International Journal of Computer Techniques – Volume 4 Issue 3, May – June 2017

RESEARCH ARTICLE OPEN ACCESS

Executing SQL Queries over Encrypted Data: A Survey

Taipeng Zhu*
*(Department of Computer Science, Jinan University, Guangzhou-510632, China)

--************************----------------------------------

Abstract:
This paper presents an overview of recent secure database systems, which are capable of executing SQL queries over encrypted data. Data security issues in a cloud database system which database-as-a-service model is one of most import concerns, a simple and effective way to guarantee confidentiality is to encrypt data before storing. Meanwhile, the ability to operate on encrypted data in the cloud needs to be guaranteed. In those systems, different encryption schemes are promoted and the capabilities of executing SQL queries over encrypted data have software and hardware implementation.

Keywords **— SQL Queries, Encrypted Data, Secure Database Systems, Encryption Schemes.**

--************************----------------------------------

I. INTRODUCTION

Encrypting data to store at the server where its data is not entirely controllable by the owner of data is an available solution to defend sensitive information exposure. While it is not arbitrary to execute SQL queries over encrypted data, for ciphertexts may lose its original features, for example, length, format; their plaintext's comparison, arithmetic operations may no longer support or partially support as well. For realizing SQL queries over encrypted data with decryption in advance, there have been various solutions in recent years, in which software-based and hardware-based cryptographic are constructed.

In general, there are much ways to build secure database systems, for example, encryption at rest, secure server, fully homomorphic encryption. All representative researches in recent years are listed in Table 1.

Table 1. List of researches on executing SQL queries over encrypted data

Time(Year)	System	Paper
2002	[1]	Executing SQL over encrypted data in the database-service-provider model
2011	CryptDB[2]	CryptDB: protecting confidentiality with encrypted query processing
2011	TrustedDB[4]	Trusteddb: A trusted hardware-based database with privacy and data confidentiality
2013	MONOMI[3]	Processing analytical queries over encrypted data
2013	Cipherbase[6]	Secure database-as-a-service with cipherbase
2014	SDB[8]	Secure query processing with data interoperability in a cloud database environment

The earlier research of executing SQL over encrypted data [1] proposes that SQL is executed over the encrypted data by rewriting a relational algebraic similarly to be executed over the unencrypted data. Decrypting data and complex queries are executed on the client. But there are some limitations on searching and querying on encrypted data, for example, certain queries with joining and sorting are not supported or highly inefficient. Moreover, in order to solve these problems, DBMS is required to be modified or some of queries are performed on the client.

In contrast, CryptDB proposes a database proxy layer to encrypt and decrypt data so that the internal structure of DBMS is not modified, namely CryptDB uses native DBMS [2]. It proposes three key ideas, the first is to execute SQL query on encrypted data, the second is to adopt adjustable encryption strategies for different queries, and the third is to chain encrypted keys to user passwords.

More detail in the second idea, CryptDB encrypts data by an 'onion' in which different query is based on different encrypting algorithm. SQL queries such as equality selection, equality join, order, range join, text searching, SUM of integer data and etc. can be performed.

Stephen et al. points out that CryptDB only supports queries including computation and hardly supports analytical query [3]. Therefore, MONOMI is established based on the design of CryptDB, which can process the complex analytical query and large data set. As executing analytical load to encrypted data on the server is very difficult, MONOMI proposes an executing method of splitting client/server. Splitting executing allows MONOMI to execute part of queries to encrypted data on untrusted server and performed through the scheme the same with CryptDB, while the rest of queries are executed after decrypting data on the trusted client.

Different from CryptDB and MONOMI in which different encrypting algorithms are used to different queries, A secure query processing system(SDB) in [8] is realized through a group of security operators (e.g., $\times$, $\pm$, π, $\oplus$, $\bowtie_S$ with data interoperation which can efficiently support a quantity of complex SQL query including all TPC-H benchmark queries on the server. In view of the limitations of fully and partially homomorphic encryption, SDB does not adopt homomorphic encryption algorithm adopted in CryptDB and MONIMI instead of using a secret-sharing scheme in SMC model [24] and the solution in ShareMind [25].

Hacigümüs thinks that hardware encryption is better than software encryption [26]. As a result, TrustedDB [4] and Cipherbase [6] take a hardware approach to provide data security. A security coprocessor unit (SCPU) hardware is introduced in TrustedDB. Sumeet Bajaj deems any encryption methods based on software have an inherent defect that expression is limited, so it is best to guarantee the privacy of data through trusted hardware. Hence, TrustedDB provides more secure data protection by SCPU hardware, and supported query types are not limited.

Cipherbase adopts from TrustDB the idea of combining trusted hardware and commodity servers in a single box, but has a more sophisticated and fine-grained hardware-software co-design. Cipherbase combines encryption at rest, secure server and fully homomorphic encryption to archive orthogonal security, and it is implemented by using FPGAs [27]. It has the completeness, user-defined confidentiality and efficient properties, which is a full-fledged SQL database system that supports the full generality of a database system while providing high data confidentiality. What's more, it allows organizations to develop their applications and set their data security goals relatively independently of any performance, scalability or cost considerations. The cost and the performance constraint of TrustedDB and Cipherbase is higher than methods of software.

II. SCHEMES OF EXECUTING SQL QUERIES OVER ENCRYPTED DATA

There are two things to handle when executing SQL queries over encrypted data, one thing is query rewriting, where we need rewrite an original query over unencrypted relations into a corresponding query over encrypted relations to run on the server; the other is post-processing to results of the server query, in which the decryption and additional processes are necessary. The execution of query rewriting is associated with encryption schemes, and the encryption schemes determine the security properties of the whole system.

A. Software-based Design

Paper in [1] proposed a solution in the database-service-as-provider model and its architecture is in Fig 1, where the encrypted database is augmented with additional information, and it developed an algebraic framework for query rewriting over encrypted representation. In detail, for each relation $R(A_1, A_2, \ldots, A_n)$, the encrypted relation stored on the server is as follow:

$$R^S(etuple, A_1^S, A_2^S, \ldots, A_n^S)$$

Where the attribute *etuple* stores an encrypted string that corresponds to a tuple in relation R and is encrypted by block cipher such as AES, Blowfish, DES. Each attribute A_i^S corresponds the index for the attribute A_i that will be used for query processing at the server.

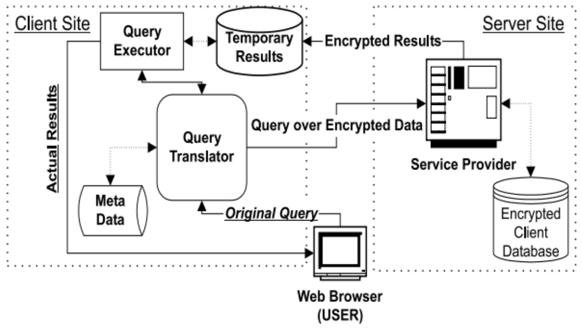


Fig 1. The service-provider architecture.

It proposes some partition function to map the domain of values of attribute into partitions, identification functions to assign an identifier to each partition of attribute and a mapping function that maps a value in the domain of attribute to the identifier of the partition, which all are auxiliary to store encrypted data. More concretely about the mapping functions, there are two types of mapping functions, the one is order preserving and the other is random so the allowed operations in mapping conditions include {=, <, >, ⩽, ⩾}, the detail conditions are specified in Table 2. It is only necessary to create an index for attributes involve in search and join predicates.

Table 2. Supported mapping conditions.

Conditions
Attribute = Value
Attribute < Value
Attribute > Value
Attribute1 = Attribute2
Attribute1 < Attribute2
Condition1 ∨ Condition2, Condition1 ∧ Conditions2

In CryptDB [2], a key insight that make it practical is that SQL uses a well-defined set of operators, each of them supports efficiently over encrypted data. CryptDB is applied a SQL-aware encryption strategy to execute SQL queries over encrypted data and provides adjustable query-based encryption to guarantee to reveal the least information to meet the encryption algorithms. It provides to prevent a curious DBA or other external attacker from learning private data, guards against the application server, proxy, and DBMS server infrastructures with being compromised arbitrarily, and its architecture is shown in **Fig. 2.**

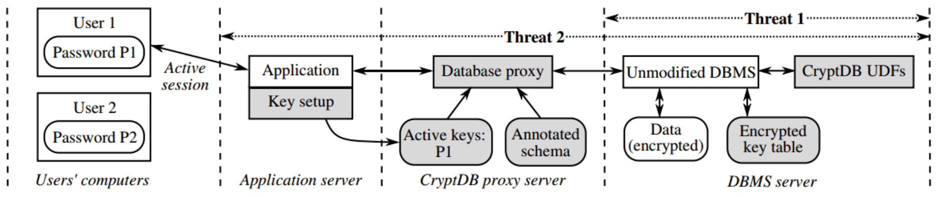


Fig. 2. CryptDB's architecture.

CryptDB enables the DBMS server to execute SQL queries on encrypted data almost as if it were executing the same queries on plaintext data, and is of high transparency to clients. Processing a query in CryptDB involves four steps:

1) The application issues a query, which the proxy intercepts and rewrites: it anonymizes each table and column name, and, using the master key MK, encrypts each constant in the query with an encryption scheme best suited for the desired operation.

2) The proxy checks if the DBMS server should be given keys to adjust encryption layers before executing the query, and if so, issues an UPDATE query at the DBMS server that invokes a UDF to adjust the encryption layer of the appropriate columns.

3) The proxy forwards the encrypted query to the DBMS server, which executes it using standard SQL (occasionally invoking UDFs for aggregation or keyword search).

4) The DBMS server returns the (encrypted) query result, which the proxy decrypts and returns to the application.

To implement the adjustable encryption, CryptDB uses onions of encryption. Onions are a novel way to compactly store multiple ciphertexts within each other in the database and avoid expensive re-encryptions. Each of layers in onions is applied different encryption algorithm to cater to the required operations for query. Onion encryption layers are show in **Fig. 3**.

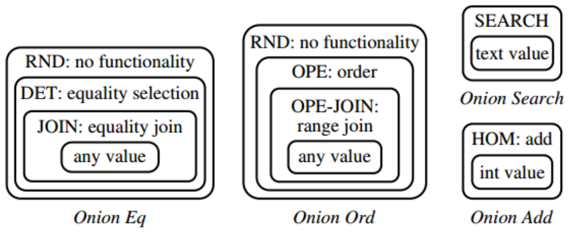
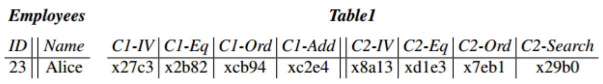


Fig. 3. Onion encryption layers and the classes of computation they allow.

CryptDB has been implemented on both MySQL and Postgres. Example of relation and corresponding encrypted relation is as shown in Fig. 4. Each attribute is applied different encryption algorithm to meet one operation that the data type of its original attribute supports.



Employees

ID	*Name*
23	Alice

Table1

C1-IV	*C1-Eq*	*C1-Ord*	*C1-Add*	*C2-IV*	*C2-Eq*	*C2-Ord*	*C2-Search*
x27c3	x2b82	xcb94	xc2e4	x8a13	xd1e3	x7eb1	x29b0

Fig. 4. Example of relation and corresponding encrypted relation.

MONOMI [3] introduces split client/server query execution based on CryptDB to support arbitrarily complex queries over encrypted data. In addition, a number of techniques that improve performance for certain kinds of queries, including pre-row precomputation, spec-efficient encryption, grouped homomorphic addition, and pre-filtering, are introduced. Its architecture is shown in Fig. 5, where MONIMI prototype consists of three major components, designer, ODBC library, and encryption database.

MONOMI's *designer* runs on a trusted client machine and determines an efficient physical design for an untrusted server during system setup. Then application issue unmodified SQL queries using the MONOMI ODBC library, which is the component that has access to the decryption keys during normal operation and uses the *planner* to determine the best split client/server execution plan for the application's query. Finally, given an execution plan, the library issues one or more queries to the encrypted database.

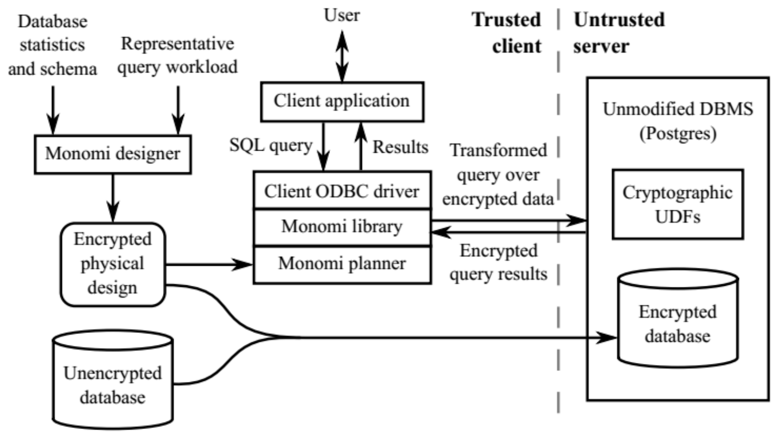


Fig. 5. The architecture of MONOMI prototype.

As MONOMI's design builds on CryptDB, it adopts the similar encryption schemes and inherits similar security properties. Given the possibility that some encryption schemes may leak more information than necessary, MONOMI never stores plaintext data on the server, and allows the administrator to further restrict the encryption schemes used for especially sensitive columns.

SDB [8] shares similar flavors with SMC/SharedMind in that sensitive data is decomposed into shares to process a secure query. Unlike ShareMind where multiparties are involved, SDB requires only two parities – the trusted data owner (DO) and one untrusted service provider (SP), and it provides a set of efficient operators with data interoperability which different operators share the same encryption and thus an operator can be applied on the results of another operator. SDB can be integrated seamlessly with existing DBMSs and utilize many of their functionality, and the architecture of SDB is shown in Fig. 6. It uses MySQL to store the encrypted data.

SDB encrypts sensitive data using a secret-sharing method and non-sensitive data are stored as plaintexts. The various operators can be applied to both encrypted data and plain data, or a mixture of them. The supported secure primitive operators are depicted in Table 3. Since its encryption scheme is based on modular arithmetic, its operators are applicable only to data values of integer domains.

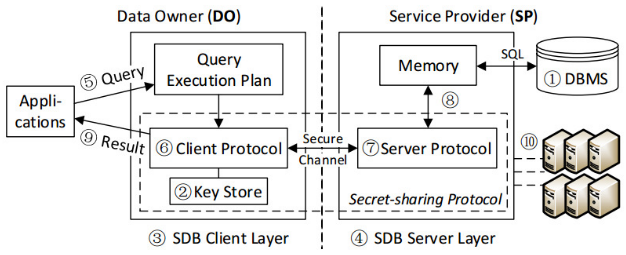


Fig. 6. SDB's architecture.

Table 3. List of secure primitive operators.

Operator	Expression	Description
$\times$	$A\times B$	Vector dot product of two columns of the same table
$\pm$	$A\pm B$	Vector addition/subtraction of two columns of the same table
=	A=B	Equality comparison on two columns of the same table and output a binary column of '0' and '1'
$>$	A>B	Ordering comparison on two columns of the same table and output a binary column of '0' and '1'
π	$\pi_S(R)$	Project table R on attributes specified in an attribute set S
$\oplus$	$R_1\oplus R_2$	Cartesian product of two relations
$\bowtie_S$	$R_1\bowtie_S R_2$	Equijoin of two relations on a set of join keys S
$\bowtie$	$R_1\bowtie R_2$	Natural join between two relations
GroupBy	GroupBy(R, A)	Group rows in relation R by column A's values
Sum/Avg	Sum/Avg(R, A)	Sum or average the value of column A in relation R
Count	Count(R)	Count the number of rows in a relation

Each sensitive data item is split into two shares, one kept at the DO and another at the SP. From the description about the encryption procedure in [8], it can be demonstrated that the encrypted relation stored on the server corresponding relation *R(A)* is as follow:

$$R^S(row\text{-}id,\ A^S)$$

Where row ids are encrypted using an additive homomorphic encryption, e.g., SIES [33], and their corresponding plaintext are used in encrypting column *A*'s values.

B. Hardware-based Design

TrustedDB [4] is an outsourced database prototype that allows clients to execute SQL queries with privacy and under regulatory compliance constraints without having to trust the server provider. The cryptographic constructs are based on trapdoor function in TrustedDB, and currently viable trapdoors are based on modular exponentiation in large fields and viable homomorphisms involve a trapdoor for computing the ciphertexts. TrustedDB extends SQL syntax by deploying keywords to designate whether one attribute is sensitive or not. In TrustedDB, all decryptions are performed within the secure confinements of the SCPU. The architecture of TrustedDB is shown in Fig. 7.

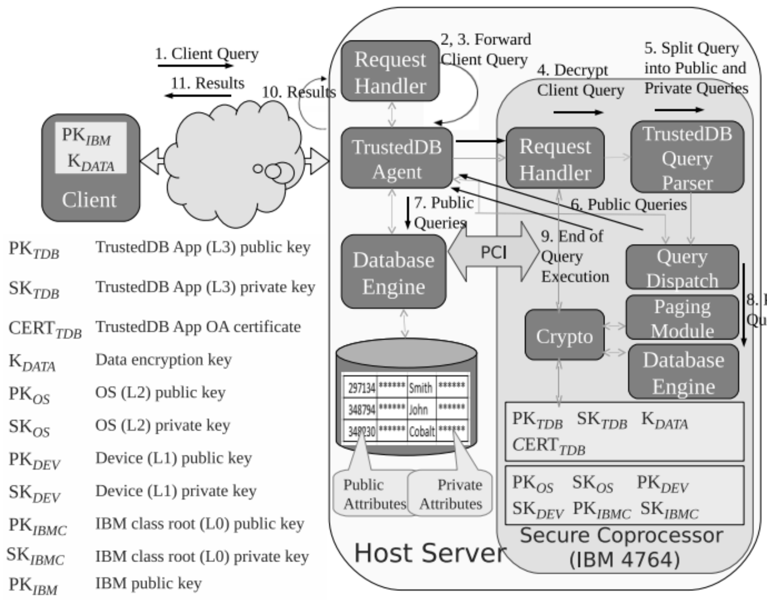

1. Client Query
11. Results
10. Results
2, 3. Forward Client Query
4. Decrypt Client Query
5. Split Query into Public and Private Queries
PK_IBM
K_DATA
Client
Request Handler
TrustedDB Agent
Request Handler
TrustedDB Query Parser
7. Public Queries
6. Public Queries
Database Engine
PCI
9. End of Query Execution
Query Dispatch
Paging Module
8. Private Queries
Crypto
Database Engine
PK_TDB SK_TDB K_DATA CERT_TDB
PK_OS SK_OS PK_DEV SK_DEV PK_IBMC SK_IBMC
Secure Coprocessor (IBM 4764)
Host Server
Public Attributes
Private Attributes
297134 ****** Smith ******
348794 ****** John ******
349030 ****** Cobalt ******
PK_TDB TrustedDB App (L3) public key
SK_TDB TrustedDB App (L3) private key
CERT_TDB TrustedDB App OA certificate
K_DATA Data encryption key
PK_OS OS (L2) public key
SK_OS OS (L2) private key
PK_DEV Device (L1) public key
SK_DEV Device (L1) private key
PK_IBMC IBM class root (L0) public key
SK_IBMC IBM class root (L0) private key
PK_IBM IBM public key

Fig. 7. TrustedDB architecture.

TrustedDB is an approach that combines the secure servers and encryption at rest approaches, and it runs a lightweight SQLite database on the SCP and a more feature-rich MySQL database on the commodity server.

Cipherbase [6] has a novel architecture that tightly integrates custom-designed trusted hardware for performing operations on encrypted data securely, and is an extension of Microsoft SQL Server. It features a novel hardware/software co-design that leverages customized hardware in which the keys can be stored in a tamper-proof way in order to enable computations on encrypted data in the cloud in a secure fashion. The Cipherbase system provides the same features as traditional database systems (e.g., support for full SQL, transaction, and recory). Fig. 8 gives an overview of the Cipherbase system. The SQL statements are processed in the server just like in any other database system. The TM (for Trusted Module) is used as submodule for core operations over encrypted data, and is placed inside the UM.

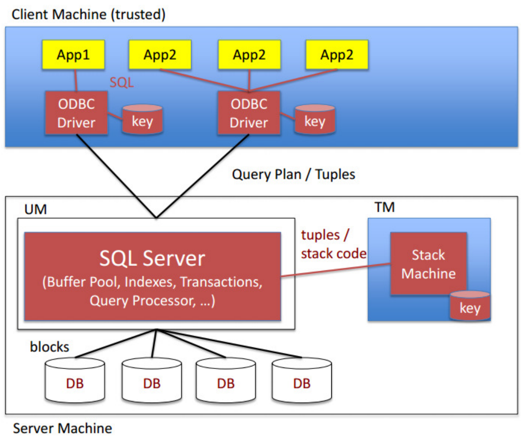


Fig. 8. Cipherbase's architecture.

Cipherbase grarantees orthogonal security, and it is implemented by using field programmable gate arrays (FPGAs [27]). Moreover, its key idea is to simulate fully homomorphic encryption on top of non-homomorphic encryption schemes by integrating trusted hardware and it uses trusted hardware to implement a core set of basic primitives to operate on encrypted data.

III. DATE TYPES, OPERATORS AND ENCRYPTION SCHEMES

When it comes to standard SQL data types, there are six basic data types, integer, floating-point, bool, character, binary, datetime. All the data types may have various keywords in a specified database system. Moreover, each data type has its basic operators to handle, and the detailed basic operators of each data type are specified in Table 4.

Table 4. Data types and their basic operators.

Data Type	Basic Operators
INTEGER/FLOATING-POINT	Equality (=, <> and etc.)
	Comparison (>, >=, <, <= and etc.)
	Arithmetic (+, -, ×, /, %)
BOOL	Equality (=, <> and etc.)
CHARACTER	Equality (=, <> and etc.)
	Comparison (>, >=, <, <= and etc.)
	Word Search (LIKE, NOT LIKE and etc.)
BINARY	Equality (=, <> and etc.)
	Comparison (>, >=, <, <= and etc.)
DATETIME	Equality (=, <> and etc.)
	Component (year, month, day and etc.)

Although no one encryption algorithm makes ciphertext retain all the properties of the plaintext, partial properties may be holding over ciphertext, for instance, block ciphers are deterministic encryption algorithms that support query with equality comparison, partially homomorphic encryption algorithms support addition or multiplication arithmetic. Consequently, both CryptDB and MONOMI employ different encryption functions to support different operators. As well as the Cipherbase system if the security model allows to do [7].

There are a number of solutions to support operations on encrypted data among the SQL queries. The CryptDB and MONOMI prototype illustrate that they support integer type, character type and binary type. Among them, Blowfish and AES are applied to encrypt data of all considered data type to cater to query with equality check, they use Paillier and EIGamal to encrypt the integer data for summation and production. The encryption schemes [34] in CryptDB are shown in Fig. 9.

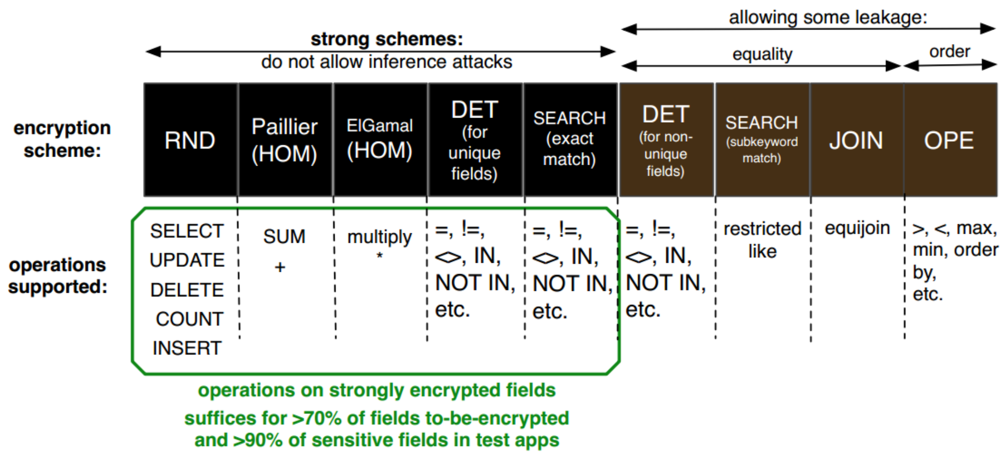

allowing some leakage:
strong schemes:
do not allow inference attacks
equality
order
encryption scheme:
RND
Paillier (HOM)
ElGamal (HOM)
DET (for unique fields)
SEARCH (exact match)
DET (for non-unique fields)
SEARCH (subkeyword match)
JOIN
OPE
operations supported:
SELECT UPDATE DELETE COUNT INSERT
SUM +
multiply *
=, !=, <>, IN, NOT IN, etc.
=, !=, <>, IN, NOT IN, etc.
=, !=, <>, IN, NOT IN, etc.
restricted like
equijoin
>, <, max, min, order by, etc.
operations on strongly encrypted fields
suffices for >70% of fields to-be-encrypted and >90% of sensitive fields in test apps

Fig. 9. Encryption Schemes in CryptDB.

A. *Order-Preserving Encryption*

To support comparison check on encrypted data, the original are adopted the order-preserving encryption algorithm. It means that if $x < y$, then $OPE_K(x) < OPE_K(y)$, for any secret key K. The other SQL query clauses or aggregate functions, such as ORDER BY, MIN, MAX, SORT, etc. can be performed as well. An order preserving encryption scheme (OPES) [9] is introduced for numeric data to allow comparison operations to be directly applied on encrypted data, without decrypting then operands in advance. The basic idea of OPES is to take as input a user-provided target distribution and transform the plaintext values in such a way that the transformation preserves the order while the transformed values follow the target distribution. This scheme does not reveal any information about the original values apart from the order. OPES works in three states:

1) Model: The input and target distributions are modeled as piece-wise linear splines.

2) Flatten: The plaintext database P is transformed into a "flat" database F such that the values in F are uniformly distributed.

3) Transform: The flat database F is transformed into the cipher database C such that the values in C are distributed according to the target distribution.

Note that

$$p_i < p_j \Rightarrow f_i < f_j \Rightarrow c_i < c_j:$$

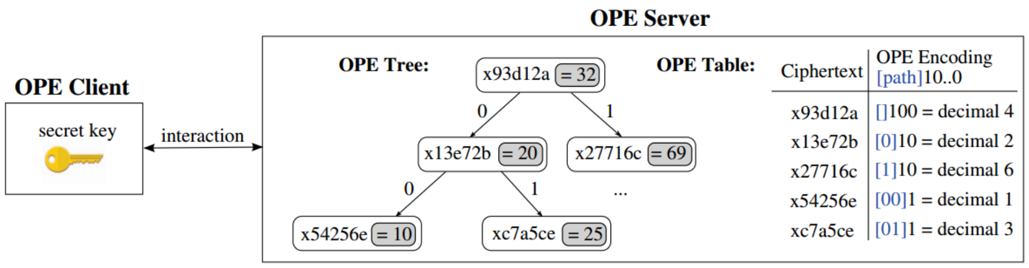
The OPES encryption scheme can be directly applied to 32-bit integers and positive 32-bit single precision floating points. Negative floating point and 64-bit double precision floating point values can be encrypted in a similar scheme. The plaintext values are 32-bit integers, and both flattened and final ciphertext values are 64-bit long.

The order-preserving symmetric encryption (OPE) in [10] is the first provably secure such scheme, where a security notion in the spirit of pseudorandom functions (PRFs) and related primitives asking that an OPE scheme look "as-random-as-possible" subject to the order-preserving constraint is proposed instead of indistinguishability against chosen-plaintext attack (IND-CPA). It proposed LayzSample and LazySampleInv algorithm in aid of encryption and decryption procedures. The encryption and decryption equation of OPE are as follows:

$$c = Enc_K^{HG}(D, R, m), \quad m = Dec_K^{HG}(D, R, c)$$

where c and m respectively represent the ciphertext and plaintext, a random $K \in Keys$ in key-space, the plaxintext are ciphertext spaces are sets of consecutive integers D, R, Algorithm Enc^{HG}, Dec^{HG} are the same as LazySample, LazySampleInv respectively. OPE not only allows efficient range queries, but also allows indexing and query processing to be done exactly and as efficiently as for unencrypted data. OPE is capable of being applied to integer, character, binary data type.

In CryptDB system, an encryption scheme that computes order queries is constructed and called mutable order-preserving encoding, or mOPE, which achieves ideal IND-OCPA security [11]. The model of OPE consists of a trusted OPE client and an untrusted OPE server. The encoding scheme is as illustrated in Fig. 10. Each node in the OPE Tree contains a DET ciphertext, and Child pointers are labeled with 0 or 1 to indicate the path encoding.



Ciphertext	OPE Encoding [path]10..0
x93d12a	[]100 = decimal 4
x13e72b	[0]10 = decimal 2
x27716c	[1]10 = decimal 6
x54256e	[00]1 = decimal 1
xc7a5ce	[01]1 = decimal 3

Fig. 10. Overview of mOPE's data structures.

[11] points out that any stateful OPE scheme that is IND-OCPA secure has ciphertext size exponential in the plaintext size. On average, mOPE stores 40 bytes per encrypted value, when encrypting 64-bit values. It also presents an extension of mOPE, called stOPE, that archives this stronger definition.

B. Word Search

There have been a log of researches on searing on encrypted data. In general, there are two types of approaches. One possibility is to build up an index that. An alternative is to perform a sequential scan without an index. The advantage of the former is that it may be faster than the sequential scan when the documents are large, while storing and updating the index may be of substantial overhead. So that the approach of using an index is more suitable for mostly-read-only data. Some of them encryption schemes only allows performing full-word keyword searches, some others may support fuzzy keyword search. The typically researches in recent years are listed in Table 5.

Table 5. Schemes of searching on encrypted data.

Time(Year)	Scheme	Paper
2000	Sequential Scan[12]	Practical techniques for searches on encrypted data
2004	Conjunctive Keyword Search[13]	Secure conjunctive keyword search over encrypted data
2010	Fuzzy Keyword Search[14]	Fuzzy keyword search over encrypted data in cloud computing
2010	Ranked Keyword Search[15]	Secure ranked keyword search over encrypted cloud data
2014	Multi-Keyword Ranked Search[16]	Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud
2016	Personalized Search[17]	Enabling personalized search over encrypted outsourced data with efficiency improvement

Queries with LIKE, NOT LIKE and etc., which need fuzzy match, can be handled by word search schemes. Nevertheless, when the data are encrypted, the database built-in function of LIKE may be not capable of adopting directly, so a user-defined function (UDF) should be created to substitute to perform LIKE operation.

C. Format-Preserving Encryption

Format-preserving encryption (FPE) encrypts a plaintext of some specified format into a ciphertext of identical format. The data type of DATETIME in database is exact identical format, like as "YYYY-MM-DD HH::MM::SS". If values of DATETIME are able to be encrypted as their original format so that their ciphertexts can be stored as the same DATETIME data type. The build-in functions, such as *year*, *month*, *day*, can be directly utilized in database in that situation. For example, a column "*prod_date*" is anonymized as "*c_prod_date*" in database, and a plantext "*2017-05-09:23:12:08*" is encrypted as "*2008-10-17:12:13:50*" by FPE. When the client submit a query, where an atom "*year(prod_date) = 2017*" is involved in where clauses, the query rewriter module would rewrite the atom as "*year(c_prod_date) = 2008*" so that we do not need to create a UDF to perform *year* function if the encryption algorithm supports the components to encrypt with the same encryption schemes. It means that FPE is suitable for encrypting "*2017*" as "*2008*" as well.

There have been various format-preserving encryptions to meet different formats [18-21], but for DATETIM type format-preserving encryption, the existing solutions do not much. [22] proposed a format-preserving encryption to encrypt DATETIME field in database. DATETIME type has format constraint and natural constraint, so a key problem is its domain. The FPE scheme for DATETIME is based on "rank-then-cipher" mode and further a new approach named "reference-based offset encryption" is proposed to resolve the FPE problem on DATETIME domain.

D. Encryption Schemes on Floating-Point

As traditional homomorphic encryption schemes do not support floating-point data type and do some arithmetic operations over ciphertexts, CryptDB and MONOMI which all are applied homomorphic encryption schemes do not take float-point into account. To allows the floating-point arithmetics of ciphertexts, thus computing encryptions of most significant bits of m_1+m_2 and m_1m_2, given encryptions of floating-point number m_1 and m_2, a floating-point homomorphic encryption (FPHE) scheme is constructed [28], which is based on BGV scheme [29].

[30] constructs a first fully homomorphic encryption scheme FHE4FX that can homomorphically compute addition and/or multiplication of encryption fixed point numbers

without knowing the secret key, which is based on FV scheme[31,32].

IV. SQL QUERY STATEMENTS

All kinds of solutions are qualified to perform all basic SQL queries, so that make executing SQL queries over encrypted data practical. A standard query statement mainly consists of SELECT clause, FROM clause and WHERE clause, and its form is as follows:

SELECT *| *column_name*| *expr*| *aggregate_func* (*column*), …
FROM *table_name*| , *table_name1*| [INNER| OUTER| FULL| LEFT| RIGHT] JOIN *table_name1* ON *join_condition*
[WHERE *where_clauses*]
[GROUP BY *column_name*| , *column_name1*,… [HAVING *having_conditions*]]
[ORDER BY *column_name* [ASC| DESC] | , *column_name1* [ASC| DESC], …]

Where SELECT clause and WHERE clause are diverse in forms. When it comes to supported SQL queries, the main considerations are whether the SELECT clause and WHERE clause are supported or not. WHERE clauses are all arithmetic atoms that returns a boolean value.

Even though there are a number of research on executing SQL queries over encrypted data, they have the query statement limitations more or less. It does not support both computations and comparison on the same column, such as WHERE *salary>age*2+10* in CryptDB [35], but it can be handled by splitting client/server query execution which is proposed in MONOMI. CryptDB can handle four out of 22 TPC-H queries, while MONOMI executes 19 out of 22 TPC-H queries by splitting client/server execution. As for A query with, such as *SUM(ps_supplycost * ps_availqty)> value*, which also involves addition, comparison and multiplication, MONOMI' encryption schemes cannot support directly over encrypted data.

Encryption scheme adopted by SDB can support complex operations executed on the server, and all queries in the TPC-H benchmark are natively supported. Whereas it has some limitations, for example, SDB does not natively support operators which their output results are non-integer values, e.g., square root (√) [7]. In addition, there are some limitations in TrustedDB as well, for example, query parser in TrustedDB cannot parse multi-level nested sub-queries and views defined by user [4].

V. CONCLUSIONS

This paper reviews the solutions that executing SQL queries over encrypted data. All of them support the basic data types and basic query statements with different encryption schemes and implementation mechanism. The software-based implementations are less costly but support type and functionality are more limited. In contrast, the hardware implementations have the cost overhead and performance limitations. Most of the secure model work as trusted proxy to provide secure channel.

REFERENCES

[1] Hacigümüş H, Iyer B, Li C, et al. Executing SQL over encrypted data in the database-service-provider model[C]//Proceedings of the 2002 ACM SIGMOD international conference on Management of data. ACM, 2002: 216-227.

[2] Popa R A, Redfield C, Zeldovich N, et al. CryptDB: protecting confidentiality with encrypted query processing[C]//Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. ACM, 2011: 85-100.

[3] Tu S, Kaashoek M F, Madden S, et al. Processing analytical queries over encrypted data[C]//Proceedings of the VLDB Endowment. VLDB Endowment, 2013, 6(5): 289-300.

[4] Bajaj S, Sion R. Trusteddb: A trusted hardware-based database with privacy and data confidentiality[J]. IEEE Transactions on Knowledge and Data Engineering, 2014, 26(3): 752-765.

[5] Bajaj S, Sion R. Trusteddb: A trusted hardware based outsourced database engine[C]//VLDB, DEMO. 2011.

[6] Arasu A, Blanas S, Eguro K, et al. Secure database-as-a-service with cipherbase[C]//Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. ACM, 2013: 1033-1036.

[7] Arasu A, Blanas S, Eguro K, et al. Orthogonal Security with Cipherbase[C]//CIDR. 2013.

[8] Wong W K, Kao B, Cheung D W L, et al. Secure query processing with data interoperability in a cloud database environment[C]//Proceedings of the 2014 ACM SIGMOD international conference on Management of data. ACM, 2014: 1395-1406.

[9] Agrawal R, Kiernan J, Srikant R, et al. Order preserving encryption for numeric data[C]//Proceedings of the 2004 ACM SIGMOD international conference on Management of data. ACM, 2004: 563-574.

[10] Boldyreva A, Chenette N, Lee Y, et al. Order-preserving symmetric encryption[C]//Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer Berlin Heidelberg, 2009: 224-241.

[11] Popa R A, Li F H, Zeldovich N. An ideal-security protocol for order-preserving encoding[C]//Security and Privacy (SP), 2013 IEEE Symposium on. IEEE, 2013: 463-477.

[12] Song D X, Wagner D, Perrig A. Practical techniques for searches on encrypted data[C]//Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on. IEEE, 2000: 44-55.

[13] Golle P, Staddon J, Waters B. Secure conjunctive keyword search over encrypted data[C]//International Conference on Applied Cryptography and Network Security. Springer Berlin Heidelberg, 2004: 31-45.

[14] Li J, Wang Q, Wang C, et al. Fuzzy keyword search over encrypted data in cloud computing[C]//INFOCOM, 2010 Proceedings IEEE. IEEE, 2010: 1-5.

[15] Wang C, Cao N, Li J, et al. Secure ranked keyword search over encrypted cloud data[C]//Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on. IEEE, 2010: 253-262.

[16] Wang B, Yu S, Lou W, et al. Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud[C]//INFOCOM, 2014 Proceedings IEEE. IEEE, 2014: 2112-2120.

[17] Fu Z, Ren K, Shu J, et al. Enabling personalized search over encrypted outsourced data with efficiency improvement[J]. IEEE transactions on parallel and distributed systems, 2016, 27(9): 2546-2559.

[18] Mattsson U, Blomkvist K. Data type preserving encryption: U.S. Patent 7,418,098[P]. 2008-8-26.

[19] Brier E, Peyrin T, Stern J. BPS: a format-preserving encryption proposal[J]. Submission to NIST, available from their website, 2010.

[20] Pauker M J, Spies T, Martin L W. Data processing systems with format-preserving encryption and decryption engines: U.S. Patent 7,864,952[P]. 2011-1-4.

[21] Palgon G, Chambers J, Konisky D. System and methods for format preserving tokenization of sensitive information: U.S. Patent 8,458,487[P]. 2013-6-4.

[22] Liu Z, Jia C, Li J, et al. Format-preserving encryption for datetime[C]//Intelligent Computing and Intelligent Systems (ICIS), 2010 IEEE International Conference on. IEEE, 2010, 2: 201-205.

[23] Bellare M, Ristenpart T, Rogaway P, et al. Format-preserving encryption[C]//International Workshop on Selected Areas in Cryptography. Springer Berlin Heidelberg, 2009: 295-312.

[24] Yao A C. Protocols for secure computations[C]//Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on. IEEE, 1982: 160-164.

[25] Bogdanov D, Jagomägis R, Laur S. A universal toolkit for cryptographically secure privacy-preserving data mining[C]//Pacific-Asia Workshop on Intelligence and Security Informatics. Springer Berlin Heidelberg, 2012: 112-126.

[26] Hacigumus H, Iyer B, Mehrotra S. Providing database as a service[C]//Data Engineering, 2002. Proceedings. 18th International Conference on. IEEE, 2002: 29-38.

[27] Mueller R, Teubner J, Alonso G. Data processing on FPGAs[J]. Proceedings of the VLDB Endowment, 2009, 2(1): 910-921.

[28] Cheon J H, Kim A, Kim M, et al. Floating-Point Homomorphic Encryption[J]. IACR Cryptology ePrint Archive, 2016, 2016: 421.

[29] Yagisawa M. Fully Homomorphic Encryption without bootstrapping[J]. IACR Cryptology ePrint Archive, 2015, 2015: 474.

[30] Arita S, Nakasato S. Fully Homomorphic Encryption for Point Numbers[J]. IACR Cryptology ePrint Archive, 2016, 2016: 402.

[31] Fan J, Vercauteren F. Somewhat Practical Fully Homomorphic Encryption[J]. IACR Cryptology ePrint Archive, 2012, 2012: 144.

[32] Lepoint T, Naehrig M. A comparison of the homomorphic encryption schemes FV and YASHE[C]//International Conference on Cryptology in Africa. Springer International Publishing, 2014: 318-335.

[33] Papadopoulos S, Kiayias A, Papadias D. Secure and efficient in-network processing of exact SUM queries[C]//Data Engineering (ICDE), 2011 IEEE 27th International Conference on. IEEE, 2011: 517-528.

[34] Popa R A, Zeldovich N, Balakrishnan H. Guidelines for Using the CryptDB System Securely[J]. IACR Cryptology ePrint Archive, 2015, 2015: 979.

[35] Popa R A, Zeldovich N, Balakrishnan H. CryptDB: A practical encrypted relational DBMS[J]. 2011.