Journal of Advanced Research in Applied Sciences and Engineering Technology

Journal homepage:

https://semarakilmujournal.com.my/index.php/araset
ISSN: 2462-1943

A Tree-Based FHE Approach for Private Set Intersection

Akshit Aggarwal[1,*]

[1] Indian Institute of Information Technology Guwahati, Assam India, 781015, India

ARTICLE INFO

Article history:
Received 2 March 2026
Received in revised form 7 March 2026
Accepted 15 March 2026
Available online 7 April 2026

Keywords:

Digit; FHE; Number; PSI; Tree

ABSTRACT

Private Set Intersection (PSI) allows two parties to compute the intersection of their datasets without revealing any additional information. Fully Homomorphic Encryption (FHE) based PSI schemes offer strong security guarantees but typically rely on hashing or oblivious transfer, resulting in two major limitations: (1) communication complexity linear in the size of client dataset and logarithmic in server dataset, that is, $O(|X|\log(|Y|))$, and (2) high memory usage when encoding large multi-digit numbers using SIMD packing. In this work, we propose a tree-structured representation for storing multi-digit numbers under FHE. Instead of packing entire numbers into slots, each digit is stored at a different depth of the tree, allowing homomorphic comparison to be performed digit-wise. This significantly reduces the dependency on dataset size, and communication complexity to $O(|d|^2)$, where $|d|$ is the number of digits in each number. We implement our protocol using TenSEAL library and experimentally demonstrate reductions in communication and memory usage. Our design introduces a new direction for performing PSI over encrypted datasets containing large numeric values.

1. Introduction

Private set intersection allows two parties to identify common elements between their datasets without revealing any non-matching items. Suppose two parties, P_1 and P_2, hold datasets X and Y. A PSI protocol lets them learn only $X \cap Y$ while hiding everything else. This capability is critical in many privacy-sensitive applications. For example, a financial institution may wish to verify whether a transaction appears in a blacklist of fraudulent activities without disclosing transaction details or other entries in blacklist. Similarly, a logistics company may want to confirm whether a shipment belongs to an approved distributor without exposing other sensitive shipping records.

To address such problems, PSI-based protocols enable parties to compare their private datasets without revealing any additional information. The financial institution can securely determine whether a transaction is flagged as fraudulent without accessing the entire blacklist, while the logistics company can verify an approved distributor without exposing other shipment details. We can divide these PSI protocols in two categories, that is, one-way PSI, and double-way PSI. In one-

* *Corresponding author.*
E-mail address: akshitaggarwal20jan@gmail.com

https://doi.org/10.37934/araset.58.5.2738

way PSI, one party works as a sender and another as receiver (that is responsible for generating output). Whereas, in double-way PSI, both parties are responsible for generating the output. In practice, one-way PSI is widely used in privacy preserving applications such as private smart contract verification [1], private ride-sharing authentication [2,3], many more [4-9]. These PSI-based protocols have a significant limitation, that is, high communication overhead.

To overcome the above limitation, fully homomorphic encryption (FHE) based protocols are widely used to reduce communication overhead. FHE enables single instruction multiple data (SIMD) packing that allows to operate over multiple data elements simultaneously [10-13]. We can divide FHE based protocols in two types, that is, polynomial based FHE, and hash based FHE.

In polynomial based PSI, dataset is interpolated in the form of polynomial and common elements are computed by finding the roots of this polynomial. The achieved communication complexity is linear in the size of both participants sets. Thereafter, hash based approaches hashes the dataset into hash tables that significantly reduces communication complexity, that is, linear in the size of one participant's set and logarithmic in the size of another participant's set. However, these approaches increase computation overhead whenever dataset entries are large.

1.1 Literature Survey

To the best of our knowledge, Meadows et al. [14] first introduce the secure PSI protocol, which was later describe by Huberman et al. [15]. The approach propose in [15] leverages the multiplicative homomorphic property of Diffie-Hellman key exchange algorithm [16] but increases the running time of protocol due to many exponential operations. Thus, to reduce running time, several other paradigms (say, hash-based approach, and oblivious transfer [17-19] are evolved.

First, Chen et al. [20] hash their data in cuckoo hash table, and homomorphic evaluation is performed over the server. The main drawback of their protocol is high communication overhead when all elements are store in a single slot of cuckoo hash. To its improvement, Pinkas et al. [21] add k auxiliary positions in cuckoo hash and map these positions to single slot of cuckoo hash. The main drawback is server communication overhead due to k auxiliary positions. Thereafter, Ghosh et al. [22] propose polynomial-based approach where all the data are interpolated into polynomials, and oblivious operations (say polynomial-related operations) are applied. The main drawback of this approach is slow running time. Later on, Ghosh et al. [23] improve their own polynomial approach by considering the minimum number of common elements between client and server (say t). The authors try to find the root of this polynomial to obtain the common elements. The main drawback of this work is being unable to find the common elements when number of common elements is less than t. Chakraborti et al. [24] extend the work of [23] when the common elements between client and server are less than t. The authors compute hamming distance between the sets, whenever the distance is 0, the element is common. Thereafter, Hu et al. [25] encode their data in Bloom filter and evaluate a polynomial over it. The main drawback of their work is high communication complexity. Later on, Ghosh et al. [26] recursively divide the datasets into smaller chunks and apply the symmetric set difference operations (say X-Y U Y-X). The proposed work in [26] is again not applicable whenever the intersection size between the two sets are less than t. Kubmaul et al. [27] recursively store the elements into cuckoo hash table and later apply the homomorphic subtraction operations. The main drawback of their work is high computation overhead in client and server side whenever the entries of dataset are consider as large.

Thus, while prior FHE-based PSI protocols aim to reduce communication overhead, they still suffer from key practical limitations, including (i) increased computation and memory cost for large numeric entries, (ii) reliance on threshold parameters that may leak structural information, and (iii)

inefficiencies in handling structured multi-digit data. These observations motivate the need for a more structured analysis of existing limitations, which we formalize as research gaps in the next subsection.

1.2 Research Gaps

Despite significant progress in PSI protocols, existing FHE based solutions still face practical limitations. In particular, existing approaches rely on hashing or polynomial interpolation, which introduce either (i) communication complexity dependent on dataset size, or (ii) high computation and memory overhead when handling large multi-digit numerical data. Moreover, existing methods are not well suited for scenarios where dataset elements are structured numeric values (say, mobile numbers), as SIMD packing becomes inefficient and leads to increased noise and resource consumption.

Therefore, there exists a clear research gap in designing a PSI protocol that (i) reduces dependency on dataset size, (ii) efficiently handles multi-digit numeric data without heavy packing, and (iii) maintains low communication and memory overhead in practical deployments.

1.3 Our Solution and Contribution

In this work, we propose a new PSI approach that avoids polynomial interpolation and hashing, and instead organizes all d-digit numbers stored on the server into a hierarchical tree. Each level of the tree corresponds to a digit position: all first digits are stored at Level 1, all second digits at Level 2, and so on up to Level d. Every digit at one level is linked to its corresponding digit at the next level, preserving internal structure of each number. Using this representation, client encrypts its query digit-by-digit and performs a homomorphic search across levels to determine whether the number exists on the server.

The key contribution of our work is as follows.

1. We introduce a novel tree-structured organization for storing all d-digit numbers (where X U Y = N), which reduces communication complexity by $O(|d|^2)$. This is particularly effective because d s typically much smaller than the dataset sizes X and Y.
2. We implement the complete protocol using TenSEAL library and demonstrate that client can determine whether its number exists on server without learning anything beyond the result.

1.3 Organization of our paper

In Section 2, we discuss the preliminaries used in this work. The proposed methodology of this work is discuss in Section 3. Section 4 discusses implementation setup and results. Later, Section 5 discusses limitations of our work. Finally, we present the conclusion of our work in Section 6.

2. Preliminary

In this section, we discuss the encryption scheme that is used in this work, that is, BFV homomorphic encryption.

2.1 BFV Homomorphic Encryption

The BFV encryption scheme facilitates simultaneous operations on encrypted data vectors and utilizes Single Instruction Multiple Data (SIMD) model. During encryption, BFV introduces noise whenever plaintext is transformed into ciphertext (where plaintext is an element of Z_p, with p being a prime number). This noise grows progressively as homomorphic operations are applied to the ciphertexts [28].

BFV encryption supports the following operations for processing plaintexts and ciphertexts while ensuring that computations are performed modulo p.

1. **Add**(c_1, c_2)): Takes two ciphertexts c_1 and c_2, which encrypt plaintexts m_1 and m_2, and returns a ciphertext encrypting (m_1 + m_2) mod p (element-wise addition).
2. **AddPlain**(c_1, m_2): Takes a ciphertext c_1 (encrypting m_1) and a plaintext m_2, and returns a ciphertext encrypting (m_1 + m_2) mod p (element-wise addition).
3. **Sub**(c_1, c_2): Takes two ciphertexts c_1 and c_2, which encrypt plaintexts m_1 and m_2, and returns a ciphertext encrypting (m_1 - m_2) mod p (element-wise subtraction).
4. **SubPlain**(c_1, m_2): Takes a ciphertext c_1 (encrypting m_1) and a plaintext m_2, and returns a ciphertext encrypting (m_1 - m_2) mod p (element-wise subtraction).
5. **Mul**(c1, c2): Takes two ciphertexts c_1 and c_2, which encrypt plaintexts m_1 and m_2, and returns a ciphertext encrypting (m_1.m_2) mod p (element-wise multiplication).
6. **MulPlain**(c_1, m_2): Takes a ciphertext c_1 (encrypting m_1) and a plaintext m_2, and returns a ciphertext encrypting (m_1.m_2) mod p (element-wise multiplication).

3. Proposed Methodology

We propose a two phase privacy-preserving method for determining whether a client specified d-digit number exists in a large server side database. The system assumes a semi-honest server, and privacy is ensured using FHE. The key idea is to convert all numbers into digit-wise hierarchical representations and organize them using a tree structure (as shown in Figure 1). The tree enables level wise comparison of digits, allowing server to operate on encrypted queries without learning any information about the client's input.

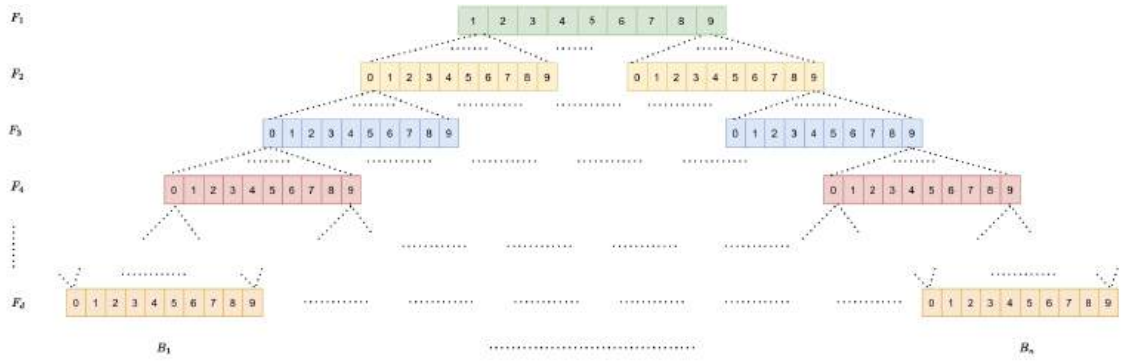

Fig. 1. Here F_i (where 1 <= i <= d) represents the Level i of tree or digits of numbers (meaning F_1 is first digit of all numbers whereas F_d is the d^{th}-digit of number). B_x (where 1 <= x <= n, n >= 1) represents the numbers of child nodes present at Level i. Dotted `Black' line represents the connection between the levels or digits.

3.1 Design of Server

In this phase, all d-digit numbers are stored on the server using tree-based approach. The first digit of all numbers are stored at Level 1 of the tree. Similarly, the second digit is stored at Level 2, and this process continues until the d^{th} digit, which is stored at Level d. Each digit at a given level is linked to its corresponding digit at the next level, preserving the hierarchical structure and relationships across the levels (as shown in Algorithm 1). The above observation is illustrate with following toy example.

Algorithm 1: Tree-Based Storage of d-Digit Numbers

Input: List of d-digit numbers: $\mathcal{N} = \{N_1, N_2, \ldots, N_n\}$
Output: Tree-like structure where each level stores digits and maintains links to the next level

Initialize: Create a root structure $\mathcal{T}$ to represent Level 1 of the tree.
Each node in the tree represents a digit and contains pointers to its child nodes.
foreach $N_i \in \mathcal{N}$ **do**
  Extract Digits: Let $N_i = d_1 d_2 \cdots d_k$, where d_k is the k^{th} digit.
  Insert into Tree:
  Set `CurrentNode` $\leftarrow \mathcal{T}$; `// Start from the root`
  for $i = 1$ *to* k **do**
    if d_i *is not a child of* `CurrentNode` **then**
      Add d_i as a child of `CurrentNode`
    `CurrentNode` $\leftarrow$ Child node corresponding to d_i ; `// Move to the next level`
Return: $\mathcal{T}$; `// The constructed tree structure`

Example 1

1. The server contains five 3-digit numbers (say 212, 221, 231, 312, 321).
2. At Level 1 of the tree:
   2.1 Digits 2 and 3 are stored.
3. At Level 2 of the tree:
   3.1 Digits 1, 2, and 3 are stored and are connected to digit 2 from Level 1.
   3.2 Digits 1 and 2 are stored and are connected to digit 3 from Level 1.
4. At Level 3 of the tree:
   4.1 Digits $2, 1, 1, 2$, and 1 are stored.
   4.2 These digits are connected to corresponding digits at Level 2.
5. The hierarchical structure and connections across levels are shown in Figure 2.

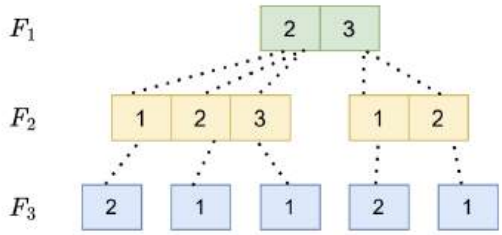

Fig. 2. Here F_i (where 1 <= i <= 3) represents the Level i of tree. Also, F_i represents the digit number (meaning F_1 is first digit, F_2 is second digit, and F_3 is third digit). Dotted `Black' line represents the connection between the levels or digits

3.2 Retrieving common elements

This phase is divided into three steps, that is, encryption, evaluation of server, and decryption. In encryption phase, client first encrypts the number digit by digit (say $Enc(d_i)$) and sends it to the server. Server homomorphically subtracts the number from digits (say d_j) at each level (say $Enc(d_i-d_j).r$), where $r>0$ is random number to maintain the privacy of other digits) and returns the result to client. Client decrypts it, and if the decryption result is non-zero, then digit d_i is not present and searches for next number; otherwise, digit d_i is present and checks for next digit d_j. For checking d_j, apply homomorphic operation only for the connecting child node. If decryption result is non-zero in any of the levels, it signifies that number is not present, and check for the next number (as discuss in Algorithm 2). The below toy example illustrates the above observations.

Example 2

1. Client needs to check whether the number 231 is present on the server (by considering Figure 2).
2. Client encrypts the first digit, 2, using Enc(2) and sends it to the server.
3. Server homomorphically subtracts the first digit from all entries in Level F_1:
   Enc((2-2).r), Enc((3-2).r),
   and returns the encrypted results to the client.
4. Client decrypts the results (that is, 0, r). Since the result contains 0, it indicates that first digit 2 is present.
5. Client encrypts the second digit, 3, using Enc(3) and sends it to server.
6. Server homomorphically subtracts the second digit from all entries in Level F_2, that is, connected to digit 3 of Level F_2:
   Enc((1-3).r), Enc((2-3).r), Enc((3-3).r),
   and sends the encrypted results back to the client.
7. Client decrypts the results (that is, -2r, -r, 0). Since the result contains 0, it indicates that second digit 3 is present.
8. Client encrypts the third digit, 1, using Enc(1) and sends it to server.
9. Server homomorphically subtracts the third digit from all entries in Level F_3:
   Enc((1-1).r),
   and returns the encrypted result to the client.
10. Client decrypts the result (that is, 0). Since the result contains 0, it indicates that the third digit 1 is present.
11. Since all digits have been verified successfully, the number 231 is confirmed to be present on the server.

Algorithm 2: Homomorphic Search for Number Presence

Input: d-digit number $N = \{d_1, d_2, \ldots, d_d\}$, Encrypted server database $Enc(F)$
Output: Presence or Absence of N in the server

Initialization:
Client encrypts each digit d_i of N using $Enc(d_i)$
Send $Enc(d_1)$ to the server for verification at Level 1 (F_1)
for *each level* F_i $(i = 1, \ldots, d)$ **do**
    Server performs homomorphic subtraction for all digits at F_i that connected with F_{i-1}:
        $Enc((x - d_i).r)$ for each $x \in F_i$, where $r > 0$ is a random multiplier
    Server sends the encrypted results back to the client.
    Client decrypts the result:
        if *Decrypted result contains* 0 **then**
            Digit d_i is present. Proceed to next digit d_{i+1} in N.
            if $i < d$ **then**
                Send $Enc(d_{i+1})$ to the server for verification at Level F_{i+1} (child nodes).
    else
        Output: Number N is not present. Stop search.
Output: Number N is present.

3.3 Complexity Discussion

In this work, we perform a worst-case analysis of our protocol. Consider a d-digit number, where each digit is homomorphically search within a block at every level of the tree. Since the search is performed sequentially for d levels, and each level involves searching through d elements in the corresponding block, the overall complexity of the protocol is $O(|d|^2)$.

4. Performance Evaluation

In this section, we describe the implementation setup and its results. We then compare our protocol with the baseline protocol [20], which retrieves the common elements. Our proof of code is written in Tenseal library and publicly available at https://github.com/akshit-aggarwal/Ring-Ring-Who-s-There-A-Privacy-Preserving-Mobile-Number-Search/tree/main.

4.1 Implementation Setup

The proof-of-concept code for our observation is written in Jupyter notebook (python version 3.9.10) using TenSeal library [29]. Jupyter notebook is installed over Intel(R) Core(TM) i5-10500 CPU @ 3.10GHz 3.10 GHz. The installed memory is 8GB.

4.1.1 Data Collection and preprocessing

In this phase, d-digit numbers are collected from running python d-digit random number (where first digit is either 6, 7, 8 or 9) generation program (in our case, we consider 10-digit number that seems to be Indian mobile numbers). Thus, total of 15000 numbers are collected. The total execution time for generating d-digit random numbers are 0.03 seconds.

4.1.2 Homomorphic encryption details

Both protocols are implemented using BFV scheme in TenSEAL library, which supports exact arithmetic on encrypted integers. The default parameters are used, including plaintext modulus, which is approximately 2^{20}, and polynomial modulus degree, which is 2^{13} (that is used to control security and computation performance).

4.2 Implementation of our protocol

In this phase, we discuss encryption, decryption, and evaluation of client's query. The random value r is taken from private subset that has a range of $(1, 10^2)$. Afterwards, implementation of server is performed, where all the 10-digits numbers are stored in tree-like manner. The total average execution time taken by our server is 0.55 seconds. We execute our program for several iterations in which, at each iteration, the client contains various numbers. At each iteration, client can find the common elements. Table 1 indicates execution time for client to privately find the common element(s). The implementation results show that the execution time is high whenever client needs to search for more elements (as it searches number using linear scan).

Table 1
Execution time of the proposed protocol for different numbers of queries

Number of Search	**Execution Time (in sec(s))**
1	1.00
10	3.83
100	32.55
1000	310.86
10000	2897.54

4.3 Implementation of Baseline protocol

In this subsection, we present the implementation results of baseline protocol [20]. First, hashing phase is implemented using cuckoo hash functions to uniformly distribute inputs across bins, ensuring load balance for subsequent computation. Further, batching is applied to pack multiple plaintext elements into a single ciphertext, significantly reducing communication and computation overhead. The average execution time for hashing and batching are 0.48 seconds and 1.12 seconds. The random value r is taken from private subset that has a range of $(1, 10^2)$. Finally, we execute the baseline and the average execution time for verification of common elements are reported in Table 2.

Table 2
Execution time of the proposed protocol for different numbers of queries

Number of Search	**Execution Time (in sec(s))**
1	4.59
10	15.16
100	99.21
1000	523.13
10000	3137.50

Table 2 indicates that execution time is high when a client needs to verify more numbers.

4.4 Evaluation Metrics

In this subsection, we compare our work in terms of execution time. Our analysis demonstrates that a client requires less execution time compared to baseline protocol. Figure 3 demonstrates the comparison analysis.

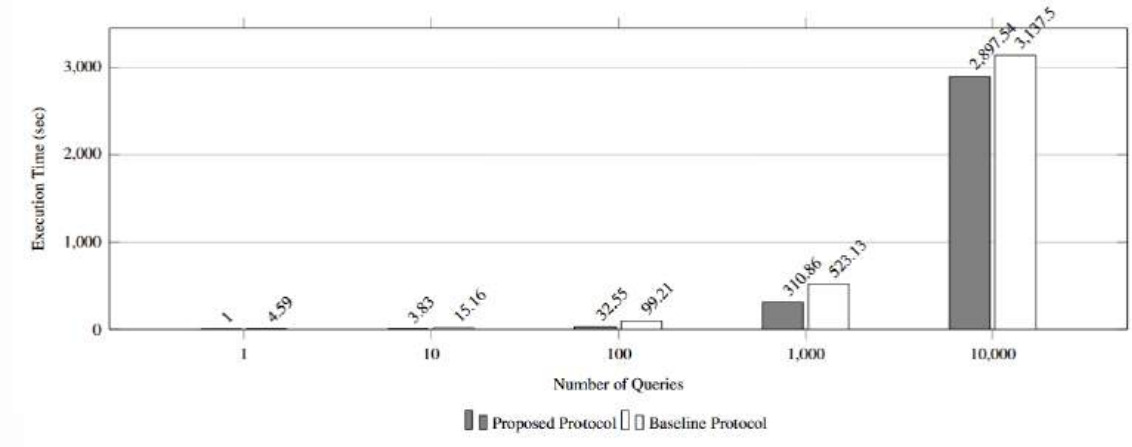

Fig. 3. Execution time comparison between proposed and baseline protocols

While the proposed approach demonstrates improved execution time compared to the baseline protocol, several challenges were observed during implementation. First, the tree based structure requires careful organization of data to preserve digit relationships, which introduces preprocessing overhead at server side. Second, the use of linear scan for digit matching increases latency when the number of queries grows significantly.

Compared to the baseline cuckoo hashing approach, our method avoids hashing overhead and reduces communication dependency on dataset size. However, this comes at the cost of sequential digit verification, which may impact scalability for large query volumes. These trade-offs highlight that while our approach is effective for structured numeric datasets, further optimizations are required for large-scale deployments.

4. Security Analysis

In this section, we analyze the security properties of the proposed protocol. We first outline the threat assumptions, then define the security objectives, and finally provide a formal proof sketch demonstrating query privacy under the assumed cryptographic hardness assumptions.

4.1 Threat Model

We consider an adversary A that can access the servers' stored data, monitor all client--server communications, and record both encrypted queries and responses. The adversary is assumed to be honest-but-curious, meaning it correctly executes the prescribed protocol steps without modifying computations or corrupting stored data. However, such an adversary may still attempt to infer sensitive information or generate incorrect outputs, thereby compromising integrity.

We further assume that A cannot break the underlying cryptographic primitives and, in particular, cannot violate the semantic security of the employed homomorphic encryption scheme.

4.2 Security Goal

The primary objective of our protocol is to ensure query privacy. Specifically, the adversary should learn no information about the client's query q or the corresponding retrieved data item. Formally, any two queries of the same length should be computationally indistinguishable from each other, even if all communications and ciphertexts are observable by the adversary.

4.3 Formal Proof of Security

We establish query privacy through a standard indistinguishability experiment defined between a challenger C and an adversary A.

The game, denoted $Game_0$, proceeds as follows:

1. The adversary chooses two distinct queries q_0 and q_1 of equal size and sends them to the challenger.
2. The challenger randomly selects a bit $\beta \in \{0,1\}$ and encrypts q_β using the BFV homomorphic encryption scheme, then executes the protocol honestly with q_β.
3. The adversary outputs a guess $\beta' \in \{0,1\}$, attempting to determine which query was used.

We now define a second game, $Game_1$, in which the challenger simulates the same steps but replaces the encrypted query with a ciphertext of a random query q_r sampled uniformly from the query space. Let E_1 denote the event that $\beta = \beta'$ in $Game_1$. Since q_r is chosen independently of q_0 and q_1, the adversary has no advantage other than random guessing, that is,

$$\Pr[E_1] = \frac{1}{2}.$$

In $Game_0$, the challenger encrypts q_β using the BFV scheme, which guarantees semantic security. Consequently, the adversary's distinguishing advantage between $Game_0$ and $Game_1$ is bounded by the negligible advantage ϵ_{BFV} of breaking the BFV encryption. Hence,

$$|\Pr[E_0] - \Pr[E_1]| \leq \epsilon_{BFV}$$

Substituting $\Pr[E_1] = \frac{1}{2}$ gives

$$\left|\Pr[E_0] - \frac{1}{2}\right| \leq \epsilon_{BFV}$$

Thus, the adversary cannot distinguish which query was issued except with negligible probability, ensuring that the proposed protocol satisfies query privacy under the semantic security of the BFV homomorphic encryption scheme.

5. Limitation

The proposed approach relies on a linear scan mechanism for digit wise homomorphic comparison, which increases computation time as the number of queries grows. Additionally, when the number of digits becomes large, the efficiency of the protocol may decrease due to repeated sequential comparisons. The tree construction phase also introduces preprocessing overhead at the server side, which may impact scalability in large-scale deployments.

6. Conclusion and Future Work

In this work, we presented a privacy-preserving protocol for checking the presence of multi-digit numbers using a tree based structure under FHE. The proposed approach enables digit wise verification without relying on SIMD packing, thereby reducing communication dependency on dataset size. However, the proposed method introduces certain trade-offs. In particular, the use of linear scan for digit verification increases computation time as the number of queries grows. Additionally, the tree construction phase introduces preprocessing overhead on the server side. Despite these limitations, the approach remains effective for structured numeric datasets where traditional packing based methods are inefficient. Overall, the proposed technique provides a promising direction for privacy-preserving search over encrypted structured data.

In future, improving scalability of proposed approach by replacing linear scan with more efficient search mechanisms. Additionally, extending the protocol to support batch queries and parallel processing can further reduce execution time. Another promising direction is integrating the proposed structure with hybrid PSI frameworks to balance communication and computation overhead.

References

[1] Angelou, N., A. Benaissa, B. Cebere, W. Clark, A. J. Hall, M. A. Hoeh, D. Liu, et al. 2020. "Asymmetric Private Set Intersection with Applications to Contact Tracing and Private Vertical Federated Machine Learning." *arXiv preprint* arXiv:2011.09350. https://arxiv.org/abs/2011.09350.

[2] Mohanty, T., V. Srivastava, S. K. Debnath, A. K. Das, and B. Sikdar. 2023. "Quantum Secure Threshold Private Set Intersection Protocol for IoT-Enabled Privacy Preserving Ride-Sharing Application." *IEEE Internet of Things Journal*.

[3] Li, M., Y. Chen, C. Lal, M. Conti, F. Martinelli, and M. Alazab. 2022. "Nereus: Anonymous and Secure Ride-Hailing Service Based on Private Smart Contract." *IEEE Transactions on Dependable and Secure Computing* 20 (4): 2849–2866.

[4] Liu, B., X. Zhang, R. Shi, M. Zhang, and G. Zhang. 2022. "SEPSI: A Secure and Efficient Privacy-Preserving Set Intersection with Identity Authentication in IoT." *Mathematics* 10 (12): 2120.

[5] Hellwig, D. P., and A. Huchzermeier. 2022. "Distributed Ledger Technology and Fully Homomorphic Encryption: Next-Generation Information-Sharing for Supply Chain Efficiency." In *Innovative Technology at the Interface of Finance and Operations: Volume II*, 31–49. Springer.

[6] Shen, L., X. Chen, D. Wang, B. Fang, and Y. Dong. 2018. "Efficient and Private Set Intersection of Human Genomes." In *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 761–764. IEEE.

[7] Zheng, Y., R. Lu, H. Zhu, S. Zhang, Y. Guan, J. Shao, F. Wang, and H. Li. 2022. "SetRkNN: Efficient and Privacy-Preserving Set Reverse kNN Query in Cloud." *IEEE Transactions on Information Forensics and Security* 18: 888–903.

[8] Hu, J., Y. Zhao, B. H. M. Tan, K. M. M. Aung, and H. Wang. 2024. "Enabling Threshold Functionality for Private Set Intersection Protocols in Cloud Computing." *IEEE Transactions on Information Forensics and Security*.

[9] Qian, Y., J. Shen, P. Vijayakumar, and P. K. Sharma. 2021. "Profile Matching for IoMT: A Verifiable Private Set Intersection Scheme." *IEEE Journal of Biomedical and Health Informatics* 25 (10): 3794–3803.

[10] Benny, P., S. Thomas, and Z. Michael. 2014. "Faster Private Set Intersection Based on OT Extension." In *USENIX Security Symposium*, 797–812.

[11] Pinkas, B., T. Schneider, G. Segev, and M. Zohner. 2015. "Phasing: Private Set Intersection Using Permutation-Based Hashing." In *24th USENIX Security Symposium*, 515–530.

[12] Dong, C., L. Chen, and Z. Wen. 2013. "When Private Set Intersection Meets Big Data: An Efficient and Scalable Protocol." In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, 789–800.

[13] Lambæk, M. 2016. "Breaking and Fixing Private Set Intersection Protocols." *Cryptology ePrint Archive*. https://eprint.iacr.org/2016/665.

[14] Meadows, C. 1986. "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*, 134. IEEE.

[15] Huberman, B. A., M. Franklin, and T. Hogg. 1999. "Enhancing Privacy and Trust in Electronic Communities." In *Proceedings of the 1st ACM Conference on Electronic Commerce*, 78–86.

[16] Diffie, W., and M. Hellman. 1976. "Diffie-Hellman Key Exchange."

[17] Naor, M., and B. Pinkas. 2001. "Efficient Oblivious Transfer Protocols." In *SODA*, 448–457.

[18] Pagh, R., and F. F. Rodler. 2004. "Cuckoo Hashing." *Journal of Algorithms* 51 (2): 122–144.
[19] Tarkoma, S., C. E. Rothenberg, and E. Lagerspetz. 2011. "Theory and Practice of Bloom Filters for Distributed Systems." *IEEE Communications Surveys & Tutorials* 14 (1): 131–155.
[20] Chen, H., K. Laine, and P. Rindal. 2017. "Fast Private Set Intersection from Homomorphic Encryption." In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 1243–1255.
[21] Pinkas, B., M. Rosulek, N. Trieu, and A. Yanai. 2020. "PSI from PaXoS: Fast, Malicious Private Set Intersection." In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 739–767. Springer.
[22] Ghosh, S., and T. Nilges. 2019. "An Algebraic Approach to Maliciously Secure Private Set Intersection." In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 154–185. Springer.
[23] Ghosh, S., and M. Simkin. 2019. "The Communication Complexity of Threshold Private Set Intersection." In *Annual International Cryptology Conference*, 3–29. Springer.
[24] Chakraborti, A., G. Fanti, and M. K. Reiter. 2023. "Distance-Aware Private Set Intersection." In *32nd USENIX Security Symposium*, 319–336.
[25] Hu, J., J. Chen, W. Dai, and H. Wang. 2023. "Fully Homomorphic Encryption Based Protocols for Enhanced Private Set Intersection Functionalities." *Cryptology ePrint Archive*.
[26] Ghosh, S., and M. Simkin. 2023. "Threshold Private Set Intersection with Better Communication Complexity." In *IACR International Conference on Public-Key Cryptography*, 251–272. Springer.
[27] Kußmaul, J., M. Akram, and A. Tueno. 2023. "Unbalanced Private Set Intersection from Homomorphic Encryption and Nested Cuckoo Hashing." *Cryptology ePrint Archive*. https://eprint.iacr.org/2023/1670.
[28] Fan, J., and F. Vercauteren. 2012. "Somewhat Practical Fully Homomorphic Encryption." *Cryptology ePrint Archive*. https://ia.cr/2012/144.
[29] Benaissa, A., B. Retiat, B. Cebere, and A. E. Belfedhal. 2021. "TenSEAL: A Library for Encrypted Tensor Operations Using Homomorphic Encryption."