DEIM Forum 2018 F7-2

Privacy-Preserving Join Processing over outsourced private datasets with Fully Homomorphic Encryption and Bloom Filters

Arisa TAJIMA † Hiroki SATO ‡ and Hayato YAMANA §

† School of Fundamental Science and Engineering, Waseda University 3-4-1 Okubo, Shinjuku-ku, Tokyo, 169-8555 Japan

‡ Graduate School of Fundamental Science and Engineering, Waseda University 3-4-1 Okubo, Shinjuku-ku, Tokyo, 169-8555 Japan

§ Faculty of Science and Engineering, Waseda University 3-4-1 Okubo, Shinjuku-ku, Tokyo, 169-8555 Japan

E-mail: {arisatajima, hsato, yamana}@yama.info.waseda.ac.jp

Abstract Cloud database services have become attractive among organizations, governments, or individuals with the increasing number of data to be analyzed. Outsourcing sensitive data and delegating data processing to the cloud services, however, cause security and privacy issues because they are not always trustable. In this paper, we address the problem of answering join queries across outsourced private databases while maintaining the data confidentiality. This query model is common in a study field where a researcher is willing to know a certain correlation between databases owned by different enterprises. We especially present a protocol that allows secure join operation toward datasets on a cloud server, introducing a new functionality of outsourced private set intersection cardinality with fully homomorphic encryption and bloom filters.

Keyword Private Database, Outsourced join operation, Cloud Computing, FHE, Bloom Filter, PSI

1. Introduction

In an outsourced database model, it is significant to provide a way to securely execute queries while protecting data privacy. The outsourced data confidentiality itself can be achieved by encrypting the whole data. Once data are encrypted by traditional encryption schemes, however, no query processing can be performed without their decryption. In this case, in order to execute outsourced queries, the data have to be decrypted on the cloud-side, which causes security issues. A way to solve this problem is to adopt fully homomorphic encryption (FHE) [1], which allows one to compute arbitrary functions on encrypted data. Although a cloud service provider can process any queries while maintaining data confidentiality, FHE requires large time and memory complexities, which results in inefficient processing.

In this study, we focus on join operations in database queries, which are used in many systems in the real world. For example, it is necessary to join databases when a researcher wants to find a certain correlation between multiple databases. We especially assume the case where two datasets maintained by different entities are outsourced to a cloud service, and the cloud service counts the number of elements in the intersection of the datasets. In this situation, computing the result without revealing the content of the data to the cloud service is required.

We propose a secure protocol for join operations on the cloud by leveraging the idea of outsourced private set intersection cardinality (OPSI-CA), adopting bloom filters [2] and FHE. The novelty of our proposed protocol is that to the best of our knowledge this is the first approach for outsourced join processing adopting FHE.

2. Background

2.1. Notations

Table 1 describes notations referred to in this paper.

Table 1. Notations

Notation	Description
D_I	A dataset of data owner I.
$N_I = \|D_I\|$	Number of elements in D_I.
$m,\ k$	Size of bloom filter and number of hash function, respectively.
fp	False positive rate in bloom filter.
fp_{MAX}	A desirable maximum false positive rate in our protocol.
n_{max}	Maximum number of elements that can be stored on the cloud while keeping a false positive rate less than fp_{MAX}.
s	Number of slots in a single ciphertext.
$p = \lceil m/s \rceil$	Number of packed ciphertexts used in a single bloom filter.
$[b_1, \ldots, b_u]$	A bit-vector with length u.
$\llbracket \overline{b_1} \| \ldots \| \overline{b_u} \rrbracket$	A single packed ciphertext, encryption of $[b_1, \ldots, b_u]$.
BF_{All}^I	A bloom filter that represents all elements of D_I.
BF_i^I	A bloom filter that represents the i-th element of a set D_I.
$BF_{i,j}^I$	The j-th splited bloom filter of BF_i^I.
$\overline{BF_{i,j}^I}$	A packed ciphertext, which is the encryption of $BF_{i,j}^I$.
$\overline{BF_i^I}$	Concatenation of ciphertexts representing i-th element of D_I.
$\oplus$	Addition over the packed ciphertexts.
$\otimes$	Multiplication over the packed ciphertexts.

2.2. Fully Homomorphic Encryption (FHE)

Fully homomorphic encryption (FHE) [1] supports an arbitrary number of computations in both addition and multiplication over encrypted data. We here present a high-level overview of a BGV-style FHE scheme [3] and introduce SV packing technique [4].

2.2.1. The BGV-Style FHE Scheme

The BGV-style FHE scheme consists of the following five algorithms:

- $FHE.SetUp(1^{\lambda})$: Given a security parameter λ, outputs a set of encryption parameters: $params$.
- $FHE.KeyGen(params)$: Generates public key pk / secret key sk pair, and evaluation keys ek.
- $FHE.Enc(pk,m)$: Given the public key pk and a message m, produces a ciphertext c.
- $FHE.Dec(sk,c)$: Given the secret key sk and a ciphertext c, produces a message m.
- $FHE.Eval(ek,f,(c_1,\dots,c_t))$: Given the evaluation key ek, an arithmetic circuit f, and ciphertexts $c_1,\dots,c_t$, where t is the number of inputs to f, outputs a ciphertext c_f.

2.2.2. SV Packing

In [4], Smart and Vercauteren propose SIMD style operations on packed ciphertexts based on polynomial-CRT (Chinese Remainder Theorem) that allows one to encrypt vector of plaintexts in a single ciphertext. Since FHE schemes generally encrypt small plaintexts in the large ciphertexts, using FHE with this packing technique is efficient in memory space and computational resources.

2.3. Bloom Filter

A bloom filter (BF) [2] is a well-known space-efficient probabilistic data structure used to test whether an element is a member of a set. The structure facilitates efficient search, which never generates false negatives but may yield a small probability of false positives: $fp = (1-(1-\frac{1}{m})^{kn})^k \approx (1-e^{-\frac{kn}{m}})^k$, where k is the number of hash functions, n is the number of elements in a set, and m is the size of bloom filter. The basic operations are adding elements to a set and processing membership queries in probabilistic set representation. To answer the membership queries, bloom filter requires only constant time to the size of the set.

3. Related Work

Private Set Intersection (PSI) [5] is a cryptographic technique that allows two parties to compute the intersection without revealing anything except the intersection. In this section, we present existing studies of PSI whose computation is delegated to an untrusted cloud server. In the delegated PSI setting, the cloud server is required to learn nothing about outsourced data content and computation results while computing set intersection.

In [6], computation of private set intersection can be outsourced to an oblivious service provider adopting bloom filter and homomorphic encryption. However, since a client encrypts and outsources a bloom filter of a set rather than that of elements, the protocol does not support data storage outsourcing. In [7], the protocol allows clients to outsource their datasets to a cloud server by hashing the datasets and randomizing the hashed data. In order to compute the set intersection collaborative operations of the clients and the server are required. Thus, the protocol is not the case of fully delegated set intersection operation. Furthermore, in this protocol, the server can know the cardinality of the intersection, which is not fully private. The protocol in [8] allows two clients to outsource their datasets and delegate the set intersection operation to an untrusted cloud server with a verifiable mechanism to faithfully conduct set intersection operation. This protocol, however, leaks the cardinality of the intersection. Recently, Abadi et al. [9] have proposed a new delegated private set intersection scheme by adopting additive homomorphic encryption and point-value polynomial representation. In this protocol, clients outsourced their datasets by representing them as blinded polynomials. Later, they proposed a more efficient protocol with hash tables and point-value polynomial representation [10]. Both protocols are secure in that the cloud server learns nothing about the exact number of the set elements and the cardinality of the intersection. The set intersection operations in these protocols, however, are not fully delegated because both clients participate in the computation delegation phase.

We summarize these related works and compare to our protocol in Table 2. PSI-CA private means that the cloud does not learn the size of a set intersection. Fully delegation means that clients do not participate in the delegation phase. Data storage outsourcing means that each element is outsourced to the cloud. In our protocol, set intersection cardinality is protected from the cloud and our scheme supports data storage outsourcing and fully delegation.

Table 2. Property comparison of OPSI-related work

Property	[6]	[7]	[8]	[9]	[10]	Ours
PSI-CA privacy	✓			✓	✓	✓
Fully delegation	✓		✓			✓
Data storage outsourcing		✓	✓	✓	✓	✓

4. Model

We present our problem setting in this section. In our scenario (see Fig. 1), there are four players: data owner A, data owner B, querier Q, and cloud server S. Data owner A owns dataset D_A and data owner B owns dataset D_A respectively, and they outsource the datasets to cloud server S. Querier Q later asks cloud server S to perform a join operation to retrieve the size of the data common to both datasets, D_A and D_B.

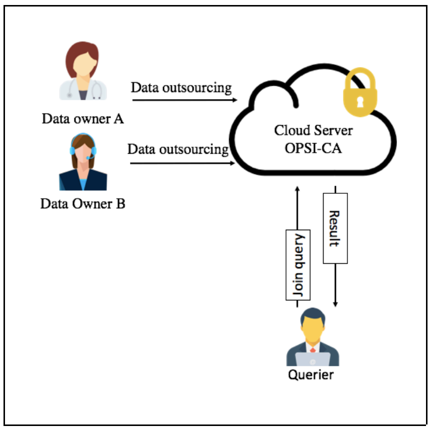

Fig. 1. Our scenario

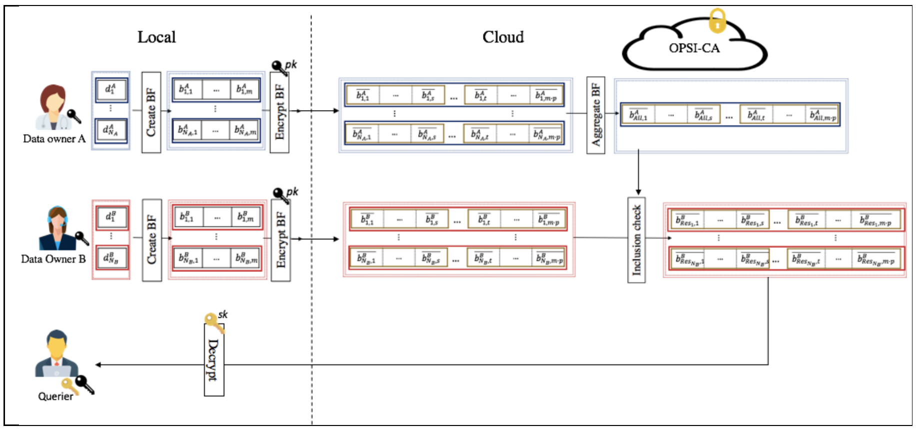

Fig.2 Overview of our approach

In this scenario, we consider honest-but-curious cloud service providers as adversaries and require the cloud server does not collude with any other parties. For security requirements in each entity are follows: The querier learns only the query result and no information about the content of the outsourced data; The data owner learns nothing about the query results and the content of the outsourced data which are owned by the other data owner; The server learns nothing about the query result and the content of the outsourced data. In our protocol, however, we accept leakage of data size that is stored in the cloud server.

5. Proposed Method

We adopt a new functionality of outsourced private set intersection cardinality (OPSI-CA) with FHE and bloom filter. We define the OPSI-CA functionality as a protocol where the cloud computes set intersection of outsourced datasets and returns the cardinality of the intersection.

5.1. Protocol

Fig. 2 depicts the overview of our approach and Fig. 3 explains our whole protocol.

Input: Data owner A's dataset D_A of size N_A; data owner B's dataset D_B of size N_B. $N_A, N_B \leq n_{max}$.
Output: Querier outputs $|D_A \cap D_B|$.

1. **Querier Setup**: Querier performs the following:
   [Create FHE parameters] Generates a public-secret key pair.
   [Send the FHE public key] Sends the public key to the data owner A, B, and the server.
   [Send BF parameters] Sends desirable fp_{max} and n_{max} to the data owner A and B.
2. **Data Owner Setup and Data Outsourcing:** Each data owner $I \in \{A, B\}$ performs the following:
   [Create BFs] Creates bloom filters for each element.
   $\forall i. BF_i^I = [b_{i,1}^I, \dots, b_{i,m}^I] (1 \leq i \leq N_I)$.
   [Split each BF] Splits each BF into p vectors, where $p = \lceil m/s \rceil$.
   $\forall i. BF_i^I = BF_{i,1}^I, .., BF_{i,p}^I$
   $= [b_{i,1}^I, \dots, b_{i,s}^I], \dots, [b_{i,m\cdot p-s+1}^I, \dots, b_{i,m\cdot p}^I] (1 \leq i \leq N_I)$.
   [Encrypt the BFs] Encrypts the BFs in batch.
   $\forall i,j. Enc_{pk}(BF_{i,j}^I) = \overline{BF_{i,j}^I}\ (1 \leq i \leq N_I, 1 \leq j \leq p)$.
   A single encrypted bloom filter $\overline{BF_i^I}$ is composed of p ciphertexts as follows.
   $\forall i. \overline{BF_i^I} = \overline{BF_{i,1}^I}, \dots, \overline{BF_{i,p}^I}$
   $= [\![\overline{b_{i,1}^I}| \dots |\overline{b_{i,m}^I}]\!], \dots, [\![\overline{b_{i,m\cdot p-s+1}^I}| \dots |\overline{b_{i,m\cdot p}^I}]\!] (1 \leq i \leq N_I)$.
   [Send the encrypted BFs] Sends the $N_I \cdot p$ ciphertexts to the cloud server.
3. **Cloud Server Set Intersection Operation**: Define X and Y as follows: If $N_A \geq N_B$, $X = A, Y = B$. Otherwise, $X = B, Y = A$.
   [Aggregate the encrypted BFs] Cloud server aggregates all the encrypted bloom filter of data owner X by bit-wise OR operation $\underline{OR}$ with a SIMD technique.
   $\forall j. \overline{BF_{All,j}^X} = \overline{BF_{1,j}^X}\ \underline{OR}\ \dots\ \underline{OR}\ \overline{BF_{N_X,j}^X}\ (1 \leq j \leq p)$.
   [Perform inclusion check] Cloud server checks if each element of the data owner Y is included in the set of the data owner X.
   $\forall i,j. \overline{BF_{Res_i,j}^Y} = ((\overline{BF_{All,j}^X} \otimes \overline{BF_{i,j}^Y}) \oplus \overline{BF_{i,j}^Y}) \oplus \overline{1}\ (1 \leq i \leq N_Y, 1 \leq j \leq p)$.
   If the element is included in the other set, the result is $\overline{1}$. Otherwise, the result ciphertext is distributed in $\{0,1\}$.
   [Send the result] Sends the $N_Y \cdot p$ ciphertexts to the querier.
4. **Querier result retrieving**:
   [Decrypt the result] Querier decrypts the encrypted BFs and count up # of BFs where all the bits within the single bloom filter are set to 1.

Fig. 3. Our proposed protocol

5.2. Security Proof

We now discuss the security of our protocol in an honest-but-curious model. By adopting FHE scheme, the cloud server does not learn data contents and query results since all the data are encrypted and it does not hold a secret key to decrypt them. For the same reason, the data owner A does not learn data contents of data owner B and vise versa. Additionally, since bloom filters are built with one way hash functions, the querier does not learn data content of the data owners even if encrypted bloom filters are decrypted by a secret key. Thus, security requirements for each party in our model are satisfied.

6. Implementation and Evaluation

6.1. Implementation

We implemented the protocol shown in Fig.1 with c++. For FHE, we used the homomorphic encryption library HElib [11] [12], which is based on the BGV scheme. For bloom filter, we implemented by ourselves and used MurmurHash3 [13] as a hash function.

6.2. Experimental Setup

Our experiment is run on CentOS 6.7, Intel Xeon CPU E7-8880 v3 @ 2.3GHz, and 1TB memory. The FHE parameters for HElib that we used are given in Table 3. In all experiments, we fix $fp_{max} = 0.001$ and $n_{max} = 100$, namely the false positive rate of a bloom filter is at most

fp_{max} if the input data size is less than n_{max}. Thus, in order to keep the false positive rate at most 0.001, we vary the number of elements up to 100, which determines the performance of our protocol. Each party's computation is executed with either single thread or multi-thread as shown in Table 4.

Table 3. FHE parameter sets for HElib

Message Space	m	s	L	security
$\mathbb{Z}_2$	9773	112	10	133

m: ring modulus, s: # of slots, L: # of ciphertext moduli.

Table 4. Number of threads in each party

Querier	Data Owner	Cloud Server
1	4	20

6.3. Performance Evaluation

In this section, we show the experimental results of our protocol and evaluate the performance. We measured the computation time in each party.

(1) Data Owner: The result of the data owner-side in Fig. 5 shows the running time of creating encrypted bloom filter. In our experiments, the length of bloom filter m and the number of hash functions k are 1437 and 9, respectively. Since m and k are determined by two fixed values: maximum false positive rate fp_{max} and maximum elements n_{max}, the computation time on the data owner-side increases linearly with the number of elements.

(2) Cloud Server: The result of the cloud server-side in Fig. 5 shows the running time of calculating the set intersection cardinality. We here emphasize that the important algorithm in our cloud-side protocol is the aggregate operation, which makes efficient in the inclusion check operation, namely the computation becomes linear with the number of elements. Otherwise, it requires quadratic time in the inclusion check operation.

(3) Querier: The result of the querier-side in Fig. 5 shows the running time of decrypting the OPSI-CA result. The time increases linearly with the number of elements.

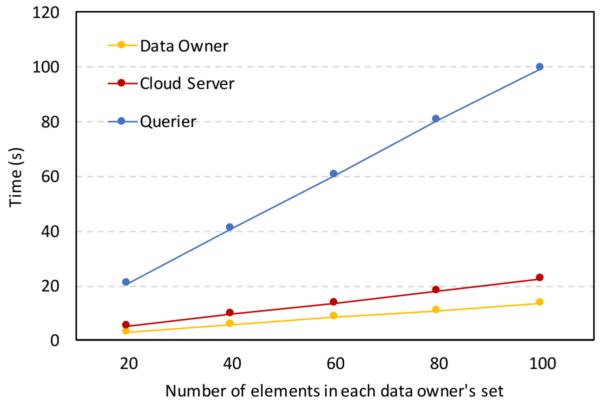

Fig. 5. Computation time in each party

7. Conclusion

In this paper, we proposed a protocol that allows secure join operations toward outsourced datasets in a cloud environment. Our protocol adopts a new functionality of outsourced private set intersection cardinality (OPSI-CA) by combining the BGV-style FHE scheme with bloom filters. Our results show that computation overhead in each party scaled linearly in the number of elements. For example, with 100 element sets, it requires 22.5s for the cloud to calculate OPSI-CA. In our protocol, our FHE scheme enables the cloud to process the join query with strong security guarantees. We believe that our study can be a help in developing practical applications with FHE.

Acknowledgment

This work was supported by JST CREST Grant Number JPMJCR1503, Japan.

Reference

[1] Gentry, C. Fully homomorphic encryption using ideal lattices. STOC '09, pp.169-178, 2009.

[2] Bloom, B. H. Space/time trade-offs in hash coding with allowable errors. Communications of the ACM, pp.422-426, 1970.

[3] Brakerski, Z., Gentry, C., & Vaikuntanathan, V. (Leveled) fully homomorphic encryption without bootstrapping. ITCS, pp.309-325, 2012.

[4] Smart, N. P., & Vercauteren, F. Fully homomorphic SIMD operations. Designs, codes and cryptography. pp.1-25, 2014.

[5] Freedman, M. J., Nissim, K., & Pinkas, B. Efficient private matching and set intersection. EUROCRYPT'04, pp. 1-19, 2004.

[6] Kerschbaum, F. Outsourced private set intersection using homomorphic encryption. ASIACCS'12, pp.85-86, 2012.

[7] Liu, F., Ng, W. K., Zhang, W., & Han, S. Encrypted set intersection protocol for outsourced datasets. IC2E, pp.135-140, 2014.

[8] Zheng, Q., & Xu, S. Verifiable delegated set intersection operations on outsourced encrypted data. IC2E, pp.175-184, 2015.

[9] Abadi, A., Terzis, S., & Dong, C. O-PSI: delegated private set intersection on outsourced datasets. IFIP Int'l Information Security Conference, pp.3-17, 2015.

[10] Abadi, A., Terzis, S., Metere, R., & Dong, C. Efficient delegated private set intersection on outsourced private datasets. IEEE Trans. on Dependable and Secure Computing, 2017.

[11] Halevi, S., & Shoup, V. Algorithms in helib. CRYPTO'14, pp.554-571, 2014.

[12] HElib. https://github.com/shaih/HElib, 2016.8.12.

[13] MurmurHash3. https://github.com/aappleby/smhasher.