An Effective Encryption Scheme on Outsourcing Data for Query on Cloud Platform

Jianchao Tang[1(✉)], Shaojing Fu[1,2], and Ming Xu[1]

[1] National University Of Defense Technology, Changsha 410073, China
tangjianchao14@nudt.edu.cn
[2] Sate Key Laboratory of Cryptology, Beijing 100878, China

Abstract. Outsourcing encrypted data to cloud platforms is widely adopted by users, but there are some problems existing in it: one is that encrypted databases only provide limited types of queries for users. Meanwhile, in the deterministic encryption, users' encrypted data is subject to the frequency attack easily. Besides, users' data privacy is disclosed to cloud platforms when their data is updated. To address these problems, in this paper, we propose an effective encryption scheme on outsourcing data for query on cloud platforms. In our scheme, users' data is encrypted according to all possible queries to meet users' diverse query demands. Furthermore, a double AES encryption method is adopted to cope with the frequency attack existing in deterministic encryption. To protect users' privacy when their data is updated, a neighbor rows exchange method is designed in our scheme. The theoretical analysis and comparative experiments demonstrate the effectiveness of our scheme.

Keywords: Cloud platform · Encrypted database · Possible queries · Double AES encryption · Neighbor rows exchange

1 Introduction

With the maturity of cloud storage technology and the proliferation of cloud platforms, more and more users outsource their data to cloud platforms [2,7,19,22,23]. For one thing, cloud platforms have enough resources to store enormous users' data, which can greatly decrease the storage burden on users' side. For another thing, cloud platforms provide the accessing interfaces for users, where users can access their data easily. However, for users, cloud servers are not trustworthy [1,13,18,21]. If users directly upload their data to cloud platforms, the sensitive information in their data will be exposed to cloud servers, which results in users' privacy disclosure. To avoid this case, users usually encrypt their data before outsourcing it. Since users' encryption keys are private, cloud servers can not decrypt users' encrypted data and users' privacy is protected.

J. Li et al. (Eds.): SPNCE 2019, LNICST 284, pp. 318–332, 2019.
https://doi.org/10.1007/978-3-030-21373-2_24

Common encryption techniques adopted by users include deterministic encryption [3,4], order-preserving encryption [5,6] and homomorphic encryption [8,15]. For deterministic encryption, its algorithm is deterministic, which can always generate the same ciphertext for the same message. Therefore, users can leverage deterministic encryption to realize equality check on encrypted data. Order-preserving encryption is another encryption technique where ciphertexts can preserve the order of plaintexts. This means that users can realize complex operations on encrypted data by order-preserving encryption (e.g., range query). Besides, homomorphic encryption allows users to aggregate their encrypted data on cloud platforms. In this encryption technique, the calculation results on ciphertexts after decryption is same as the working directly on the raw data. Although users' privacy is protected under these encryption techniques, there exists another problem for users: how to query for their encrypted data effectively.

To address the above problem, an early method is proposed in [10] for executing SQL queries over encrypted data by performing approximate filtering at the server and performing final query processing at the client. This method is extended to handle aggregation queries in [11,12]. However, this method consumes a lot of hardware resources. Then, a query-based encryption method is proposed in [17,20]. In this method, based on users' query demands, users' data is encrypted by multiple encryption techniques simultaneously (e.g., deterministic encryption, order-preserving encryption or homomorphic encryption). By this method, users can have rich queries on their encrypted data and avoid some unnecessary post-processes to their data after queries. However, query-based encryption requires users to provide their query sets in advance. In fact, this may be difficult for users because they do not have clear plans for their data. Furthermore, this method fails to defeat the frequency attack in deterministic encryption [14]. To address the second problem, a system named Seabed is proposed in [16]. Seabed adopts additively symmetric homomorphic encryption (ASHE) to implement data encryption and greatly reduce the overhead of data aggregation in the encrypted domain. Meanwhile, Seabed introduces a splayed ASHE method to cope with the frequency attack by splaying sensitive columns to multiple columns. However, splayed ASHE results in heavy storage overheads because multiple new columns are added in original databases. Besides, users' privacy will be leaked out if their data is updated in ASHE.

To overcome the deficiencies in the existing schemes, in this paper, we will propose an efficient encryption scheme on outsourcing data for query on cloud platforms. Specifically, in our scheme, users' data is encrypted according to all possible queries rather than users' query sets. This method is more reasonable than the query-based encryption. Meanwhile, a double AES encryption method is proposed in our scheme, which leverages AES and row identifies of data to encrypt raw data twice to cope with the frequency attack in deterministic encryption. Its cost is much lower than that of splayed ASHE. To implement the dynamic update of users' encrypted data, a neighbor rows exchange method is designed in our scheme. In this method, when the data in some rows faces with the update, the updated value in neighbor rows will exchange their storing positions. This method ensures that users' privacy is not exposed to cloud

servers when their data is updated. In summary, our contributions in this paper are listed as follow.

- We propose an effective encryption scheme on outsourcing data in this paper. In our scheme, the possible encryption method provides all possible queries on encrypted data for users. Meanwhile, the double AES encryption method can effectively defend against the frequency attack. Besides, the neighbor rows exchange method can ensure that users' encrypted data is updated without disclosing users' privacy.
- We present detailed theoretical analysis to demonstrates the effectiveness of the possible encryption method, the double AES encryption and the neighbor rows exchange method.
- The comparative experiments are executed in this paper to show that our scheme has good performance on users' data encryption cost and users' aggregation query cost compared with the existing schemes.

The remaining of this paper is organized as follows. Section 2 describes the system model and the adversary model of our scheme. Section 3 introduces the preliminaries of our scheme including query-based encryption and ASHE. Section 4 presents our scheme in detail. Section 5 provides the theoretical analysis for our scheme and Sect. 6 shows the experimental results. Finally, Sect. 7 concludes this paper.

2 Problem Statement

2.1 System Model

Three entities are involved in our scheme as illustrated in Fig. 1: users, the user proxy and the cloud server. Here, users are the owner of data and they generate their own secret keys for encryption. The user proxy is the middleman who is in charge of transmitting data between users and the cloud server. The cloud server is the entity who stores users' encrypted data. The process of this model is described as follows. First, users share their keys with the user proxy and submit their original data to the user proxy. Then, the user proxy encrypts the data according to all possible queries and outsources the encrypted data to the cloud server. To query the encrypted data, users submit their queries to the user proxy. Then, these original queries are parsed and transformed into the queries used in the encrypted domain by the user proxy. Next, the user proxy submits the new version of queries to the cloud server. Finally, the cloud server returns the query results to the user proxy according to submitted queries. Since the user proxy has users' keys, he can decrypt these results and send them to users. Once users obtain the query results, the whole procedure is finished.

2.2 Adversary Model

In our scheme, we assume that users and the cloud server are semi-honest [9]. That is, they will strictly follow our scheme but they are also curious about

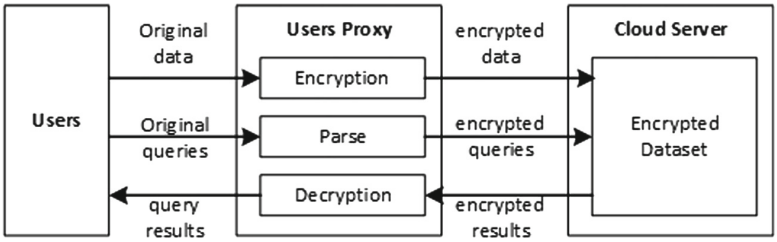


Fig. 1. The framework of our scheme

the other entities' sensitive information. Meanwhile, for users, the user proxy is trustworthy. This means the user proxy will not tamper the transmitted data. Besides, he will not collude with the cloud server to leak out users' privacy.

3 Preliminaries

3.1 Query-Based Encryption

Query-based encryption in [17,20] aims to improve the efficiency of querying encrypted databases and it adopts multiple encryption techniques to achieve this goal, including randomization, deterministic encryption (DE), order-preserving encryption (OPE) and homomorphic encryption (HE). For randomization, two identical values are mapped to different ciphertexts. Thus, ciphertext operations are not allowed under this technique. For DE, two equal values are mapped to the same ciphertext and it allows the equality checks in the encrypted domain. For OPE, the order of plaintexts is preserved after encryption and the range query is allowed. For HE, the ciphertexts are allowed to perform aggregation calculations and the decrypted results are still correct.

In query-based encryption, users first submit their data and query sets to the user proxy. Then, the user proxy selects the corresponding encryption techniques to encrypt users' data according to users' query sets. Finally, the user proxy uploads the encrypted data to the cloud server.

3.2 ASHE

ASHE in [16] assumes that there exists an additive group $\mathbb{Z}_n = \{0, 1, \cdots, n-2, n-1\}$ and a secret key k is shared between the encrypting entity and the decrypting entity. A message $m \in \mathbb{Z}_n$ is encrypted by ASHE as follow.

$$Enc_k(m, i) = ((m - F_k(i) + F_k(i-1)) \ mod \ n, \{i\}) \tag{1}$$

Here, i is an identifier from a set I. $F_k : I \rightarrow \mathbb{Z}_n$ is a pseudo-random function (PRF) that maps an identifier i in I to a value in $\mathbb{Z}_n$ and it is implemented by AES. For ease of presentation, The ciphertexts in ASHE is also denoted as (c, S). Here, c is an element of $\mathbb{Z}_n$ and S is a multiset of identifiers. That is, the

ciphertext $Enc_k(m, i)$ can be also denoted as $(m, \{i\})$. To create the additive homomorphism in ASHE, a special operation $\oplus$ is defined as follow.

$$(c_1, S_1) \oplus (c_2, S_2) = ((c_1 + c_2)\ mod\ n, S_1 \cup S_2) \tag{2}$$

That is, the elements are added together and the multisets of identifiers are combined in the operation $\oplus$. Besides, the ciphertext (c, S) is decrypted as follow.

$$Dec_k(c, S) = (c, \sum_{i \in S}(F_k(i-1) + F_k(i)))\ mod\ n \tag{3}$$

The additive result of two ciphertexts is decrypted by computing:

$$Dec_k(Enc_k(m_1, i_1) \oplus Enc_k(m_2, i_2)) = (m_1 + m_2)\ mod\ n \tag{4}$$

As shown in Eq. (1), the encryption function in ASHE is designed as $(m - F_k(i) + F_k(i-1))$ which has great advantages on data aggregation in the encrypted domain. For example, the ciphertexts of ASHE with consecutive identifiers $\{i, i+1, \cdots, n-1, n\}$ are added together. Due to the clever design of encryption function, the final result of these ciphertexts only contains $F_k(i) - F_k(n)$ and the other F_k is offset during the aggregation. Besides, $F_k(i)$ and $F_k(n)$ are easy to be worked out. Since the F_k is implemented by AES, the total computation overheads are low. Even if the identifiers of ciphertexts are consecutive partly, the overhead of data aggregation in ASHE is still much lower than that in the Paillier Homomorphic Encryption [15] adopted in [17] and [20].

4 Our Scheme

In this section, we will introduce our scheme in detail, which includes three methods: possible query encryption, double AES encryption and neighbor rows exchange.

4.1 Possible Query Encryption

To meet users' diverse query demands for encrypted databases, the possible query encryption in our scheme also adopts the same encryption techniques as the query-based encryption described in Sect. 3.1. However, there are some obvious differences between them. First, DE is implemented directly by AES in the query-based encryption, which can not defend against the frequency attack. In contrary, in the possible query encryption, a double AES encryption is proposed to implement the DE. This encryption method can cope with the frequency attack effectively and will be discussed later. Second, in the possible query encryption, HE is implemented by ASHE rather than Paillier homomorphic encryption. From Sect. 3.2, we can see that the ASHE is much more efficient than Paillier homomorphic encryption in terms of data aggregation in the encrypted domain. Last but not least, instead of users' query sets, all possible

queries for users' data are taken into consideration in the possible query encryption. For one thing, users do not have clear understanding of their data so that they can not provide valid query sets. For another thing, users' query demands may change over time. This means that users' data should not be encrypted by a single encryption technique. Based on such considerations, the thought of all possible queries is adopted in the possible query encryption.

paginationAssume users' original data is presented as Table 1. The user proxy encrypts the data by columns according to the possible query encryption method. First, the user proxy picks out the column of gender and figures out the possible queries on it. Since the equality check is the only operation on this column, the user proxy adopts DE to encrypt it. Similarly, since the equality check and the range query are possible operations on the column of age, the user proxy encrypts it by DE and OPE. For the column of salary, data aggregation is the common operation on it. Therefore, in addition to DE and OPE, HE is also used to encrypt it. Here, HE is implemented by ASHE. Besides, an identifier is introduced to each row of encrypted database due to the application of ASHE. By the possible query encryption method, the encrypted version of users' data in Table 1 is shown as Table 2.

Table 1. The incomes of employees.

···	Gender	Age	Salary	···
···	Male	31	7000	···
···	Female	25	4800	···
···	Female	37	10000	···
···	Male	45	20000	···
···	···	···	···	···

Table 2. The encrypted version of Table 1.

ID	···	DE(Gender)	DE(Age)	OPE(Age)	DE(Salary)	OPE(Salary)	ASHE(Salary)	···
1	···	DE(male)	DE(31)	OPE(31)	DE(7000)	OPE(7000)	ASHE(7000)	···
2	···	DE(female)	DE(25)	OPE(25)	DE(4800)	OPE(4800)	ASHE(4800)	···
3	···	DE(female)	DE(37)	OPE(37)	DE(10000)	OPE(10000)	ASHE(10000)	···
4	···	DE(male)	DE(45)	OPE(45)	DE(20000)	OPE(20000)	ASHE(20000)	···
···	···	···	···	···	···	···	···	···

4.2 Double AES Encryption

For one thing, the frequency attack is a common form of attack in DE. Specifically, the attacker can obtain the occurrence frequency of plaintexts in advance.

If these plaintexts are encrypted by the existing DE, then the attacker can infer the corresponding plaintexts according to the occurrence frequency of ciphertexts. This is because the same plaintexts have the same ciphertexts in DE. For another thing, the ASHE in the possible query encryption introduces an identifier for each row in the encrypted database. By these identifiers, the double AES encryption method in our scheme can defend against the frequency attack.

Double AES encryption method adopts two rounds of AES to encrypt users' data. During the first round of AES encryption, the user proxy encrypts users' data m by using the secret key k shared by users. The encrypted result $Enc_k(m)$ is calculated as follow.

$$Enc_k(m) = AES_k(m) \tag{5}$$

Then, at the second round of AES encryption, the intermediate encrypted result $Enc_k(m)$ is viewed as the secret key of AES to encrypt the identifier i (i is the identifier of the row where m is). The final encrypted result $DE(m)$ is shown as follow.

$$DE(m) = AES_{Enc_k(m)}(i) = AES_{AES_k(m)}(i) \tag{6}$$

Since the identifier of each row is unique, the final encryption results of two identical data in different rows will be different by using double AES encryption. This means the double AES encryption in our scheme can defeat the frequency attack effectively. It is worth noting that we use $AES_k(m)$ to encrypt the identifier i rather than i to encrypt $AES_k(m)$. This is because the cloud server can directly access i and the final encryption result $DE(m)$. Since AES is a symmetric encryption technique, if i is the secret key to encrypt $AES_k(m)$, then the cloud server can directly decrypt $DE(m)$ and obtain $AES_k(m)$. In this case, the cloud server can still launch the frequency attack. In contrast, using $AES_k(m)$ as the key can avoid this because $AES_k(m)$ is unknown to the cloud server and AES is currently not vulnerable to known-plaintext attacks.

To support equality queries on encrypted databases implemented by double AES encryption, the user proxy should submit the intermediate encrypted result $Enc_k(m)$ to the cloud server. Then, the cloud server calculates the $DE(m)$ row by row according to Eq. (6) and performs the equality checks in the encrypted database. If the $DE(m)$ is equal to the data stored in the database, then the data meets users' query demands. Although the cloud server can know the counts of data being queried, he can not infer the corresponding plaintexts by the

Table 3. The double AES encryption version of Table 1.

ID	$\cdots$	Gender	Age	Salary	$\cdots$
1	$\cdots$	$AES_{AES(male)}(1)$	$AES_{AES(31)}(1)$	$AES_{AES(7000)}(1)$	$\cdots$
2	$\cdots$	$AES_{AES(female)}(2)$	$AES_{AES(25)}(2)$	$AES_{AES(4800)}(2)$	$\cdots$
3	$\cdots$	$AES_{AES(female)}(3)$	$AES_{AES(37)}(3)$	$AES_{AES(10000)}(3)$	$\cdots$
4	$\cdots$	$AES_{AES(male)}(4)$	$AES_{AES(45)}(4)$	$AES_{AES(20000)}(4)$	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

frequency attack because the occurrence frequency of other data is unknown to him under the double AES encryption. The double AES encryption version of Table 1 is shown as Table 3.

4.3 Neighbor Rows Exchange

As mentioned before, in the possible query encryption, we adopt the ASHE to implement the HE. However, if users update their data encrypted by ASHE, their data privacy will be leaked out to the cloud server. Assume a user's original data is m_1. According to Eq. (1), it is encrypted by ASHE as follow.

$$Enc_k(m_1, i) = ((m_1 - F_k(i) + F_k(i-1)) \ mod \ n, \{i\}) \tag{7}$$

Then, the $Enc_k(m_1, i)$ is stored in the i-th row of database on the cloud server. Now, this user intends to change the m_1 to m_2. Then, m_2 is encrypted by ASHE as follow.

$$Enc_k(m_2, i) = ((m_2 - F_k(i) + F_k(i-1)) \ mod \ n, \{i\}) \tag{8}$$

The $Enc_k(m_2, i)$ is sent to the cloud server to update the content of the i-th row in database. However, the curious cloud server can disclose this user's data privacy by calculating:

$$\begin{aligned} \Delta m &= Enc_k(m_2) - Enc_k(m_1) \\ &= ((m_2 - F_k(i) + F_k(i-1)) - (m_1 - F_k(i) + F_k(i-1))) \ mod \ n \\ &= m_2 - m_1 \end{aligned} \tag{9}$$

The Δm may indicate the changes in users' salaries or the personnel changes of a company. Anyway, the private information can be easily obtained by the cloud server, which results in the disclosure of users' privacy.

To address this problem, a neighbor rows exchange method is proposed in our scheme. Neighbor rows in this method are defined as two update rows which are adjacent to each other. Assume the data in the i-th, $(i+3)$-th, $(i+9)$-th and $(i+14)$-th row faces with update. Then, the i-th row and the $(i+3)$-th row are neighbor rows. Similar, the $(i+9)$-th row and the $(i+14)$-th row are also neighbor rows. In the case of multiple data updates at the same time, each pair of neighbor rows are divided into an exchange group and their updated value is stored in each other's locations. That is, for the data m_1 to be updated in i-th row and the m_2 to be updated in j-th row, assume i and j are neighbor rows. Then, in neighbor rows exchange method, m_1 and m_2 are divided into an exchange group. Meanwhile, the updated value of m_1 is stored in j-th row and the updated value of m_2 is stored in i-th row. Since the rows where users' data locates have changed after the update, the cloud server can not infer users' sensitive information anymore and users' privacy is protected.

In the case of a single data update, after receiving the i-th user's update data m, the user proxy stores this update data locally and does not modify the corresponding data on the cloud platform for the time being. Once some other users submit their update data, the user proxy will take out m from the local and combines it with other update data. The next steps will the same as those in multiple data updates. It is worth noting that the i-th user can query his update data at any time and the user proxy can ensure the correctness of query result. The above procedure can protect the i-th user's privacy.

In neighbor rows exchange method, the neighbor rows rather than two random rows exchange their stored data is to minimize the location changes of update data, which can take full advantage of the homomorphic property in ASHE. In addition, the validity of neighbor rows exchange method will be demonstrated in the next section. The neighbor rows exchange method is summarized as Algorithm 1.

Algorithm 1. Neighbor Rows Exchange

Input: Users' update data sets $M = \{m_1, \cdots, m_i, \cdots, m_n\}$
Output: The updated database on the cloud platform
1 The user proxy counts the number of update data in M: N_m
2 **if** $N_m > 1$
3 The user proxy devides the update data into multiple neighbor rows
4 The user proxy encrypts the update data according to ASHE where the identifiers of their
neighbor row are used
5 The cloud server stores the encrypted data in their neighbor row
6 **else**
7 The user proxy stores the only update data m_o in the local
8 **if** other update data is submitted from users
9 m_o is combined with these update data
10 repeat the step 3, 4 and 5
11 end

Table 4. Different outsourcing data encryption schemes.

Scheme	Defeat frequence attack	Update	Special
Our scheme	√	√	Possible query encryption
Scheme in [17]	×	√	Query-based encryption
Scheme in [20]	×	√	Query-based encryption
Scheme in [16]	√	×	ASHE

At the end of this section, we compare our scheme with other existing outsourcing data encryption schemes as shown in Table 4.

5 Theoretical Analysis

In this section, we will present the theoretical analysis of our scheme. Specifically, we will respectively analyze the effectiveness of possible query encryption, double AES encryption and neighbor rows exchange.

Theorem 1. *Possible query encryption in our scheme can provide users with richer queries on encrypted data.*

Proof. In the possible query encryption, users' data is encrypted according to all possible queries on their data. Each type of data has its own characteristics: some are suitable for equality checks but range queries and data aggregations are meaningless to them (e.g., gender). DE is enough for this type of data. Some are not only suitable for equality checks but also for range queries and even data aggregations (e.g., salary). This type of data should be encrypted by DE, OPE and HE simultaneously. Considering these cases, possible query encryption encrypts users' data according to the characteristics of the data. This can avoid a lot of unnecessary encryption for users' data.

Compared with query-based encryption, possible query encryption fully excavates the potential characteristics of users' data and provides a more complete query view for users. For one thing, this method does not depend on users' query sets, which can avoid users' subjective limitations. For another thing, users can change their query plans at any time. In this case, their queries will not become invalid. Therefore, possible query encryption is more reasonable than the query-based encryption.

Theorem 2. *Double AES encryption in our scheme can defend against the frequency attack.*

Proof. In the double AES encryption, users' data has a unique identifier and the different identifiers ensure that the equal data has different ciphertexts. Assume m_1 is equal to m_2 and their identifiers are i and j (Here, $i \neq j$). According to Eqs. (5) and (6), m_1 and m_2 is encrypted by double AES encryption as follow.

$$DE(m_1) = AES_{Enc_k(m_1)}(i) = AES_{AES_k(m_1)}(i) \tag{10}$$

$$DE(m_2) = AES_{Enc_k(m_2)}(j) = AES_{AES_k(m_2)}(j) \tag{11}$$

Since m_1 is equal to m_2, then $AES_k(m_1)$ is equal to $AES_k(m_2)$. But i is not equal to j, then $AES_{AES_k(m_1)}(i)$ is not equal to $AES_{AES_k(m_2)}(j)$. That is, $DE(m_1)$ is not equal to $DE(m_2)$.

From the above discussion, we can find two equal data are mapped to different ciphertexts by double AES encryption. This can prevent the attacker from inferring the occurrence frequency of plaintexts from the occurrence frequency of ciphertexts. Therefore, double AES encryption can defend against the frequency attack effectively.

Theorem 3. *Neighbor rows exchange method in our scheme can protect users' privacy when their data is updated.*

Proof. In the neighbor rows exchange method, in the case of multiple data update, the update data of neighbor rows exchanges their storage locations. Suppose m_1 in the i-th row and m_2 in the j-th row are facing updates and their update value are m_1' and m_2' respectively. Meanwhile, the i-th row and the j-th row are the neighbor rows. According to Eq. (1), m_1 and m_2 are encrypted by ASHE as follow.

$$Enc_k(m_1, i) = ((m_1 - F_k(i) + F_k(i-1)) \ mod \ n, \{i\}) \tag{12}$$

$$Enc_k(m_2, j) = ((m_2 - F_k(j) + F_k(j-1)) \ mod \ n, \{j\}) \tag{13}$$

According to the neighbor rows exchange method, m_1' is encrypted by ASHE with the identifier j and m_2' is encrypted by ASHE with the identifier i:

$$Enc_k(m_1', j) = ((m_1' - F_k(j) + F_k(j-1)) \ mod \ n, \{j\}) \tag{14}$$

$$Enc_k(m_2', i) = ((m_2' - F_k(i) + F_k(i-1)) \ mod \ n, \{i\}) \tag{15}$$

To disclose users' privacy, the cloud server will try to obtain Δm_1 by calculating:

$$\begin{aligned} \Delta m_1 &= Enc_k(m_1') - Enc_k(m_1) \\ &= ((m_1' - F_k(j) + F_k(j-1)) - (m_1 - F_k(i) + F_k(i-1))) \ mod \ n \\ &= m_1' - m_1 + (F_k(i) + F_k(j-1) - F_k(j) - F_k(i-1)) \end{aligned} \tag{16}$$

Since the cloud server does not know the key k, he can not work out the $F_k(i) + F_k(j-1) - F_k(j) - F_k(i-1)$ and obtain the Δm_1 according to the Eq. (16). Similar, the cloud server can not obtain Δm_2 either. In the case of a single data update, the only update data m_o is stored in the user proxy temporarily. At this stage, the cloud server can not disclose users' privacy because the encrypted database on the cloud platform has no change. When other update data is submitted, m_o is combined with them and they are updated by following the method of multiple update data. Therefore, at this stage, users' privacy is also protected. In summary, the neighbor rows exchange method in our scheme can protect users' privacy when their data is updated.

6 Experiment

6.1 Experiment Configure

In this section, we will run some simulated experiments to evaluate the performance of our scheme. These experiments are run on a laptop with Intel i5-5200U CPU @ 2.20 GHz and 4GB RAM. Meanwhile, the operating system is Windows 10 and the programming language is Java 1.8.0. To implement OPE and Pallier

Homomorphic encryption, the opetoolbox[1] and the pailliertoolbox[2] are used in our experiments. Meanwhile, AES and AHSE are also implemented in our experiments. In addition, the experimental data is synthetic which includes 21 users. Each user has gender data, age data and salary data, as shown in Table 1. We will evaluate the performance of our scheme from two aspects: the encryption cost of the user proxy and the aggregation query cost of users.

6.2 The Encryption Cost of the User Proxy

In this experiment, we will compare the encryption cost of the user proxy in our scheme with that of the user proxy in [17]. Concretely, in our scheme, the user proxy needs to encrypt users' data by double AES encryption, OPE and AHSE, as shown in Table 2. While In [17], the user proxy adopts DE, OPE and Pallier Homomorphic encryption to encrypt users' data. Meanwhile, in this experiment, users' query sets in [17] includes all possible queries. To observe the encryption cost of the user proxy, we measure the encryption time of the user proxy under the different number of users which varies from 3 to 21. Repeat 10 times for each experiment and calculate the averages. The experimental result is shown as Fig. 2.

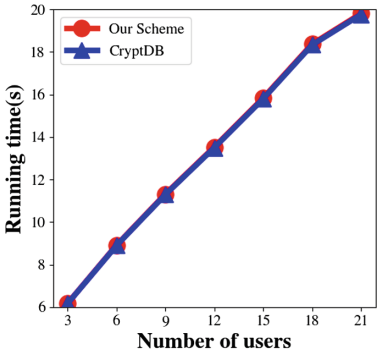
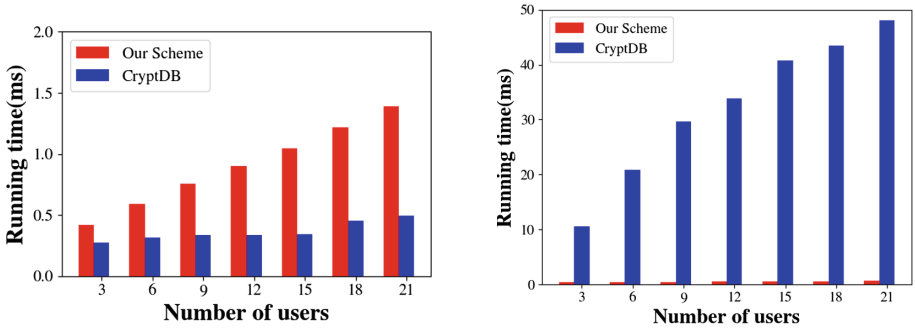


Fig. 2. The user proxy's encryption cost

From Fig. 2, we can find the user proxy in the two schemes has similar encryption cost. Through our analysis, we find OPE is most time-consuming in all of the encryption techniques mentioned in this paper. OPE in our scheme has the same implementation as OPE in [17], which results in the similar encryption cost of the user proxy in the two schemes. To further compare the DE cost and HE cost of the user proxy in the two schemes, we use double AES encryption, AES, ASHE and Paillier homomorphic encryption to encrypt users' salary data. Similarly, in these experiments, the number of users is varied from 3 to 21. Each experiment is repeated 10 times and the averages are calculated. The experimental results are shown as Fig. 3.

[1] https://github.com/ssavvides/jope.
[2] http://cs.utdallas.edu/dspl/cgi-bin/pailliertoolbox.



(a) Deterministic encryption (b) Homomorphic encryption

Fig. 3. The encryption cost of the user proxy

From Fig. 3(a), we can find the deterministic encryption cost of the user proxy in our scheme is higher than that in [17]. This is because double AES encryption in our scheme is implemented by two rounds of AES while the deterministic encryption in [17] is implemented by one round of AES. To defend against the frequency attack, in our scheme, the extra encryption cost for the user proxy is acceptable. From Fig. 3(b), we can find the homomorphic encryption cost of the user proxy in our scheme is much lower than that in [17]. This is because ASHE used in our scheme is implemented by the symmetric encryption AES. Compared with the asymmetric encryption (i.e., Paillier homomorphic encryption) used in [17], ASHE is obviously much more efficient.

6.3 Users' Aggregation Query Cost

In this experiment, assume users intend to query the sum of their salaries. This is a typical aggregation query in the encrypted domain, which is supported by our scheme and [17]. To compare the users' aggregation query cost in the two schemes, we measure users' query time under the different number of users which varies from 3 to 21. Each experiment is repeated 10 times and the averages are calculated. The experimental result is shown as Fig. 4.

From Fig. 4, we can find that users' aggregation query cost in our scheme is much lower than that in [17]. This is because users' data is encrypted by ASHE in our scheme. When aggregating the encrypted data, many calculation items are automatically offset in our scheme, as discussed in Sect. 3.2. In contrast, in [17], since users' data is encrypted by Paillier homomorphic encryption, many exponent operations are executed when aggregating the encrypted data. This results in huge time cost. Therefore, users' aggregation query in our scheme is much more efficient than that in [17].

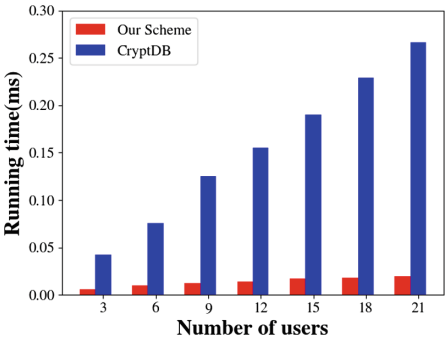


Fig. 4. The users' aggregation query cost

7 Conclusion

In this paper, we propose an effective scheme to support for query on encrypted databases on cloud platforms. In our scheme, the possible query encryption provides a complete query view for users and users can have more query choices. Meanwhile, the double AES encryption can defend against the frequency attack in deterministic encryption. Besides, the neighbor rows exchange method can protect users' privacy when their data is updated on encrypted databases. The theoretical analysis in this paper demonstrates the effectiveness of the three methods of our scheme. Meanwhile, The comparative experiments show that our scheme has good performance on the encryption cost of the user proxy and users' aggregation query cost.

References

1. Almorsy, M., Grundy, J., Müller, I.: An analysis of the cloud computing security problem. arXiv preprint arXiv:1609.01107 (2016)
2. Baldwin, P.K.: Cloud Services. Encyclopedia of Big Data, pp. 1–4 (2017)
3. Bellare, M., Boldyreva, A., O'Neill, A.: Deterministic and efficiently searchable encryption. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 535–552. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74143-5_30
4. Bellare, M., Fischlin, M., O'Neill, A., Ristenpart, T.: Deterministic encryption: definitional equivalences and constructions without random oracles. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 360–378. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85174-5_20
5. Boldyreva, A., Chenette, N., Lee, Y., O'Neill, A.: Order-preserving symmetric encryption. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 224–241. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01001-9_13
6. Boldyreva, A., Chenette, N., O'Neill, A.: Order-preserving encryption revisited: improved security analysis and alternative solutions. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 578–595. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_33

7. Calder, B., et al.: Windows azure storage: a highly available cloud storage service with strong consistency. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, pp. 143–157. ACM (2011)
8. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. Inf. Theory **31**(4), 469–472 (1985)
9. Goldreich, O.: Foundations of cryptography: basic applications. J. ACM **10**(509), 359–364 (2004)
10. Hacigümüş, H., Iyer, B., Li, C., Mehrotra, S.: Executing SQL over encrypted data in the database-service-provider model. In: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, pp. 216–227. ACM (2002)
11. Hacıgümüş, H., Iyer, B., Mehrotra, S.: Efficient execution of aggregation queries over encrypted relational databases. In: Lee, Y.J., Li, J., Whang, K.-Y., Lee, D. (eds.) DASFAA 2004. LNCS, vol. 2973, pp. 125–136. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24571-1_10
12. Hacıgümüş, H., Iyer, B., Mehrotra, S.: Query optimization in encrypted database systems. In: Zhou, L., Ooi, B.C., Meng, X. (eds.) DASFAA 2005. LNCS, vol. 3453, pp. 43–55. Springer, Heidelberg (2005). https://doi.org/10.1007/11408079_7
13. Li, J., Liu, Z., Chen, X., Xhafa, F., Tan, X., Wong, D.S.: L-ENcDB: a lightweight framework for privacy-preserving data queries in cloud computing. Knowl.-Based Syst. **79**, 18–26 (2015)
14. Naveed, M., Kamara, S., Wright, C.V.: Inference attacks on property-preserving encrypted databases. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 644–655. ACM (2015)
15. 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). https://doi.org/10.1007/3-540-48910-X_16
16. Papadimitriou, A., et al.: Big data analytics over encrypted datasets with seabed. In: OSDI, pp. 587–602 (2016)
17. Popa, R.A., Redfield, C., Zeldovich, N., Balakrishnan, H.: CryptDB: protecting confidentiality with encrypted query processing. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, pp. 85–100. ACM (2011)
18. Singh, A., Chatterjee, K.: Cloud security issues and challenges: a survey. J. Netw. Comput. Appl. **79**, 88–115 (2017)
19. Stanek, J., Sorniotti, A., Androulaki, E., Kencl, L.: A secure data deduplication scheme for cloud storage. In: Christin, N., Safavi-Naini, R. (eds.) FC 2014. LNCS, vol. 8437, pp. 99–118. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45472-5_8
20. Tu, S., Kaashoek, M.F., Madden, S., Zeldovich, N.: Processing analytical queries over encrypted data. In: Proceedings of the VLDB Endowment, vol. 6, pp. 289–300. VLDB Endowment (2013)
21. Wang, C., Ren, K., Yu, S., Urs, K.M.R.: Achieving usable and privacy-assured similarity search over outsourced cloud data. In: INFOCOM, 2012 Proceedings IEEE, pp. 451–459. IEEE (2012)
22. Wu, J., Ping, L., Ge, X., Wang, Y., Fu, J.: Cloud storage as the infrastructure of cloud computing. In: 2010 International Conference on Intelligent Computing and Cognitive Informatics (ICICCI), pp. 380–383. IEEE (2010)
23. Yang, J., He, S., Lin, Y., Lv, Z.: Multimedia cloud transmission and storage system based on Internet of Things. Multimedia Tools Appl. **76**(17), 17735–17750 (2017)