DEIM Forum 2010 B5-5

An Encryption Scheme to Prevent Statistical Attacks in the DAS Model

Hasan KADHEM†, Toshiyuki AMAGASA†,††, and Hiroyuki KITAGAWA†,††

† Graduate School of Systems and Information Engineering
†† Center for Computational Sciences
University of Tsukuba
Tennodai 1–1–1 Tsukuba Ibaraki 305-8573
E-mail: †hsalleh@kde.cst.tsukuba.ac.jp, ††{amagasa,kitagawa}@cs.tsukuba.acjp

Abstract Encryption can provide strong security for sensitive data against inside and outside attacks. This is especially true in the "Database as Service" model, where confidentiality and privacy are important issues for the client. However, existing encryption approaches are vulnerable to a statistical attack because each value is encrypted to another fixed value. We present a novel database encryption scheme called MV-OPES, which allows privacy-preserving queries over encrypted databases with an improved security level. Our idea is to encrypt a value to different multiple values to prevent statistical attacks. At the same time, MV-OPES preserves the order of the integer values to allow comparison operations to be directly applied on encrypted data. We also present techniques to execute as many relational operators as possible over an encrypted database to minimize the processing of decrypted data. Our scheme can easily be integrated with current database and it is robust against statistical attack and the estimation of true values.

Key words Encryption, order-preserving, database as service, statistical attack

1. Introduction

Encryption can provide strong security for sensitive data against inside and outside attacks. The primary interest in database encryption results from the recently proposed "database as service" (DAS) architecture [1]. In DAS or database outsourcing, a database owner outsources its management to a "database service provider", which provides online access mechanisms for querying and managing the hosted database. At the same time, the service provider incurs most of the server management and query execution load.

Clients would like to take advantage of the provider's robust storage, but in many cases they cannot trust the provider. Specifically, the provider should be prevented from observing any of the outsourced database contents. Encryption is a common technique used to protect the confidentiality and privacy of stored data in the DAS model. However, the traditional attribute level encryption approach for a database encrypts each value to another fixed value: $X_1 = X_2 \Longrightarrow E^k(X_1) = E^k(X_2)$

This approach is vulnerable to statistical attacks. A statistical attack against an encrypted database seeks to use some apparently anonymous statistical measures to infer individual data. Using such an approach, an attacker, especially an inside attacker, can infer some data by joining tables and using additional statistical information. This problem arises clearly in joining lookup tables with other tables. The lookup tables usually consist of a small and fixed values scale or domain such as gender (male or female), marital status (single, married, separated,...), and city, .etc.

Example 1: Consider the plaintext database shown in Figure 1(a) with the traditional encrypted database in Figure 1(b). In the encrypted employee table, the third column contains two distinct values (67653, 564564), which are clearly gender data. When an attacker knows that there are more male employees than female employees, then the attacker can infer the encrypted values for numbers 1 and 2. Also, it is possible to infer information by joining a project table with the (emp_proj) table. Knowing that only one employee works on project "projectAS", the attacker can recognize the encrypted value for that project from table (emp_proj). Also, the attacker can get the encrypted (emp_id) for the employee who works on that project. Using the same technique, the attacker can infer much more information from the encrypted database, and then try to determine the key used in the encryption process.

A straightforward solution to solve the problem of statis-

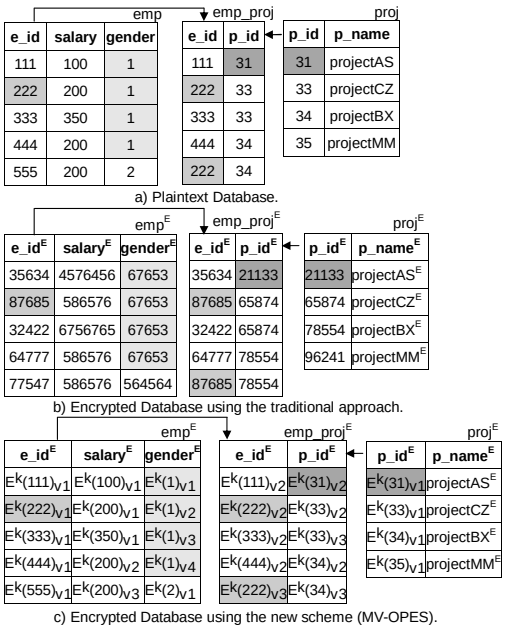

Fig. 1 Database encryption using two approaches.

tical attack is to use different encryption keys for different fields. However, using different keys leads to serious performance degradation, because we need to decrypt all data to execute queries, such as join. Another naive solution is to add to the database a randomly generated data value for each plaintext value and then both values are encrypted together. This approach is used by [2] to encrypt XML documents. Also, it is used with the Cipher-Block Chaining (CBC) [3] as initial vector IV. This approach leads to storage cost. In addition, it is cause serious performance degradation because the plaintext and the additional value associated to the plaintext should be decrypted together before executing any query.

This paper presents a new database encryption scheme called MV-OPES (Multivalued - Order Preserving Encryption Scheme), which allows one integer to be encrypted to many values using the same encryption key while preserving the order of the integer values. Figure 1(c) describes the new encryption technique applied on the employee plaintext database shown in Figure 1(a). The equal plaintext values in Figure 1(a) are encrypted to different ciphertext values in Figure 1(c) as opposed to the ciphertext values in Figure 1(b). For instance, the emp_id (222) is encrypted to $E^K(222)_{v_1}$ in the emp table and $E^K(222)_{v_2}$, $E^K(222)_{v_3}$ in the emp_proj table, with high probability that $(E^K(222)_{v_1} \neq E^K(222)_{v_2} \neq E^K(222)_{v_3})$. Also, the encrypted values for the emp_id (111) are always less than the encrypted values for the emp_id (222). In this scheme, attackers cannot infer individual information from the encrypted database even if they have statistical knowledge about the plaintext database. Unlike the bucketing approach [4], [5], which generates a superset of answers with false positive tuples in all queries, our scheme does so only on some condition types; the results contain false positive tuples. MV-OPES can easily be integrated with current database systems as it is designed to work with existing indexing structures such as B-trees. MV-OPES is efficient in insertions and updates. A new value can be inserted in a column, or a value in a column can be modified without requiring changes to the encryption of other values.

1.1 Organization of the Paper

The rest of paper is organized as follows. We first discuss related work in Section 2. Section 3 introduces our new database encryption schema. Section 4 describes the condition translations. Section 5 discusses implementation of the relational operators such as selection, join, and sort over encrypted relations. Section 6 reports the experimental results. We conclude with a summary and directions for future work in Section 7.

2. Related Work

Many database encryption techniques have been proposed, but most are not intended for the one-to-many encryption algorithm.

Order preserving encryption schema (OPES): The idea of OPES is to take as input a user-provided target distribution and transform the plaintext values in such a way that the transformation preserves the order while the transformed values follow the target distribution [6]. Under OPES, it is ideal from the viewpoint of query performance, because comparisons can operate directly on ciphertext, thereby saving the cost of expensive decryptions. From a security point of view, OPES is vulnerable to a tight estimation and statistical attacks if an adversary has knowledge of the distribution. Also, OPES is vulnerable to chosen plaintext attacks if the adversary can choose any number of unencrypted values to his liking and encrypt them into their corresponding encrypted values. In OPES, most probably two columns of two tables do not have the same distribution. That means they are not directly comparable. When this is true, the join query involves expensive decryptions and/or encryptions, because one side must be converted to the other.

Order preserving encryption with splitting and scaling (OPESS): The authors in [2] proposed a new encryption scheme based on the OPES to index the encrypted values in the outsourced XML databases. The idea in OPESS is to map the same plaintext values to different ciphertext values to protect the data against frequency-based or statistical attacks. OPESS consists of two stages splitting and scaling. In splitting, each plaintext value is encrypted into one or more ciphertext values by using different keys. The number of keys used to encrypt a plaintext value is based on the number of occurrence for the plaintext. Scaling is done

after splitting. By using scaling, number of occurrences of encrypted values is multiplied by a scale factor. One of the limitations of OPESS is that security achieved by scaling encrypted data causes an increase in data size. Also, this approach is not efficient in insertions and updates because the encryption method is mainly based on the number of occurrences. OPESS proposed mainly for XML database but it is not applicable for relational database. The reason behind that is the different in executing queries on the relational and the XML databases such as the join operation between different encrypted values.

Structure preserving database encryption scheme: The authors in [7] proposed a new encryption scheme that breaks the correlation between ciphertext and plaintext values by encrypting each database value with its unique cell coordinates. In this scheme, one plaintext value can be encrypted to different values according to their position in the database. There are two immediate advantages to this scheme. First, it eliminates substitution attacks attempting to switch encrypted values. Second, pattern matching attacks attempting to gather statistics based on the encrypted values will fail. However, this scheme can be used only on a trusted server, where a DBA can manage the new index structure in the encrypted database. The join operation is covered by this approach. We think that the only way it can be used to join tables in this scheme is to decrypt tables first, and then perform the join over the decrypted database, which adds overhead. Moreover, if a database reorganization process changes cell coordinates, all affected cells need to be re-encrypted with their new coordinates.

Other relevant work: Another variation to secure databases that has recently been studied is that of "distributed architecture" [8]∼[10] for enabling privacy-preserving outsourced storage of data. However, distributing the database content to many servers is not the concern of this body of work. We focus on securing the central database content stored in a trusted, un-trusted, or semi-trusted [11] server. The encryption scheme proposed in this paper (MV-OPES) is a novel technique to encrypt an integer value to many different values, and in the sense of performing queries over an encrypted database based on inequality.

3. Proposed Multivalued Order Preserving Encryption Scheme

Encrypting plaintext values in a column having values in the range $[D_{min}, D_{max}]$, the boundaries for all integers in the domain $(B_{D_{min}}, ..., B_{D_{max}}, B_{D_{max+1}})$ are generated using an increasing/decreasing function (order preservation). The generated boundaries identify the intervals. For instance, interval I_i is identified by $[B_i, B_{i+1})$. We then generate the encrypted values for integer i as random values from the interval I_i (Multivaluedness). Details regarding the random distribution used to choose the encrypted values are discussed in Section 3.2.

3.1 Generate Bucket Boundaries

Bucket boundaries are generated using two functions: initial and increasing/decreasing. Details of these two functions are discussed next.

3.1.1 Initial Function

We are given a domain $[D_{min}, D_{max}]$, with $(D_{max} - D_{min} + 1)$ integers: $\{D_{min}, D_{min+1}, ..., D_{max}\}$. Initially, we choose the starting $(initial)$ point from the domain. We then compute the boundary for the initial point using the following function:

$B_{initial} = Enc^K(initial)$

where Enc is the function used to encrypt the $(initial)$ value using key K. Any block cipher algorithm such as DES [12], TDES, Blowfish [13], AES [14], RSA [15], etc., or a hashing function can be used to encrypt the value. We assume that the boundaries are stored in memory or secondary storage, so the initial point can be any point in the domain. For instance, the initial point could be D_{min}; then only the increasing function is used to generate the remaining boundaries.

3.1.2 Increasing/Decreasing Function

To preserve the order of the integers, we use two functions to generate boundaries. First, boundaries for values greater than the initial point are generated by an increasing function. Second, a decreasing function is used to generate boundaries for values less than the initial point. The goal for the increasing/decreasing function is to create encrypted interval scales for all integers in the domain with different sizes. Differences in intervals size are ensured by predefined percentage and a sequence of random numbers.

Given the initial point $(initial)$, the interval size IS, and the difference percentage on the encrypted interval size DP, the boundaries are derived by the following function:

$$B_i = \begin{cases} B_{i+1} - (Enc^K(IS) + (Enc^K(IS) * DP) * R_i), \\ \qquad\qquad D_{min} \leqq i < initial \\ B_{i-1} + (Enc^K(IS) + (Enc^K(IS) * DP) * R_i), \\ \qquad\qquad initial < i \leqq D_{max+1} \end{cases}$$

where R_i is a sequence of random numbers in the range $[-1, 1]$. There are many pseudorandom number generators with useful security properties.

The DP used in the formula to control the differences between intervals size that will be in the range $[-Enc^K(IS) * DP, Enc^K(IS) * DP]$.

3.2 Encryption Functions

Here we discuss how to encrypt a plaintext relation R. For

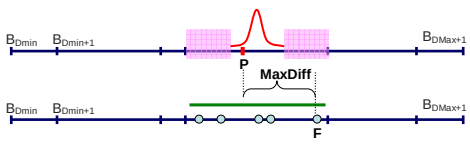

Fig. 2 Calculating the MaxDiff in MV-OPES.

each tuple $t = (A_1, A_2, ..., A_n)$ in R, the encrypted relation R^E stores a tuple:

$(E(A_1), E(A_2), ..., E(A_n))$

where E is the function used to encrypt an attribute value of the tuple in the relation. The encryption function $E(i)$ is performed by choosing a random number in the interval I_i, which is identified by $[B_i, B_{i+1})$. The random distribution used to choose the encrypted values depends on whether attribute values are unique (primary key) or redundant (foreign key). The integers in the primary key table are connected to the values in the foreign key table using range distance, which leads to false positive results. We call this range distance **MaxDiff**, which is the greatest distance (among all intervals in the domain) between value P in the primary key table and value F in the foreign key tables. MaxDiff is used to execute multiple relational operators over an encrypted database, such as join.

When inserting a new record in the foreign key table, MaxDiff should be updated by comparing the current MaxDiff with the distance between the primary key and the new foreign key in the same interval. Figure 2 shows how to calculate MaxDiff; it also shows how to choose a random number for both primary key P and foreign key F. The random function used to encrypt integers in the primary key table **(random_primary)** and a foreign key table **(random_foreign)** are different. Choosing random distribution for both functions is based on the following perspectives:

- **Security perspective:** Hide the interval boundaries in the primary key table because there is only one integer that represents each interval. This goal is achieved by maximizing the distance between primary key values that represent two sequential intervals. Based on that premise, the primary key should be as close as possible to the middle of the interval. This matches the normal distribution. Many records in the foreign key table represent the same interval. So to hide the interval boundaries we need to distribute the values over the whole interval so the attacker cannot differentiate between intervals. This condition matches the uniform distribution.
- **Performance perspective (reduce the false positives):** Reduce the overlap between intervals when connecting (joining) the primary key table with the foreign key table. This goal is achieved by minimizing the distance between the primary key and the foreign keys in the same interval (reducing MaxDiff). This is achieved by using normal distribution to choose a random number in both the primary key and foreign key tables. However, we already decided that uniform distribution is the best distribution to hide interval boundaries in the foreign key table, not normal distribution.

We can improve the security and performance also by set left and right margins in each interval. The primary encrypted value is then chosen to be between those margins. The margin sizes affect security and the percentage of false positives. We will investigate the optimal margin size in future work. In this paper, for simplicity, we assume equi-width partitioning. The interval is divided into three equal parts (left margin, middle part, right margin). The primary key is then chosen based on normal distribution within the middle part only. In this case, the distance between two primary keys that represent two sequential intervals will be *(right margin of the first interval + the left margin of the second interval).*

3.3 Decryption Functions

Given the operator E, which encrypts a plaintext value to many ciphertext values, we define its inverse operator D, which decrypts the ciphertext value to its corresponding plaintext value. Simply, the decryption function D in MV-OPES searches for the interval where the encrypted value is located. Specifically, to decrypt an encrypted value C, the decryption function searches for the closer boundary B_p that is greater than C, then returns the plaintext value, which is the left boundary $p - 1$.

Given the boundaries $(B_{D_{min}}, ..., B_{D_{max+1}})$, which are stored either in memory with a small domain, or using a secondary storage-based indexing structure such as B+ tree for a large domain. The decryption function can be a sequential search or binary search. The sequential search is used when decrypting ciphertext values in a sorted column; the binary search is used to decrypt ciphertext values in an unsorted column.

4. Condition Translations

This section explains how to translate a query condition C over a plaintext database in operations (such as selection and join) to corresponding conditions over encrypted database C^E. We consider query conditions characterized by the following grammar rules:

- Condition ← Attribute θ Value
- Condition ← Attribute θ Attribute
- Condition ← (Condition $\vee$ Condition) | (Condition $\wedge$ Condition) | ($\rightarrow$ Condition)

where θ is a binary operation in the set $\{=, <, \leqq, >, \geqq\}$.

The conditions are divided into two groups according to the type of result, that is, whether or not the result contains false positives. The first group consists of conditions that

Table 1 Translation of (Attribute θ Value) conditions.

C	c^E
$A = v$	$A^E\ BETWEEN\ B_v\ and\ (B_{v+1} - 1)$
$A < v$	$A^E < B_v$
$A \leqq v$	$A^E < B_{v+1}$
$A > v$	$A^E \geqq B_{v+1}$
$A \geqq v$	$A^E \geqq B_v$

Table 2 Translation of (Attribute θ Attribute) conditions.

C	c^E
$P = F$	$F^E\ BETWEEN\ (P^E - MaxDiff)$ $(P^E + MaxDiff)$
$P < F$	$(P^E + MinMarg)\ <\ F^E$
$P \leqq F$	$(P^E\ + MaxDiff)\ \leqq\ F^E$
$P > F$	$(P^E - MinMarg)\ >\ F^E$
$P \geqq F$	$(P^E\ - MaxDiff)\ \geq\ F^E$

contain a binary operation between attribute and value (Attribute θ Value). The result based on those conditions contains neither false positives nor missed answer tuples. The second group consists of conditions that contain a binary operation between two attributes (Attribute θ Attribute) such as equi-join. The result based on those conditions contains false positive tuples.

Each condition based on MV-OPES is translated as arrange query. This is necessary because of (Multivaluedness) that applied in encryption scheme. Table 1 shows how each condition in the form ($Attribute\theta Value$) is translated into corresponding condition over encrypted database.

There are two constraints in performing conditions in the form ($Attribute\theta Attribute$). First, the two attributes should have the same domain. Second, the condition is performed between two primary keys or between a primary key and the related foreign key. Given a condition (P θ F), such that P and F have same domain, P is the primary key and F is a foreign or primary key. The translation for this condition is shown in Table 2, where (MinMarg) is the minimum margin among all intervals in the domain.

(Condition1 $\vee$ Condition2), (Condition1 $\wedge$ Condition2), ($\rightharpoonup$ Condition): Two composite conditions are translated directly over the encrypted domain by translating each condition individually. The translation is given as follows: $C1 \vee C2 \rightarrow C1^E \vee C2^E$, $C1 \wedge C2 \rightarrow C1^E \wedge C2^E$

However, when C is in the form of *(Attribute θ Attribute)*, this condition ($\rightharpoonup C$) cannot be translated directly because of the false positive result. This paper does not discuss this translation. Neither are conditions that involve more than one attribute and operator discussed.

5. Implementing Relational Operators over Encrypted Relations

This section describes the process of implementing relational operators (such as selection, projection, and sorting) in the proposed scheme. The relational operators are implemented, as much as possible, to be executed over the encrypted database. However, when a condition of the form *(Attribute θ Attribute)* is attached to the operator, the returned answers might contain false positives. These answers are then filtered in client-side after decryption to generate the exact result. Beyond that, some operators cannot be performed fully on the encrypted relations. When that happens, a post process operation is performed on the result after decryption. We attempt to minimize the amount of work done in post process operations. Table 3 shows the implementation of the operators over encrypted databases. The E on the operators emphasizes the fact that the operator is to be executed over the encrypted database. The L^E refers to the encrypted attributes.

Table 3 Implementation of the operators over encrypted databases.

Operator	op	op^E
Selection (σ)	$\sigma_C(R)$	$D(\sigma^E_{C^E}(R^E))$
Join ($\bowtie$)	$R \overset{\bowtie}{C} T$	$\sigma_C\Big(D(R^E \overset{\bowtie^E}{C^E} T^E)\Big)$
Sorting (τ)	$\tau_L(R)$	$\tau_L\Big(D(\tau^E_{L^E}(R^E))\Big)$
Projection (π)	$\pi_L(R)$	$D(\pi^E_{L^E}(R^E))$
Grouping and Aggregation (γ)	$\gamma_L(R)$	$\gamma_L\Big(D(\tau^E_{LG^E}(R^E))\Big)$
Duplicate Elimination (δ)	$\delta(R)$	$\delta\Big(D(\tau^E_{L^E}(R^E))\Big)$
Union ($\cup$)	$R \cup T$	$D(R^E \cup^E T^E)$ (based on bag) $\delta\Big(D(\tau^E_L(R^E \cup^E T^E))\Big)$ (set)
Difference ($-$)	$R - T$	$D(\tau^E_{LR^E}(R^E)) - D(\tau^E_{LT^E}(T^E))$

Query Splitting: We split the computation of a query Q across the server and the client. The client will use the implementation of the relational operators to send part of the query Q_s to the server to be executed on the encrypted database. The second part, which is client query part Q_c, is performed on the decrypted data. Query splitting is as follows:

$$\underbrace{op\ (R)}_{Q} = \underbrace{op^c\ D}_{Q_c}\ \underbrace{\left(op^E\ (R^E)\right)}_{Q_s}$$

where op^c refers to operations performed on the client side, and op^E is operations performed on encrypted relations R^E on the server side.

6. Experiments

We have conducted many experiments to examine the validity and effectiveness of the architecture proposed in this paper. However, because of space limitations, we will discuss just two sets of experiments. The experiments were conducted by implementing MV-OPES on MS SQL Server 2008. The algorithms were implemented in VB.NET as a

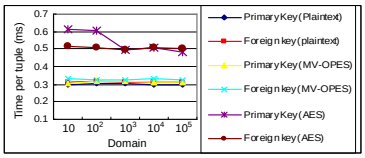

Fig. 3 Time per tuple (in ms) required to insert tuples.

client side application. The experiments were run using version 3.0 of the Microsoft.Net framework and on a Microsoft XP workstation with a 2.6 GHz Intel Core 2 processor and 3 GB of memory. The results sketched in this section are the average for at least 10 executions.

The first set of evaluations studied the encryption performance in our scheme using different domains and various difference percentages (DP). Also, we compare the performance of our scheme with a database encrypted using AES. Two tables were used to perform this evaluation. The first table is the primary key table, which contains all integers in the domain. The second table is the foreign key table, which holds $100,000$ records picked randomly from a uniform distribution between D_{min} and D_{max}. Figure 3 shows the times for encryption and inserting values in the primary key and foreign key tables for different domains. The results show that AES takes the longest time to insert tuple in both tables since the encryption time is much more than in MV-OPES. The small difference in time shown in the figure between plaintext and our scheme is the cost of encryption. The figure shows that this overhead is negligible.

In the equijoin operation, we studied the percentage of false positives returned by performing a join operation over encrypted relations. Also, we studied the overhead on both the server and client sides. The percentage of false positives shown in Figure 4(a) increases with the domain size and DP. That results due to the increase in the overlap between intervals in the encrypted scale when performing a join operation based on $MaxDiff$. From Figure 4(b), we can easily see that the time required to perform a join operation on the server side in our scheme increases according to the size of domain and takes approximately the same shape as the join operation on the plaintext database. While the cost of join operation using AES is much more than our scheme. This is especially when using large domains ($> 10^3$) since the index is essentially unusable for many operations (including join) which turn into full table scans [16].

Figure 4(c) shows the client side performance to decrypt and filter the result returned by performing a join operation on the server side. The figure shows that our scheme has only small overhead on the client side. We also observe that the time slightly increases as the domain and DP increase, because of increased false positives. On the other hand, we can see the performance degradation when using AES compared

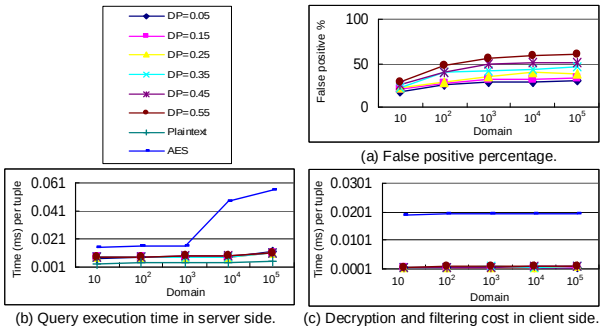

Fig. 4 Equijoin cost.

with our scheme.

7. Conclusion and Future Work

Encryption can be used to provide confidentiality and privacy for sensitive databases, which are important issues, especially in the DAS model. Unfortunately, traditional attribute level encryption is vulnerable to statistical attacks because each value is encrypted to another fixed value. We propose a novel encryption scheme (MV-OPES) that is robust against statistical attack and estimation of the true value because it allows one integer to be encrypted to many different values using the same encryption key. It also preserves the order of the integer values to allow any comparison operation to be directly applied to the encrypted data. We have developed techniques so that most processes in executing SQL queries can be done on encrypted databases. In some cases, a small amount of work to filter false positives or perform relational operations is needed on the decrypted data. In the future, we will focus on security analysis of our scheme and some improvement issues. We also plan to study the encryption of non-integer data such as strings.

Acknowledgments

This study has been partially supported by Grant-in-Aid for Young Scientists (B) (#21700093) and Grant-in-Aid for Scientific Research on Priority Areas from MEXT (#21013004).

References

[1] H. Hacigumus, S. Mehrotra, and B. Iyer, "Providing database as a service," Data Engineering, International Conference on, vol.0, p.0029, 2002.

[2] H. Wang and L.V.S. Lakshmanan, "Efficient secure query evaluation over encrypted xml databases," VLDB '06: Proceedings of the 32nd international conference on Very large data bases, pp.127–138, VLDB Endowment, 2006.

[3] M.C.H.W.S.J.L.T.W.L. Ehrsam, William F., "Message verification and transmission error detection by block chaining,", February 1978.

[4] B. Hore, S. Mehrotra, and G. Tsudik, "A privacy-preserving index for range queries," VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases, pp.720–731, VLDB Endowment, 2004.

[5] H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra, "Execut-

ing sql over encrypted data in the database-service-provider model," SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data, New York, NY, USA, pp.216–227, ACM, 2002.

[6] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data, New York, NY, USA, pp.563–574, ACM, 2004.

[7] Y. Elovici, R. Waisenberg, E. Shmueli, and E. Gudes, "A structure preserving database encryption scheme.," Secure Data Management, ed. W. Jonker and M. Petkovic, Lecture Notes in Computer Science, vol.3178, pp.28–40, Springer, 2004.

[8] G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, R. Motwani, U. Srivastava, D. Thomas, and Y.X. 0002, "Two can keep a secret: A distributed architecture for secure database services," CIDR, pp.186–199, 2005.

[9] F. Emekci, D. Agrawal, A.E. Abbadi, and A. Gulbeden, "Privacy preserving query processing using third parties," ICDE '06: Proceedings of the 22nd International Conference on Data Engineering, Washington, DC, USA, p.27, IEEE Computer Society, 2006.

[10] D. Agrawal, A.E. Abbadi, F. Emekci, and A. Metwally, "Database management as a service: Challenges and opportunities," ICDE '09: Proceedings of the 2009 IEEE International Conference on Data Engineering, Washington, DC, USA, pp.1709–1716, IEEE Computer Society, 2009.

[11] H. Kadhem, T. Amagasa, and H. Kitagawa, "A novel framework for database security based on mixed cryptography," International Conference on Internet and Web Applications and Services, vol.0, pp.163–170, 2009.

[12] DES, "Data encryption standard," Federal Information Processing Standards Publication, vol.FIPS PUB 46, 1977.

[13] B. Schneier, "Description of a new variable-length key, 64-bit block cipher (blowfish)," Fast Software Encryption, Cambridge Security Workshop, London, UK, pp.191–204, Springer-Verlag, 1994.

[14] AES, "Advanced encryption standard," National Institute of Science and Technology, vol.FIPS 197, 2001.

[15] R.L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," Commun. ACM, vol.21, no.2, pp.120–126, 1978.

[16] S. Hsueh, "Database encryption in SQL server 2008 enterprise edition," Microsoft White Papers, vol.SQL Server 2008, February 2008.