A Solution for Privacy-Preserving Data Manipulation and Query on NoSQL Database

Guo Yubin[a], Zhang Liankuan[b], Lin Fengren[a], Li Ximing[a,*]
[a] College of Informatics, South China Agricultural University, Guangzhou 510640, China
Email: {guoyubin,linfengren,liximing}@scau.edu.cn
[b] College of Science, South China Agricultural University, Guangzhou 510640, China
Email: zhangliankuan@scau.edu.cn

Abstract— **Privacy of data owners and query users is vital in modern clouding data management. Many researches have been done on cloud security, but most of them are focused on the privacy of data owners or of query users separately. How to protect the privacy of the data owners and users simultaneously is a great challenge. In this paper, a solution of data storage and query protocol based on classical homomorphic encryption scheme is given to preserve privacy of both data owners and query users. Our main efforts are put on NoSQL database which is less structural than relational database. Storage and indexing structure on NoSQL database, query protocol are proposed, and algorithms for updating and querying are also given. To implement our solution, Berkley DB, an excellent storage solution for NoSQL database is chosen and data are encrypted/decrypted using Elgamal and Paillier encryption system, using basic Java package. Experiments are done under different parameters in order to achieve better efficiency.**

Index Terms— **NoSQL; cloud data management; privacy preserving**

I. Introduction

Today cloud computing and data outsourcing provide much convenience for kinds of enterprises. For instance, enterprises can concentrate on their main business while outsourcing their complex data management and query service to service providers in cloud. These service providers in cloud focus on data management, and provide high quality service. But in such kind of computing pattern, a bottleneck, privacy preserving of data owners and query users, seriously restricts progress of cloud computing.

Consider environment illustrated in Fig. 1, data owners outsource their data and query services, but the data is private assets of them and should be protected against the service providers and querying users in some extent. On one hand, data owner can update, query and authorize access of data, while the service providers in cloud should know nothing about especially detailed data, and query users should know not more than the exact answers for what she/he is querying. On the other hand, query users need to query data from cloud, but the query might disclose sensitive information, behavior patterns of the user. For example, when Alice searches a website, such as Facebook, for friends who share the similar backgrounds (e.g., age, education, home address) with her, she should not disclose the query that involves her own details to the cloud. Privacy of data owners and query users are defined as data privacy and user privacy respectively.

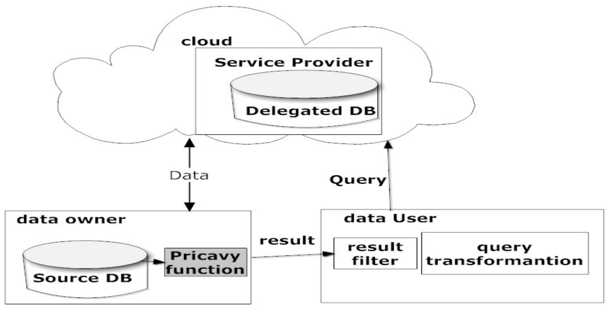

Figure 1: Architecture of Data Service on Cloud

A. Related works

For data privacy, the most general solution in recently research papers are encryption that means data deposited to service provider must be encrypted to avoid information leakage. Agrawal et al [5] proposed an order preserving encryption scheme (OPES) by which indexes can be built directly on ciphertext. OPES can handle directly (without decryption) any interesting SQL query types, except SUM and AVG. But order preserving would leak information about data, and is not a good solution to privacy preserving. Hacigumus et al [9] proposed to handle SUM and AVG using homomorphic encryption function in the database context. Ayman Mousa et al. [14] uses classic REA, a symmetric encryption algorithm, to encrypted data respectively, and in this way the query processing performance is assured, but information leakage and query privacy are not considered. Privacy homomorphism [17] is encryption transformations which map a set of operations on cleartext to another set of operations on ciphertext. In essence, privacy homomorphism enables complex computations (such as distances) based solely on ciphertext, without decryption. Unfortunately, as pointed

Partially supported by National Science Foundation of China (61103232, 61272402, 61202294),Guangdong Provice Nature Science Foundation (10351806001000000, 10151064201000028), Guangdong Science Technology Plan Project (2010B010600046, 2011B090400325), Guangzhou Science Technology Plan Project (12C42101606).
* Contact author.

out by Mykletun and Tsudik [15], its encryption scheme is insecure, demonstrated by its vulnerability to a basic ciphertext-only attack. However, for encrypted database, efficiency of query processing is a great challenge.

In [8], [10], [11], user privacy is considered together with data privacy. Yonghong Yu and Wenyang Bai discussed how to enforce data privacy and user privacy over outsourced database service in [18]. Hu et al. [11] proposed a solution based on secure traversal framework and privacy homomorphism based encryption scheme. Yong Hu et.al in [12] constructed an intelligent analysis model for outsourced software. And secure protocols for processing k-nearest-neighbor queries (kNN) on R-tree index is given. In the authors following work [10], they integrated indexing techniques with secure multi-party computation (SMC) based protocols to construct a secure index traversal framework. In this framework, the service provider cannot trace the index traversal path of a query during evaluation, and thus keep privacy of users. Their protocols for query are complex, and hard to implement. The thought of composed key in index is directly prompted by Tingjian Ge's work [8]. In his paper, keyword are composed together to improve the efficiency of aggregation operations in database. It is intuitively that addition of keywords in block is dramatically efficient than adding them one by one. But the authors have not considered range or single key search. As to protecting data privacy and user privacy, we use the block structure to hide real structure of keys in index. And key search is efficient for key comparison can be done $k-in-1$ where the k is key number in a single block, that is decided by block and key size.

B. Our contribution

For data privacy and user privacy, a solution of data storage, manipulation and query is presented in this paper. In main database files, data are stored in key/value pair which is a typical NoSQL storage structure and are encrypted with Elgamal homomorphic encryption scheme. Keys in index are ciphertext of combinations of real keys in big blocks (in our experiments, one block is set to 1024 bits), which are encrypted with Paillier encryption scheme [16] which is an additive homomorphic crypto-system. When a key is queried, comparison can be done on ciphertext in blocks that improves efficiency of query. Protocols of data manipulation and query among data owner, service provider and querying user is given. Algorithm for data updating and querying are implemented to verify usefulness of the solution. As to implementation of our solution, Berkley DB, a typical key/value pair model database, is chosen to construct a prototype system. It is an excellent storage solution for NoSQL database for its high efficiency and convenience.

C. Outline of the paper

The rest of the paper is organized as follows. Section II provides background information on homomorphic encryption scheme, NoSQL database and index. Section III describes the query protocol and main algorithms, security analysis is also given in the section. Performance and analysis of experiments results are shown in section IV. Finally, section V gives conclusions and future works.

II. Preliminary

A. Homomorphic Encryption

Homomorphic encryption allows specific types of computations to be carried out on ciphertext and obtains an encrypted result which is ciphertext of the result of operations performed on the plain text. The additive homomorphic property of a homomorphic cryptosystem is $Enc(a) \times Enc(b) = Enc(a + b)$, where a and b are two plain text message blocks, and Enc is the encryption function that takes a plaintext message block (and an encryption key) and returns the ciphertext block. Thus, in the above equation, $+$ operates on the plain text, and $\times$ operates on the ciphertext. An example of such an encryption scheme is the Paillier system. Elgamal encryption scheme [6], [7] is multiplicative homomorphic with $Enc(a) \times Enc(b) = Enc(a * b)$ where a, b, Enc and $\times$ share the same meanings with formula above, and operation $+$ is multiple operation on the plaintext. The Legion of the Bouncy Castle [1] provided open source libraries of Java and c# Cryptography Architecture. Both Paillier and ElGamal encryption scheme have great practical implications on the outsourcing of private computations, such as, in the context of cloud computing and outsourcing [13].

B. NoSQL database and index

NoSQL database is defined as the next Generation Databases mostly because of the following characteristics: being non-relational, distributed, open-source and horizontally scalable [2]. The concept NoSQL is prompted by Carlo Strozzi in 1998 [3], and the current NoSQL movement beginning from 2009 often more characteristics apply such as: schema-free, easy replication support, simple API, eventually consistent / BASE (not ACID), a huge amount of data and more. From NoSQL Data Modeling Techniques [4] data model for NoSQL Database can be cataloged into key/value or tuple store, Bigtable style databases, Document databases, and graph databases. Berkeley DB is a robust solution on which to build a NoSQL system, and the storage system of its key/value pair is more efficient than other database. That is the reason for us to choose it as our base database in our experiments.

III. Solution for storage and updating

In this section some symbols are defined for simplification. $owner$ means data owner, sp is service provider in cloud, and $user$ data user for query. We use Paillier crypto-system to encrypt keys and values in NoSQL database. System parameter is taken as n. $Enc(m, pk)$ is the function to encrypt plaintext m with public key

pk, and $Dec(c, sk)$ function to decrypt ciphertext c with private key sk. Denote the public key and secret key of data owner as (pk_{owner}, sk_{owner}). Denote the public key and secret key of user $user_i$ as $(pk_{user_i}, sk_{user_i})$. All data are encrypted using homomorphic encryption algorithm, each one of $owner$, sp and $user$ publishes his public key and uses the private key to decrypt ciphers.

A. data storage

In our solution, data are stored in database, and in key/data model. That means each tuple is composed of one key and one data, the key and data are encrypted respectively. To construct indices of data, several keys are composited together to form a 1024 binary bits block. Number of keys in one block is determined by length of key. Let l be length of key, then the number of keys in one block is $\lfloor (1024)/(l+1) \rfloor$ and one bit is added to each key to deal with overflow.. Let $k_1, k_2, \ldots, k_m$ be keys which will be combined together, then the key block in index k can be computed as follows:

$$k = (k_1 << (n-1) * (l+1)) \cup$$
$$(k_2 << (n-2) * (l+1)) \cup \ldots \cup (k_n)$$

In this formula, $String << n$ is left transferring operation which will left transfers bit-string $String$ for n bits. To k, only one encryption operation is needed. The block of keys in plaintext and ciphertext are illustrated in Fig. 2.

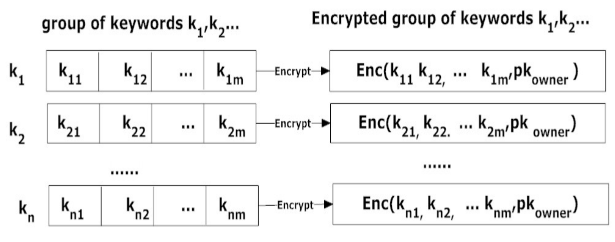
group of keywords k1,k2...
Encrypted group of keywords k1,k2...
k1
k11
k12
...
k1m
Encrypt
Enc(k11 k12, ... k1m,pkowner)
k2
k21
k22
...
k2m
Encrypt
Enc(k21, k22. ... k2m,pk owner)
......
......
kn
kn1
kn2
...
knm
Encrypt
Enc(kn1, kn2, ... knm,pkowner)

Figure 2: the structure of index keys

In the index, the pointers pointing to tuples in main database file are composed in according to the key blocks. Algorithm for index construction is presented as Algorithm. 1.

B. Data manipulation

When a data owner outsources his data to service provider in cloud, some preparation works must be done. At initial stage, data are encrypted in key/value pair, indices are constructed with Algorithm 1. Then, service provider uses the indices to improve query efficiency. And maintenance of data and indices are task of the data owner. When there are some tuples to be inserted or deleted, data owner must arrange an insert or delete operation on main database files which is stored at service provider, reindex the changed data and sent the index to service provider.

Algorithm 1: Index constructing algorithm

```
Input: key, pointer*
Output: index file f
1  $f$= New(file );
2  $l$= length of key;
3  $n = int(1024/(l+1))$;
4  while not end of input do
5      $i = 0$;
6      $m_1 = m_2 = 0$;
7      while $i <= n$ and not (end of input) do
8          $m_1 = (m_1 << (l+1) \vee key) \mod n^2$ ;
9          //$n_2$ is sqare of $n$ in encrypting algorithm;
10         $m_2 = m_2 << l' \vee \mod n^2$ ;
11         //l' is the length of key in database main
           files;
12         i++;
13     Add $< Enc(m_1, pk_{owner}), Enc(m_2, pk_{owner}) >$
       into index file f;
14 return $f$;
```

For the data encrypted under homomorphic encryption scheme, order of data is not kept in ciphertext. In order to simplify data maintenance, data in plaintext can be ordered in time, which means the data owner need to attach the data when it is put into database. When a fresh tuple is going to be inserted, it must be at the end of database files. When a tuple is going to be deleted, it will not be deleted physically, it will only be marked as ‘deleted’. The ‘deleted’ data will not appear in any query results, and can be rearranged after a certain amount of manipulation. Let updating operation be done by deleting and inserting operation, data inserting and deleting algorithm are given as Algorithm. 2 and Algorithm. 3 respectively.

C. Data querying protocol

In this section, we illustrate the query details first, and then propose the query protocol. As the keys are in block, the queried key must be repeated n times and together to meet each key in a key block. Fig. 3 presents an example of query procedure. As shown in Fig. 3, there are 4 keys in a key bloc. Then in plaintext keys $0, 1, 10, 2$ is combined into a big integer as 000001010002, and then encrypted into ciphertext. As to the queried key, the additive reverse is used to composite into block, then the queried block is $-002 - 002 - 002 - 002$ when the queried key equals to 2. We use additive reverse of the queried key to transfer minus into addition operation in plaintext, then multiplication on ciphertext can be used to implement subtraction, according to the additive homomorphic property of Paillier encryption scheme.

Algorithm 2: Data inserting algorithm

```
Input: $< key, value >$ //a new tuple that is inserting into the database
Output:
1  //Insert a new tuple $< Enc(key, pk_{owner}), Enc(key, pk_{owner}) >$ into database.;
2  $T =< Enc(key, pk_{owner}), Enc(value, pk_{owner}) >$;
3  Attach $T =< Enc(key, pk_{owner}), Enc(value, pk_{owner}) >$ to the end of main database file;
4  //index maintaining;
5  for each index do
6      Construct its new keys $key'$;
7      $t =< Enc(key', pk_{owner}), Enc(key, pk_{owner}) >$;
8      if the last data block of index file is not full then
9          $(c_{key}, c_{value})$= ciphertext of last block ;
10         $c_{key} = c_{key}^{l+1} + Enc(key, pk_{owner}) \mod n^2$;
11         $c_{value} = c_{value}^{l+1} + Enc(value, pk_{owner}) \mod n^2$;
12         Write $(c_{key}, c_{value})$ back to file;
13     else
14         Attach $T =< Enc(key, pk_{owner}), Enc(value, pk_{owner}) >$ to the end of index file.
15 Send the index back to $sp$ and replace old one ;
```

Algorithm 3: Data deleting algorithm

```
Input: $t =< Enc(key, pk_{owner}), Enc(value, pk_{owner}) >$, database(main file and indices)
Output: database(main file and indices) after deletion
1  Select $t =< Enc(key, pk_{owner}), Enc(value, pk_{owner})) >$ in main file and indices.;
2  //deleting from main database file, is done by $sp$;
3  Select the last tuple of main database file as $t_1$;
4  Replace $t$ with $t_1$;
5  //deleting from index;
6  for each index do
7      Construct key $key'$;
8      Let $t =< Enc(key', pk_{owner}), Enc(key, pk_{owner}) >$;
9      Find the block $b_t$ in which $t$ is the $ith$ key;
10     Find the last key block $b_l$ in in which key $t_l$ is in $b_l$ ;
11     $b_t = b_t / t^{i*(l+1)} * t_l^{i*(l+1)} \mod n^2$ ;
12     $b_l = b_l / t_l \mod n^2$ ;
13 Send the index file to $sp$ and replace old one;
```

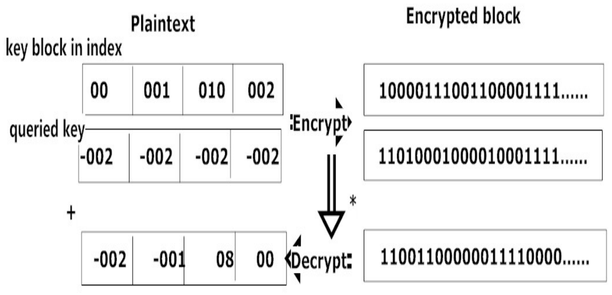

Figure 3: An example of querying

We know, the query algorithm is oblivious for data user. The data user encrypted additive reverse of queried key at first, and send it to service provider. Then the service provider extends it into queried block, and chooses proper index to multiple the queried block with a key block, and as follows, the product is sent to data owner one by one. When the data owner receives the product, it decrypts the cipher and decomposes the plaintext to find which one is 0, which means the according key in the block is equal to queried key. The serial numbers of the key are sent back to the service provider, service provider can get the key in main data file and get queried data for the data owner. This process will terminated when the queried key is found or all blocks in the index is searched. To database applications, querying is the most common operation. In our solution, all of the three roles, service provider, data owner and data user must participant in the querying process. To preserve both data owner and user privacy, querying process is more complex than in traditional database system. Fig. 4 presents the querying protocol in detail.

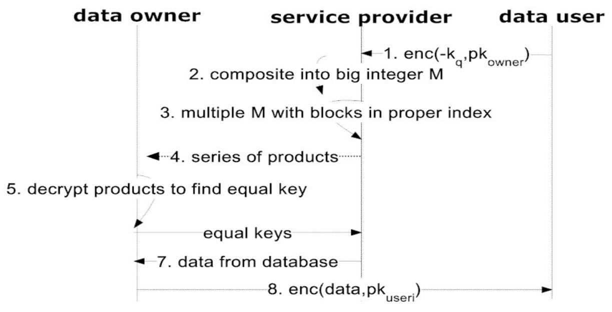

Figure 4: Query protocol

D. Security analysis

In this protocol, data user prompts a query by encrypting additive inverse of queried key with public key of data owner and sends it to the service provider. The second step starts when the service provider receives a query request. The service provider chooses proper index, and then a 1024 bits big integer M is composed by repeating the queried key several times. Then multiplies M by each key block respectively, and sends the results to data owner. When the data owner receives the products, he decrypts the cipher and decomposes the plaintext to find 0 in each product. This procedure can be ended when the key is found or all products are searched. Then the serial number of equal key is sent back to the service provider, and all the queried data are sent to the data owner. At last step, the data owner decrypts the data with his owner private key and encrypts it with public key of the data user and sent the result to him.

During the query process, additive reverse of the queried keyword is encrypted before sending to the service provider, and the queried data is sent by data owner, therefore the service provider can get no information about what the data user is querying on the database. Security can also be enforced by adding disturbing data when the data owner requests query data from service provider. And as to the data owner, during the query process, only product of the queried block and index key blocks are received and decrypted to find the order of equal ones, while the queried key is kept invisible. The data owner do not know which index is chosen and cannot deduce what the data user is querying. It is obvious that we cannot complete query without leaking no information about the user and what she is querying. At least, queried result must be sent back to her. What we really want to do and can do is to limit the information leakage as much as possible. From the analysis above, we can see, confidence of data owner can surely be protected for homomorphic encryption scheme is used. Data privacy and user privacy are all kept by the scheme we propose to some extent.

IV. Experiments

A. Setup

Our experiments are conducted on BDB database system on Windows 7. We implements the generalized Paillier system with basic Java package, and the Elgamal scheme is from open source library of bouncy castle [1]. All experiments of the solution are implemented in Java with JDK 1.7 and the prototype system are run on personal computer with Intel 2Ghz processor and 2GB memory.

B. Experiments design and analysis

A series of experiments have been done to test efficiency of our solution. Some vital parameters, like quantity of data, thread number, and length of key, have been changed to find difference. Fig. 5 illustrates query efficiency of our solution. In this figure, x axis is number of tuples which is set to be 20000, 30000, 40000, 50000, 75000 and 100000, while y axis is average time used for a single value query. And we can get 3 curves when the length of key is set to 5, 10 and 20 decimal bits respectively. (as in binary, it should be 1, 2, and 4 bytes approximately.) On the whole, query efficiency is much better when length of key is not so long. The reason lies in that when the key is short, more keys can be put into one single block, therefore a comparison on block is equivalent to much more comparisons on single key.

In Fig. 6, tuple number is set to 50,000, to illustrate query efficiency variation on thread number and key

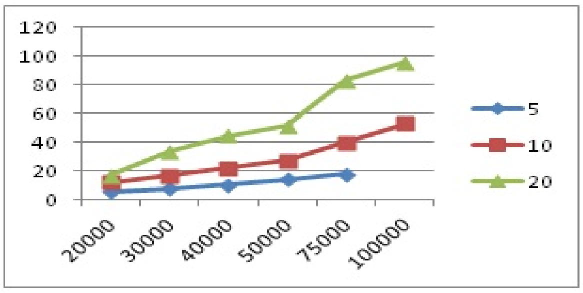

Figure 5: Query efficiency on data quantity, and length of key

length. In this figure thread number is x axis, and there 3 curves are with different key length. It is oblivious that the best value of thread number is 4. When thread is few, computing power of CPU cannot be fully used. And when threads are too much, communication and context change decrease the efficiency of the solution. Thread number is vital for most service providers in clouds. And parallel process of key words comparison can improve query performance drastically. In our prototype system, block comparison is divided into several parts simply, query efficiency can be heightened further with sophisticated technology of parallel programming. Note that thread number is a hardware-depended parameter.

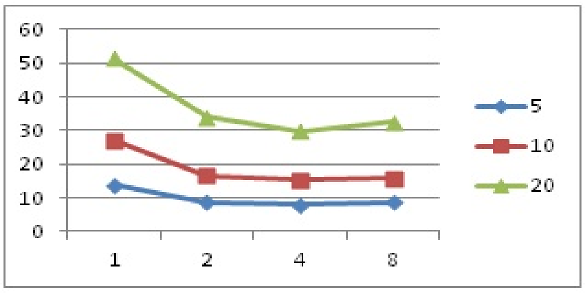

Figure 6: Query efficiency on thread number and key length

Fig. 7 illustrates efficiency variation according to thread number and tuple number when the key size is fixed to 10. From the figure, we can see, query time increases more quickly when tuples are more than 50,000. And it means performance of our solution is more better to middle scale database.

V. Conclusion

Homomorphic encryption scheme provides a good solution to privacy preservation for database system. We present a storage solution for NoSQL database using homomorphic encryption algorithms. Protocol of data querying is proposed, and algorithms for data manipulation are given also. In indices, keys are composed into big blocks to improve the performance of encryption and decryption, therefore accelerate the process of data manipulation and

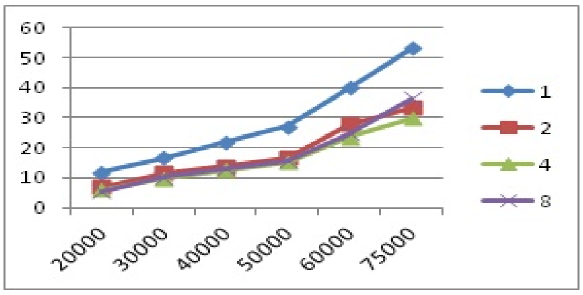

Figure 7: Query efficiency on thread number and tuple number

query. Although Paillier and Elgamal encryption scheme is not so efficient comparing to symmetric encryption schemes like DES and SHA. But it is good enough for some cases that users pay more attention on information security than computation performance.

Future work includes improving efficiency of the system and extending system functionality, such as extended query on range, aggregation, and join.

References

[1] The legion of the bouncy castle. http://www.bouncycastle.org/, 2013. [Online; accessed 10-Jan-2013].

[2] Non - relational universe. http://nosql-database.org/, 2013. [Online; accessed 10-Jan-2013].

[3] Nosql, a relational database management system. http://www.strozzi.it/cgi-bin/CSA/tw7/I/en-US/nosql/Home\%20Page, 2013. [Online; accessed 10-Jan-2013].

[4] Nosql data modeling techniques. http://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-techniques/, 2013. [Online; accessed 10-Jan-2013].

[5] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, SIGMOD '04, pages 563–574, New York, NY, USA, 2004. ACM.

[6] Haipeng Chen, Xuanjing Shen, and Yingda Lv. An implicit elgamal digital signature scheme. *JSW*, 6(7):1329–1336, 2011.

[7] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO*, pages 10–18, 1984.

[8] Tingjian Ge, Stanley B. Zdonik, and Stanley B. Zdonik. Answering aggregation queries in a secure system model. In *VLDB*, pages 519–530, 2007.

[9] Hakan Hacgm, Bala Iyer, and Sharad Mehrotra. Efficient execution of aggregation queries over encrypted relational databases. In YoonJoon Lee, Jianzhong Li, Kyu-Young Whang, and Doheon Lee, editors, *Database Systems for Advanced Applications*, volume 2973 of *Lecture Notes in Computer Science*, pages 125–136. Springer Berlin Heidelberg, 2004.

[10] Haibo Hu and Jianliang Xu. Non-exposure location anonymity. In Yannis E. Ioannidis, Dik Lun Lee, and Raymond T. Ng, editors, *ICDE*, pages 1120–1131. IEEE, 2009.

[11] Haibo Hu, Jianliang Xu, Chushi Ren, Byron Choi, and Byron Choi. Processing private queries over untrusted data cloud through privacy homomorphism. In *ICDE*, pages 601–612, 2011.

[12] Yong Hu, Xizhu Mo, Xiangzhou Zhang, Yuran Zeng, Jianfeng Du, and Kang Xie. Intelligent analysis model for outsourced software project risk using constraint-based bayesian network. *JSW*, 7(2):440–449, 2012.

[13] Daniele Micciancio. A first glimpse of cryptography's holy grail. page 96, 2010.

[14] Ayman Mousa, Elsayed Nigm, El-Sayed El-Rabaie, Osama S. Faragallah, and Osama S. Faragallah. Query processing performance on encrypted databases by using the rea algorithm. pages 280–288, 2012.

[15] Einar Mykletun and Gene Tsudik. Aggregation queries in the database-as-a-service model. In Ernesto Damiani and Peng Liu, editors, *Data and Applications Security XX*, volume 4127 of *Lecture Notes in Computer Science*, pages 89–103. Springer Berlin / Heidelberg, 2006.

[16] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[17] R. Rivest, L. Adleman, and M. Dertouzos. On data banks and privacy homomorphisms. pages 169–177. Academic Press, 1978.

[18] Yonghong Yu and Wenyang Bai. Enforcing data privacy and user privacy over outsourced database service. *JSW*, 6(3):404–412, 2011.

GuoYubin Received Ph. D. from South China University of Technology in 2007. She is now lecturer in South China Agricultural University. Her research interests include Database theory and technology, cryptography and network computing.

Zhang Liankuan Received Ph. D. from South China Agricultural University in 2012. He is now lecturer in South China Agricultural University. His research interests include Database theory, technology and network computing.

Lin Fengren He is now Bachelor student in South China Agricultural University. His research interests include Database theory and technology.

Li Ximing Received Ph.D. degree from College of Informatics, South China Agricultural University, Guangzhou, Guangdong, China, in 2011. His current research interests include computer theory and cryptography.