SEVDSI: Secure, Efficient and Verifiable Data Set Intersection

Ozgur Oksuz[1], Iraklis Leontiadis[2], Sixia Chen[3], Alexander Russell[4], Qiang Tang[2], and Bing Wang[4]

[1] Washington State University
[2] New Jersey Institute of Technology
[3] Central Connecticut State University
[4] University of Connecticut

Abstract. Private set intersection is one of the most well studied and useful secure computation protocols. Many specific two party secure computation protocols have been constructed for such a functionality, but all of them incur large communication between the parties. A cloud assisted protocol was also considered to provide better efficiency, but with the potential risk of leaking more information to the cloud.
In this paper, we achieve the best of the two worlds: We design and analyze SEVDSI: a Secure, Efficient and Verifiable Data Set Intersection protocol which is non-interactive and cloud based in a stronger security model. Our protocol assures privacy on data set inputs in case of a *malicious* cloud and enforces authorized only computations by the users. Moreover, the computation is verifiable and we achieve $O(m^3)$ asymptotic communication cost for m users in contrast with the fastest two party computation protocols, which obtain a $O(m^4)$ communication complexity, in case of multiparty PSI. SEVDSI is provably secure in the random oracle model.

1 Introduction

Private Set Intersection (PSI) protocols allow two users to learn the intersection of their dataset, without revealing information for the elements not in the intersection set. PSI is applicable to a variety of real world scenarios:

- Airline companies in order to determine whether specific flight passengers appear in a black list, perform a private set intersection operation between the private passenger list and the government's list.
- National agencies (e.g., the FBI) need to obtain information on suspects from other agencies, e.g., local police departments, the military, DMV, IRS, or employers. The FBI may wish to search suspects on other agencies' lists, but no agency wants to divulge members of its suspect list unnecessarily to others.
- In an online advertisement setting, a content provider G has information about users, who access content online. A transaction provider T keeps a

log of users with all the online purchases a user did. In the end, G wants to jointly compute the common users with T, in order to learn the percentage of users who access its content and proceed to a purchase, to further analyze how much the viewing of some content affects spending.

Over the years, due to its importance, researchers have designed numerous PSI protocols [4,7–11,13,16–18,23,24,26–28]. Alas, for any two users to jointly compute the set intersection, there is a communication burden at least proportional to the size of their datasets. Thus, each user must have large network bandwidth to run PSI with others. Communication costs become of paramount importance when the intersection is computed over multiple datasets of many users. Another issue with jointly computing the set intersection is that a malicious user may change its dataset every time it computes the intersection in order to gradually learn others' datasets over time.

A cloud infrastructure with its cost effective storage and computation resources may alleviate the communication burden two users need to exchange. However, the advantage in communication costs with the incorporation of a cloud infrastructure is dampened as the cloud may act maliciously and try to infer plaintext of users' datasets even if datasets are encrypted. The most crucial attacks to infer plaintext datasets from encrypted datasets are brute-force attacks, whereby a malicious or semi-honest adversary leverages public information, such as the ciphertexts of the datasets, to compromise dataset privacy. These adversaries are able to recover any user's dataset after guessing the plaintext from a non-random distribution of the ciphertext. As such users' privacy should be taken into account.

Recent research surveys [3, 15] have shown that the cloud cannot be fully trusted and it may misbehave by exposing or tampering with users' sensitive data, or changing the results of the computation. It is essential though, for users 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.

State of the art work in cloud-based or interchangeable dubbed server-aided solutions enables efficient PSI computation without linear size information to be exchanged in between users [32]. However, to achieve that improvement the cloud has to run the computations with the aid of some auxiliary information, forwarded by each data owner. We realize this information renders the protocol vulnerable to brute-force attacks on the ciphertexts of the datasets. In order to address the above limitations, we re-randomize the final ciphertext with an extra secret key, which is canceled out by the cloud using some auxiliary information that is sent by the users. The auxiliary information is a function of common randomness (secret to the malicious cloud).

Our scheme also uses a multi-accumulator primitive that allows a verifier to test if multiple dataset values that are sent by the cloud are truly part of the original dataset. While a single accumulator scheme allows a verifier to test data item existence one-by-one, the multi-accumulator scheme allows verifier to test multi data items at once. The multi-accumulator value is further signed in order to protect from tampering.

Contributions Our major contributions are threefold.

- First, we design a verifiable PSI protocol, which is communication efficient, without requiring heavy communication in between the users; in contrast with previous work [2, 7, 8, 10, 11, 14, 17–20]. Such an improvement in communication bandwidth is of crucial importance as it will allow deployment of PSI protocol in critical infrastructures as federal institutions and big online advertisement companies with millions of users, whereby the private set intersection is computed over many users and not between two users. In two party computation (2PC) based PSI solutions, each user needs to send n data items to all the other $m-1$ users. The total communication complexity between any two users is $O(nm)$. The overall communication complexity in case of multi-party PSI between all users is $O(nm^2)$. In SEVDSI we achieve communication complexity $O(km^2+mn)$, for k common elements which outperforms 2PC solutions [10, 24, 27–29] in case of multiple users interested in running PSI. As shown in table 1, when the intersection size for each PSI is $k << m$ our protocol reduces communication complexity from $O(m^4)$ to $O(m^3)$.

	SEVDSI $O(km^2 + mn)$	2PC PSI $O(nm^2)$
$n = m, k = c$	$O(m^2)$	$O(m^3)$
$n = m, k = m^{1/2}$	$O(m^{2.5})$	$O(m^3)$
$n = m, k = m$	$O(m^3)$	$O(m^3)$
$n = m^2, k = c$	$O(m^3)$	$O(m^4)$
$n = m^2, k = m$	$O(m^3)$	$O(m^4)$
$n = m^2, k = m^2$	$O(m^4)$	$O(m^4)$

Table 1. Communication complexity comparison of the most communication efficient 2PC based PSI protocols and ours (SEVDSI).

- Second, the proposed scheme does not allow malicious cloud to apply brute-force attacks on the encrypted datasets. As such private information about users datasets are kept secret without compromising its confidentiality.
- Third, in contrast with previous work on cloud based PSI protocols, our PSI protocol does not allow a malicious cloud to run the PSI protocol at will in an unauthorized way without users' permissions.

In this paper, we present a Secure, Efficient and Verifiable Data Set Intersection protocol, dubbed SEVDSI, with the following salient features:

1. *Verifiability:* Users can check that the cloud has honestly computed the set intersection. Moreover, the verification for each user requires an amount of computation at most linear in the cardinality of the set intersection.

2. *Communication Efficiency:* Any two users outsource their encrypted datasets and afterwards they are engaged in a communication round with the cloud which is constant. We call this constant size information as auxiliary information. Moreover, the cloud sends the intersection result, whose size is linear in the set intersection cardinality.
3. *Computational Efficiency:* The verification computation complexity for users is at most linear to the intersection cardinality, i.e. $O(k)$. This is more efficient than the straightforward solution whereby each user needs to access all the encrypted inputs and perform verification operations linear to the number of inputs, i.e. $O(n)$ if $k << n$, where n is the number of values in a users dataset, k is the set intersection cardinality.
4. *Non-Interactive PSI:* Users encrypt, upload and learn the private set intersection of their datasets with the aid of a cloud, avoiding the need to interact with each other during the protocol execution.
5. *Privacy:* Users learn nothing beyond the intersection about other users' datasets. The cloud learns nothing about either datasets or their intersection, beyond the size of datasets and intersection size.
6. *Authorized PSI:* The cloud is allowed to run PSI between datasets owned by users who have granted permission to it. In other words, the cloud needs to be authorized by both users to execute the PSI protocol.

Outline: In Section 2 we introduce state-of-art PSI schemes in the 2PC, and the cloud setting. We present the overview of the cryptographic primitives and assumptions used for SEVDSI in Section 3. Section 4 formulates the problem and the security guarantees. In Section 5, we present an overview of the protocol, its full details with the security and efficiency analysis.

2 Related Work

Two-Party Computation (2PC) based PSI: The first study that introduces PSI [11] is based on oblivious polynomial evaluation (OPE). Then, a plethora protocols of PSI [4, 7–10, 13, 16–18, 23, 27, 28] has been proposed with different adversarial models: Semi-honest, malicious, and covert adversarial models. In the semi-honest adversarial model, users faithfully follow the protocol specification and do not change the content of their datasets. In the malicious adversarial model, malicious users can arbitrarily deviate from the protocol. They can change their inputs or respond with different outputs. In the covert adversarial model, adversaries are willing to actively cheat (and as such are not semi-honest), but only if they will not be caught (and as such they are not arbitrarily malicious) [5]. The protocols in [4, 6, 9, 16, 23, 27, 28] are secure in the presence of semi-honest adversaries, while solutions in [7, 8, 10, 11, 14, 17, 18, 29] are secure in the presence of malicious adversaries and [13] is secure against covert adversaries. The studies in [4,6] hide the users' input sets size. Authors in [13] propose a different approach, designing a PSI protocol based on oblivious pseudorandom function evaluation, which is later improved by [9, 17]. Garbled circuits for PSI introduced in [16], while [10] employs garbled bloom filters and has $O(n)$ communication and computation complexity but introduces false positives. In [6], users need an extra round of communication. The size of communication is linear in a bound t that is chosen by the user. The user is able to check at most t of his data values against the other user's datasets. The recent study in [31] proposed $t-$threshold private set intersection. Using this primitive, two parties, a server and a client can perform the PSI computation using their datasets as long as the intersection is larger than a pre-agreed threshold value t. A tailored solution for unequal datasets is proposed in [22]. The authors proposed 4 protocols based on blind RSA signatures, Diffie-Hellman, garbled circuits and oblivious PRF in order to eliminate online computational and communication costs and push the linear to the size of the smallest dataset costs in an offline phase.

	Scheme	Verify	PPR	FPR	U-U	CS-U	U-CS
2PC	[11]	yes	yes	yes	$O(n\lambda)$	n/a	n/a
	[23]	no	yes	yes	$O(n\lambda)$	n/a	n/a
	[8]	yes	yes	yes	$O(n\lambda^2 \log^2 n)$	n/a	n/a
	[17]	yes	no	yes	$O(n\lambda)$	n/a	n/a
	[18]	yes	yes	yes	$O(n\lambda)$	n/a	n/a
	[7]	yes	no	yes	$O(n\lambda)$	n/a	n/a
	[14]	yes	yes	yes	$O(n\lambda)$	n/a	n/a
	[9]	no	no	yes	$O(n\lambda)$	n/a	n/a
	[4]	no	no	yes	$O(n\lambda)$	n/a	n/a
	[16]	no	yes	yes	$O(n\lambda \log n)$	n/a	n/a
	[10]	yes	yes	no	$O(n\lambda^2)$	n/a	n/a
	[28]	no	yes	yes	$O(n\lambda \log n)$	n/a	n/a
	[27]	no	yes	yes	$O(n\lambda \log n)$	n/a	n/a
	[6]	no	yes	yes	$O(n\lambda)$	n/a	n/a
Cloud	[20]	yes	no	yes	communication free	$O(n\lambda)$	$O(n\lambda)$
	[19]	yes	no	yes	$O(k\lambda)$	$O(n\lambda)$	$O(n\lambda)$
	[21]	no	no	no	communication free	$O(n\lambda)$	$O(n^2\lambda t)$
	[25]	no	yes	yes	communication free	$O(n\lambda)$	$O(n\lambda)$
	[32]	yes	yes	yes	communication free	$O(k\lambda)$	$O(1)$
	[1]	no	no	yes	$O(n\lambda)$	$O(n\lambda)$	$O(n\lambda)$
	[2]	yes	no	yes	$O(n\lambda)$	$O(n\lambda)$	$O(n\lambda)$
	SEVDSI	**yes**	**yes**	**yes**	**communication free**	$\mathbf{O(k\lambda)}$	$\mathbf{O(1)}$

Table 2. Comparison of protocols in the 2PC, and Cloud settings: λ is the security parameter, n is the number of values in a user's dataset, k is the set intersection cardinality. The column *U-CS* denotes the size of the information sent to the cloud from users after outsourcing of the encrypted dataset, the column *CS-U* is the size of the information sent to users from the cloud. The column *PPR* applies to protocols where users do not share secret values with each other interactively or there is not a trusted third party that provides secret values to users to encrypt their datasets. The column *U-U* indicates the total communication between two users and the column *FPR* indicates whether a protocol is a false positive resilient protocol. Some of the protocols use a bloom filter. In those cases, t represents the number of hash functions that are used in the bloom filter.

Cloud-based PSI: The studies in [19, 20] show that users share some secrets interactively to encrypt their datasets. Later, they send their encrypted datasets to a cloud. The cloud computes the set intersection on behalf of the users. Kerschbaum [20] and Kamara et al. [19] propose protocols whereby the users need to agree on a common secret key and to send $O(n)$ information to the cloud for each intersection operation. Another flaw of these schemes is that a honest but curious user can brute force other users' datasets. To alleviate the brute force attacks a trusted third party authorizes users' datasets to prevent them from changing their input values. The protocols in [19, 20] need linear $O(n)$ communication complexity between cloud and the user. The protocol in [19] introduces extra $O(n)$ interaction between users. Our scheme does not require any collaborative preprocessing. Users outsource their encrypted datasets only once and only need to engage in minimal communication with the cloud on a per-intersection basis.

The studies in [21, 25] propose PSI protocols, in which the cloud is semi-honest. Furthermore, the authors in [21] proposed a protocol that may yield false positives and requires quadratic communication complexity. Authors in [32] suggest a verifiable set intersection protocol secure under a malicious cloud: If the cloud dishonestly executes the set intersection protocol, it is caught with high probability at the cost of $O(k)$ communication complexity between users and the cloud, where k is the set intersection cardinality. However, their protocol leaks plaintexts and does not have the authorized PSI property. In this paper we mitigate these shortcomings.

The study in [1] proposes a secure private set intersection protocol, which does not provide verifiability. A later study by the same authors in [2] provides verifiability, but it is not efficient. Specifically, if user A wants to know which data items are common with user B's data items, A sends a secret value and $O(n)$ information to B. Then, B performs $O(n)$ internal computations in order to embed the secret value (chosen by A) in his dataset, and then sends the resulting values to the cloud. An inherent drawback of this scheme is that B may not agree on the secret information chosen by A. The second drawback in [1] is that B performs $O(n)$ computation and sends $O(n)$ information to the cloud on behalf of A. In other words, user B does the heavy work. Even worse in case m users want to engage in a PSI with A or B, the latter needs to perform $O(mn)$ computations and send $O(mn)$ information to the cloud.

3 Preliminaries and Complexity Assumptions

In this section we present the cryptographic primitives used in our protocol and the underlying mathematical assumptions.

3.1 Cryptographic primitives

Bilinear Maps: Let G and G' denote two multiplicative cyclic groups of prime order p and let g be a generator of G. A map $e : G \times G \to G'$ is *bilinear* if it has the following properties: (1) for all $u, v \in G$ and $a, b \in \mathbb{Z}_p$, we have

$e\left(u^a, v^b\right) = e\left(u, v\right)^{ab}$; (2) the map is not degenerate, i.e., $e\left(g, g\right) \neq 1$, and (3) e is an efficiently computable function.

Unforgeable Digital Signature [12]: A digital signature scheme, Sig, consists of three algorithms, Sig = (sigKeyGen, sigSign, sigVerify), where sigKeyGen generates public and private keys $sigPK, sigSK$, sigSign generates a signature for a message, and sigVerify determines if a signature is generated under the corresponding message. We say that a digital signature scheme is secure if the signature scheme is existentially unforgeable under adaptive chosen message attack (UF-CMA). UF-CMA means that an adversary who is given signatures for some messages of its choice adaptively should not be able to produce a signature for a new message. Any signature scheme satisfying the standard definition of UF-CMA can be used in our construction.

Multi-Accumulator [32]: We present the details of the multi-accumulator scheme that we use in our protocol as presented in [32]. A multi-accumulator scheme can be based on a single-accumulator scheme that supports both membership and non-membership proofs, as follows: the cloud generates a witness for each element of D_b showing the element is a member or non-member of D_a and simply puts them together as the witness for $acRslt = D_b \cap D_a$. However, this straightforward approach is costly because both the computational and communication complexities are linear to $|D_b|$. In SEVDSI we employ the multi-accumulator scheme in [32], where the size of the witness is constant which does not depend on the cardinality of the sets, D_a (D_b). Suppose user A's data set is $D_a = \{d_{a,0}, \ldots, d_{a,n-1}\}$, user B's data set is $D_b = \{d_{b,0}, \ldots, d_{b,m-1}\}$ (we assume that $n = m$), and $acRslt = D_a \cap D_b$. We can encode D_a via polynomial $R(x) = \prod_{t \in D_a}(x + t)$, encode D_b via polynomial $W(x) = \prod_{t \in D_b}(x + t)$, encode the intersection set $acRslt$ via polynomial $T(x) = \prod_{t \in acRslt}(x + t)$, and encode the subset $D_b - acRslt$ via polynomial $Q(x) = \prod_{t \in (D_b - acRslt)}(x + t)$. These polynomials satisfy the following: (i) $T(x)Q(x) = W(x)$, (ii) $T(x)$ is a divisor of $R(x)$, and (iii) $Q(x)$ is co-prime to $R(x)$. For the special case $acRslt = \emptyset$, the three conditions also hold since $T(x) = 1$, $Q(x) = W(x) = \prod_{t \in D_b}(x + t)$ and $R(x) = \prod_{t \in D_a}(x + t)$. Therefore, based on this idea, the multi-accumulator scheme allows the cloud to show the correctness of the intersection set, which can be either empty or non-empty. It can be constructed as follows:

- **acKeyGen(λ)**: This algorithm takes security parameter λ, outputs system parameters $acPk = (g^{\alpha}, g^{\alpha^2}, \ldots, g^{\alpha^q})$, where random $\alpha \in \mathbb{Z}_p$, $acSk = (\alpha)$, where q is bounded by a polynomial in security parameter λ.
- **acGen($acPk, D_a$)**: Given user A's data set $D_a = \{d_{a,0}, \ldots, d_{a,n-1}\} \in Z_p^n$, where $n \leq q$, compute his digest as $s_a = g^{\prod_{i=0}^{n-1}(d_{a,i}+\alpha)}$.
- **acProve($acPk, D_b, D_a$)**: Given user B's data set $D_b = (d_{b,0}, \ldots, d_{b,m-1}) \in Z_p^m$, where $m \leq q$, compute $acRslt = D_b \cap D_a$, and generate a witness as follows:
  - Let $T'(x) = \prod_{t \in (D_a - acRslt)}(x + t)$ and compute $g^{T'(\alpha)}$ by substituting x with α.

- Let $Q(x) = \prod_{t \in (D_b - acRslt)}(x+t)$ and $R(x) = \prod_{t \in D_a}(x+t)$, and find two polynomials $Q'(x)$, $R'(x)$ such that $Q(x)Q'(x)+R(x)R'(x) = 1 \bmod p$ by taking advantage of $gcd(Q(x), R(x)) = 1$. Compute $(g^{Q(\alpha)}, g^{Q'(\alpha)}, g^{R'(\alpha)})$ by substituting x with α. Set $acRslt = D_b \cap D_a$ and $acWit = (g^{Q(\alpha)}, g^{Q'(\alpha)}, g^{R'(\alpha)}, g^{T'(\alpha)})$.

– **acVerify**$(acPk, s_b, acRslt, acWit, s_a)$: Given $acWit$ and $acRslt$ from the prover, the verifier proceeds as follows:
  1. If $acRslt \neq \emptyset$, compute $g^{T(\alpha)}$ according to $T(x) = \prod_{t \in acRslt}(x+t)$. Otherwise, let $T(x) = 1$ and $g^{T(\alpha)} = g$.
  2. If $e(g^{Q(\alpha)}, g^{T(\alpha)}) \neq e(s_b, g)$, return 0; otherwise, proceed to next step.
  3. If $e(g^{T(\alpha)}, g^{T'(\alpha)}) \neq e(s_a, g)$, return 0; otherwise, proceed to next step.
  4. If $e(g^{Q(\alpha)}, g^{Q'(\alpha)})e(s_a, g^{R'(\alpha)}) \neq e(g, g)$, return 0; otherwise, return 1.

Definition 1. *A multi-accumulator scheme is correct if the following holds:*

$$\Pr\left[\begin{array}{c} \forall D_a, D_b, \\ (acSk, acPk) \leftarrow acKeyGen(\lambda), \\ s_b \leftarrow acGen(acPk, D_b), \\ s_a \leftarrow acGen(acPk, D_a), \\ (acRslt, acWit) \leftarrow acProve(acPk, D_b, D_a): \\ 1 \leftarrow acVerify(acPk, s_b, acRslt, acWit, s_a) \end{array}\right] = 1$$

Definition 2. *A multi-accumulator scheme is secure if*

$$\Pr\left[\begin{array}{c} (acPk, acSk) \leftarrow acKeyGen(\lambda), \\ D_a \leftarrow \mathcal{A}^{acProve, acVerify}(acPk), \\ s_a \leftarrow acGen(acSk, D_a), \\ (D_b, acRslt, acWit) \leftarrow \mathcal{A}^{acProve, acVerify}(acPk, D_a): \\ s_b \leftarrow acGen(acPk, D_b), \\ 1 \leftarrow acVerify\left(acPk, s_b, acRslt, acWit, s_a\right), \\ acRslt \neq D_b \cap D_a \end{array}\right]$$

$$\leq \epsilon(\lambda)$$

The security proof is based on the q-SDH assumption. For the details of the proof the reader can defer to Zheng *et al.* paper [32, Theorem 1].

3.2 Assumptions

We denote as λ the security parameter and $\epsilon(\lambda)$ a negligible function on input the security parameter λ.

Bilinear q-strong Diffie-Hellman assumption (q-SDH): For given $(g, g^{\alpha}, .., g^{\alpha^q})$, where $\alpha \in \mathbb{Z}_p$, and q is bounded by a polynomial in λ, there exists no PPT algorithm $\mathcal{A}$ that can compute $(s, e(g,g)^{1/(\alpha+s)})$, where $s \in \mathbb{Z}_p$

with non-negligible probability ϵ. The probability is defined over the random choices of parameters and random coins used by $\mathcal{A}$. The advantage of $\mathcal{A}$ is

$$\Pr[\mathcal{A}(g, g^{\alpha}, \dots, g^{\alpha^q}) = (s, e(g,g)^{1/(\alpha+s)})]$$

Definition 3. *q-SDH holds over bilinear groups G, G' if no polynomial time adversary $\mathcal{A}$ has advantage in breaking q-SDH greater than $\epsilon(\lambda)$.*

Variant Decisional Diffie-Hellman Assumption ($VDDH$): For given

$$T = \left(g, v, g^{\gamma_a}, g^{\gamma_b}, g^{\beta_a}, g^{\beta_b}, g^{r\beta_a}, g^{r\beta_b}, g^{r\frac{\beta_a}{\gamma_a}}, g^{r\frac{\beta_b}{\gamma_b}}, Z\right)$$

and γ_a,γ_b,β_a,β_b,r uniformly at random elements in $\mathbb{Z}_p$, generator $g \in G$ of prime order p and v any element in G, $VDDH$ assumption asks an adversary $\mathcal{A}$ to distinguish $Z = v^r$ from a value $g^c \in G, c \xleftarrow{\$} \mathbb{Z}_p$ with non-negligible probability ϵ. The advantage of an algorithm $\mathcal{A}$ in distinguishing v^r from $g^c, c \xleftarrow{\$} \mathbb{Z}_p$ is

$$\left|\Pr\left[\mathcal{A}\left(T, v^r\right) = 1\right] - \Pr\left[\mathcal{A}\left(T, g^c\right) = 1\right]\right|.$$

Definition 4. *$VDDH$ holds in G if no polynomial time advesary can achieve non-negligible advantage in deciding correctly the $VDDH$ assumption.*

To provide some security confidence for the VDDH assumption we show a lower bound on the computational complexity of an adversary $\mathcal{A}$ attacking the VDDH problem in the generic group model (GGM) as presented by Shoup [30]. The idea of the GGM model is to "mirror" the elements of bilinear groups with random encodings accessible to an adversary through random encoding injective function $\xi : \mathbb{Z}_p \rightarrow \{0,1\}^*$ for elements in G. That is, ξ maps elements from $\mathbb{Z}_p$ through G to random encoding string in $\{0,1\}^*$. We write $\xi(x)$ to represent the encoding of g^x. Let $\beta_a, \beta_b, \gamma_a, \gamma_b, c, r \xleftarrow{\$} \mathbb{Z}_p, T_0 = v^r, T_1 = g^c$ and $d \leftarrow \{0,1\}$. $\mathcal{A}$ receives the encodings of $g, v, g^{\gamma_a}, g^{\gamma_b}, g^{\beta_a}, g^{\beta_b}, g^{r\beta_a}, g^{r\beta_b}, g^{r\frac{\beta_a}{\gamma_a}}, g^{r\frac{\beta_b}{\gamma_b}}, T_d, T_{d-1}$. Finally we show that after at most q queries, $\mathcal{A}$ can guess d with a probability no greater than $\frac{1}{2} + O(\frac{q^2}{p})$.

Theorem 1. *Suppose $\mathcal{A}$ is an algorithm that solves the VDDH problem, making at most q oracle queries for the group operations in G and G'. Suppose that the integers $\beta_a, \beta_b, \gamma_a, \gamma_b, r$ and the encoding function ξ are chosen at random. Then*

$$Pr\left[\begin{array}{c}\mathcal{A}\left(\begin{array}{c}p, \xi(1), \xi(u), \xi(\beta_a), \xi(\beta_b), \xi(\gamma_a), \\ \xi(\gamma_b), \xi(r\beta_a), \xi(r\beta_b), \xi(\frac{r\beta_a}{\gamma_a}), \\ \xi(\frac{r\beta_b}{\gamma_b}), \xi(t_0), \xi(t_1)\end{array}\right) = d : \\ c, r, \beta_a, \beta_b, \gamma_a, \gamma_b, \xleftarrow{\$} \mathbb{Z}_p^*, \\ d \leftarrow \{0,1\}, t_d \leftarrow v^r, t_{1-d} \leftarrow g^c\end{array}\right] - \frac{1}{2}$$

$$\leq \frac{(q+11)^2}{p}$$

Proof. We show how a challenger $\mathcal{B}$ interacts with $\mathcal{A}$. During the interaction $\mathcal{B}$ encodes elements with the random encoding function and answers algebraic operations in group G with the encoding of the result.

$\mathcal{B}$ defines polynomial $F_1 \in \mathbb{Z}_p[B_a, B_b, \Gamma_a, \Gamma_b, T_0, T_1], i \in \{1, \tau\}$ with determinants $B_a, B_b, \Gamma_a, \Gamma_b, T_0, T_1 \in \mathbb{Z}_p$. It also stores the list $L_1 = \{F_{1,i}, \xi_{1,i} : i = 0, \ldots, \tau_1\}$, with the invariant that at each step τ, $\tau_1 = \tau + 11$. Initially $F_{1,0} = 1, F_{1,1} = B_a, F_{1,2} = B_b, F_{1,3} = \Gamma_a, F_{1,4} = \Gamma_b, F_{1,5} = RB_a, F_{1,6} = RB_b, F_{1,7} = R\frac{B_a}{\Gamma_a}, F_{1,8} = R\frac{B_b}{\Gamma_b}, F_{1,9} = T_0, F_{1,10} = T_1$, so as to $\tau_1 = 11$.

At the beginning of the game $\mathcal{A}$ gets the random encodings $\xi_{1,2}, \ldots, \xi_{1,13}, \xi_{\tau,1} = \xi'(t_0)$. These random encodings are mapped to the random polynomial F_{1,τ_1} by $\mathcal{B}$. The polynomial is kept secret and never exposed to $\mathcal{A}$. Whenever $\mathcal{A}$ asks for group operations $\mathcal{B}$ simulates it as follows:
Group Operations: For any two operands ξ_i, ξ_j with $0 \leq i, j < \tau_1$ $\mathcal{B}$ computes $F_{1,\tau_1} \leftarrow F_{1,\tau_i} + / - F_{1,\tau_j}$ depending the sign action be it multiplication or division. $\mathcal{B}$ checks if $F_{1,\tau}$ exists in the list L_1 and returns that ξ_{τ_1} to $\mathcal{A}$. Otherwise sets $\tau_1 = \tau_1 + 1$, returns $\xi_{\tau_1} \xleftarrow{\$} \{0,1\}^*$ to $\mathcal{A}$ and adds $(F_{1,\tau_1}, \xi_{\tau_1})$ to the list L_1.

Eventually $\mathcal{A}$ outputs its guess $d' \in \{0,1\}$. Notice that at any time the maximum degree of any F_1 is at most 2. $\mathcal{B}$ assigns random elements $\beta_a, \beta_b, \gamma_a, \gamma_b \xleftarrow{\$} \mathbb{Z}_p$ and sets $t_0 \leftarrow v^r, t_1 \leftarrow g^c$ for the variables $B_a, B_b, \Gamma_a, \Gamma_b, T_0, T_1$. We will bound the success probabilities of randomly assigning values to the polynomial F_1 with the Schwartz-Zippel lemma. Namely, in order $\mathcal{A}$ to guess correctly the following should hold:

$$F_{1,i}(\beta_a, \beta_b, \gamma_a, \gamma_b, r, c) - F_{1,j}(\beta_a, \beta_b, \gamma_a, \gamma_b, r, c) = 0$$

, for $F_{1,i} \neq F_{1,j}$. The success probability of $\mathcal{A}$ is thus: $\epsilon \leq \binom{\tau_1}{2}\frac{2}{p}$. It is also true that: $\tau_1 \leq \tau + 11$. Plugging $q = \tau$ where q represents the total number of queries it holds that:

$$\epsilon \leq \frac{(q+11)^2}{p}$$

□

4 Definitions

In this section we give the definitional framework of our secure and efficient verifiable set intersection protocol (SEVDSI) and we present its privacy and security definitions.

4.1 SEVDSI Function Definition

Users first send their encrypted datasets to the cloud, later they run the PSI protocol with the aid thereof. Let $D_a = \{d_{a,0}, ..., d_{a,n-1}\}$ denote A's plaintext dataset and let $D_b = \{d_{b,0}, ..., d_{b,n-1}\}$ denote B's plaintext dataset. We assume that users A and B have the same number of elements in their datasets, for the sake of simplicity. User A (B) outsources his encrypted dataset C_a (C_b)

to the cloud. When users A and B want to compute $D_a \cap D_b$, they delegate the computation to the cloud by providing auxiliary information au_a (au_b) to compute the set intersection. As an important note, the common elements do not leak any information about any plain data to the cloud. These values are still encrypted.

Definition 5. *A SEVDSI scheme comprises seven algorithms:*

- $pm \leftarrow$ *Setup*(λ)*: This algorithm is run by a trusted third party. It takes a security parameter λ and outputs system public parameters pm.*
- $(pk_a, sk_a) \leftarrow$ *KeyGen*(pm)*: It is a randomized algorithm run by user A. It takes the system public parameter (pm) and outputs a pair of public and private keys (pk_a, sk_a), where pk_a is made public and sk_a is kept secret by A. Similarly, user B generates his public and private key pairs, (pk_b, sk_b).*
- $C_a \leftarrow$ *Enc*($(sk_a, pk_a), D_a$)*: This algorithm takes dataset D_a, public/secret key pair, (sk_a, pk_a) of A and outputs ciphertext C_a, which is outsourced to the cloud. B generates C_b similarly.*
- $(au_a, s_a) \leftarrow$ *AuGen*(sk_a, D_a, pm, pk_b)*: User A authorizes the cloud to conduct the set intersection operation on the outsourced ciphertexts C_a and C_b with AuGen algorithm. This algorithm takes as input (sk_a, D_a, pm, pk_b) and it outputs some auxiliary information au_a and an internal secret s_a. Then au_a is sent to the cloud through the authenticated user-to-cloud private communication channel while s_a is kept secret. Similarly, user B can generate au_b with (sk_b, D_b, pm, pk_a) as input. Then, B sends au_b to the cloud.*
- $\{(rslt_a, proof_a), (rslt_b, proof_b)\} \leftarrow$ *SetInt*(C_a, au_a, C_b, au_b)*: This algorithm runs by the cloud. It takes C_a, au_a, C_b, au_b as input and outputs $(rslt_a, proof_a)$ and $(rslt_b, proof_b)$. Then, the cloud sends $(rslt_a, proof_a)$ to A and $(rslt_b, proof_b)$ to B, where $proof_a$ and $proof_b$ are proofs to demonstrate faithful private set intersection computation by the cloud.*
- $\{D, \perp\} \leftarrow$ *Dec*($sk_a, rslt_a$)*: Each user A, B individually executes this algorithm to decrypt the intersection result. It takes ciphertext $rslt_a$ as input, which is the output of the delegated set intersection operation computed by the cloud on ciphertexts C_a and C_b, and sk_a and it outputs the intersection set $D = D_a \cap D_b$. If $D_a \cap D_b = \emptyset$, the algorithm outputs $\perp$. Similarly, B obtains $D_a \cap D_b$.*
- $\{0, 1\} \leftarrow$ *Verify*($sk_a, s_a, rslt_a, proof_a$)*: A runs this algorithm to verify whether $rslt_a$ is honestly generated. It takes $(sk_a, s_a, rslt_a, proof_a)$ and outputs 1 if the $rslt_a$ has been honestly generated. It outputs 0 if the cloud is cheating. Similarly, B verifies that if the outputs of the computation is correct.*

4.2 Adversarial model and Security Guarantees

We assume that users are honest-but-curious adversaries, meaning that they use their benign inputs for the protocol, but they are curious about other users' data. We assume that the cloud is malicious: It can arbitrarily deviate from the prescribed protocols in any way. The malicious cloud can manipulate the integrity of the outsourced data. We also assume that the cloud does not collude with any data owners. This is a reasonable assumption that is also explained in [1]. If the cloud is controlled by an adversary, the adversary also has control over all the communication channels. SEVDSI scheme leaks only the set intersection size of two datasets.

Function Output Secrecy. Function output secrecy assures that even if previous information about any data $v \in D_a \cup D_b$ is given to the malicious cloud, it will not be able to perform brute-force attacks to figure out if any ciphertext has encoding of v in $C_a \cup C_b$. In order to eliminate the brute-force attacks that are performed by the cloud, the resulting ciphertexts C_a, C_b should have a common secret value in order to allow the cloud to compare ciphertexts in the set intersection phase. Moreover, having this common secret value also prevents the adversary (cloud) to run the PSI protocol between different users without having their permission. In SEVDSI, this secret element is the Diffie-Hellman key $tk_{ab} = g^{\gamma_a \gamma_b}$ of A and B.

The function outputs secrecy game between a challenger $\mathcal{B}$ and an attacker $\mathcal{A}$ is the following:

Setup. Challenger $\mathcal{B}$ runs Setup algorithm, it outputs public/secret key pair pk_a, sk_a for user A and pk_b, sk_b for user B. Then, it gives pk_a, pk_b to attacker $\mathcal{A}$.
Query. $\mathcal{A}$ issues encryption queries. For query q, $\mathcal{A}$ outputs D'_a, D'_b. $\mathcal{B}$ runs $(C'_a, C'_b) \leftarrow$ Enc algorithm and gives (C'_a, C'_b) to $\mathcal{A}$.
SetInt. $\mathcal{A}$ issues set intersection queries. $\mathcal{B}$ runs $(au'_a, s'_a, au'_b, s'_b) \leftarrow$ AuGen algorithm and gives (au'_a, au'_b) to $\mathcal{A}$.
Challenge. $\mathcal{A}$ outputs (D_a, D_b, v) to be challenged upon, where v is any dataset element in $D_a \cup D_b$. $\mathcal{B}$ randomly chooses a bit $b \in \{0, 1\}$ and sends $C_{a,v}$ and au_a (assuming that v is in D_a) to $\mathcal{A}$, where if $b = 0$; Otherwise, it sends $C_{a,v'}, au_a$, where v' is a random element in G and $C_{a,v}$ is the encryption of v with keys pk_a and sk_a.
Guess. $\mathcal{A}$ outputs $b' \in \{0, 1\}$.

Definition 6 (Function Output Secrecy). *We say that SEVDSI assures Function Output Secrecy if the advantage of $\mathcal{A}$ in winning the aforementioned game is $Adv_{\mathcal{A}} = \Pr[b = b'] \leq 1/2 + \epsilon(\lambda)$, for a negligible function ϵ and security parameter λ.*

Verifiability. In order to assure that the cloud honestly computes the set intersection, the users ask the malicious cloud to generate a proof about the result of the computation. With the proof and the result users are able to check whether the malicious cloud has honestly executed the delegated set intersection operations or not. Verifiability exposes to the user malicious private set intersection computations performed by a malicious cloud server. We use the same security game for verifiability as in [32], defined between an adversary $\mathcal{A}$ and a challenger $\mathcal{B}$ as follows:

– **KeyGen:** Given public parameters pm, $\mathcal{B}$ runs KeyGen(pm) algorithm to obtain encryption/decryption keys (pk_a, sk_a) for user A, (pk_b, sk_b) for user B and gives pk_a, pk_b to $\mathcal{A}$.
– **Phase 1:** $\mathcal{A}$ issues the following queries:
  - **Enc:** Given the dataset D'_a, $\mathcal{B}$ runs $C'_a \leftarrow$ Enc$((sk_a, pk_a), D'_a)$ and returns C'_a to $\mathcal{A}$.
  - **Enc:** Given the dataset D'_b, $\mathcal{B}$ runs $C'_b \leftarrow$ Enc$((sk_b, pk_b), D'_b)$ and returns C'_b to $\mathcal{A}$.
  - **AuGen:** $\mathcal{B}$ runs $au'_a, s'_a \leftarrow$ AuGen(sk_a, D'_a, pm, pk_b) and returns au'_a to $\mathcal{A}$.
  - **AuGen:** $\mathcal{B}$ runs $au'_b, s'_b \leftarrow$ AuGen(sk_b, D'_b, pm, pk_a) and returns au'_b to $\mathcal{A}$.
  - **Verify:** $\mathcal{B}$ runs Verify$(sk_a, s'_a, rslt_a, proof_a)$ and returns the output to $\mathcal{A}$.
  - **Verify:** $\mathcal{B}$ runs Verify$(sk_b, s'_b, rslt_b, proof_b)$ and returns the output to $\mathcal{A}$.

Challenge: $\mathcal{A}$ selects D_a, D_b of its choice, and sends them to $\mathcal{B}$. $\mathcal{B}$ runs $C_a \leftarrow$ Enc$((sk_a, pk_a), D_a)$ and $C_b \leftarrow$ Enc$((sk_b, pk_b), D_b)$, $au_a, s_a \leftarrow$ AuGen(sk_a, D_a, pm, pk_b) and $au_b, s_b \leftarrow$ AuGen(sk_b, D_b, pm, pk_a), and returns C_a, au_a, C_b, au_b to $\mathcal{A}$.
Phase 2: $\mathcal{A}$ and $\mathcal{B}$ follow the similar steps as that are in Phase 1.
Guess: $\mathcal{A}$ outputs $(rslt_a, proof_a), (rslt_b, proof_b)$ to $\mathcal{B}$.

The adversary wins the game if

– Verify$(sk_a, s_a, rslt_a, proof_a) = 1$ is valid and
– Verify$(sk_b, s_b, rslt_b, proof_b) = 1$ is valid and
– Dec$(sk_a, rslt_a) \neq$ Dec$(sk_b, rslt_b) \vee$ Dec$(sk_a, rslt_a) \neq (D_a \cap D_b)$.

Definition 7. *A SEVDSI scheme satisfies the verifiability functionality if the following holds:*

$$\Pr\left[\begin{array}{c} (pm) \leftarrow Setup(\lambda), \\ (pk_a, sk_a) \leftarrow KeyGen(pm), \\ (pk_b, sk_b) \leftarrow KeyGen(pm) \\ (D_a, D_b) \leftarrow \mathcal{A}^{Enc, AuGen, SetInt, Verify}(pk_a, pk_b) \\ C_a \leftarrow Enc((sk_a, pk_a), D_a), C_b \leftarrow Enc((sk_b, pk_b), D_b), \\ au_a, s_a \leftarrow AuGen(sk_a, D_a, pm, pk_b), \\ au_b, s_b \leftarrow AuGen(sk_b, D_b, pm, pk_a) \\ \{(rstl_a, proof_a), (rslt_b, proof_b)\} \leftarrow \\ \mathcal{A}^{Enc, SetInt, Verify}(pk_a, au_a, D_a, C_a, pk_b, au_b, D_b, C_b) : \\ 1 \leftarrow Verify(sk_a, s_a, rslt_a, proof_a) \wedge \\ 1 \leftarrow Verify(sk_b, s_b, rslt_b, proof_b) \wedge \\ Dec(sk_a, rslt_a) \neq Dec(sk_b, rslt_b) \\ \vee Dec(sk_a, rslt_a) \neq (D_a \cap D_b) \end{array}\right]$$

$$\leq \epsilon(\lambda)$$

for a negligible function ϵ and security parameter λ.

5 SEVDSI Protocol

5.1 Solution Overview

The PSI protocol in [32] is vulnerable to brute force attacks: the malicious cloud is able to infer users' plain datasets from ciphertexts by checking all possible dataset values. Another issue in [32] is that the malicious cloud is allowed to do set intersection operation between any two users, without having any permission by them. The cloud can proceed as follows: (1) In time t_0, users A and B wish to run the set intersection protocol with the aid of the cloud by sending their corresponding auxiliary information to it. (2) In time t_1, users A and C run the set intersection protocol with the help of the cloud by sending their corresponding auxiliary information to it. (3) In time t_2, where $t_2 > t_1 \wedge t_2 > t_0$, the cloud is able to perform PSI operation on the encrypted datasets of user B and user C, without their permission. This contradicts their protocol specifications, whereby users are required to send auxiliary information to the cloud in order the latter conduct the set intersection protocol.

Before delving into SEVDSI protocol, we present the idea for resiliency to cloud-side brute-force attacks. Also, we demonstrate how to achieve authorized and verifiable PSI.

- In order to build a system resilient against the cloud side brute force attacks, the ciphertexts are blinded with an extra random value x_a by user A (x_b by user B). Later, x_a, x_b are converted to a common random tk_{ab} that is independently computed by user A and user B to allow the cloud perform the set intersection operation on the encrypted data. tk_{ab} prevents the malicious cloud to perform brute force attacks on the datasets in order to infer the users' plaintext values.
- To avoid permissionless PSI by the malicious cloud on behalf of two users without getting any permission by them, users agree on a common random tk_{ab} which is Diffie-Hellman key exchange value. This value is unique for each pair of users. According to the given example above, the cloud is not able to perform PSI protocol on behalf of user B and user C without getting any authorization (permission) from them. The only way the malicious cloud can perform PSI is to extract tk_{bc}.
- The auxiliary information that is sent to the cloud by user A consists of two parts. The first part is a function of x_a and tk_{ab}. The second part is the aggregated user A's dataset values (s_a) that is masked under user B's public key. User A also signs this masked value and appends it to the second part of the auxiliary information which is directly forwarded to user B later by the cloud.
- In order to allow user B to verify if the set intersection result that is computed by the cloud is correct, the cloud sends a $(proof_b, rslt_b)$ tuple to user B. $proof_b$ consists of two parts: The first part is a *witness* that is generated by the cloud. The second part is the second part of user A's auxiliary information.

- In our scheme user A (B) also uses $d_{a,0}$ ($d_{b,0}$) random data value as in [32] to randomize his accumulated dataset, s_a. That prevents a semi-honest user B (A) to infer useful information about A's (B's) dataset items.

5.2 Protocol Description

We now present SEVDSI in full details:

- **Setup(λ):** Given security parameter λ, the trusted party runs $(acPk, acSk) \leftarrow acKeyGen(\lambda)$ and outputs system parameters $pm = acPk, g, e(,), G, G', p, H_1, H_2$, where g is the generator of the group G of prime order p, $H_1 : G \rightarrow \mathbb{Z}_p$ and $H_2 : G' \rightarrow \mathbb{Z}_p$.
- **KeyGen(pm):** Users A and B take system public parameters, pm, and generate their secret and public keys. The secret and public keys for users A and B are:

$$(sk_a, pk_a) = ((x_a, \beta_a, \gamma_a, sigSK_a), (g^{\beta_a}, g^{\gamma_a}, sigPK_a)),$$
$$(sk_b, pk_b) = ((x_b, \beta_b, \gamma_b, sigSK_b), (g^{\beta_b}, g^{\gamma_b}, sigPK_b))$$

  , where $sigPK$ and $sigSK$ are secret and public keys of a signature: $(sigPK, sigSK) \leftarrow$ sigKeyGen(λ).
- **Enc($(sk_a, pk_a), D_a$):** User A on input the dataset $D_a = (d_{a,1}, \ldots, d_{a,n-1})$, his secret and public key pair (sk_a, pk_a)
  - picks a random $d_{a,0} \in G$ and two random values $r_{a,i1}, r_{a,i2}$ for each $i = \{0, \ldots, n-1\}$,
  - computes

$$cph_{a,i} = (g^{r_{a,i2}}, g^{r_{a,i1}\gamma_a}, d_{a,i}^{x_a} g^{(r_{a,i1}+r_{a,i2})x_a\beta_a})$$
$$= C_{a,i,1}, C_{a,i,2}, C_{a,i,3}$$

    and sets $C_a = \{cph_{a,0}, \ldots, cph_{a,n-1}\}$.

  Similarly, user B with $D_b = (d_{b,0}, d_{b,1}, \ldots, d_{b,n-1})$ where $d_{b,i} \in G$ for $0 \leq i \leq n-1$, can compute:

$$C_b = \{cph_{b,0}, \ldots, cph_{b,n-1}\}$$

  .
- **AuGen(sk_a, D_a, pm, pk_b):** Once two users (A and B) agree on set intersection operation that is delegated to the cloud, user A
  - generates rekey:

$$rk_a = (g^{H_1(tk_{ab})\beta_a/\gamma_a}, g^{H_1(tk_{ab})\beta_a}, g^{H_1(tk_{ab})/x_a})$$
$$= (RK_{a,1}, RK_{a,2}, RK_{a,3}),$$

    where $tk_{ab} = g^{\gamma_a\gamma_b}$.

- computes, for $0 \leq i \leq n-1$,

$$T_i = H_2\left(e\left(d_{a,i}^{H_1(tk_{ab})}, g\right)\right)$$

and $s_a = s_a \leftarrow acGen(acPk, \{T_0, \ldots, T_{n-1}\})$

- masks the secret information s_a using user B's public key pk_b to obtain $cph_b = \left(g^{r_2}, g^{\gamma_b r_1}, s_a g^{\beta_b (r_1+r_2)}\right)$, where $r_1, r_2 \leftarrow \mathbb{Z}_p$. Then, user A runs $\rho_a \leftarrow sigSign(sigSK_a, cph_b)$ to obtain a signature ρ_a on message cph_b. Finally, user A sets $au_a = (rk_a, cph_b, \rho_a)$ and sends it to the cloud. Similarly, user B can generate $au_b = (rk_b, cph_a, \rho_b)$ and forwards it to the cloud.

– **SetInt**(C_a, au_a, C_b, au_b)**:** The cloud:
  - transforms ciphertexts $cph_{a,i}$ for $0 \leq i \leq n-1$ into $T_{a,i}$ using au_a and computes T_a as follows: for $r_{ab} = r_{a,i1} + r_{a,i2}$,
    $T_{a,i}$ equals

$$\begin{aligned}
&= \frac{e\left(C_{a,i,3}, RK_{a,3}\right)}{e\left(C_{a,i,2}, RK_{a,1}\right) e\left(C_{a,i,1}, RK_{a,2}\right)} \\
&= \frac{e\left(d_{a,i}^{x_a} g^{x_a \beta_a (r_{ab})}, g^{H_1(tk_{ab})/x_a}\right)}{e\left(g^{\gamma_a r_{a,i1}}, g^{H_1(tk_{ab})\beta_a/\gamma_a}\right) e\left(g^{r_{a,i2}}, g^{H_1(tk_{ab})\beta_a}\right)} \\
&= \frac{e\left(d_{a,i}^{x_a}, g^{H_1(tk_{ab})/x_a}\right) e\left(g^{x_a \beta_a (r_{ab})} g^{H_1(tk_{ab})/x_a}\right)}{e\left(g, g\right)^{r_{a,i1} H_1(tk_{ab})\beta_a} e\left(g, g\right)^{r_{a,i2} H_1(tk_{ab})\beta_a}} \\
&= \frac{e\left(d_{a,i}, g\right)^{H_1(tk_{ab})} e\left(g, g\right)^{H_1(tk_{ab})\beta_a (r_{ab})}}{e\left(g, g\right)^{H_1(tk_{ab})\beta_a (r_{ab})}} \\
&= e\left(d_{a,i}, g\right)^{H_1(tk_{ab})}
\end{aligned}$$

    $T_a = \{H_2(T_{a,0}), ..., H_2(T_{a,n-1})\}$
    It follows the same steps to compute T_b.
  - generates the intersection sets $rslt_a, rslt_b$ and the proofs $proof_a, proof_b$ with respect to C_a and C_b as follows: $(acRslt, acWit_a) \leftarrow$ acProve$(acPk, T_a, T_b)$ and sets $rslt_a = \{cph_{a,i} \mid H_2\left(T_{a,i}\right) \in acRslt\}$, $proof_a = (acWit_a, cph_a, \rho_b)$. Accordingly it procceds to compute $(acRslt, acWit_b) \leftarrow$ acProve$(acPk, T_b, T_a)$, $rslt_b = \{cph_{b,i} \mid H_2\left(T_{b,i}\right) \in acRslt\}$ and $proof_b = (acWit_b, cph_b, \rho_a)$.

– **Dec**$(sk_a, rslt_a)$**:** Given the cloud-generated ciphertext intersection set $rslt_a = \{cph_{a,j}, ..., cph_{a,k}\}$ where $0 \leq j, k \leq n-1$, user A decrypts ciphertexts $cph_{a,i}$ for $j \leq i \leq k$ as follows:
$\frac{\left(C_{a,i,3}\right)^{x_a^{-1}}}{\left(C_{a,i,1}^{\beta_a}\right)\left(C_{a,i,2}^{\beta_a/\gamma_a}\right)} = \frac{\left(d_{a,i}^{x_a} g^{x_a \beta_a (r_{a,i1}+r_{a,i2})}\right)^{x_a^{-1}}}{\left(g^{r_{a,i2}\beta_a}\right)\left(g^{\gamma_a r_{a,i1}\beta_a/\gamma_a}\right)} = d_{a,i}.$
The decryption of $rslt_a$ is $D_a \cap D_b = \{d_{a,j}, ..., d_{a,k}\}$. Note that this algorithm can also be used to decrypt C_a without involving any delegated set operations. Similarly, user B can decrypt the cloud-generated ciphertext intersection set $rslt_b = \{cph_{b,j}, ..., cph_{b,k}\}$ where $j \leq i \leq k$ to obtain $D_a \cap D_b$.

– **Verify**$(sk_a, s_a, rslt_a, proof_a)$: Given $rslt_a$ and $proof_a$, user A verifies that the cloud faithfully executed the SetInt protocol as follows:
  - verifies the integrity of cph_a by running sigVerify$(sigPK_b, cph_a, \rho_b)$. If the signature verification fails the protocol halts, otherwise, proceed to the next step.
  - decrypts cph_a using private key sk_a according to $s_b = \frac{s_b g^{\beta_a(r_1+r_2)}}{(g^{r_2\beta_a})(g^{\gamma_a r_1 \beta_a/\gamma_a})}$.
  - if $rslt_a$ is not empty, decrypts $rslt_a$ to obtain the plaintexts and computes $Y_a = \{e\left(d_{a,i}^{H_1(tk_{ab})}, g\right) \mid cph_{a,i} \in rslt_a\}$. Otherwise, let $Y_a = \emptyset$.
  - runs acVerify$(acPk, s_a, Y_a, acWit_a, s_b)$. If the multi-accumulator fails to verify the protocol halts. Otherwise, the algorithm calls Dec$(sk_a, rslt_a)$ to obtain $D_a \cap D_b$. Similarly, user B runs the same algorithm to verify that the cloud faithfully computed the set intersection computation.

Remark 1. When users A and B wish to learn their intersection on new datasets, D_a^{new}, D_b^{new}, user A uses a new value y_a instead of using x_a to encrypt D_a^{new} (B uses y_b to encrypt D_b^{new}) and generates a new rk_a^{new} (rk_b^{new} for user B). To generate a new rk_a, the users agree on a new common secret key, tk_{ab}. Therefore, au_a^{new} (au_b^{new} for user B) is generated. The reason to use new y_a to encrypt D_a^{new} and choose new rk_a is to eliminate the compatibility of C_a^{new} and au_a^{old}. Otherwise, the cloud with $\left(C_a^{new}, au_a^{old}\right)$ and $\left(C_b^{new}, au_b^{old}\right)$ can compute the set intersection operation on users' new datasets in a permissionless manner.

5.3 Efficiency Analysis

In Table 3 we analyze the computation efficiency of SEVDSI. As for the multi-accumulator operations acGen and acProve commit to n exponentiations and acVerify results in k exponentiations and 7 pairings. We assume that $|D_a| = |D_b| = n$ and $|D_a \cap D_b| = k$. We also present in table 3 the overhead of VDSI scheme [32]. Overall, SEVDSI requires $3n + 2k + 4$ more exponentiations than VDSI [32] as a result of the extra x_a, tk, rk values embedded to each ciphertext, for datasets of size n, with k common elements.

Algorithm	VDSI [32]	SEVDSI
Enc	$3n$ E + n P + acGen	$4n$ E
Dec	$2n$ E	$3n$ E
AuGen	4 E + sigSign	$(n + 7)$ E + n P + sigSign + acGen
SetInt	$6n$ P + 2 acProve	$6n$ P + 2 acProve
Verify	$2(k + 1)$ E + k P + acVerify + sigVerify	$(4k{+}3)$ E + k P + acVerify + sigVerify

Table 3. Computational complexity comparison of the protocol in [32] and ours (SEVDSI), where E denotes the exponentiation operation, P denotes the pairing operation.

SEVDSI outperforms asymptotically in the communication overhead compared with 2PC PSI solutions. More specifically assuming m is the total number of users in the system, n is the total number of dataset items that each user has, k is the intersection size of two different datasets then:

1. Each user outsources n data items to the cloud. Therefore, m users outsource mn data items to the cloud in outsourcing phase.
2. During AuGen and SetInt phases user u_i sends $7(m-1)$ group elements to the cloud in order allow all the other users u_j, $1 \leq i \neq j \leq m$ to perform PSI with all the other users, by running AuGen algorithm. The cloud sends $(k+6)(m-1)$ elements to each user u_i (k is the intersection size and 6 is the proof size for each PSI), for $i = 1, \ldots, m$, as the intersection results by using SetInt algorithm. The total communication complexity between the users and the cloud is $(k+13)(m-1)m = O(km^2)$.
3. Thus from 1) and 2) we conclude that the overall communication complexity for SEVDSI is $O(km^2 + mn)$.

5.4 Security

In this section we analyze the security of SEVDSI scheme that is formally defined in Section 4. SEVDSI protocol adheres to two security guarantees: function output secrecy and verifiability. We model H_1 as a random function and H_2 as a collision resistant hash function.

Theorem 2. *SEVDSI scheme assures function output secrecy if the VDDH assumption holds.*

The reductionist proof is based on the VDDH assumption. We establish that if there exists an adversary $\mathcal{A}$ winning the function output secrecy game with non-negligible probability ϵ, then there is an adversary $\mathcal{B}$ breaking the VDDH assumption in G with non-negligible probability.

Proof. The VDDH oracle chooses a uniformly random $t \xleftarrow{\$} \{0,1\}$ and passes $T = \left(g, v, g^{\gamma_a}, g^{\gamma_b}, g^{\beta_a}, g^{\beta_b}, g^{r\beta_a}, g^{r\beta_b}, g^{r\frac{\beta_a}{\gamma_a}}, g^{r\frac{\beta_b}{\gamma_b}}, Z\right)$ to $\mathcal{B}$, where $Z = v^r$ if $t = 0$ and Z some random element in G if $t = 1$. $\mathcal{B}$ simulates $\mathcal{A}$'s queries during the game as follows:
Setup. $\mathcal{B}$ obtains $sigPK_a$ and $sigSK_a$ using $(sigPK_a, sigSK_a) \leftarrow$ sigKeyGen(λ); $sigPK_b$ and $sigSK_b$ are generated similarly. $\mathcal{B}$ gives $pk_a = \left(g^{\gamma_a}, g^{\beta_a}, sigPK_a\right)$ and $pk_b = \left(g^{\gamma_b}, g^{\beta_b}, sigPK_b\right)$ to attacker $\mathcal{A}$.
Query. $\mathcal{A}$ issues encryption queries. For query q, $\mathcal{A}$ outputs $D'_a = \{d'_{a,0}, \ldots, d'_{a,n-1}\}, D'_b = \{d'_{b,0}, \ldots, d'_{b,n-1}\}$. $\mathcal{B}$ runs $(C'_a, C'_b) \leftarrow$ Enc algorithm as follows:

- $\mathcal{B}$ finds the corresponding data items from T such that $d'_{a,i} = v_{a,i}$ and $d'_{b,j} = v_{b,j}$,
- picks random values $r', r_{a,i,1}, r_{a,i,2}, x_a, x_b \in \mathbb{Z}_p$, where $i = 0, \ldots, n-1$,
- computes $cph_{a,i} = g^{r_{a,i,2}}, g^{r_{a,i,1}\gamma_a}, v_{a,i}^{rx_a} g^{(r_{a,i,1}+r_{a,i,2})x_a r\beta_a} = (C_{a,i,1}, C_{a,i,2}, C_{a,i,3})$

– computes $(C_{b,i,1}, C_{b,i,2}, C_{b,i,3})$ by the same way above and and gives (C'_a, C'_b) to $\mathcal{A}$.

SetInt. $\mathcal{A}$ issues set intersection queries for D'_a, D'_b. $\mathcal{B}$ runs $(au'_a, s'_a, au'_b, s'_b) \leftarrow$ AuGen algorithm as follows:

– generates rekey, $rk'_a = \left(g^{rr'\beta_a/\gamma_a}, g^{rr'\beta_a}, g^{r'/x_a}\right) = (RK_{a,1}, RK_{a,2}, RK_{a,3})$.
– computes, for $0 \leq i \leq n-1$, $T_i = H_2\left(e\left(v_{a,i}^{rr'}, g\right)\right)$ and $s'_a \leftarrow acGen(acPk, \{T_0, \ldots, T_{n-1}\})$.
– encrypts the secret information s'_a using user B's public key pk_b to obtain ciphertext $cph'_b = \left(g^{r_5}, g^{\gamma_b r_6}, s'_a g^{\beta_b(r_5+r_6)}\right)$, where $r_5, r_6 \leftarrow \mathbb{Z}_p$. Then, user A runs $\rho'_a \leftarrow sigSign(sigSK_a, cph'_b)$ to obtain a signature ρ'_a on message cph'_b. Finally, user A sets $au'_a = (rk'_a, cph'_b, \rho'_a)$. Similarly, user B can generate $au'_b = (rk'_b, cph'_a, \rho'_b)$. Then, it gives (au'_a, au'_b) to $\mathcal{A}$.

Challenge. $\mathcal{A}$ outputs $D_a = \{d_{a,0}, \ldots, d_{a,n-1}\}, D_b = \{d_{b,0}, \ldots, d_{b,n-1}\}, v$ to be challenged upon. $\mathcal{B}$ randomly chooses $y_a, r'' \leftarrow \mathbb{Z}_p$. Then, if $v \in D_a$ and $v = v_{a,n-1}$, $\mathcal{B}$

– randomly chooses a bit $b \in \{0,1\}$ and sets $Z = v_{a,n-1}^r$ if $b = 0$. Then, it finds the corresponding data items for T such that $d_{a,i} = v_{a,i}$,
– uniformly at random selects two values $r_{a,1}, r_{a,2}$, and sends $C_v = g^{r_{a,1}}, g^{r_{a,2}\gamma_a}, \left(Z^{y_a} = v_{a,n-1}^{ry_a}\right)\left(g^{ry_a\beta_a(r_{a,1}+r_{a,2})}\right)$ to $\mathcal{A}$.
– generates rekey, $rk_a = \left(g^{r''r\beta_a/\gamma_a}, g^{r''r\beta_a}, g^{r''/y_a}\right) = (RK_{a,1}, RK_{a,2}, RK_{a,3})$.
– computes, $T_i = H_2\left(e\left(v_{a,i}^{rr''}, g\right)\right)$ and $s_a \leftarrow acGen(acPk, \{T_0, \ldots, T_{n-1}\})$ for $0 \leq i \leq n-1$.
– encrypts the secret information s_a using other user's public key pk_b to obtain ciphertext $cph_b = \left(g^{r_7}, g^{\gamma_b r_8}, s_a g^{\beta_b(r_7+r_8)}\right)$, where $r_7, r_8 \leftarrow \mathbb{Z}_p$. Then, $\mathcal{B}$ runs $\rho_a \leftarrow sigSign(sigSK_a, cph_b)$ to obtain a signature ρ_a on message cph_b. Finally, $\mathcal{B}$ sets $au_a = (rk_a, cph_b, \rho_a)$.

Guess. $\mathcal{A}$ outputs $b \in \{0,1\}$ and sends it to $\mathcal{B}$. If $t = b$ then $\mathcal{A}$ wins the game. As the game is perfectly simulated by $\mathcal{B}$, then if $\mathcal{A}$ chooses the correct b then $\mathcal{B}$ also chooses as $t = b$ and breaks the VDDH assumption, with non-negligible probability $\geq \frac{1}{2} + \epsilon'(\lambda)$.

□

Theorem 3. *If Sig is an unforgeable signature scheme, multi-accumulator scheme is secure under q-SDH assumption, H_1 is a random function and H_2 is a collision resistance hash function, SEVDSI scheme assures the verifiability property.*

Proof. We show that if there exists a probabilistic polynomial-time adversary $\mathcal{A}$ who breaks the verifiability of the SEVDSI scheme with a non-negligible probability ϵ, then we show how a probabilistic polynomial time adversary $\mathcal{B}$ can

break the assumption that Sig is an unforgeable signature scheme or Ac is a secure multi-accumulator. In the proof we assume H_1 is a random function and H_2 is a collision resistant hash function. $\mathcal{B}$ proceeds as follows.
Setup: $\mathcal{B}$ runs $pm \leftarrow \text{Setup}(\lambda)$ and makes pm publicly known. It then runs KeyGen(pm) to obtain $sk_a = (sigSK_a, \beta_a, \gamma_a, x_a)$ and $pk_a = (sigPK_a, g^{\beta_a}, g^{\gamma_a})$, runs KeyGen$(pm)$ to obtain $sk_b = (sigSK_b, \beta_b, \gamma_b, x_b)$, $pk_b = (sigPK_b, g^{\beta_b}, g^{\gamma_b})$, and returns pk_a, pk_b to $\mathcal{A}$.
Phase 1: $\mathcal{A}$ can make the following queries polynomially many times.

- **Enc:** Given the dataset D'_a, $\mathcal{B}$ runs $C'_a \leftarrow \text{Enc}((sk_a, pk_a), D'_a)$ and returns C'_a to $\mathcal{A}$.
- **Enc:** Given the dataset D'_b, $\mathcal{B}$ runs $C'_b \leftarrow \text{Enc}((sk_b, pk_b), D'_b)$ and returns C'_b to $\mathcal{A}$.
- **AuGen:** $\mathcal{B}$ runs $au'_a, s'_a \leftarrow \text{AuGen}(sk_a, D'_a, pm, pk_b)$ and returns au_a to $\mathcal{A}$.
- **AuGen:** $\mathcal{B}$ runs $au'_b, s'_b \leftarrow \text{AuGen}(sk_b, D'_b, pm, pk_a)$ and returns au_b to $\mathcal{A}$.
- **Verify:** $\mathcal{B}$ runs Verify$(sk_a, s'_a, rslt_a, proof_a)$ and returns the output to $\mathcal{A}$.
- **Verify:** $\mathcal{B}$ runs Verify$(sk_b, s'_b, rslt_b, proof_b)$ and returns the output to $\mathcal{A}$

Challenge: $\mathcal{A}$ selects D_a, D_b of its choice, and sends them to $\mathcal{B}$. $\mathcal{B}$ runs $C_a \leftarrow \text{Enc}((sk_a, pk_a), D_a)$ and $C_b \leftarrow \text{Enc}((sk_b, pk_b), D_b)$, $au_a, s_a \leftarrow$ AuGen(sk_a, D_a, pm, pk_b) and $au_b, s_b \leftarrow \text{AuGen}(sk_b, D_b, pm, pk_a)$, and returns C_a, au_a, C_b, au_b to $\mathcal{A}$.
Phase 2: $\mathcal{A}$ issues queries in the same way as in Phase 1.
Guess: $\mathcal{A}$ outputs $(rslt_a, proof_a), (rslt_b, proof_b)$ to $\mathcal{B}$. This completes the simulation. First let us consider the verification for $(rslt_a, proof_a)$. Note that cph_a specified by $proof_a$ cannot be manipulated, otherwise it breaks the unforgeability of Sig. $\mathcal{B}$ decrypts cph_a to obtain s_b. In addition, $\mathcal{B}$ decrypts Dec$(sk_a, rslt_a)$, and obtains
$T_a = \{H_2(e(d'_{a,i}, g)^{H_1(tk_{ab})}) \mid cph_{a,i} \in rslt_a\}$ where $d'_{a,i}$ is the plaintext with respect to $cph_{a,i}$ and $H_1(tk_{ab})$ is a random value.

$$T = \{H_2(e(d_{a,i}, g)^{H_1(tk_{ab})}) \mid (d_{a,i} \in D_a) \wedge (tk_{ab} = g^{\gamma_a \gamma_b})\} \cap$$

$$\{H_2(e(d_{b,i}, g)^{H_1(tk_{ab})}) \mid (d_{b,i} \in D_b) \wedge (tk_{ab} = g^{\gamma_a \gamma_b})\}$$

. If $\mathcal{A}$ breaks the verifiability with $(rslt_a, proof_a)$, then at least one of the following cases should hold:
Case 1:

$1 \leftarrow \text{acVerify}(acPk, s_a, T_a, acWit_a, s_b)$
$1 \leftarrow \text{acVerify}(acPk, s_a, T, acWit_a, s_b)$
$T = T_a$
$\exists d'_{a,i} \neq d_{a,i}, H_2(e(d'_{a,i}, g)^{H_1(tk_{ab})}) = H_2(e(d_{a,i}, g)^{H_1(tk_{ab})}),$

Case 2:

$1 \leftarrow \text{acVerify}(acPk, s_a, T_a, acWit_a, s_b)$
$1 \leftarrow \text{acVerify}(acPk, s_a, T, acWit_a, s_b)$
$T \neq T_a$

If $\mathcal{A}$ breaks the verifiability regarding $(rslt_a, proof_a)$ with respect to case 1, then it breaks the assumption that H_2 is collision resistant: $(d'_{a,i} \neq d_{a,i})$

leads to $e(d'_{a,i}, g)^{H_1(tk_{ab})} \neq e(d_{a,i}, g)^{H_1(tk_{ab})}$ while $H_2(e(d'_{a,i}, g)^{H_1(tk_{ab})}) = H_2(e(d_{a,i}, g)^{H_1(tk_{ab})})$.

If $\mathcal{A}$ breaks the verifiability regarding $(rslt_a, proof_a)$ with respect to case 2, then it breaks the security of the multi-accumulator scheme by presenting $acRslt = T_a$, which is different from T.

Therefore, we proved that $\mathcal{A}$ breaks the verifiability of SEVDSI scheme with respect to $(rslt_a, proof_a)$ or $(rslt_b, proof_b)$ with negligible probability under the assumptions that Sig is unforgeable, H_1 is a random function, H_2 is a collision resistant hash function and Ac is a secure multi-accumulator scheme.

□

References

1. A. Abadi, S. Terzis, and C. Dong. O-PSI: delegated private set intersection on outsourced datasets. In *30th IFIP TC 11 International Conference*, 2015.
2. A. Abadi, S. Terzis, and C. Dong. VD-PSI: Verifiable delegated private set intersection on outsourced private datasets. In *FC*, 2016.
3. C. A. Ardagna, R. Asal, E. Damiani, and Q. H. Vu. From security to assurance in the cloud: A survey. *ACM Comput. Surv.*, 2015.
4. G. Ateniese, E. De Cristofaro, and G. Tsudik. (if) size matters: Size-hiding private set intersection. In *PKC*, 2011.
5. Y. Aumann and Y. Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In *TCC*, 2007.
6. T. Bradley, S. Faber, and G. Tsudik. Bounded size-hiding private set intersection. Technical report, https://eprint.iacr.org/2016/657, 2016.
7. E. D. Cristofaro, J. Kim, and G. Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In *ASIACRYPT*, 2010.
8. D. Dachman-Soled, T. Malkin, M. Raykova, and M. Yung. Efficient robust private set intersection. In *ACNS*, 2009.
9. E. De Cristofaro and G. Tsudik. Practical private set intersection protocols with linear computational and bandwidth complexity. In *FC*, 2010.
10. C. Dong, L. Chen, and Z. Wen. When private set intersection meets big data: an efficient and scalable protocol. In *CCS*, 2013.
11. M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *EUROCRYPT*, 2004.
12. S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 1988.
13. C. Hazay and Y. Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *TCC*, 2008.
14. C. Hazay and K. Nissim. Efficient set operations in the presence of malicious adversaries. In *PKC*, 2010.
15. W. Huang, A. Ganjali, B. H. Kim, S. Oh, and D. Lie. The state of public infrastructure-as-a-service cloud security. *ACM Comput. Surv.*, 2015.
16. Y. Huang, D. Evans, and J. Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS*, 2012.
17. S. Jarecki and X. Liu. Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection. In *TCC*, 2009.
18. S. Jarecki and X. Liu. Fast secure computation of set intersection. In *SCN*, 2010.

19. S. Kamara, P. Mohassel, M. Raykova, and S. S. Sadeghian. Scaling private set intersection to billion-element sets. In *FC*, 2014.
20. F. Kerschbaum. Collusion-resistant outsourcing of private set intersection. In *SAC*, 2012.
21. F. Kerschbaum. Outsourced private set intersection using homomorphic encryption. In *ASIACCS*, 2012.
22. A. Kiss, J. Liu, T. Schneider, N. Asokan, and B. Pinkas. Private set intersection for unequal set sizes with mobile applications. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 2017(4):97–117, 2017. To appear. Code: `http://encrypto.de/code/MobilePSI`.
23. L. Kissner and D. Song. Privacy-preserving set operations. In *CRYPTO*, 2005.
24. V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu. Efficient batched oblivious prf with applications to private set intersection. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 818–829, New York, NY, USA, 2016. ACM.
25. F. Liu, W. K. Ng, W. Zhang, D. H. Giang, and S. Han. Encrypted set intersection protocol for outsourced datasets. In *IC2E*, 2014.
26. M. Orr, E. Orsini, and P. Scholl. Actively secure 1-out-of-n ot extension with application to private set intersection. Cryptology ePrint Archive, Report 2016/933, 2016. `http://eprint.iacr.org/2016/933`.
27. B. Pinkas, T. Schneider, G. Segev, and M. Zohner. Phasing: Private set intersection using permutation-based hashing. In *USENIX Security 15*, 2015.
28. B. Pinkas, T. Schneider, and M. Zohner. Faster private set intersection based on OT extension. In *USENIX*, 2014.
29. P. Rindal and M. Rosulek. Improved private set intersection against malicious adversaries. In *EUROCRYPT*, pages 235–259. Springer, Cham, 2017.
30. V. Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT*, 1997.
31. Y. Zhao and S. S. M. Chow. Are you the one to share? secret transfer with access structure. *PoPETs*, 2017(1):149–169, 2017.
32. Q. Zheng and S. Xu. Verifiable delegated set intersection operations on outsourced encrypted data. In *IC2E*, 2015.