Delegated Multi-party Private Set Intersection from Secret Sharing

Jingwei Hu[1], Zhiqi Liu[2], Cong Zuo[2], San Ling[1] and Huaxiong Wang[1]

[1] Nanyang Technological University, Singapore, {davidhu,lingsan,hxwang}@ntu.edu.sg
[2] Beijing Institute of Technology, China, {liuzhiqi25,zuocong10}@gmail.com

Abstract. In this work, we address the problem of Delegated PSI (D-PSI), where a cloud server is introduced to handle most computational and communication tasks. D-PSI enables users to securely delegate their private sets to the cloud, ensuring the privacy of their data while allowing efficient computation of the intersection. The cloud operates under strict security requirements, learning nothing about the individual sets or the intersection result. Moreover, D-PSI minimizes user-to-user communication and supports "silent" processing, where the cloud can perform computations independently without user interaction, apart from set delegation and result retrieval.

We formally define the D-PSI problem and propose a novel construction that extends beyond two-party scenarios to support multi-party settings. Our construction adheres to the D-PSI requirements, including security against semi-honest adversaries, and achieves computational and communication complexities close to the ideal "perfect" D-PSI protocol. Additionally, we demonstrate the practicality of our approach through a baseline implementation and an optimized version that further reduces computational overhead. Our results establish a strong foundation for secure and efficient PSI in real-world cloud computing scenarios.

Keywords: Multi-party PSI · Secret Sharing · Randomized Intersection Encoding

1 Introduction

Private Set Intersection (PSI) is arguably one of the most extensively studied secure multiparty computation (MPC) protocols. PSI is essentially a specialized form of MPC, where multiple parties, each holding a private set, collaboratively compute the intersection of their sets. During the computation, no information about the individual sets except for the intersection is leaked to either party.

The earliest PSI protocols were based on Diffie-Hellman key exchange [Mea86, HFH99]. In recent years, the research focus has shifted toward using Oblivious Key-Value Stores (OKVS) and Vector Oblivious Linear Evaluation (VOLE) as core constructions to achieve high-performance PSI protocols. OKVS-based constructions encode the input sets in a specialized manner, as described in OKVS [GPR+21], resulting in transparent data structures known as oblivious data structures. VOLE-based method [RS21], on the other hand, utilizes VOLE to construct Oblivious Pseudorandom Functions (OPRF), which are then employed to develop OPRF-PSI protocols.

The research line mentioned above performs set intersection securely under the assumption that no cloud server is involved, and all computing parties hold their sets locally. However, in the real world, the cloud computing paradigm is ubiquitous, where a cloud server is significantly more powerful in terms of both computational and communication resources compared to individual users in the PSI protocol.

Therefore, it is desirable for all computing parties (users) in PSI to securely delegate their sets to the cloud, allowing the cloud to handle most of the computational tasks required to compute the intersection. A critical feature of this setup is that the computing parties may not trust the cloud. Thus, the data delegated by the users must remain oblivious to the cloud, and any operations performed on the cloud should not violate the privacy of the sets. Additionally, it is preferable that, at the end of the protocol, the cloud does not learn the intersection result. Instead, only the authorized user(s) should be allowed to learn the intersection.

Definition of D-PSI To capture the intuitions above, we formally define the problem of Delegated Private Set Intersection (Delegated PSI, D-PSI) as follows. Unlike the standard multi-party private set intersection problem, D-PSI introduces an additional computing party, a cloud server. Three types of computing parties are defined:

- A cloud server C,
- Delegated users $A_1, A_2, \ldots, A_n$,
- A querying user B.

In this setup, each delegated user A_i securely delegates their set to the cloud server C, and the querying user B also securely delegates their set to the cloud server C, and ultimately learns the intersection result.

The goal of D-PSI is to collaboratively compute the intersection without revealing the individual sets:

- Each delegated user A_i securely (indistinguishable from random values) stores their set on the cloud C.
- The querying user B securely stores their set on the cloud C.
- The cloud C, together with $\{A_i\}$ and B, collaboratively computes the intersection such that B can efficiently obtain the result $(\cap A_i) \cap B$.

The requirements are:

- Delegated users $\{A_i\}$ only know their private sets.
- The cloud C learns nothing.
- The querying user B only knows their private set B and the intersection $(\cap A_i) \cap B$.
- The cloud C does not collude with the users and a user does not collude with another user
- The cloud C should handle the majority of the computational and communication tasks.
- Users $\{A_i\}$ and B should not interact directly or should keep such interaction minimal.
- Silent processing mode is desirable: the cloud can perform the computational tasks independently, without being interrupted by users. This means that users and the cloud can remain offline during the computation and only come online when the user delegates their set or requests the intersection result.

An important feature of D-PSI is its flexibility. If B learns the final intersection result, other computing parties $\{A_i\}$ can also be informed of the result through an encrypted channel. This allows full control over who can access the intersection result, making D-PSI much more flexible and desirable than the standard multi-party private set intersection, which requires all parties to know the intersection.

A natural question that arises here is: what are the best achievable computation and communication complexities for a D-PSI protocol? We analyze the performance of an ideal "perfect" D-PSI protocol under the assumption that the cloud C is fully trustworthy. This means that all users can directly delegate their sets to the cloud, and the cloud C neither leaks any intermediate results observed during the PSI computation nor the final intersection result. The protocol proceeds as follows:

1. The delegating users $\{A_i\}$ upload their respective sets to the cloud C through an encrypted channel.

2. The querying user B also uploads their set to C through an encrypted channel.

3. The cloud C computes the intersection $(\cap A_i) \cap B$ locally and privately.

4. The cloud C returns the intersection $(\cap A_i) \cap B$ to the querying user B through an encrypted channel.

Assume there are N delegating users, each with a set size of k. If no advanced data compression techniques are applied to reduce the size of each user's set, the communication complexity is clearly:

$$\Omega(Nk).$$

The computational complexity requires a more detailed analysis. A hash table can be used to represent a set, where the hash table has $h = \Omega(k)$ rows, and each row contains at most $\Omega(\log k)$ set elements. The total number of elements in the hash table is k. To compute the intersection, for example $A_1 \cap A_2$, it suffices to compare the set elements in the hash tables HT_{A_1} and HT_{A_2} row by row according to their row correspondence. This results in an intersection hash table $\mathrm{HT}_{A_1 \cap A_2}$, which also has h rows.

Using this technique, the computational complexity of the D-PSI protocol is:

$$\Omega(Nk).$$

Our Contributions We summarize the contributions of this paper as follows:

1. We propose a new construction for delegated PSI that extends beyond two-party scenarios to support multi-party settings. Our construction adheres to the definition of delegated PSI, emphasizing the "silent" processing feature and minimal user-to-user communication.

2. We rigorously prove the security of the new construction against semi-honest adversaries in the standard security model, without relying on any random oracle assumption.

3. Our baseline implementation achieves $\Omega(Nk^2)$ computational complexity and optimal $\Omega(Nk)$ communication complexity. Furthermore, our computation-optimized implementation achieves $\Omega(Nk(\log k)^2)$ computational complexity and $\Omega((N+\log k)k \log k)$ communication complexity. The optimized implementation remains sub-optimal in terms of both computational and communication overhead compared to the ideal "perfect" delegated PSI.

2 Related works

The research line of priviate set intersection is extensive and thus we limit our review to the most relevant works related to delegated multi-party private set intersection. Delegated PSI protocols leverage cloud computing for both computation and storage, and they can be categorized into two types: protocols supporting one-off delegation and those enabling repeated delegation for PSI computation.

The one-off delegation protocols, such as those in [Ker12, KMRS14], require clients to locally re-encode their data for each intersection computation and do not allow the reuse of outsourced encrypted data. However, this protocol is designed and implemented solely for a two-party setting.

In contrast, repeated PSI delegation protocols allow clients to outsource their encrypted data to the cloud once and, with the data owner's consent, perform multiple PSI computations without requiring clients to retain a local copy of their data or re-encode it for each computation. Among repeated PSI delegation protocols, [LNZ+14] employs hash functions and symmetric key encryption, while [QLS+15] utilizes public key encryption and hash functions. The protocol in [ZX15] incorporates verification capabilities using cryptographic accumulators and proxy re-encryption. However, these protocols ([LNZ+14, QLS+15, ZX15]) are not fully secure and are vulnerable to various attacks, as highlighted in [ATD15, ATMD17]. In contrast, the protocols in [ATD15, ATD17, ATMD17] provide robust security. Protocols in [ATD15, ATD17] use public key encryption and represent the entire dataset as a blinded polynomial that is outsourced to the cloud. Among them, the protocol in [ATMD17] is more efficient, leveraging a hash table to enhance PSI computation performance.

It is important to note that all these secure protocols ([ATD15, ATD17, ATMD17]) are designed and implemented for two-party settings and only support static datasets. A more recent study [ATD20] proposes a new solution to overcome this barrier to support multi-party and dataset updates.

Now let us focus on the recent study [ATD20]. This study explores advanced data structures such as hash tables, bloom filters, and point-evaluation formats to improve the concrete efficiency of the protocol. However, these enhancements make it challenging to provide an overview of their algorithms and, for our purposes, to identify the security flaws in their protocol design.

To address this challenge, we rewrite their protocol in a more concise form, omitting the advanced data structures and the complex set update operations. The core intuition of their algorithm involves setting up a cloud entity C that accepts one secret share of the delegating user's (denoted as A) private set. Due to the security of the secret sharing scheme, the cloud C learns nothing about the delegating user's private set. The crux of computing the intersection between the delegating user A and the querying user B lies in calculating a special encoding, which we refer to as "Randomized Intersection Encoding (RIE)," within the secret sharing format.

2.1 Randomized Intersection Encoding (RIE)

Before delving into the description of the PSI protocol proposed in [ATD20], we provide a brief introduction to the RIE technique. This serves as a foundation for explaining the insecurity we identified in [ATD20], while a formal discussion of RIE is deferred to the Preliminaries section.

A set $A = \{a_i\}_i$ can be equivalently represented as a polynomial:

$$f^{(A)}(x) = \prod_i (x - a_i),$$

which means the elements of the set a_i are encoded as the roots of the polynomial $f^{(A)}(x)$.

Moreover, the intersection of two sets A and B can be expressed as the sum of their corresponding polynomials:

$$f^{(A\cap B)}(x) \stackrel{def}{=} f^{(A)}(x) + f^{(B)}(x) = \prod_i (x - a_i) + \prod_i (x - b_i).$$

Clearly, the roots of $f^{(A\cap B)}(x)$ represent the intersection $A\cap B$. Either party can determine the intersection $A \cap B$ through a zero-point test.

However, in private set intersection problems, the polynomial $f^{(A\cap B)}(x)$ not only reveals the intersection but can also be used to infer $f^{(A)}(x)$ or $f^{(B)}(x)$, compromising privacy. Therefore, it is essential to construct a polynomial that satisfies two requirements:

- Its roots represent the intersection $A \cap B$.
- It is a randomized polynomial that does not provide additional information to reverse-engineer $f^{(A)}(x)$ or $f^{(B)}(x)$.

A novel method was proposed in [KS05] for the first time to fulfil the two requirements mentioned above, referred to as "Random Intersection Encoding (RIE)":

$$\mathrm{RIE}_{\omega^{(A)}(x),\,\omega^{(B)}(x)}(f^{(A)}(x), f^{(B)}(x)) \stackrel{def}{=} \omega^{(A)}(x) \cdot f^{(A)}(x) + \omega^{(B)}(x) \cdot f^{(B)}(x).$$

Privacy Guarantee The RIE encoding process ensures that $f^{(A)}(x)$ and $f^{(B)}(x)$ are not disclosed given that the random polynomials $\omega^{(A)}(x), \omega^{(B)}(x)$ are kept secret. Notably, the RIE encoding output is a "randomized" polynomial that only contains information about the intersection.

Linearity of RIE Encoding RIE encoding is linear. Specifically, for any polynomials $f^{(A)}(x), g^{(A)}(x)$ and $f^{(B)}(x), g^{(B)}(x)$, we have:

$$\begin{aligned}&\mathrm{RIE}_{\omega^{(A)}(x),\,\omega^{(B)}(x)}(f^{(A)}(x), f^{(B)}(x)) + \mathrm{RIE}_{\omega^{(A)}(x),\,\omega^{(B)}(x)}(g^{(A)}(x), g^{(B)}(x))\\ &= \mathrm{RIE}_{\omega^{(A)}(x),\,\omega^{(B)}(x)}(f^{(A)}(x) + g^{(B)}(x), f^{(A)}(x) + g^{(B)}(x)).\end{aligned}$$

This property ensures that the encoding process preserves the algebraic structure of the input polynomials.

2.2 Case study

Now, let us return to the framework proposed in [ATD20]. For simplicity of notation, we represent a polynomial $p(x)$ directly as p, omitting "(x)". Assume that the we have only one delegated user A, and one querying user B. The delegated user A secret-shares their set polynomial as $([p^{(A)}]_0, [p^{(A)}]_1)$, such that $p^{(A)} = [p^{(A)}]_0 + [p^{(A)}]_1$, and the querying user B also secret-shares their set polynomial as $([p^{(B)}]_0, [p^{(B)}]_1)$. User A delegates one share $[p^{(A)}]_0$ to the cloud while keeping $[p^{(A)}]_1$ secret. Similarly, user B delegates $[p^{(B)}]_0$ to the cloud while retaining $[p^{(B)}]_1$ privately.

The PSI protocol proceeds as follows:

1. User B splits their private share $[p^{(B)}]_1$ into two parts (o, z), such that $[p^{(B)}]_1 = o + z$.
2. User B sends o to the cloud and z to user A.
3. User A generates two random polynomials $\omega^{(A)}$ and $\omega^{(B)}$, and computes an RIE encoding:
$$q \leftarrow \mathrm{RIE}_{\omega^{(A)},\,\omega^{(B)}}([p^{(A)}]_1, z).$$

4. User A secret-shares q as $[q]_0, [q]_1$.
5. User A sends $[q]_0$, $\omega^{(A)}$ and $\omega^{(B)}$ to the cloud.
6. The cloud C computes another RIE encoding and masks it with $[q]_0$:

$$t \leftarrow \mathrm{RIE}_{\omega^{(A)},\omega^{(B)}}([p^{(A)}]_0, [p^{(B)}]_0 + o) + [q]_0.$$

7. User A and the cloud C send $[q]_1$ and t, respectively, to user B.
8. For each $b_i \in B$:
   (a) User B adds b_i to the intersection $A \cap B$ if b_i is a root of the polynomial $[q]_1 + t$.

Correctness Proof We prove here that at the end of the protocol, the querying user B indeed learns the intersection.

Proof. In the final step (Step 7), the querying user B recovers the polynomial:

$$[q]_1 + t = \mathrm{RIE}([p^{(A)}]_0 + [p^{(A)}]_1, [p^{(B)}]_0 + o + z) = \mathrm{RIE}(p^{(A)}, p^{(B)}),$$

due to the linearity of RIE. Therefore, the roots of this polynomial correspond to the intersection $A \cap B$, proving the correctness of the protocol. □

Security Analysis We identify several security weaknesses in the protocol.

First, the distribution of the shares in Step 2 is insecure. Anyone eavesdropping on the public channel can intercept both o and z, thereby recovering the sensitive $[p^{(A)}]$, which should remain private to user B only.

Second, in Step 5, the sensitive random polynomials $\omega^{(A)}$ and $\omega^{(B)}$ are transmitted over the public channel. User B can intercept them, violating the security requirement that $\omega^{(A)}$ and $\omega^{(B)}$ must remain secret. If user B obtains these random polynomials, the RIE encoding $\mathrm{RIE}(p^{(A)}, p^{(B)})$ could reveal more information than the set intersection, thereby breaking the security assumption.

Third, transmitting the share $[q]_0$ to the cloud in Step 5 and subsequently transmitting the other share $[q]_1$ to B in Step 7 is jointly insecure. User B can combine $[q]_0$ and $[q]_1$ to derive q. Moreover, B also receives another RIE encoding t in Step 7, which uses the same randomness $\omega^{(A)}$ and $\omega^{(B)}$. Since the derived RIE encoding q also uses the identical randomness, this violates the security requirements of RIE.

Fortunately, these weaknesses can be mitigated by transmitting these values through encrypted channels. The computing parties can initiate a key encapsulation mechanism or a key exchange protocol to securely establish a shared symmetric key. Subsequently, this key can be used to encrypt the transmitted values, ensuring that only encrypted values are sent through the public and insecure channel.

Performance To analyze the protocols computational and communication complexity, we assume that the size of each partys set is k, i.e., $|A_i| = k$, and there are N delegated parties $\{A_1, \cdots, A_N\}$ and 1 querying parties.

It is easy to identify every delegated party needs to perform one RIE encoding shown in step 3, which is dominated by two polynomial multiplications. Therefore, the total computational cost is $\Omega(Nk^2)$.

As for the communication overhead, the bottleneck occurs in step 5 and step 7 where two shares $[q]_0, [q]_1$ of the RIE encoding are sent to the user B and thus the communication complexity is $\Omega(Nk)$.

Finally, we highlight a critical practical concern regarding deploying this PSI protocol in real-world applications. In Step 2, user B needs to broadcast their shares to every

delegated user A. This effectively means that user B is delegating their data to each delegated user, with the delegated users themselves acting as cloud endpoints. However, this approach may not be practical as the number of delegated users increases, because the communication channels established between individual users are often less stable than the end-to-end communication between users and a centralized cloud.

Moreover, it appears infeasible to revise the protocol to eliminate the large amount of frequent end-to-end communication between users while maintaining the protocol's security. To address this issue, we present a different but efficient solution that resolves this problem without increasing the asymptotic computational and communication complexity.

3 Preliminaries

3.1 Random Intersection Encoding

This section introduces an essential (secure) tool for computing set intersections—Random Intersection Encoding (RIE). Assume the presence of a trusted third party denoted as C. Then, N computing parties $\{A_1, \ldots, A_N\}$ can compute the intersection using the following RIE algorithm:

Input: a trusted third party denoted as C and N computing parties $\{A_1, \ldots, A_N\}$

Output: the intersection among the N computing parties $\{A_1, \ldots, A_N\}$

1 Each party A_i re-encodes its set elements $a \in A_i$ by appending a hash value calculated using a hash function, denoted as

$$a \leftarrow a||h(a)$$

2 Each party A_i encodes its private set into a polynomial

$$p_i = \prod_{a \in A_i} (x - a)$$

3 A_i and C agree on a key, and A_i securely transmits p_i to C using this key

4 C generates N random polynomials $\omega_1, \ldots, \omega_N$ and computes the polynomial

$$r \leftarrow \sum_i \omega_i p_i$$

5 C sends r to each A_i through an encrypted channel

6 A_i determines the intersection $\cap A_i$ by checking whether its set elements are roots of the polynomial r

Algorithm 1: Randomized Intersection Encoding (RIE) protocol

We argue the correctness of Algorithm 1 by proving that the algorithm indeed returns the intersection $\cap_i A_i$.

Proof. Let $g = \gcd(p_1, \ldots, p_N)$, then $r = \left(\sum_i \omega_i \frac{p_i}{g}\right) g$. The roots of r have two sources:

- Roots from the intersection polynomial g.

- Roots from the random polynomial $\sum_i \omega_i \frac{p_i}{g}$.

The formats of these roots are easily distinguishable:

- Roots of the random polynomial are highly unlikely (with negligible probability) to have the form $a||h(a)$.
- Roots of the intersection polynomial g have the form $a||h(a)$, which constitute the intersection $\cap A_i$.

Thus, the algorithm is correct. □

3.1.1 Security Proof

To prove the algorithm's security, we show that A_i learns nothing beyond the intersection information. Equivalently, this requires proving that the output r of RIE does not reveal information beyond $\cap A_i$.

Randomization Lemma Let $p_1, p_2 \in \mathbb{Z}_q^{\leq\alpha}[x]$ be coprime polynomials, and $\omega_1, \omega_2 \sim \mathbb{Z}_q^{\leq\beta}[x]$, where $\alpha \leq \beta$. Then, $\sum_{i=1}^{2} p_i\omega_i$ is a random polynomial over $\mathbb{Z}_q^{\leq\alpha+\beta}$.

Proof. Consider solutions (ω_1, ω_2) to the equation $\sum_{i=1}^{2} p_i\omega_i = r \in \mathbb{Z}_q^{\leq\alpha+\beta}[x]$. Assume there is a particular solution (ω_1^*, ω_2^*). Then, the general solution can be expressed as:

$$\omega_1 = \omega_1^* - k \cdot p_2, \quad \omega_2 = \omega_2^* + k \cdot p_1, \quad k \in \mathbb{Z}_q^{\leq\beta-\alpha}[x].$$

The number of solutions equals the number of k, which is $q^{\beta-\alpha+1}$. As there are $q^{2\beta+2}$ possible combinations of (ω_1, ω_2), we conclude that r can take any value in $\mathbb{Z}_q^{\leq\alpha+\beta}[x]$, completing the proof. □

Security Proof From the randomization lemma, if $\omega_i \sim \mathbb{Z}_q^{\leq\beta}[x]$ and $\frac{p_i}{g} \sim \mathbb{Z}_q^{\leq\alpha}[x]$, then $\sum_{i=1}^{N} \omega_i \frac{p_i}{g}$ is a random polynomial over $\mathbb{Z}_q^{\leq\alpha+\beta}[x]$.

Finally, since $\sum_{i=1}^{N} \omega_i p_i \sim \mathbb{Z}_q^{\leq\alpha+\beta}[x]$, it follows that $(\sum_{i=1}^{N} \omega_i p_i)$ and r are indistinguishable. Thus, the algorithm is secure.

3.2 Delegated Private Set Intersection

Let the query user $B = A_{N+1}$, and adjust the RIE algorithm step 5 as "only send r to A_{N+1}". In this case, delegated private set intersection can be directly reduced to Random Intersection Encoding (RIE). The only remaining issue is that RIE assumes that C is a trusted third party that knows the private set polynomial p_i of each A_i. To remove the trusted third-party assumption, a direct idea is to use homomorphic encryption to construct a strengthened version of RIE, which is the solution proposed. Here, we rewrite its simplified version in Algorithm 2.

3.2.1 Brief Security Analysis

Through homomorphic encryption, A_i can securely delegate its private set to C in step 4. Additionally, C computes r in step 5 without learning p_i. Thus, the privacy of each A_i's private set is preserved during the computation.

Input: an untrusted cloud denoted as C, N delegating parties $\{A_1, \ldots, A_N\}$ and one enquiring party A_{N+1}

Output: the enquiring user A_{N+1} acquires the intersection among the $N+1$ computing parties $\{A_1, \ldots, A_N\}$

1 Each party A_i re-encodes its set elements $a \in A_i$ by applying a hash function to compute the hash value and appending it to the element. Denote the result as $a \leftarrow a||h(a)$;

2 Each party A_i encodes its private set as a polynomial:

$$p_i = \prod_{a \in A_i} (x - a).$$

3 A_i and C agree on the keys of a threshold homomorphic encryption scheme. Each A_i holds its private key sk_i, and C holds the public key pk

4 A_i encrypts its set polynomial as $Enc_{pk}(p_i)$ and sends it to C

5 C generates N random polynomials $\omega_1, \ldots, \omega_N$ and computes the encrypted result polynomial homomorphically:

$$Enc(r) \leftarrow \sum_i \omega_i \cdot Enc_{pk}(p_i).$$

6 C sends $Enc_{pk}(r)$ to A_i

7 All A_i collaboratively decrypt $Enc_{pk}(r)$ and ensure that only A_{N+1} obtains the final decryption result r

8 A_{N+1} determines the intersection $\bigcap A_i$ by checking whether its set elements are roots of the polynomial r

Algorithm 2: Delegated Multi-party Private Set Intersection Protocol from Homomorphic Encryptions

3.3 Multiplication Triples

In (linear) secret sharing schemes, securely performing multiplication is complex. The problem is formally described as follows:

A trusted third party (Trusted Dealer) secret-shares $x \in \mathbb{F}_p$ and $y \in \mathbb{F}_p$ as:

$$[x] = (x_1, x_2), \quad [y] = (y_1, y_2).$$

The trusted third party securely sends the shares to computing parties P_1 and P_2 via secure channels:

$$P_1 \leftarrow (x_1, y_1), \quad P_2 \leftarrow (x_2, y_2).$$

The goal is for P_1 and P_2 to compute a secret share of xy without revealing x and y.

3.3.1 Beaver's Triples

A common approach is to use Beaver's triples, i.e., pre-generated random values (a, b, c) such that $c = a \times b$. The specific steps are as follows:

Preprocessing Phase

1. The trusted third party generates a random Beaver triple (a, b, c) and creates its secret shares $([a], [b], [c])$.

2. The trusted third party securely sends:

$$P_1 \leftarrow (a_1, b_1, c_1), \quad P_2 \leftarrow (a_2, b_2, c_2).$$

Online Computation Phase

1. P_1 computes:
$$[x-a]_1 \leftarrow x_1 - a_1, \quad [y-b]_1 \leftarrow y_1 - b_1.$$

2. P_2 computes:
$$[x-a]_2 \leftarrow x_2 - a_2, \quad [y-b]_2 \leftarrow y_2 - b_2.$$

3. P_1 and P_2 collaboratively reconstruct:
$$e = x - a, \quad d = y - b.$$

4. P_1 computes its share of xy:
$$[xy]_1 \leftarrow c_1 + e \cdot b_1 + d \cdot a_1.$$

5. P_2 computes its share of xy:
$$[xy]_2 \leftarrow c_2 + e \cdot b_2 + d \cdot a_2 + e \cdot d.$$

3.3.2 Correctness of the Algorithm

The correctness of the algorithm is demonstrated as follows:

$$[xy]_1 + [xy]_2 = c + eb + da + ed = ab + (x-a)b + (y-b)a + (x-a)(y-b) = xy.$$

3.3.3 Security of the Algorithm

- **Preprocessing Phase:** Since a trusted third party is involved, the secret shares received by P_1 and P_2 are uniformly distributed and indistinguishable from random values.
- **Online Computation Phase:** Both P_1 and P_2 receive e and d. As a and b are unknown random values, e and d appear as uniformly distributed random variables to P_1 and P_2.
- P_1's share $[xy]_1$ includes the random value c_1, making it completely random from P_2's perspective. Similarly, P_2's share $[xy]_2$ appears random to P_1.

Thus, the algorithm preserves the secrecy of the inputs while securely computing the secret-shared product.

4 Delegated PSI Framework

This section presents a new framework for delegated PSI on the basis of Random Intersection Encoding (RIE) and secret sharing. For a polynomial

$$p = \sum_{i=0}^{n} p[i] \cdot x^i \in \mathbb{Z}_q^{\leq n}[x],$$

a simple $(2, 2)$ secret sharing scheme is used, where:

$$[p] = ([p]_1, [p]_2) \quad \text{such that} \quad p = [p]_1 + [p]_2.$$

Secret sharing is linear, so addition in the shares (denoted as algorithm `A_add`) is straightforward:

$$[p + q] = ([p]_1 + [q]_1, [p]_2 + [q]_2).$$

The main challenge is performing multiplication in the shares, which requires designing an algorithm `A_mul` such that:

$$P_1 \text{ holds } [p]_1, [q]_1, \quad P_2 \text{ holds } [p]_2, [q]_2, \quad \text{and computes } [p \cdot q]_1, [p \cdot q]_2.$$

4.1 Polynomial multiplication in the shares

To address this, Beaver's Triples are introduced for securely multiplying shares over $\mathbb{Z}_q$. For any $a \in \mathbb{Z}_q$, $b \in \mathbb{Z}_q$, Beaver's Triples are defined as:

$$[a \cdot b]_1, [a \cdot b]_2 \leftarrow \texttt{A\_Beaver}([a], [b]).$$

Polynomial multiplication $p \cdot q$ can be rewritten as:

$$p \cdot q = \sum_{i=0}^{2n} \sum_{j=0}^{i} (p[j] \cdot q[i-j]) x^i.$$

where any coefficient of the polynomials p and q with index i beyond $[0, n]$ is considered zero. Using this property, the algorithm `A_mul` is constructed as follows:

1. P_i receives its shares, including:

$$([p[0]]_i, \ldots, [p[n]]_i), \quad ([q[0]]_i, \ldots, [q[n]]_i).$$

2. P_i invokes `A_Beaver` to compute:

$$[p[j] \cdot q[k-j]]_i \quad \text{for all } 0 \leq k \leq 2n,\ 0 \leq j \leq k.$$

3. P_i calculates the sums:

$$[p \cdot q[k]]_i \leftarrow \sum_{j=0}^{k} [p[j] \cdot q[k-j]]_i \quad \text{for all } k.$$

4.2 Distributive Generation of Beaver's Triples

It is important to note that the method shown in the aforementioned section for generating multiplication triples assumes the existence of a trusted third party during the preprocessing phase. However, in real-world scenarios, such an idealized trusted third party often does not exist. Therefore, a critical requirement is to design a solution that allows computing parties to efficiently generate multiplication triples even in a setting where they do not trust each other. To address this challenge, we propose a construction of Beaver's triples generation protocol between the user A_i and the cloud server C under the semi-honest security model using Fully Homomorphic Encryption (FHE). The detailed algorithm for genereating the random tiples $([a], [b], [ab])$ is described as follows:

1. The party A_i generates a private key:

$$sk \leftarrow \mathcal{FHE}.KeyGen().$$

2. A_i randomly generates secret shares a_1, b_1 sampled from $\mathbb{F}_q$:
$$a_1, b_1 \xleftarrow{\$} \mathbb{F}_q.$$
3. C randomly generates secret shares a_2, b_2 sampled from $\mathbb{F}_q$:
$$a_2, b_2 \xleftarrow{\$} \mathbb{F}_q.$$
4. A_i encrypts his secret shares:
$$\mathrm{Enc}_{sk}(a_1) \leftarrow \mathcal{FHE}.Encrypt(sk, a_1), \quad \mathrm{Enc}_{sk}(b_1) \leftarrow \mathcal{FHE}.Encrypt(sk, b_1),$$
and sends these encrypted values to C through a secret channel between A_i and C.
5. Party C performs homomorphic additions to compute:
$$\mathrm{Enc}_{sk}(a) \leftarrow a_2 + \mathrm{Enc}_{sk}(a_1), \quad \mathrm{Enc}_{sk}(b) \leftarrow b_2 + \mathrm{Enc}_{sk}(b_1).$$
6. Party C performs a homomorphic multiplication to compute the encrypted product:
$$\mathrm{Enc}_{sk}(ab) \leftarrow \mathrm{Enc}_{sk}(a) \cdot \mathrm{Enc}_{sk}(b).$$
7. Party C randomly generates a secret share:
$$c_2 \xleftarrow{\$} \mathbb{F}_q.$$
8. Party C computes another secret share of the product:
$$\mathrm{Enc}_{sk}(c_1) \leftarrow \mathrm{Enc}_{sk}(ab) - c_2.$$
and sends the computed value to A_i through a secret channel between A_i and C.
9. Party A_i decrypt $\mathrm{Enc}_{sk}(c_1)$:
$$c_1 \leftarrow \mathcal{FHE}.Decrypt(sk, \mathrm{Enc}_{sk}(c_1)).$$

Remarks The user A_i first generates a private key, then encrypts his secret shares a_1 and b_1, and sends the encrypted values to C. Party C computes the homomorphic addition and multiplcation to obtain the encrypted ab. Subsequently, the two parties sequentially retrieve the secret shares of ab, *i.e.*, $[ab] = (c_1, c_2)$ ensuring both privacy and correctness.

4.3 Key Exchange

To ensure the functionality of our PSI framework, all computing parties, including $A_1, \ldots, A_{N+1}$, must share a common secret key s^* without the cloud server knowing the key. To adhere to the definition of delegated PSI, all communication must go through the cloud, avoiding direct communication between users. We propose the following simple yet efficient key exchange protocol based on any semantically secure public key encryption scheme $\mathcal{PKE}$:

1. Each A_i invokes the key generation algorithm to generate its key pair:
$$sk_i, pk_i \leftarrow \mathcal{PKE}.\mathrm{KeyGen}().$$
2. Each A_i broadcasts its public key pk_i via the cloud to all other parties $A_j, j \neq i$.
3. One randomly selected party among $\{A_i\}_i$ generates a common symmetric secret key s^* and encrypts it for each $A_j, j \neq i$:
$$c_j \leftarrow \mathcal{PKE}.\mathrm{Encrypt}(pk_j, s^*),$$
and broadcasts c_j via the cloud to A_j.
4. Each A_j decrypts c_j using its private key sk_j to obtain the common symmetric secret key s^*.

4.4 New Framework

With the capability to perform addition and multiplication on secret shares, we present a complete delegated multi-party PSI framework in Algorithm 3. We assume that the common secret key s^* and the Beaver's triples have been pre-generated prior to the initiation of the PSI protocol.

Input: An untrusted cloud server C, N delegating parties $\{A_1, \ldots, A_N\}$, and one querying party A_{N+1}.

Output: The querying party A_{N+1} obtains the intersection among the $N+1$ computing parties $\{A_1, \ldots, A_{N+1}\}$.

1 Each A_i re-encodes its set elements $a \in A_i$ by appending a hash value:

$$a \leftarrow a||h(a).$$

2 Each A_i encodes its private set into a polynomial:

$$p_i = \prod_{a \in A_i} (x - a).$$

3 Each A_i create secret shares of its polynomial $[p_i]$ such that $[p_i] = ([p_i]_0, [p_i]_1)$,
4 retaining $[p_i]_1$ and sending $[p_i]_0$ to the cloud server C.
5 Each A_i generate a random polynomial ω_i and create its shares:
$[\omega_i] = ([\omega_i]_0, [\omega_i]_1), \quad \forall i$. Retain $[\omega_i]_1$ and send $[\omega_i]_0$ to the cloud C.
6 The cloud C computes:

$$\left[\sum_{i=1}^{N} \omega_i p_i\right]_0 = \sum_i [\omega_i]_0 \cdot [p_i]_0,$$

using algorithms `A_add` and `A_mul` multiple times.
7 Each A_i computes:

$$[\omega_i p_i]_1 = [\omega_i]_1 \cdot [p_i]_1,$$

using `A_mul`.
8 The cloud C sends $\left[\sum_{i=1}^{N} \omega_i p_i\right]_0$ to A_{N+1} through an encrypted channel.
9 Each A_i sends the encrypted $[\omega_i p_i]_1$ (using a common secret key s^*) to C, which then forwards it to A_{N+1}.
10 Party A_{N+1} decrypts the received $[\omega_i p_i]_1$ for all i and, together with $\left[\sum_{i=1}^{N} \omega_i p_i\right]_0$, reconstructs:

$$r = \sum_i \omega_i p_i,$$

and determines the intersection:

$$\cap A_i.$$

Algorithm 3: Delegated Multi-party Private Set Intersection Protocol Using Secret Sharing

Remarks The correctness of the algroithm can be proved using the analogous argument we used to prove Algorithm 2. We focus here to argue that the proposed framework is secure against semi-honest adversaries.

Proof. The security of the protocol is proved using the simulation paradigm. The ideal model is described as follows:

1. Each computing party A_i holds a private set A_i and constructs the corresponding set polynomial p_i.
2. The cloud server C holds private random polynomials ω_i for all i.
3. Through some "supernatural" operation (oracle machine access), A_{N+1} directly obtains:
$$r = \sum_i \omega_i p_i,$$
and computes the intersection:
$$\cap_i A_i.$$

To show security, simulators S_i, S_c, and S_{N+1} are constructed to mimic the views of A_i, C, and A_{N+1}, respectively, in the real world.

Simulator S_i simulates A_i's perspective in the real-world protocol (Algorithm 3):

1. Generate random polynomials:
$$r_i \leftarrow \mathbb{Z}_q^{\leq k}[x] \quad \text{for all } i,$$
to simulate Step 3 in the real model.
2. Generate random polynomials:
$$s_i \leftarrow \mathbb{Z}_q^{\leq k}[x] \quad \text{for all } i,$$
to simulate Step 5 in the real model.
3. Generate two additional random polynomials to simulate Steps 8 and 9 in the real model.

Simulator S_c simulates C's perspective in the real-world protocol:

1. Generate random polynomials:
$$r_i \leftarrow \mathbb{Z}_q^{\leq k}[x] \quad \text{for all } i,$$
to simulate Step 3 from C's perspective in the real model.
2. Generate a random polynomial:
$$s \leftarrow \mathbb{Z}_q^{\leq k}[x],$$
to simulate Step 9 from C's perspective in the real model.

Simulator S_{N+1} simulates A_{N+1}'s perspective in the real-world protocol:

1. Generate random polynomials:
$$r_i \leftarrow \mathbb{Z}_q^{\leq k}[x] \quad \text{for all } i,$$
to simulate Step 3 from A_{N+1}'s perspective.

2. Generate random polynomials:

$$s_i \leftarrow \mathbb{Z}_q^{\leq k}[x] \quad \text{for all } i,$$

to simulate Step 5 from A_{N+1}'s perspective.

3. Extract the real-world value of: $r = \sum_i \omega_i p_i$, and construct secret shares: $[r]_0, \ldots, [r]_N$.

4. Encrypt $[r]_0$ to simulate Step 8 as $\mathrm{Enc}([r]_0)$.

5. Encrypt $[r]_1, \ldots, [r]_N$ to simulate Step 9 as $\mathrm{Enc}([r]_1), \ldots, \mathrm{Enc}([r]_N)$.

The views generated by simulators S_i, S_c, and S_{N+1} are indistinguishable from the real-world protocol executions. This is because the random polynomials r_i and s_i do not reveal any private information about the sets A_i or the polynomials p_i, and the encryption ensures that the cloud server C and other parties cannot access unauthorized information. Thus, the protocol securely emulates the ideal model. □

4.5 Performance

To analyze the protocol's computational and communication complexity, we assume that the size of each party's set is k, i.e., $|A_i| = k$.

Computational Complexity The computational bottleneck of the protocol occurs in Step 6, which involves multiple invocations of `A_mul`. Specifically, Step 6 performs $\Omega(N)$ polynomial multiplications in the share state. Each polynomial multiplication requires $\Omega(k^2)$ `A_mul` invocations, and each `A_mul` invocation completes in constant time. Consequently, the total computational complexity of the protocol is:

$$\Omega(Nk^2).$$

Communication Complexity The polynomial p_i representing the set A_i has a length of $k \log q$ bits. The communication bottleneck arises in Steps 3 and 4, where $\Omega(N)$ polynomials are transmitted in total. Thus, the total communication complexity is:

$$\Omega(Nk \log q).$$

5 Optimization Strategies

We propose the following techniques to reduce the computational complexity of the protocol.

5.1 Binning Technique

We introduce another set representation method: the hash table (HT) on top of the set polynomial representation.

The hash table of a set S is a two-dimensional linked list (or equivalently, a matrix) denoted as $\mathrm{HT}[\cdot]$, which satisfies:

$$\forall s \in S \Leftrightarrow s \in \mathrm{HT}[H(s)],$$

where $H(\cdot) : \{0,1\}^* \to \{0, \cdots, h-1\}$ is a hash function. The algorithmic process (denoted as $\mathcal{A}_{ht}$) for constructing a hash table for a set S procceds as follows:

1. For every $s \in S$:

(a) Compute the address $H(s)$.

(b) Access the corresponding position $H(s)$ in the hash table, $\mathrm{HT}[H(s)]$.

(c) Insert the element s into $\mathrm{HT}[H(s)]$.

2. return the hash table HT

Remark Each position $j \in \{0, 1, \ldots, h-1\}$ in the hash table may contain multiple elements from the set S. Thus, $\mathrm{HT}[j]$ itself forms a small subset comprising elements that share the same hash address. In the literature, these subsets $\mathrm{HT}[j]$ are commonly referred to as "bins."

According to the balls-into-bins principle, when the row number of the hash table (denoted as h) is asympotically as large as set size $|S|$ (denoted as k), such that $h = \Omega(k)$, each bin will contain at most $\Omega(\log k)$ elements with high probability. For security reason, the number of elements in each row should be identical to prevent inforamtion leakage on the dataset distribution. This can be achived by inserting 'dummy' items (dentoed as s_{dummy}) to make the length of each row identical. In other terms, the size of the hash table should be $\Omega(k \log k)$. Moreover, with this technique, the computational complexity in Step 6 of Algorithm 3 is reduced to $\Omega(Nk(\log k)^2)$.

Numerical Analysis Nevertheless, in practical implementations, it is essential to compute the exact value of the maximum load, defined as the maximum number of set elements in a single row of the hash table, to optimize storage overhead. Asymptotic analysis alone does not provide sufficient precision for this purpose. To address this, we introduce a numerical computation method based on the Chernoff bounds.

Chernoff Bounds are a powerful tool for estimating the probability that a random variable deviates significantly from its expected value. They are particularly useful for analyzing large-scale independent random events. The upper tail of the Chernoff Bound is specifically used to study cases where a random variable exceeds its expectation significantly, which is precisely our concern when analyzing the upper deviation of a distribution.

Consider $X_1, X_2, \ldots, X_k$ as k independent Bernoulli random variables, where each X_i takes values in $\{0, 1\}$. Define $X = \sum_{i=1}^{k} X_i$ as their sum, with an expected value $\mu = \mathbb{E}[X]$. The upper tail Chernoff Bound provides an upper limit on the probability that X deviates above its expectation:

$$\Pr\left(X \geq (1+\delta)\mu\right) \leq \exp\left(-\frac{\delta^2 \mu}{2+\delta}\right)$$

where $\delta > 0$ represents the deviation parameter, meaning that $(1+\delta)\mu$ denotes a deviation of $\delta\mu$ beyond the expectation μ. It is evident that as δ increases, the probability of exceeding the expected value decreases exponentially.

Returning to the maxload problem, let $\mu = \frac{k}{h}$. The upper tail Chernoff Bound formula provides an estimate for the maximum load of a single bin. However, in the maxload problem, we need to estimate the probability that *none* of the h rows of the hash table exceeds the maximum load. To achieve this, we introduce the Union Bound.

For a set of events $E_1, E_2, \ldots, E_h$, the probability of their union satisfies the following inequality:

$$\Pr\left(\bigcup_{i=1}^{h} E_i\right) \leq \sum_{i=1}^{h} \Pr(E_i)$$

The Union Bound states that the probability of at least one of multiple events occurring does not exceed the sum of their individual probabilities. In other words, it provides an upper bound for the probability of a union of events.

Clearly, if we define E_i as the event that "the number of set elements in the i-th bin exceeds the maximum load," then $\Pr\left(\bigcup_{i=1}^{h} E_i\right)$ represents the probability that *at least one* of the h hash table rows exceeds the maximum load. Applying the Union Bound, we obtain:

$$\Pr\left(\bigcup_{i=1}^{h} E_i\right) \leq h \cdot \exp\left(-\frac{\delta^2 \cdot k}{h(2+\delta)}\right)$$

Based on the union bound formula, we construct an algorithm to provide a numerical estimation of the maximum load and the number of hash table rows as follows:

Input: Total number of set elements N_{elts}, bias parameter δ, target probability P_{target}

Output: Maximum load per row $L_{\max}$, final number of hash table rows N_{rows}, computed probability P_{computed}

Set initial number of hash table rows $N_{\text{rows}} \leftarrow N_{\text{elts}}$;

Compute average load per bin $L_{\text{avg}} \leftarrow \frac{N_{\text{elts}}}{N_{\text{rows}}}$;

Compute initial probability bound:

$$P_{\text{computed}} \leftarrow N_{\text{rows}} \cdot \exp\left(-\frac{\delta^2 \cdot L_{\text{avg}}}{2+\delta}\right)$$

while $P_{computed} > P_{target}$ **do**

  Decrease the number of hash table rows $N_{\text{rows}} \leftarrow N_{\text{rows}} - 0.01 \cdot L_{\text{avg}}$

  Recompute the average load $L_{\text{avg}} \leftarrow \frac{N_{\text{elts}}}{N_{\text{rows}}}$

  Recompute the probability bound:

$$P_{\text{computed}} \leftarrow N_{\text{rows}} \cdot \exp\left(-\frac{\delta^2 \cdot L_{\text{avg}}}{2+\delta}\right)$$

Compute the maximum load $L_{\max} \leftarrow \lfloor (1+\delta) \cdot L_{\text{avg}} \rfloor$

Set final number of hash table rows $N_{\text{rows}} \leftarrow \lfloor N_{\text{rows}} \rfloor$

return $(L_{\max}, N_{\text{rows}}, P_{\text{computed}})$

Algorithm 4: Computation of Maximum Load Per Row and Row Allocation

Finally, we provide several concrete parameter sets used in our actual PSI implementation, as shown in Table 1. In these configurations, we set the probability of "exceeding the maximum load" to a negligible value, i.e.,

$$P_{\text{target}} \stackrel{def}{=} \Pr\left(\bigcup_{i=1}^{h} T_i\right) \leq 2^{-40},$$

and set the deviation parameter δ to 1.5, meaning that the maximum load is 2.5 times the average load $\frac{k}{h}$.

Table 1: Concrete hash table parameters with estimated maximum loads.

Set Size k	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
Row Number h	21	81	314	1209	4658	17964
Deviation Parameter δ	1.5	1.5	1.5	1.5	1.5	1.5
Maximum Load	120	124	130	135	140	145

5.2 Lagrange Interpolation

A polynomial $p(x)$ can be uniquely represented using its interpolated form, given k distinct interpolation points $(x_i, p(x_i))$ for $i \in \{1, \ldots, k\}$. Using Lagrange interpolation, $p(x)$ can be reconstructed in $\Omega(k^2)$ time.

Addition and multiplication of polynomials in the interpolated form become element-wise operations:

$$p(x) + q(x) \Leftrightarrow (x_i, p(x_i) + q(x_i)) \quad \text{for } i \in \{1, \ldots, k\},$$

$$p(x) \cdot q(x) \Leftrightarrow (x_i, p(x_i) \cdot q(x_i)) \quad \text{for } i \in \{1, \ldots, 2k\}.$$

Note that the complexity of the multiplication operation is reduced to $\Omega(k)$. By combining this with the binning technique, the overall protocol complexity is further reduced to:

$$\Omega(Nk \log k + k(\log k)^2).$$

Remark Note that each party's hash table has $\Omega(k)$ rows, and each row contains $\Omega(\log k)$ set elements. Thus, each row is represented by a set polynomial of degree $\Omega(\log k)$. As discussed earlier, the overall protocol requires $\Omega(Nk)$ polynomial multiplications. Since each polynomial multiplication in interpolation format has a computational cost of $\Omega(\log k)$, the total computation cost for polynomial multiplications is $\Omega(Nk \log k)$. Moreover, Step 10 of Algorithm 3 involves Lagrange interpolation, which requires performing an inverse transform to reconstruct the original standard polynomial representation. This step incurs an additional computation cost of $\Omega(k(\log k)^2)$. Therefore, we conclude that the overall complexity of the protocol is: $\Omega(Nk \log k + k(\log k)^2)$

Lagrange Interpolation in Quadratic Time We present an efficient method to compute Lagrange interpolation within quadratic time. Given k distinct interpolation points:

$$(x_i, f(x_i)) \quad \text{for } i \in \{1, \ldots, k\},$$

our goal is to reconstruct the polynomial $f(x)$.

The core idea of Lagrange interpolation is to construct a set of basis polynomials $\ell_i(x)$ that satisfy:

$$\ell_i(x) = \begin{cases} 1, & \text{if } x = x_i, \\ 0, & \text{if } x \neq x_i \text{ and } x \in \{x_1, \ldots, x_k\}. \end{cases}$$

Using these basis polynomials, we can uniquely reconstruct $f(x)$ as:

$$f(x) = \sum_{i=1}^{k} f(x_i)\ell_i(x).$$

The Lagrange basis polynomials take the explicit form:

$$\ell_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^{k} \frac{x - x_j}{x_i - x_j}.$$

It is observed that the Lagrange basis polynomials $\ell_i(x)$ can be rewritten in an equivalent form, often referred to in the literature as the "barycentric form":

$$\ell_i(x) = w_i \cdot \frac{\ell(x)}{x - x_i},$$

where

$$\ell(x) = \prod_{j=1}^{k}(x - x_j),$$

$$w_i = \frac{1}{\prod_{\substack{j=1 \\ j\neq i}}^{k}(x_i - x_j)}.$$

If the interpolation points x_j are fixed, both $\ell(x)$ and w_i can be precomputed. Thus, the main computational bottleneck for evaluating $\ell_i(x)$ is reduced to performing the polynomial division $\frac{\ell(x)}{x-x_i}$, which has a complexity of $\Omega(k)$. Leveraging this optimization, the overall computational complexity of the fast Lagrange interpolation algorithm is reduced to $\Omega(k^2)$.

6 Set Update

6.1 A Simple Update Algorithm Without Information Leakage

We present a method for a delegated user to securely delegate their set without compromising privacy, denoted as $\mathcal{A}_{\text{simple\_delegate}}$:

1. The delegated user A invokes $\mathcal{A}_{\text{ht}}$ to create a private hash table associated with their set:
$$\text{HT}_A \leftarrow \mathcal{A}_{\text{ht}}(A).$$

2. The delegated user replaces each row of HT_A with a corresponding set polynomial, i.e., a polynomial with all elements in that row of HT_A as its roots.

3. The delegated user A randomly selects a seed r and uses a pseudo-random number generator (PRG) to produce a sequence of random values matching the length of HT_A, treating this sequence as one secret share of the hash table:
$$[\text{HT}_A]_1 \leftarrow \text{PRG}(r).$$

4. The delegated user computes the other secret share as:
$$[\text{HT}_A]_0 \leftarrow \text{HT}_A - [\text{HT}_A]_1.$$

5. The delegated user retains r and uploads $[\text{HT}_A]_0$ to the cloud C through an encrypted channel.

Remark The security of this delegation process is straightforward, based on the security of the pseudo-random number generator and the additive secret sharing scheme. The share $[\text{HT}_A]_1$ retained by user A is essentially a random sequence. Therefore, it suffices to store only the random seed r and discard $[\text{HT}_A]_1$ generated in Step 3. This approach significantly reduces the delegated user's local storage requirements while fully delegating their set to the cloud.

Next, we describe how a delegated user can securely update their set, denoted as $\mathcal{A}_{\text{simple\_update}}$:

1. The delegated user A reconstructs their private share using the random seed r:
$$[\text{HT}_A]_1 \leftarrow \text{PRG}(r).$$

2. A requests the other secret share $[\mathrm{HT}_A]_0$ from the cloud through an encrypted channel and reconstructs the hash table:

$$\mathrm{HT}_A \leftarrow [\mathrm{HT}_A]_0 + [\mathrm{HT}_A]_1.$$

3. If A wants to add an item s to their set:
   (a) Compute the address $H(s)$.
   (b) Update the set polynomial in the corresponding bin:

$$\mathrm{HT}_A[H(s)] \leftarrow \mathrm{HT}_A[H(s)] \cdot \frac{x - s}{x - s_{\mathrm{dummy}}}.$$

4. If A wants to delete an item s from their set:
   (a) Compute the address $H(s)$.
   (b) Update the set polynomial in the corresponding bin:

$$\mathrm{HT}_A[H(s)] \leftarrow \mathrm{HT}_A[H(s)] \cdot \frac{x - s_{\mathrm{dummy}}}{x - s}.$$

5. Randomly select a new seed r' and compute one share:

$$[\mathrm{HT}_A]_1 \leftarrow \mathrm{PRG}(r').$$

6. Compute the other share:

$$[\mathrm{HT}_A]_0 \leftarrow \mathrm{HT}_A - [\mathrm{HT}_A]_1.$$

7. Retain r' and upload $[\mathrm{HT}_A]_0$ to the cloud C through an encrypted channel.

Remark The security of the update operation is straightforward, as all sensitive operations are performed locally by A. By secret-sharing the updated hash table, A ensures the cloud learns nothing about what or where updates occurred.

The computational complexity of the update process is minimal compared to the multi-party intersection protocol in Algorithm 3, as the primary bottleneck is the randomness generation in Step 1 and Step 5, which is very efficient. The communication complexity involves downloading and uploading shares, amounting to $k \log k$, where k denotes the size of the delegated user's set. For moderately sized datasets, this communication overhead is entirely practical.

6.2 A more efficient update algorithm with access pattern leakage

We describe another approach inspired by a novel updating mechanism proposed in [ATD20] to achieve a more efficient updating algorithm, assuming that a certain level of information leakage is acceptable. Suppose the user wishes to update only a small number of items, such as adding a single item to the hash table. Without privacy guarantees, a natural approach is to compute the address (row number of the hash table) associated with the item, extract the specific row from the cloud, and insert the item into the hash table. This approach achieves constant-time updates and constant communication overhead, independent of the user's dataset size. However, the privacy issue arises because the cloud learns which row of the hash table is accessed (i.e., learns $H(s)$), thereby leaking critical side information about the element s.

To address this information leakage, the user can perform a random permutation on their secret hash table before uploading it to the cloud. Consequently, the cloud cannot

infer the sensitive $H(s)$ by observing the user's access pattern. However, the cloud may still deduce hash collisions (e.g., $h(s_1) = h(s_2)$) when the user accesses the same row multiple times during updates.

We formalize the delegating process as $\mathcal{A}_{\text{efficient\_delegate}}$ as follows:

1. User A repeats Steps 1–4 in $\mathcal{A}_{\text{simple\_delegate}}$ to generate:

$$\{r_i\}_{i=1,\cdots,h}, [\mathrm{HT}_A]_0, [\mathrm{HT}_A]_1,$$

   where each row of the hash table HT_A is masked with a unique randomness seed r_i.

2. User A obfuscates the mapping between the row indices of the hash table and the rows of the vector by applying a random permutation:

$$[\hat{\mathrm{HT}}_A]_0 = \pi_A([\mathrm{HT}_A]_0),$$

   where π_A is a random permutation applied to the rows of HT_A.

3. User A uploads $[\hat{\mathrm{HT}}_A]_0$ to the cloud C, while retaining the randomness $\{r_i\}$ and the permutation π_A.

The updating process $\mathcal{A}_{\text{efficient\_update}}$ proceeds as follows:

1. User A initializes h counters $\{cnt_i\}$ to zero, indicating the number of times the i-th row of the hash table HT_A has been accessed.

2. User A computes the row number for the targeted item s as $H(s)$.

3. User A computes the scrambled row number $\pi_A(H(s))$ and downloads $[\hat{\mathrm{HT}}(\pi_A(H(s)))]_0$ from the cloud.

4. User A reconstructs the polynomial $\mathrm{HT}(H(s))$.

5. User A updates (either adds or deletes) the polynomial $\mathrm{HT}(H(s))$ using the techniques described in Steps 3 and 4 of $\mathcal{A}_{\text{simple\_delegate}}$.

6. User A increments the counter $cnt_{H(s)}$ as:

$$cnt_{H(s)} \leftarrow cnt_{H(s)} + 1,$$

   indicating that the $H(s)$-th row of HT_A has been accessed.

7. User A re-shares the polynomial $\mathrm{HT}(H(s))$ as $[\mathrm{HT}(H(s))]_0, [\mathrm{HT}(H(s))]_1$ with newly generated randomness:

$$r' \leftarrow \mathrm{PRG}(r_{H(s)} || cnt_{H(s)}).$$

8. User A uploads $[\mathrm{HT}(H(s))]_0$ to $[\hat{\mathrm{HT}}(\pi_A(H(s)))]_0$ in the cloud and retains $\{r_i\}_i$ and $\{cnt_i\}$.

Remark Suppose q updates are performed for items $s_1, \ldots, s_q$. The access pattern can be quantified in a table $\mathrm{T}_{\text{pattern\_access}}$. During the q updates, if the cloud observes that the i-th and j-th updates access the same k-th ($1 \le k \le h$) row, it records i, j in the k-th row of $\mathrm{T}_{\text{pattern\_access}}$. The information leakage is fully captured in $\mathrm{T}_{\text{pattern\_access}}$, and we assume this level of leakage is acceptable for most real-world applications.

It is evident that the user downloads/uploads only one row of the hash table share $[\hat{\mathrm{HT}}_A]_0$ and updates this row locally. As a result, both computational and communication overheads are reduced to $\Omega(\log k)$.

7 Conclusion

In this work, we redefine the Delegated PSI problem, emphasizing the importance of silent processing and minimal user-to-user communication as critical requirements for real-world cloud computing scenarios. We propose an ideal construction for the Delegated PSI protocol, demonstrating that the best achievable computational and communication complexities are both $\Omega(Nk)$, where N is the number of users in the protocol and k is the size of each user's set.

We first present a baseline construction based on randomized intersection encoding and additive secret sharing, augmented by Beaver's multiplication trick. This baseline construction achieves $\Omega(Nk^2)$ computational overhead and optimal $\Omega(Nk)$ communication overhead. We then introduce a computationally optimized construction that reduces the computational overhead to $\widetilde{\Omega}(Nk)$, at the cost of increasing the communication complexity to $\widetilde{\Omega}(Nk)$. The optimization techniques employed include hash table representation of a set and evaluation point representation of a polynomial. However, we argue that the optimized construction achieves near-optimal performance compared to the ideal construction, ignoring logarithmic factors in the asymptotic complexity analysis.

Additionally, we explore the problem of performing Delegated PSI efficiently in dynamic settings, where users may frequently update their delegated sets stored on the cloud.

We are currently implementing the protocols proposed in this work to provide experimental evidence demonstrating that these protocols are not only asymptotically efficient but also concretely practical. Upon completing our experiments, we will release the source code and update this paper with the corresponding experimental results.

References

[ATD15] Aydin Abadi, Sotirios Terzis, and Changyu Dong. O-psi: delegated private set intersection on outsourced datasets. In *ICT Systems Security and Privacy Protection: 30th IFIP TC 11 International Conference, SEC 2015, Hamburg, Germany, May 26-28, 2015, Proceedings 30*, pages 3–17. Springer, 2015.

[ATD17] Aydin Abadi, Sotirios Terzis, and Changyu Dong. Vd-psi: verifiable delegated private set intersection on outsourced private datasets. In *Financial Cryptography and Data Security: 20th International Conference, FC 2016, Christ Church, Barbados, February 22–26, 2016, Revised Selected Papers 20*, pages 149–168. Springer, 2017.

[ATD20] Aydin Abadi, Sotirios Terzis, and Changyu Dong. Feather: Lightweight multi-party updatable delegated private set intersection. *Cryptology ePrint Archive*, 2020.

[ATMD17] Aydin Abadi, Sotirios Terzis, Roberto Metere, and Changyu Dong. Efficient delegated private set intersection on outsourced private datasets. *IEEE Transactions on Dependable and Secure Computing*, 16(4):608–624, 2017.

[GPR+21] Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Oblivious key-value stores and amplification for private set intersection. In *Annual International Cryptology Conference*, pages 395–425. Springer, 2021.

[HFH99] Bernardo A Huberman, Matt Franklin, and Tad Hogg. Enhancing privacy and trust in electronic communities. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 78–86, 1999.

[Ker12] Florian Kerschbaum. Outsourced private set intersection using homomorphic encryption. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, pages 85–86, 2012.

[KMRS14] Seny Kamara, Payman Mohassel, Mariana Raykova, and Saeed Sadeghian. Scaling private set intersection to billion-element sets. In *Financial Cryptography and Data Security: 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers 18*, pages 195–215. Springer, 2014.

[KS05] Lea Kissner and Dawn Song. Privacy-preserving set operations. In *Annual International Cryptology Conference*, pages 241–257. Springer, 2005.

[LNZ+14] Fang Liu, Wee Keong Ng, Wei Zhang, Shuguo Han, et al. Encrypted set intersection protocol for outsourced datasets. In *2014 IEEE International Conference on Cloud Engineering*, pages 135–140. IEEE, 2014.

[Mea86] Catherine Meadows. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In *1986 IEEE Symposium on Security and Privacy*, pages 134–134. IEEE, 1986.

[QLS+15] Shuo Qiu, Jiqiang Liu, Yanfeng Shi, Ming Li, and Wei Wang. Identity-based private matching over outsourced encrypted datasets. *IEEE Transactions on cloud Computing*, 6(3):747–759, 2015.

[RS21] Peter Rindal and Phillipp Schoppmann. Vole-psi: fast oprf and circuit-psi from vector-ole. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 901–930. Springer, 2021.

[ZX15] Qingji Zheng and Shouhuai Xu. Verifiable delegated set intersection operations on outsourced encrypted data. In *2015 IEEE International Conference on Cloud Engineering*, pages 175–184. IEEE, 2015.