Privacy-Preserving Data Collaboration: Implementing Private Set Intersection using Diffie-Hellman Key Exchange

Abdul Rafi Radityo Hutomo
School of Electrical Engineering and Informatics
Institut Teknologi Bandung
Bandung, Indonesia
13522089@std.stei.itb.ac.id
abdulrafi623@gmail.com

***Abstract*—In the era of strict data privacy regulations, organizations often face the challenge of identifying common datasets—such as shared customers or cross-institutional fraud suspects without revealing their entire private databases to one another. Traditional methods, such as sharing hashed lists, are vulnerable to dictionary and brute-force attacks, particularly when the input domain (e.g., email addresses or phone numbers) is small. This paper implements a solution to this problem using a Private Set Intersection (PSI) protocol based on the Diffie-Hellman key exchange. By leveraging the property of commutative encryption, our system allows two parties to compute the intersection of their datasets where neither party can determine the other's non-shared elements. We present a Python-based Proof of Concept (PoC) that simulates a secure handshake between two organizations. The results demonstrate that the protocol effectively identifies shared records while maintaining computational feasibility for small-to-medium datasets, providing a secure alternative to naive data sharing.**

***Index Terms*—Private Set Intersection, Cryptography, Diffie-Hellman, Commutative Encryption, Data Privacy.**

I. Introduction

In the modern digital ecosystem, data collaboration between organizations is often necessary for critical business functions such as fraud detection, contact tracing, and employee background checks. For instance, two financial institutions may wish to identify a list of shared money launderers without disclosing their legitimate client bases to one another. Similarly, companies may need to verify if a potential hire is currently employed by a competitor without revealing their full employee roster.

However, this need for collaboration directly conflicts with increasingly stringent data privacy regulations, such as the General Data Protection Regulation (GDPR) in Europe and the Personal Data Protection (PDP) Law in Indonesia. These regulations mandate that Personally Identifiable Information (PII) must not be shared without explicit consent or a valid legal basis.

A common but flawed approach to this problem is "naive hashing." In this scenario, organizations agree on a hash function (e.g., SHA-256) and exchange the hashes of their records. While the raw data is not exposed, this method is vulnerable to Dictionary Attacks. Since the input domain of PII (such as email addresses or National ID numbers) is relatively small and structured, a malicious party can pre-compute the hashes of all possible inputs and compare them against the received list to reverse-engineer the private data.

To address this, we explore the concept of Private Set Intersection (PSI). PSI is a cryptographic protocol that allows two parties, a sender and a receiver, to compute the intersection of their private sets such that the receiver learns only the elements in the intersection, and the sender learns nothing.

In this paper, we implement a PSI protocol based on the Diffie-Hellman key exchange. This method utilizes the property of commutative encryption, allowing two parties to encrypt data in any order and arrive at the same result. This ensures that the intersection can be determined on double-encrypted data, which appears pseudo-random to both parties, thereby thwarting dictionary attacks.

The contributions of this paper are as follows:

1) We design a PSI architecture using modular exponentiation on a large prime field.
2) We implement a proof-of-concept in Python that simulates the multi-round handshake required for the protocol.
3) We analyze the security of the system against honest-but-curious adversaries and demonstrate why it offers superior privacy compared to simple hash exchange.

II. Theoretical Foundation

A. Private Set Intersection (PSI)

Private Set Intersection (PSI) is a secure multi-party computation technique. Formally, consider two parties, Alice and Bob, holding private datasets S_A and S_B respectively. The goal of a PSI protocol is to allow one or both parties to compute the intersection $I = S_A \cap S_B$, without revealing any element x such that $x \notin I$.

An ideal PSI protocol must satisfy two properties:

- **Correctness:** The output must strictly be the true set intersection.

- **Privacy:** Neither party should learn information about the cardinality or content of the other party's set beyond what can be inferred from the intersection itself.

B. Commutative Encryption

Our implementation relies on a specific class of cryptographic functions known as Commutative Encryption. A cryptosystem is commutative if the order of encryption and decryption (or multiple encryptions) does not affect the final result.

Mathematically, let E be an encryption function and D be a decryption function. For any message m and distinct keys k_1, k_2, the system satisfies:

$$E_{k_1}(E_{k_2}(m)) = E_{k_2}(E_{k_1}(m)) \tag{1}$$

This property is crucial for PSI because it allows Alice and Bob to lock data with their own keys successively. Alice locks her data with key a, then Bob locks it with key b. The result is data locked by both (ab). If Bob starts first (locks with b, then Alice locks with a), the result is the same (ba). Since $ab = ba$, the final ciphertexts can be compared directly.

C. Diffie-Hellman and Modular Exponentiation

To implement commutative encryption practically, we utilize the multiplicative group of integers modulo a large prime p, consistent with the Diffie-Hellman key exchange protocol.

Given a large prime p, the encryption function for a message x (where x is mapped to an integer in the group) and a private key k is defined as modular exponentiation:

$$C = x^k \pmod p \tag{2}$$

This operation satisfies the commutative property. If Alice holds private key α and Bob holds private key β, the double encryption of an element x is:

$$(x^\alpha)^\beta \equiv x^{\alpha\cdot\beta} \equiv x^{\beta\cdot\alpha} \equiv (x^\beta)^\alpha \pmod p \tag{3}$$

The security of this scheme relies on the Discrete Logarithm Problem (DLP). Given g, p, and $g^k \pmod p$, it is computationally infeasible to determine the exponent k. This ensures that even if Alice sees Bob's encrypted set $\{x_1^\beta, x_2^\beta, \dots\}$, she cannot remove the encryption exponent β to recover the original inputs x_i without knowing β.

D. Cryptographic Parameter Selection

The security of the Diffie-Hellman protocol is not guaranteed solely by the use of large numbers; it strictly depends on the algebraic structure of the underlying group. A critical vulnerability in naive implementations is the **Small Subgroup Attack**.

1) Small Subgroup Confinement: The security of the discrete logarithm relies on the order of the multiplicative group $\mathbb{Z}_p^*$, which is $p-1$. If $p-1$ is a smooth number (i.e., it factors entirely into small primes $p_1, p_2, \dots, p_k$), an adversary can utilize the **Pohlig-Hellman algorithm**.

The Pohlig-Hellman algorithm reduces the problem of computing discrete logarithms in a group of order N to computing them in groups of order $p_i^{e_i}$ for each prime factor of N. If these factors are small, the total computational effort becomes trivial, effectively breaking the encryption regardless of the size of p.

2) Safe Primes: To mitigate this, our protocol enforces the use of a **Safe Prime**. A safe prime is a prime number p of the form:

$$p = 2q + 1 \tag{4}$$

where q is also a large prime (known as a Sophie Germain prime). In this configuration, the order of the group is $p - 1 = 2q$. The only subgroups of $\mathbb{Z}_p^*$ are of order:

- 1 (Identity element)
- 2 (The element -1)
- q (The large subgroup of quadratic residues)
- $2q$ (The full group)

By validating that the public keys utilized in the protocol are not 1 or $p-1$, we ensure that any operations occur within the subgroup of order q. Since q is a large prime ($\approx 2^{2047}$), the Pohlig-Hellman attack provides no advantage over standard generic algorithms like Pollard's Rho, preserving the full $O(\sqrt{p})$ security margin. This aligns with the specifications of **RFC 3526 Group 14**, which provides the parameters utilized in our implementation.

III. System Design

A. System Architecture

The proposed system operates on a peer-to-peer architecture involving two distinct entities:

1) **Alice (Initiator):** Holds a private dataset S_A. Alice wishes to find which of her elements exist in Bob's dataset without revealing the non-matching elements.
2) **Bob (Peer):** Holds a private dataset S_B. Bob cooperates in the calculation but follows a strict protocol to ensure data confidentiality.

Both parties agree beforehand on a large 2048-bit safe prime p to define the finite field $\mathbb{F}_p$ for cryptographic operations.

B. Protocol Workflow

The execution of the protocol follows a four-step process, as illustrated in Fig. 1.

C. Formal Algorithm Definition

The complete procedure for the Private Set Intersection handshake is defined in Algorithm 1. This formalization highlights the independent operations performed by the Initiator (Alice) and the Peer (Bob).

Algorithm 1: Diffie-Hellman PSI Protocol

```
Require: Prime $p$, Hash Function $H(\cdot)$
Require: $S_A$ (Alice's Set), $S_B$ (Bob's Set)
Ensure: $I = S_A \cap S_B$
 1: Setup:
 2: Alice generates private key $a \in_R [2, p-2]$
 3: Bob generates private key $b \in_R [2, p-2]$
 4: Round 1: Initial Encryption
 5: for all $x_i \in S_A$ do
 6:     $h_i \leftarrow H(x_i) \pmod p$
 7:     $A'[i] \leftarrow (h_i)^a \pmod p$
 8: end for
 9: Alice sends $A'$ to Bob.
10: for all $y_j \in S_B$ do
11:     $h_j \leftarrow H(y_j) \pmod p$
12:     $B'[j] \leftarrow (h_j)^b \pmod p$
13: end for
14: Bob sends $B'$ to Alice.
15: Round 2: Double Encryption
16: // Bob processes Alice's set
17: for all $val \in A'$ do
18:     $A''[k] \leftarrow (val)^b \pmod p$
19: end for
20: Bob sends $A''$ to Alice.
21: // Alice processes Bob's set
22: for all $val \in B'$ do
23:     $B''[k] \leftarrow (val)^a \pmod p$
24: end for
25: Intersection Calculation (Alice)
26: $I \leftarrow \emptyset$
27: for all $z \in A''$ do
28:     if $z \in B''$ then
29:         // Map encrypted $z$ back to original $x$
30:         $x \leftarrow \text{Lookup}(z, S_A)$
31:         $I \leftarrow I \cup \{x\}$
32:     end if
33: end for
34: return $I$
```

1) Data Sanitization and Hashing: Before encryption, raw identifiers (e.g., email addresses) are normalized (lowercased/trimmed) and hashed using SHA-256. The hexadecimal output is converted to an integer modulo p. This step maps the variable-length strings to group elements in $\mathbb{F}_p$.

$$h_x = \text{SHA256}(x) \pmod p \tag{5}$$

2) Round 1: Initial Encryption: Each party generates a private key (a for Alice, b for Bob). They independently encrypt their hashed sets using modular exponentiation:

- Alice computes $A' = \{h_x^a \pmod p \mid x \in S_A\}$ and sends A' to Bob.
- Bob computes $B' = \{h_y^b \pmod p \mid y \in S_B\}$ and sends B' to Alice.

3) Round 2: Double Encryption: Upon receiving the counterparty's set, each party applies their own private key to the already encrypted data.

- Bob computes $A'' = \{(h_x^a)^b \pmod p\} = \{h_x^{ab} \pmod p\}$ and returns it to Alice.
- Alice computes $B'' = \{(h_y^b)^a \pmod p\} = \{h_y^{ba} \pmod p\}$.

4) Intersection Analysis: Due to the commutative property of the encryption ($h^{ab} \equiv h^{ba}$), Alice now holds two lists of double-encrypted values. She computes the intersection:

$$I_{enc} = A'' \cap B'' \tag{6}$$

Any match found in I_{enc} corresponds to a shared element. Since Alice retains the index mapping of her original list S_A, she can map the matching encrypted value back to the original cleartext record.

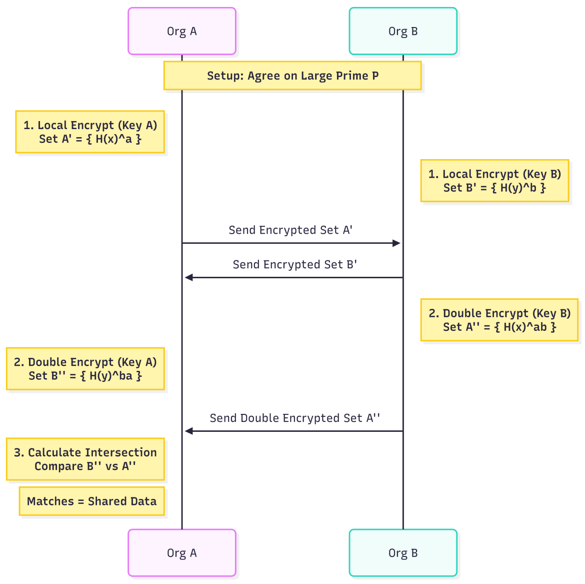

Fig. 1. Sequence diagram illustrating the Diffie-Hellman PSI exchange. Note that neither party ever reveals their raw keys or unencrypted data.

IV. Implementation and Results

A. Implementation Environment

The Proof of Concept (PoC) was developed in **Python 3.12**. We utilized the standard library `hashlib` for SHA-256 hashing and `secrets` for generating cryptographically secure random numbers (CSPRNG).

To simulate a realistic secure environment, the system was configured to use the 2048-bit Modular Exponential (MODP) Diffie-Hellman group defined in **RFC 3526**. The simulation architecture separates the "Alice" and "Bob" entities into distinct objects that cannot access each other's internal memory, enforcing the network constraints of a real-world deployment.

B. Functional Verification

We conducted a functional test using a dataset of employee records stored in JSON format. The datasets were constructed with a known intersection:

- **Org A (Alice):** Includes unique employees and one shared target: `sarah@shared.com`.
- **Org B (Bob):** Includes unique employees and the same shared target: `sarah@shared.com`.

Upon execution, the protocol successfully identified the intersection. As shown in the system logs, Org A received the double-encrypted set from Org B and compared it against its own double-encrypted values. The matching index was successfully mapped back to the original local record.

C. Privacy and Security Analysis

The primary goal of this system is to ensure that non-shared elements remain confidential. To verify this, we inspected the data payloads exchanged during ”Round 2” (Double Encryption).

Table I demonstrates the view from Org A's perspective. While Org A possesses the double-encrypted value of Org B's unique employee (*bob@partner.org*), this value appears as a pseudo-random integer.

TABLE I
DATA VISIBILITY ANALYSIS (ORG A'S PERSPECTIVE)

Data Element	**State**	**Org A's View**
sarah@shared.com	Shared	**MATCH FOUND** (Maps to local ID 102)
bob@partner.org	Unique to B	`0x92f8a1...` (Garbage)

Because the value `0x92f8a1...` is the result of $H(\text{email})^{ba} \pmod p$, Org A cannot reverse this to find the original email without knowing Org B's private key b. This confirms that the protocol satisfies the privacy requirement.

D. Performance Observations

The use of 2048-bit modular exponentiation introduces a computational cost compared to simple hashing. To evaluate the scalability, we simulated the protocol execution across varying dataset sizes ranging from $N = 100$ to $N = 10,000$.

As illustrated in Fig. 2, although the DH-PSI approach is computationally more expensive than naive hashing, the time complexity remains linear, $O(N)$. In our tests, processing a batch of 5,000 records took approximately 99.4 seconds. This confirms that while the method trades raw speed for security, the overhead is well within acceptable limits for batch processing applications (e.g., nightly reconciliation jobs).

V. DISCUSSION

In this section, we analyze the security properties of the proposed Diffie-Hellman PSI protocol, specifically contrasting it with the ”Naive Hashing” approach commonly used in the industry.

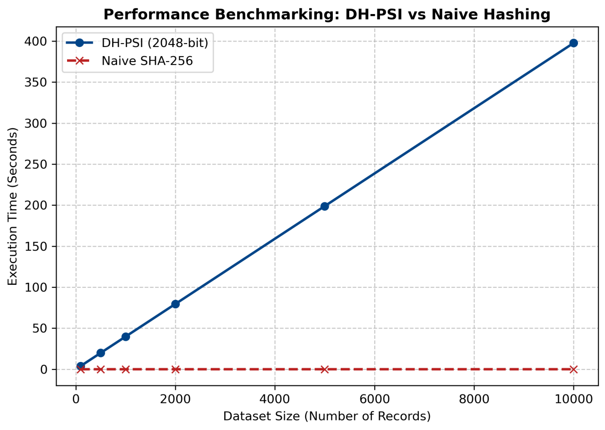

Fig. 2. Performance scalability comparison. While DH-PSI (blue line) introduces computational overhead compared to naive hashing (red dashed line), it maintains linear scalability $O(N)$.

A. The Vulnerability of Naive Hashing

The ”Naive Hashing” method involves two parties agreeing on a cryptographic hash function $H(\cdot)$ (e.g., SHA-256) and exchanging the list of hashed records: $L = \{H(x) \mid x \in S\}$.

While this obscures the plaintext, it is fundamentally insecure for datasets with low-entropy domains, such as email addresses, phone numbers, or national ID numbers. An adversary (or even the counterparty) can launch a Dictionary Attack or Brute-Force Attack:

1) The adversary generates a list of all probable inputs (e.g., all valid email formats or phone numbers).
2) They compute the hash for every probable input.
3) They compare these computed hashes against the received list L.

Because hash functions are deterministic and unkeyed, anyone can verify a guess. If the domain size is manageable (e.g., 10^8 possible phone numbers), the privacy protection collapses entirely.

B. Security of Diffie-Hellman PSI

The Diffie-Hellman PSI protocol mitigates the dictionary attack vulnerability by introducing a secret key into the transformation process. The exchanged data is not merely hashed, but exponentiated: $C = H(x)^k \pmod p$.

This structure provides two critical security layers:

1) Keyed Transformation: Unlike a standard hash, the transformation $F_k(x) = x^k \pmod p$ depends on a secret scalar k kept private by the sender. An adversary who intercepts the list $\{H(x)^a\}$ cannot perform a dictionary attack because they lack the key a. They cannot compute $(x_{guess})^a$ to check if it matches the intercepted value. To the adversary, the list appears as a sequence of random elements in the group $\mathbb{G}$.

2) The Discrete Logarithm Hardness: The security relies on the Computational Diffie-Hellman (CDH) assumption. Even if an adversary knows the base $H(x)$ (a guess) and the result $H(x)^a$ (from the list), recovering the secret key a requires solving the Discrete Logarithm Problem (DLP), which is computationally infeasible for a sufficiently large prime p (e.g., 2048-bit).

C. Comparative Summary

Table II summarizes the security distinctions between the two approaches. The Diffie-Hellman PSI approach effectively trades computational overhead for resistance against pre-computation attacks.

TABLE II
SECURITY COMPARISON: NAIVE HASHING VS. DH-PSI

Feature	**Naive Hashing**	**DH-PSI (Ours)**
Transformation	Deterministic ($H(x)$)	Keyed ($H(x)^k$)
Dict. Attack	**Vulnerable**	**Resistant**
Rainbow Table	Effective	Ineffective
Privacy	Low (computational)	High (cryptographic)

D. Adversarial Models and Mitigations

Our current security analysis assumes the **Honest-but-Curious (HbC)** adversary model. In this model, parties follow the protocol specification correctly—they do not alter the control flow or inject malformed packets—but they attempt to learn as much information as possible from the legitimate transcript. However, real-world deployment requires considering **Malicious Adversaries**.

1) The Cardinality/Dictionary Attack: A malicious participant (e.g., Bob) could attempt to harvest Alice's private data by deviating from the protocol. Instead of encrypting his legitimate dataset of size N, Bob could generate a "brute-force dictionary" containing millions of possible email addresses (e.g., $N' \gg N$).

$$S_{Bob}^{Malicious} = \{\text{guess}_1, \text{guess}_2, \ldots, \text{guess}_{10^7}\} \tag{7}$$

If Alice blindly processes this oversized set, she will perform the double-encryption step on all 10^7 items and return the intersection. Bob effectively checks Alice's private set against his massive dictionary, violating the privacy expectations.

2) Mitigation Strategies: To defend against malicious adversaries, two extensions to the protocol are necessary:

1) **Set Size Restrictions:** The simplest defense is enforcing a hard limit on the number of items exchanged. If the parties agree that $N = 10,000$, Alice should abort the protocol if Bob sends $10,001$ items.
2) **Zero-Knowledge Proofs (ZKP):** A robust defense involves Bob providing a Zero-Knowledge Proof that his input set is well-formed and generated from a valid, committed database. While computationally expensive, this ensures that the inputs are not dynamically generated dictionary guesses.
3) **Circuit-Based PSI:** Moving to Generic Multi-Party Computation (MPC) frameworks (such as Garbled Circuits) allows the intersection logic to be evaluated such that neither party sees the intermediate encrypted values, strictly revealing *only* the final result.

Future iterations of this project will explore implementing a basic **Cardinality Check** mechanism to reject payloads that exceed a standard deviation of the expected dataset size.

VI. FUTURE WORK

While the current implementation provides a robust Proof of Concept for privacy-preserving collaboration, several avenues exist for optimization and hardening in a production environment.

A. Elliptic Curve Cryptography (ECC)

The current implementation utilizes Modular Exponentiation over a 2048-bit field. While secure, this results in relatively large keys and ciphertexts (256 bytes per record). Future iterations of this protocol should transition to **Elliptic Curve Diffie-Hellman (ECDH)** using the `secp256r1` or `Curve25519` curves. In the ECC setting, the commutative property holds under scalar multiplication:

$$b \cdot (a \cdot P) = (ba) \cdot P = (ab) \cdot P = a \cdot (b \cdot P) \tag{8}$$

ECC would reduce the payload size by approximately 87% (from 2048 bits to 256 bits per record), significantly lowering bandwidth costs for large datasets.

B. Bloom Filters for Bandwidth Optimization

To further reduce communication overhead, the protocol could integrate **Bloom Filters**. Instead of transmitting the full list of encrypted items A', Alice could insert her encrypted elements into a Bloom Filter and transmit the bit array. Bob would then query his encrypted elements against this filter. While this introduces a small probability of False Positives, it can reduce the communication complexity from $O(N)$ integers to a constant number of bits per element (approx. 10 bits per item for a 1% error rate), offering a dramatic speedup for massive datasets ($N > 10^6$).

C. Post-Quantum Security

It is well-established that Shor's Algorithm, when run on a sufficiently powerful quantum computer, can solve the Discrete Logarithm Problem in polynomial time, rendering both standard Diffie-Hellman and ECC insecure. To ensure long-term privacy (Forward Secrecy), future work must explore **Post-Quantum Cryptography (PQC)**. Lattice-based PSI protocols, such as those based on the Learning With Errors (LWE) problem, offer resistance to quantum adversaries. Migrating this system to use a lattice-based commutative encryption scheme (e.g., NTRU-based) would future-proof the application against emerging quantum threats.

VII. CONCLUSION

This paper presented the design and implementation of a Private Set Intersection (PSI) protocol based on the Diffie-Hellman key exchange. In an era where data privacy regulations like GDPR and the PDP Law strictly limit inter-organizational data sharing, our solution provides a mathematically secure method to identify common records without exposing sensitive non-shared data.

Our functional analysis confirmed that the commutative property of modular exponentiation allows for accurate intersection detection on double-encrypted data. Security analysis demonstrated that, unlike Naive Hashing, our approach is resistant to dictionary and brute-force attacks due to the reliance on the Discrete Logarithm Problem.

Performance benchmarks quantified the cost of this privacy: the cryptographic operations are approximately 2.3×10^5 times slower than standard hashing. However, with linear scalability $O(N)$, the system remains practical for real-world batch applications where data confidentiality is paramount. Future work could optimize this further using Elliptic Curve Cryptography (ECC) to reduce the computational overhead while maintaining equivalent security levels.

REFERENCES

[1] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644-654, 1976.
[2] M. J. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2004, pp. 1-19.
[3] R. Munir, "Algoritma Diffie-Hellman," Course Slides IF4020 Kriptografi, Institut Teknologi Bandung, 2025. [Online]. Available: https://informatika.stei.itb.ac.id/ rinaldi.munir/Kriptografi/2025-2026/22-Algoritma-Diffie-Hellman-2025.pdf

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 24 Desember 2025

[illegible]

Abdul Rafi Radityo Hutomo
NIM 13522089