β has the same distribution in both models.

From the above argument we conclude that:

$$\mathsf{IDEAL}_{F,SIM_{B(z)}}(\Lambda, S_A, S_B) \stackrel{c}{\equiv} \mathsf{REAL}_{\pi,R_{B(z)}}(\Lambda, S_A, S_B).$$

Case 3: Client A is corrupted. This is a trivial case, because client A has no output, and she receives a set of uniformly random values and a vector of encrypted values using semantically secure encryption scheme. A simulator can always be constructed. □

6 Evaluation

We evaluate VD-PSI by comparing its properties to those protocols that support verifiable delegated PSI [18, 24] and preserve the privacy of the intersection in the cloud. We also compare the protocols in terms of communication, computation and verification complexity. Table 1 summarises the results.

Properties. All the three protocols support multiple clients. In [24] the clients can securely delegate an arbitrary computation to the cloud an unlimited number of times. Nevertheless, every client needs to receive all the (hash values of encrypted) inputs of the computation to verify the result. Therefore, the protocol is not suitable for cases where the number of clients or the size of datasets is large. In [18] the clients need to interact with each other in order to jointly generate a key for the pseudo-random function used to encode the datasets. Also, the clients need to re-encode their datasets each

Property	VD-PSI	[18]	[24]
Non-interactive Setup	✓	×	✓
Many Set Intersections Without Re-preparation	✓	×	✓
Multiple Clients	✓	✓	✓
Computation Integrity Verification	✓	✓	✓
Supporting Arbitrary Computation	×	×	✓
Using Expensive Generic Proof Systems (e.g. Zero Knowledge)	×	×	✓
Overall Communication Complexity	$O(d)$	$O(d)$	$O(m)$
Overall Computation Complexity	$O(d)$	$O(d)$	$O(m)$
Verification Computation Complexity	$O(k)$	$O(\lambda k)$	$O(m)$

Table 1: Comparison of the properties of verifiable delegated PSI protocols. We denote set cardinality by d, set intersection cardinality by k, sum of the cardinality of all sets by m, and the security parameter by λ.

time they compute the intersection. So, the protocol does not support outsourcing of the datasets.

In VD-PSI, the clients can independently prepare and outsource the datasets to the cloud. Furthermore, once the clients upload their datasets, they can securely delegate PSI to the cloud an unlimited number of times. As a result, they do not need to download and re-encode the outsourced datasets every time they delegate the computation.

Thus, VD-PSI and [24] support verifiable delegated PSI over outsourced private datasets; whereas, [18] does not, as it lacks some of the properties.

Communication Complexity. In VD-PSI the communication complexity for client B who receives the result is $O(d)$, where d is the set cardinality; as client B sends to client A vector $\vec{\boldsymbol{e}}^{(B)}$ containing $n = 2d + 3$ encrypted values (see step 3c). The communication complexity for client A who grants the computation is $O(d)$, because the client sends to the cloud $\vec{\boldsymbol{v}}^{(A)}$, $\vec{\boldsymbol{v}}'^{(A)}$, $\vec{\boldsymbol{v}}'^{(B)}$, $\vec{\boldsymbol{v}}^{(B)}$ where each of the first three vectors contains n encrypted elements and the last one contains n random elements of the field (see step 3f). The communication complexity for the cloud is $O(d)$, as it sends to client B vector $\vec{\boldsymbol{t}}^{(C_3)}$ that contains n encrypted elements (see step 4d). Hence, the overall communication complexity of our protocol is $6n$, which is linear to the dataset size, so it is $O(d)$.

The protocol in [18] has also $O(d)$ communication complexity. In [24], two protocols dealing with a malicious adversary are proposed. The overall communication complexity of each protocol is linear to the total number of computation inputs m, i.e. $O(m)$. In one of the protocols, the cloud broadcasts all the encrypted inputs to the clients; while in the other, the cloud broadcasts the hash value of the encrypted inputs.

Although all the three protocols have overall communication complexity linear to the dataset size, most messages in VD-PSI are ciphertexts of Paillier encryption, in [24] ciphertexts of fully homomorphic encryption, and in [18] ciphertexts of symmetric key encryption.

Computation Complexity. Since computation complexity of VD-PSI is dominated by the exponentiation operations, we evaluate its computational cost by counting the number of such operations. Client B in step 3b carries out n exponentiations to encrypt the elements of $\vec{\boldsymbol{e}}^{(B)}$. Furthermore, in step 5a he performs n exponentiations to decrypt the elements of $\vec{\boldsymbol{t}}^{(C_3)}$. Client A carries out n exponentiations in steps 3d and $2n$

exponentiations in step 3e. The cloud carries out $2n$ exponentiations in step 4a, $2n$ exponentiations in step 4b, and n exponentiations in step 4c. In total, $10n$ exponentiation operations are carried out, so the overall computation complexity is linear to the dataset size, i.e. $O(d)$. In VD-PSI a verifier only checks whether β is among the elements of the intersection. Therefore, the verification computation complexity is at most linear to the intersection cardinality, i.e. $O(k)$.

The computation complexity of the two protocols dealing with a malicious adversary in [24] is dominated by fully homomorphic encryption operations. In each protocol, the overall number of such operations is linear to the size of the inputs m. So, the overall computation complexity for each protocol is $O(m)$. Moreover, in order for each client to verify the computation correctness, he needs to access all the (encrypted) inputs and perform generic proof system operations linear to the number of inputs. Therefore, the verification complexity at the verifier side is $O(m)$, too. While, in [18] each participant has overall computation complexity linear to the dataset size d, i.e. $O(d)$. In order for the client to verify the integrity of the result, he checks whether λ copies of all intersection elements exist in the result. So, its verification complexity is $O(\lambda k)$ where λ and k are the security parameter and intersection cardinality, respectively.

Thus, all the schemes have overall computation complexity linear to the dataset size; while, VD-PSI uses Paillier encryption, [24] uses fully homomorphic, and [18] uses symmetric key encryption. The verification mechanisms in [24] is based on expensive generic proof systems, while [18] and VD-PSI use lightweight mechanisms.

Remark. In VD-PSI after the client outsources its dataset, it has to keep locally only two secret keys. During the computation, the client who is interested in the result generates four values that he needs to keep until he retrieves the result. At the end of the protocol he can discard the four values. In contrast, a variant of the protocol in [24] requires the client to have the hash value of his encrypted inputs for verification. This introduces a storage overhead linear to the number of his outsourced inputs. Similar to VD-PSI, the clients in [18] need to locally store only the secret keys for a pseudo-random function in order to verify the computation result.

In conclusion, although [18] is faster than VD-PSI (and [24]), VD-PSI enjoys two properties that [18] lacks. First, VD-PSI supports non-interactive setup at client-side. Second, it allows clients to upload their datasets once but verifiably delegate the computation to the cloud an unlimited number of times. These properties are vital, because in the real world individuals and businesses can upload their datasets to the cloud at different points in time without necessarily knowing the other cloud users. Also, the users can fully benefit from the cloud's storage and computation capabilities. Compared to [24], VD-PSI offers the same security properties much more efficiently. Thus, VD-PSI allows businesses to get the full benefits of the cloud in a more cost-effective way.

7 Conclusions and Future Work

Integrity and privacy of data and computation results are major concerns for clients using the cloud. In this work we proposed VD-PSI, an efficient protocol that allows a client to delegate both storage and computation of private set intersection to the cloud who may misbehave. VD-PSI allows a result recipient to efficiently detect if the cloud

tampers with the datasets, or deviates from the protocol, even though the client does not know its own outsourced dataset and the other clients' datasets. The protocol allows clients to independently prepare and store their private datasets in the cloud, and later on ask the cloud to compute the intersection of their outsourced datasets. It ensures that the cloud can compute the intersection only when all the clients agree. Clients can delegate PSI computation over their outsourced datasets an unlimited number of times with no need to download and re-prepare the datasets. We have shown that our protocol is secure in the presence of a malicious cloud and semi-honest clients. Its communication and computation complexity is linear to the dataset size, and its verification mechanism is lightweight. Overall, VD-PSI is a clear step forward towards efficient verifiable delegated PSI on outsourced private datasets.

In the future we would like to investigate how to further improve the efficiency of the protocol and how to support other privacy preserving delegated set operations on outsourced private datasets such as set difference, set union and subset.

Acknowledgments 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/M013561/1).

References

1. Abadi, A., Terzis, S., Dong, C.: O-PSI: delegated private set intersection on outsourced datasets. In: ICT Systems Security and Privacy Protection - 30th IFIP TC 11 International Conference, SEC 2015, Germany. pp. 3–17 (2015)
2. Aggarwal, C.C., Yu, P.S. (eds.): Privacy-Preserving Data Mining - Models and Algorithms, Advances in Database Systems, vol. 34. Springer (2008)
3. Aho, A.V., Hopcroft, J.E.: The Design and Analysis of Computer Algorithms. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edn. (1974)
4. Ardagna, C.A., Asal, R., Damiani, E., Vu, Q.H.: From security to assurance in the cloud: A survey. ACM Comput. Surv. 48(1), 2:1–2:50 (Jul 2015)
5. BBC-NEW: The interview: A guide to the cyber attack on hollywood. http://www.bbc.co.uk/news/entertainment-arts-30512032
6. Choi, S.G., Katz, J., Kumaresan, R., Cid, C.: Multi-client non-interactive verifiable computation. In: TCC. pp. 499–518 (2013)
7. Cristofaro, E.D., Kim, J., Tsudik, G.: Linear-complexity private set intersection protocols secure in malicious model. In: Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security. pp. 213–231 (2010)
8. Dong, C., Chen, L., Camenisch, J., Russello, G.: Fair private set intersection with a semi-trusted arbiter. In: Data and Applications Security and Privacy XXVII - 27th Annual IFIP WG 11.3 Conference, DBSec 2013, USA. pp. 128–144 (2013)
9. Dong, C., Chen, L., Wen, Z.: When private set intersection meets big data: an efficient and scalable protocol. In: 20th ACM Conference on Computer and Communications Security. pp. 789–800 (2013)
10. Fiore, D., Gennaro, R., Pastro, V.: Efficiently verifiable computation on encrypted data. In: 21st ACM Conference on Computer and Communications Security, Scottsdale, AZ, USA. pp. 844–855 (2014)

11. Freedman, M.J., Nissim, K., Pinkas, B.: Efficient private matching and set intersection. In: EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland. pp. 1–19 (2004)
12. Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In: Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, , USA. pp. 465–482 (2010)
13. Goldreich, O.: The Foundations of Cryptography - Volume 2, Basic Applications. Cambridge University Press (2004)
14. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating computation: interactive proofs for muggles. In: Proceedings of the 40th Annual ACM Symposium on Theory of Computing, , Canada. pp. 113–122 (2008)
15. Golle, P., Mironov, I.: Uncheatable distributed computations. In: Topics in Cryptology - CT-RSA 2001, The Cryptographer's Track at RSA Conference 2001, San Francisco, CA, USA. pp. 425–440 (2001)
16. Gordon, S.D., Katz, J., Liu, F., Shi, E., Zhou, H.: Multi-client verifiable computation with stronger security guarantees. In: 12th Theory of Cryptography Conference, TCC , Poland. pp. 144–168 (2015)
17. Huang, W., Ganjali, A., Kim, B.H., Oh, S., Lie, D.: The state of public infrastructure-as-a-service cloud security. ACM Comput. Surv. 47(4), 68:1–68:31 (Jun 2015)
18. Kamara, S., Mohassel, P., Raykova, M., Sadeghian, S.: Scaling private set intersection to billion-element sets. In: 18th International Conference on Financial Cryptography and Data Security. pp. 863–874 (2014)
19. Kedlaya, K.S., Umans, C.: Fast polynomial factorization and modular composition. SIAM J. Comput. 40(6), 1767–1802 (2011)
20. Kerschbaum, F.: Collusion-resistant outsourcing of private set intersection. In: 27th ACM Symposium on Applied Computing, Riva, Trento, Italy. pp. 1451–1456 (2012)
21. Kerschbaum, F.: Outsourced private set intersection using homomorphic encryption. In: Computer and Communications Security, ASIACCS '12. pp. 85–86 (2012)
22. Kissner, L., Song, D.X.: Privacy-preserving set operations. In: CRYPTO 2005, 25th International Cryptology Conference. pp. 241–257 (2005)
23. Liu, F., Ng, W.K., Zhang, W., Giang, D.H., Han, S.: Encrypted set intersection protocol for outsourced datasets. In: IEEE International Conference on Cloud Engineering. pp. 135–140. IC2E '14, IEEE Computer Society, Washington, DC, USA (2014)
24. López-Alt, A., Tromer, E., Vaikuntanathan, V.: On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: Symposium on Theory of Computing Conference, USA. pp. 1219–1234 (2012)
25. Marston, S., Li, Z., Bandyopadhyay, S., Ghalsasi, A.: Cloud computing - the business perspective. In: 44th Hawaii International International Conference on Systems Science (HICSS-44 2011), USA. pp. 1–11 (2011)
26. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic. pp. 223–238 (1999)
27. Pinkas, B., Schneider, T., Segev, G., Zohner, M.: Phasing: Private set intersection using permutation-based hashing. In: 24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015. pp. 515–530 (2015)
28. Pinkas, B., Schneider, T., Zohner, M.: Faster private set intersection based on OT extension. In: 23rd USENIX Security Symposium, San Diego, CA, USA. USENIX (2014)
29. Zheng, Q., Xu, S.: Verifiable delegated set intersection operations on outsourced encrypted data. IACR Cryptology ePrint Archive p. 178 (2014)