L-EncDB: A Lightweight Framework for Privacy-Preserving Data Queries in Cloud Computing

Jin Li [a,*]

[a] *School of Computer Science and Educational Software*
Guangzhou University, Guangzhou, 510006 P.R. China

Zheli Liu [b,*]

[b] *College of Information Technical Science*
Nankai University, P.R. China

Xiaofeng Chen [c]

[c] *State Key Laboratory of Integrated Service Networks (ISN)*
Xidian University, Xi'an, P.R. China

Fatos Xhafa [d]

[d] *Department of Languages and Informatics Systems*
Technical University of Catalonia, Spain

Xiao Tan, Duncan S. Wong [e]

[e] *Department of Computer Science*
City University of Hong Kong, Hong Kong

Abstract

With the advent of cloud computing, individuals and organizations have become interested in moving their databases from local to remote cloud servers. However, data owners and cloud service providers are not in the same trusted domain in practice. For the protection of data privacy, sensitive data usually have to be encrypted before outsourcing, which makes effective database utilization a very challenging task. To address this challenge, in this paper, we propose L-EncDB, a novel lightweight encryption mechanism for database, which i) keeps the database structure, and ii) supports efficient SQL-based queries. To achieve this goal, a new format-preserving encryption (FPE) scheme is constructed in this paper, which can be used to encrypt all types of character strings stored in database. Extensive analysis demonstrates

that the proposed L-EncDB scheme is highly efficient and provably secure under existing security model.

Key words: Data query, outsourcing, privacy, format-preserving encryption

1 Introduction

The ever-increasing amount of valuable digital data both at home and in business needs to be protected, since its irrevocable loss is unacceptable. The advent of cloud storage motivates enterprises and organizations to outsource data storage to third-party cloud providers, as evidenced by many real-life case studies. Cloud storage services promise to be a solution for this problem. In recent years, their popularity has increased dramatically. They offer user-friendly, easily accessible and costsaving ways to store and automatically back up arbitrary data, as well as data sharing between users and synchronization of multiple devices.

As in any existing application and system, security and privacy play an extremely important role for the success, and certainly raise new challenges among the many others that cloud storage is confronted with. Specifically, when entrusting data to the cloud, data owner also releases control over the data, resulting that their trust is put in the cloud service provider's integrity and in the security of its access control mechanisms. However, individuals and especially businesses hesitate to entrust their data to cloud storage services since they fear that they will lose control over it. Recent successful attacks on cloud storage providers have exacerbated these concerns. The providers are trying to alleviate the situation and have taken measures to keep their customers' data secure. The simple and popular solutions adopted for data privacy are traditional encryption techniques such as public key encryption or symmetric key encryption. Through these encryption methods before outsourcing, the security of users's data can be protected.

However, traditional database encryption will change the data structure of original data, and results in the impracticability of database application for various kinds of SQL operations. If the data structure is changed, it cannot support data operations over ciphertext such as range query and fuzzy query, etc. Especially, there has been considerable recent interest in the paradigm of

* Corresponding author. Jin Li and Zheli Liu contribute to this work equally.
Email addresses: jinli71@gmail.com (Jin Li), liuzheli1978@163.com (Zheli Liu).

data mining-as-service: a company (data owner) lacking in expertise or computational resources can outsource its mining needs to a third party service provider. However, both the outsourced database and the knowledge extracted through data mining are considered private property of the data owner in many applications. Thus, to protect data privacy while realizing data mining and knowledge extraction, the data owner is required to transform its data without changing its structure before outsourcing.

Contribution. To realize effective data utilization after secure outsourcing, we propose a lightweight encrypted database mechanism denoted by L-EncDB. This new mechanism is able to protect sensitive information while keeping the data structure in outsourcing service for big database application. In the proposed L-EncDB system, the encryption and query are based on SQL. Through only one interface, all SQL sentences for database can be interpreted.

Furthermore, based on format-preserving encryption (FPE) technique and a new character string FPE scheme, L-EncDB can be implemented to preserve data type and length in ciphertext. It enables i) to encrypt data and store them without changing original database structure, ii) to perform SQL operations on all kinds of databases, including text database such as SQLite and Access, iii) to support SQL-based operations including advanced fuzzy and range queries.

Innovation. In this paper, a novel FPE scheme with the method of "multi-radix modular addition" is proposed to support the L-EncDB lightweight framework for privacy-preserving outsourced database. The new proposed FPE can preserve both length and storage size of character strings, which cannot be efficiently achieved in the traditional FPE schemes. Based on the FPE scheme, data operations such as data mining and SQL-based queries can be directly executed over ciphertexts in the proposed L-EncDB framework. Furthermore, L-EncDB framework can be extended to text database (such as SQLite used in mobile) and NoSQL databases, which have not been considered in the previous related work.

1.1 Organization

The rest of this paper proceeds as follows. In Section 2, we give a survey for the related work to ours. In Section 3, we propose the system architecture and construction method for the L-EncDB system. In Section 4, we propose a practical construction of FPE for character string. Its security and performance analysis is also given in this section. In Section 5, we present the implementation of prototype for L-EncDB with the proposed FPE, and in Section 6, we present an extension of the L-EncDB to NoSQL database encryption. Finally

we draw conclusion and show the future work in Section 7.

2 Related Work

We briefly discuss FPE technique and privacy-preserving database encryption solutions in this Section.

2.1 FPE

The notion of FPE [1–4] has been proposed to generate ciphertext with the same format as plaintext while encrypting sensitive information. More specifically, FPE can keep data type and length in the ciphertext, therefore, without changing database structure and field type. Thus, the use of FPE enables upgrading database security in a transparent way. The goal of FPE is to generate ciphertext which falls in the same *domain* as the plaintext. Some practical FPE schemes have been proposed for simple domains such as integer [5], character data [3] and datetime [6]. Character data is the common data type in database, which appears in the form of character strings, i.e., the finite sequences of characters from some character sets. However, there is no suitable character FPE solution to preserve both length and storage size of strings above. For a string with character from iso-8859-1 or ASCII, where the storage size of each character is 1 byte, the length of string is equal to its storage size, and FFX is also suitable in this case.

However, most of character sets are represented using more powerful encoding formats, and different characters may require different byte counts to represent. In this paper, such a character set is called "multi-byte character set". Consider a character string of length n with each character in UTF-8, its storage size will be from n bytes to $4n$ bytes. In this case, FFX is unsuitable. In 2012, Li et al. [7] proposed a solution based on cycle-walking [8]. However, cycle-walking can not guarantee stable efficiency, which is impractical in most applications. In this paper, we develop a new FPE method in Section 4.

2.2 Privacy-preserving Database Encryption

A number of research results [9–13] were proposed for privacy-preserving database encryption. However, most of them cannot provide the complete solution for general SQL-base operations over encrypted data. To support query over encrypted numerical data, Hakan et al. [9] firstly presented a basic

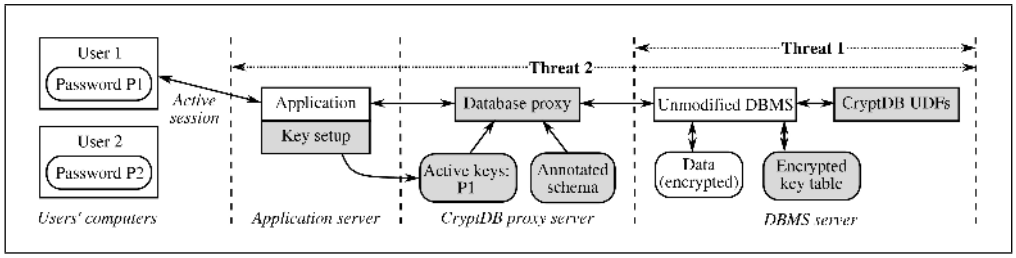


Fig. 1. CryptDB architecture

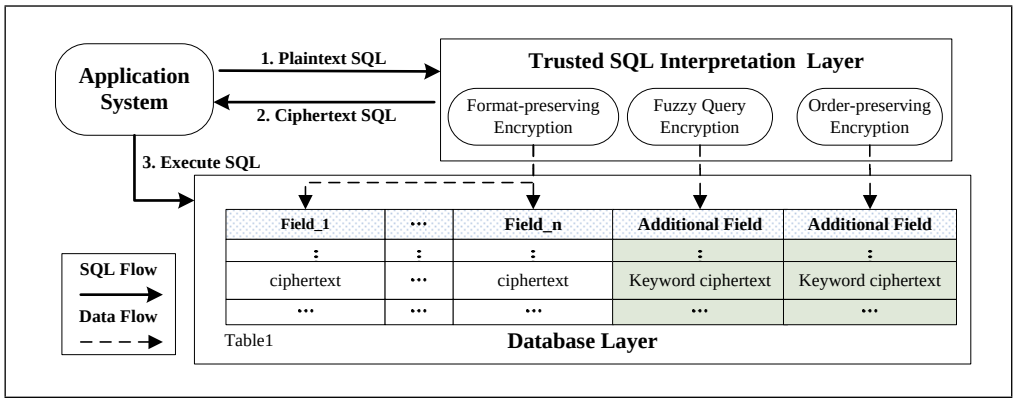


Fig. 2. L-EncDB Architecture

framework of how to ensure data security in "Database-As-Service" (DAS) model, in which a coarse query is executed by the database service provider. Based on this basic framework, Wu et al. [12] described a solution for query over encrypted character strings.

One of the most typical database encryption solutions is CryptDB [14], which explores an intermediate design point to provide confidentiality for applications that use database management systems (DBMSes). As shown in Fig. 1, CryptDB works by intercepting all SQL queries in a database proxy, which rewrites queries to execute over encrypted data (CryptDB assumes that all queries go through the proxy). The proxy encrypts and decrypts all the data, and changes some query operators, while preserving the semantics of the query. However, CryptDB is not designed for existing database applications and the DAS model of cloud storage. In cloud computing, users are able to store, modify and retrieve data from anywhere in the world, as long as they have access to the Internet. CryptDB changes the database structure and stores the ciphertexts generating by different encryption methods.

2.3 Other Related Work

The notion of order preserving encryption (OPE) [15–18] is another important encryption method in database to achieve the confidentiality while keeping the order of underlying plaintexts. Such a property allows users to perform

comparison and range query over encrypted data without decrypting them. Another notion related is searchable encryption (SE) [19,20], which provides functionalities to perform keyword search over encrypted data without decrypting them. There are also some other related privacy-preserving methods proposed for the security in database [21–23,14].

3 The New Advanced Secure Database System

3.1 System Model

The architecture of L-EncDB system is shown in Fig. 2. To provide data privacy protection solution and save upgrade cost for existing DB applications, L-EncDB system utilizes FPE technique to encrypt data.

There are two layers in the L-EncDB system, that is, the application system layer and database layer.

(1) SQL interpretation interface deployed in database application system will interpret all SQL sentences, encrypt constants in SQL sentence and form SQL sentence with ciphertext. For different SQL queries, different encryption methods (FPE, fuzzy query encryption (FQE) or OPE) are used. Note that SQL interpretation interface is viewed as an application program interface (API).
(2) Database layer will only provide data services and not allow developers to do any operation beyond SQL-based functions. As shown in Fig. 2, original fields are used to store ciphertext of original data, but additional fields are used to store additional ciphertexts for fuzzy query or range query.

Query. SQL interpretation interface is one of the key components of L-EncDB. As shown in Fig. 2, the application system receives the interpreted SQL sentence by calling SQL interface, which takes the original SQL sentence as the input. It then sends the interpreted SQL sentence to database. For most of general database applications, there are two types of SQL sentences: one is for data operation, such as insert, delete, and update. The other is for data query, such as exact query, join query, fuzzy query, and range query. We show how to process the interpretation for each type in our system.

As shown in Fig. 3.(a), for SQL data operation sentences, such as to insert, or to update, each constant in the query will be encrypted using FPE. To delete a record, the constant in the query will be encrypted using FPE as well. For fuzzy query or range query, SQL interpretation interface will further generate

Table 1
FPE for data types in DB

Types	Subtypes	SQL Field Type	FPE Scheme
Numeric	integer	$smallint, int$	FFSEM[5]
	decimal	$numeric$, $float$	
Char	finite length	$nchar$, $nvarchar$	FFX[3]
	finite space	$char$, $varchar$	MR-FPE
Datetime	N/A	$datetime$	Liu et al.[6]
Binary	N/A	$binary$, $varbinary$	Block cipher

the query for ciphertext by using FQE or OPE and store it into an additional filed.

As shown in Fig. 3.(b), for SQL exact keyword query, join query or nested query, the keywords will be encrypted by FPE. For fuzzy query or range query, the interface will use FQE or OPE to encrypt keywords, and change the query field to its corresponding additional field.

3.2 Security Notions

For L-EncDB system, the interface for SQL interpretation can be deployed at client side or the application service layer. We assume that there exist authentication and access control methods to protect the key used in L-EncDB.

We consider two types of attackers for L-EncDB: (1) attackers with access to database, including DBA or cloud service provider. They have access to the encrypted data and DB structure; (2) attackers with access to both application system and database. In another word, they are able to access SQL interpretation interface deployed in DB applications, construct SQL sentences with plaintext, gain interpreted SQL sentences with encrypted data, and view all fields and structure of database.

3.3 SQL-based Data Operations

L-EncDB uses SQL interpretation interface to interpret all SQL sentence, which is viewed as an API that can be flexibly used by developers. Next, we describe the interpretation processes in details and show how L-EncDB supports SQL operations and queries over encrypted data.

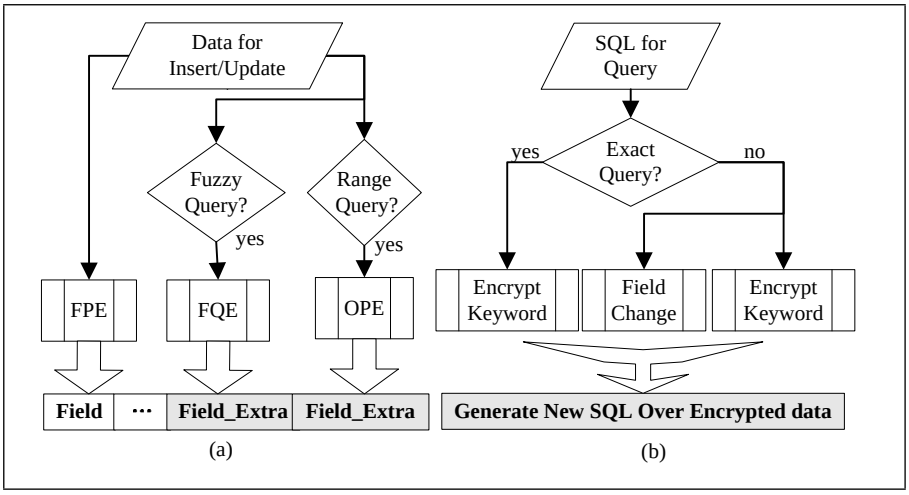


Fig. 3. Interpreting SQL sentences

3.3.1 Basic SQL Operations

For basic data operation (insert, update, and delete) and query (exact query, join query, and nested query), the interface for SQL interpretation will replace the plaintexts with the corresponding ciphertexts encrypted by FPE. Note that FPE is deterministic, which means that SQL-based data operation and query operation can be directly executed over the encrypted database. For example, "Insert into Table1(Field1, Field2) values (*String1*, *String2*)", will be interpreted to "Insert into Table1(Field1, Field2) values (fpe_k (*String1*), fpe_k(*String2*))", where fpe is the adopted FPE algorithm, k is the selected encryption key, and $fpe_k(x)$ means to encrypt x with fpe.

Similarly, the interface for SQL interpretation will replace the constants in the queries with ciphertexts of FPE in the following SQL sentences:

- *Update*, "Update Table1 set Field2=*String3* where Field1=*String1*";
- *Delete*, "Delete from Table1 where Field1 = *String1*";
- *Exact query*, "Select * from Table1 where Field1 = *String1*";
- *Join query*, "Select Table1.* from Table1,Table2 where Table1.Field1= Table2.Field1 and Table2.Field2=*String1*";
- *Nested query*, "Select * from Table1 where Field1 in (select Field1 from Table2 where Field2=*String3*)".

3.3.2 Advanced Queries

Range query over encrypted data. For range query, the interface applies OPE to generate ciphertexts and store them in additional fields. For example, for a SQL sentence like "Insert into Table1(Field1) values (*String1*)", where Field1 is for range query, the interpreted SQL sentence will be "Insert into Table1(Field1, `Field1Extra`) values (fpe_k(*String1*),OPE_k(*String1*))", where

`Field1Extra` is additional field for Field1, OPE is adopted OPE scheme and $OPE_k(x)$ means to encrypt x with OPE scheme using encryption key k.

To perform range query, the SQL interpretation interface will change the query filed into its additional field and generate the ciphertexts. For example, "Select * from Table1 where Field1>*key1*" will be interpreted as "Select * from Table1 where `Field1Extra`>OPE_k(*key1*)".

For data x, y and $x < y$, we can have $OPE_k(x) < OPE_k(y)$. Hence, the above interpreted SQL sentence will still work in the encrypted database.

Fuzzy query over encrypted data. The interpretation for fuzzy query is similar to range query. For example, for a SQL sentence like "Insert into Table1(Field1) values (*String1*)", where Field1 is for fuzzy query, the interpreted SQL sentence will be "Insert into Table1(Field1, `Field1Extra`) values (fpe_k(*String1*),FQE_k(*String1*))", where `Field1Extra` is additional field for Field1, FQE is adopted FQE scheme and $FQE_k(x)$ means to encrypt x with FQE scheme using encryption key k.

To perform fuzzy query, the interface changes the query filed to its additional field and generate keyword ciphertexts. For example, "Select * from Table1 where Field1 like '%*key1*%*key2*%"' will be interpreted as "Select * from Table1 where `Field1Extra` like '%FQE_k(*key1*)%FQE_k(*key2*)%"'.

To ensure the interpreted SQL sentence works in encrypted database, an FQE scheme supporting SQL-based fuzzy query over encrypted data is required. In CryptDB, Popa et al. proposed a keyword search scheme based on SE scheme [19]. However, this scheme supports only full-word keyword searches but not arbitrary regular expressions.

The idea of adopted FQE scheme in [24] is very simple. For an n-character string $D = d_1 \parallel d_2 \parallel \cdots \parallel d_n$, where $\parallel$ denotes concatenation, it replaces each character with its ciphertext: for each character $d_i, 1 \leq i \leq n$, FQE scheme firstly expands it to l-characters string by $str \leftarrow d_i \parallel \overbrace{11\cdots1}^{l-1}$, and secondly encrypts str to str' using character FPE such as FFX, then hashes str' into a short number using short hash function in [25], and finally, FQE scheme encodes the resulting short number to a Unicode character. We define the $gen(c)$ as above cryptology function to output a Unicode character and assume the plaintext is $d_1 \parallel d_2 \parallel \cdots \parallel d_n$. Then, its query for ciphertext stored in additional field will be $gen(d_1) \parallel gen(d_2) \parallel \cdots \parallel gen(d_n)$. For a keyword *key1*=$d_i \parallel \cdots \parallel d_j$, $1 \leq i \leq j \leq n$, its ciphertext is computed as FQE_k(*key1*)=$gen(d_i) \parallel \cdots \parallel gen(d_j)$.

Table 2 compares L-EncDB with Popa et al.'s CryptDB model.

Table 2
Comparison between L-EncDB and CryptDB

Model	L-EncDB	CryptDB
Fuzzy query	Yes	Partial
Data operation	Application	DB proxy and UDFs
Change DB structure	Add fields	Anonymize tables and columns

(1) CryptDB can only partially support fuzzy query. Our proposed L-EncDB is more flexible and able to support the fuzzy query.
(2) On data operation, the SQL-based operations are directly executed in DBMS. Both data encryption and decryption are executed in application in L-EncDB. However, direct SQL-based operations cannot be supported in CryptDB, where the trusted database proxy is used to intercept all the SQL queries and decrypt their (encrypted) results. To intercept SQL queries and implement encryption and decryption, CryptDB is required to build corresponding UDFs on the DBMS server.
(3) On database structure, CryptDB anonymizes each table and column name to achieve confidentiality. L-EncDB preserves most of original DB structure to reduce the cost of application codes. Hence, for existing database applications, L-EncDB is more suitable and lightweight when enterprises and organizations outsource data storage to third-party cloud providers.

In short, compared with the other database encryption solutions such as CryptDB, L-EncDB is lightweight to support SQL-based operation directly in DBMS and can be flexibly deployed in database applications.

4 New FPE Scheme for Character String

As described in section 2.1, there is no suitable FPE scheme for *varchar* data type with the restriction that the ciphertext has the same length with its corresponding plaintext. In this section, we propose a new FPE scheme for character string with arbitrary data type, which will be used in L-EncDB.

4.1 Preliminary

Throughout the rest of the paper, we let $Chars$ be a multi-byte character set, and $Chars^*$ be character strings over $Chars$ of any length. Moreover, for any set S, let $|S|$ be the number of elements in S. For a multi-byte character set $Chars$, it can be divided into subsets and each subset contains characters

of same size. Let $Chars_m$ be a subset containing all characters with storage size m. Let $cmin = min(|Chars_i|, i = 1, 2, \cdots, I)$, $cmax = max(|Chars_i|, i = 1, 2, \cdots, I)$, respectively be the number of members in the smallest and biggest subset of $Chars$.

The message space of character strings is described as $\mathcal{X}[Chars] = \{X|X \in Chars^*\}$. Given any two character strings $A, B \in \mathcal{X}[Chars]$, denote $A \parallel B$ as their concatenation. $\forall\ X \in \mathcal{X}[Chars] \Leftrightarrow X = x_1 \parallel x_2 \parallel \cdots \parallel x_i \parallel \cdots \parallel x_n, x_i \in Chars$. For any string $X \in \mathcal{X}[Chars]$, let $l(X)$ and $s(X)$ denote the length and storage size respectively. The storage size of any character $c \in Chars$ is also represented by $s(c)$.

We now give a review of the classical definition for FPE given by Morris et al. [2].

Definition 1 *A format-preserving encryption scheme is a function* $F : \mathcal{K} \times \mathcal{N} \times \mathcal{T} \times \mathcal{X} \to \mathcal{X} \times \{\perp\}$*, where* $\perp \neq \mathcal{X}$*, and nonempty sets* $\mathcal{K}$*,* $\mathcal{N}$*,* $\mathcal{T}$*,* $\mathcal{X}$ *denote the* key space*,* format space*,* tweak space *and* domain*, respectively.*

There are two kinds of character data, that is, *nvarchar* and *varchar*.

(1) If a field is defined as $nvarchar(n)$, it means that the field can store arbitrary character string with length (or character number) not more than n. The FFX method [3] is suitable for this type of domain. For example, for a plaintext string "abcd" with the length of 4, its ciphertext encrypted by FFX is "eadf", which has the same length and each character is in the same character set.
(2) If a field is defined as $varchar(n)$, it means that the field can store arbitrary character string and the storage size is not more than n.

4.2 Problem Statement

New FPE. The following explains the new FPE for message space $\mathcal{X}[Chars]$:

Definition 2 *A Character FPE is a function* $F : \mathcal{K} \times \mathcal{N}[Chars] \times \mathcal{T} \times \mathcal{X}[Chars] \to \mathcal{X}[Chars]$*, where* $\mathcal{N}[Chars]$ *is the* $formatspace$ *of character strings,* $\mathcal{K}$*,* $\mathcal{T}$*,* $\mathcal{X}[Chars]$ *are respectively the* key space*,* tweak space *and* domain*.*

Let $\mathcal{X}[Chars]$ denote message space defined by format $\mathcal{N}$. In our FPE for strings with type of *varchar*, the format space is defined by both length and storage size, that is $\mathcal{N}[Chars] = \{(l(X), s(X))|X \in \mathcal{X}[Chars]\}$.

An example. Assume that a concrete $Chars$ is defined as {'a', 'b', 'c', 'd́', 'é'}. In $Chars$, s('a')=s('b')=s('c')=1, i.e., the storage size of character 'a', 'b' and 'c' is 1 byte. But s('d́')= s('é')=2, i.e., the storage size of character 'd́' and 'é' is 2 bytes. Thus, $Chars$ is a multi-byte character set.

To better express the format of string $X = x_1 \parallel x_2 \parallel \cdots \parallel x_i \parallel \cdots \parallel x_n$, where $x_i \in Chars, 1 \leq i \leq n$. Let $\Omega(X)$ be its structure and $\Omega(X) = \{\omega_1, \cdots, \omega_I\}$, where $\omega_i = |\{x_j \in X | s(x_j) = i\}|$ and ω_i is the number of characters in string X with storage size i. For example, for string X="abéédé́", its structure is $\Omega(X)$=$\{2, 3\}$, i.e., 2 characters of storage size 1 byte, 3 Latin characters of 2 bytes, and its format is $N(X) = (5, 8)$, which means that length is 5 and storage size is 8 bytes.

4.3 Scheme Description

4.3.1 Basic Idea

For a multi-byte character set $Chars$, we establish a mapping from a subset $Chars_m$ to an integer set $Z_n = \{0, 1, \cdots, n-1\}$, where n is the character elements of $Chars_m$, that is $n = |Chars_m|$. For each character c, let $v(c)$ be its mapping value in Z_n.

For strings with type of *varchar*, FPE will preserve both length and storage size. Each character c will be represented as "$< value, radix >$", where $value$ is mapping value of c in Z_n, i.e., $value = v(c)$, and $radix$ is the element number of $Chars_{s(c)}$, i.e., $radix = n = |Chars_{s(c)}|$. For convenience, let $r(c)$ be the $radix$ of character c.

A new FPE for character string based on Feistel network is given, where Modular addition is an important component. In the new FPE scheme, the result of addition has the same $radix$ as that of left operand because the *modulo* is $radix$ of left operand. Thus, the output has the same format as the input. We call such modular addition as "multi-radix modular addition" and denote it as $\boxplus$, while $\boxminus$ as its inverse operation.

Definition 3 *For x and y, which are represented as "$< v(x), r(x) >$" and "$< v(y), r(y) >$" respectively, the multi-radix modular addition is defined as: $x \boxplus y$=$(v(x) + v(y))$ mod $r(x)$. Its inverse is defined as $x \boxminus y$=$(v(x) - v(y))$ mod $r(x)$.*

An example. Assume that we have a multi-byte character set $Chars$={'a', 'b', 'c', 'd́', 'é'} and two subsets, that is, $Chars_1$={'a', 'b', 'c'} and $Chars_2$={'d́', 'é'}. We also assume that we have two characters x='a' and y='é', which are represented as "<0, 3>" and "<1, 2>" respectively. "$x \boxplus y$" is computed by

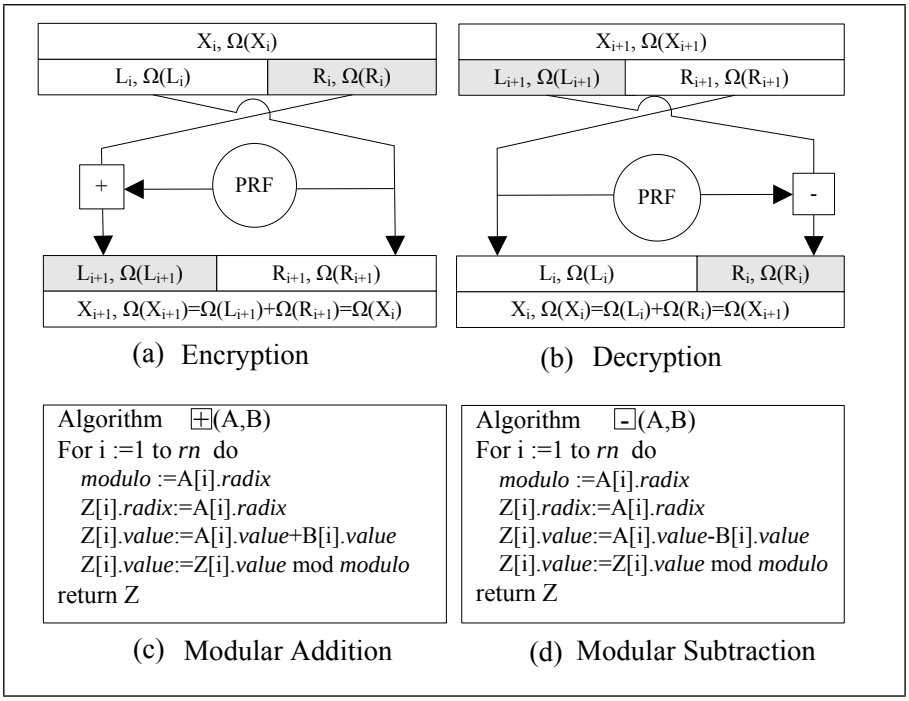


Fig. 4. FPE for character string and modular operations

'a'⊞'é'=(0+1) *mod* 3 and its result is "<1, 3>", which represented by character 'b' in $Chars_1$. The inverse operation is defined by 'b'⊟'é'=(1-1) *mod* 3 and its result is "<0, 3>", which represented by character 'a' in $Chars_1$. Similarly, the addition "$y \boxplus x$" is defined by 'é'⊞'a' =(1+0) *mod* 2 and results in "<1, 2>", which represented by character 'é' in $Chars_2$. The inverse operation is defined by 'é'⊟'a'=(1-0) *mod* 2 and results in "<1, 2>", which represented by character 'é' in $Chars_2$.

4.3.2 Description

Our new FPE can be described by three algorithms, that is, *Setup*, *Encrypt*, and *Decrypt*.

Setup: It generates the initial parameters, including the encryption key k and the number of Feistel rounds rn.

Encrypt: It takes as input the string X, key k and the round number rn. For the i-th round, its process is shown in Fig. 4. More specifically, three steps will be included. First, it divides the input X_i into a left part L_i and a right part R_i. L_i is returned as its right part, $L_{i+1} = R_i \boxplus PRF(L_i)$ as left part, in which PRF is instantiated by AES-CBC. Fig. 4 also describes the *multi-radix modular addition* algorithm, which ensures $\Omega(L_{i+1}) = \Omega(R_i)$ and $\Omega(X_{i+1}) = \Omega(X_i)$, i.e., the output of i-th round has the same format as the input.

Decrypt: It takes as input string X', key k and the round number rn. As shown in Fig. 4.(b), it divides input X_{i+1} into L_{i+1} and R_{i+1}. Then it outputs R_{i+1} as the left part, $R_i = L_{i+1} \boxminus PRF(R_{i+1})$ as the right part. The *multi-radix modular subtraction* algorithm is described in Fig. 4.(d), which ensures $\Omega(R_i) = \Omega(L_{i+1})$ and $\Omega(X_{i+1}) = \Omega(X_i)$, i.e., the output of this Feistel round has the same format as its input.

4.4 Security analysis

According to [2], the PRP security notion under chosen plaintext attack, i.e., PRP-CPA, is defined by PRP game $PRP_{\xi}^{\mathcal{A}}$ between challenger $\mathcal{C}$ and adversary $\mathcal{A}$ as follows.

Setup: $\mathcal{C}$ selects a boolean value $b \leftarrow \{0, 1\}$ in random, generates the symmetric key K for FPE scheme ξ in domain $\mathcal{X}$, and selects a uniform permutation π on $\mathcal{X}$.

Phase: $\mathcal{A}$ can adaptively ask $\mathcal{C}$ for the corresponding ciphertext for any string $X \in \mathcal{X}$. If $b = 0$, $\mathcal{C}$ responds with $\pi(X)$, otherwise with $\xi_K(X)$.

Guess: $\mathcal{A}$ outputs a predicate value b'. If $b = b'$, the security game returns 1. Otherwise, it returns 0.

Definition 4 *An FPE scheme ξ is PRP-CPA secure if any polynomial time adversary has only a negligible advantage in PRP game shown above, where the advantage is defined as*

$$\boldsymbol{Adv}_{\xi}^{PRP}(\mathcal{A}) \stackrel{def}{=} Pr[PRP_{\xi}^{\mathcal{A}} \Rightarrow 1] - \tfrac{1}{2}$$

Theorem 1 *If the underlying round function is a secure pseudo random permutation, our new FPE scheme achieves the PRP security.*

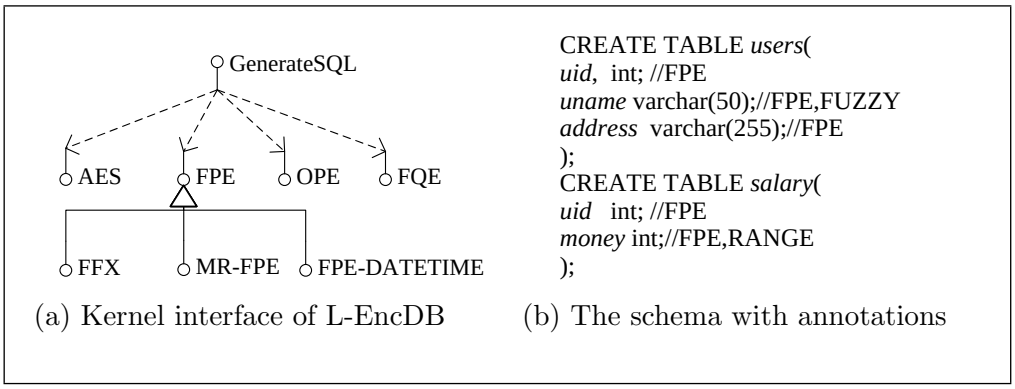
Proof 1 *Assume that there exists an adversary breaks the security of our FPE scheme, a simulator will be built to show the insecurity of PRF, that is, the simulator could break the indistinguishability of PRF from a truly random permutation P. Next, we show how to use $\mathcal{A}$ to construct a distinguisher $\mathcal{D}$. Whenever $\mathcal{A}$ queries the encryption oracle with a string X, $\mathcal{D}$ selects a random value $b = \{0, 1\}$. If $b = 0$, it computes X' with PRF, otherwise it computes X' with P. Finally, $\mathcal{D}$ returns the result X' back to $\mathcal{A}$.*

It can be seen that the view of $\mathcal{A}$ when run as a sub-routine by $\mathcal{D}$ is distributed identically to the view of $\mathcal{A}$ in game $PRP_{\xi}^{\mathcal{A}}$. Thus if the adversary can succeed in attacking the FPE, there is a distinguisher $\mathcal{D}$ having the same probability on distinguishing PRF with P in polynomial time.

5 Implementation and Evaluation

5.1 Implementation Details

The experiment for our L-EncDB system is conducted to evaluate its efficiency. We implement the system through an open kernel API of C++ DLL for L-EncDB, called `GenerateSQL`, which takes as input a plaintext SQL sentence and outputs interpreted encrypted SQL sentence. To implement FPE schemes, AES and big number in open source library *polarssl* are used. Users can improve their database security based on such a DLL with authentication and access control mechanisms.



(a) Kernel interface of L-EncDB (b) The schema with annotations

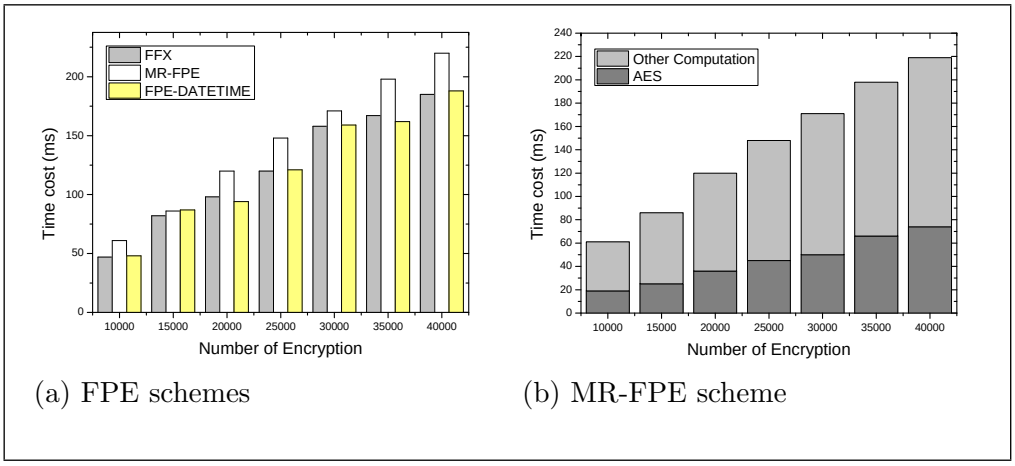
Fig. 5. Implementation details

As shown in Fig. 5.(a), `GenerateSQL` API uses techniques of FPE ("MR-FPE" is used to identify the proposed FPE scheme for character string), OPE, FQE to interpret SQL sentences. To provide correct interpretation to users, it requires that: (1) the fields of fuzzy query field and range query are named as `Field_Fuzzy` and `Field_Order` respectively; (2) DB structure is open to L-EncDB DLL. That is, L-EncDB DLL must know DB structure, and the fields should be encrypted for fuzzy query or range query. To achieve this goal, the schema with annotations shown in Fig. 5.(b) is used, in which annotation "FPE" denotes the encryption of FPE, "FUZZY" and "RANGE" denote fuzzy query and range query respectively.

The SQL sentences interpretation performs as follows. Firstly, it analyzes SQL sentence, decides operations and tables to execute. Secondly, based on operation and table structure, it decides whether encryption, fuzzy query or range query are needed. Finally, it completes the interpretation of SQL sentences with suitable encryption schemes.

Table 3
Execution time of encryption

Scheme	Encryption	Decryption
AES	0.00015 ms	0.00015 ms
OPE	9.80000 ms	0.00000 ms
FQE	FFX*n ms	0.00000 ms



(a) FPE schemes (b) MR-FPE scheme

Fig. 6. Execution time of FPE schemes

5.2 Experimental Evaluation

L-EncDB is external encryption mechanism independent from database. The adopted encryption algorithms affect DB operation performance. To evaluate its performance, two issues are addressed: 1) the performance of FPE encryption algorithms for batch data encryption, 2) the performance of OPE.

As shown in Fig. 5.(a), encryption algorithms include FFSEM, FFX, MR-FPE, FPE-DateTime, FQE and OPE. We programm for these encryption algorithms and experiment for their average execution time. For FFSEM, FFX and MR-FPE, we set Feistel rounds number as 12, use AES-CBC to construct random function. All our experiments are performed in Windows 7 operation system with the Intel(R)Core(TM)i5-3337U @ 1.80GHZ and 4GB memory.

The performance evaluation of encryption algorithms are shown in Fig. 6 and Table 3. From them, we can get the following results:

(1) the average execution time of AES is about 0.15us;
(2) as shown in Fig. 6.(a), two FPE algorithms based on Feistel network, i.e., FFX and MR-FPE, the average execution times are very close, which are around 30 times of that of AES algorithm. FPE-DateTime algorithm uses FFX (with the character set of {'0',$\cdots$,'9'}) as integer FPE to compute

the offset; hence average execution time is also close to FFX;

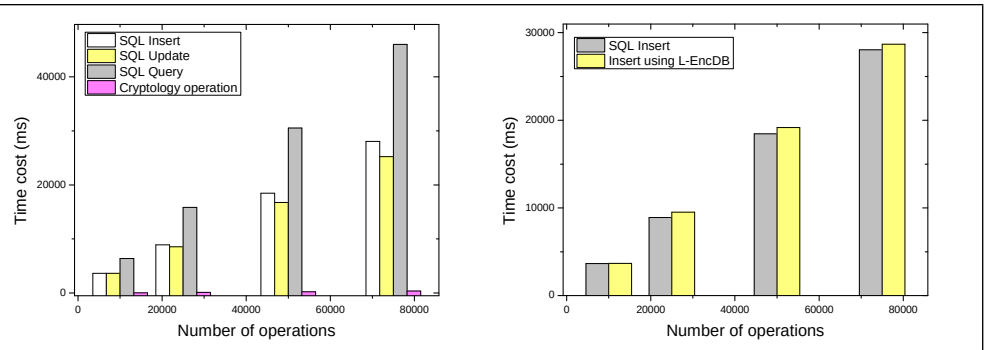
(3) as shown in Fig. 6.(b), the average execution time of MR-FPE is about 5us, which is about 30 times of that of AES. For each run, MR-FPE algorithm executes not only AES algorithm and modular arithmetic, but also coding and decoding operations. Among them, the execution time of AES algorithm is about 1.8us for 12 Feistel rounds, but other operations including modular arithmetic and coding operations cost about 3.2us.

(4) OPE's average execution time is the lowest; its decryption time is 0ms for never decrypting OPE ciphertext in L-EncDB;

(5) FQE's execution time is related to the length n of plaintext and FPE's execution time, and in practical applications, the data for fuzzy query is often less than 100, thus FQE scheme will be more efficient than OPE.

In practical applications using L-EncDB, for each DB operation, the rounds of encryption are different, which are linearly increasing with the number of fields in the operation. For a SQL insert sentence with 30 fields and no range query or fuzzy query, the execution time for each insert operation is about 0.09ms, i.e., the system can interpret 11000 SQL sentences within 1s. Thus, the system can meet needs for most of applications.

Evaluation in web applications. The L-EncDB is a lightweight mechanism and can be easily deployed in various kinds of database applications. In our evaluation, we use it to build a secure website based on the above implementation details. The webpage programming language is Java (jsp, javabean, jdbc, etc), and the web server is Resin (with version of 3.1.12). Java Native Interface (JNI) technique is used to call functions of C++ DLL. To construct the test platform, we use Java language to construct an application, in which the open source library named "HTTPClient" is used to visit the specified webpage in our website. Moreover, the Mysql is selected as the database server.



(a) SQL operations (b) Two kinds of Insert operation

Fig. 7. Comparison between SQL operations and encryption

To evaluate the performance in real web applications, we focus on the comparison of execution time between SQL operations and L-EncDB operations. We

test the basic SQL operations including insert, update and query. From Fig. 7, we can see that the execution time of SQL query operation is longer than the other SQL operations. As shown in Fig. 7.(a), the average execution time of query is about 0.6ms, but that of insert and update is only about 0.3ms. Execution time of SQL insert operation using L-EncDB is very close to that without encryption as shown in Fig. 7.(b).

6 Extension of L-EncDB to NoSQL Database Encryption

With the advent of the Internet Web2.0 site, traditional SQL-based relational databases cannot be applied and many challenges arise, including:

(1) Performance problem. To provide dynamic pages and information for user, thousands of read and write requests are produced each second in web2.0 site, and results in high concurrent load of database. However, SQL-based databases can not meet this requirement.
(2) Storage problem. In big data background, each user will produce massive dynamic information. For example, in *Friendfeed*, 250 million of user dynamic records are generated in a month. In this case, the efficiency will be unbearable when the traditional SQL-based relational databases are used to query over such records.

To solve the above challenges, NoSQL (Not Only SQL) databases have been developed and widely used in practice. *BigTable* of Goole and *Dynamo* of Amazon are successful implementation of NoSQL.

In the cloud computing, more and more enterprises also need to outsource NoSQL database to construct their business applications. Our L-EncdDB can be easily extended to NoSQL database encryption with the following properties.

(1) Independence of database. L-EncDB are always deployed in the client side, in which data encryption and decryption are all executed in the application.
(2) Queries over encrypted data. In L-EncDB, exact query can be reserved because of the deterministic encryption. The range query can also be applied for the order-preserving encryption.

After extending the implementation of SQL interpretation interface according to operation syntax of different NoSQL databases, L-EncDB can be used to encrypt the data in these databases while preserving the query syntax. For example, in the case of *MongoDB*, SQL interpretation interface can be implemented by replacing the constants in the queries:

- *Insert*, "db.users.save({name:"*String1*" age:*Age1*, country:"*String2*"})" will insert a record with three attribute name, age and country;
- *Update*, " db.users.update({ "name": "*String1*"}, { $set:{ "country":"*String3*" }})" will update "country" as "*String3*" in the record in which the value of attribute "name" is "*String1*";
- *Delete*, "db.users.remove({"name": "*String1*"})" will delete the record in which the value of attribute "name" is "*String1*";
- *Exact query*, "db.users.find({"name": "*String1*"})" will query the record in which the value of attribute "name" is "*String1*";
- *Range query*, "db.users.find({"age":{$in:[*Age1*, *Age2*]}})" will query the records in which the value of attribute"age" is in the range of *Age1* to *Age2*;
- *Fuzzy query*, *MongoDB* uses regular expression to achieve fuzzy query, for example, the SQL sentence "SELECT * FROM users where name like 'A%"' will be "db.users.find("name" :/^A/)". So, the fuzzy query expression can be interpreted as "db.users.find("name" :/^FQE_k(A)/)".

7 Conclusion and Future Work

In this paper, we proposed a novel L-EncDB mechanism, which provides a secure and privacy-preserving data utilization for outsourced database such as SQL-based encryption and query mechanism. Such a new mechanism for database does not change the data structure after encryption and can be efficiently realize data utilization such as privacy-preserving knowledge extraction, after outsourcing database into the cloud. In this new mechanism, it utilizes a core interface provided as API to interpret SQL operations, which allows to protect sensitive information in database applications. Experimental results demonstrate that the new L-EncDB is efficient and can be applied to big database for privacy-preserving applications. Finally, we also showed how to extend our L-EncDB to realize the privacy-preserving queries over encrypted NoSQL Database.

To make the L-EncDB mechanism more practical, SQL-based range query methods with better performance will be investigated in future to support comparison over ciphertexts. Especially, we will extend it to privacy-preserving knowledge extraction for outsourcing database, and further provide some practical data publishing methods suitable for our framework.

References

[1] J. Black, P. Rogaway, Ciphers with arbitrary finite domains, in: Topics in Cryptology–CT-RSA, Vol. 2271 of Lecture Notes in Computer Science, Springer, 2002, pp. 114–130.

[2] B. Morris, P. Rogaway, T. Stegers, How to encipher messages on a small domain: Deterministic encryption and the thorp shuffle, in: Advances in Cryptology–CRYPTO 2009, Vol. 5677 of Lecture Notes in Computer Science, Springer, 2009, pp. 286–302.

[3] M. Bellare, P. Rogaway, T. Spies, The ffx mode of operation for format-preserving encryption, NIST submission.

[4] V. T. Hoang, B. Morris, P. Rogaway, An enciphering scheme based on a card shuffle, in: Advances in Cryptology–Crypto 2012, Vol. 7417 of Lecture Notes in Computer Science, Springer, 2012, pp. 1–13.

[5] S. Terence, Feistel finite set encryption mode., NIST Proposed Encryption Mode.

[6] Z. Liu, C. Jia, J. Li, Format-preserving encryption for datetime, in: Intelligent Computing and Intelligent Systems, Vol. 2, Springer, 2010.

[7] M. Li, Z. Liu, J. Li, C. Jia, Format-preserving encryption for character data, Journal of Networks (2012) 1239–1244.

[8] J. Li, C. Jia, Z. Liu, Cycle-walking revisited: consistency, security, and efficiency, in: Security and Communication Networks, 2012.

[9] H. Hakan, L. Bala, L. Chen, M. Sharad, Executing sql over encrypted data in the database-service-provider model, in: Proceedings of the 2002 ACM SIGMOD international conference on Management of data, ACM, 2002, pp. 216–227.

[10] S. Evdokimov, O. Guenther, Encryption techniques for secure database outsourcing, in: Computer Security – ESORICS 2007, Vol. 4734 of Lecture Notes in Computer Science, Springer, 2007, pp. 327–342.

[11] C. Wang, Q. Wang, K. Ren, Towards secure and effective utilization over encrypted cloud data, in: the 31st International Conference on Distributed Computing Systems Workshops, IEEE, 2011, pp. 282–286.

[12] Z. Wu, G. Xu, Z. Yu, X. Yi, E. Chen, Y. Zhang, Executing sql queries over encrypted character strings in the database-as-service model, in: Knowledge-Based Systems, Vol. 35, 2012, pp. 332–348.

[13] K. Choy, W. Lee, H. C. Lau, L. Choy, A knowledge-based supplier intelligence retrieval system for outsource manufacturing, in: Knowledge-Based Systems, Vol. 18, 2005, pp. 1–17.

[14] R. A. Popa, N. Zeldovich, H. Balakrishnan, Cryptdb: protecting confidentiality with encrypted query processing, in: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, ACM, 2011, pp. 85–100.

[15] R. Agrawal, J. Kiernan, R. Srikant, Y. Xu, Order preserving encryption for numeric data, in: Proceedings of the 2004 ACM SIGMOD international conference on Management of data, ACM, 2004, pp. 563–574.

[16] A. Boldyreva, N. Chenette, Y. Lee, A. O'Neill, Order-preserving symmetric encryption, in: Advances in Cryptology–EUROCRYPT 2009, Vol. 5479 of Lecture Notes in Computer Science, Springer, 2009, pp. 224–241.

[17] R. A. Popa, F. H. Li, N. Zeldovich, An ideal-security protocol for order-preserving encoding., in: Proc. of the 34th IEEE Symposium on Security and Privac, 2013.

[18] I. Yakut, H. Polat, Estimating nbc-based recommendations on arbitrarily partitioned data with privacy, in: Knowledge-Based Systems, Vol. 36, 2012, p. 353C362.

[19] D. X. Song, D. Wagner, A. Perrig, Practical techniques for searches on encrypted data, in: Proceedings of the 21st IEEE Symposium on Security and Privacy, IEEE, 2000, pp. 44–55.

[20] D. Boneh, B. Waters, Conjunctive, subset, and range queries on encrypted data, in: 4th Theory of Cryptography Conference, Vol. 4392 of Lecture Notes in Computer Science, Springer, 2007, pp. 535–554.

[21] K. W. Lin, Y.-C. Lo, Efficient algorithms for frequent pattern mining in many-task computing environments, in: Knowledge-Based Systems, Vol. 49, 2013, pp. 10–21.

[22] M. I. M. A. S. Umer Khalid, Abdul Ghafoor, Cloud based secure and privacy enhanced authentication & authorization protocol, in: Knowledge-Based Systems, Vol. 22, 2013, pp. 680–688.

[23] A. S. Sattar, J. Li, X. Ding, J. Liu, M. Vincent, A general framework for privacy preserving data publishing, in: Knowledge-Based Systems, Vol. 54, 2013, pp. 276–287.

[24] Z. Liu, H. Ma, J. Li, C. Jia, Secure storage and fuzzy query over encrypted databases, in: Network and System Security, 2013, pp. 439–450.

[25] L. H. Nguyen, A. W. Roscoe, Short-output universal hash functions and their use in fast and secure data authentication, in: Fast Software Encryption, Vol. 7549 of Lecture Notes in Computer Science, Springer, 2012, pp. 326–345.

L-EncDB: A Lightweight Framework for Privacy-Preserving Data Queries in Cloud Computing

Jin Li [a,*]

[a] *School of Computer Science and Educational Software*
Guangzhou University, Guangzhou, 510006 P.R. China

Zheli Liu [b,*]

[b] *College of Information Technical Science*
Nankai University, P.R. China

Xiaofeng Chen [c]

[c] *State Key Laboratory of Integrated Service Networks (ISN)*
Xidian University, Xi'an, P.R. China

Fatos Xhafa [d]

[d] *Department of Languages and Informatics Systems*
Technical University of Catalonia, Spain

Xiao Tan, Duncan S. Wong [e]

[e] *Department of Computer Science*
City University of Hong Kong, Hong Kong

Abstract

With the advent of cloud computing, individuals and organizations have become interested in moving their databases from local to remote cloud servers. However, data owners and cloud service providers are not in the same trusted domain in practice. For the protection of data privacy, sensitive data usually have to be encrypted before outsourcing, which makes effective database utilization a very challenging task. To address this challenge, in this paper, we propose L-EncDB, a novel lightweight encryption mechanism for database, which i) keeps the database structure, and ii) supports efficient SQL-based queries. To achieve this goal, a new format-preserving encryption (FPE) scheme is constructed in this paper, which can be used to encrypt all types of character strings stored in database. Extensive analysis demonstrates

that the proposed L-EncDB scheme is highly efficient and provably secure under existing security model.

Key words: Data query, outsourcing, privacy, format-preserving encryption

1 Introduction

The ever-increasing amount of valuable digital data both at home and in business needs to be protected, since its irrevocable loss is unacceptable. The advent of cloud storage motivates enterprises and organizations to outsource data storage to third-party cloud providers, as evidenced by many real-life case studies. Cloud storage services promise to be a solution for this problem. In recent years, their popularity has increased dramatically. They offer user-friendly, easily accessible and costsaving ways to store and automatically back up arbitrary data, as well as data sharing between users and synchronization of multiple devices.

As in any existing application and system, security and privacy play an extremely important role for the success, and certainly raise new challenges among the many others that cloud storage is confronted with. Specifically, when entrusting data to the cloud, data owner also releases control over the data, resulting that their trust is put in the cloud service provider's integrity and in the security of its access control mechanisms. However, individuals and especially businesses hesitate to entrust their data to cloud storage services since they fear that they will lose control over it. Recent successful attacks on cloud storage providers have exacerbated these concerns. The providers are trying to alleviate the situation and have taken measures to keep their customers' data secure. The simple and popular solutions adopted for data privacy are traditional encryption techniques such as public key encryption or symmetric key encryption. Through these encryption methods before outsourcing, the security of users's data can be protected.

However, traditional database encryption will change the data structure of original data, and results in the impracticability of database application for various kinds of SQL operations. If the data structure is changed, it cannot support data operations over ciphertext such as range query and fuzzy query, etc. Especially, there has been considerable recent interest in the paradigm of

* Corresponding author. Jin Li and Zheli Liu contribute to this work equally.
Email addresses: jinli71@gmail.com (Jin Li), liuzheli1978@163.com (Zheli Liu).

data mining-as-service: a company (data owner) lacking in expertise or computational resources can outsource its mining needs to a third party service provider. However, both the outsourced database and the knowledge extracted through data mining are considered private property of the data owner in many applications. Thus, to protect data privacy while realizing data mining and knowledge extraction, the data owner is required to transform its data without changing its structure before outsourcing.

Contribution. To realize effective data utilization after secure outsourcing, we propose a lightweight encrypted database mechanism denoted by L-EncDB. This new mechanism is able to protect sensitive information while keeping the data structure in outsourcing service for big database application. In the proposed L-EncDB system, the encryption and query are based on SQL. Through only one interface, all SQL sentences for database can be interpreted.

Furthermore, based on format-preserving encryption (FPE) technique and a new character string FPE scheme, L-EncDB can be implemented to preserve data type and length in ciphertext. It enables i) to encrypt data and store them without changing original database structure, ii) to perform SQL operations on all kinds of databases, including text database such as SQLite and Access, iii) to support SQL-based operations including advanced fuzzy and range queries.

Innovation. In this paper, a novel FPE scheme with the method of "multi-radix modular addition" is proposed to support the L-EncDB lightweight framework for privacy-preserving outsourced database. The new proposed FPE can preserve both length and storage size of character strings, which cannot be efficiently achieved in the traditional FPE schemes. Based on the FPE scheme, data operations such as data mining and SQL-based queries can be directly executed over ciphertexts in the proposed L-EncDB framework. Furthermore, L-EncDB framework can be extended to text database (such as SQLite used in mobile) and NoSQL databases, which have not been considered in the previous related work.

1.1 Organization

The rest of this paper proceeds as follows. In Section 2, we give a survey for the related work to ours. In Section 3, we propose the system architecture and construction method for the L-EncDB system. In Section 4, we propose a practical construction of FPE for character string. Its security and performance analysis is also given in this section. In Section 5, we present the implementation of prototype for L-EncDB with the proposed FPE, and in Section 6, we present an extension of the L-EncDB to NoSQL database encryption. Finally

we draw conclusion and show the future work in Section 7.

2 Related Work

We briefly discuss FPE technique and privacy-preserving database encryption solutions in this Section.

2.1 FPE

The notion of FPE [1–4] has been proposed to generate ciphertext with the same format as plaintext while encrypting sensitive information. More specifically, FPE can keep data type and length in the ciphertext, therefore, without changing database structure and field type. Thus, the use of FPE enables upgrading database security in a transparent way. The goal of FPE is to generate ciphertext which falls in the same *domain* as the plaintext. Some practical FPE schemes have been proposed for simple domains such as integer [5], character data [3] and datetime [6]. Character data is the common data type in database, which appears in the form of character strings, i.e., the finite sequences of characters from some character sets. However, there is no suitable character FPE solution to preserve both length and storage size of strings above. For a string with character from iso-8859-1 or ASCII, where the storage size of each character is 1 byte, the length of string is equal to its storage size, and FFX is also suitable in this case.

However, most of character sets are represented using more powerful encoding formats, and different characters may require different byte counts to represent. In this paper, such a character set is called "multi-byte character set". Consider a character string of length n with each character in UTF-8, its storage size will be from n bytes to $4n$ bytes. In this case, FFX is unsuitable. In 2012, Li et al. [7] proposed a solution based on cycle-walking [8]. However, cycle-walking can not guarantee stable efficiency, which is impractical in most applications. In this paper, we develop a new FPE method in Section 4.

2.2 Privacy-preserving Database Encryption

A number of research results [9–13] were proposed for privacy-preserving database encryption. However, most of them cannot provide the complete solution for general SQL-base operations over encrypted data. To support query over encrypted numerical data, Hakan et al. [9] firstly presented a basic

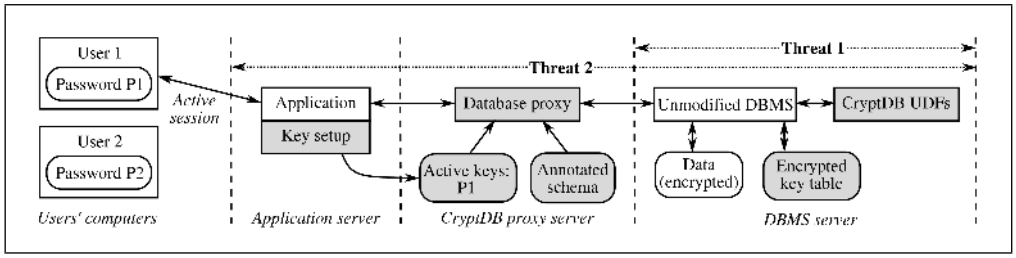


Fig. 1. CryptDB architecture

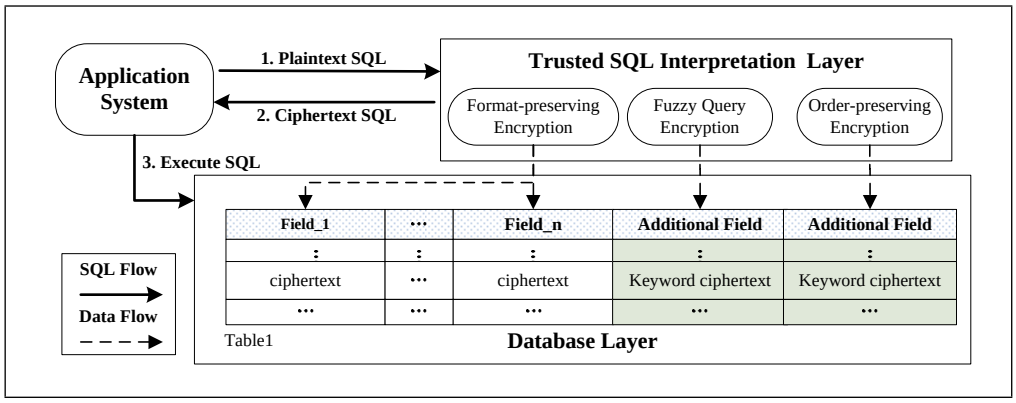


Fig. 2. L-EncDB Architecture

framework of how to ensure data security in "Database-As-Service" (DAS) model, in which a coarse query is executed by the database service provider. Based on this basic framework, Wu et al. [12] described a solution for query over encrypted character strings.

One of the most typical database encryption solutions is CryptDB [14], which explores an intermediate design point to provide confidentiality for applications that use database management systems (DBMSes). As shown in Fig. 1, CryptDB works by intercepting all SQL queries in a database proxy, which rewrites queries to execute over encrypted data (CryptDB assumes that all queries go through the proxy). The proxy encrypts and decrypts all the data, and changes some query operators, while preserving the semantics of the query. However, CryptDB is not designed for existing database applications and the DAS model of cloud storage. In cloud computing, users are able to store, modify and retrieve data from anywhere in the world, as long as they have access to the Internet. CryptDB changes the database structure and stores the ciphertexts generating by different encryption methods.

2.3 Other Related Work

The notion of order preserving encryption (OPE) [15–18] is another important encryption method in database to achieve the confidentiality while keeping the order of underlying plaintexts. Such a property allows users to perform

comparison and range query over encrypted data without decrypting them. Another notion related is searchable encryption (SE) [19,20], which provides functionalities to perform keyword search over encrypted data without decrypting them. There are also some other related privacy-preserving methods proposed for the security in database [21–23,14].

3 The New Advanced Secure Database System

3.1 System Model

The architecture of L-EncDB system is shown in Fig. 2. To provide data privacy protection solution and save upgrade cost for existing DB applications, L-EncDB system utilizes FPE technique to encrypt data.

There are two layers in the L-EncDB system, that is, the application system layer and database layer.

(1) SQL interpretation interface deployed in database application system will interpret all SQL sentences, encrypt constants in SQL sentence and form SQL sentence with ciphertext. For different SQL queries, different encryption methods (FPE, fuzzy query encryption (FQE) or OPE) are used. Note that SQL interpretation interface is viewed as an application program interface (API).
(2) Database layer will only provide data services and not allow developers to do any operation beyond SQL-based functions. As shown in Fig. 2, original fields are used to store ciphertext of original data, but additional fields are used to store additional ciphertexts for fuzzy query or range query.

Query. SQL interpretation interface is one of the key components of L-EncDB. As shown in Fig. 2, the application system receives the interpreted SQL sentence by calling SQL interface, which takes the original SQL sentence as the input. It then sends the interpreted SQL sentence to database. For most of general database applications, there are two types of SQL sentences: one is for data operation, such as insert, delete, and update. The other is for data query, such as exact query, join query, fuzzy query, and range query. We show how to process the interpretation for each type in our system.

As shown in Fig. 3.(a), for SQL data operation sentences, such as to insert, or to update, each constant in the query will be encrypted using FPE. To delete a record, the constant in the query will be encrypted using FPE as well. For fuzzy query or range query, SQL interpretation interface will further generate

Table 1
FPE for data types in DB

Types	Subtypes	SQL Field Type	FPE Scheme
Numeric	integer	$smallint, int$	FFSEM[5]
	decimal	$numeric$, $float$	
Char	finite length	$nchar$, $nvarchar$	FFX[3]
	finite space	$char$, $varchar$	MR-FPE
Datetime	N/A	$datetime$	Liu et al.[6]
Binary	N/A	$binary$, $varbinary$	Block cipher

the query for ciphertext by using FQE or OPE and store it into an additional filed.

As shown in Fig. 3.(b), for SQL exact keyword query, join query or nested query, the keywords will be encrypted by FPE. For fuzzy query or range query, the interface will use FQE or OPE to encrypt keywords, and change the query field to its corresponding additional field.

3.2 Security Notions

For L-EncDB system, the interface for SQL interpretation can be deployed at client side or the application service layer. We assume that there exist authentication and access control methods to protect the key used in L-EncDB.

We consider two types of attackers for L-EncDB: (1) attackers with access to database, including DBA or cloud service provider. They have access to the encrypted data and DB structure; (2) attackers with access to both application system and database. In another word, they are able to access SQL interpretation interface deployed in DB applications, construct SQL sentences with plaintext, gain interpreted SQL sentences with encrypted data, and view all fields and structure of database.

3.3 SQL-based Data Operations

L-EncDB uses SQL interpretation interface to interpret all SQL sentence, which is viewed as an API that can be flexibly used by developers. Next, we describe the interpretation processes in details and show how L-EncDB supports SQL operations and queries over encrypted data.

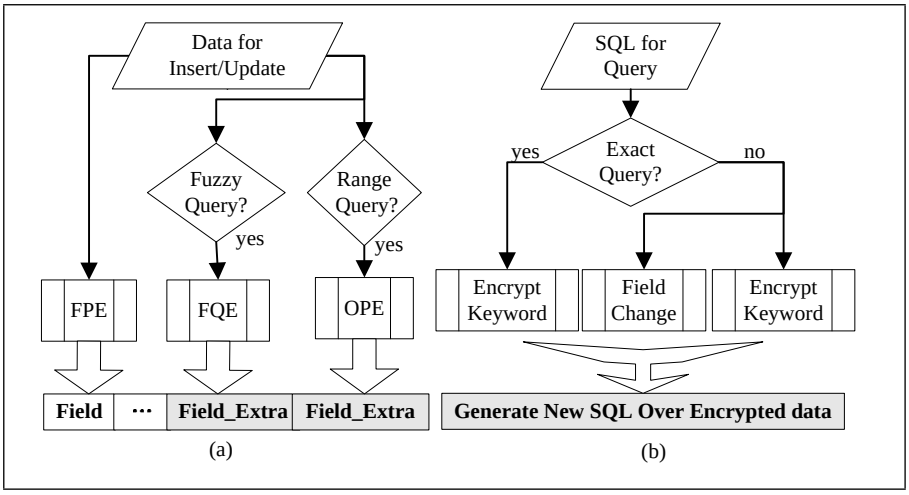


Fig. 3. Interpreting SQL sentences

3.3.1 Basic SQL Operations

For basic data operation (insert, update, and delete) and query (exact query, join query, and nested query), the interface for SQL interpretation will replace the plaintexts with the corresponding ciphertexts encrypted by FPE. Note that FPE is deterministic, which means that SQL-based data operation and query operation can be directly executed over the encrypted database. For example, "Insert into Table1(Field1, Field2) values (*String1*, *String2*)", will be interpreted to "Insert into Table1(Field1, Field2) values (fpe_k (*String1*), fpe_k(*String2*))", where fpe is the adopted FPE algorithm, k is the selected encryption key, and $fpe_k(x)$ means to encrypt x with fpe.

Similarly, the interface for SQL interpretation will replace the constants in the queries with ciphertexts of FPE in the following SQL sentences:

- *Update*, "Update Table1 set Field2=*String3* where Field1=*String1*";
- *Delete*, "Delete from Table1 where Field1 = *String1*";
- *Exact query*, "Select * from Table1 where Field1 = *String1*";
- *Join query*, "Select Table1.* from Table1,Table2 where Table1.Field1= Table2.Field1 and Table2.Field2=*String1*";
- *Nested query*, "Select * from Table1 where Field1 in (select Field1 from Table2 where Field2=*String3*)".

3.3.2 Advanced Queries

Range query over encrypted data. For range query, the interface applies OPE to generate ciphertexts and store them in additional fields. For example, for a SQL sentence like "Insert into Table1(Field1) values (*String1*)", where Field1 is for range query, the interpreted SQL sentence will be "Insert into Table1(Field1, `Field1Extra`) values (fpe_k(*String1*),OPE_k(*String1*))", where

`Field1Extra` is additional field for Field1, OPE is adopted OPE scheme and $OPE_k(x)$ means to encrypt x with OPE scheme using encryption key k.

To perform range query, the SQL interpretation interface will change the query filed into its additional field and generate the ciphertexts. For example, "Select * from Table1 where Field1>*key1*" will be interpreted as "Select * from Table1 where `Field1Extra`>OPE_k(*key1*)".

For data x, y and $x < y$, we can have $OPE_k(x) < OPE_k(y)$. Hence, the above interpreted SQL sentence will still work in the encrypted database.

Fuzzy query over encrypted data. The interpretation for fuzzy query is similar to range query. For example, for a SQL sentence like "Insert into Table1(Field1) values (*String1*)", where Field1 is for fuzzy query, the interpreted SQL sentence will be "Insert into Table1(Field1, `Field1Extra`) values (fpe_k(*String1*),FQE_k(*String1*))", where `Field1Extra` is additional field for Field1, FQE is adopted FQE scheme and $FQE_k(x)$ means to encrypt x with FQE scheme using encryption key k.

To perform fuzzy query, the interface changes the query filed to its additional field and generate keyword ciphertexts. For example, "Select * from Table1 where Field1 like '%*key1*%*key2*%"' will be interpreted as "Select * from Table1 where `Field1Extra` like '%FQE_k(*key1*)%FQE_k(*key2*)%"'.

To ensure the interpreted SQL sentence works in encrypted database, an FQE scheme supporting SQL-based fuzzy query over encrypted data is required. In CryptDB, Popa et al. proposed a keyword search scheme based on SE scheme [19]. However, this scheme supports only full-word keyword searches but not arbitrary regular expressions.

The idea of adopted FQE scheme in [24] is very simple. For an n-character string $D = d_1 \parallel d_2 \parallel \cdots \parallel d_n$, where $\parallel$ denotes concatenation, it replaces each character with its ciphertext: for each character $d_i, 1 \leq i \leq n$, FQE scheme firstly expands it to l-characters string by $str \leftarrow d_i \parallel \overbrace{11\cdots1}^{l-1}$, and secondly encrypts str to str' using character FPE such as FFX, then hashes str' into a short number using short hash function in [25], and finally, FQE scheme encodes the resulting short number to a Unicode character. We define the $gen(c)$ as above cryptology function to output a Unicode character and assume the plaintext is $d_1 \parallel d_2 \parallel \cdots \parallel d_n$. Then, its query for ciphertext stored in additional field will be $gen(d_1) \parallel gen(d_2) \parallel \cdots \parallel gen(d_n)$. For a keyword *key1*=$d_i \parallel \cdots \parallel d_j$, $1 \leq i \leq j \leq n$, its ciphertext is computed as FQE_k(*key1*)=$gen(d_i) \parallel \cdots \parallel gen(d_j)$.

Table 2 compares L-EncDB with Popa et al.'s CryptDB model.

Table 2
Comparison between L-EncDB and CryptDB

Model	L-EncDB	CryptDB
Fuzzy query	Yes	Partial
Data operation	Application	DB proxy and UDFs
Change DB structure	Add fields	Anonymize tables and columns

(1) CryptDB can only partially support fuzzy query. Our proposed L-EncDB is more flexible and able to support the fuzzy query.
(2) On data operation, the SQL-based operations are directly executed in DBMS. Both data encryption and decryption are executed in application in L-EncDB. However, direct SQL-based operations cannot be supported in CryptDB, where the trusted database proxy is used to intercept all the SQL queries and decrypt their (encrypted) results. To intercept SQL queries and implement encryption and decryption, CryptDB is required to build corresponding UDFs on the DBMS server.
(3) On database structure, CryptDB anonymizes each table and column name to achieve confidentiality. L-EncDB preserves most of original DB structure to reduce the cost of application codes. Hence, for existing database applications, L-EncDB is more suitable and lightweight when enterprises and organizations outsource data storage to third-party cloud providers.

In short, compared with the other database encryption solutions such as CryptDB, L-EncDB is lightweight to support SQL-based operation directly in DBMS and can be flexibly deployed in database applications.

4 New FPE Scheme for Character String

As described in section 2.1, there is no suitable FPE scheme for *varchar* data type with the restriction that the ciphertext has the same length with its corresponding plaintext. In this section, we propose a new FPE scheme for character string with arbitrary data type, which will be used in L-EncDB.

4.1 Preliminary

Throughout the rest of the paper, we let $Chars$ be a multi-byte character set, and $Chars^*$ be character strings over $Chars$ of any length. Moreover, for any set S, let $|S|$ be the number of elements in S. For a multi-byte character set $Chars$, it can be divided into subsets and each subset contains characters

of same size. Let $Chars_m$ be a subset containing all characters with storage size m. Let $cmin = min(|Chars_i|, i = 1, 2, \cdots, I)$, $cmax = max(|Chars_i|, i = 1, 2, \cdots, I)$, respectively be the number of members in the smallest and biggest subset of $Chars$.

The message space of character strings is described as $\mathcal{X}[Chars] = \{X|X \in Chars^*\}$. Given any two character strings $A, B \in \mathcal{X}[Chars]$, denote $A \parallel B$ as their concatenation. $\forall\ X \in \mathcal{X}[Chars] \Leftrightarrow X = x_1 \parallel x_2 \parallel \cdots \parallel x_i \parallel \cdots \parallel x_n, x_i \in Chars$. For any string $X \in \mathcal{X}[Chars]$, let $l(X)$ and $s(X)$ denote the length and storage size respectively. The storage size of any character $c \in Chars$ is also represented by $s(c)$.

We now give a review of the classical definition for FPE given by Morris et al. [2].

Definition 1 *A format-preserving encryption scheme is a function* $F : \mathcal{K} \times \mathcal{N} \times \mathcal{T} \times \mathcal{X} \to \mathcal{X} \times \{\perp\}$*, where* $\perp \neq \mathcal{X}$*, and nonempty sets* $\mathcal{K}$*,* $\mathcal{N}$*,* $\mathcal{T}$*,* $\mathcal{X}$ *denote the* key space*,* format space*,* tweak space *and* domain*, respectively.*

There are two kinds of character data, that is, *nvarchar* and *varchar*.

(1) If a field is defined as $nvarchar(n)$, it means that the field can store arbitrary character string with length (or character number) not more than n. The FFX method [3] is suitable for this type of domain. For example, for a plaintext string "abcd" with the length of 4, its ciphertext encrypted by FFX is "eadf", which has the same length and each character is in the same character set.
(2) If a field is defined as $varchar(n)$, it means that the field can store arbitrary character string and the storage size is not more than n.

4.2 Problem Statement

New FPE. The following explains the new FPE for message space $\mathcal{X}[Chars]$:

Definition 2 *A Character FPE is a function* $F : \mathcal{K} \times \mathcal{N}[Chars] \times \mathcal{T} \times \mathcal{X}[Chars] \to \mathcal{X}[Chars]$*, where* $\mathcal{N}[Chars]$ *is the* $formatspace$ *of character strings,* $\mathcal{K}$*,* $\mathcal{T}$*,* $\mathcal{X}[Chars]$ *are respectively the* key space*,* tweak space *and* domain*.*

Let $\mathcal{X}[Chars]$ denote message space defined by format $\mathcal{N}$. In our FPE for strings with type of *varchar*, the format space is defined by both length and storage size, that is $\mathcal{N}[Chars] = \{(l(X), s(X))|X \in \mathcal{X}[Chars]\}$.

An example. Assume that a concrete $Chars$ is defined as {'a', 'b', 'c', 'd́', 'é'}. In $Chars$, s('a')=s('b')=s('c')=1, i.e., the storage size of character 'a', 'b' and 'c' is 1 byte. But s('d́')= s('é')=2, i.e., the storage size of character 'd́' and 'é' is 2 bytes. Thus, $Chars$ is a multi-byte character set.

To better express the format of string $X = x_1 \parallel x_2 \parallel \cdots \parallel x_i \parallel \cdots \parallel x_n$, where $x_i \in Chars, 1 \leq i \leq n$. Let $\Omega(X)$ be its structure and $\Omega(X) = \{\omega_1, \cdots, \omega_I\}$, where $\omega_i = |\{x_j \in X | s(x_j) = i\}|$ and ω_i is the number of characters in string X with storage size i. For example, for string X="abéédé́", its structure is $\Omega(X)$=$\{2, 3\}$, i.e., 2 characters of storage size 1 byte, 3 Latin characters of 2 bytes, and its format is $N(X) = (5, 8)$, which means that length is 5 and storage size is 8 bytes.

4.3 Scheme Description

4.3.1 Basic Idea

For a multi-byte character set $Chars$, we establish a mapping from a subset $Chars_m$ to an integer set $Z_n = \{0, 1, \cdots, n-1\}$, where n is the character elements of $Chars_m$, that is $n = |Chars_m|$. For each character c, let $v(c)$ be its mapping value in Z_n.

For strings with type of *varchar*, FPE will preserve both length and storage size. Each character c will be represented as "$< value, radix >$", where $value$ is mapping value of c in Z_n, i.e., $value = v(c)$, and $radix$ is the element number of $Chars_{s(c)}$, i.e., $radix = n = |Chars_{s(c)}|$. For convenience, let $r(c)$ be the $radix$ of character c.

A new FPE for character string based on Feistel network is given, where Modular addition is an important component. In the new FPE scheme, the result of addition has the same $radix$ as that of left operand because the *modulo* is $radix$ of left operand. Thus, the output has the same format as the input. We call such modular addition as "multi-radix modular addition" and denote it as $\boxplus$, while $\boxminus$ as its inverse operation.

Definition 3 *For x and y, which are represented as "$< v(x), r(x) >$" and "$< v(y), r(y) >$" respectively, the multi-radix modular addition is defined as: $x \boxplus y$=$(v(x) + v(y))$ mod $r(x)$. Its inverse is defined as $x \boxminus y$=$(v(x) - v(y))$ mod $r(x)$.*

An example. Assume that we have a multi-byte character set $Chars$={'a', 'b', 'c', 'd́', 'é'} and two subsets, that is, $Chars_1$={'a', 'b', 'c'} and $Chars_2$={'d́', 'é'}. We also assume that we have two characters x='a' and y='é', which are represented as "<0, 3>" and "<1, 2>" respectively. "$x \boxplus y$" is computed by

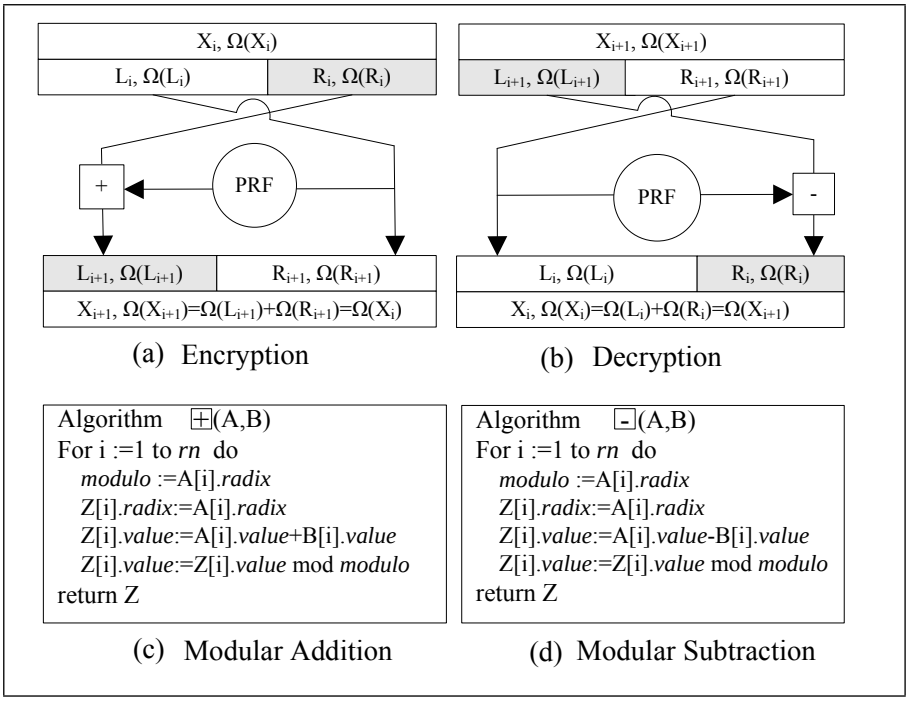


Fig. 4. FPE for character string and modular operations

'a'⊞'é'=(0+1) *mod* 3 and its result is "<1, 3>", which represented by character 'b' in $Chars_1$. The inverse operation is defined by 'b'⊟'é'=(1-1) *mod* 3 and its result is "<0, 3>", which represented by character 'a' in $Chars_1$. Similarly, the addition "$y \boxplus x$" is defined by 'é'⊞'a' =(1+0) *mod* 2 and results in "<1, 2>", which represented by character 'é' in $Chars_2$. The inverse operation is defined by 'é'⊟'a'=(1-0) *mod* 2 and results in "<1, 2>", which represented by character 'é' in $Chars_2$.

4.3.2 Description

Our new FPE can be described by three algorithms, that is, *Setup*, *Encrypt*, and *Decrypt*.

Setup: It generates the initial parameters, including the encryption key k and the number of Feistel rounds rn.

Encrypt: It takes as input the string X, key k and the round number rn. For the i-th round, its process is shown in Fig. 4. More specifically, three steps will be included. First, it divides the input X_i into a left part L_i and a right part R_i. L_i is returned as its right part, $L_{i+1} = R_i \boxplus PRF(L_i)$ as left part, in which PRF is instantiated by AES-CBC. Fig. 4 also describes the *multi-radix modular addition* algorithm, which ensures $\Omega(L_{i+1}) = \Omega(R_i)$ and $\Omega(X_{i+1}) = \Omega(X_i)$, i.e., the output of i-th round has the same format as the input.

Decrypt: It takes as input string X', key k and the round number rn. As shown in Fig. 4.(b), it divides input X_{i+1} into L_{i+1} and R_{i+1}. Then it outputs R_{i+1} as the left part, $R_i = L_{i+1} \boxminus PRF(R_{i+1})$ as the right part. The *multi-radix modular subtraction* algorithm is described in Fig. 4.(d), which ensures $\Omega(R_i) = \Omega(L_{i+1})$ and $\Omega(X_{i+1}) = \Omega(X_i)$, i.e., the output of this Feistel round has the same format as its input.

4.4 Security analysis

According to [2], the PRP security notion under chosen plaintext attack, i.e., PRP-CPA, is defined by PRP game $PRP_{\xi}^{\mathcal{A}}$ between challenger $\mathcal{C}$ and adversary $\mathcal{A}$ as follows.

Setup: $\mathcal{C}$ selects a boolean value $b \leftarrow \{0, 1\}$ in random, generates the symmetric key K for FPE scheme ξ in domain $\mathcal{X}$, and selects a uniform permutation π on $\mathcal{X}$.

Phase: $\mathcal{A}$ can adaptively ask $\mathcal{C}$ for the corresponding ciphertext for any string $X \in \mathcal{X}$. If $b = 0$, $\mathcal{C}$ responds with $\pi(X)$, otherwise with $\xi_K(X)$.

Guess: $\mathcal{A}$ outputs a predicate value b'. If $b = b'$, the security game returns 1. Otherwise, it returns 0.

Definition 4 *An FPE scheme ξ is PRP-CPA secure if any polynomial time adversary has only a negligible advantage in PRP game shown above, where the advantage is defined as*

$$\boldsymbol{Adv}_{\xi}^{PRP}(\mathcal{A}) \stackrel{def}{=} Pr[PRP_{\xi}^{\mathcal{A}} \Rightarrow 1] - \tfrac{1}{2}$$

Theorem 1 *If the underlying round function is a secure pseudo random permutation, our new FPE scheme achieves the PRP security.*

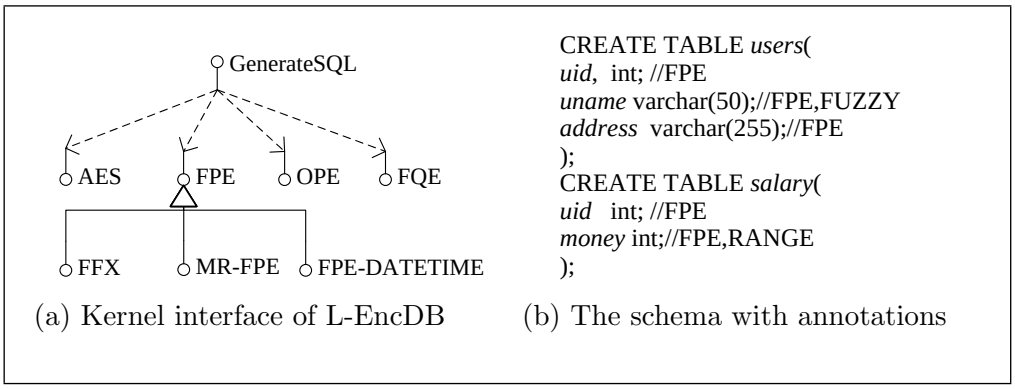
Proof 1 *Assume that there exists an adversary breaks the security of our FPE scheme, a simulator will be built to show the insecurity of PRF, that is, the simulator could break the indistinguishability of PRF from a truly random permutation P. Next, we show how to use $\mathcal{A}$ to construct a distinguisher $\mathcal{D}$. Whenever $\mathcal{A}$ queries the encryption oracle with a string X, $\mathcal{D}$ selects a random value $b = \{0, 1\}$. If $b = 0$, it computes X' with PRF, otherwise it computes X' with P. Finally, $\mathcal{D}$ returns the result X' back to $\mathcal{A}$.*

It can be seen that the view of $\mathcal{A}$ when run as a sub-routine by $\mathcal{D}$ is distributed identically to the view of $\mathcal{A}$ in game $PRP_{\xi}^{\mathcal{A}}$. Thus if the adversary can succeed in attacking the FPE, there is a distinguisher $\mathcal{D}$ having the same probability on distinguishing PRF with P in polynomial time.

5 Implementation and Evaluation

5.1 Implementation Details

The experiment for our L-EncDB system is conducted to evaluate its efficiency. We implement the system through an open kernel API of C++ DLL for L-EncDB, called `GenerateSQL`, which takes as input a plaintext SQL sentence and outputs interpreted encrypted SQL sentence. To implement FPE schemes, AES and big number in open source library *polarssl* are used. Users can improve their database security based on such a DLL with authentication and access control mechanisms.



(a) Kernel interface of L-EncDB (b) The schema with annotations

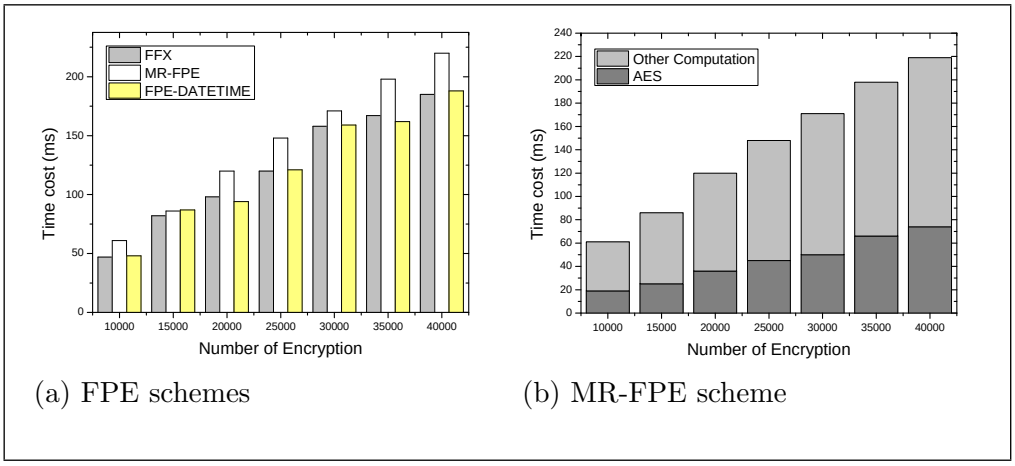
Fig. 5. Implementation details

As shown in Fig. 5.(a), `GenerateSQL` API uses techniques of FPE ("MR-FPE" is used to identify the proposed FPE scheme for character string), OPE, FQE to interpret SQL sentences. To provide correct interpretation to users, it requires that: (1) the fields of fuzzy query field and range query are named as `Field_Fuzzy` and `Field_Order` respectively; (2) DB structure is open to L-EncDB DLL. That is, L-EncDB DLL must know DB structure, and the fields should be encrypted for fuzzy query or range query. To achieve this goal, the schema with annotations shown in Fig. 5.(b) is used, in which annotation "FPE" denotes the encryption of FPE, "FUZZY" and "RANGE" denote fuzzy query and range query respectively.

The SQL sentences interpretation performs as follows. Firstly, it analyzes SQL sentence, decides operations and tables to execute. Secondly, based on operation and table structure, it decides whether encryption, fuzzy query or range query are needed. Finally, it completes the interpretation of SQL sentences with suitable encryption schemes.

Table 3
Execution time of encryption

Scheme	Encryption	Decryption
AES	0.00015 ms	0.00015 ms
OPE	9.80000 ms	0.00000 ms
FQE	FFX*n ms	0.00000 ms



(a) FPE schemes (b) MR-FPE scheme

Fig. 6. Execution time of FPE schemes

5.2 Experimental Evaluation

L-EncDB is external encryption mechanism independent from database. The adopted encryption algorithms affect DB operation performance. To evaluate its performance, two issues are addressed: 1) the performance of FPE encryption algorithms for batch data encryption, 2) the performance of OPE.

As shown in Fig. 5.(a), encryption algorithms include FFSEM, FFX, MR-FPE, FPE-DateTime, FQE and OPE. We programm for these encryption algorithms and experiment for their average execution time. For FFSEM, FFX and MR-FPE, we set Feistel rounds number as 12, use AES-CBC to construct random function. All our experiments are performed in Windows 7 operation system with the Intel(R)Core(TM)i5-3337U @ 1.80GHZ and 4GB memory.

The performance evaluation of encryption algorithms are shown in Fig. 6 and Table 3. From them, we can get the following results:

(1) the average execution time of AES is about 0.15us;
(2) as shown in Fig. 6.(a), two FPE algorithms based on Feistel network, i.e., FFX and MR-FPE, the average execution times are very close, which are around 30 times of that of AES algorithm. FPE-DateTime algorithm uses FFX (with the character set of {'0',$\cdots$,'9'}) as integer FPE to compute

the offset; hence average execution time is also close to FFX;

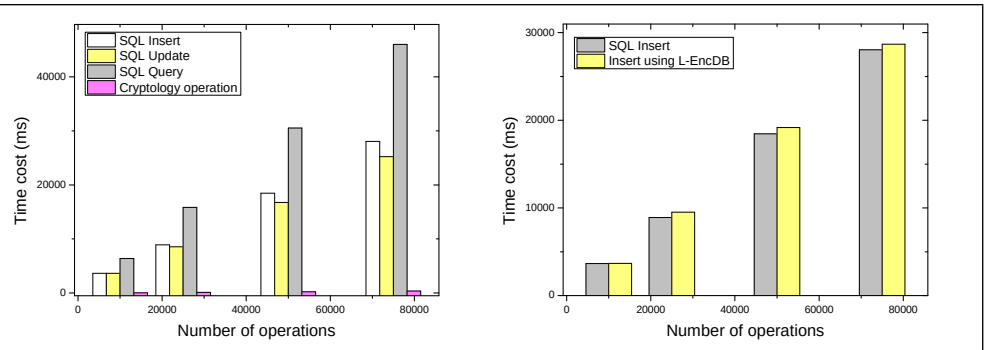
(3) as shown in Fig. 6.(b), the average execution time of MR-FPE is about 5us, which is about 30 times of that of AES. For each run, MR-FPE algorithm executes not only AES algorithm and modular arithmetic, but also coding and decoding operations. Among them, the execution time of AES algorithm is about 1.8us for 12 Feistel rounds, but other operations including modular arithmetic and coding operations cost about 3.2us.

(4) OPE's average execution time is the lowest; its decryption time is 0ms for never decrypting OPE ciphertext in L-EncDB;

(5) FQE's execution time is related to the length n of plaintext and FPE's execution time, and in practical applications, the data for fuzzy query is often less than 100, thus FQE scheme will be more efficient than OPE.

In practical applications using L-EncDB, for each DB operation, the rounds of encryption are different, which are linearly increasing with the number of fields in the operation. For a SQL insert sentence with 30 fields and no range query or fuzzy query, the execution time for each insert operation is about 0.09ms, i.e., the system can interpret 11000 SQL sentences within 1s. Thus, the system can meet needs for most of applications.

Evaluation in web applications. The L-EncDB is a lightweight mechanism and can be easily deployed in various kinds of database applications. In our evaluation, we use it to build a secure website based on the above implementation details. The webpage programming language is Java (jsp, javabean, jdbc, etc), and the web server is Resin (with version of 3.1.12). Java Native Interface (JNI) technique is used to call functions of C++ DLL. To construct the test platform, we use Java language to construct an application, in which the open source library named "HTTPClient" is used to visit the specified webpage in our website. Moreover, the Mysql is selected as the database server.



(a) SQL operations (b) Two kinds of Insert operation

Fig. 7. Comparison between SQL operations and encryption

To evaluate the performance in real web applications, we focus on the comparison of execution time between SQL operations and L-EncDB operations. We

test the basic SQL operations including insert, update and query. From Fig. 7, we can see that the execution time of SQL query operation is longer than the other SQL operations. As shown in Fig. 7.(a), the average execution time of query is about 0.6ms, but that of insert and update is only about 0.3ms. Execution time of SQL insert operation using L-EncDB is very close to that without encryption as shown in Fig. 7.(b).

6 Extension of L-EncDB to NoSQL Database Encryption

With the advent of the Internet Web2.0 site, traditional SQL-based relational databases cannot be applied and many challenges arise, including:

(1) Performance problem. To provide dynamic pages and information for user, thousands of read and write requests are produced each second in web2.0 site, and results in high concurrent load of database. However, SQL-based databases can not meet this requirement.
(2) Storage problem. In big data background, each user will produce massive dynamic information. For example, in *Friendfeed*, 250 million of user dynamic records are generated in a month. In this case, the efficiency will be unbearable when the traditional SQL-based relational databases are used to query over such records.

To solve the above challenges, NoSQL (Not Only SQL) databases have been developed and widely used in practice. *BigTable* of Goole and *Dynamo* of Amazon are successful implementation of NoSQL.

In the cloud computing, more and more enterprises also need to outsource NoSQL database to construct their business applications. Our L-EncdDB can be easily extended to NoSQL database encryption with the following properties.

(1) Independence of database. L-EncDB are always deployed in the client side, in which data encryption and decryption are all executed in the application.
(2) Queries over encrypted data. In L-EncDB, exact query can be reserved because of the deterministic encryption. The range query can also be applied for the order-preserving encryption.

After extending the implementation of SQL interpretation interface according to operation syntax of different NoSQL databases, L-EncDB can be used to encrypt the data in these databases while preserving the query syntax. For example, in the case of *MongoDB*, SQL interpretation interface can be implemented by replacing the constants in the queries:

- *Insert*, "db.users.save({name:"*String1*" age:*Age1*, country:"*String2*"})" will insert a record with three attribute name, age and country;
- *Update*, " db.users.update({ "name": "*String1*"}, { $set:{ "country":"*String3*" }})" will update "country" as "*String3*" in the record in which the value of attribute "name" is "*String1*";
- *Delete*, "db.users.remove({"name": "*String1*"})" will delete the record in which the value of attribute "name" is "*String1*";
- *Exact query*, "db.users.find({"name": "*String1*"})" will query the record in which the value of attribute "name" is "*String1*";
- *Range query*, "db.users.find({"age":{$in:[*Age1*, *Age2*]}})" will query the records in which the value of attribute"age" is in the range of *Age1* to *Age2*;
- *Fuzzy query*, *MongoDB* uses regular expression to achieve fuzzy query, for example, the SQL sentence "SELECT * FROM users where name like 'A%"' will be "db.users.find("name" :/^A/)". So, the fuzzy query expression can be interpreted as "db.users.find("name" :/^FQE_k(A)/)".

7 Conclusion and Future Work

In this paper, we proposed a novel L-EncDB mechanism, which provides a secure and privacy-preserving data utilization for outsourced database such as SQL-based encryption and query mechanism. Such a new mechanism for database does not change the data structure after encryption and can be efficiently realize data utilization such as privacy-preserving knowledge extraction, after outsourcing database into the cloud. In this new mechanism, it utilizes a core interface provided as API to interpret SQL operations, which allows to protect sensitive information in database applications. Experimental results demonstrate that the new L-EncDB is efficient and can be applied to big database for privacy-preserving applications. Finally, we also showed how to extend our L-EncDB to realize the privacy-preserving queries over encrypted NoSQL Database.

To make the L-EncDB mechanism more practical, SQL-based range query methods with better performance will be investigated in future to support comparison over ciphertexts. Especially, we will extend it to privacy-preserving knowledge extraction for outsourcing database, and further provide some practical data publishing methods suitable for our framework.

References

[1] J. Black, P. Rogaway, Ciphers with arbitrary finite domains, in: Topics in Cryptology–CT-RSA, Vol. 2271 of Lecture Notes in Computer Science, Springer, 2002, pp. 114–130.

[2] B. Morris, P. Rogaway, T. Stegers, How to encipher messages on a small domain: Deterministic encryption and the thorp shuffle, in: Advances in Cryptology–CRYPTO 2009, Vol. 5677 of Lecture Notes in Computer Science, Springer, 2009, pp. 286–302.

[3] M. Bellare, P. Rogaway, T. Spies, The ffx mode of operation for format-preserving encryption, NIST submission.

[4] V. T. Hoang, B. Morris, P. Rogaway, An enciphering scheme based on a card shuffle, in: Advances in Cryptology–Crypto 2012, Vol. 7417 of Lecture Notes in Computer Science, Springer, 2012, pp. 1–13.

[5] S. Terence, Feistel finite set encryption mode., NIST Proposed Encryption Mode.

[6] Z. Liu, C. Jia, J. Li, Format-preserving encryption for datetime, in: Intelligent Computing and Intelligent Systems, Vol. 2, Springer, 2010.

[7] M. Li, Z. Liu, J. Li, C. Jia, Format-preserving encryption for character data, Journal of Networks (2012) 1239–1244.

[8] J. Li, C. Jia, Z. Liu, Cycle-walking revisited: consistency, security, and efficiency, in: Security and Communication Networks, 2012.

[9] H. Hakan, L. Bala, L. Chen, M. Sharad, Executing sql over encrypted data in the database-service-provider model, in: Proceedings of the 2002 ACM SIGMOD international conference on Management of data, ACM, 2002, pp. 216–227.

[10] S. Evdokimov, O. Guenther, Encryption techniques for secure database outsourcing, in: Computer Security – ESORICS 2007, Vol. 4734 of Lecture Notes in Computer Science, Springer, 2007, pp. 327–342.

[11] C. Wang, Q. Wang, K. Ren, Towards secure and effective utilization over encrypted cloud data, in: the 31st International Conference on Distributed Computing Systems Workshops, IEEE, 2011, pp. 282–286.

[12] Z. Wu, G. Xu, Z. Yu, X. Yi, E. Chen, Y. Zhang, Executing sql queries over encrypted character strings in the database-as-service model, in: Knowledge-Based Systems, Vol. 35, 2012, pp. 332–348.

[13] K. Choy, W. Lee, H. C. Lau, L. Choy, A knowledge-based supplier intelligence retrieval system for outsource manufacturing, in: Knowledge-Based Systems, Vol. 18, 2005, pp. 1–17.

[14] R. A. Popa, N. Zeldovich, H. Balakrishnan, Cryptdb: protecting confidentiality with encrypted query processing, in: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, ACM, 2011, pp. 85–100.

[15] R. Agrawal, J. Kiernan, R. Srikant, Y. Xu, Order preserving encryption for numeric data, in: Proceedings of the 2004 ACM SIGMOD international conference on Management of data, ACM, 2004, pp. 563–574.

[16] A. Boldyreva, N. Chenette, Y. Lee, A. O'Neill, Order-preserving symmetric encryption, in: Advances in Cryptology–EUROCRYPT 2009, Vol. 5479 of Lecture Notes in Computer Science, Springer, 2009, pp. 224–241.

[17] R. A. Popa, F. H. Li, N. Zeldovich, An ideal-security protocol for order-preserving encoding., in: Proc. of the 34th IEEE Symposium on Security and Privac, 2013.

[18] I. Yakut, H. Polat, Estimating nbc-based recommendations on arbitrarily partitioned data with privacy, in: Knowledge-Based Systems, Vol. 36, 2012, p. 353C362.

[19] D. X. Song, D. Wagner, A. Perrig, Practical techniques for searches on encrypted data, in: Proceedings of the 21st IEEE Symposium on Security and Privacy, IEEE, 2000, pp. 44–55.

[20] D. Boneh, B. Waters, Conjunctive, subset, and range queries on encrypted data, in: 4th Theory of Cryptography Conference, Vol. 4392 of Lecture Notes in Computer Science, Springer, 2007, pp. 535–554.

[21] K. W. Lin, Y.-C. Lo, Efficient algorithms for frequent pattern mining in many-task computing environments, in: Knowledge-Based Systems, Vol. 49, 2013, pp. 10–21.

[22] M. I. M. A. S. Umer Khalid, Abdul Ghafoor, Cloud based secure and privacy enhanced authentication & authorization protocol, in: Knowledge-Based Systems, Vol. 22, 2013, pp. 680–688.

[23] A. S. Sattar, J. Li, X. Ding, J. Liu, M. Vincent, A general framework for privacy preserving data publishing, in: Knowledge-Based Systems, Vol. 54, 2013, pp. 276–287.

[24] Z. Liu, H. Ma, J. Li, C. Jia, Secure storage and fuzzy query over encrypted databases, in: Network and System Security, 2013, pp. 439–450.

[25] L. H. Nguyen, A. W. Roscoe, Short-output universal hash functions and their use in fast and secure data authentication, in: Fast Software Encryption, Vol. 7549 of Lecture Notes in Computer Science, Springer, 2012, pp. 326–345.