DEMO: Query Encrypted Databases Practically

Dongxi Liu Shenlu Wang[*]
CSIRO ICT Centre, Marsfield, NSW 2122, Australia
dongxi.liu@csiro.au, shenlu.wang@gmail.com

ABSTRACT

The cloud database services are attractive for managing outsourced databases. However, the data security and privacy is a big concern hampering the acceptance of cloud database services. A straightforward way to address this concern is to encrypt the database, but an encrypted database cannot be easily queried.

In this demo paper, we demonstrate that aggregate SQL queries with range conditions can be performed efficiently over encrypted databases, without decrypting the databases first, by using our new homomorphic encryption scheme. The techniques in this paper can be applied to existing Database Management Systems (DBMSs). Moreover, the techniques do not need to predetermine the maximum sum and number of data in one database table column. These features make our technologies suitable to manage long-standing and large encrypted databases.

Categories and Subject Descriptors

E.3 [**Data**]: Data Encryption; H.2 [**Information Systems**]: Database Management

General Terms

Security

Keywords

Database, Homomorphic Encryption, Order-Preserving Index, SQL Query

1. INTRODUCTION

Cloud database services, such as Amazon Relational Database Service (RDS) and Microsoft SQL Azure, are attractive for enterprises to outsource their databases. The database services allow enterprises to deploy their databases quickly without making the large upfront investment on their proprietary hardware and software, hence reducing the total cost of ownership. Moreover, due to the elasticity of cloud database services, an enterprise can dynamically increase or decrease the compute resources allocated to its databases according to its business requirements.

When a database is deployed into a public database service, the service provider has the complete physical control over the database. The data in the database might be improperly accessed by the service provider accidentally or intentionally, or by attackers who compromise the database service platforms. Hence, though attractive, cloud database services may not be fully exploited if the problem of data privacy and security is not satisfyingly addressed [1].

For cloud database services, a straightforward approach to addressing the security and privacy problem is to encrypt data before they are sent to the cloud. By this way, the service provider or an attacker only can see meaningless encrypted data. However, after being encrypted, a database cannot be easily queried. When a database is large, it is not acceptable to decrypt the entire database before performing each query because the decryption might be very slow. On the other hand, if the decryption is done on the cloud, the decrypted database is again at the risk of having its security and privacy breached. Ideally, a query should be executed directly over the encrypted database, producing encrypted query result, which can only be decrypted by service users.

The CryptDB [10] is a system supporting SQL queries over encrypted databases. This system needs the extension of existing DBMSs to support homomorphic operations like SUM and AVG, because the used homomorphic encryption scheme [9] performs multiplication on ciphertexts to get the sum of corresponding plaintexts. The existing DBMSs cannot natively support multiplication of values in one table column.

In [5], a mechanism of supporting aggregate queries is proposed, which is designed only for column-based databases by encrypting multiple values in one table column into one ciphertext. Hence, the mechanism in [5] is not flexible for data insertion and deletion, since the data to be updated is always packed together with other data not to be updated.

In [8, 4], a homomorphic encryption scheme is proposed to be efficient and practical. But it needs users to determine the maximum sum of plaintexts, which should not be bigger than the modulo. Otherwise, the scheme is not homomorphic. That is, if it is used to encrypt values in a table column, the maximum sum of such values must be predetermined and not be bigger than the modulo. This requirement is not practical for long standing and large encrypted databases, since the data there might have their characteristics changed over a long period of time.

In this demo paper, we demonstrate the management of encrypted databases by using a new homomorphic encryption scheme [6] together with an order-preserving indexing scheme [7]. Our techniques in this demo can be applied to existing Database Management Systems (DBMSs) without any change or extension. The encrypted data are still stored relationally, so they can be updated by using standard SQL statements. Moreover, there is no any requirement on the maximum sum of plaintexts in one table column, and

[*]Shenlu was a vacation student in CSIRO ICT Centre, coming from RMIT university, Australia.

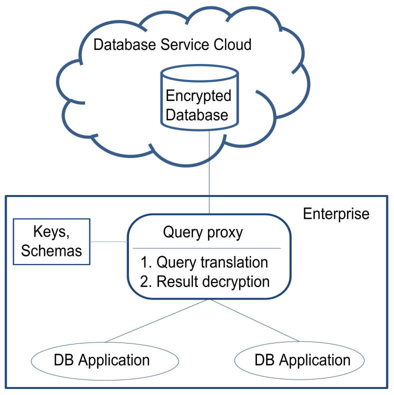

Figure 1: Architecture of Querying Encrypted Databases

their number and range. Our demo also shows that a query over encrypted database can be completed efficiently.

2. ARCHITECTURE OF MANAGING ENCRYPTED DATABASES

The architecture of querying encrypted databases is shown in Figure 1. In this architecture, there is a database service provided in a public cloud, and an enterprise that deploys into the cloud a database, which is encrypted by the enterprise to protect its privacy.

To query or update the encrypted database, the enterprise has a query proxy managing the communication between the database applications and the encrypted database. In this architecture, the proxy is deployed into the administrative boundary of the enterprise. Hence, the untrusted cloud database administrators can access only the ciphertexts stored in the cloud databases and cannot know the encryption keys and schemas of the encrypted databases.

3. OVERVIEW OF TECHNIQUES IN DEMO

We demonstrate a system of managing encrypted databases. Particularly, the system can efficiently process the aggregate queries over encrypted databases with range conditions. For example, such a query can be: select the average income of persons, who are born from year 1980 to 1990. Our system processes such queries by implementing the homomorphic encryption scheme [6] and the order-preserving indexing scheme [7].

3.1 Order-Preserving Indexing

The order-preserving indexing scheme [7] has the following feature: given two values $v_1 < v_2$, we have $\mathtt{OPS}(k, v_1) < \mathtt{OPS}(k, v_2)$, where $\mathtt{OPS}(k, v_i)$ means the order-preserving index of value v_i under the secret key k. Unlike the order-preserving encryption schemes [2, 3], the order-preserving indexing scheme is not required to recover the value v from $\mathtt{OPS}(k, v)$. Hence, the design of the order-preserving indexing scheme can be simpler than the order-preserving encryption schemes. The order-preserving index scheme is enhanced in our recent work by introducing nonlinearity into indexes.

The order-preserving indexing scheme is designed to perform range queries over encrypted databases, together with existing encryption algorithms. In [7], the scheme is used together with the AES algorithm. Since AES is not homomorphic, the aggregate queries are not supported in [7]. In this paper, the order-preserving indexing scheme is combined with homomorphic encryption, so aggregate queries with range conditions can be processed.

3.2 Homomorphic Encryption

The (additive) homomorphic encryption scheme used in this demo is proposed in a patent application [6]. Let $\mathtt{Enc}$ be the encrypting operation in the scheme, $\mathtt{Dec}$ the decrypting operation and $K(n)$ the key. Then, given a value v, the encryption $\mathtt{Enc}(K(n), v)$ will generate a ciphertext $(c_1, ..., c_n)$, which consists of n subciphertexts c_1, ..., and c_n. The parameter n in a key indicates the number of subciphertexts to be generated. In decryption, the operation $\mathtt{Dec}(K(n), (c_1, ..., c_n))$ will return v. Given another value v', let $\mathtt{Enc}(K(n), v') = (c'_1, ..., c'_n)$. Then, this scheme ensures $\mathtt{Dec}(K(n), (c_1 + c'_1, ..., c_n + c'_n)) = v + v'$ for homomorphism.

The scheme [6] defines the generic forms of the $\mathtt{Enc}$ and $\mathtt{Dec}$ operations, and also their correctness conditions. Based on the generic forms and conditions, many instances of the scheme can be defined. Here, we introduce two instances to make this demo paper more self-contained.

In the first instance, the key $K(n)$ is a list of n pairs of real numbers, $[(k_1, s_1), ..., (k_n, s_n)]$, where $n \geq 3$, $\sum_{i=1}^{n-2} k_i \neq 0$ and $k_{n-1} \neq 0$ and $k_n + s_n \neq 0$. The operation $\mathtt{Enc}$, encrypting v into $(c_1, ..., c_n)$, is defined by the following steps.

- Let r_1,..., r_{n-2} be $n - 2$ random numbers.
- $c_i = k_i * v + s_i + r_i$ for $1 \leq i \leq n - 2$.
- $c_{n-1} = k_{n-1} * \sum_{i=1}^{n-2} r_i + s_{n-1}$.
- $c_n = k_n + s_n$.

For this instance, the operation $\mathtt{Dec}$, decrypting $(c_1, ..., c_n)$ into v, is defined below, where the division of c_n is used for counting the number of ciphertexts added or averaged.

- $L = \sum_{i=1}^{n-2} k_i$.
- $S = c_n/(k_n + s_n)$.
- $I = c_{n-1} - S * s_{n-1}$.
- $v = (\sum_{i=1}^{n-2} (c_i - S * s_i)/L) - I/(L * k_{n-1})$.

In the second instance, we let the key $K(n)$ be a list of n tuples of real numbers, $[(k_1, s_1, t_1), ..., (k_n, s_n, t_n)]$, where $n \geq 3$, $k_i \neq 0$ $(1 \leq i \leq n-1)$, $\sum_{i=1}^{n-2} s_i \neq 0$, $s_{n-1} \neq 0$ and $k_n + s_n + t_n \neq 0$. The operation $\mathtt{Enc}$ of the second instance encrypts v into $(c_1, ..., c_n)$ by the following steps.

- Let r_1,..., r_{n-2} be $n - 2$ random numbers.
- $c_i = k_i * t_i * v + s_i + k_i * r_i$ for $1 \leq i \leq n - 2$.
- $c_{n-1} = k_{n-1} * t_{n-1} * \sum_{i=1}^{n-2} r_i + s_{n-1}$.
- $c_n = k_n + s_n + t_n$.

The operation $\mathtt{Dec}$ of the second instance decrypts the ciphertext $(c_1, ..., c_n)$ into v by the steps below. Compared with the first instance, the last step below divides different $c_i - S * s_i$ $(1 \leq i \leq n - 2)$ with different secret values $L * k_i$, rather than only by L, hence increasing the robustness against brute-force attacks.

- $L = \sum_{i=1}^{n-2} t_i$.
- $S = c_n/(k_n + s_n + t_n)$.
- $I = c_{n-1} - S * s_{n-1}$.
- $v = (\sum_{i=1}^{n-2} (c_i - S * s_i)/(L * k_i)) - I/(L * k_{n-1} * t_{n-1})$.

Moreover, the instances of the encryption scheme [6] can be composed into new instances, which are still homomorphic. A composed instance is more robust than its constituent instances. For example, each subcipertext from the second instance above can be encrypted again by using the first instance, and vice versa.

Our homomorphic scheme is very efficient. For example, on a Dell Latitude E4310 laptop, the second instance takes averagely 4.1 seconds to encrypt and decrypt 10000 values, with 64 subciphertexts for each value, and with key components and noises containing 5 digits. To increase the robustness, a user just needs to increase the number of subciphertexts, instead of producing large subciphertexts. Hence, each subciphertext in our scheme can still be treated as a numeric value by existing DBMSs, rather than as a long string. Our homomorphic scheme is designed to be scalable with respect to the number of subciphertexts. On the contrary, the encrypting operation in [11] and the decrypting operation in [8, 4] have the exponential complexity with respect to the number of secret shares, that is, they are not scalable with the number of shares.

3.3 Database Encryption and Query

Suppose a value is encrypted into n subciphertexts by using our scheme. Then, in our system, the n subciphertexts are stored into n columns in one record, not necessarily in the order of subciphertexts generated from encryption. Moreover, the subcipertexts of one value can be mixed with the subcipertexts of another value in one record. Hence, the attackers lose the information on which subciphertexts can be meaningfully combined in a correct order to perform an attack.

In the same record, there is also a column storing the order-preserving index and another column storing the hash of the value. The indexes are used by DBMSs to check range query conditions, and the hashes are used to check equality query conditions. For encrypted data satisfying the conditions, all corresponding subciphertexts are included in the query result, or their sum or average are included in the query result. That is, for the SUM or AVG operations in a query, DBMSs just need to perform the standard SQL SUM or AVG operations over each column of subciphertexts.

In [1], the Shamir's secret sharing mechanism [11] is used to produce a number of secret shares, which are then stored into databases managed by different service providers. Our system does not require a user to use multiple cloud database services to store subciphertexts. Due to this feature, our system is also suitable for databases deployed within the premises of an enterprise.

As shown above, our scheme does not reply on modulo operations, so it can work on infinite fields. Hence, using our scheme does not need to predetermine the maximum sum of values in one table column. This feature allows our system to manage large and long-standing encrypted databases.

4. CASE STUDIES

Our demo assumes that a family census has been done by a government agency. Since the census data contains sensitive information, such as name, birth date and income, the government agency hopes that the census data can be encrypted in their databases, which might be deployed into a public cloud, and more importantly the encrypted data can still be managed and queried smoothly.

To demonstrate the correctness of our system, we put each data into three databases: 1) a plain database, where data is not encrypted, 2) an encrypted database, where data is encrypted by using the first homomorphic encryption instance introduced above, and 3) another encrypted database, where data is encrypted by using the second homomorphic encryption instance. Note that the plain database is only used to check query results, just for the demo purpose and not needed in practice. In addition, the names of databases, tables and columns are all hashed in encrypted databases.

Our demo includes the creation and deletion of encrypted databases, in particular, the table schemas in the plain database and two encrypted databases are totally different. Hence, from the schemas, the untrusted cloud database administrators or attackers cannot learn any information about the stored data.

Our demo shows the operations of updating, inserting and deleting data in the encrypted database, demonstrating that the encrypted data can be managed smoothly in our system. These operations are an integral part of encrypted database management. For example, we can insert family members into the person table, change the salaries of particular family members, and delete records of persons satisfying some conditions.

We also demonstrate various queries by using our system, in particular aggregate queries with range conditions. For example, over the encrypted data, we can query the average income for persons in an area specified by postcode, who are born in some range of years.

Our demo is available at http://150.229.2.229/familySys/home, where detailed steps are included to guide the demo.

5. REFERENCES

[1] D. Agrawal, A. E. Abbadi, F. Emekçi, and A. Metwally. Database management as a service: Challenges and opportunities. In *Proceedings of the 25th International Conference on Data Engineering*, pages 1709–1716, 2009.

[2] R. Agrawal, J. Kiernan, R. Srikant, and Y. 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, 2004.

[3] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill. Order-preserving symmetric encryption. In *Proceedings of the 28th Annual International Conference on Advances in Cryptology*, EUROCRYPT '09, pages 224–241, 2009.

[4] Z. Brakerski and V. Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *Proceedings of the 31st annual conference on Advances in cryptology*, CRYPTO'11, pages 505–524, 2011.

[5] T. Ge and S. Zdonik. Answering aggregation queries in a secure system model. In *the 33rd international conference on Very large data bases*, pages 519–530, 2007.

[6] D. Liu. Homomorphic encryption for database querying. Australian Provisional Patent (filed by CSIRO), 2012.

[7] D. Liu and S. Wang. Programmable order preserving secure index for encrypted database query. In *Proceedings of the 5th IEEE International Conference on Cloud Computing*, pages 502–509, 2012.

[8] M. Naehrig, K. Lauter, and V. Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, CCSW '11, pages 113–124, 2011.

[9] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the 17th international conference on Theory and application of cryptographic techniques*, EUROCRYPT'99, 1999.

[10] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 85–100, 2011.

[11] A. Shamir. How to share a secret. *Commun. ACM*, 22:612–613, November 1979.