EAFS: An Efficient, Accurate, and Forward Secure Searchable Encryption Scheme Supporting Range Search

Srilakshmi.CH, Neaha RK, Nivashini S
Associate Professor, Department of CSBS ,R.M.D Engineering College
UG Scholar, Pre Final year, Department of CSBS, R.M.D Engineering College, ucb20127@rmd.ac.in
UG Scholar, Pre Final year, Department of CSBS, R.M.D Engineering College, ucb20201@rmd.ac.in

***Abstract*—Forward privacy in existing searchable symmetric encryption approaches generally supports only keyword searches,rather than range searches. In addition, existing approaches that support range search over encrypted datasets have limitations associated with efficiency, accuracy, and security. Seeking to ad- dress these limitations, we formally propose the definition of the forward privacy for secure range searches. Also, we demonstrate how three widely used cryptographic tools— order-preserving en- cryption (OPE), pseudorandom function, and one-time pad–can be leveraged to design an efficient, accurate, and forward secure (EAFS) searchable encryption scheme supporting range search inencrypted numerical databases. In the proposed EAFS scheme, a trapdoor only matches the last data record that satisfies the searchrange, and the other results are found iteratively using the previous result. The chain-like search and the embedded ciphertexts of OPE simultaneously guarantee its efficiency, forward privacy, and accu- racy. We also use the simulation-based method to demonstrate that our scheme is secure. Then, we implement EAFS in Microsoft Azure and evaluate its performance. The evaluation findings demonstrate that our proposed scheme supports secure range search practically and efficiently.**

***Index Terms*—Encrypted database, forward secure, range search, searchable encryption (SE).**

II. Introduction

Our data are increasingly stored in cloud servers (CSs) for a range of reasons, such as lower costs of data manage- ment, higher quality of service, and ability to access our data anywhere anytime using any computing device. In some cases, data (on our mobile devices and laptops) are being outsourced to the CSs (e.g., iCloud or OneDrive) by default. There are known security and privacy issues associated with the outsourcing of data, such as the cloud service provider (CSP) getting access to user contents without their explicit permission. This reinforces the importance of searchable encryption (SE), a cryptography-based scheme that enables searching in the ciphertext domain without leaking any information to untrusted servers. There are many different SE schemes, with varying functionalities (e.g., keyword searches and similarity searches).

Range searches are another common function in databases that focuses on numeric comparisons, for example to locate users between 20 and 40 years old. To achieve better efficiency in range searches, the data may be processed to fit some specific data structure (e.g., B-tree), in which the data are stored in order. This is easy to achieve in the plaintext domain, but not the case when

I.

databases are encrypted. First, the numeric comparisons cannot be made in the encrypted environment due to the lack of semantic information. Also, it is impractical to enumerate all possible cases for the entire range in a search request in order to check the equality. Second, the order between different data records is also sensitive information. In other words, the data owner (DO) generally does not want the CS to know about the order between different data records, and the index should not expose the order information prior to searching. Therefore, it is challenging to conduct range search in the ciphertext domain.

In 2016, Zhang *et al*. [1] proposed the file-inject attacks on SE schemes, and many forward secure solutions were proposed to mitigate such an attack. A forward secure scheme guarantees

that the update operation does not leak the relationship between the updating keywords and the previous documents. That isto say, the cloud cannot learn whether entries that have been updated recently contain some specific keywords. However, all the forward secure schemes discussed so far support only keyword search rather than range search.

TABLE I
COMPARATIVE SUMMARY OF OUR SCHEME AND PRIOR ART

scheme	Search complexity	Multi-dimensional	Without false-positive	Single server model	Forward privacy
[11]	$O(1)$	No	No	Yes	No
[13]	$\mu_Q O(\log(n))$	Yes	No	Yes	No
[16]	$O(n)$	No	Yes	No	N/A
[17]	$O(n)$	No	Yes	No	N/A
[21]	$\mu_Q O(\log(n))$	No	No	Yes	No
Ours	$O(\mu_Q + \mu_F)$	Yes	Yes	Yes	Yes

n denotes the total number of data entities in the database, μ_Q is the number of results satisfying the query, and μ_F is the number of false positives

A number of schemes have been proposed to conduct securerange search in encrypted databases.
such as those described in [5] and [6] are designedto facilitate numeric comparisons in the ciphertext domain by using order-preserving encryption (OPE), a special functional cryptosystem. However, OPE leads to the leakage of the order of encrypted data, which is not desirable in some situations. A number of other schemes such as those described in [7]–[9] support numeric comparisons through computations in the ciphertext domain, using the homomorphic encryption, another functional cryptosystem. However, such schemes rely on a two-cloud model in which one CS (who owns the encrypted databases) primarily provides storage resources and the other CS (who owns the secret key used to encrypted the databases) mainly provides the computational resources. There is a strong assumption that both CSs are independent of each other and do not collude, which is difficult to guarantee in practice. Another inherent drawback is that the number of calculations is related to the total number of data entities.

To ensure the efficiency of range searches, there is another kind of bucket-based schemes [2]–[4] where the range is divided into different partitions, called buckets, and each encrypted data entity is mapped with the respective bucket. The numeric comparison is transformed to equality checking for specific buckets, which achieves $O(1)$ search complexity. However, such schemes lead to false-positive. An additional operation of filtering may be needed in the client side, which adversely affects the user's experience.

One would observe from the above discussions reinforces the importance of designing an index structure to achieve more efficient range searches, since the index contains the connections between each data item and can expedites the search phase. However, by updating some elaborate data items and observing the change of the index, attackers can learn the specific distribution of an encrypted database (such an attack is also referred to as the file-inject attack in range search). In other words, while attackers cannot obtain the exact value of each data item, more sensitive information (e.g., the exact range of each partition in the bucket-based schemes) is exposed. This necessitates the support for forward security in range search.

There are tradeoffs between efficiency, accuracy, and privacy. It is clear that SE schemes with $O(1)$ search complexity cannot achieve forward privacy since the connections of data items are apparent in the index, and schemes with $O(n)$ search complexity cannot be deployed in large-scale databases. Hence, to achieve forward secure range search, the connections must be concealed in indexes and appear step by step during the search phase. Besides, the index-based range search schemes cannot achieve high accuracy. Hence, one can conclude that it is challenging to present a SE scheme that supports efficient range search and guarantees both accuracy and forward privacy.

In this article, we present the formal definition for forward privacy in range searches, and then propose an efficient, accurate, and forward secure (EAFS) SE scheme that satisfies this definition. To be specific, our scheme achieves secure range search in a single CS model, and the accurate results can be obtained without redundant interactions and postprocessing operations.

The search phase consists of two parts. The first part is to find the dataset that completely meets the search request, and the second part is to find the dataset that partially meets the search request and then compare to return the correct results. Thus, the complexity of the search is related to the size of the datasets mentioned above. Both order information and similarity information are hidden before a search is conducted, and forward privacy is guaranteed when the data records are updated. Furthermore, we also compare our proposed scheme with Wu et al.'s [4] state-of-the-art tree-based SE scheme (ServeDB). Based on the comparison, we observe that our scheme achieves both accuracy and forward privacy, unlike ServeDB. Findings from the experiments also demonstrate that the computational cost of our scheme is less than that of ServeDB. Table I gives a comparative summary of our scheme and prior art.

The following summarizes the article.

1) We propose the first formal definition of forward privacy in secure range searches.
2) We design an EAFS SE scheme that supports secure range searches in encrypted databases. This is the first scheme that simultaneously achieves efficiency, accuracy, and forward security.
3) We demonstrate that the EAFS is secure under the semi-honest model by utilizing the simulation-based method.
4) We implement the EAFS and evaluate its performance on Microsoft Azure CSs. A comparison with another state-of-the-art tree-based SE scheme [4] demonstrates that our scheme achieves both efficiency, accuracy and supports forward privacy.

The structure of our article is as follows. The literature related to secure range search is discussed in Section II. Section III presents the problem statement, and Section IV presents the relevant preliminary information. In Section V, we describe our proposed EAFS, and evaluate its security and performance in Sections VI and VII. Section VIII concludes this article.

III. EXTANT LITERATURE

As discussed in the preceding section, SE facilitates the searching over encrypted data, for example keyword searches, etc. [1]–[9]. There have also been attempts in recent years to support secure range searches, as demonstrated in the recent literature. For example, Hacigumus *et al.* [2] proposed the first SQL search scheme over secure databases. Specifically, data are mapped into a finite number of buckets, and each bucket is proposed a multidimensional bucket-based scheme. Wu *et al*. [4] designed a verification mechanism to ensure the completeness of the results. However, the verification algorithm must be run by the data user (DU). It can be regarded as a postprocessing operation that impacts the user's experience. These approaches may also result in false-positive results.

OPE and HE are two technologies that can be used to eliminate or minimize false-positive results. CryptDB [5], for example, achieves order comparison on encrypted data by multilayer encryption (also referred to as onion encryption) on OPE, but it has low efficiency and leaks sensitive information. Mavroforakis *et al*. [6] utilized OPE to design two algorithms, one of which can be used over well-spread query distributions with the other being used over skewed query distributions. However, the distribution and the order of the data encrypted by OPE would be exposed to the CS.

Xue *et al*. [7] conducted range searches using a two-cloud architecture and Paillier encryptions in an encrypted database. The secure comparison can be implemented by the intersection between two CSs. However, the DO must share her/his secret key to one of the clouds, and the two CSs must not collude. To enhance the security of the architecture, each of the two CSs holds a portion of the secret key in the scheme proposed by Cheng *et al*. [8]. However, it still suffers from the inherent drawback that both CSs may collude with each other. Wong *et al*. [9] proposed a HE-based scheme supporting data interoperability, with only a single CS. However, the scheme can only be used in a limited scenario. In addition, schemes based on HE incur significant computational overhead.

There have also been designs of schemes that support secure range search on spatial data. However, these schemes generally are inefficient, inaccurate, and/or insecure. Li *et al*. [10], for example, designed a bloom filter-based range search scheme in which the index is indistinguishable. However, the scheme's worst-case complexity is large. Demertzis *et al*. [11] proposed tree-like indices to achieve query indistinguishability, but there is a tradeoff between performance and accuracy. In addition, the secure range search can be realized by some specific cryptographic tools, such as oblivious RAM, garbled circuit, or trusted hardware (e.g., FPGA and Intel SGX). The downside is the large communication overhead.

Bost [12] formally defined forward privacy in SSE, which is a strong property against file-inject attacks. Subsequently, there have been other attempts to design forward secure schemes that are also efficient. However, existing works generally support only keyword searches and not range searches. Therefore, this is the gap we seek to address, by designing a forward secure range search scheme that is accurate and efficient.

IV. Problem Statement

A. System Architecture

As a traditional model of SE, the system architecture of our proposed approach consists of three entities, i.e., the DO, the DUs, and the CS—see also Fig. 1.

Data Owner: DO is an entity that stores its dataset on the CS. In our solution, database D contains m attributes

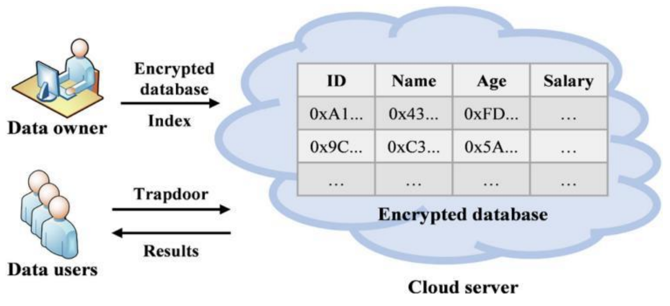

Fig. 1. System architecture.

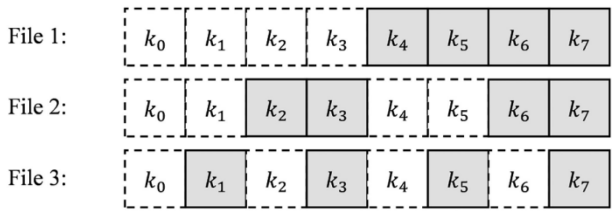

Fig. 2. Example file-inject attack on keyword search.

$\{a_1, a_2, \dots, a_m\}$ and n records $\{d_1, d_2, \dots, d_n\}$. Due to the restriction of the cryptography, the value of a data record, d_i, for an attribute, a_j, is a positive integer. In order to enable the functionality of a secure range search as well as ensuring the confidentiality of the original data, DO encrypts D to generate an encrypted database (searchable index), EDB. Then, the encrypted database EDB is outsourced by DO.

Data User: DU is an entity that makes a query and wants the result of that query. The query, q, has the same number of attributes as database D, denoted as $q = \{\langle q_1.\mathsf{low}, q_1.\mathsf{up}\rangle, \langle q_2.\mathsf{low}, q_2.\mathsf{up}\rangle, \dots, \langle q_m.\mathsf{low}, q_m.\mathsf{up}\rangle\}$, where $q_j.\mathsf{low}$ and $q_j.\mathsf{up}$ are the lower bound and the upper bound, respectively, of the search range for the jth attribute. When DU wants to make a query q, an encrypted search token (trapdoor) is generated and sent to the cloud for further search.

Cloud Server: CS is an entity providing a large amount of storage and computation resources. CS receives database EDB, from DO for storage. When a query is to be issued, a corresponding trapdoor is also received by CS. By running a search algorithm with inputting the EDB and the trapdoor, the result that satisfies the query will be returned.

B. Threat Model

As discussed earlier, in a file-inject attack the attackers may inject some elaborate keyword-document pairs to a database. We replicate the example of [1] in Fig. 2, where attackers may inject three files and each file contains four keywords which are shaded in the figure. If file 3 is returned but file 1 and file 2 are not when issuing a search toke, it means that the keyword k_1 responds to the token. In SE, the file-inject attack mainly refers to a situation where attackers inject some files and observe change(s) in the index in order to learn the relationship between the injected files and the documents stored in CS. Accordingly, the secure range search scheme can also be attacked by updating some elaborate

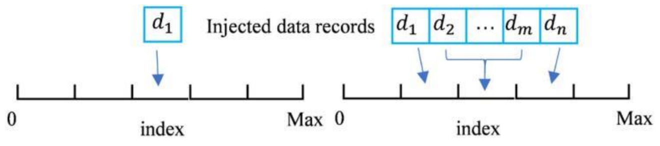

Fig. 3. Example file-inject attack on range search.

data records, and consequently some sensitive information about the newly added data would be exposed. As shown in Fig. 3, there is a frequently used index [2]–[4] that maps the data records into different buckets. An attacker can infer which data records are similar to d_1 according to the position they are located during the updating of d_1. The attacker also can infer the exact range of each bucket by updating a series of continuous data records.

As per the general assumption in the SE literature, we assume that CS is honest-but-curious. To be specific, CS follows the designed algorithms honestly, but is curious about any sensitive information about the data, the indices stored on CS, or the trapdoors to be searched. We also assume that the proposed scheme should mitigate file-inject attacks.

C. Syntax

An SE which supports secure range search is a protocol among DO, DU, and CS, which are formulated as follows.

- $(K, \mathsf{EDB}, \sigma) \leftarrow \mathsf{Init}(1^\lambda)$ is a probabilistic algorithm, which is run by the DO; it takes the secure parameter, λ, as input, and outputs two secret keys K an (initially empty) encrypted database EDB and a state σ.
- $(\sigma'; \mathsf{EDB}') \leftarrow \mathsf{Update}(K, \sigma, D; \mathsf{EDB})$ is a protocol between the client (with two secret keys, K, a state index, σ, and a set of added records, D), and the server (with an encrypted database EDB). The client outputs an updated state σ', and the server outputs an updated encrypted database EDB'.
- $(T) \leftarrow \mathsf{GenTrapdoor}(K, \sigma, q)$ is an algorithm that is run by DU; it takes two secret keys K, a state σ, and the query q as input, and it outputs the trapdoor T.
- $(\mathsf{RST}) \leftarrow \mathsf{Search}(\mathsf{EDB}, T)$ is a deterministic algorithm that is run by CS; it takes the encrypted database EDB, and the trapdoor T as input, and outputs the result set RST iteratively.

D. Design Goals

The performance goals and safety goals achieved in our proposed EAFS are designed as follows.

1) *Search Efficiency:* The search complexity on CS should be $O(\mu_Q + \mu_F)$, where μ_Q is the number of results that satisfy the query, and μ_F is the number of false-positive data records.
2) *Correctness:* Given a trapdoor, T, of query, q, CS outputs the result set $\mathsf{RST} = \{\mathsf{rst}_1, \mathsf{rst}_2, \ldots, \mathsf{rst}_k\}$. RST is correct if and only if each data record d_i ($i = 1, 2, \ldots, k$), decrypted from rst_i, is covered by the query q.
3) *Completeness:* Given a trapdoor T of query q CS outputs the result set, $\mathsf{RST} = \{\mathsf{rst}_1, \mathsf{rst}_2, \ldots, \mathsf{rst}_k\}$. RST is complete if and only if each data record, d_j, decrypted from ($C - \mathsf{RST}$), is out of the range of query, q, where C is the set of all encrypted data records.
4) *Data Privacy:* CS cannot learn the information about the data and the index encrypted by DO.
5) *Forward Privacy:* CS cannot learn the relationship between the newly updated records and the previous records by observing only the change of EDB. By observing the change of the index.

E. Security Definitions

1) Index Indistinguishability: A searchable encrypted index EDB is generated and stored in CS when DO outsources his/her database to CS. For security, the searchable index EDB does not reveal any information about the underlying data records. To be specific, we make the following distinguishing game: An adversary issues two random and different sets of data records that hold the same size, and outsources them to a challenger. The challenger randomly selects one of the two sets and generates the corresponding index EDB which will be outsourced to the adversary. After receiving the index, the adversary determines which set of data records relates to the index.

The game $\mathsf{IND-IDX}$ is described as follows.

1) *Setup:* The challenger $\mathcal{C}$ runs $\mathsf{Init}(1^\lambda)$ to generate two secret keys K and a state σ.
2) *Challenge:* The adversary $\mathcal{A}$ selects two random and distinct sets of data records s_0, s_1, from the original database where $|s_0| = |s_1|$. Then, $\mathcal{A}$ randomly selects a bit $b \in \{0, 1\}$ and sends s_b to $\mathcal{C}$. The challenger $\mathcal{C}$ runs $\mathsf{Update}(K, \sigma, s_b)$ to generate a corresponding index EDB_b. The index EDB_b is sent to the adversary $\mathcal{A}$.
3) *Guess*: $\mathcal{A}$ wins the game if the output b' satisfies that $b' = b$.

W_b is defined that $\mathcal{A}$ outputs b and wins the game.

Definition 1: Our EAFS achieves index indistinguishability in the aforementioned game if the advantage

$$\mathsf{Adv}^{\mathsf{IND-IDX}}_{\mathsf{EAFS},\mathcal{A}} = |\Pr[W_0] - \Pr[W_1]|$$

is negligible for any probabilistic polynomial-time adversary $\mathcal{A}$.

2) Forward Privacy: Forward privacy was proposed to guarantee that the updated documents do not reveal any information about the updated keywords [12]. To be specific, CS cannot tell whether or not an updated document matches a keyword that has been queried before. However, forward privacy only relates to the keyword search in SSE according to the previous work. It also should be achieved for the range search to guarantee that the newly updated data records have no relationship with the data in the database. We formally define the forward privacy for range search in the following.

Definition 2:A L-adaptively secure SSE scheme that supports range search in the encrypted database is forward private if the update leakage function L_{Update} can be written as

$$L_{\mathsf{Update}}(op, \mathsf{in}) = L'(op, m, n)$$

where in is the input of the update, m is the number of attributes, n is the number of data records of the input, and L' is stateless.

V. Preliminaries

A. Pseudorandom Function

A pseudorandom function is indistinguishable computationally from a random function. Given pairs $(x_1, f(k, x_1)), (x_2, f(k, x_2)), \ldots, (x_\tau, f(k, x_\tau))$, an adversary cannot predict $f(k, x_{\tau+1})$ for any $x_{\tau+1}$. We say that a function $f : \{0,1\}^\lambda \times \{0,1\}^s \rightarrow \{0,1\}^n$ is a (t, ε, q)- pseudorandom function if:

1) $f(k, x) \underline{\underline{\text{def}}} f_k(x)$ can be computed efficiently from key $k \in \{0,1\}^\lambda$ and input $x \in \{0,1\}^s$.
2) If every oracle algorithm, A, making at most q oracle queries and with running time at most t

$$\left| \Pr\left[A^{f(\cdot,k)} = 0 \mid k \xleftarrow{R} \{0,1\}^\lambda\right] - \Pr\left[A^g = 0 \mid g \xleftarrow{R} \{F : \{0,1\}^s \rightarrow \{0,1\}^n\}\right] \right| < \varepsilon.$$

B. Order Preserving Encryption

The confidentiality of data in an environment, in which users are not trusted, can be guaranteed by encryption. However, for processing some classes of queries, such as comparison and sorting, the ciphertexts must be decrypted. OPE is used primarily to issue SQL queries in encrypted databases. It enables the plaintexts and the corresponding ciphertexts hold the same order, i.e., $\mathsf{OPE.Enc}(x) > \mathsf{OPE.Enc}(y)$ iff $x > y$. Therefore, the untrusted entity can perform order operations in the ciphertext domain. The ideal security of OPE is defined as indistinguishable underordered chosen plaintext attack (IND-OCPA), which leaks nothing but the order.

C. Oblivious Transfer (OT)

OT is an important cryptographic tool used for secret exchange or other extended protocols. Two parties, i.e., a sender, S, and a receiver, R, participate in an OT function. The sender holds several secrets, and the receiver plans to get part of them. In such a function, some conditions must be satisfied, i.e., the receiver obtains only the secrets that he/she choses, and learns nothing about the rest of the secrets, and the sender learns nothing about which secrets the receiver has obtained.

K-out-of-n oblivious transfer (OT^k_n) is an extensively used type of OT protocol in which the receiver wants to get k ($k < n$) secrets simultaneously from the n secrets held by the sender. The OT^k_n can be described as follows:

1) Input:

S inputs n strings $x_1, x_2, \ldots, x_n \in \{0,1\}^m$ where $m \in N$. R inputs a set of selections $\omega \subseteq \{1, 2, \ldots, n\} = U$ where $|\omega| = k$.

2) Output:

S has no output. *R* outputs x_i where $i \in \omega$ and learns nothing about x_j where $j \in (U - \omega)$.

VI. Proposed Scheme

In this section, we firstly introduce the design rationale of the secure range search and the notations used in our scheme. Then, we present our proposed EAFS, which consists of four algorithms: initialization; update; generate trapdoor; and search. The notations we use in this article are given in Table II.

TABLE II
NOTATIONS AND DEFINITIONS

Notations	Definitions
D_{mn}	the database D with m attributes and n data records
d	the data record
$\mathcal{M}$	a map function
B_i	the identifier of the i^{th} bucket
cnt_i	the counter of B_i
$\|B\|$	the total number of buckets
σ	the state index stored in the client
EDB	the encrypted database in the server
K_{PRF}, K_{OPE}	the secret keys of PRF and OPE
F	a secure pseudo-random function
H_1, H_2	two secure hash functions
len	the bit length of an element
q	the search range of a query
$q.B$	the buckets mapped from q

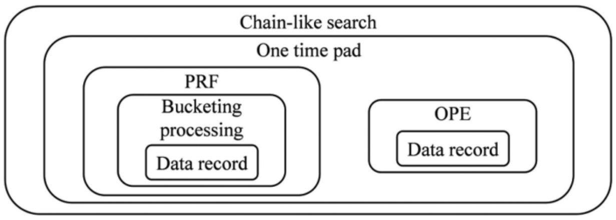

Fig. 4. Building blocks of our scheme.

A. Design Rationale

Based on the existing literature, we can conclude that a simple cryptographic tool or technology cannot guarantee the security, efficiency, and accuracy of range searches in encrypted databases simultaneously. Therefore, we build on the onions of encryption, which is a multilayer structure that stores multiple ciphertexts compactly. Each layer provides either the security property or the functionality property to inform the design of a secure range search. The building blocks of our scheme are summarized in Fig. 4. Note that the secure range search whose complexity relates to the size of database (i.e., the total number of data records) is impractical. The original data records should be processed (i.e., bucketing) to generate an index that helps to accelerate the search of the range. And a deterministic encryption (i.e., PRF) is needed to check the equality and protect the privacy of the index.

It is inevitable that some false positives will exist as a tradeoff for increasing the efficiency. A functional encryption (i.e., OPE) is needed to compare the ciphertexts. However, PRF and OPE are two weaker encryption schemes. The PRF reveals which encrypted values relate to the same plaintext, which can be easily compromised by the file-inject attack. The OPE reveals the order of the original data records. Therefore, a stronger encryption (i.e., one-time pad) is needed to guarantee the privacy of the range search.

Finally, the chain-like search implements the secure range search where the complexity only relates to the number of results

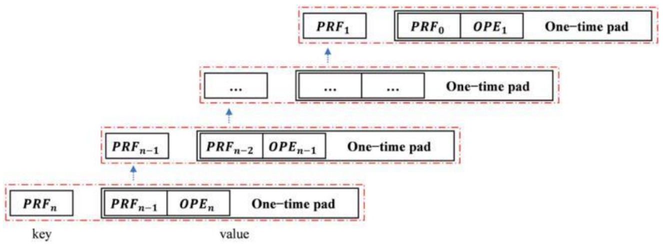

Fig. 5. Chain-like search.

```
Algorithm 1: Init.
  Input: $1^\lambda$
  Output: $\sigma$, EDB, K, F, $H_1$, $H_2$
  1: M ← map function
  2: $\sigma \leftarrow \{\langle B_1, 0\rangle, \langle B_2, 0\rangle, \ldots, \langle B_{|B|}, 0\rangle\}$
  2: EDB ← empty dictionary
  3: $K = (K_{PRF}, K_{OPE}) \leftarrow KenGen(\lambda)$
  4: $F : \{01\}^\lambda \times \{01\}^* \rightarrow \{01\}^\lambda$
  5: $H_1 : \{0,1\}^\lambda \rightarrow \{0,1\}^\lambda$,
     $H_2 : \{0,1\}^\lambda \rightarrow \{0,1\}^{(1+m)\lambda+len(ID(d))}$
```

satisfying the search request and some false positives during the entire search process, as well as ensuring the forward privacy. More precisely, the secure range search is implemented in a dictionary with key-value pairs, as shown in Fig. 5. The key is used for checking equality, and it also can be used to unmask the corresponding value encrypted by the one-time pad. The decrypted value helps to eliminate false positive results and contains the information about another key relating to the data record that satisfies the search range. Such iterative operations of searching look like a "chain."

B. Initialization

To achieve a secure range search, some parameters must be generated in the initialization stage. We assume that the database D_{mn} is to be outsourced to CS.

First, DO generates a map function M to preprocess the data records in D_{mn}. Without loss of generality, the map function is defined by equally dividing the range of each attribute into a fixed number of (e.g., f) continuous sub-ranges, which generates f^m buckets.

Then, to accomplish the secure range search, two indices must be generated. As mentioned above, the state index σ is initialized as $\{\langle B_1, 0\rangle, \langle B_2, 0\rangle, \ldots, \langle B_{|B|}, 0\rangle\}$, and the encrypted database EDB is initialized as $\emptyset$. By inputting the security parameter λ DO generates two secret keys, K_{PRF} and K_{OPE}, which are used to build the encrypted database EDB.

Finally, DO selects a secure pseudorandom function F and two secure hash functions, H_1, H_2, where $F : \{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^\lambda$, $H_1 : \{0,1\}^\lambda \rightarrow \{0,1\}^\lambda$, and $H_2 : \{0,1\}^\lambda \rightarrow \{0,1\}^{(1+m)\lambda+\text{len}(ID(d))}$. The terms $\{0,1\}^n$ and $\{0,1\}^*$ are the sets of binary strings with length n and all finite length, respectively. The term len($ID(d)$) is the length of the identifier of a data record.

```
Algorithm 2: Update.
  Input: K, $\sigma$, $d$; EDB
  Output: $\sigma'$; EDB'
  1: Data owner:
  2: $o_i \leftarrow$ OPE.Enc($K_{OPE}, d_i$)
  3: $B_j \leftarrow M(d_i)$
  4: $cnt_j \leftarrow Get(\sigma[B_j])$
  5: if $cnt_j = 0$ do
  6:     $F_{cnt_j} = 0^\lambda$, $F_{cnt_j+1} = F(K_{PRF}, B_j||cnt_j + 1)$
  7: else
  8:     $F_{cnt_j} = F(K_{PRF}, B_j||cnt_j)$, $F_{cnt_j+1} = F(K_{PRF}, B_j||cnt_j + 1)$
  9: end if
  10: $cnt_j + 1 \leftarrow \sigma'[B_j]$
  11: send
      $\{H_1(F_{cnt_j+1}), (F_{cnt_j}||o_i||ID(d_i)) \oplus H_2(F_{cnt_j+1})\}$ to server
  12: Cloud server:
  13: $EDB' = EDB \cup \{H_1(F_{cnt_j+1}), (F_{cnt_j}||o_i||ID(d_i)) \oplus H_2(F_{cnt_j+1})\}$
```

C. Update of an Encrypted Database

In order to update data, the following requirements must be met. First, the update for a data record only makes small changes on the index, which ensures the efficiency of the update. Second, the newly updated data record has no relationship with the previous entities in the encrypted database, which ensures forward privacy. Third, the sensitive information leaked from the index is minimized to the extent possible.

- *Step 1: Data preprocessing (lines 1 and 2 in Algorithm 2):* Given the updated data record $d_i = (d_{i,1}, d_{i,2}, \ldots, d_{i,m})$, DO runs OPE.Enc($K_{OPE}, d_i$) to generate the ciphertext $o_i = (o_{i,1}, o_{i,2}, \ldots, o_{i,m})$.
- *Step 2: Index generation (lines 3–8 in Algorithm 2):* DO runs $M(d_i)$ to map the data record d_i into one of the f^m buckets, denoted as B_j, and retrieves the corresponding counter $\sigma[B_j]$, denoted as cnt_j. The two PRF values, F_{cnt_j} and F_{cnt_j+1}, are computed as follows:
$$F_\tau = F(K_{PRF}, B_j||\tau)$$
where $F_0 = 0^\lambda$.
- *Step 3: Index update (lines 9–11 in Algorithm 2):* DO updates $\sigma[B_j]$ to $cnt_j + 1$ and outsources the key-value pair
$$\{H_1(F_{cnt_j+1}), F_{cnt_j}||o^i||ID(d^i) \oplus H_2(F_{cnt_j+1})\}$$
denoted as $\{K_i, V_i\}$ to CS. Finally, DO stores the updated state index σ' locally, and CS updates the encrypted database $EDB' = EDB \cup \{K_i, V_i\}$ for further secure range search. The whole process is summarized in the *Update* algorithm.

As mentioned above, the counter $\sigma[B_j]$ of the state index is changed into $cnt_j + 1$ and the encrypted database EDB is added a key-value pair only when updating a data record. The first requirement has been satisfied. Each data record d_i relates

Algorithm 3: GenTrapdoor.

```
Input: $K, \sigma, q$
Output: $T$
1: $q.B = \{B_{q,1}, \ldots, B_{q,s}, \ldots, B_{q,t}\} \leftarrow M(q)$
2: $\{cnt_{q,1}, \ldots, cnt_{q,s}, \ldots, cnt_{q,t}\} \leftarrow OT(\sigma; q.B)$
3: $\{o_q^{low}, o_q^{up}\} \leftarrow OPE.Enc(K_{OPE}, q)$
4: for $1 \le k \le t$ do
5:     if $cnt_{q,k} = 0$:
6:         $F_{cnt_{q,k}} = \perp$
7:     else
8:         $F_{cnt_{q,k}} = F(K_{PRF}, B_{q,k} || cnt_{q,k})$
9:     end if
10: end for
11: return $T = \langle \{o_q^{low}, o_q^{up}\}, \{F_{cnt_{q,1}}, \ldots, F_{cnt_{q,s}}\}, \{F_{cnt_{q,s+1}}, \ldots, F_{cnt_{q,t}}\} \rangle$
```

to a unique $B_j || cnt_j$, which means that there are no two identical key-value pairs in the EDB. The requirement for forward privacy has been satisfied. Furthermore, we mask the ciphertexts encrypted by OPE, which could leak the order information of data records, by one-time pad. The CSP only lean about the number of attributes of the database from the index, which satisfies the third requirement.

D. Generation of the Trapdoor

If DU generates a trapdoor, the state index will be needed as input because it holds the counters that represent the number of data records stored in every bucket, which is regarded as sensitive information about the database. The counters related to the search range are needed, but the counters out of the search range are exposed. If DO generates a trapdoor, he/she must know the search range, which also can be regarded as sensitive information about DU. Therefore, the following requirements should be satisfied. First, DU only obtains the counters that he/she needs, but learns nothing about the other ones in the state index. Second, DO knows nothing about the query issued by DU.

- *Step 1: Query preprocessing (lines 1 and 2 in Algorithm 3):* Given a query, q, DO runs $M(q)$ to map the query q into several buckets, denoted as $q.B = \{B_{q,1}, \ldots, B_{q,s}, \ldots, B_{q,t}\}$. It is clear that the false positives only exist in the buckets that intersect query q. All the data records stored in the buckets that are totally covered by the query q belong to the result set. Therefore, we can generate a two-level trapdoor to minimize the number of comparisons. We assume that $B_{q,k}$ is covered by the query q when $1 \le k \le s$ and that $B_{q,k}$ intersects query q when $s+1 \le k \le t$.

To improve the security of our scheme, DU runs a t-out-of-$|B|$ OT protocol to get the corresponding counters with DO. Specifically, DO takes the state index as input, and DU takes the $q.B$ as input. After the interactions between them, DO outputs nothing, and DU gets $\sigma[q.B] = \{cnt_{q,1}, \ldots, cnt_{q,s}, \ldots, cnt_{q,t}\}$, which satisfies the requirements mentioned above.

Algorithm 4: Search.

```
Input: $EDB, T$
Output: $RST$
1: $RST \leftarrow \emptyset$
2: for $1 \le k \le s$ do
3:     while $F_{cnt_{q,k}} \neq 0^\lambda$
4:         $V_\alpha \leftarrow EDB[H_1(F_{cnt_{q,k}})]$
5:         $F_{cnt_{q,k}-1} || p_\alpha || ID(d_\alpha) \leftarrow V_\alpha \oplus H_2(F_{cnt_{q,k}})$
6:         $RST \leftarrow Add(RST, ID(d_\alpha))$
7:     end while
8: end for
9: for $s+1 \le k \le t$ do
10:     while $F_{cnt_{q,k}} \neq 0^\lambda$
11:         $V_\alpha \leftarrow EDB[H_1(F_{cnt_{q,k}})]$
12:         $F_{cnt_{q,k}-1} || o_\alpha || ID(d_\alpha) \leftarrow V_\alpha \oplus H_2(F_{cnt_{q,k}})$
13:         if $o_q^{low} < o_\alpha < o_q^{up}$ do
14:             $RST \leftarrow Add(RST, ID(d_\alpha))$
15:         end if
17:     end while
18: end for
19: return $RST$
```

- *Step 2: Trapdoor generation (lines 3–11 in Algorithm 3):* For the query $q = \{\langle q_1.low, q_1.up\rangle, \ldots, \langle q_m.low, q_m.up\rangle\}$, where *low* and *up* respectively represent the lower bound and upper bound of an attribute in the query, DO encrypts it by computing

$$\{o_q^{low}, o_q^{up}\} \leftarrow OPE.Enc(K_{OPE}, q)$$

where $\{o_q^{low}, o_q^{up}\} = \{o_{q,1}^{low}, o_{q,1}^{up}, \ldots, o_{q,m}^{low}, o_{q,m}^{up}\}$, and the trapdoor is generated by computing

$$T = \langle \{o_q^{low}, o_q^{up}\}, \{F_{cnt_{q,1}}, \ldots, F_{cnt_{q,s}}\}, \{F_{cnt_{q,s+1}}, \ldots, F_{cnt_{q,t}}\} \rangle$$

denoted as $T = \langle T_1, T_2, T_3 \rangle$. The trapdoor T is outsourced to CS for further search on q. The entire process is summarized in the *GenTrapdoor* algorithm.

E. Secure Range Search

The following requirements should be satisfied when implementing the search process. First, the search efficiency relates to the size of results that satisfies the search request, which enables our scheme to be used in large-scale databases. Second, the output from the secure range search is correct and complete, which ensures the quality of the search. Third, there is no extra interaction between CS and DU or any post-process operation implemented by DU, which ensures that DU has a great experience.

- *Step 1: Checking equality:* Given the trapdoor, T, CS outputs the query results by searching in the encrypted database, EDB. For each PRF value $F_{cnt_{q,k}}$ in the trapdoor, where $1 \le k \le t$, CS computes $H_1(F_{cnt_{q,k}})$ and obtains the corresponding key-value pair $\{K_\alpha, V_\alpha\}$ in EDB, where

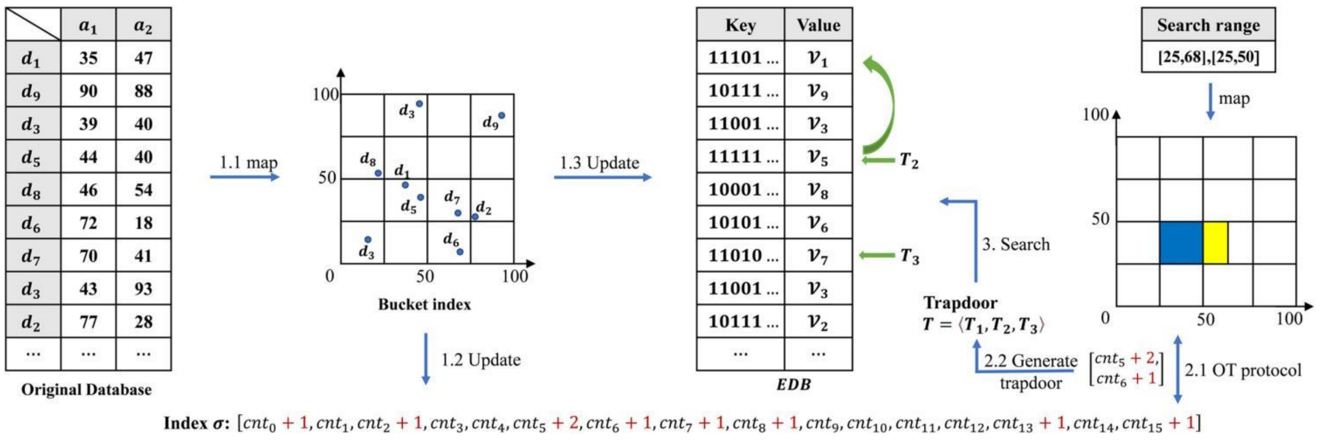

Fig. 6. Simplified example of EAFS.

$K_\alpha = H_1(F_{cnt_{q,k}})$ and $V_\alpha = F_{cnt_{q,k}-1} \| o_\alpha \| ID(d_\alpha) \oplus H_2(F_{cnt_{q,k}})$. Then, CS can unmask V_α using $H_2(F_{cnt_{q,k}})$. If $F_{cnt_{q,k}} \in T_2$, the identifier $ID(d_\alpha)$ will be added into the result set RST; otherwise the data record d_α must be estimated to determine whether or not it is a false positive by comparing o_α with T_1.

- *Step 2:* Chain-like search**:** CS obtains the PRF value $F_{cnt_{q,k}-1}$, which is related to a previous data record in the same bucket. $F_{cnt_{q,k}-1}$ can be regarded as the current entry in the search process. CS performs the above operations iteratively until the current entry is equal to 0^λ. The chain-like search is described as follows:

$$F_{cnt_{q,k}-1} \| o_\alpha \| ID(d_\alpha) \oplus H_2(F_{cnt_{q,k}}) \oplus H_2(F_{cnt_{q,k}})$$
$$F_{cnt_{q,k}-2} \| o_\beta \| ID(d_\beta) \oplus H_2(F_{cnt_{q,k}-1}) \oplus H_2(F_{cnt_{q,k}-1})$$
$$\cdots$$
$$0^\lambda \| o_\gamma \| ID(d_\gamma) \oplus H_2(F_1) \oplus H_2(F_1).$$

During the entire process, the *Search* algorithm takes the encrypted database EDB and the trapdoor T as input, and outputs a set of correct and complete results without any extra interaction or any post-process operation. The complexity of the search is only $O(\mu_Q + \mu_F)$, where μ_Q is the number of results that satisfy the query, and μ_F is the number of false positives.

Fig. 6 shows a toy example of our proposed EAFS. Let us assume that there are nine data records (i.e., d_1 to d_9) to be updated. Initially, the *Update* algorithm is run. The nine data records, the order of which can be random, are mapped into the bucket index. The counters in the index σ are updated, and nine key-value pairs of the encrypted EDB are generated. When DU wants to issue a query (i.e., $25 < a_1 < 68$ and $25 < a_2 < 50$), he/she runs the *GenTrapdoor* algorithm. The search range is mapped into the bucket index, and an OT protocol is needed to get the corresponding counters in σ. When running the *Search* algorithm, first, T_2 finds the last data record (i.e., d_5) located in the blue rectangle and iteratively finds the previous record in a privacy-preserving manner. The data record found by T_3 (i.e., d_7) must be estimated by comparing it with T_1 to determine whether or not it is in the yellow rectangle.

VII. Security Analysis

We demonstrate the security of our scheme by utilizing the simulation-based model that was proposed in [13]. In this model, first, we define the leakage function L, which indicates what is leaked to an adversary. To be specific, we define two leakage functions, i.e., L_1 and L_2, which represent the information leaked from the index and the queries in our scheme, respectively.

1) $L_1(D) = \{n, \text{len}(H)\}$, where n is the size of database D, and len(H) is the bit length of hash values with respect to the hash functions h_1 and h_2.
2) $L_2(D, q) = \{|T_2|, |T_3|, m, O(o)\}$. $|T_2|$ and $|T_3|$ are the numbers of elements in T_2 and T_3, respectively, which means the number of buckets covered by the search range and intersecting with the search range, m is the number of attributes, and $O(o)$ is the order information of the ciphertexts in a trapdoor.

Then, we define two games, i.e., a real game and a simulated game. The essential requirement of the real game is to execute the real-world algorithms proposed in our scheme. The essential role of the simulated game is to attempt to achieve the secure range search by a simulator given ideal conditions. The simulator only knows the leakage function. The two games, i.e., $\text{Real}_\pi^A(1^\lambda)$ and $\text{Sim}_{L,S}^A(1^\lambda)$, are defined as follows.

1) $\text{Real}_\pi^A(1^\lambda)$: The game is implemented by the interactions between a challenger C and an adversary A. C runs $\text{Init}(1^\lambda)$ to generate some parameters, and A selects a subset D' of D and sends it to C. The challenger C runs $\text{Update}(K, \sigma, D'; \text{EDB})$ and outputs EDB' to A. Then, the adversary A adaptively issues a polynomial number of search requests, i.e., $Q = (q_1, q_2, \ldots)$. For each query in Q, the challenger C runs *GenTrapdoor*(K, σ, q) and outputs the trapdoor T to A. Finally, A returns a bit b as the output of the experiment $\text{Real}_\pi^A(1^\lambda)$.

2) $\mathsf{Sim}^{\mathcal{A}}_{L,\mathcal{S}}(1^\lambda)$: The game is implemented by the interactions between a simulator $\mathcal{S}$ and an adversary $\mathcal{A}$. The adversary $\mathcal{A}$ selects a subset D' of D and sends it to $\mathcal{S}$. The simulator $\mathcal{S}$ generates the index EDB by $L_1(D')$ and sends it to $\mathcal{A}$. The adversary $\mathcal{A}$ adaptively issues a polynomial number of search requests, i.e., $Q = (q_1, q_2, \ldots)$. For each query in Q, the simulator $\mathcal{S}$ generates the trapdoor T by $L_2(D', q)$ and sends it to $\mathcal{A}$. Finally, $\mathcal{A}$ returns a bit b as the output of the experiment $\mathsf{Sim}^{\mathcal{A}}_{L,\mathcal{S}}(1^\lambda)$.

Definition 3: The EAFS scheme π is L-secure against adaptive attacks if, for any polynomial-time adversary $\mathcal{A}$, there exists an efficient polynomial-time simulator, $\mathcal{S}$, such that

$$\left|\Pr\left[\mathsf{Real}^{\mathcal{A}}_{\pi}\left(1^\lambda\right) = 1\right] - \Pr\left[\mathsf{Sim}^{\mathcal{A}}_{L,\mathcal{S}}\left(1^\lambda\right) = 1\right]\right| \leq \mathsf{negl}\left(1^\lambda\right)$$

where $\mathsf{negl}(1^\lambda)$ is a negligible function.

First, we demonstrate that our EAFS scheme achieves forward privacy if $\mathcal{H}_1$ and $\mathcal{H}_2$ are secure, one-wayness hash functions. As discussed in Section V, a key-value pair is added to the encrypted database EDB, when a data record is updated. The newly updated key-value pair has no relationship with the previous data records before searching. The key is a hash value of $\mathcal{H}_1$, and the corresponding value is masked by the hash value of $\mathcal{H}_2$, which can be regarded as one-time padding. Therefore, the leakage function of update L_{Update} can be formalized as follows:

$$L_{\mathsf{Update}}\,(\mathsf{op}, \mathsf{in}) = L'\,(\mathsf{op}, m, n)$$

where m is the number of attributes, n is the number of data records of the input in, and L' is stateless. We obtain the following theorem.

Theorem 1: Our EAFS scheme achieves forward privacy if $\mathcal{H}_1$ and $\mathcal{H}_2$ are secure, one-wayness hash functions.

Then, we prove that the encrypted database EDB of two databases with the same size are computationally indistinguishable if our EAFS scheme is forward secure. A forward secure SE scheme guarantees that the newly updated entry has no relationship with the previous index. The EDB is a dictionary that essentially is built by a large amount of update operations. Therefore, each entry in the index is mutually independent before searching. Given the size of the database, the simulator can simulate a random index that has the identical structure and the same size as the real index. Thus, we obtain the following theorem.

Theorem 2: The encrypted database EDB of two databases with the same size is computationally indistinguishable if our EAFS scheme is forward secure.

Finally, we prove that the trapdoor achieves semantic security if $\mathcal{F}$ is the secure pseudorandom function and that the OPE is IND-OCPA. The Trapdoor T consists of two parts. The first part T_1 includes the search ranges that were encrypted by OPE. And the second part, i.e., T_2 and T_3, includes some combinations of the bucket identifiers and corresponding counters that are computed by PRF. Given the size of the secure parameter, λ, the simulator can simulate a random trapdoor that is distinguishable from the real one. We obtain the following theorem.

Theorem 3: Trapdoor achieves semantic security if $\mathcal{F}$ is pseudorandom function and the order preserving encryption is IND-OCPA.

According to the defined leakage functions, if there exists an efficient polynomial-time simulator $\mathcal{S}$ such that, for any polynomial-time adversary $\mathcal{A}$, the outputs between $\mathsf{Real}^{\mathcal{A}}_{\pi}(1^\lambda)$ and $\mathsf{Sim}^{\mathcal{A}}_{L,\mathcal{S}}(1^\lambda)$ are computationally distinguishable, we obtain the following theorem.

Theorem 4: Our EAFS scheme is L-secure against adaptive attacks if $\mathcal{H}_1$ and $\mathcal{H}_2$ are secure, one-way hash functions, if $\mathcal{F}$ is a pseudorandom function, and if the order preserving encryption is IND-OCPA.

VIII. Evaluation of Performance

In this section, we evaluate the performance of our scheme. Since the computational cost of a secure range search is correlated highly with the distribution of the data, we used both an artificial dataset, i.e., a manmade uniform distribution dataset namely dataset 1, and an actual dataset, i.e., the diabetes dataset[1] in the UCI Machine Learning Repository, namely dataset 2. Fig. 7 shows the distribution of dataset 2. More precisely, the three subfigures in Fig. 7 present the distribution of the each dimensional data in dataset 2. The entire process of our scheme is coded using the Python programing language. DO and DU were simulated by our PC. To ensure that our scheme is practical, we used the real CS of Microsoft Azure to store the encrypted database and perform the secure range search. Their configurations were as follows.

1) PC: 2.3 GHz two-core Intel Core i5 8 G RAM
2) CS: Intel Xeon Platinum 8171M CPU @ 2.60 GHz 8 GB RAM

Considering the tradeoff between security and efficiency in our scheme, the security parameter λ was set to be 1024-bit. If there was no explicit statement to the contrary, the number of attributes, m, was equal to 2. We compared our scheme with the state-of-the-art work proposed in [4], which is very efficient but has false positives, to demonstrate that our scheme can achieve better performance.

Fig. 8 shows the relationship between the size of the index (aka., EDB) and the number of data records. The index stored in CS is a dictionary with a key-value pair that is computed by the secure hash functions. The length of the dictionary relates to the number of data records in the database. Therefore, the size of index increased linearly as the number of data records increased. As the dimensions increased, more ciphertexts encrypted by OPE would be generated. The size of the index increased linearly as the number of dimensions increased.

Fig. 9 shows the relationship between the computational cost for secure range search and the number of data records. As mentioned above, the search complexity of the EAFS is $O(\mu_Q + \mu_F)$, where μ_Q is the number of data that satisfy the query, and μ_F is the number of false positive data. We set the number of $(\mu_Q + \mu_F)$ as almost two percent of the total number of the dataset when the number of attributes was $m = 2$ in

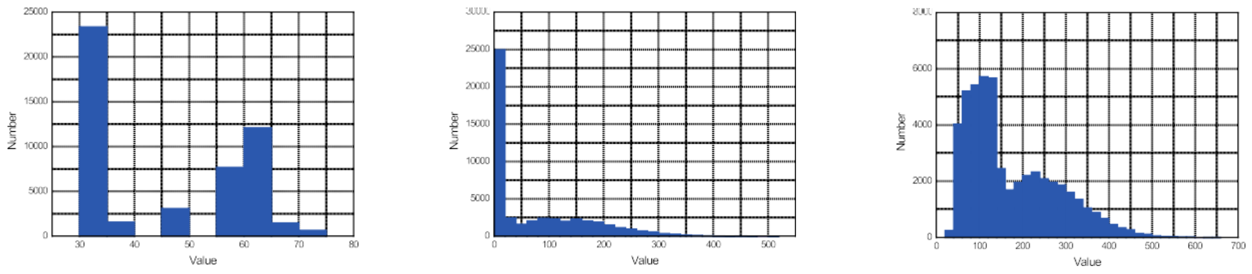

Fig. 7. Data distribution.

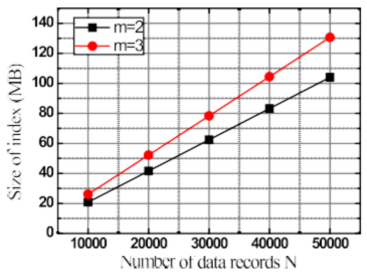

Fig. 8. Size of index.

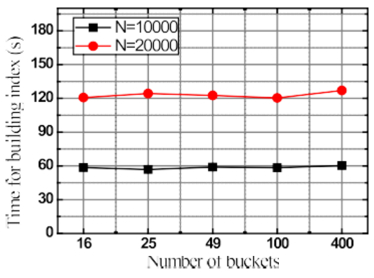

Fig. 11. Computation cost for building index.

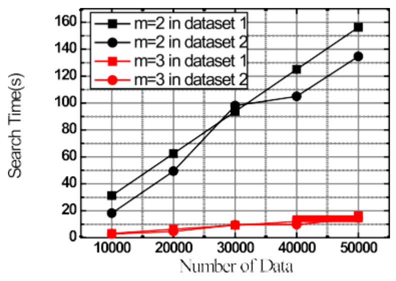

Fig. 9. Computation cost for different number of data records.

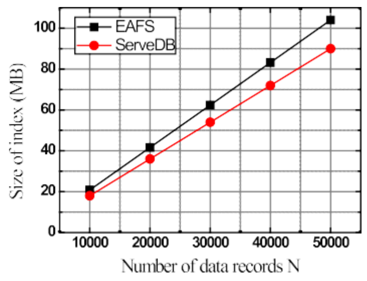

Fig. 12. Comparison of the index size.

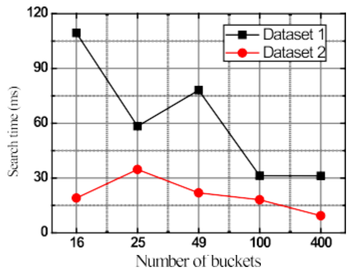

Fig. 10. Computation cost for different number of buckets.

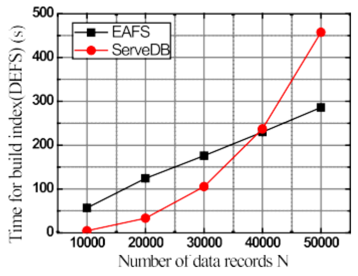

Fig. 13. Comparison of the time for building index.

database 1. When the number of attributes was increased to 3, one more restriction was generated, and less data records can satisfy the search request. Therefore, the search time when $m = 3$ is less than the search time when $m = 2$.

Figs. 10 and 11 show the relationship between the computation cost of our scheme and the total number of buckets, |B|. If the total number of buckets is small, which means that the range of each bucket is wide, more false positive data records are searched for the same search token, so, subsequently, the search time will increase. Fig. 10 shows the search time of a different number of buckets for the two datasets. As mentioned above, the length of the index of the EAFS is related to the number of data records in the database. Therefore, by fixing the number of data records and the number of attributes, the time for building the index has no relationship with the total number of buckets, as illustrated in Fig. 11.

We compared our scheme with ServeDB [4]. Figs. 12 and 13 show the comparison of the index between our scheme and the

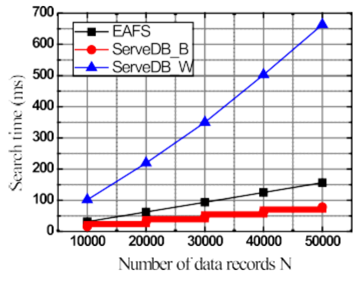

Fig. 14. Comparison of computation cost for different number of data records.

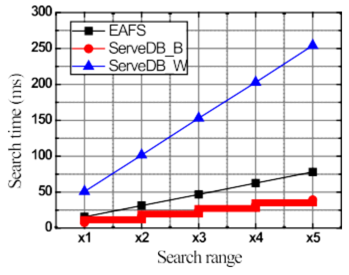

Fig. 15. Comparison of computation cost for different search range.

ServeDB. Fig. 12 shows that the size of the index of the EAFS increased linearly as the number of data records increased. Wu *et al*. [4] proposed a tree-based index that they called SVETree. Each node of the SVETree contains a bloom filter. In ServeDB, the data record is mapped into one of several cubes and then mapped into a bloom filter. This results in many duplicated cubes being mapped into the bloom filter. The length of the bloom filter relates to the total number of cubes, which is a constant. Therefore, since there are $2n\_1$ bloom filters in a SVETree when the number of data is n, the index size of ServeDB increased linearly as the number of data records increased. And the index sizes of ServerSB and EAFS are close.

However, the time required to build the index of ServeDB relates to the number of data records and the number of nodes in the SVETree. Fig. 13 shows that it increased super-linearly as the number of data records increased.

Figs. 14 and 15 describe the comparison of the computational cost when performing secure range search in the two schemes. To ensure a clear comparison, the two schemes were compared using an ideal dataset (i.e., dataset 1). In ServeDB, the results are generated by searching in the SVETree, which is a balanced binary tree. If all of the data records are ordered, searching for a continuous range only requires comparing several nodes in the tree. This can be regarded as the best case for secure range search in the ServeDB, as denoted by ServeDB_B. However, the ordered data records will expose their relationships with each other (i.e., the size of data stored in the right subtree is bigger than the data in the left tree). To eliminate leakage, the data records in the SVETree must be in a random order. Thus, searching for data with random order in the SVETree can be regarded as the worst case, as denoted by SeverDB_W. The search complexity with worst case is $\mu_Q O(\log(n))$. From Fig. 14, we can conclude that the computational cost of our scheme is less than that of ServeDB. Fig. 15 shows that the computational cost of the two schemes increased linearly as the number of the search range increased.

IX. Conclusion

We proposed the definition of forward privacy in range search, and designed an EAFS SE scheme that supports secure range search in encrypted databases. The computational cost for a range search in our EAFS only relates to the number of results that satisfy the query and the number of false positives in the intersecting buckets. The chain-like search ensures that any newly updated data entities have no relationship with the previous data, which achieves forward privacy. We also evaluated the security and performance of our approach to demonstrate the utility of the EAFS.

References

[1] Y. Zhang, J. Katz, and C. Papamanthou, "All your queries are belong to us: The power of file-injection attacks on searchable encryption," in *Proc. USENIX Secur. Symp.*, 2016, pp. 707–720.

[2] H. Hacigümüs, B. Iyer, C. Li, and S. Mehrotra, "Executing SQL over encrypted data in the database-service-provider model," in *Proc. Int. Conf. Manag. Data Conf.*, 2002, pp. 216–227.

[3] B. Hore, S. Mehrotra, M. Canim, and M. Kantarcioglu, "Secure multidimensional range queries over outsourced data," *VLDB J.*, vol. 21, no. 3, pp. 333–358, 2012.

[4] S. Wu, Q. Li, G. Li, D. Yuan, X. Yuan, and C. Wang, "ServeDB: Secure, verifiable, and efficient range queries on outsourced database," in *Proc. IEEE 35th Int. Conf. Data Eng.*, 2019, pp. 626–637.

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

[6] C. Mavroforakis, N. Chenette, A. Neill, G. Kollios, and R. Canetti, "Modular order-preserving encryption, revisited," in *Proc. SIGMOD Int. Conf. Manag. Data Conf.*, 2015, pp. 763–777.

[7] K. Xue, S. Li, J. Hong, Y. Xue, N. Yu, and P. Hong, "Two-cloud secure database for numeric-related SQL range queries with privacy preserving," *IEEE Trans. Inf. Forensics Secur.*, vol. 12, no. 7, pp. 1596–1608, Jan. 2017.

[8] K. Cheng *et al.*, "Strongly secure and efficient range queries in cloud databases under multiple keys," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 2494–2502.

[9] W. Wong, B. Kao, D. W. Cheung, R. Li, and S. Yiu, "Secure query processing with data interoperability in a cloud database environment," in *Proc. Int. Conf. Manag. Data*, 2014, pp. 1395–1406.

[10] R. Li, A. Liu, A. Wang, and B. Bruhadeshwar, "Fast and scalable range query processing with strong privacy protection for cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 4, pp. 2305–2318, Aug. 2016.

[11] I. Demertzis, S. Papadopoulos, O. Papapetrou, A. Deligiannakis, M. Garofalakis, and C. Papamanthou, "Practical private range search in depth," *ACM Trans. Database Syst.*, vol. 43, no. 1, pp. 1–52, 2018.

[12] R. Bost, "Ζοφος: Forward secure searchable encryption," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2016, pp. 1143–1154.

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