# Enabling Reliable and Secure Data Query Services for the Power Utilities

Qiheng Yuan(✉), Yingjie Ren, Shenglong Liu, Jingwei Liu, Gang Wang, Zhongyi Shang, Haifeng Liu, and Xiyang Liu

Big Data Center, State Grid Corporation of China, Beijing, China
yuanqiheng123@gmail.com

**Abstract.** Data confidentiality and reliable permission management have always been primary concerns that hinder enterprise customers from adopting data services of three-party professional providers (e.g., Amazon AWS). This situation is especially prominent in the State Grid Corporation of China (SGCC), in which the data is particularly sensitive. Accordingly, we hope only authorized users have access to the expected data, while unauthorized entities, including cloud service providers and unapproved internal staff, know nothing about the data. In SGCC, we utilize cloud facilities to maintain our data, and multifold efforts have been made to achieve these requirements. Specifically, for reliable authority management and access control, we devise an authority separation mechanism; the data is stored in encrypted form, and we design a set of mechanisms to enable search and query on encrypted data using searchable encryption and homomorphic encryption. In this paper, we present architectures, designs, and experiences in launching our systems.

**Keywords:** Database services · Query processing · Homomorphic encryption · Authority separation mechanism

## 1 Introduction

Managing large amounts of data *securely* and *efficiently* has always been an essential task for the State Grid [28]. Traditionally, access control and permission management methods have been deployed to protect sensitive data, *e.g.,* users' personal private information, management personnel information, and financial information. However, their limited security guarantees make them vulnerable to attacks, which normally result in terrible consequence [1]. Hence, a more rigorous method, encryption, is generally taken as a promising solution.

In practice, however, applying cryptography methods for database protection is in the face of two challenges, which seriously hinder their feasibility. First, encryption disables any parties to manipulate the data. This implies that any data query requires the users first to download the entire database, then decrypt it locally, and conduct data queries on the decrypted data. Downloading the entire data lies a huge burden for the network transmission. Second, this workflow


G. Zhai et al. (Eds.): IFTC 2021, CCIS 1560, pp. 485–499, 2022.
https://doi.org/10.1007/978-981-19-2266-4_37

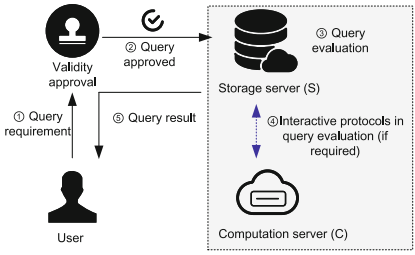


**Fig. 1.** Architecture and workflow of the data query system.

presents high requirements for the client – it requires each client to have enough storage space to hold the data and has the computational capability to perform queries on the entire dataset (Fig. 1).

Thus, we expect to design a scheme that can perform queries on the encrypted data directly, *i.e.,* do not reveal the data to the server and without downloading the data. Depending on the data type, queries can be divided into two main categories: character-oriented data query and numerical data query. Specifically, a keyword-oriented search is performed to obtain records/data entries containing specific keywords (*e.g.,* to find information on experts whose research interests are "smart grid" and "superconducting DC" in the expert database). On the other hand, when we want to query a specific record to satisfy certain numerical characteristics (*i.e.,* querying the oldest employee with more than 25 years of project review experience), we need to perform calculations (including numerical operations and data comparisons) on specific data records, etc.

In this paper, we build a complete encrypted database query scheme that supports the most common queries. Specifically, for queries that rely only on addition operations, we adopt a somewhat homomorphic encryption scheme to encrypt the data so that the server can perform queries directly on the ciphertext. For complex queries that rely on multiplication operations, we adopt a dual-server architecture, in which two servers that store encrypted data independently work together to make queries. Moreover, we devise an Authority Separation mechanism further to restrict the data exploration of the cloud server, so that the relevant third-party department must approve the user before querying and using the data. Based on the proposed framework, we can mitigate the risk of information leakages caused by unreliable management and employees while maintaining the usability of sensitive data.

## 2 Preliminary

### 2.1 Basic Cryptographic Tools

**The BCP Crypto-System.** BCP cryptosytem [12,23] is quin-tuple (`Setup, KeyGen, Enc, Dec` and `mDec`) that consists of the following algorithms:
`SetUp`($\kappa$): for the security parameter $\kappa$, choose a safe prime RSA -modulus $N = qp$ of bit $\kappa$, s.t. $p = 2p' + 1$ and $q = 2q' + 1$ for distinct prime $p'$ and $q'$. Pick a random element $g \in \mathbb{Z}^*_{N^2}$ of order $pp'qq'$ s.t. $g^{p'q'} \mod N^2 = 1 + kN$.

The plaintext space of the crypto-system is $\mathbb{Z}^*$. This algorithm output the public parameters $\texttt{PP} = (N, k, g)$ and the master key is $\texttt{MK}=(p'q')$.
`KeyGen(PP)`: the key generation algorithm which take the public parameter as input and out put the *secret/public* key pairs. For each public-secret key pair, it firstly randomly pick $a \in \mathbb{Z}_{N^2}$ and computes $h = g^a \mod N^2$ and then our put the public key $\texttt{pk} = h$ and secret key $\texttt{sk} = a$.
$\texttt{Enc}_{(\texttt{PP},\texttt{pk})}(m)$: the encryption algorithm which take the plaintext, the public key and public parameters as input and output the ciphertext. The algorithm firstly pick a random $r \in \mathbb{Z}_{N^2}$ and generate the ciphertext $(\alpha, \beta)$ as:

$$\alpha = g^r \mod N^2 and \beta = h^(1 + mN) \mod N^2$$

$\texttt{Dec}_{(\texttt{PP},\texttt{sk})}(\text{PP})$: the regular decryption algorithm that take the ciphertext, the secret key and public parameters as input and output its corresponding plaintext. In terms a key pair $\texttt{pk} = g^a \mod N^2$ $\texttt{sk}$=a, the ciphertext $(\alpha, \beta)$, it output the plaintext as:

$$m = \frac{\frac{\beta}{\alpha^a} - 1 \mod N^2}{N}$$

$\texttt{mDec}_{(\texttt{PP},\texttt{pk},\texttt{mk})}(\alpha, \beta)$: the alternative decryption algorithm which take the public key, the master key, the ciphertext and public parameters as input and output the corresponding plaintext. It first computed $a \mod N$ as:

$$a \mod N = \frac{h^{p'q'} - 1 \mod N^2}{N} \cdot k^{-1} \mod N$$

in which $k^{-1}$ denotes the inverse of $k$ modulo $N$. Then compute $r \mod N$ as

$$r \mod N = \frac{A^{p'q'} - 1 mod N^2}{N} k^{-1} \mod N$$

Then let $\delta$ denote the inverse of $p'q'$ modulo $N$ and compute $\gamma = ar \mod N$. The plaintext is finally computed as

$$m = \frac{(\frac{\beta}{g^\gamma})^{p'q'} - 1 \mod N^2}{N} \cdot \delta \mod N$$

The BCP Crypto system has flowing special properties.

1. **Additive Homomorphism.** For the cipertext encrypted using the same public key, the BCP crytosystem satisfies the addictive homomorphism.
2. **Key Homomorphism.** The key homomorphism refers to the property that, if a plaintext is encrypted with the public key $pk = pk_1 \times pk_2$, it can be successfully decrypted with the secret key $sk = sk_1 + sk_2$. Formally, it satisfies

$$m = Dec_{sk_1+sk_2}[Enc_{pk_1+pk_2}(m))]$$

3. **Double Trapdoor.** The BCP cryptosystem provides two independent mechanisms for decryption. Except for decrypting a ciphertext (encrypted with public key $p_i$) with the corresponding secret key (*i.e.,* $sk_i$) as the traditional public key cryptosystem does, the BCP cryptosystem allows a master to hold the master key and decrypt ciphertext encrypted by any public-key without knowing its corresponding secret key (*i.e.,* the second trapdoor).

**The Goldwasser–Micali Cryptosystem** is a bitwise probabilistic (semantically secure) public key encryption scheme based on the quadratic residuosity assumption. It satisfies the additive homomorphic property over $\mathbb{Z}_2$. That is, for two plaintext bit $m_1$ and $m_2 \in \{0, 1\}$,

$$Enc(m_1) \cdot Enc(m_2) \mod N = Enc(m_1 \oplus m_2)$$

As its special bitwise homomorphism, the Goldwasser-Micali Cryptosystem is often used as a basic building block to construct high-level primitives.

### 2.2 Symbols and Notations

For a plaintext $a \in \mathbb{Z}_{N^2}$ we use $Enc_{pk_A}(a)$ to denote its corresponding ciphertext encrypted by the BCP cryptosystem with public key $pk_A$. If we do not care which public key is used in encryption, we use [a] to represent the ciphertext of $a$ for short. For a plaintext $b \in \mathbb{Z}_2$, we use $\|b\|$ to denote the corresponding ciphertext encrypted with the Goldwasser-Micali encryption algorithm.

## 3 Scheme Description

### 3.1 System Architecture High Level Description

This scheme adopts two non-collusive servers – the storage server $\mathcal{S}$ and the (auxiliary) computing server $\mathcal{C}$. The whole database is stored in server $\mathcal{S}$, and each sensitive attribute (column) is encrypted using BCP cryptosystem with a separate public key, whereas the non-sensitive columns remain in plaintext form. This public key is also stored in the server S along with the ciphertext of this column. The computing server holds the master key $mk$ of the BCP cryptosystem, whereas no data is stored on the computing server.

**Operation Modes.** As some columns are encrypted whereas some are not, we denote inter-column operations between an encrypted and an unencrypted column as the [**EP**] mode and denote inter-column operations in two encrypted columns as [**EE**] mode. If there is no sensitive column involved, the database management system will execute operations as in the traditional manner, and in this paper, we will not discuss this case. For each operation, if the serve $\mathcal{S}$ gets the encrypted result (which is either used for further operations such as aggregation or used as a final classified result), we denote this case as [$\rightarrow$**E**] mode. Likewise, if the serve $\mathcal{S}$ gets the plaintext result (which is normally used for data retrieval such as select clause), we denote this case as [$\rightarrow$**P**] mode.

### 3.2 Scheme Construction

**Aggregation.** The `SUM()` operation considers the setting that, given $k$ values $a_1, a_2, ..., a_k$ in one column, it requires to compute the sum $\sigma_a = \sum_{i=1}^{k} a_i$ It serves as the basic step for other aggregation operations such as `AGV(), COUNT()`, etc.

Actually, as the addictive homomorphism of the BCP cryptosystem, the sum operation within a column can simply achieved by $\mathcal{S}$ independently, without the computing server $\mathcal{C}$ involved.

**Addition.** [**EP→E mode**] Given an encrypted column $A$ encrypted with public key $pk_A$ and a non-sensitive column $B$ in the plaintext form, we want to compute the column $C = A + B$, such that each item in C is in encrypted form and can be decrypted with secret key $sk_A$. Specifically, if $a$ is the element in $i$-th row of column A and $b$ is the element in $i$-th row of column B, we want to get c such that $Enc(c) = Enc(a + b)$ This can be achieved by the following two steps (*i.e.,* entire done by $\mathcal{S}$ without interaction)

- Encrypt $b$ with public key $pk_A$, which will result in $Enc_{pk_A}(b)$.
- based on the addictive homomorphism of the BCP cryptosystem, compute $Enc_{pk_A}(c) = Enc_{pk_A}(a) \cdot Enc_{pk_A}(b)$

[**EE →E mode**] we want to compute the column $C = A + B$, s.t. each item in $C$ is in encrypted form and can be decrypted by an individual secret key $sk_C$.

**Algorithm 1.** The cross-column addition in EE→E mode

**Input:** The storage server $\mathcal{S}$ holds encrypted column $A$ and $B$, as well as its corresponding keys $pk_A$ and $pk_B$; The computing server $\mathcal{C}$ holds the master key $mk$

**Output:** Server $\mathcal{S}$ gets the encrypted column $C$

1: Server $\mathcal{S}$:
2: $pk_C = pk_A \cdot pk_B mod N^2$;
3: **for** $i$=1 to $n$ **do**
4: $r_i \xleftarrow{\$} \mathbb{Z}_N$;
5: $\tau_i \xleftarrow{\$} \mathbb{Z}_N$;
6: $c_i = \texttt{ADD}(Enc_{pk_A}(a_i), Enc_{pk_A}(r_i))$;
7: $d_i = \texttt{ADD}(Enc_{pk_B}(b_i), Enc_{pk_B}(\tau_i))$;
8: **end for**
9: $C = (c_1, c_2, ..., c_n)$ $D = (d_1, d_2, ..., d_n)$;
10: send $pk_A, pk_B, C$ and $D$ to $\mathcal{C}$;
11: Server $\mathcal{C}$:
12: $pk_C = pk_A \cdot pk_B \ mod N^2$;
13: **for** $i$=1 to $n$ **do**
14: $x_i = mDec_{(pk_A, mk)}(c_i)$;
15: $d_i = mDec_{(pk_b, mk)}(d_i)$;
16: $z_i = Enc_{pk_C}(x_i + y_i)$;
17: **end for**
18: $Z = (z_1, z_2, ..., z_i)$;
19: send $Z$ to $\mathcal{C}$;
20: Server $\mathcal{C}$:
21: **for** $i$=1 to $n$ **do**
22: $e_i = \texttt{ADD}(z_i, Enc_{pk_C}(-r_i))$;
23: $t_i = \texttt{ADD}(e_i, Enc_{pk_B}(-\tau_i))$;
24: **end for**
25: **Output** $T = (t_1, t_2, ..., t_n)$

**Multiplication.** [**EP→E mode**] In this setting, given a sensitive column $A$ encrypted by BCP-cryptosystem with public key $pk$ and a none-sensitive column $X$ in plaintext form (with $x$ denoting an item in column $X$), we want to compute $B$ such that $B = AX$.

Recall that the ciphertext of $m$ under the BCP encryption is in the form of $c = Enc(m) = (\alpha, \beta)$, in which $\alpha = g^r mod\ N^2$ and $\beta = h^r(1 + mN) mod\ N^2$. Thus when multiplay a constant $x$ into the ciphertext $c$, the storage server just computer $c' = <\alpha^x, \beta^x>$, which will get the ciphertext of $xm$.(as proved below).

$$xc = <\alpha^x, \beta^x>$$

**Algorithm 2.** The cross-column multiplication in EE→ E mode

**Input:** The storage server $\mathcal{S}$ holds encrypted column $A$ and $B$, as well as its corresponding keys $pk_A$ and $pk_B$; The computing server $\mathcal{C}$ holds the master key $mk$

**Output:** Server $\mathcal{S}$ gets the encrypted column $C$

1: Server $\mathcal{S}$:
2: $m = \texttt{ADD}([a], Enc_{pk_A}(-r_1))$;
3: $n = \texttt{ADD}([b], Enc_{pk_A}(-r_2))$;
4: send $pk_A, pk_B, m$ and $n$ to $\mathcal{C}$;
5: Server $\mathcal{C}$:
6: $pk_C = pk_A \times pk_B$;
7: $x = mDec_{(pk_A,mk)}(m)$;
8: $y = mDec_{(pk_B,mk)}(n)$ ;
9: $z = Enc_{pk_A}(m \times n)$;
10: send $z$ to Server $\mathcal{S}$:
11: Server $\mathcal{S}$:
12: At this phase the server $\mathcal{S}$ aquires $CT_0$–the ciphertext of $ab - ar_2 - br_1 + r_1r_2$ encrypted under public key $pk_C$
13: compute $CT_a = Enc_{pk_A}(ar_2), CT_b = Enc_{pk_B}(br_1)$
14: send $CT_a, CT_b$ to the Server $\mathcal{C}$
15: Server $\mathcal{C}$ decrypt the $CT_a$ and $CT_b$, re-encrypt them with $pk_C$, getting $CT'_a$ and $CT'_b$, the ciphertext of $ar_2$ and $br_1$ encrypted with the public key $pk_C$
16: send $CT'_a$ and $CT'_b$ to Server $\mathcal{S}$
17: Server $\mathcal{S}$:
18: $CT_{rr} = Enc_{pk_c}(r_1r_2)$
19: $CT = CT_0 + CT'_a + CT'_b - CT_{rr}$
20: **Output** $CT$

$$\alpha = g^r mod\ N^2, \beta = h^2(1+mN) mod\ N^2$$

$$\alpha^a = g^{rax} mod\ N^2$$

$$\frac{\beta^x}{\alpha^{xa}} = \frac{h^{xr}(1+mN)^x mod\ N2}{g^{rxa} mod\ N^2} = \frac{g^{xar}(1+mN)^x mod\ N2}{g^{rxa} mod\ N^2} =$$

$$(1+mN)^x mod\ N^2 = (1 + xmN + C_x^2 mN^2 + ...) mod N^2 = (1+xmN) mod\ N^2$$

$$Dec = \frac{\frac{\beta^x}{\alpha^{xa}} - 1 mod\ N^2}{N} mod N^2 = \frac{xmN}{N} mod\ N^2 = xm$$

**[EE→E mode]** Given an encrypted column $A$ (encrypted with $pk_A$) and column $B$ (encrypted with $pk_B$), we want to computed $C = A \times B$, such that $C$ is in the encrypt form and can be decrypted by a separate key $sk_C$. Recall that the ciphertext of BCP cryptosystem is in the form like $< \alpha, \beta >$. We denote the $i$-th item in column $A$ as $[a] =< \alpha, \beta >$ and the $i$-th item in column $B$ as $[b] =< \alpha', \beta' >$. The output $[c]$ of Algorithm 2 (*i.e., CT*) is the ciphertext of $a \times b$ that can be decrypted with secret key $sk_C = sk_A + sk_B$.

**Comparison.** For the functionality of secure comparison protocol, the server $\mathcal{S}$ holds two values (either both in encrypted form, or one in encrypted and the other in unencrypted form) and hopes to know the relationship (*i.e.,* whether $a \leq b$ or not) among the two numbers. Normally, the server $\mathcal{C}$ is an external service provider that concentrates on providing auxiliary computing service (especially cryptographic operation). Thus, as one design principle, we hope the server $\mathcal{C}$ gets as little information as possible.

As the server $\mathcal{S}$ holds the BCP public key of each column, both EE or EP mode can be transformed to EE mode.

**Table 1.** Functionality of comparison protocols.

| The protocols | Inputs of $\mathcal{S}$ | Inputs of $\mathcal{C}$ | Protocols' Output |
|---|---|---|---|
| P1: Comparing encrypted data enrypted by same key with plaintext comparison output | $pk, sk_{QR}, [a], [b]$ | $mk, pk_{QR}$ | $(a \leq b)?$ |
| P2: Comparing encrypted data enrypted by same key with encrypted comparison output | $pk, pk_{QR}, [a], [b]$ | $mk, sk_{QR}$ | $\|(a \leq b)?\|$ |
| P3: Comparing encrypted data enrypted by different keys with plaintext comparison output | $pk_A, pk_B, sk_{QR}, [a], [b]$ | $mk, pk_{QR}$ | $(a \leq b)?$ |
| P4: Comparing encrypted data enrypted by different keys with encrypted comparison output | $pk_A, pk_B, pk_{QR}, [a], [b]$ | $mk, sk_{QR}$ | $\|(a \leq b)?\|$ |
| P5: Changing encryption schemes | $pk_{QR}, \|\|t\|\|$ | $mk, sk_{QR}$ | $\|\|t\|\|$ |
| P6: Equality test | $pk_A, pk_B, pk_{QR}, [a], [b]$ | $mk, sk_{QR}$ | $\|(a = b)\|?$ |

**[EE→E mode]** Get the BCP-encrypted comparison result, which is normally used for `SELECT COUNT(*)`.

**[EE→P mode]** Get the comparison result in plaintext form. Normally used for directly retrieve the tuples that satisfy the query condition (*e.g.,* `SELECT COUNT` * where *Production* ≤ *Sells*).

The functionality of comparison protocols in different modes is listed in Table 1.

**Equality Test.** Similar to the comparison protocol, all cases can be converted to equality test among two encrypted ciphertexts (either encrypted in the same or different keys). As demonstrated in Sect. 3.2, we have composed protocols for encrypted ciphertext comparison, *i.e.,* for two encrypted items $[a]$ and $[b]$, the server $\mathcal{S}$ can get the encrypted or unencrypted bit t indicating whether $a \leq b$. In order to determine whether $a = b$, the servers firstly run the comparison protocol to get the encrypted bit $||t_1||$, indicating whether $a \leq b$; and then, ran the comparison to get the encrypted bit $||t_2||$ indicating whether $b \leq b$. Note that, during the excursion of these two protocol, the server $\mathcal{S}$ get the encrypt bits whereas the server $\mathcal{C}$ get no information about $a$ and $b$. Then the server $\mathcal{S}$ computer $||t|| = ||t_1|| \cdot ||t_2||$, which equals to $||t_1 \oplus t_2||$, and $t_1 \oplus t_2 = 1$ if and only if $t_1 = t_2$ (*i.e.,* when $a \leq b$ and $b \leq a$ satisfies simultaneously, indicating $a = b$).

After acquiring the QR-encrypted comparison result, if we want to get the plaintext of the result, the server $\mathcal{S}$ first generate a random bit $\tau$ and blind the QR-encrypted $||t||$ with $\tau$ and send the blind result to server $\mathcal{C}$. Then, the server $\mathcal{C}$ decrypted the blind equality test result and sent back the decrypted result to $\mathcal{S}$, which will afterward remove the blind factor $\tau$ and get the result in plaintext setting (Algorithm 8). Likewise, if the server is required to get BCP encrypted result, then we apply the *Chenging Encryption Schemes* (Algorithm 9).

Unlike theirs, in our scheme, each column is encrypted with BCP cryptosystem of independent keys, and the randomness of each ciphertext is derived from

**Algorithm 3.** Comparing same BCP-key encrypted data with plaintext output

**Input:**
1: Server $\mathcal{S}$: $pk_A, pk_B, sk_{QR}, [a], [b]$
2: Server $\mathcal{C}$: $mk, pk_{QR}$
**Output:** $\mathcal{S}$ get $(a \leq b)$?
3: Server $\mathcal{S}$:
4: $[x] \leftarrow [2^l] \cdot [b] \cdot [a]^{-1} mod\ N^2$;
5: $r \xleftarrow{\$} \{0,1\}^{l+\lambda}$;
6: $[z] \leftarrow [x] \cdot [r]\ mod\ N^2$
7: sent $[z]$ to $\mathcal{C}$;
8: Server $\mathcal{C}$:
9: $z = mDec([z])$;
10: Server $\mathcal{S}: c \leftarrow r\ mod\ 2^l$;
11: Server $\mathcal{C}: d \leftarrow z\ mod\ 2^l$;
12: Using another comparison protocol (DGK protocol) to get $\|t'\|$(*i.e.,* the ciphertext of $t'$ with Goldwasser-Micali cryptosystem), s.t.

$$t' = \begin{cases} 1, if\ \text{c<d} \\ 0, \text{otherwise} \end{cases}$$

13: $\mathcal{S}$: encrypt the $l$-th bit of $r$ getting $\|r_l\|$ and send $\|r_l\|$ to $\mathcal{C}$;
14: $\mathcal{C}$: encrypt the $l$-th bit of $z$ getting $\|z_l\|$
15: Compute $\|t\| \leftarrow \|t'\| \cdot \|z_l\| \cdot \|r_l\|$
16: send $\|t\|$ to $\mathcal{S}$
17: $\mathcal{S}$: decrypt $||t||$ and get the result $t$

the cryptosystem itself. In this way, a random number does not exist that influences all the encrypted data items in a whole tuple, and thus, the Cartesian product is natively supported. Joint is actually performing selection operation on the Cartesian product, and accordingly, it is natively supported.

## 4 Performance Evaluation

We deployed our system on top of Amazon AWS and Aliyun and conducted a comprehensive performance evaluation of our scheme. Specifically, we deploy the storage server (*i.e.,* server $\mathcal{S}$) on a standard VM (with a dual-core CPU @2.5 GHz, 32-GB memory, and 256-GB SSD storage) rented from Amazon EC2 located at N. Virginia datacenter. The computing server (*i.e.,* server $\mathcal{C}$) is run on Aliyun (Silicon Valley datacenter). Both server $\mathcal{S}$ and server $\mathcal{C}$ are connected with a high-bandwidth network. Our experiments are designed to answer the following two questions:
Q1: What is the efficiency of the building blocks, and
Q2: How practical our proposed method is for typical database query workloads.

To answer Q1 we implement each cryptographic primitive with C++, and we adopt the GMP(https://gmplib.org) and NTL (http://www.shoup.net/ntl/) library for large integer representation and algebraic manipulation. To answer question Q2, we build a prototype based on our design. We use MySQL 5.7.19 as the underlying DBMS. To examine the performance of our scheme in the general setting, we conducted the evaluation with the TPC-H benchmark.

**Performance of Each Cryptographic Primitive.** We tested each building block used in our scheme (*i.e.,* used in query evaluation). In our experiment, we run the aforementioned algorithms/protocols 50 times and record their average exerting time in Table 2. From Table 2 we can see that all the building blocks

**Algorithm 4.** Comparing data encrypted by same BCP-key with QR-encrypted comparison output

**Input:**
1: Server $\mathcal{S}$: $pk_A, pk_B, pk_{QR}, [a], [b]$
2: Server $\mathcal{C}$: $mk, sk_{QR}$
**Output:** $\mathcal{S}$ get $||(a \leq b)?||$
3: Server $\mathcal{S}$:
4: $[x] \leftarrow [2^l] \cdot [b] \cdot [a]^{-1} mod\ N^2$;
5: $r \xleftarrow{\$} \{0,1\}^{l+\lambda}$;
6: $[z] \leftarrow [x] \cdot [r]\ mod\ N^2$
7: sent $[z]$ to $\mathcal{C}$;
8: Server $\mathcal{C}$:
9: $z = mDec([z])$;
10: Server $\mathcal{S} : c \leftarrow r\ mod\ 2^l$;
11: Server $\mathcal{C} : d \leftarrow z\ mod\ 2^l$;
12: Using another comparison protocol (DGK protocol) the server get the $\|t'\|$(*i.e.,* the ciphertext of $t'$ with Goldwasser-Micali cryptosystem), s.t.

$$t' = \begin{cases} 1, if\ \text{c<d} \\ 0, \text{otherwise} \end{cases}$$

13: $\mathcal{S}$: encrypt the $l$-th bit of $r$ getting $\|r_l\|$;
14: $\mathcal{C}$: encrypt the $l$-th bit of $z$ getting $\|z_l\|$ and send $\|z_l\|$ to $\mathcal{S}$;
15: $\mathcal{S}$ compute $\|t\| \leftarrow \|t'\| \cdot \|z_l\| \cdot \|r_l\|$

**Table 2.** Overhead of each cryptographic algorithm and protocol.

| Protocals | $\mathcal{S}$ (ms) | $\mathcal{C}$ (ms) | Network lattency |
|---|---|---|---|
| Homomorphic add | 0.74 | 0.00 | 0.00 |
| Cross-column add in [EP → E] mode | 1.22 | 0.00 | 0.00 |
| Cross-column add in [EE→E] mode | 4.31 | 1.12 | 204.7 |
| Cross-column multiplication in [EP→E] mode | 2.53 | 0.00 | 0.00 |
| Cross-column multiplication in [EE→E] mode | 5.56 | 4.37 | 832.1 |
| Comparison in [EE→E] mode | 8.43 | 0.00 | 0.00 |
| Comparison in [EE→p] mode | 4.87 | 1.77 | 243.1 |
| Fast comparison | 2.21 | 2.11 | 242.8 |
| Equality test | 9.11 | 9.72 | 1523.4 |
| Changing encryption scheme | 4.43 | 2.21 | 265.4 |

can be finished in millisecond-level, which indicates that they are efficiently constructed and can be used to build practical query evaluation schemes on an encrypted database.

**The TPC-H Benchmark.** We run our scheme on TPC-H (version 3.0) benchmark (http://www.tpc.org/tpch). Meanwhile, we also compare the performance of our scheme with CryptDB, a state-of-the-art system for encrypted database queries. There are 22 decision-support queries named Q1 to Q22 in the TPC-H workload, and we select Q1-18 to present the performance evaluation. Among them, some of the workloads (*e.g.,* Q13 and Q16) are so complicated that they cannot be run on both our scheme and CryptDB in encrypted form. We record the running time of each workload and record the total time consumption on Table 3. In general, encrypted data query in both our system and CryptDB takes

**Algorithm 5.** Comparing data encrypted by different BCP keys with plaintext output

**Input:**
Server $\mathcal{S}$: $pk_A, pk_B, sk_{QR}, [a], [b]$
Server $\mathcal{C}$: $mk, pk_{QR}$
**Output:** $\mathcal{S}$ get $(a \leq b)$?
Server $\mathcal{S}$:
$r_1, r_2 \xleftarrow{\$} \{0,1\}^{l+\lambda}$
$[x] = [a] \cdot Enc_{pk_A}(r_1)$
$[y] = [b] \cdot Enc_{pk_B}(r_2)$;
Sent $[x]$ and $[y]$ to $\mathcal{C}$
Server $\mathcal{C}$:
$x = mDec_{(mk,pk_A)}([x])$;
$y = mDec_{(mk,pk_B)}([y])$;
$z = 2^l + y - x$;
$d = z \mod 2^l$;
Send $z$ to $\mathcal{S}$
Server $\mathcal{S}$:
$r = r_2 - r_1$;
$c = r \mod 2^l$;
Using another comparison protocol (DGK protocol) to get $\|t'\|$(*i.e.,* the ciphertext of $t'$ with Goldwasser-Micali cryptosystem),s.t.

$$t' = \begin{cases} 1, if \text{ c<d} \\ 0, \text{otherwise} \end{cases}$$

Server$\mathcal{S}$:
encrypt the $l$-th bit of $r$ getting $\|r_l\|$ and send $\|r_l\|$ to $\mathcal{C}$;
Server $\mathcal{C}$:
encrypt the $l$-th bit of $z$ getting $\|z_l\|$ ;
Compute $\|t\| \leftarrow \|t'\| \cdot \|z_l\| \cdot \|r_l\|$
send $\|t\|$ to $\mathcal{S}$
Server $\mathcal{S}$: decrypt $||t||$ and get the comparison result $t$

**Table 3.** Performance of our scheme under TPC-H workload, compared with CryptDB; "–" denotes the corresponding system cannot work on this workload.

| Benchmark | Q1 | Q2 | Q3 | Q4 | Q5 | Q6 | Q7 | Q8 | Q9 |
|---|---|---|---|---|---|---|---|---|---|
| Plaintext | 121.51 | 64.44 | 81.23 | 45.33 | 176.32 | 62.32 | 291.27 | 201.29 | 672.01 |
| CryptoDB | 4860.22 | 148.23 | 332.12 | 182.23 | 534.34 | 582.21 | 973.32 | 523.23 | 2932.21 |
| Our scheme | 3731.98 | 127.23 | 1623.32 | 164.71 | 201.23 | 1243.32 | 628.23 | 667.23 | 2123.21 |
| Benchmark | Q10 | Q11 | Q12 | Q13 | Q14 | Q15 | Q16 | Q17 | Q18 |
| Plaintext | 132.42 | 56.73 | 71.20 | 398.22 | 72.34 | 64.23 | 41.22 | 3423.32 | 1187.23 |
| CryptoDB | 542.23 | 263.12 | 362.12 | – | 534.32 | – | – | 17342.66 | 73238.78 |
| Our scheme | 412.31 | 59.23 | 372.81 | – | 342.23 | 372.23 | – | 12029.30 | 36439.91 |

several times (less than 10 ×) than query on plaintext setting. Only a few of them take longer times (*e.g.,* Q1, Q17 and Q18), but still less than 100× of plaintext data query. Most importantly, we can also see that our scheme achieves a better performance than CryptDB in most cases.

## 5 Related Works

**Encrypted Database.** The notion of executing SQL query on encrypted relational database was firstly considered by Hacigümüş *et al.* [16]. They proposed a prospective architecture in which the client outsources his encrypted database along with some additional auxiliary information (*i.e.,* serves as a secure index)

**Algorithm 6 .** Comparing data encrypted by different BCP-key with QR-encrypted comparison output

**Input:**
1: Server $\mathcal{S}$: $pk_A, pk_B, pk_{QR}, [a], [b]$
2: Server $\mathcal{C}$: $mk, sk_{QR}$
**Output:** $\mathcal{S}$ get $||(a \leq b)?||$
3: Server $\mathcal{S}$:
4: $r_1, r_2 \xleftarrow{\$} \{0,1\}^{l+\lambda}$;
5: $[x] = [a] \cdot Enc_{pk_A}(r_1)$;
6: $[y] = [b] \cdot Enc_{pk_B}(r_2)$;
7: Sent $[x]$ and $[y]$ to $\mathcal{C}$
8: Server $\mathcal{C}$:
9: $x = mDec_{(mk,pk_A)}([x])$;
10: $y = mDec_{(mk,pk_B)}([y])$;
11: $z = 2^l + y - x$;
12: $d = z \mod 2^l$;
13: Send $z$ to $\mathcal{S}$
14: Server $\mathcal{S}$:
15: $r = r_2 - r_1$;
16: $c = r \mod 2^l$;
17: Using another comparison protocol (DGK protocol) the server get the $\|t'\|$(*i.e.,* the ciphertext of $t'$ with Goldwasser-Micali cryptosystem), s.t.

$$t' = \begin{cases} 1 & if\ c¦d \\ 0 & otherwise \end{cases}$$

18: Server$\mathcal{S}$: encrypt the $l$-th bit of $r$ getting $\|r_l\|$;
19: Server $\mathcal{C}$: encrypt the $l$-th bit of $z$ getting $\|z_l\|$ and send it to $\mathcal{S}$;
20: Server $\mathcal{S}$: compute $\|t\| \leftarrow \|t'\| \cdot \|z_l\| \cdot \|r_l\|$

**Algorithm 7.** Equality Test (protocol 4)

**Output:** $\mathcal{S}$ get $\|u\| = (a = b?)$
1: Using protocol 2 to compute $\|u_1\|$, which is the comparison result of $(a \leq b?)$ encrypted under QR.
2: Using protocol 2 to compute $\|u_2\|$, which is the comparison result of $(b \leq a?)$ encrypted under QR.
3: $\|u\| = \|u_1 \oplus u_2\| = \|u_1\| \cdot \|u_2\|$

into the server and can latterly issue the SQL queries to this database. In their scheme, the database is encrypted by a traditional encryption scheme (*i.e.,* symmetric encryption scheme such as DES and AES), and the aforementioned split-and-transformation mechanism is achieved by a delicately designed algebraic framework. Followed by this approach, Hacıgümüs *et al.* proposed a query optimization method in [18] and constructed a secure DBMS that support aggregation query in [17]. However, for the sake of query processing efficiency, DBMS built in this approach only provides very limited notions of data privacy.

**Property-Preserving Encryption Approach.** In order to accelerate data processing operations meanwhile provide appropriate privacy guarantees, researchers proposed some specific cryptographic primitives that support particular calculations to be performed on encrypted data. Specifically, for example, Order-Preserving Encryption [2,9] enables the ciphertext to maintain the numerical order of its corresponding plaintext, and the Order-Revealing Encryption[11] enables efficient comparison among multiple (randomly-encrypted) ciphertexts. Deterministic encryption [7] have advantage on equality search and match. Using these primitives, Popa *et al.* [25] proposed the first practical system, CryptDB,

**Algorithm 8.** Changing QR-encrypted (equality test) result into plaintext

**Input:**
Server $\mathcal{S}$ : $pk_{QR}, ||t||$
Server $\mathcal{C}$ : $sk_{QR}$
**Output:** $\mathcal{S}$ get $t = (a = b?)$
Server $\mathcal{S}$:
$\tau \xleftarrow{\$} \{0,1\}$
$x' = ||t|| \cdot ||\tau||$
Send $x'$ to server $\mathcal{C}$
Server $\mathcal{C}$
$x = Dec_{QR}(x')$;
Send back $x$ to $\mathcal{S}$
Server $\mathcal{S}$:
$t = x \oplus \tau$

**Algorithm 9.** Changing Encryption Schemes (Protocol 5)

**Input:**
Server $\mathcal{S}$ : $pk_{QR}, ||t||$
Server $\mathcal{C}$ : $sk_{QR}$
**Output:** $\mathcal{S}$ get $v = [t]$
Server $\mathcal{S}$:
$x_1 = ||t|| \cdot ||0||$;
$x_2 = ||t|| \cdot ||1||$;
$r \xleftarrow{\$} \{0,1\}$
**if** $r = 1$ **then**
    $s_1 = x_1, s_2 = x_2$;
**else**
    $s_1 = x_1, s_2 = x_1$;
**end if**
Send $s_1$ and $s_2$ tp $\mathcal{C}$
Server $\mathcal{C}$:
$m_1 = Dec_{QR}(s_1), m_2 = Dec_{QR}(s_2)$;
$n_1 = Enc^{(BCP)}_{pk_A}(m_1)$
$n_2 = Enc^{(BCP)}_{pk_A}(m_2)$;
Send $n_1, n_2$ to $\mathcal{S}$
Server $\mathcal{S}$:
**if** $r = 1$ **then**
    $v = n_1$;
**else**
    $v = n_2$
**end if**
**Output** $v$;

an integrated system that can perform query processing on encrypted data. Afterwards, several systems, like MONOMI [29], are proposed.

**FHE and SMPC Based Approach.** Fully homomorphic encryption (FHE) [15] is a cryptographic encryption primitive that enables to compute any function on encrypted data. Boneh *et al.* [10] firstly implement the database-query functionally with a specific homomorphic encryption scheme. This line of work is based on the prerequisite that efficient fully homomorphic encryption schemes exist; nevertheless, the efficiency for FHE is still far more satisfactory. Similarly, SMPC enables multiple distributed parties to jointly compute an arbitrary functionality in an privacy-preserving manner. Secure database systems implemented SMPC include Blindseer [14,22], Arx [24], sharemind [8] and SDB [19,32].

**Secure Hardware Based Approach.** Secure hardware enclaves (*e.g.,* Intel SGX [13] and Catalyst [21] *etc.*) promise data confidentiality and secure execution of arbitrary computation. The TrustDB [5,6], constructed with an ordinary commodity cryptographic co-processor, is the first practical outsourced database prototype that supports a subset of SQL operation, and the Cipherbase [3,4] achieves the similar functionality with FPGA serving as the tamper-proof hard-

ware. SGX based systems include include VC3 [26], Ryoan [20] and Opaque [33]. However, the secure hardware may not be as "secure" as it claims [31], and there are lots of effective attacks targeted to hardware that can totally devastate the underlying systems[27,30].

## 6 Conclusion

Reliable, secure, and trustworthy services are essential for industries with highly sensitive data. In SGCC, we deployed our systems that ensure only authorized users have access to the expected data, while unauthorized entities, including cloud service providers and unapproved internal staff, know nothing about the data. To achieve these goals, we devise an authority separation mechanism, store data in encrypted form, and design a set of mechanisms to enable search and query on encrypted data using searchable encryption (SE) and homomorphic encryption (HE). Experimental and real-work running experiences indicate the practicability of our system.

**Acknowledgement.** This work is funded by the "Research on Key Technologies for Secure Query on Encrypted Electric Power Data" program of the Big Data Center, SGCC.

## References

1. Yahoo! data breaches. https://en.wikipedia.org/wiki/Yahoo!_data_breaches
2. Agrawal, R., Kiernan, J., Srikant, R., Xu, Y.: Order preserving encryption for numeric data. In: Proceedings of SIGMOD, pp. 563–574. ACM (2004)
3. Arasu, A., et al.: Secure database-as-a-service with cipherbase. In: Proceedings of SIGMOD, pp. 1033–1036. ACM (2013)
4. Arasu, A., Eguro, K., Joglekar, M., Kaushik, R., Kossmann, D., Ramamurthy, R.: Transaction processing on confidential data using cipherbase. In: Proceedings of ICDE, pp. 435–446. IEEE (2015)
5. Bajaj, S., Sion, R.: TrustedDB: a trusted hardware based database with privacy and data confidentiality. In: Proceedings of SIGMOD, pp. 205–216. ACM (2011)
6. Bajaj, S., Sion, R.: TrustedDB: a trusted hardware-based database with privacy and data confidentiality. IEEE Trans. Knowl. Data Eng. **26**(3), 752–765 (2014)
7. Bellare, M., Fischlin, M., O'Neill, A., Ristenpart, T.: Deterministic encryption: definitional equivalences and constructions without random oracles. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 360–378. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85174-5_20
8. Bogdanov, D., Laur, S., Willemson, J.: Sharemind: a framework for fast privacy-preserving computations. In: Jajodia, S., Lopez, J. (eds.) ESORICS 2008. LNCS, vol. 5283, pp. 192–206. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88313-5_13
9. Boldyreva, A., Chenette, N., O'Neill, A.: Order-preserving encryption revisited: improved security analysis and alternative solutions. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 578–595. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_33

10. Boneh, D., Gentry, C., Halevi, S., Wang, F., Wu, D.J.: Private database queries using somewhat homomorphic encryption. In: Jacobson, M., Locasto, M., Mohassel, P., Safavi-Naini, R. (eds.) ACNS 2013. LNCS, vol. 7954, pp. 102–118. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38980-1_7
11. Boneh, D., Lewi, K., Raykova, M., Sahai, A., Zhandry, M., Zimmerman, J.: Semantically secure order-revealing encryption: multi-input functional encryption without obfuscation. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 563–594. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_19
12. Bresson, E., Catalano, D., Pointcheval, D.: A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications. In: Laih, C.-S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 37–54. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-40061-5_3
13. Costan, V., Lebedev, I., Devadas, S., et al.: Secure processors part I: background, taxonomy for secure enclaves and Intel SGX architecture. Found. Trends® Electron. Des. Autom. **11**(1–2), 1–248 (2017)
14. Fisch, B.A., et al.: Malicious-client security in blind seer: a scalable private DBMS. In: Proceedings of SP, pp. 395–410. IEEE (2015)
15. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of STOC, p. 169. ACM Press (2009)
16. Hacigümüş, H., Iyer, B., Li, C., Mehrotra, S.: Executing SQL over encrypted data in the database-service-provider model. In: Proceedings of SIGMOD, pp. 216–227. ACM (2002)
17. Hacıgümüş, H., Iyer, B., Mehrotra, S.: Efficient execution of aggregation queries over encrypted relational databases. In: Proceedings of DASFAA, pp. 633–650 (2004)
18. Hacıgümüş, H., Iyer, B., Mehrotra, S.: Query optimization in encrypted database systems. In: Zhou, L., Ooi, B.C., Meng, X. (eds.) DASFAA 2005. LNCS, vol. 3453, pp. 43–55. Springer, Heidelberg (2005). https://doi.org/10.1007/11408079_7
19. He, Z., et al.: SDB: a secure query processing system with data interoperability. Proc. VLDB **8**(12), 1876–1879 (2015)
20. Hunt, T., Zhu, Z., Xu, Y., Peter, S., Witchel, E.: Ryoan: a distributed sandbox for untrusted computation on secret data. In: Proceedings of OSDI, pp. 533–549. USENIX Association (2016)
21. Liu, F., Lee, R.B.: Random fill cache architecture. In: Proceedings of MICRO, pp. 203–215. IEEE (2014)
22. Pappas, V., et al.: Blind seer: a scalable private DBMS. In: Proceedings of S&P, pp. 359–374. IEEE (2014)
23. Peter, A., Tews, E., Katzenbeisser, S.: Efficiently outsourcing multiparty computation under multiple keys. IEEE Trans. Inf. Forensics Secur. **8**(12), 2046–2058 (2013)
24. Poddar, R., Boelter, T., Popa, R.A.: Arx: a DBMS with semantically secure encryption (2017). https://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-111.pdf
25. Popa, R.A., Redfield, C., Zeldovich, N., Balakrishnan, H.: CryptDB: protecting confidentiality with encrypted query processing. In: Proceedings of SOSP, pp. 85–100. ACM (2011)
26. Schuster, F., et al.: VC3: trustworthy data analytics in the cloud using SGX. In: Proceedings of S&P, pp. 38–54. IEEE (2015)
27. Shih, M.-W., Lee, S., Kim, T., Peinado, M.: T-SGX: eradicating controlled-channel attacks against enclave programs. In: Proceedings of NDSS (2017)

28. Tu, C., He, X., Shuai, Z., Jiang, F.: Big data issues in smart grid-a review. Renew. Sustain. Energy Rev. **79**, 1099–1107 (2017)
29. Tu, S., Kaashoek, M.F., Madden, S., Zeldovich, N.: Processing analytical queries over encrypted data. In: Proceedings of VLDB, vol. 6, pp. 289–300. VLDB Endowment (2013)
30. Van Bulck, J., Weichbrodt, N., Kapitza, R., Piessens, F., Strackx, R.: Telling your secrets without page faults: stealthy page table-based attacks on enclaved execution. In: Proceedings of USENIX Security. USENIX Association (2017)
31. Weichbrodt, N., Kurmus, A., Pietzuch, P., Kapitza, R.: AsyncShock: exploiting synchronisation bugs in intel SGX enclaves. In: Askoxylakis, I., Ioannidis, S., Katsikas, S., Meadows, C. (eds.) ESORICS 2016. LNCS, vol. 9878, pp. 440–457. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45744-4_22
32. Wong, W.K., Kao, B., Cheung, D.W.L., Li, R., Yiu, S.M.: Secure query processing with data interoperability in a cloud database environment. In: Proceedings of SIGMOD, pp. 1395–1406. ACM (2014)
33. Zheng, W., Dave, A., Beekman, J.G., Popa, R.A., Gonzalez, J.E., Stoica, I.: Opaque: an oblivious and encrypted distributed analytics platform. In: Proceedings of NSDI, pp. 283–298. USENIX Association (2017)