Private Boolean Query Processing on Encrypted Data

Hoang Giang Do(✉) and Wee Keong Ng

School of Computer Science and Engineering, Nanyang Technological University, Singapore, Singapore
do0004ng@e.ntu.edu.sg, wkn@pmail.ntu.edu.sg

Abstract. Outsourcing the data to the clouds offers an opportunity to drastically reduce costs of storing and processing data. On the other hand, it deprives the data owners of direct control over their data and that introduces new privacy risks. Data encryption has been introduced to tackle the data confidentiality issue. However, data encryption also brings a new challenge of query processing over encrypted data. Recently, solutions for supporting query over encrypted data have been developed. However, they are either failing to support complex queries or insecure regarding certain security requirements (i.e. access patterns, query privacy). In this paper, we propose a novel privacy-preserving query processing framework to support boolean queries over encrypted data. Our framework utilizes Bloom filter and additive homomorphic encryption to systematically derive the query evaluation results in a privacy-preserving manner. We theoretically and empirically analyze the performance of the proposed protocols and demonstrate their practical values.

1 Introduction

The cloud computing paradigm has offered the user an opportunity to drastically reduce costs of storing and processing data. Outsourcing data storage as well as data management to the clouds is useful in many services including law enforcement, finance, healthcare, etc. However, data outsourcing deprives the data owners of direct control over their data, and it brings new crucial security risks.

Since there are many possibilities for data leakage to occur at the server side, the user should not fully trust the clouds for data privacy. One possibility comes from corrupted employees who do not follow privacy policies. They may intentionally or unintentionally reveal sensitive information such as personal health records, financial transaction, or personal voice, etc. Even when the cloud service provider claims to enforce sufficient policies on such privacy violence, there are still chances that cloud computing systems may be vulnerable to external malicious attacks. Once intrusions take place, any single detail of these sensitive data will be publicly exposed. Finally, due to privacy regulations of different countries, the cloud data may be required to be shared with a certain third party. Therefore, storing plaintext data in the cloud might lead to full exposure of your data.

K.-Y. Lam et al. (Eds.): ICICS 2016, LNCS 9977, pp. 321–332, 2016.
DOI: 10.1007/978-3-319-50011-9_25

Data encryption has been introduced to tackle the problem of data confidentiality on the clouds. However, at the same time, it also creates a new challenge of data usability over remotely encrypted data. While the data owner should have the ability to query and obtain useful data when needed, the simple encryption method is insufficient for the basic requirement. There are different approaches to address this problem of privacy preserving encrypted data retrieval in the clouds. They include (i) downloading the entire dataset and doing a local search on the decrypted data or (ii) encrypting the data by a searchable encryption method and performing searching over the encrypted data with the aid of the cloud server. Since the first approach is not feasible as it incurs heavy communicational and computational cost on the end-user, the latter has gained significant attention in recent years. Following this direction, various methods have been proposed for evaluating search queries over encrypted data. However, most of the existing works focus on the problem of single keyword matching and inherently, are not suitable to execute complex queries.

In this paper, we study the problem of supporting an important class of complex search operations: boolean keyword retrieval. We consider a scenario where an encrypted dataset contains a number of encrypted documents. Each document is associated with a certain set of keywords. An authorized user is able to perform search queries which are the combination of logic predicates on the keyword set. The output of the query processing is the index set of satisfied documents and nothing else to the data consumer. During the query processing, not only the outsourced data are kept private from the clouds, but also the user's input query remains in confidentiality.

We organize the paper as follows: In the next section, we review the existing works on secure query processing over encrypted data. Section 3 describes our problem statement, data model, and query model as well as the requirements for the designed framework. Section 4 reviews the necessary building blocks which are Bloom Filters and additively homomorphic public-key encryption scheme (i.e. the Pailliers encryption scheme). The proposed solution for complex boolean query processing on encrypted data is presented in Sect. 5. The section systematically discusses four stages of the solution. Section 6 presents our experimental evaluations of the proposed sub-protocols. The final section discusses future works and concludes the paper.

2 Related Work

There are two general approaches that tackle the problem of secure query processing over encrypted data without downloading the entire dataset. The first approach deploys tamper-proof trusted hardware (which is either trusted or certified by the clients) on the cloud-side. The hardware provides a secure environment that allows the cloud to perform secure operations over the data in critical query processing stages. Along this direction, Bajaj and Sion [1] leveraged the existence of trusted hardware to design TrustedDB, an outsourced database prototype that allows a client to execute SQL queries with privacy and

under regulatory compliance constraints. However, the secure hardware is very expensive and may not be suitable for general cloud computing paradigm which typically makes use of cheap commodity machines.

The second approach makes use of cryptographic protocols to perform operations over the encrypted data. Song *et al.* [14] proposed the first searchable symmetric-key encryption. Goh [9] provided the first security definition for searchable symmetric encryption and presented a Bloom-filter based solution with linear complexity. Curtmola *et al.* [5] introduced the notion of adaptive semantic security for symmetric searchable encryption together with a sub-linear time and sub-linear space constructions. These techniques only support single keyword search, and not suitable for complex queries.

Boneh and Waters [3] presented a new cryptographic primitive called hidden vector encryption that allows evaluating conjunctive queries over encrypted data. The method also supports general subset and range queries. However, it is worth to note that their technique is very expensive when data domain is large and complex to implement. Do and Ng [7] proposed another scheme that can handle conjunctive keywords search and multidimensional range queries by performing prefix encoding before encryption. We note that the methods are only suitable for conjunctive queries, but not directly applicable to disjunctive queries. Moreover, almost all existing techniques fail to provide rigorous privacy protection due to access pattern leakage. Islam *et al.* [11] showed that data access pattern leakage could lead to the disclosure of a significant amount of sensitive information. In our proposed protocol, we design a secure protocol that not only preserves query privacy but also keep the access patterns in confidentiality.

3 Definitions and Assumptions

3.1 Problem Statement

In our problem settings, we consider three different parties: the data owner, the cloud and the data consumer. Let D denote Alice's data with n documents. Each document is associated with a certain set of keywords that enable to search or retrieve it efficiently. We assume that the data owner wishes to encrypt D using his/her own key and outsources the encrypted data to a cloud so that latter an authorized data consumer is able to search on the encrypted data. The input query is represented by a logical combination (i.e. negation, conjunction, and disjunction) of the keyword predicates. At the same time, several security issues should be addressed such as data confidentiality, the privacy of query content, etc.

Formally, let $W = \{0,1\}^*$ be a universe of words and $D \subseteq W$ be corpus. Let $Kw = \{w_1, \cdots, w_n\}$ denote a set of searchable keywords. The keyword set Kw is publicly known (hence it is called a common reference keyword set). While Kw can be any the searchable property set of the corpus D, for simplicity, we assume the problem as a general boolean keyword search. That means $Kw \subseteq D$ and the predicate $d(w_i) = 1$ if and only if the document d contains the keyword w_i. Otherwise, $d(w_i) = 0$.

A boolean query q_K contains keyword predicate and a set of logical expressions $\wedge, \vee, \neg$. Let $d_C = \{d^{(1)}, \ldots, d^{(n)}\}$ be n documents stored in a server C. With a set of keywords $K \subseteq Kw$, we define a query $q_K : d \rightarrow \{0, 1\}$ that takes a document d as input and outputs 1, if and only if d matches the criteria.

3.2 Security Assumptions

In this paper, we assume that the data owner, the cloud and the data consumer are semi-honest. The cloud server will correctly follow the protocol specification, however, at the same time, it is also curious about stored data as well as the query content. In general, a secure complex boolean query should meet the following privacy requirements:

- Data Confidentiality. The data are encrypted by a provably secure cryptosystem. Besides, during the query processing, the cloud should not gain any new knowledge on the stored data.
- Query Privacy. At any point of time, the data consumer's query should never be revealed to the cloud and the data owner.
- Access Pattern Privacy. Data access patterns of the data consumer should not be disclosed to the cloud and the data owner. Data access patterns are the information of the documents satisfy the query (even the attackers do not know the query content).
- End-user's Privacy. At the end of the query processing protocol, only the satisfied results should be revealed to the data consumer and nothing else.

4 Building Blocks

4.1 Bloom Filter

Bloom filter [2] provides a way to probabilistically represent set membership of elements using a small amount of space, even when the universe set is large. It represents a set $S = \{s_1, \ldots, s_n\}$ of n element by a space-efficient m-element array $B = \{B[1], B[2], \ldots, B[m]\}$. A random set of hash functions $h_1, \ldots, h_t$, where each function $h_i : \{0, 1\}^* \rightarrow [0, \ldots, m]$ are chosen to associate with the Bloom filter B.

The filter algorithm is constructed as follows. The bit array B is initially set to 0. For each element $s_i \in S$, the bits corresponding to the positions $h_1(s_i), h_2(s_i), ..., h_t(s_i)$ are set. The same bit in the array may be set several times without any restriction. Figure 1 depicts how an element is inserted into a Bloom filter.

After the Bloom filter is constructed, membership queries can be easily answered. To determine whether an element x belongs to the set S, we check all the bits corresponding to the positions $h_1(s), h_2(s), ..., h_t(s)$. If at least one bit is 0, then $x \notin S$ for sure. Otherwise, we conclude that $x \in S$. Actually, a false positive may occur when an element $x \notin S$ is recognized as an element of the set. However, mathematical analysis shows that the probability when the algorithm returning 1 for $x \notin S$ is approximately $(1 - e^{\frac{-tn}{m}})^t$, which is small enough for the practical use.

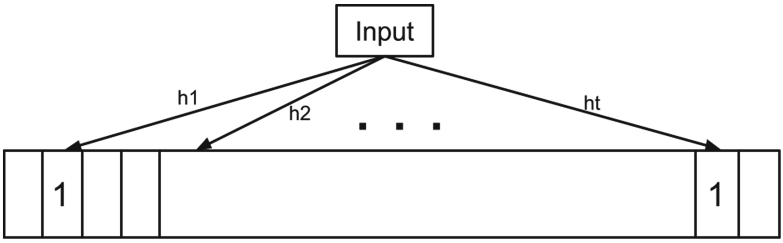

Fig. 1. Bloom filter insertion

4.2 Additive Homomorphic Encryption

A homomorphic encryption scheme is a cryptosystem that allows arithmetic operations to be performed on the cirphertext without decryption or knowing the actual values. Consider two operators $\times$ and $+$ on the ciphertext and plaintext domain, respectively, if m_1 and m_2 are two plaintext elements, an additive homomorphic scheme Enc satisfies

$$Enc(m_1) \times Enc(m_2) = Enc(m_1 + m_2).$$

Efficient additive homomorphic cryptosystems have been proposed such as the Pallier cryptosystem [12], or the Damgard and Jurik cryptosystem [6] which is length-flexible Paillier's encryption scheme. We emphasize that any additive homomorphic encryption scheme that satisfies the above properties can be utilized to implement our proposed framework. However, for simplicity, we assume that the additive homomorphic encryption we are using in this paper is the Paillier cryptosystem. The security of the Paillier encryption scheme relies on the computational hardness assumption of a novel mathematical problem called the composite residuosity. The decision version of this problem class assumes that no polynomial time algorithms can distinguish *N-th* residues modulo N^2 with non-negligible probability. We refer to the reader [12] for more details.

4.3 Oblivious Transfer and OT Extension

Oblivious transfer (OT) is a major building block for designing a number of secure computation protocols. The protocol consists of two parties: the receiver and the sender. The basic 1-out-of 2 OT_1^2 allows the receiver choose either one from two input of the senders without learning anything regarding the other. OT_1^2 was introduced by Even *et al.* [8] as a generalization of Rabin "oblivious transfer" [13]. Brassard *et al.* [4] further extended OT_1^2 to 1-out-of n OT_1^n where the receiver is able to obtain one from n messages possessed by the sender. Since then, many efficient protocols for OT with different security assumptions have been proposed over the years. k-out of n OT_n^k scheme is the final form of OT schemes and the one we make use in our solution. In it, from n encrypted messages sent by the sender, the chooser can obtain k of them which he had

chosen without the senders knowledge about which part of the messages can be obtained by him. While it is clear that OT_n^k can be constructed by applying k repetitions of OT_1^n, there have been more efficient protocols. Wu *et al.* [15] introduced two-lock cryptosystems to improve the efficiency of OT_n^k protocol from $O(kn)$ to $O(k+n)$. Recently, Guo *et al.* [10] proposed a cryptographic concept called subset membership encryption and applied it to construct two round OT_n^k protocol against semi-honest adversaries. The algorithm only requires the communication cost of $O(n)$ for the sender and $O(k)$ for the receiver.

5 Proposed Framework

5.1 System Overview

As mentioned in Sect. 3, the data owner outsources a corpus D of encrypted documents so that later the data consumer is able to perform complex boolean keyword queries. A query is represented by logical expressions $\wedge, \vee, \neg$ of boolean keyword predicates q_k. The results of the query are the indices of satisfied documents.

This paper explicitly assumes that each document d_i is associated with a subset $S^{(i)} \subseteq K$ of keywords. For simplicity, we assume that each document has the same number of associated keywords. For each document, we represent these associated keywords by a Bloom-filter of size m. Let $\{h_1, h_2, \ldots, h_t\}$ be independently keyed hash functions. A keyed hash function h_i inputs a secret key from key space K and a keyword k and outputs an integer in the range of $[1, \ldots, m]$. The preprocessing stage when the data owner prepares the data and uploads them to the cloud is described as follows:

1. The data owner generates a secret pseudorandom function F mapping an integer to a random key in the key space K.
2. For each document $d^{(i)}$, the data owner does the following:
   - Generating a key for hash functions: $k^{(i)} \leftarrow F(i)$.
   - Creating a Bloom-filter $B^{(i)}$ associated the documents.
   - Inserting the document keyword set $S^{(i)}$ into Bloom-filter $B^{(i)}$ with the hash functions: $h_1, h_2, ..., h_t$ using the key $k^{(i)}$. Concretely, for each keyword $w_j^{(i)}$ of the document $d^{(i)}$, we set the bit $h_1(k^{(i)}, w_j^{(i)}), \ldots, h_t(k^{(i)}, w_j^{(i)})$ in Bloom-filter $B^{(i)}$.
   - Encrypting the content of the document by a standard cryptosystem (i.e. AES).
3. The data owner sends the encrypted dataset as well as the Bloom-fiters for all encrypted documents to the cloud server.
4. The data owner shares the secret function F with the authorized data consumer.

Let $D^{'}$ denote the outsourced data of the data owner. $D^{'}$ consists of multiple records and each of them has the following form:

$$\Big\langle \text{Document index},\ \text{Encrypted content},\ \text{Bloom Filter data} \Big\rangle$$

Two arbitrary documents have two different Bloom filters which are generated by two different sets of hash functions (i.e. different keys for keyed hash functions). Moreover, the keys are generated by a pseudorandom function F. The input of F is the index of the document. Hence, with the view of Bloom filters data, the cloud cannot make any conclusion about the associated keyword sets.

Now consider an authorized data consumer (which would typically be authorized by the data owner) who wants to securely retrieve data from $D^{'}$ in the cloud. The satisfied documents are defined by his/her private boolean query. The query phase consists of three stages as the following:

- Secure Single Keyword Evaluation - In this stage, the data consumer evaluates a single keyword query for each encrypted document. The output of this stage is the encryption of either 1 or 0, depending on whether the document contains the keyword.
- Secure Complex Boolean Query Evaluation - Based on the results of the previous stage, the data consumer collaborates with the cloud to compute the result of the complex logical combination of boolean predicates. Again, the output of this stage is the encryption of either 1 or 0 depending on whether the document satisfies the input query.
- Retrieval of Output Data - At this stage, the data consumer collaborates with the cloud to securely retrieve the indices of satisfied documents, and obtains the final documents by private information retrieval or oblivious RAM with known indices.

5.2 Secure Single Keyword Evaluation

We consider the scenario of evaluating a single boolean predicate $q_k(d)$ for each document d in the encrypted data corpus $D^{'}$. More concretely, we propose an algorithm to answer the query whether a particular encrypted document d contains a given keyword k.

We denote Alice, S, Bob be the data owner, the cloud, and the data consumer respectively. Let (pk, sk) be the Paillier key pair of S, and $Enc(\cdot)$ be the encryption under public key pk. Protocol 1 describes the algorithm that inputs an index i, and a keyword w and outputs an encrypted bit b. $b = 1$ if the document contains the keyword and 0 otherwise. The result is held by Bob but remains encrypted under S's public key pk.

In line 5 of Protocol 1, Bob and S collaboratively compute the mulitplication operation on the encrypted data. In this protocol (i.e. SecMul), Bob holds two private encrypted inputs $(Enc(x), Enc(y))$ and S keeps the Paillier secret key sk, where x and y are unknown to both two parties. The output of $SecMul(Enc(x), Enc(y))$ is $Enc(x \times y)$ and revealed only to Bob. The protocol $SecMul$ is presented in Protocol 2. Regarding the definition of $SecMul$ (i.e. Protocol 2), Bob iteratively computes the product of $\prod Enc(B^{(i)}[h_j(k, w)]$ in the encrypted form for $j = 1, \ldots, t$. The product equals to 1 if any only if all the bits of $\{Enc(B^{(i)}[h_j(k, w)]\}$ are set and 0 otherwise. Hence, the correctness of the protocol follows that observation.

Protocol 1. Secure Single Keyword Evaluation

Input: Integer i denotes the index of document $d^{(i)}$, keyword w, pk is Pailier public key of S

Output: Encrypted bit $Enc(b)$, where $b = q(w, d^{(i)})$ - whether $d^{(i)}$ contain w

1 S encrypts each bit in the Bloom filter $B^{(i)}$ by its Pailier public key;
2 Bob generates key $k = F(i)$;
3 S and Bob perform (t, m) oblivious transfer to send encrypted $\{Enc(B^{(i)}[h_j(k, w)])\}$ to Bob, $j = \overline{1, t}$;
4 Bob computes $r = Enc(1)$;
5 For each correspoding bit $h_j(k, w)$, Bob computes $r = SecMul(r, Enc(B^{(i)}[h_j(k, w)]))$;
6 Bob outputs r.

Regarding in line 1, 2 of Protocol 1, S is required to encrypt the Bloom filter sets each time for a single keyword evaluation. However, since a complex boolean query normally contains multiple of keyword evaluations, one time Bloom filters encryption and communication for a query are sufficient. The protocol requires n encrypted bit transfers for the Bloom filter and $2 \times t$ encrypted integer communication for t *Secure Multiplication* rounds.

Protocol 2. Secure Multiplication

Input: Bob holds $(Enc(x), Enc(y))$, and S holds the private key sk

Output: Bob holds $Enc(x \times y)$

1 Bob generates two random number r, s;
2 Bob computes $Enc(x + r), Enc(y + s)$ sends them to S;
3 S decrypts and obtains $x + r,\ y + s$;
4 S computes $(x + r)(y + s)$ and send $Enc((x + r)(y + s))$ to Bob;
5 Bob computes $Enc(x \times y) = Enc((x + r)(y + s)) - Enc(x \times s) - Enc(y \times r) - Enc(r \times s)$.

The computations at line 2, 5 of Protocol 2 are simply performed by the homomorphic property of Paillier encryption. During the protocol, Bob only works on encrypted data, while the server receives two random numbers. Hence, no information regarding x and y is gained by *Bob* and S. The correctness of the protocol is trivial as $(x + r)(y + s) = x \times y + x \times s + y \times r + r \times s$. The protocol requires 2 encrypted integer transfer for communication cost. Bob has to perform 5 multiplicative operations and 5 exponential operations in the ciphertext space.

5.3 Secure Complex Boolean Query Evaluation

At this stage, Bob holds the encrypted result of the evaluation for each document with each keyword appearing in the private query. This sub-section discusses

three basic primitives that operate on the encrypted inputs of the stage. With these primitives, Bob has the capability to compute the encryption of the desired bit result for each document's query evaluation. The output of this stage is a single encrypted bit for each document. That single bit indicates whether the document satisfied the query.

Inputs of the three primitives are either one single encrypted bit (NOT operation) or two encrypted bits (AND and OR operations). The descriptions of them are presented as follows:

1. $\neg$ (NOT) - It is straightforward to derive the formula for bit negation operation: $Enc(\neg x) = Enc(1) - Enc(x)$. Clearly, the operation leaks no information regarding the encrypted bit x to both the cloud S and Bob. It requires 1 exponential operation and 1 multiplicative operation. Clearly, the protocol leaks no information to Bob and S, since S receives no more data while Bob only works on his inputs which are encrypted data.
2. $\wedge$ (AND) - Because $x \wedge y = x \times y$ for any two bits x, y. The primitive is exactly the same with the description of SecMul (Protocol 2). The protocol requires 5 multiplicative operations and 5 exponential operations in the ciphertext space. The security of the protocol follows the analysis of Secure Multiplication (i.e. Protocol 2).
3. $\vee$ (OR) - Since $x \vee y = x + y - x \times y$, we can derive the definition of the OR primitive as in the Protocol 3. The protocol requires 7 multiplicative operations and 6 exponential operations in the ciphertext space. During the protocol, the data that Bob and S receive are exactly the same with those received during Protocol 2, hence, Bob and S gain nothing after the protocol execution.

Protocol 3. Secure OR Operation

Input: Bob holds $(Enc(x), Enc(y))$, and S holds the private key sk
Output: Bob holds $Enc(x \vee y)$

1 Bob and S collaboratively compute $Enc(x \times y)$ using protocol 2 ;
2 Bob computes $r = Enc(x) + Enc(y) - Enc(x \times y)$;
3 Bob outputs r;

5.4 Secure Retrieval of Output Data

Following the output of the Secure complex boolean query evaluation stage, Bob has the evaluation result (in encrypted form) for the combination of all predicates in the input on each data records. The goal of this stage is to utilize these results to reveal the raw evaluation result to Bob. The results are the indices of satisfied records. It is still worthy to point out that after this final stage, Bob can obtain only the result and nothing else, at the same time S gain nothing regarding Bob's query.

Let denote (pkB, skB) as Bob's Paillier key pair, and $Enc(pkB, \cdot)$ as the encryption using Bob's public key. The process of Secure retrieval of output data is presented in Protocol 4.

Protocol 4. Secure Retrieval of Output Data

Input: Bob holds $Enc(b_i)$- the encrypted evaluation result of each document,
Output: Bob holds S_r the set of statisfied indices

1 For each index id_i
(1.1) S sends $Enc(id_i)$ to Bob.
(1.2) Bob and S compute $Enc(b_i \times id_i) = SecMul(Enc(b_i), Enc(id_i))$
(1.3) Bob generates a random integer r_i, and computes $Enc(b_i \times id_i + r_i)$
(1.4) Bob sends $Enc(b_i \times id_i + r_i)$ and $Enc(pkB, r_i)$ to S.
(1.5) S decrypts to get $b_i \times id_i + r_i$
(1.6) S generates a random integer s_i and encrypts $Enc(pkB, b_i \times id_i + r_i + s_i)$
(1.7) S computes $Enc(pkB, s_i + r_i)$;
2 S sends pairs of $\{Enc(pkB, b_i \times id_i + r_i + s_i), Enc(pkB, s_i + r_i)\}$ to Bob in a random order;
3 Bob decrypts and computes $p_i = b_i \times id_i$;
4 If $p_i \neq 0$, Bob adds p_i to S_r;
5 Bob outputs S_r;

At line 1.4, the cloud S receives $Enc(pkB, r_i)$ (encrypted by Bob's public key) and a random number $b_i \times id_i + r_i$ for each document. Clearly, it learns nothing regarding the evaluation results of Bob's input query. On the other hand, Bob recieves two random number $b_i \times id_i + r_i + s_i$ and $s_i + r_i$ for document id_i, the only information he obtains is $b_i \times id_i$ and nothing else.

6 Implemetation

We implemented and calculated the CPU time required to run the sub-protocols that we propose in Sect. 5. Our experiments were conducted on a Windows 10.0 machines with a processor 3.0 GHz and 16 GB RAM. We used Paillier cryptosystem as the underlying additive homomorphic encryption scheme and implemented the proposed sub-protocols in Java. In order to make all the same sub-protocol to play the similar role, we implemented a simplified version of Secure Retrieval of Output Data (i.e. Protocol 4) protocol. In this simplified version, we consider there is only one document. As the result of the assumption, the output of the sub-protocol is either 0 or the index of the single document. Table 1 shows the processing time of four sub-protocols with different Paillier encryption key sizes.

While the specification of secure OR protocol requires 4 more multiplicative operations compared to secure AND protocol in the ciphertext space, the result shows that there is not much difference between the running times of secure AND and secure OR. On the other hands, we note that the running time of

Table 1. Running times of different sub-protocols in our implementation

Key size	Secure negation	Secure AND	Secure OR	Secure retrieval
512	4 ms	20 ms	22 ms	38 ms
1024	17 ms	73 ms	86 ms	150 ms
2048	81 ms	517 ms	558 ms	1090 ms

secure Negation is significantly larger than the difference between the previous two protocol. That means the encryption/decryption operations are more computationally expensive than performing arithmetic calculations on the ciphertext space. If we fix the Paillier encryption key size to 1024 bits, we note that the running times of secure retrieval of output data is 150 milliseconds, hence, we can easily handle thousands of data records in a reasonable time (i.e. 30 min). However, we observed that the computation cost of the sub-protocols increases by almost a factor of 7 when the Paillier key size is doubled.

7 Conclusion and Future Work

Thanks to various benefits (i.e. such as cost-efficiency and flexibility), outsourcing data storage and operational services to clouds has gained significant attention from both industry and academia. However, due to privacy concerns, data are typically encrypted before being outsourced. On the other hand, data encryption brings new challenges to both academia and industry which is query processing over encrypted data. Various techniques have been proposed to securely perform data retrieval over encrypted data. However, these techniques are either not directly applicable for evaluating complex queries over encrypted data or insecure regarding certain security requirements.

This paper presents a framework to securely evaluate boolean queries over encrypted data in the cloud. We applied Bloom filter and additive homomorphic encryption to construct a secure single keyword evaluation. We also presented an efficient mechanism to systematically combine the evaluation results of individual predicates to compute the corresponding query evaluation result. Our protocol not only protects data confidentiality and privacy of users input query but also hides the access patterns of the queries. The experimental results show that our protocol is practical for a small and medium size of data. As future work, we will design, implement and evaluate our protocols in parallel paradigms such as Map-Reduce or GPU. We also plan to extend our solutions to address other adversary models, such as fully malicious adversaries.

References

1. Bajaj, S., Sion, R.: TrustedDB: a trusted hardware based database with privacy and data confidentiality. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece (2011)

2. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7), 422–426 (1970)
3. Boneh, D., Waters, B.: Conjunctive, subset, and range queries on encrypted data. IACR Cryptology ePrint Archive 2006 (2006)
4. Brassard, G., Crépeau, C., Robert, J.-M.: All-or-nothing disclosure of secrets. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 234–238. Springer, Heidelberg (1987). doi:10.1007/3-540-47721-7_17
5. Curtmola, R., Garay, J.A., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA (2006)
6. Damgård, I., Jurik, M.: A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 119–136. Springer, Heidelberg (2001). doi:10.1007/3-540-44586-2_9
7. Do, H.G., Ng, W.K.: Privacy-preserving approach for sharing and processing intrusion alert data. In: Tenth IEEE International Conference on Intelligent Sensors, Sensor Networks and Information Processing, ISSNIP 2015, Singapore (2015)
8. Even, S., Goldreich, O., Lempel, A.: A randomized protocol for signing contracts. Commun. ACM **28**(6), 637–647 (1985)
9. Goh, E.: Secure indexes. IACR Cryptology ePrint Archive (2003)
10. Guo, F., Mu, Y., Susilo, W.: Subset membership encryption and its applications to oblivious transfer. IEEE Trans. Inf. Forensics Secur. **9**(7), 1098–1107 (2014)
11. Islam, M.S., Kuzu, M., Kantarcioglu, M.: Access pattern disclosure on searchable encryption: ramification, attack and mitigation. In: 19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA (2012)
12. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999). doi:10.1007/3-540-48910-X_16
13. Rabin, M.O.: How to exchange secrets with oblivious transfer. IACR Cryptology ePrint Archive 2005 (2005)
14. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: 2000 IEEE Symposium on Security and Privacy, Berkeley, California, USA (2000)
15. Wu, Q.-H., Zhang, J.-H., Wang, Y.-M.: Practical t-out-n oblivious transfer and its applications. In: Qing, S., Gollmann, D., Zhou, J. (eds.) ICICS 2003. LNCS, vol. 2836, pp. 226–237. Springer, Heidelberg (2003). doi:10.1007/978-3-540-39927-8_21