Enabling Encrypted Rich Queries in Distributed Key-Value Stores

Yu Guo, Xingliang Yuan, Xinyu Wang, Cong Wang, *Senior Member, IEEE*, Baochun Li, *Fellow, IEEE*, and Xiaohua Jia, *Fellow, IEEE*

Abstract—To accommodate massive digital data, distributed data stores have become the main solution for cloud services. Among others, key-value stores are widely adopted due to their superior performance. But with the rapid growth of cloud storage, there are growing concerns about data privacy. In this paper, we design and build EncKV, an encrypted and distributed key-value store with rich query support. First, EncKV partitions data records with secondary attributes into a set of encrypted key-value pairs to hide relations between data values. Second, EncKV uses the latest cryptographic techniques for searching on encrypted data, i.e., searchable symmetric encryption (SSE) and order-revealing encryption (ORE) to support secure exact-match and range-match queries, respectively. It further employs a framework for encrypted and distributed indexes supporting query processing in parallel. To address inference attacks on ORE, EncKV is equipped with an enhanced ORE scheme with reduced leakage. For practical considerations, EncKV also enables secure system scaling in a minimally intrusive way. We complete the prototype implementation and deploy it on Amazon Cloud. Experimental results confirm that EncKV preserves the efficiency and scalability of distributed key-value stores.

Index Terms—Encrypted key-value store, searchable encryption, order-revealing encryption

1 Introduction

In order to manage massive data records in large-scale applications, distributed data stores are fast developed and draw increasing attention. Among others, key-value (KV) stores such as Redis [2], DynamoDB [3] and RAMCloud [4] are one of the most popular production systems, due to their strength of performance, scalability, and fault tolerance. To enrich their data management features, efforts [2], [5], [6] are being made to allow expressive queries via secondary attributes, in addition to accessing data via the primary key. On the other hand, plaintext data stores face critical concerns of data privacy due to growing data breach incidents [7]. Therefore, there is an urgent need to design new solutions for distributed KV stores to meet stringent privacy requirements for big data applications.

To address this issue, two research directions are recently explored. The first is to focus on enabling specific queries over encrypted data, such as searchable symmetric encryption (SSE) for keyword search [8], [9], [10] and order-revealing encryption (ORE) for order comparison [11], [12], [13]. The other direction is to develop comprehensive solutions to support rich queries via various primitives, such as CryptDB [14] and BlindSeer [15]. Unfortunately, neither of them is specifically designed for distributed KV stores. Cryptographic primitives do not consider the deployment in real-world systems, while most of existing encrypted databases focus on the centralized setting, treating the underlying data store as a black box. It is questionable whether the performance benefits of KV stores can still be preserved.

In this work, we aim to design EncKV, an encrypted KV store with rich query support. As a starting point, we build EncKV on top of the framework as proposed in our previous work [16]. There are two salient features of this framework. The first is that data values of each record are mapped into encrypted KV pairs, which protects the relations between data values and mitigates inference attacks [17]. The second feature is that this distributed index framework facilitates secure queries in parallel. It co-locates the encrypted data records and corresponding indexes at the same nodes, so as to avoid inter-node interaction during queries. However, this preliminary work enables limited functionality, and rich queries over encrypted data are yet to be supported.

To achieve our goal, we first classify queries in KV stores into two common categories, i.e., exact-match queries and range-match queries. To balance security and performance, EncKV leverages the practical primitives for searching over encrypted data, i.e., searchable symmetric encryption (SSE) [8], [10] and order-revealing encryption (ORE) [13], [18]. They are recognized with stronger security notions than property-preserving encryption, while

- *Y. Guo and X. Jia are with the Department of Computer Science, City University of Hong Kong, Kowloon Tong, Hong Kong. E-mail: y.guo@my.cityu.edu.hk, csjia@cityu.edu.hk.*
- *X. Yuan is with the Faculty of Information Technology, Monash University, Clayton, VIC 3800, Australia. E-mail: xingliang.yuan@monash.edu.*
- *X. Wang and C. Wang are with the Department of Computer Science, City University of Hong Kong, Hong Kong, and also with the City University of Hong Kong, Shenzhen Research Institute, Shenzhen, Nanshan 518172, China. E-mail: xy.w@my.cityu.edu.hk, congwang@cityu.edu.hk.*
- *B. Li is with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ontario M5S 1A1, Canada. E-mail: bli@ece.toronto.edu.*

Manuscript received 7 Jan. 2018; revised 8 Oct. 2018; accepted 26 Nov. 2018. Date of publication 7 Dec. 2018; date of current version 15 May 2019. (Corresponding author: Cong Wang.) Recommended for acceptance by S. Chen. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TPDS.2018.2885519

outperforming sophisticated ORAM and secure multi-party computation from the perspective of efficiency [19].

For exact-match queries, EncKV carefully integrates Cash et al.'s SSE scheme [10] into the local index framework, and customizes it to support exact-match queries via encrypted secondary attributes. Here, we use KV pairs to index records that match the same attribute, and each record of this attribute is distinguished by a stateful counter. As a result, EncKV's exact-match indexes hold the security of SSE and can be stored in any KV store for easy deployment.

For range-match queries, EncKV carefully integrates Lewi and Wu's scheme [13], one of the latest ORE schemes. Intuitively, ORE ciphertext achieves semantic security, while order relations are revealed during dedicated comparison protocols. By applying ORE schemes to our system, an outside attacker who can only access ORE ciphertexts will not derive any useful information. But we observe that directly using this ORE scheme would suffer from recent leakage-abuse attacks [12], [20], because the comparison protocol leaks the order relations of ciphertexts, as well as the first block position in which the ciphertext differs. The latter information could reveal the order relations of ciphertexts which are not compared yet. To address this issue, we first propose to protect order relations by tokenizing the orders (i.e., ">" and "<") embedded in queries and ORE ciphertexts. Accordingly, the servers will not know whether the matched results are greater or smaller than the query values. To protect the position of the first block that differs, we follow the design philosophy of Cash et al.'s scheme [21] to conduct the comparison protocol on permuted block ciphertexts. But unlike that scheme performing expensive pairing operations, our design still preserves efficiency by leveraging symmetric-key based operations only. Due to the reduced leakage, our design effectively mitigates the existing attacks against ORE.

To further enable system scaling, we note that adding new nodes or removing old nodes will involve relocation of the encrypted index and data. Simply relying on the client to relocate the index and data of the affected nodes will stop the query service, as the query cannot be processed until the relocation is finished. But if we provision the server a capability of relocation, additional information of data values will be learned, i.e., underlying relations between encrypted key-value pairs. To solve the problems, we propose a secure scaling protocol for EncKV in a minimally intrusive fashion. The idea is to copy data and index to the new nodes, and conduct the relocation during the query procedure incrementally. This treatment enables seamless query services while preserving the security as query functions.

In summary, our contributions are listed as follows:

- We design an encrypted and distributed KV store called EncKV. It supports rich query functions over encrypted data with guaranteed security, i.e., SSE's security notion [10] for exact-match queries, and ORE [13]'s security notion for range-match queries.
- We propose an enhanced ORE scheme to reduce the leakage in range-match queries. The new ORE construction conducts order comparison via token matching in a random fashion. It hides the order relations and partial information between ciphertexts.
- We provide two mechanisms for indexing: a bulk update mechanism to build indexes for a set of data records, as well as an incremental update mechanism to insert individual index entries for newly added records.
- We devise new protocols for secure system scaling. It enables data relocation smoothly without loss of data and index confidentiality when the system scales out.
- We implement our system prototype and deploy it on Amazon Web Service. The results show that it preserves linear scalability of distributed data stores with respect to their performance. The throughput of processing encrypted indexes increases linearly with the number of nodes in the cluster. By facilitating query processing in parallel with its local index framework, the query latency is reduced when more nodes are added.

The rest of this paper proceeds as follows. Section 2 reviews related work, and Section 3 introduces EncKV's architecture and threat assumptions. Our system design is presented in Section 4.1. The security analysis is conducted in Section 5, and an extensive array of evaluation results is shown in Section 6.2. Section 7 concludes the paper.

2 Related Work

Encrypted Database Systems. To enable secure rich queries over encrypted data, a line of work on encrypted database systems has been proposed [14], [15], [22], [23], [24]. The first functionally rich database system is CryptDB [14], which used deterministic encryption (DET) and order-preserving encryption (OPE) schemes to support SQL queries over encrypted relational databases. However, recent inference attacks [17] showed that CryptDB would be vulnerable to frequency analysis due to the leakage revealed in the ciphertexts of DET and OPE.

BlindSeer [15] supported secure rich queries by using a Bloom filter tree as the index, which is then searched via the evaluation on Yao's garbled circuits. Arx [24] devised a range query protocol based on a tree-based data structure. It used chained garbled circuits to compare the encrypted values with tree (range index) nodes. Since the circuits cannot be reused for security, the nodes of the tree should be re-garbled after each range query. A recent design called Seabed [25] was proposed to support data analytics over encrypted datasets. It introduced a customized schema to partition sensitive columns into multiple columns to defend against frequency attacks. But this customization incurs large storage overhead, up to 10× as reported. Other encrypted databases [26], [27], [28], [29] using trusted hardware aimed to support full functionality. But their security assumption relies on trusted hardware at the server side.

The systems discussed above were not explicitly optimized for distributed data stores. In [16], an encrypted distributed key-value store was designed with a secure multi-data model to support and secure distributed query enabled. Yet, this initial work only introduced a blueprint of the local index framework for practical query performance. Here, we employ this index framework, and carefully design and adapt the encrypted exact-match and range-match indexes to this framework to enable secure, efficient, and different types of queries over encrypted data records.

Searchable Symmetric Encryption. Another line of related works [8], [9], [10], [30] targeted on cryptographic primitive for encrypted keyword search, i.e., searchable symmetric encryption (SSE). Curtmola et al. [8] formalized the security notions of SSE and presented the first constructions against non-adaptive and adaptive chosen-keyword attacks. After that, Kamara et al. [9] formalized the notion of dynamic SSE. In [10], Cash et al. implemented a dynamic SSE scheme for large-scale databases by considering I/O efficiency. Recent efforts on SSE focus on improving the security of dynamic SSE, aka achieving forward privacy [31], [32] and backward privacy [33], [34]. As mentioned, those primitives do not consider the deployment in real-world systems, and they normally assume a centralized setting.

Order-Revealing Encryption. Order-revealing encryption (ORE) [13], [18], [21] is symmetric encryption that allows public comparison on ciphertexts for secure range queries. The initial result, also known as order-preserving encryption (OPE) [11], only supports simple numerical comparison. To improve security of OPE, new schemes were proposed [35], [36], [37]. But the tradeoff was that multiple rounds of interaction were introduced, i.e., $O(\log n)$, n is the number of indexed values. The first practical ORE scheme was introduced by Chenette et al. in [18], while it leaked the location of the first different bits. Very recently, Lewi et al. [13] presented a new ORE construction. The comparison result only revealed the location of the first different block of two ciphertexts rather than a bit. In a concurrent and independent work, Cash et al. [21] provided another ORE construction based on bilinear pairings.

Differences from Conference Version. Portions of the work presented in this paper have previously appeared in [1]. The primary improvements are summarized as follows: First, we enhance our ORE scheme proposed in the conference version to further reduce the leakage in range queries. Such improvement effectively mitigates the existing attacks on ORE. Second, we detail the protocol for record insertion, whereas only the sketch of this operation was shown in the conference version. Third, we design new protocols for secure system scaling that fit within the practicality realm. Fourth, we redo all the experiments, extend the performance evaluation, as well as performing query performance comparison with the plaintext system and prior works.

3 Overview

3.1 System Architecture

Fig. 1 illustrates our system architecture. It consists of two entities: a trusted *client* and a cluster of *server nodes* deployed in the environments such as cloud or off-premise data centers. In the present embodiment, EncKV is particularly suitable for clients which store their sensitive data records in distributed data stores. The client encrypts data records and outsources the encrypted datasets to the server nodes. Later, it can retrieve data from the nodes based on the query tokens. The nodes are adapted to provide the storage and the computation service. Each node processes query requests from the client and utilizes the APIs of the underlying KV stores to perform put/get operations.

EncKV uses standard symmetric encryption and secure cryptographic hash functions to build encrypted label-value

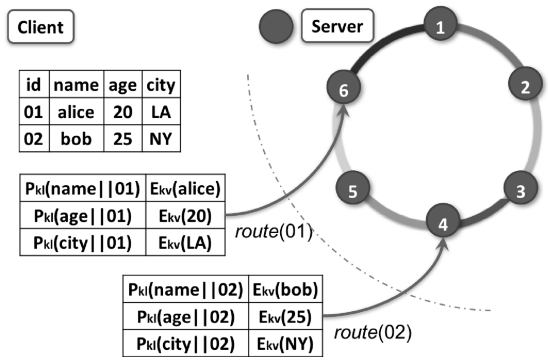


Fig. 1. The architecture of EncKV.

(LV) pair(s).[1] It formulates an extensible abstraction, which maps the data formatted from different data models to encrypted key-value pairs such that both data values and their inherent relationships are strongly protected. As a result, EncKV can distribute these encrypted pairs across the nodes by using the standard data partition algorithm [3].

To enable secure rich queries based on secondary attributes of data, EncKV leverages a framework for distributed local indexes. It requires the client to maintain a small-sized consistent hashing ring to track the label range associated with each node. Therefore, the corresponding encrypted indexes can be inserted to targeted nodes. To retrieve data records via secondary attributes, the client first generates query tokens from the query condition attribute, and then broadcasts the tokens to each node respectively. When the node receives the tokens from the client, it performs token matching over its local index. Once a matched record is found, it returns the encrypted record IDs to the client. Finally, the client decrypts the record IDs and generates labels to fetch the encrypted result values.

3.2 Threat Assumption

EncKV considers the threats from semi-honest (aka honest-but-curious) adversaries, who faithfully follows the prescribed protocols but may intend to learn the information of data and queries. The attackers cannot access the private keys stored at the client, but they could dump the entire contents of the datasets from the server nodes. They could also learn about the query tokens, accessed index entries, and the matched data records. Currently, EncKV does not consider that attackers can learn the background information about the datasets and queries, e.g., the partial (entire) content of queries or the query distribution [17], [38], [39]. Nevertheless, we make discussions on how to leverage off-the-shelf techniques to mitigate those threats later in Section 4.4. In addition, EncKV does not consider actively malicious attackers (e.g., mentioned in [40]), which can be addressed by orthogonal studies [31], [41].

3.3 Cryptographic Primitives

A symmetric encryption scheme is a set of three polynomial time algorithms $\Pi = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$: The key generation algorithm KGen takes a security parameter k as input and outputs a secret key K; The encryption algorithm Enc takes

1. We use the term "label" instead of "key" to avoid ambiguity.

a key K and a value $v \in \{0,1\}^*$ as inputs and outputs a ciphertext $v^* \in \{0,1\}^*$; The decryption algorithm Dec takes a key K and a ciphertext v^* as inputs and returns v.

Define pseudo-random function $F : \mathcal{K} \times X \rightarrow R$, if for all probabilistic polynomial-time distinguishers Y, $|Pr[Y^{F(k,\cdot)} = 1|k \leftarrow \mathcal{K}] - Pr[Y^g = 1|g \leftarrow \{\mathsf{Func} : X \rightarrow R\}]| < negl(k)$, where $negl(k)$ is a negligible function in k.

Define pseudo-random permutation $F : \mathcal{K} \times X \rightarrow X$, if for all $k \in \mathcal{K}$, $F(k,\cdot)$ is a permutation on X and no efficient distinguishers Y can distinguish the outputs of $F(k,\cdot)$ from the outputs of $\pi(\cdot)$, where π is a random permutation on X.

4 The EncKV Design

In this section, we present the designs of EncKV's encrypted indexes for secure exact-match and range-match queries, as well as the corresponding query protocols following the index constructions. Features such as data relocation, incremental updates, and batch queries are also introduced for practical and security considerations.

4.1 The Underlying Encrypted KV Store

Our system is built on top of a recently proposed encrypted KV store [16]. This prior work has two features. First, it provides a solution that securely partitions the encrypted data and distributes them across multiple nodes to preserve high-performance and linear scalability. Second, it devises a framework for encrypted and distributed indexes that support secure and efficient queries on secondary attributes of data records. By leveraging its design philosophy, we carefully integrate EncKV's index design with this framework to support secure rich queries.

Fig. 1 illustrates how a column-oriented data set is inserted into EncKV. Note that other data models such as documents and graphs are also supported as demonstrated in [16]. Specifically, for each KV pair (l, v) with label l and value v, EncKV protects it as a pseudo-random label and an encrypted value:$\langle l, v\rangle = \langle P(k_l, C||R), E(k_v, v)\rangle$, where P is a secure PRF, k_l, k_v are private keys, R is the record ID (primary key), C is a column (secondary) attribute, v is a value on C, and E is a symmetric encryption algorithm.

To preserve the data locality during the queries, EncKV uses the unique record ID R as the label for partition. This approach allows the encrypted values for a given record are stored at the same node, while protecting both the schema and value relations of each record. Note that the record IDs can be stored at either the client or server nodes in ciphertexts for system scaling.

4.2 Exact-Match Index and Query Protocol

4.2.1 Encrypted Index Design

The design of exact-match indexes is inspired by a recently SSE scheme proposed in [10]. The idea of this scheme is to use encrypted keyword-document pairs to index documents matching the same keyword. EncKV adopts this idea and indexes the record IDs that match the same values on a certain attribute.

Algorithm 1 presents the exact-match index building procedure. For each value v_j on column C_v, the client first initializes n counters, where n is the number of nodes. Given the value v_j's record ID R, the client finds the node location on the consistent hashing ring. Then it generates two tokens $t_1 = G1(k_e, C_v||v_j||i)$, $t_2 = G2(k_e, C_v||v_j||i)$ that protect the values v_j via PRF. After that, it builds the encrypted index $\langle \alpha = H1(t_1, c_i^j), \beta = H2(t_2, c_i^j) \oplus Enc(k_R, R)\rangle$ that securely indexes R, where c_i^j is the index counter to differentiate records with the same values on C. The exact-match index holds the security of SSE. Without querying, no information of the index (except the size) is known. Note that the counters will not be used in the later query protocol, and thus they can be dropped if no records will further be inserted.

Algorithm 1. $\mathsf{Build}_{\mathsf{ext}}$: Build Exact-Match Indexes

Input: Private key k_e; values $\{v_1, \ldots, v_m\}$ on attribute C_v; secure PRFs $\{H1, H2, G1, G2\}$.
Output: Encrypted indexes $\{I_1^{ext}, \ldots, I_n^{ext}\}$.

```
Initialize a hash table S to maintain counters;
for v_j ∈ {v_1, ..., v_m} do
    i ← route(R); // R is v_j's ID, i ∈ {1, n} is node ID
    t_1 ← G1(k_e, C_v||v_j||i);
    t_2 ← G2(k_e, C_v||v_j||i);
    if S.find(i||j) = ⊥ then
        c_i^j ← 0;
    else
        c_i^j ← S.find(i||j);
    end if
    α ← H1(t_1, c_i^j);
    β ← H2(t_2, c_i^j) ⊕ Enc(k_R, R);
    c_i^j ++;
    S.put(i||j, c_i^j);
    I_i^ext.put(α, β);
end for
```

4.2.2 Secure Query Protocol

Based on the construction of the exact-match index, we present the corresponding query protocol in Algorithm 2. Given a query with two attributes, the client wants to find all the values $\{v_r\}$ in attribute C_r on a matching condition such that another attribute C_v's value is equal to v. First, the client generates query tokens $t_1 = G1(k_e, C_v||v||i)$, $t_2 = G2(k_e, C_v||v||i)$, where i from 1 to n, and broadcasts $\{t_1, t_2\}$ to each node. After receiving the tokens, each node processes the requests in parallel. Specifically, the node retrieves the matched index entries via computing $H1(t_1, c_i)$, where c_i is a self-incremental counter. Then it unmasks the entry via XORing $H2(t_2, c_i)$ and returns the encrypted ID set $\{r\}$ to the client for decryption. Finally, the client generates the corresponding label $P(k_l, C_r||R)$ one by one and fetches the encrypted result values from the KV store.

The proposed query protocol provides strong protection for data values and attribute associations. During a query procedure, the node only sees the access pattern, i.e., the accessed index entries, return encrypted values, and query tokens. It also learns the query pattern, i.e., the repeated queries, because tokens are deterministic. Recall that EncKV's query protocol requires two rounds of interaction. The first is to obtain the encrypted record IDs, and the second round is to fetch the matched results. This treatment leads to an immediate security improvement, hiding the relations between data values on different attributes. Each node only learns the

matched values associated with the same attribute. It will reveal neither the relations between values in different attributes, nor the relations between different values in same attributes. Therefore, it effectively addresses the inference attacks [17]. More detailed analysis can be found in Section 5.1. Regarding the query performance, the query time complexity is linear in the number of matched result values, where the complexity of each match is $O(1)$.

Algorithm 2. $\mathsf{Query_{ext}}$: Secure Exact-Match Query Protocol

Input: Private key k_e; result value attribute C_r; query condition value v on attribute C_v.
Output: Encrypted matched results $\{v_r\}$.

Client.Token
1: **for** $i \in \{1, \ldots, n\}$ **do**
2: $t_1 \leftarrow G1(k_e, C_v||v||i)$;
3: $t_2 \leftarrow G2(k_e, C_v||v||i)$;
4: Send (t_1, t_2) to node i;
5: **end for**

$Node_i$.ExtQuery
1: $c_i \leftarrow 0$;
2: $\alpha \leftarrow H1(t_1, c_i)$;
3: **while** $find(\alpha) \neq \perp$ **do**
4: $\beta \leftarrow find(\alpha)$;
5: $r \leftarrow I_i^{ext}.get(H2(t_2, c_i) \oplus \beta)$;
6: c_i++;
7: $\alpha \leftarrow H1(t_1, c_i)$;
8: Return r to client for decryption;
Client
9: $R \leftarrow Dec(k_R, r)$;
10: $l \leftarrow P(k_l, C_r||R)$;
11: Fetch v_r via l;
12: **end while**
13: // Note: in the implementation, all matched $\{r\}$ are sent back in a batch, and $\{v_r\}$ are fetched in a batch.

4.3 Range-Match Index and Query Protocol

4.3.1 Design Rationale

In current literature, a strong candidate for secure range query is order-revealing encryption (ORE), which is first introduced by Boneh et al. [42]. ORE has two advantages while OPE does not have. First, an ORE scheme allows a publicly computable function to compare two ciphertexts, which does not restrict the structure of the ciphertext space. Second, attackers who can only access the ORE ciphertext will not derive any useful information due to its semantic security. Recently, Lewi et al. [13] proposed the first practical ORE scheme that achieves a balance on security and efficiency. The core idea of their scheme is to split a message into bit blocks with equal length, and conduct encrypted comparison from the first blocks of two messages. However, their scheme is generic and reveals the order relations of compared ciphertexts. In addition, the index of the first bit block that differs is revealed between two ciphertexts. Recent leakage-abuse attacks [12], [20] show that the above information could be exploited to recover the underlying values of the ciphertexts.

For instance, let the block size is 2 bits, and one compares a message token $m_1 =$ ''1011'' with two ORE ciphertexts $m_2 =$ ''1000'' and $m_3 =$ ''0100'' respectively. Under the leakage profile of [13], the leakage on (m_1, m_2) is $m_1 > m_2$ and the index of the first different block is 2, i.e., the 2nd block. On the comparison of m_1 and m_3, the leakage is $m_1 > m_3$ and the index of the first different block is 1. As a result, the attacker can infer that $m_2 > m_3$ even without comparison based on the above information.

Motivated by the observations above, our goal is to design an ORE scheme that introduces less leakage compared to the existing ones [1], [13], [18]. To this end, our design intuition lies in the following two aspects. First, we aim to hide the order relations in queries and results. This property is achieved in our conference version by tokenizing the orders in ORE ciphertexts. Specifically, the messages are encrypted into ciphertext blocks, where each block embeds its value's attribute, sub index values, and tokens of the order relations. As a result, the servers learn neither the order of underlying ORE ciphertexts, nor whether two queries are conducted in the same order condition if the query attributes are different.

Algorithm 3. OREtoken: ORE Token Generation

Input: Private key k_o; value v on attribute C_v; order cmp; secure PRFs $\{F1, F2, F3\}$; secure PRP π; random PRP ϕ.
Output: ORE query token ct_L.
1: Derive k_1, k_2, k_3 from k_o;
2: **for** $i \in \{1, b\}$ **do**
3: $j^* \leftarrow \pi(F2(k_2, v_{|i-1}), v_{|i})$;
4: $q_i \leftarrow F3(k_3, cmp||C_v||j^*)$;
5: $ct_{L|i} \leftarrow F1(k_1, v_{|i-1}||j^*), q_i$;
6: **end for**
7: $ct_L \leftarrow \{ct_{L|\phi(1)}, \ldots, ct_{L|\phi(b)}\}$;

Algorithm 4. OREenc: ORE Ciphertext Encryption

Input: Private key k_o; value v on attribute C_v; secure PRFs $\{F1, F2, F3, Q\}$; secure PRP π; random PRP φ.
Output: ORE ciphertext ct_R.
1: Derive k_1, k_2, k_3 from k_o;
2: Generate a nonce γ;
3: **for** $i \in \{1, b\}$ **do**
4: **for** $j \in \{1, 2^d\}$ **do**
5: $\tilde{v_{|i}} \leftarrow \pi^{-1}(F2(k_2, v_{|i-1}), j)$;
6: **if** $\tilde{v_{|i}} \neq v_{|i}$ **then**
7: $s_{i,j} \leftarrow F3(k_3, cmp(\tilde{v_{|i}}, v_{|i})||C_v||j)$;
8: $z_{i,j} \leftarrow s_{i,j} + Q(F1(k_1, v_{|i-1}||j), \gamma)$;
9: **end if**
10: **end for**
11: $ct_{R|i} \leftarrow z_{i,\varphi(1)}, \ldots, z_{i,\varphi(2^d-1)}$;
12: **end for**
13: $ct_R \leftarrow \{ct_{R|\varphi(1)}, \ldots, ct_{R|\varphi(b)}\}, \gamma$;

Second, we aim to hide the index of the first different blocks of two ORE ciphertexts. Inspired by Cash et al.'s scheme in [21], this can be achieved by random permutation. However, their construction is built on bilinear mapping, which is completely different from our current design. How to leverage random permutation on block ciphertexts poses two challenges. The first challenge is to preserve the correctness of the ORE comparison. The original design requires the comparison to be conducted blocks by blocks, and simply permuting those blocks could cause mismatches. To solve this

	ct_L(00 10)	
block	$ct_{L\|1}$(00)	$ct_{L\|2}$(10)
PRP	$\pi(F2_{k2}(null),00)\rightarrow 1$	$\pi(F2_{k2}(00),10)\rightarrow 2$
tokens	$F3_{k3}$("<"‖age‖1); $F1_{k1}$(null‖1)	$F3_{k3}$("<"‖age‖2); $F1_{k1}$(00‖2)
$\phi_1(ct_L)$= {$ct_{L\|2}$(10), $ct_{L\|1}$(00)}		

	ct_L(10 10)	
block	$ct_{L\|1}$(10)	$ct_{L\|2}$(10)
PRP	$\pi(F2_{k2}(null),10)\rightarrow 4$	$\pi(F2_{k2}(10),10)\rightarrow 4$
tokens	$F3_{k3}$("<"‖age‖4); $F1_{k1}$(null‖4)	$F3_{k3}$("<"‖age‖4); $F1_{k1}$(10‖4)
$\phi_2(ct_L)$= {$ct_{L\|1}$(10), $ct_{L\|2}$(10)}		

(a) ORE token ct_L

ct_R(10 11)	
$ct_{R\|1}$(10)	$ct_{R\|2}$(11)
$F3_{k3}$(11>10‖age‖2)+Q[$F1_{k1}$(null‖2),r_1]	$F3_{k3}$(01<11‖age‖1)+Q[$F1_{k1}$(10‖1),r_1]
$F3_{k3}$(00<10‖age‖1)+Q[$F1_{k1}$(null‖1),r_1]	$F3_{k3}$(00<11‖age‖3)+Q[$F1_{k1}$(10‖3),r_1]
$F3_{k3}$(01<10‖age‖3)+Q[$F1_{k1}$(null‖3),r_1]	$F3_{k3}$(10<11‖age‖4)+Q[$F1_{k1}$(10‖4),r_1]
$\varphi_1(ct_R)$= {$ct_{R\|2}$(11), $ct_{R\|1}$(10)}	

ct_R(01 01)	
$ct_{R\|1}$(01)	$ct_{R\|2}$(01)
$F3_{k3}$(00<01‖age‖1)+Q[$F1_{k1}$(null‖1),r_2]	$F3_{k3}$(00<01‖age‖2)+Q[$F1_{k1}$(01‖2),r_2]
$F3_{k3}$(10>01‖age‖4)+Q[$F1_{k1}$(null‖4),r_2]	$F3_{k3}$(10>01‖age‖1)+Q[$F1_{k1}$(01‖1),r_2]
$F3_{k3}$(11>01‖age‖2)+Q[$F1_{k1}$(null‖2),r_2]	$F3_{k3}$(11>01‖age‖4)+Q[$F1_{k1}$(01‖4),r_2]
$\varphi_2(ct_R)$= {$ct_{R\|1}$(01), $ct_{R\|2}$(01)}	

(b) ORE ciphertext ct_R

Fig. 2. The ORE comparison algorithm. (1) Case 1 (red highlight): Both ORE ciphertexts $ct_R(1011)$ and $ct_R(0101)$ match the token block $ct_{L|1}(00)$ of the query token $ct_L(0010)$. (2) Case 2 (blue highlight): Only the ORE ciphertext $ct_R(1011)$ matches the token block $ct_{L|2}(10)$ of the query token $ct_L(1010)$. Without background knowledge about the query condition, the attackers cannot infer the order relation between $ct_R(1011)$ and $ct_R(0101)$ even after multiple comparisons.

problem, our observation is that there exists one and only one sub block matched during the comparison. Therefore, we propose to embed the hash value of each block's entire prefix into the block ciphertext. Note that the original scheme embeds the previous block into the ciphertext. Due to the uniqueness of the prefix in each block, the token matching operation can still correctly be performed even blocks are shuffled, as illustrated in Fig. 2.

The second challenge is to ensure the security of the ORE comparison. Note that straightforwardly applying secure permutation on the encrypted blocks still reveals the block equality in each comparison. Then the attacker can find out the index of the first differing block by counting how many matched blocks have in common. To reduce this leakage, we remove the equality information in each block and replace it with a dummy value. Therefore, the server cannot match the "equal" value during the comparison.

4.3.2 Enhanced ORE Construction

We define $[v]$ as the message space. Let $F : \{0,1\}^{\lambda} \times [v] \rightarrow \{0,1\}^{\lambda}$ be a secure PRF, the length of data value $|v| > 0$, and integers $b, d > 0$ such that $b \times d = |v|$. Let $\pi : \{0,1\}^{\lambda} \times [d] \rightarrow [d]$ be a secure PRP. Given a $|v|$-bit string, let $v_{|i}$ denotes the ith block of the value v, and $v_{|i-1}$ represents its entire prefix. We use $''||''$ to denote the concatenation. Our ORE scheme $\Pi = \{\mathsf{OREtoken}, \mathsf{OREenc}, \mathsf{OREcmp}\}$ is defined as follows:

- $\mathsf{OREtoken}(\mathsf{k_o}, \mathsf{v}, \mathsf{cmp})$: Given a data value v on attribute C_v and order condition $\mathsf{cmp} \in \{>, <\}$, the algorithm first splits the value v into b blocks with length of d bits. For each block $i \in [1, b]$, it computes the sub index value $j^* \in \{1, 2^d\}$ via $\pi(F2(k_2, v_{|i-1}), v_{|i})$, and generates the encrypted query condition as $q_i = F3(k_3, cmp||C_v||j^*)$. Then it sets $ct_{L|i} = \{F1(k_1, v_{|i-1}||j^*), q_i\}$. Finally, the algoritm applies a random PRP ϕ on $\{ct_{L|1}, \ldots, ct_{L|b}\}$, and outputs the ORE query token ct_L.
- $\mathsf{OREenc}(\mathsf{k_o}, \mathsf{v})$: First, the algorithm uniformly chooses a nonce γ for value v. Then for each block $i \in [1, b]$, the algorithm computes all possible block value $\tilde{v_{|i}} = \pi^{-1}(F2(k_2, v_{|i-1}), j)$, and compares it with the current block $v_{|i}$. If the comparison result $cmp(\tilde{v_{|i}}, v_{|i})$ is not $''equal''$, then the algorithm generates the encrypted entry as $z_{i,j} = s_{i,j} + Q(F1(k_1, v_{|i-1}||j), \gamma)$, where the protected order is $s_{i,j} = F3(k_3, cmp(\tilde{v_{|i}}, v_{|i})||C_v||j)$. Finally, the algorithm permutes all the encrypted blocks via random PRP φ, and outputs the ORE ciphertext ct_R.
- $\mathsf{OREcmp}(\mathsf{ct_L}, \mathsf{ct_R})$: On input the query token $ct_L = \{ct_{L|1}, \ldots, ct_{L|b}\}$ and the ORE ciphertext $ct_R = \{ct_{R|1}, \ldots, ct_{R|b}, \gamma\}$, the algorithm finds the specific block pair $(ct_{L|i}, ct_{R|i'})$ that matches the query condition q_i, where $i, i' \in [1, b]$. Symmetric to the block encryption, the encrypted order is obtained via $s_{\tilde{i}} = z_{i',\tilde{i}} - Q(x_i, \gamma)$, where x_i is the query token of corresponding block $ct_{L|i}$, $z_{i',\tilde{i}}$ is the block of $ct_{R|i'}$, and γ is the nonce of ct_R ciphertext. If no such block pair exists, output $false$. Otherwise, output $true$.

Algorithm 5. OREcmp: ORE Ciphertext Comparison

Input: ORE query token ct_L; ORE ciphertext ct_R;
Output: *true* or *false*.

```
ct_{L|1}, ..., ct_{L|b} ← ct_L;
ct_{R|1}, ..., ct_{R|b}, γ ← ct_R;
for i ∈ {1, ..., b} do
    x_i, q_i ← ct_{L|i};
    for i' ∈ {1, ..., b} do
        z_{i',1}, ..., z_{i',2^d-1} ← ct_{R|i'};
        for ĩ ∈ {1, ..., 2^d − 1} do
            s_ĩ ← z_{i',ĩ} − Q(x_i, γ);
            if s_ĩ = q_i then
                return true; // condition matched
            end if
        end for
    end for
end for
return false;
```

Our enhanced ORE construction leaks only the equality pattern of the most significant differing block, not the original location of these blocks. As an example, if $m_1 > m_2$ and $m_1 > m_3$, and same blocks are matched in two comparisons, the server only knows m_2 and m_3 matches the same order condition (not the order) of m_1, and the matches appear in the same blocks. If different blocks are matched in two comparisons, the server will never know the orders of m_2 and m_3, because blocks are shuffled. We are aware that the equality information would be revealed after all sub blocks are

matched just like all existing ORE schemes. To address this issue, one straightforward solution is to re-encrypt the matched sub block, as proposed in [24], [33].

Regarding the comparison complexity, the worst-case scenario of block comparison is b^2 operations. For each sub block, the worst-case scenario is $2^d - 1$ operations, because the equality entry is removed from the result set, the number of encrypted entry is $2^d - 1$ instead of 2^d. We emphasize that all above algorithms are conducted by symmetric key based operations. The experiments later show that our enhanced ORE scheme is still scalable for large-scale applications.

4.3.3 Encrypted Index Design

The construction of encrypted range-match indexes follows the same treatment as the exact-matched indexes. The detailed procedure that indexes values $\{v_1, \ldots, v_m\}$ for a given attribute C_v is shown in Algorithm 6. Given the record ID R, the client first finds the target node i where the record is stored. Then it builds the encrypted index entry $\langle \alpha, \beta \rangle$ by securely embedding attribute C_v, node ID i and the counter c_i. Note that the underlying content of β also contains the ORE ciphertext ct_R which is generated from the ORE encryption scheme $\mathsf{OREenc}(k_o, v)$ in Algorithm 4.

Algorithm 6. $\mathsf{Build}_{\mathsf{rng}}$: Build Range-Match Indexes

Input: Private keys k_r, k_o; values $\{v_1, \ldots, v_m\}$ on attribute C_v; secure PRFs $\{H1, H3, G1, G2\}$.
Output: Encrypted indexes $\{I_1^{rng}, \ldots, I_n^{rng}\}$.

Initialize a hash table S to maintain counters;
for $v_j \in \{v_1, \ldots, v_m\}$ **do**
  $i \leftarrow route(R)$; // R is v_j's ID, $i \in \{1, n\}$ is node ID
  $t_1 \leftarrow G1(k_r, C_v||i)$;
  $t_2 \leftarrow G2(k_r, C_v||i)$;
  if $S.find(i) = \perp$ **then**
    $c_i \leftarrow 0$;
  else
    $c_i \leftarrow S.find(i)$;
  end if
  $\alpha \leftarrow H1(t_1, c_i)$;
  $ct_R \leftarrow \mathsf{OREenc}(k_o, v_j)$; // shown in Algorithm 4
  $\beta \leftarrow H3(t_2, c_i) \oplus (ct_R||Enc(k_R, R))$;
  $c_i + +$;
  $S.put(i, c_i)$;
  $I_i^{rng}.put(\alpha, \beta)$;
end for

4.3.4 Secure Query Protocol

The corresponding query protocol following the range-match index construction is presented in Algorithm 7. Given two range query attributes and the order condition $cmp \in \{>, <\}$, the client asks EncKV to return all values $\{v_r\}$ in attribute C_r on the matching condition such that the query attribute C_v's value should be larger than the value v. Similar to the exact-match query protocol, the client generates query tokens $\{t_1, t_2\}$ from C_v for each node i. After that, it calls the $\mathsf{OREtoken}(k_o, v, cmp)$ function to generate ct_L as shown in Algorithm 3, and sends $\{t_1, t_2, ct_L\}$ to each node. When the query tokens arrive, each node first unmasks the corresponding ORE index entries via incremental counters. If the tokens $\{t_1, t_2\}$ correctly recover an ORE ciphertext ct_R, the server node performs the ORE comparison OREcmp (ct_L, ct_R) as presented in Algorithm 5. Once the ORE token ct_L matches the query condition, the server returns the corresponding record ID ciphertext $Enc(k_R, R)$ to the client to fetch the result values on attribute C_r. In current treatment, the query time complexity is $O(m_{C_v})$, where m_{C_v} is the number of values on C_v at a certain node.

Algorithm 7. $\mathsf{Query}_{\mathsf{rng}}$: Secure Range-Match Query Protocol

Input: Private keys k_r, k_o; result value attribute C_r; query condition value v on attribute C_v; order condition cmp.
Output: encrypted matched results $\{v_r\}$.

Client.Token
for $i \in \{1, \ldots, n\}$ **do**
  $t_1 \leftarrow G1(k_r, C_v||i)$;
  $t_2 \leftarrow G2(k_r, C_v||i)$;
  $ct_L \leftarrow \mathsf{OREtoken}(k_o, v, cmp)$;// Algorithm 3
  Send (t_1, t_2, ct_L) to node i;
end for

$Node_i$.RngQuery
$c_i \leftarrow 0$;
$\alpha \leftarrow H1(t_1, c_i)$;
while $find(\alpha) \neq \perp$ **do**
  $\beta \leftarrow find(\alpha)$;
  $r \leftarrow I_i^{rng}.get(H3(t_2, c_i) \oplus \beta)$;
  Parse r as $r_x \leftarrow Enc(k_R, R)$, $r_y \leftarrow ct_R$;
  $c_i + +$;
  if $\mathsf{OREcmp}(ct_L, ct_R) = true$ **then** // Algorithm 5
    Return r_x to the client;
  end if
  $\alpha \leftarrow H1(t_1, c_i)$;
end while
// Note: we ignore the steps to fetch final results, which is the same in Line 9 to 11 in Algorithm 2.

4.4 Batch Queries

EncKV's rich query protocols via secondary attributes are conducted in two phases. The first phase is to obtain the encrypted record IDs from the nodes if the index entries on a query attribute match the query condition. The second phase is to fetch the result values of these matched records on a targeted attribute. Such treatment will let the server nodes to know the relations between index entries of different attributes and values of the same records (see formal analysis in Section 5).

To reduce this leakage, one immediate improvement is to conduct queries in two rounds of interaction in a batched manner. We are aware that such treatment can be realized via a dedicated query planner, which is also used in [22], [24]. Based on the batch query mechanism, EncKV can further obfuscate data correlations on different attributes.

4.5 Secure Update Operations

Our system supports two types of update operations when new data records are added, i.e., the incremental update and the bulk update. The bulk update is suitable for the case when adding a large number of records to EncKV, such as

migrating an unencrypted database to EncKV. To achieve this, the client may utilize the proposed index building functions in Algorithm 1 and Algorithm 6 to generate encrypted exact-match and range-match indexes respectively.

The incremental update is suitable for the case when data records are occasionally inserted/updated into EncKV. To update a record value, the client first removes the old value and then inserts the new value to the KV store. This is done by generating the update token $l_i \leftarrow P(k_l, C_v||R)$ using the update attribute C_v and record ID R. For the index update, the procedure follows the same treatment as the record update. The obsolete index entry will be removed by fetching the matched entries on the old value v_o via token $t_o \leftarrow G1(k_e, C_v||v_o)$. After deletion, the client re-encrypts all the entries and inserts them back. For the new entry, the index entries that match the new value v will be fetched to get the maximum counter. Then the new LV pair is inserted via the incremented counter.

During the update operation, server nodes will learn the relations between newly inserted index entries and those queried attributes (also known as update leakage [9], [31], [43]). As acknowledged in prior dynamic SSE schemes [9], [10], we consider this leakage as the cost pay for enabling update operations. To improve the security, a recent SSE scheme with forward privacy [31] can readily be adapted to EncKV, since we both employ the encrypted dictionary [10] for the index construction. The idea is to use a one-way trapdoor permutation to generate the newly inserted entries so that they are unlinkable to previous query tokens. As a performance tradeoff, the update and query throughput would downgrade due to the computation of public key based trapdoor permutation, and the client needs to store some state information to generate new index entries. One recent work [33] achieves backward privacy via a heavy primitive, i.e., puncturable encryption. We will leave the design of efficient forward and backward secure searchable encryption schemes as future work.

4.6 System Scaling

Adding new node smoothly is a fundamental requirement for distribution data stores to handle the rapidly increasing data records. The native solution is to rebuild the index entries and data records for the affected nodes at the client side. However, this is not suitable for a large-scale dataset because the client needs to download all the record entries on the affected nodes before relocating them to the new one.

In this work, we propose to conduct the data relocation gradually during the queries as introduced in Algorithm 8. For each query, only the matched data records on the affected nodes will be relocated if their labels point to the newly added node. The benefits are two-fold: (1) The movement of the indexes can be achieved without interrupting the query service. (2) The security will not be affected, because the relation is still operated by the client.

4.6.1 Data Relocation

Recall that EncKV dispatches the encrypted LV pairs via the consistent hashing algorithm. Therefore, when a node is added, the client may locate its position on the updated ring along with its neighbors. Namely, the preceding neighbor node can directly move the corresponding values to the newly added node without any extra overhead. For instance, assuming a newly added node $node_j$ is assigned to locate between $node_i$ and $node_z$ where i, j and z are nodes' positions on the consistent hashing ring. First, $node_i$ copies all the encrypted pairs to $node_j$. As a result, both $node_i$ and $node_j$ contain the same encrypted dataset which records IDs $R \in [i, z)$. To eliminate the duplicated data, the client will sequentially remove them on $node_i$ during each query, as shown in Line 16 of Algorithm 8.

Algorithm 8. AddNodes: Secure Data Relocation Protocol

Input: Private key k_e; query token (t_1, t_2); query condition attribute C_v; result value attribute C_r; new node $node_j$.
Output: Encrypted data $\langle l_j, v_j \rangle$ and indexes $\{I_i^{ext}, I_i^{rng}\}$.

$Node_i.Relocation$
1: $c_i \leftarrow 0$;
2: $\alpha_i \leftarrow H1(t_1, c_i)$;
3: **while** $find(\alpha_i) \neq \perp$ **do**
4: $\beta_i \leftarrow find(\alpha_i)$;
5: **if**operation is $exact - match$ **then**
6: $r \leftarrow I_i^{ext}.get(H2(t_2, c_i) \oplus \beta_i)$;
7: **else if**operation is $range - match$ **then**
8: $r \leftarrow I_i^{rng}.get(H3(t_2, c_i) \oplus \beta_i)$;
9: **end if**
10: $c_i + +$;
11: $\alpha_i \leftarrow H1(t_1, c_i)$;
12: Return r to client for relocation;
Client
13: $R \leftarrow get(k_R, r)$;
14: $l \leftarrow P(k_l, C_r||R)$;
15: **while** route(R)=j**do** //j is new node's ID
16: $node_i.del(l)$;
17: **if**operation is $exact - match$ **then**
18: $I_i^{ext}.del(\alpha_i)$;
19: $(\alpha_j^{ext}, \beta_j^{ext}) \leftarrow \mathsf{Build_{ext}}$;//Algorithm 1
20: **else if**operation is $range - match$ **then**
21: $I_i^{rng}.del(\alpha_i)$;
22: $(\alpha_j^{rng}, \beta_j^{rng}) \leftarrow \mathsf{Build_{rng}}$;//Algorithm 6
23: **end if**
24: **end while**
25: **end while**

$Node_j.Relocation$
1: Get (l_j, v_j) from $node_i$;// data relocation from $node_i$;
2: $I_j^{ext}.put(\alpha_j^{ext}, \beta_j^{ext})$;
3: $I_j^{rng}.put(\alpha_j^{rng}, \beta_j^{rng})$;

4.6.2 Index Relocation

Regarding the encrypted index relocation, we propose to conduct the index rebuilding procedure during the queries. In particular, when the client obtains the record IDs in each query, those IDs are used to trace the locations of encrypted value via the updated consistent hashing ring. Then the client rebuilds the updated indexes, and inserts them to the newly added node as presented in Algorithm 8. Given query tokens (t_1, t_2), the $node_i$ locates all the matched index entries via $\alpha_i \leftarrow H1(t_1, c_i)$ and returns r to the client for index rebuilding via XORing operation. If the index relocation is the exact-match index, r is the encrypted record ID R. Otherwise, it contains both the encrypted record ID R and ORE ciphertext ct_R for range-match index. Then, the

$node_i$ deletes the corresponding index entries if the hash value of the record ID indicates the newly added node. After that, the client rebuilds the index entries based on the record ID R (and ct_R for range-match index), and inserts them to the newly added node $node_j$ directly via the consistent hashing ring.

5 Security Analysis

In this section, we perform a formal security analysis for EncKV. EncKV randomly maps data records to encrypted LV pairs to hide the relations of the underlying ciphertexts, and each data value is encrypted via symmetric encryption with semantic security. Therefore, it can effectively defend against inference attacks [17], which rely on statistic information and relations of encrypted data values.

On the other hand, we evaluate the security of secure exact-match queries and range-match queries respectively. We first define the leakage in EncKV's query protocols, and quantify its security guarantees following the primitives we adopted, i.e., SSE [10] and ORE [13] respectively. In addition, we will discuss our enhanced ORE scheme on the protection against a series of recent attacks using range query leakage.

5.1 Security on Exact-Match Queries

The exact-match index design is built on the framework of SSE [8], which provisions the nodes a controlled capability to return the encrypted results. Once the client uploads the encrypted index to the server node, the index size will be learned. During the query procedure, the access pattern and search pattern will be revealed, where access pattern indicates the accessed entries, and search pattern is the repeated query tokens. Since our query protocol includes multiple column attributes, the access pattern also contains the associations between those columns. Following the security notion of SSE, we first define the leakage functions in EncKV's exact-match queries as follows:

$$\mathcal{L}_1^{ext}(\mathbf{C}) = (\{m_i\}_n, \langle|\alpha|, |\beta|\rangle),$$

where $\mathbf{C}$ is the secondary attribute set, n is the number of nodes, m_i is the size of exact-match index I_i^{ext} of node i, and $|\alpha|, |\beta|$ are the index lengths of label and value.

$$\mathcal{L}_2^{ext}(v_C, C_v, C_r) = (\{t_1^i, t_2^i\}_n, \{\{\langle\alpha, \beta\rangle, \langle l, v^*\rangle\}_{c_i}\}_n),$$

where v_C is the value of query condition, C_v is the column attribute of v_C and C_r is the result value's attribute. $\{t_1^i, t_2^i\}_n$ are query tokens for n nodes, and $\{\langle\alpha, \beta\rangle, \langle l, v^*\rangle\}_{c_i}$ are matched index entries and result set at each node.

$$\mathcal{L}_3^{ext}(\mathbf{Q}) = (M_{q\times q}, T_{v^*\to\alpha}),$$

where $\mathbf{Q}$ is q number of adaptive queries. $M_{q\times q}$ is a symmetric bit matrix that traces the repeated queries. For $i, j \in [1, q]$, the matrix element $M_{i,j}$ and $M_{j,i}$ are equal to 1 if the tokens $t_1^i = t_1^j$. Otherwise, they are equal to 0. $T_{v^*\to\alpha}$ is an inverted list to trace the accessed index entries [17]. For each posting list $v^*|\{\alpha_1, \ldots, \alpha_a\}$ in T, the associations between the matched index entries and data values on different attributes are also learned. Following the simulation-based security definition [8], [9], we give the formal security definition as follows:

Definition 1. *Let* $\mathsf{Ext} = (\mathsf{KGen}, \mathsf{Build}_{\mathsf{ext}}, \mathsf{Query}_{\mathsf{ext}})$ *be our scheme for secure exact-match query, and let* $\mathcal{L}_1^{ext}$, $\mathcal{L}_2^{ext}$ *and* $\mathcal{L}_3^{ext}$ *be the leakage functions. Given a probabilistic polynomial time (PPT) adversary* $\mathcal{A}$ *and a PPT simulator* $\mathcal{S}$*, define the following probabilistic games* $\mathbf{Real}_{\mathcal{A}}(k)$ *and* $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(k)$*:*

$\mathbf{Real}_{\mathcal{A}}(k)$*: The client calls* $\mathsf{KGen}(1^{\mathsf{k}})$ *to get a private key* K*.* $\mathcal{A}$ *selects a dataset* $\mathbf{D}$ *and asks the client to build* $\{I_1^{ext}, \ldots, I_n^{ext}\}$ *via* $\mathsf{Build}_{\mathsf{ext}}$*. Then* $\mathcal{A}$ *adaptively conducts a polynomial number of* q *queries with the tokens and ciphertexts generated from the client. Finally,* $\mathcal{A}$ *returns a bit as the output.*

$\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(k)$*:* $\mathcal{A}$ *selects* $\mathbf{D}$*, and* $\mathcal{S}$ *builds* $\{I_1'^{ext}, \ldots, I_n'^{ext}\}$ *for* $\mathcal{A}$ *based on* $\mathcal{L}_1^{ext}$*. Then* $\mathcal{A}$ *adaptively performs a polynomial number of* q *queries. From* $\mathcal{L}_2^{ext}$ *and* $\mathcal{L}_3^{ext}$ *in each query,* $\mathcal{S}$ *generates the simulated tokens and ciphertexts, which are processed over* $\{I_1'^{ext}, \ldots, I_n'^{ext}\}$*. Finally,* $\mathcal{A}$ *returns a bit as the output.*

Ext *is adaptively secure with* $(\mathcal{L}_1^{ext}, \mathcal{L}_2^{ext}, \mathcal{L}_3^{ext})$ *if for all PPT adversaries* $\mathcal{A}$*, there exists a PPT simulator* $\mathcal{S}$ *such that:* $Pr[\mathbf{Real}_{\mathcal{A}}(k) = 1] - Pr[\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(k) = 1] \leq negl(k)$*, where* $negl(k)$ *is a negligible function in* k*.*

Theorem 1. Ext *is adaptively secure with* $(\mathcal{L}_1^{ext}, \mathcal{L}_2^{ext}, \mathcal{L}_3^{ext})$ *leakages under the random-oracle model if* $G1, G2, H1, H2$ *and* P *are secure PRF.*

Proof. The objective is to prove that the adversary $\mathcal{A}$ cannot distinguish between the real index and the simulated one as defined in Definition 1. We first define random oracles $\{\mathcal{H}_{G1}, \mathcal{H}_{G2}, \mathcal{H}_{H1}, \mathcal{H}_{H2}, \mathcal{H}_P\}$.

From $\mathcal{L}_1^{ext}$, the simulator $\mathcal{S}$ simulates the encrypted exact-match indexes $\{I_1'^{ext}, \ldots, I_n'^{ext}\}$ for n nodes, which have the same size as the real encrypted indexes. It contains m_i index entries, where each entry uses $|\alpha|$-bit and $|\beta|$-bit random string as a label-value pair.

When the first query sample (v_C, C_v, C_r) is sent to node i, $\mathcal{S}$ generates $t_1' = \mathcal{H}_{G1}(k_e'||C_v||v_C||i)$ and $t_2' = \mathcal{H}_{G2}(k_e'||C_v||v_C||i)$ as simulated tokens, where k_e' is a random string. After that, a random oracle $\mathcal{H}_{H1}$ is operated in the way of $\alpha' = \mathcal{H}_{H1}(t_1', c_i)$ from 1 to c_i to find the matched entries, where c_i is the number of matched index from L_2^{ext}. For each accessed entries, another random oracle $\mathcal{H}_{H2}$ is operated to obtain R'^* inside via computing $R'^* = \mathcal{H}_{H2}(t_2', c_i) \oplus \beta'$. R'^* can be derived from $(\lambda, \mathcal{H}_R(k_R'||\lambda) \oplus R)$, where $\mathcal{H}_R$ is a random oracle, k_R' and λ are random strings, and R is the record ID. Then, with a random string k_l', $\mathcal{S}$ simulates $l' = \mathcal{H}_P(k_l'||C_r||R)$, and generates random strings $v^{*'}$ as the simulated value with the same length to the real one. And from L_3^{ext}, $\mathcal{S}$ updates $M_{1,1}' = 1$ in a matrix $M_{q\times q}'$, and inserts $v^{*'}|\alpha'$ for each $v^{*'}$ in the inverted list $T'_{v^{*'}\to\alpha'}$.

In the subsequent jth queries ($j \in \{2, q\}$), if the query appears repeatedly, $\mathcal{S}$ will choose the same tokens simulated before, and return the repeated matching results. Meanwhile, it will update the corresponding element in $M_{1,j}'$ and $M_{j,1}'$ to be "1". Otherwise, $\mathcal{S}$ will generate simulate tokens and operate random oracle to get the results as shown in the first query procedure. Note that $T_{v^*\to\alpha}$ from L_3^{ext} traces the repeated results queried from different attributes, therefore all $\langle l', v^{*'}\rangle$ appeared before can be copied from $T'_{v^{*'}\to\alpha'}$.

Due to the semantic security of secure PRF, $\mathcal{A}$ cannot differentiate the simulated tokens and results from the real tokens and results. □

5.2 Security on Range-Match Queries

The range-match index is built on the ORE scheme proposed in [13], which achieves semantic security. Further, we leverage random permutation and secure PRF to enhance the leakage profile of ORE ciphertexts. The resulting improvement is that the order result and the position of the first different block are protected. To integrate the ORE ciphertext into the local index framework, we use SSE scheme as an overlay to protect ORE ciphertexts in the encrypted indexes. In terms of quantified leakage, we provide the security definition as follows:

$$\mathcal{L}_1^{rng}(\mathbf{C}) = (\{m_i\}_n, \langle|\alpha|, |\beta|\rangle).$$

Where $\mathbf{C}$ is the secondary attribute set, n is the number of data nodes and m_i is the size of range index I_i^{rng} at node i. Given an index entry, $|\alpha|, |\beta|$ are the bit lengths of the encrypted label and value.

$$\mathcal{L}_2^{rng}(v_C, C_v, C_r) = (\{t_1^i, t_2^i\}_n, ct_L, \{\{\langle\alpha, \beta\rangle, \langle l, v^*\rangle\}_{c_i}\}_n),$$

where v_C is the query value on column C_v and C_r is the column of query result. $\{t_1^i, t_2^i\}_n$ are query tokens for n nodes and ct_L is the ORE tokens. $\{\langle\alpha, \beta\rangle, \langle l, v*\rangle\}_{c_i}$ are the matched index entries and the corresponding result values.

$$L_3^{rng}(v_C, cmp) = (\{\{b_{dif}\}_{c_i}\}_n),$$

where b_{dif} is the permuted block that differs.

$$L_4^{rng}(\mathbf{Q}) = (M_{q\times q}, T_{v*\rightarrow\alpha}),$$

where $\mathbf{Q}$ is q number of range-match queries. $M_{q\times q}$ is the symmetric bit matrix that maintains the repeated queries. Each element in the $M_{q\times q}$ is initialized as 0. For $i, j \in [1, q]$, the elements of matrix $M_{i,j}$ and $M_{j,i}$ are equal to 1 if two tokens $t_1^i = t_1^j$. $T_{v^*\rightarrow\alpha}$ is an inverted list to trace the associations between the accessed index entries on different attributes and the corresponding encrypted result, as defined in exact-match queries. Given the above definitions of adversary views, the security definition of range-match queries is given as follows:

Definition 2. *Let* $\mathsf{Rng} = (\mathsf{KGen}, \mathsf{Build}_{\mathsf{rng}}, \mathsf{Query}_{\mathsf{rng}})$ *be our scheme for secure range-match query, and let* $\mathcal{L}_1^{rng}$, $\mathcal{L}_2^{rng}$, $\mathcal{L}_3^{rng}$, *and* $\mathcal{L}_4^{rng}$ *be the leakage functions. Given a PPT adversary* $\mathcal{A}$ *and a PPT simulator* $\mathcal{S}$, *define the following probabilistic experiments* $\mathbf{Real}_{\mathcal{A}}(k)$ *and* $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(k)$:

$\mathbf{Real}_{\mathcal{A}}(k)$: *The client calls* $\mathsf{KGen}(1^k)$ *to get a private key* K. $\mathcal{A}$ *selects a dataset* $\mathbf{D}$ *and asks the client to build* $\{I_1^{rng}, \ldots, I_n^{rng}\}$ *via* $\mathsf{Build}_{\mathsf{rng}}$. *Then* $\mathcal{A}$ *conducts a polynomial number of* q *queries with the tokens and ciphertexts generated from the client. Finally,* $\mathcal{A}$ *returns a bit as the output.*

$\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(k)$: $\mathcal{A}$ *selects* $\mathbf{D}$, *and* $\mathcal{S}$ *builds* $\{I_1'^{rng}, \ldots, I_n'^{rng}\}$ *for* $\mathcal{A}$ *based on* $\mathcal{L}_1^{rng}$. *Then* $\mathcal{A}$ *performs a polynomial number of non-adaptive* q *queries. From* $\mathcal{L}_2^{rng}$, $\mathcal{L}_3^{rng}$ *and* $\mathcal{L}_4^{rng}$ *in each query,* $\mathcal{S}$ *generates the simulated tokens and ciphertexts. Finally,* $\mathcal{A}$ *returns a bit as the output.*

Rng *is non-adaptively secure with* $(\mathcal{L}_1^{rng}, \mathcal{L}_2^{rng}, \mathcal{L}_3^{rng}, \mathcal{L}_4^{rng})$ *if for all PPT adversaries* $\mathcal{A}$, *there exists a simulator* $\mathcal{S}$ *such that:*$Pr[\mathbf{Real}_{\mathcal{A}}(k) = 1] - Pr[\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(k) = 1] \leq negl(k)$, *where* $negl(k)$ *is a negligible function in* k.

Theorem 2. Rng *is non-adaptively secure with* $(\mathcal{L}_1^{rng}, \mathcal{L}_2^{rng}, \mathcal{L}_3^{rng}, \mathcal{L}_4^{rng})$ *if* $G1, G2, H1, H3, P, F1, F2, F3$ *are secure PRF.*

Proof. The objective is to prove that the adversary $\mathcal{A}$ cannot distinguish between the real index and the simulated one as defined in Definition 2. We first define random oracles $\{\mathcal{H}_{G1}, \mathcal{H}_{G2}, \mathcal{H}_{H1}, \mathcal{H}_{F1}, \mathcal{H}_{F2}, \mathcal{H}_{F3}, \mathcal{H}_O\}$.

$\mathcal{S}$ iterates over q queries all at once, and generates random strings $\{k_r', k_o', k_R', k_1', k_2', k_3', k_l'\}$. For the query (cmp, v_C, C_v, C_r) $(cmp \in \{<, >\})$ to the node i, $\mathcal{S}$ generates $t_1' = \mathcal{H}_{G1}(k_r', C_v||i)$ and $t_2' = \mathcal{H}_{G2}(k_r', C_v||i)$ as simulated tokens. Regarding the ORE query token ct_L', $\mathcal{S}$ first splits v_C into b blocks, and generates $ct_{L|i}' = \{\mathcal{H}_{F1}(k_1', v_{|i-1}||\tilde{v_{|i}'}), q_i'\}$ as simulated tokens, where $\tilde{v_{|i}'} = \pi(\mathcal{H}_{F2}(k_2', v_{|i-1}), v_{|i})$ and $q_i' = \mathcal{H}_{F3}(k_3', cmp||C_v||\tilde{v_{|i}'})$. From $\mathcal{L}_2^{rng}$, $\mathcal{S}$ computes $\alpha' = \mathcal{H}_{H1}(t_1', c_i)$ from 0 to c_i, where c_i is the number of matched entries. To simulate an ORE ciphertext ct_R' on C_v, $\mathcal{S}$ first obtains b_{dif} from $\mathcal{L}_3^{rng}$, which is the first permuted block that differ between ct_L and ct_R. Then $\mathcal{S}$ simulates $2^d - 1$ sub blocks in each block. For the block b_{dif}, $\mathcal{S}$ simulates $z' = q_i' + \mathcal{H}_O(\mathcal{H}_{F1}(k_1', v_{|i-1}||\tilde{v_{|i}'}), \gamma')$ as the matched sub block, where γ' is a random string. After that, it generates random strings for the rest of blocks.

To simulate index entries, $\mathcal{S}$ computes $\beta' = \mathcal{H}_{H3}(t_2', c_i) \oplus (ct_R'||R'^*)$, where R'^* is $\mathcal{H}_R(k_R'||\lambda \oplus R)$. Then, with a random string k_l', $\mathcal{S}$ simulates $l' = \mathcal{H}_P(k_l', C_r||R)$, and generates random strings $v^{*'}$ as the simulated value with the same length to the real result. For repeated queries, $\mathcal{S}$ can return corresponding tokens and ciphertexts from $\mathcal{L}_4^{rng}$ just like mentioned in exact-match queries. After all queries are simulated, $\mathcal{S}$ inserts m_i random index entries to $I_i'^{rng}$, where m_i is obtained from $\mathcal{L}_1^{rng}$.

Due to the semantic security of secure PRF, $\mathcal{A}$ cannot differentiate the simulated tokens and results from the real tokens and results. □

5.3 Correctness and Security Discussion on ORE

Our ORE scheme is built on top of the Lewi and Wu's scheme [13]. We customize it in the following two ways. First, the order information is fully protected by secretly embedding it in each ORE ciphertext block via PRF with the attribute and block index. Second, the random permutation technique is carefully utilized to hide the location of the first different block of two ciphertexts during the comparison. We will later show that the above two improvements hide the information against attackers to launch attacks on secure range queries. Next, we perform correctness analysis of the proposed ORE scheme, and discuss how our ORE scheme can defend against existing leakage-abuse attacks [12], [17], [20], [39], [44], [45].

5.3.1 Correctness Analysis

To prove that the proposed ORE scheme returns correct comparison results, we show that there exists one and only one bit block matched during the comparison between the ORE token and ORE ciphertext. Accordingly, we present the following theorem:

Theorem 3. *Given a query token* m_1 *on the attribute* C *and the ORE ciphertext* m_2 *with* b *blocks with the length of* d *bits, and the matching condition* $\tilde{\times} \leftarrow \{>, <\}$, *if* $m_1 \tilde{\times} m_2$ *stands, then there exists one and only one matched sub-block* $m_{2|i}^j$, *where* $i \in \{1, b\}$ *and* $j \in \{1, 2^d - 1\}$.

Proof. Recall that the sub-block in an ORE ciphertext m_2 is encrypted as $F3(k_3, cmp||C||j) + Q(F1(k_1, m_{2|i-1}||j), \gamma)$, where $cmp \in \{>, <\}$, C is the attribute, $j \in \{2^d - 1\}$ is the unique sub-index, and $m_{2|i-1}$ is the prefix value. Therefore, if a matched sub-block is found, m_1 and m_2 must coincide with all the following conditions: (1) the query condition and attribute should be the same. (2) the sub-index j should be the same. (3) the prefix values should be the same, i.e., $m_{1|i-1} = m^j_{2|i-1}$.

Assume that $m_{1|s}$ and $m^j_{2|s}$ match the query condition, where $s \in \{1, b\}$, and there exists another entry $m^p_{2|l}$ in m_2 that matches the query condition. In other words, $m^j_{2|s} = m^p_{2|l}$, where $s \neq l$ or $j \neq p$. For the case $s \neq l$, it means that there exist two matched entries in different blocks. Recall that the matched blocks should have the same prefix value. If the two entries are in two different blocks, the bit length of their prefix must be different. Thus, this case is untenable. For the case $j \neq p$, it means that the matched entries are in two different sub blocks. As mentioned, the sub-index in a specific block is unique. Namely, $m^j_{2|s} = m^p_{2|l}$ if and only if $j = p$. Thus, there exists only one encrypted entry matched the query condition. □

5.3.2 More Discussion

As analyzed in Section 5.2, our range-match index based on the new ORE scheme ensures strong protection against attackers with snapshot access to the encrypted database. In this subsection, we generalize the existing attacks on OPE/ORE schemes, and discuss how our enhanced ORE design is secure against these attacks.

Against Sorting Attack. In [17], Naveed et al. propose a conceptually simple approach to exploit OPE-encrypted databases. Particularly, the sorting attack exploits the similarity of the order information between the OPE ciphertexts and a publicly available dataset to map; it maps each well-ordered OPE ciphertext to the element of the plaintext space with the same ranks. Later, Grubbs et al. [20] devise a new leakage-abuse attack that abstracts the sorting attack as a non-crossing bipartite matching problem to improve attack accuracy. We are aware that all these sorting attacks are only applicable to deterministic OPE schemes with the rank leakage.

In contrast, the order information in our scheme is transformed into tokens, and is securely embedded in the ORE ciphertext block with a random nonce. After comparison, no plaintext order information is revealed. Therefore, our ORE design can effectively defend against sorting attacks.

Against Multi-Column Attack. In [12], Durak et al. extend the above sorting attack by using range queries among different data attributes. This multi-column attack aims to exploit the inter-column correlations to reveal more information about the underlying values. To resist this attack, our ORE construction embeds the column attribute with different comparison results into random masks in ORE ciphertext block encryption. Thus, the attacker will learn neither the order of the underlying values on a column attribute, nor whether two different queries are conducted in the same order condition. Besides, the query tokens for the same column attribute are different at different data nodes. Thus, this approach can further hide the inter-column correlations. On the other hand, EncKV utilizes an interactive batch query mechanism to further hide the associations across data values on different attributes. During the range query procedure, each node only learns the matched value associated to the same column attribute, and it cannot exploit the associations between values in different attributes of a data record, thereby effectively addressing inference attacks. Thus, the multi-column attack is not applicable to our range-match query protocol.

Against Access Pattern Attack. In [12], Durak et al. also propose a specific ORE attack on Chenette et al.'s scheme [18]. As mentioned, the comparison result in [18] leaks the order/equality information, as well as the position of the first bit that differs between two ciphertexts. Thus, the attacker can directly learn the value of each different bit after multiple comparisons. We note that this attack can hardly to determine the value in our ORE construction, because values are encrypted into ciphertext blocks. Meanwhile, since both ORE ciphertext blocks and token blocks are randomly shuffled before uploading to the server node, the attacker cannot learn the original location of these matched blocks. As a result, our new ORE construction prevents the access pattern attack.

Against Reconstruction Attack. In [44], Kellaris et al. introduce a generic reconstruction attack on encrypted databases using range query leakage, such as access pattern and communication volume. Essentially, the attacker first conducts enough well-ordered range queries and then determines the order relations of encrypted data by observing the distribution of query results. Thus, this attack can recover the ciphertexts without knowing the orders of ciphertexts and queries. We note that this attack highly relies on prior knowledge of query distributions, which makes it difficult to be launched in our distributed data store. Because the attacker needs to compromise all the server nodes in EncKV to obtain the full query results. Besides, adding dummy records can mitigate the recovery rate of this attack. Recently, Lacharite and Minaud extend the above attack and show that it can recover the ciphertext without prior knowledge of the query distribution when the dataset is dense [45]. Likewise, one may always add dummy records to improve security. The authors also propose another reconstruction attack approach without the assumption of the dense condition. But this attack still relies on the auxiliary information of order results to launch the attack, which is protected in our ORE scheme.

6 Experimental Evaluation

6.1 Prototype Implementation

We implement the proposed system prototype[2] in C++ and perform the evaluation on Amazon Web Services. We create AWS "M4-xlarge" instances with 4 vcores (2.4 GHz Intel Xeon ® E5-2676 v3 CPU), 40 GB SSD and 16 GB RAM. In this experiment, we generate a Redis (v3.2.0) cluster that consists of 9 AWS "M4-xlarge" instances as the data nodes of the KV store and 4 AWS "M4-xlarge" instances as the clients of data applications. All of these instances are installed on Ubuntu server 14.04. Our system uses Apache Thrift (v0.9.2) to implement the remote procedure call (RPC) between the clients and servers.

For cryptographic primitives, we use OpenSSL (v1.01f) to implement the symmetric encryption via AES-128 and the

2. EncKV: An Encrypted Key-value Store with Rich Queries: online at https://github.com/CongGroup/ASIACCS-17.

TABLE 1
Space Consumption of Encrypted Index

# Values	Exact-match	2 bit block	4 bit block	8 bit block
400K	0.012(GB)	0.155(GB)	0.370(GB)	3.052(GB)
600K	0.018(GB)	0.232(GB)	0.554(GB)	4.578(GB)
800K	0.024(GB)	0.310(GB)	0.739(GB)	6.104(GB)

TABLE 2
ORE Performance Comparison

OPE/ORE Schemes	Block size	Encryption	Comparison
Boldyreva OPE [11]	-	$\sim 3601.82\ \mu$ s	$\sim 0.36\ \mu$ s
Chenette ORE [18]	1 bit	$\sim 2.06\ \mu$ s	$\sim 0.48\ \mu$ s
Lewi ORE [13]	4 bits	$\sim 54.48\ \mu$ s	$\sim 0.38\ \mu$ s
Our ORE scheme	4 bits	$\sim 81.68\ \mu$ s	$\sim 0.97\ \mu$ s

pseudo-random function via HMAC-256. EncKV's encrypted exact-match and range-match indexes are integrated into the implementation[3] of the distributed local index framework [16]. The proposed ORE design builds on top of the implementation[4] of the ORE scheme [13]. In this evaluation, we evaluate 2-bit, 4-bit and 8-bit parameter settings as the block size for ORE encryption. In total, EncKV's implementation consists of 17083 lines of C++ code (which includes 2508 lines for the enhanced ORE scheme).

6.2 Performance Evaluation

Our evaluation targets on the encrypted indexes, query performance, system scalability, and bandwidth overhead.

Index Evaluation. Table 1 presents the index space consumption of the exact-match index and range-match index respectively. For the exact-match index, each entry is generated by using AES-128 encryption algorithm. Thus, the size of each index pair $\langle \alpha, \beta \rangle$ is 256 bits. As we can see, the size of exact-match indexes grows linearly from 0.012 GB to 0.024 GB as the number of indexed values increases to 800K. Regarding the space consumption of the range-match indexes, each index entry additionally contains an ORE ciphertext ct_R for the range comparison. As the construction of an ORE ciphertext is encrypted using blocks, the size of ciphertext depends on the bit length of blocks d. Specifically, each block ciphertext contains $2^d - 1$ sub blocks, where each is 64 bits (truncated from AES cipher output). With a 128-bit nonce, encrypting a 32-bit value requires $128 + 64 \times (2^d - 1) \times 32/d$ bits. Table 1 shows that the size of range-match index grows quickly in the block size d. As mentioned in [13], there is a tradeoff in ciphertext space and security. The larger block size has stronger security while introducing more space cost. In Table 2, we compare existing OPE/ORE schemes [11], [13], [18] with our ORE scheme by generating ORE ciphertexts locally without column attributes. It shows that the encryption time cost of our ORE scheme is significantly faster than the OPE scheme [11]. Specifically, generating an ORE ciphertext with 4-bit design only requires 81.68 μs, which is almost 44 times faster compared to the OPE scheme. For order comparison, our scheme introduces additional computation cost due to the block permutation. Nonetheless, the average processing time of the ORE comparison is still less than 0.97 μs, which is within an acceptable level.

Figs. 3a and 3b present the time cost of building the encrypted indexes at the client side. Both time costs increase linearly with the growing number of indexed values. Building the range-match index takes more time because it needs to construct ORE ciphertexts. Fig. 3b also compares the time cost of range-match index in different block sizes of the ORE encryption scheme. The result shows that the 8-bit ORE design slowly increases the building time by almost 32 percent, when there are 160K indexed values. In addition, we compare the initialization time cost of our 8-bit index design with our conference version (denoted as YGWWLJ17). Since each ORE ciphertext block should be generated independently with random permutation, it introduces additional time cost for building the encrypted range-match index. As we can see from the Fig. 3b, although the building time is roughly 5 times higher compared to YGWWLJ17, it is a one-time setup cost, and this can further be improved via parallelization.

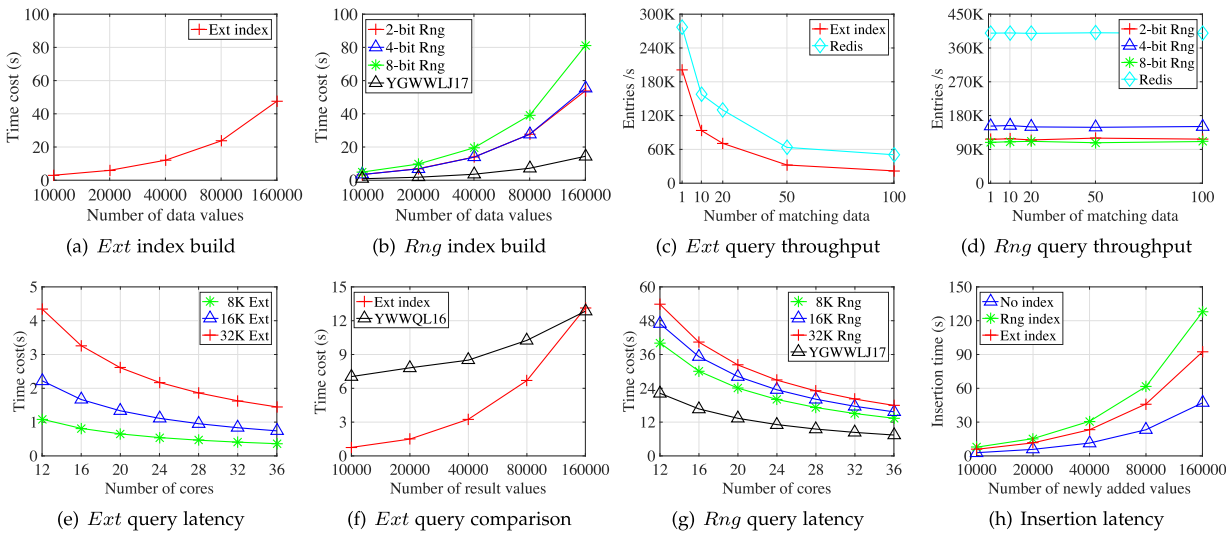
Query Evaluation. To assess the efficiency and scalability of EncKV, we evaluate the query performance using batch queries. These queries are generated by one target attribute using exact-match and range-match query protocols. In this experiment, we pre-loaded the database with 160K data records to evaluate the practicality of EncKV.

Fig. 3c compares the throughput of exact-match index with a plaintext Redis index, when varying the number of duplicates per value of the indexed attribute. In our design, each node increments a counter to find all matched records. Then the client expands these matched records to relevant tokens for nodes to fetch their underlying encrypted values. Thus, the number of tokens corresponding to a query value increases with the number of duplicates. This treatment hides the relations between data values on different attributes, but it introduces a significant part of the query time cost. In contrast, a redis index maps duplicates to a single reference and locates them all in a scan. Overall, our evaluation shows that both exact-match index and Redis index decrease in similar proportions. The throughput of EncKV decreases from about 93.5K to 21.9K as the number of duplicate values grows from 10 to 100. when the whole column values are unique, the query throughput of exact-match queries can achieve up to 200K entries per second, while that of Redis is 277K entries per second.

Regarding the throughput of range-match query, we compare a plaintext version with different types of range-match index scheme as presented in Fig. 3d. The throughput of range-match query remains stable because the query time complexity is O(n) (n is the number of records at a certain attribute). Specifically, the encrypted range-match index incurs an overall throughput loss of approximately $3.6\times$ in the worst case. The overhead comes from the cost of blocks matching and cryptographic operations during the ORE comparison. Furthermore, we measure the ORE comparison overhead via varying the block size of the ORE encryption scheme. An interesting result shows that 4-bit ORE scheme achieves a better efficiency than other ORE schemes. Specifically, the throughput with 4-bit ORE scheme is about 37.5 percent higher than that of 8-bit design,

3. An encrypted, distributed, and searchable key-value store: online at https://github.com/CongGroup/BlindDB.

4. An implementation of order-revealing encryption: online at https://github.com/kevinlewi/fastore.



(a) Ext index build (b) Rng index build (c) Ext query throughput (d) Rng query throughput

(e) Ext query latency (f) Ext query comparison (g) Rng query latency (h) Insertion latency

Fig. 3. Evaluation of the system performance.

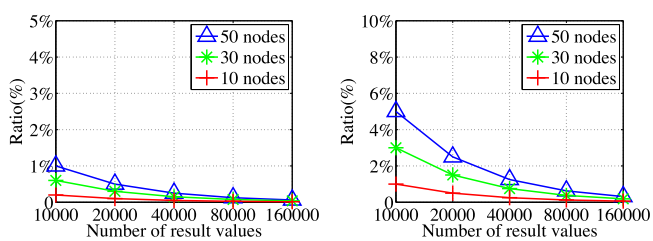
which is around 151K entries per second. On the one hand, the ORE ciphertext size increases by $8.4\times$ from 4-bit design to 8-bit design when encrypting 32 bits value. Reading a large size ciphertext introduces a significant overhead. On the other hand, the comparison complexity for a d-bit ORE design is $(32/d)^2$. Thus, the throughput of 4-bit ORE design is slightly higher than 2-bit design. Overall, there is a tradeoff in data security and query performance. The larger block size has stronger security while incurring more performance penalty. Meanwhile, the result shows that the time cost for 8-bit ORE comparison ranges from around 9.46s to 9.77s. This evaluation result confirms that our enhanced ORE design can support secure range query efficiently.

We further evaluate the query latency of exact-match and range-match queries respectively. In Fig. 3e, we can find that as the number of nodes increases, the latency of exact-match queries that returns a fixed number of results is reduced dramatically in similar proportions. Specifically, when the number of matched records is 32K, the query latency with 36 cores (i.e, 9 nodes) is around 1.45s, which is roughly one-third of the latency with 12 cores. For the evaluation of range query latency, we pre-inserted 160K data records with 8-bit ORE design. Fig. 3g shows that the result follows a similar downward trend as the number of nodes increases. The latency of range-match queries with 32 cores is roughly half of the latency with 16 cores for returning 32K matched encrypted values. The results confirm that EncKV benefits from the encrypted local index framework and can effectively handle queries in parallel. Fig. 3g measures the performance comparison between our new range-match index design and previous design. Kindly recall that in our conference version, the order comparison is conducted with the block sequence. Thus, compared with the previous design, our new ORE comparison protocol on shuffled block is slower. Nonetheless, as shown in Fig. 3g, processing 8K matched records with 36 cores instance requires just 13.3s, which is modest for common data store applications. In addition, in exchange, our new design leaks strictly less information than the state-of-the-art OPE/ORE schemes as discussed previously. According to the evaluation result, our new index design in EncKV is shown to be still efficient while improving the security.

To gain a deeper understanding of EncKV's query performance, we compare the exact-match query performance with an existing work [16] (denoted as *YWWQL16*) using the same set of queries, as presented in Fig. 3f. Intuitively, our design achieves optimal time complexity that scales in the number of matched results. In contrast, *YWWQL16* performs token matching by enumerating all values on an attribute. The query time of *YWWQL16* slowly improves with the result size because the number of tokens increases with the returned records. And the computation of generating a token is a constant-time operation. When the result size is 10K, the query time of EncKV is around 0.75s, which is almost $10\times$ faster than the *YWWQL16*. Further, when all records are returned, the time cost of exact-match query is the same in both works.

In this experiment, we also evaluate the incremental scalability of EncKV by measuring the time cost for index entry insertion. As shown in Fig. 3h, it will not introduce much overhead compared to the case without indexing. Note that the time cost includes the network transmission for each new indexed value, and thus it is much higher than the index building cost (as shown in Figs. 3a and 3b). When the number of newly added values is 10K, it takes 5.86s to index entries for exact-match indexes, of which 48.6 percent due to the network transmission.

Bandwidth Evaluation. Recall that EncKV requires the client to generate query tokens for each node. To assess the bandwidth overhead, Fig. 4 shows the ratio between the query token size and result size. As shown in Fig. 4a, the ratio of exact-match query decreases gradually with the increased



(a) Ext bandwidth ratio (b) Rng bandwidth ratio

Fig. 4. Query token bandwidth overhead.

size of results. Specifically, the ratio for 30 nodes decreases from around 0.6 to 0.15 percent when the number of result values increases from 10K to 40K. Meanwhile, the result shows that the increasing number of nodes can render a rise in the bandwidth. The ratio of 10K result size increases from about 0.2 to 1 percent as the number of nodes increases to 50. Likewise, Fig. 4b shows that the ratio of range-match queries follows a similar downward trend as the number of result size increases. But the corresponding ratio is higher than the exact-match queries because the ORE token ct_L enlarges the size of the query token. As shown, the ratio for 50 nodes reaches around 5 percent when the number of result size is 10K. Nevertheless, the bandwidth overhead of query tokens is negligible to the size of results in our experiment.

7 Conclusion

EncKV is a functionally rich key-value store that can handle large volumes of encrypted data records with guaranteed data protection. It leverages the latest practical primitives for searching over encrypted data (i.e., SSE and ORE) and provides encrypted local indexes to support exact-match and range-match queries via secondary attributes of data records. EncKV's prototype is deployed on a Redis cluster. The extensive experiments for performance evaluation confirm that it preserves advantages in existing distributed data stores such as high throughput and linear scalability. As future work, we plan to explore advanced encryption techniques to support other rich queries, such as Boolean search and Join search. Meanwhile, we leave how to extend EncKV to support a multi-client setting as our future work.

Acknowledgments

This work was supported by the China National Science and Technology Major Project Grant No. 2016YFB0800804, the National Natural Science Foundation of China under Project 61732022, 61572412 and 61672195, and the Research Grants Council of Hong Kong under Project CityU 11204215, CityU 11276816, CityU 11212717, and CityU C1008-16G, the Oceania Cyber Security Centre POC scheme, and an AWS in Education Research Grant award. A preliminary version [1] of this paper was presented at the 12th ACM Asia Conference on Computer and Communications Security (ASIACCS'17).

References

[1] X. Yuan, Y. Guo, X. Wang, C. Wang, B. Li, and X. Jia, "Enckv: An encrypted key-value store with rich queries," in *Proc. ACM Asia Conf. Comput. Commun. Security*, 2017, pp. 423–435.

[2] Redis, "An advanced key-value cache and store," 2015. [Online]. Available: http://redis.io/

[3] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," *ACM SIGOPS Operating Syst. Rev.*, vol. 41, no. 6, pp. 205–220, 2007.

[4] J. Ousterhout, A. Gopalan, A. Gupta, A. Kejriwal, C. Lee, B. Montazeri, D. Ongaro, S. J. Park, H. Qin, M. Rosenblum, et al., "The ramcloud storage system," *ACM Trans. Comput. Syst.*, vol. 33, no. 3, 2015, Art. no. 7.

[5] FoundationDB, "FoundationDB: Data Modeling," [Online]. Available: http://www.odbms.org/wp-content/uploads/2013/11/data-modeling.pdf

[6] A. Kejriwal, A. Gopalan, A. Gupta, Z. Jia, S. Yang, and J. Ousterhout, "Slik: Scalable low-latency indexes for a key-value store," in *Proc. USENIX Annu. Tech. Conf.*, 2016, pp. 57–70.

[7] Information is Beautiful, "World's biggest data breaches," 2016. [Online]. Available: http://www.informationisbeautiful.net/visualizations/worlds-biggest-dat a-breaches-hacks/

[8] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proc. 13th ACM Conf. Comput. Commun. Security*, 2006, pp. 79–88.

[9] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proc. ACM Conf. Comput. Commun. Security*, 2012, pp. 965–976.

[10] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner, "Dynamic searchable encryption in very large databases: Data structures and implementation," in *Proc. Netw. Distrib. Syst. Security Symp.*, 2014, pp. 23–26.

[11] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill, "Order-preserving symmetric encryption," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 2009, pp. 224–241.

[12] F. B. Durak, T. M. DuBuisson, and D. Cash, "What else is revealed by order-revealing encryption?" in *Proc. ACM ACM SIGSAC Conf. Comput. Commun. Security*, 2016, pp. 1155–1166.

[13] K. Lewi and D. J. Wu, "Order-revealing encryption: New constructions, applications, and lower bounds," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2016, pp. 1167–1178.

[14] R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan, "Cryptdb: Protecting confidentiality with encrypted query processing," in *Proc. ACM 23rd ACM Symp. Operating Syst. Principles*, 2011, pp. 85–100.

[15] V. Pappas, B. Vo, F. Krell, S. Choi, V. Kolesnikov, A. Keromytis, and T. Malkin, "Blind seer: A scalable private DBMS," in *Proc. IEEE Symp. Security Privacy*, 2014, pp. 359–374.

[16] X. Yuan, X. Wang, C. Wang, C. Qian, and J. Lin, "Building an encrypted, distributed, and searchable key-value store," in *Proc. 11th ACM Asia Conf. Comput. Commun. Security*, 2016, pp. 547–558.

[17] M. Naveed, S. Kamara, and C. V. Wright, "Inference attacks on property-preserving encrypted databases," in *Proc. ACM 22nd ACM SIGSAC Conf. Comput. Commun. Security*, 2015, pp. 644–655.

[18] N. Chenette, K. Lewi, S. A. Weis, and D. J. Wu, "Practical order-revealing encryption with limited leakage," in *Proc. Int. Conf. Fast Softw. Encryption*, 2016, pp. 474–493.

[19] B. Fuller, M. Varia, A. Yerukhimovich, E. Shen, A. Hamlin, V. Gadepally, R. Shay, J. D. Mitchell, and R. K. Cunningham, "Sok: Cryptographically protected database search," in *Proc. IEEE Symp. Security Privacy*, 2017, pp. 172–191.

[20] P. Grubbs, K. Sekniqi, V. Bindschaedler, M. Naveed, and T. Ristenpart, "Leakage-abuse attacks against order-revealing encryption," in *Proc. IEEE Symp. Security Privacy*, 2017, pp. 655–672.

[21] D. Cash, F.-H. Liu, A. O'Neill, and C. Zhang, "Reducing the leakage in practical order-revealing encryption," Cryptology ePrint Archive, Report 2016/661, 2016. [Online]. Available: http//eprint.iacr.org/2016/661

[22] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich, "Processing analytical queries over encrypted data," in *Proc. VLDB Endowment*, vol. 6, pp. 289–300, 2013.

[23] H. Shafagh, A. Hithnawi, A. Droescher, S. Duquennoy, and W. Hu, "Talos: Encrypted query processing for the internet of things," in *Proc. 13th ACM Conf. Embedded Networked Sensor Syst.*, 2015, pp. 197–210.

[24] R. Poddar,T. Boelter, and R.A. Popa, "Arx: A strongly encrypted database system," Cryptology ePrint Archive, Report 2016/591, 2016. [Online]. Available: http//eprint.iacr.org/2016/591

[25] A. Papadimitriou, R. Bhagwan, N. Chandran, R. Ramjee, A. Haeberlen, H. Singh, A. Modi, and S. Badrinarayanan, "Big data analytics over encrypted datasets with seabed," in *Proc. USENIX Symp. Operating Syst. Design Implementation*, 2016, pp. 587–602.

[26] S. Bajaj and R. Sion, "Trusteddb: A trusted hardware based database with privacy and data confidentiality," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2011, pp. 205–216.

[27] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich, "VC3: Trustworthy data analytics in the cloud using SGX," in *Proc. IEEE Symp. Security Privacy*, 2015, pp. 38–54.

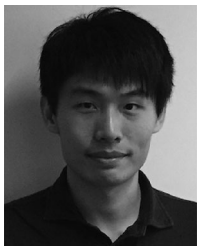
[28] A. Baumann, M. Peinado, and G. Hunt, "Shielding applications from an untrusted cloud with haven," in *Proc. USENIX Symp. Operating Syst. Design Implementation*, 2014, pp. 267–283.

[29] C. Priebe, K. Vaswani, and M. Costa, "Enclavedb: A secure database using SGX," in *Proc. IEEE Symp. Security Privacy*, 2018, pp. 264–278.

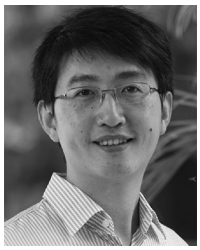
[30] Q. Wang, M. He, M. Du, S. S. M. Chow, R. W. F. Lai, and Q. Zou, "Searchable encryption over feature-rich data," *IEEE Trans. Depend. Secure Comput.*, vol. 15, no. 3, pp. 496–510, May/Jun. 2018.
[31] R. Bost, "Sophos - forward secure searchable encryption," in *Proc. ACM Conf. Comput. Commun. Security*, 2016, pp. 1143–1154.
[32] K. S. Kim, M. Kim, D. Lee, J. H. Park, and W.-H. Kim, "Forward secure dynamic searchable symmetric encryption with efficient updates," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2018, pp. 1449–1463.
[33] R. Bost, B. Minaud, and O. Ohrimenko, "Forward and backward private searchable encryption from constrained cryptographic primitives," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2017, pp. 1465–1482.
[34] S.-F. Sun, X. Yuan, J. Liu, R. Steinfeld, A. Sakzad, V. Vo, and S. Nepal, "Practical backward-secure searchable encryption from symmetric puncturable encryption," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2018, pp. 763–780.
[35] R. A. Popa, F. H. Li, and N. Zeldovich, "An ideal-security protocol for order-preserving encoding," in *Proc. IEEE Symp. Security Privacy*, 2013, pp. 463–477.
[36] F. Kerschbaum, "Frequency-hiding order-preserving encryption," in *Proc. ACM 22nd SIGSAC Conf. Comput. Commun. Security*, 2015, pp. 656–667.
[37] D. S. Roche, D. Apon, S. G. Choi, and A. Yerukhimovich, "Pope: Partial order preserving encoding," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2016, pp. 1131–1142.
[38] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-abuse attacks against searchable encryption," in *Proc. ACM Conf. Comput. Commun. Security*, 2015, pp. 668–679.
[39] P. Grubbs, T. Ristenpart, and V. Shmatikov, "Why your encrypted database is not secure," in *Proc. ACM 16th Workshop Hot Topics Operating Syst.*, 2017, pp. 162–168.
[40] P. Grubbs, R. McPherson, and M. Naveed, "Breaking web applications built on top of encrypted data," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2016, pp. 1353–1364.
[41] Y. Zhang, J. Katz, and C. Papamanthou, "Integridb: Verifiable SQL for outsourced databases," in *Proc. ACM 22nd ACM SIGSAC Conf. Comput. Commun. Security*, 2015, pp. 1480–1491.
[42] D. Boneh, K. Lewi, M. Raykova, A. Sahai, M. Zhandry, and J. Zimmerman, "Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 2015, pp. 563–594.
[43] M. Du, Q. Wang, M. He, and J. Weng, "Privacy-preserving indexing and query processing for secure dynamic cloud storage," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 9, pp. 2320–2332, Sep. 2018.
[44] G. Kellaris, G. Kollios, K. Nissim, and A. O'Neill, "Generic attacks on secure outsourced databases," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2016, pp. 1329–1340.
[45] M.-S. Lacharit, B. Minaud, and K. G. Paterson, "Improved reconstruction attacks on encrypted data using range query leakage," in *Proc. IEEE S&P*, 2018, pp. 297–314.

Yu Guo received the BE degree in software engineering from Northeastern University, in 2013, and the MS degree in electronic commerce from the City University of Hong Kong, in 2014. He is currently working toward the PhD degree with the Department of Computer Science, City University of Hong Kong. His research interests include cloud computing security, searchable encryption, and privacy-preserving data processing.

Xingliang Yuan received the BS degree in electrical engineering from the Nanjing University of Posts and Telecommunications, in 2008, the MS degree in electrical engineering from the Illinois Institute of Technology, in 2009, and the PhD degree in computer science from the City University of Hong Kong, in 2016. He is a lecturer with the the Faculty of Information Technology, Monash University, Australia. His research interests include cloud security, privacy-aware computing, and secure networked systems.

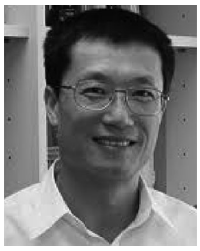


Xinyu Wang received the BE degree from East China Jiaotong University, in 2011. He is currently working toward the PhD degree with the Department of Computer Science, City University of Hong Kong. He worked for Tencent as a software engineer from 2011 to 2013. His research interests include cloud computing, network security, and big data.



Cong Wang (SM'17) received the BE degree in electronic information engineering and the ME degree in communication and information system from Wuhan University, China, and the PhD degree in electrical and computer engineering from the Illinois Institute of Technology. He is currently an associate professor with the Department of Computer Science, City University of Hong Kong. His research has been supported by multiple government research fund agencies, including the National Natural Science Foundation of China, the Hong Kong Research Grants Council, and the Hong Kong Innovation and Technology Commission. His current research interests include data and computation outsourcing security in the context of cloud computing, network security in emerging Internet architecture, multimedia security and its applications, and privacy-enhancing technologies in the context of big data and IoT. He received the Presidents Award from the City University of Hong Kong in 2016. He was a co- recipient of the Best Student Paper Award at CHINACOM 2009, the IEEE MSN 2015, and the IEEE ICDCS 2017. He has been serving as the TPC co-chairs for a number of IEEE conferences/workshops. He is a member of the ACM. He is a senior member of the IEEE.

Baochun Li (F'15) received the BEngr degree from the Department of Computer Science and Technology, Tsinghua University, China, in 1995, and the MS and PhD degrees from the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, in 1997 and 2000, respectively. Since 2000, he has been with the Department of Electrical and Computer Engineering, University of Toronto, where he is currently a professor. He holds the Bell Canada Endowed chair in computer engineering since August 2005. His research interests include cloud computing, multimedia systems, applications of network coding, and wireless networks. He was the recipient of the IEEE Communications Society Leonard G. Abraham Award in the Field of Communications Systems in 2000, the Multimedia Communications Best Paper Award from the IEEE Communications Society in 2009, and the University of Toronto McLean Award in 2009. He is a member of ACM and a fellow of the IEEE.



Xiaohua Jia (F'13) received the BSc and MEng degrees from the University of Science and Technology of China, in 1984 and 1987, respectively, and the DSc degree in information science from the University of Tokyo, in 1991. He is currently the chair professor with the Department of Computer Science, City University of Hong Kong. His research interests include cloud computing and distributed systems, computer networks, wireless sensor networks and mobile wireless networks. He is an editor of the *IEEE Transactions on Parallel and Distributed Systems (2006-2009), Wireless Networks, Journal of World Wide Web, Journal of Combinatorial Optimization, etc.* He is the general chair of ACM MobiHoc 2008, TPC co-chair of IEEE MASS 2009, area-chair of IEEE INFOCOM 2010, TPC co-chair of IEEE GlobeCom 2010-Ad Hoc and Sensor Networking Symposium, and Panel co-chair of IEEE INFOCOM 2011. He is a fellow of the IEEE Computer Society.