A Privacy-preserving Multi-pattern Matching Scheme for Searching Strings in Cloud Database

Meiqi He, Jun Zhang, Gongxian Zeng, Siu Ming Yiu
Department of Computer Science
The University of Hong Kong
Email: {mqhe, jzhang3, gxzeng, smyiu}@cs.hku.hk

***Abstract*—Searching encrypted database is an important topic as more users want to leverage a third-party cloud system to store and process their data in encrypted form. Despite a lot of wonderful results, there are still a number of unsolved problems. In particular, the problems of pattern matching (not keyword search), e.g. with wildcards, that supports secure boolean queries and how to determine the value of k automatically of a top-k search for different queries on encrypted data are not properly addressed. In this paper, we provide solutions to solve these problems. Also, most existing secure databases employ different encryption functions to support different operators. The only exception is SDB (SIGMOD'2014) that was designed to support data interoperability between integers with a unified encryption scheme so that sophisticated queries can be answered by the database. However, SDB does not support string matching queries. We show that our solutions can be made compatible with SDB to fill this gap. To the best of our knowledge, we are the first to investigate these problems. We prove that our scheme is secure against chosen query attack. We have evaluated the performance of our scheme on large (10^5 strings) real-world datasets, and showed that our scheme can achieve a high search quality of 99.9% recall and 98.6% accuracy with reasonable response time.**

I. Introduction

Advancement in cloud technology has facilitated data owners to outsource their huge amount of data onto third-party cloud servers to save storage and computational cost. In a cloud database system, which deploys Database-as-a-service model (DBaaS), a data owner (DO) uploads its database to a service provider (SP) and SP hosts high performance machines and sophisticated database software to process queries on behalf of the DO. However, outsourcing creates the security problem of the data. An intuitive solution is to encrypt the data before outsourcing, which makes the data processing by cloud a challenging task.

Secure database: Security issues in cloud databases have been considered in [1]–[4]. Both [1], [2] focus on developing access control mechanisms to prevent information leaks caused by outside attackers. Another issue, probably a more important issue for applications, is how to execute SQL operators over encrypted data. CryptDB [3] and MONOMI [4] are two well-known systems for processing SQL queries on encrypted data. However, both of them employ different encryption functions to support different operators. The same data item is encrypted multiple times. The supported operators are thus not data interoperable. In [5], Wong *et al.* proposed a secure query processing system SDB which can support data interoperability between integers using one unified encryption scheme. Data interoperability allows a wide range of SQL queries to be processed by the SP on encrypted information. However, SDB is limited in the domain of integers. SQL functions on other data types such as SQL-like operators for string matching are not feasible in SDB.

Keyword search over encrypted data: For string matching on encrypted data, searchable symmetric encryption (SSE) has received much attention from researchers in recent years [6]–[10]. All the keyword search schemes require clients to use the whole keywords as input. Pattern matching with substrings and/or with complex matching rules (e.g. wildcards) are not supported. Despite its importance, pattern matching over encrypted data has not been sufficiently investigated and many problems remain unresolved in the literature.

Multi-pattern matching: Pattern matching [11]–[13] is a more difficult problem and does not have a breakthrough until recently. In 2015, Wang *et al.* [12] designed a general scheme for pattern-matching on ciphertext stored in a cloud server. The high level idea is as follows. They transformed the matching problem into a vector-matrix multiplication problem (see the details in Section II). They abstract and represent each string (to be searched or indexed, they call it *index string*) using a *fingerprint vector*, which captures the characteristics of the pattern for the string while the matching rule is represented by a weighted matrix. By multiplying the vector with its weight matrix and comparing the weighted Euclidean distance between the query vector and the index vectors, the algorithm produces top-k (for a fixed k) index strings as a result of the query. The underlying cryptographic technology is based on the secure kNN computation [14].

However, the support for multi-pattern query still remains unsolved in their work while it is an essential problem in practice, especially in SQL query (AND, OR). In standard SQL grammar, the AND & OR operators are used to filter records based on more than one condition. In reality, there are boolean queries like: SELECT $*$ FROM users WHERE Country LIKE 'Germ%' AND City Like '%erli%'. This boolean query aims at retrieving users coming from Berlin, Germany. Previously, to solve such a multi-restriction search problem, DO enables the server to perform search for every individual condition. For every restriction, the server finds

the set of documents/records that satisfy that condition, then returns, for example, the intersection of all those sets for an AND operation. Obviously, this approach allows the server to learn some extra information in addition to the results of the conjunctive (disjunctive) query, i.e., the server can observe which documents contain each individual pattern. Over a period of time, the server can combine this information to infer information about the user's documents.

In the field of SSE, the secure multi-*keyword* search problem has been widely investigated recently [15]–[17]. [15] requires the clients to decrypt the encrypted IFV (inverted file identifier vector) returned by the server to obtain the eligible file identifiers locally, and ask the server for this set of documents. [16] applied the well-known TF×IDF rules in plaintext information retrieval to count the total occurrence of keywords (in the query) in each document. However, counting an aggregate score induces more false positives and false negatives caused by some extremely high or low scores.

On the other hand, multi-*pattern* queries differ a lot from multi-*keyword* queries. In SSE, distinct keywords are well-defined. Documents are split into distinct keywords, which are encrypted and indexed independently. In pattern matching, any sub-strings (and may include wildcards) can be targets for searching. Unlike keywords, sub-strings to be searched may not be disjoint, i.e., sub-string are not independent, one can include the other. How one can split strings into sub-patterns for indexing is not trivial and techniques developed in SSE cannot be applied to solve multi-pattern queries directly.

Top-k vs threshold-based search: Top-k based search and threshold-based search are two kinds of pruning strategies or criteria for returning the results, used in the similarity search problem. Each has its own set of pros and cons. For threshold-based search, the user would have a good expectation of how similar the returned results to the query will be. Threshold-based searching is less expensive than top-k search since ranking is not needed. On the other hand, top-k is more suitable for the scenario (*e.g.* web search) where the user, who is not familiar with the system, does not know the best threshold in advance. Unlike threshold-based search, for the top-k search, the number of correct answers for different queries can be quite different (*e.g.* common pattern such as "tion" and rare pattern "ural"), how to set the k value may be difficult for users who have no background knowledge of the database. Very few previous works, if there is any, can support both strategies. Existing studies on top-k based search [18]–[20] always choose a fixed k value for every query (*e.g.* retrieve the top 10 index strings as the results, which may not be appropriate).

Difficulties and contributions: We target to solve the following two technical problems: (i) secure pattern matching that supports boolean query; and (ii) design a mechanism to enable the system to set the value of k in the top-k search automatically for single query. And we want to make sure that our solution is compatible to SDB. We adopt the fingerprint extraction algorithm in [12] to our scheme as to transform strings into vectors and then encrypt the values in SDB system. Server executes the data interoperability protocols to compute similarity scores between the string pattern and each index strings based on the encrypted database.

To tackle Problem (i), our main idea is to combine multi-dimensional range query and threshold-based similarity search to help performing conjunctive (and disjunctive) comparison. Take an AND query of two patterns (sp_1, sp_2) as an example, our objective is to decide if "$dis(s, sp_1) \leq threshold$ AND $dis(s, sp_2) \leq threshold$" for a given string s, where $dis()$ is the distance between the two input strings. For Problem (ii), we adopt the selective estimation technique [21] and design a novel scheme to enable the system to automatically choose the appropriate parameter for top-k query.

To summarize, the followings highlight the contributions of our work:

- We propose a secure multi-pattern matching scheme for searching strings in a secure database system, which can be easily integrated into SDB. To the best of our knowledge, we are the first to investigate the problems of secure pattern matching that support boolean query.
- For single query, we design a new operator - sorting for SDB to support both threshold-based similarity search and top-k similarity search and can automatically choose the appropriate parameter for top-k search.
- We present a thorough security analysis of our construction under the non-adaptive chosen query attack (CQA) model.
- We have conducted a set of experiments against a large dictionary dataset. Our experimental results show that our schemes can achieve high search quality in terms of $recall$ and $accuracy$.

II. Problem statement and preliminaries

A. Problem definition

In this paper, we study the secure boolean pattern matching problem in SDB system. Consider a sensitive column $D = \{s_1, \cdots, s_n\}$ as a set of original strings collected by the DO. After encrypting and outsourcing D on SP, given a query $\mathbf{q} = \{sp_1, \cdots, sp_m\}$. The objective is to find strings in D that contains all $sp_i \in \mathbf{q}$ without leaking individual results. For each pattern in $\mathbf{q}$, we consider three types of matching problems: substring matching ($\%str\%$),prefix matching $str\%$ and suffix matching ($\%str$). For single query, we target to design sorting protocol for SDB to support ranking and algorithms to automatically choose the appropriate parameter for top-k search.

B. Preliminaries

1) Fingerprint Extraction Algorithm: We extend the fingerprint extraction algorithm in [12] to solve our problem and make it compatible to SDB while still maintaining high search quality. In this subsection, we first review how to construct the fingerprint vector. The algorithm takes an input string str and outputs a fingerprint vector V. str can be either an index string or a query pattern. V is generated based on n-gram Counting

Order of str (see below). The concept of n-gram is widely used in text processing and categorization [22]–[24].

Definition II.1. *(n-gram list L^n_{str}): The n-gram list of a string str are $str[i, i+n-1]$ with $1 \leq i \leq |str| - n + 1$.*

An n-gram is an n-character sub-string of a longer string. For example, the word "sequence" has seven 2-gram: "{se, eq, qu, ue, en, nc, ce}". Given a string str with length l_{str}, it has $(l_{str} - n + 1)$ n-grams. The n-Gram Counting Order algorithm GCO(L_{str}, N, sk) takes str as input and generates a vector using a hash function. The i-th element of the vector is computed as the number of n-grams whose hash values modulo N will give the value of i (i.e., $V^n_{\mathsf{GCO}}(i) = |\{g \in L^n_{str} | h_{sk}(g) \mod N == i\}$, where h_{sk} is a cryptographic hash function with the secret key sk and N is to control the collision probability). In our method, we use a pseudo-random function to replace the hash function.

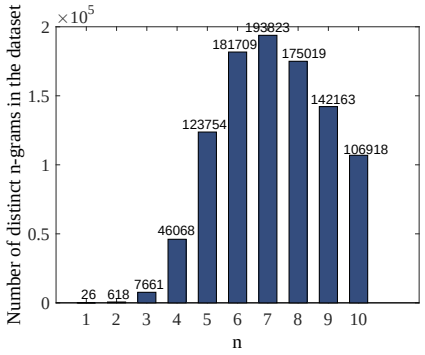


Fig. 1. The number of different n-grams within a dataset (#string = 100000)

In general, for string search that uses n-gram, the longer the gram length is, the more accurate the search will be. However, from the statistic analysis shown in Figure 1, we can conclude that when computing hash value, longer n-gram requires much larger N (modulo in GCO($\cdot$)) to decrease hash collision, thus incurring higher complexity of search. According to [25], it has been observed that the gram length of 3 is accurate enough for string search. Therefore, we also consider gram length up to 3.

The following algorithm shows the detailed process of fingerprint vector extraction. $\mathsf{FVE}(\cdot)$ algorithm generates a vector V, which is consist of 3 parts: V_{cnt}, $V_{\mathsf{prefixpos}}$, $V_{\mathsf{suffixpos}}$. $V_{\mathsf{prefixpos}}, V_{\mathsf{suffixpos}}$ representing the first (last) three position information of n-grams in L_{str} and will be used when answering prefix (suffix) matching queries. For instance, for prefix matching, the n-grams appearing at the front will be assigned greater weight.

$V \leftarrow \mathsf{FVE}(str, sk_f)$:

1. For each $n \in [1, 3]$, generate n-gram list of str denoted as L^n_{str};
2. For each n-gram list $L^n_{str}, n \in [1, 3]$:
   - Compute the n-gram counting order vector: $V^n_{\mathsf{GCO}} = \mathsf{GCO}(L_{str}, N, sk_f)$;
   - Let d_n denote the size of V^n_{GCO}. Define two vector $V^n_{\mathsf{prefixpos}}$ and $V^n_{\mathsf{suffixpos}}$ of size d_n.
   - For $i \in [1, d_n]$, d_n denote the size of V^n_{GCO}:
     (i) If $V^n_{\mathsf{GCO}}(i) \leq CT$, $V^n_{cnt}(i) = V^n_{\mathsf{GCO}}(i)$; Else $V^n_{cnt}(i) = CT$, CT is a gram counting threshold, which is introduced to cap the value of elements in $V^n_{\mathsf{GCO}}(i)$.
     (ii) If $pos(i) \in \{0, 1, 2\}$, $V^n_{\mathsf{prefixpos}}(i) = \mu^2 \times \mu^{-pos(i)}$; Else if $pos(i) \in \{l_{str}-3, l_{str}-2, l_{str}-1\}$, $V^n_{\mathsf{suffixpos}}(i) = \mu^2 \times \mu^{pos(i)-l_{str}+1}$; Else $V^n_{\mathsf{prefixpos}}(i) = 0$, $V^n_{\mathsf{suffixpos}}(i) = 0$. $pos(i)$ is the first position of the n-gram in L^i_{str} that is hashed into $V^n_{\mathsf{GCO}}(i)$, l_{str} is the length of str, μ is used to adjust the weight of position information, which will be helpful in prefix and suffix matching.

Obtain $V_{cnt} = \mu^2 \times (V^1_{cnt}, V^2_{cnt}, V^3_{cnt})$, $V_{\mathsf{prefixpos}} = (V^1_{\mathsf{prefixpos}}, V^2_{\mathsf{prefixpos}}, V^3_{\mathsf{prefixpos}})$ and $V_{\mathsf{suffixpos}} = (V^1_{\mathsf{suffixpos}}, V^2_{\mathsf{suffixpos}}, V^3_{\mathsf{suffixpos}})$.

3. Output $V = (V_{cnt}, V_{\mathsf{prefixpos}}, V_{\mathsf{suffixpos}})$.

2) SDB encryption and computation: **Encryption and Decryption:** The DO maintains two secret numbers g and n. The number n is the product of two big random prime numbers. The number g is a positive number that is co-prime with n. Given a row id r and a column key $ck = \langle m, x \rangle$, the item key v_k is given by, $v_k = \mathsf{SDB.Gen}(r, \langle m, x \rangle) = mg^{(rx \mod \phi(n))} \mod n$. Given a sensitive value v which is an integer and its item key v_k, the encrypted value v_e is given by $v_e = \mathsf{SDB.Enc}(v) = vv_k^{-1} \mod n$. With v_e and v_k, we can recover v, $v = \mathsf{SDB.Dec}(v_e) = v_e v_k \mod n$.

Secure operators: The data model of SDB is column-based. An operator takes one or more columns as input and produces one or more columns as output. In SDB, *multiplication, key update, addition, subtraction and comparison* are supported. Readers can refer to [5] for details of the respective protocols.

New Operator - Sorting: In original SDB, the comparison between columns can be executed, while the comparison between the values in different rows has not been achieved, neither the sorting function. Since encrypted fingerprint vectors of each string is stored in different rows and the computed distances are thus in a column. In order to offer top-k based search, we have to implement a sorting protocol to make ranking possible. In this section, we will explain how to achieve this purpose.

As in the SDB encryption scheme, each item has an item key, and the item key is generated by the column key and the row-id. For the column of score we want to sort, the values are encrypted by the same column key $ck = \langle m, x \rangle$ and different row-id r. The item key v_k is given by $v_k = mg^{(rx \mod \phi(n))} \mod n$. From the equation, we can perform a transformation, the item key can be seen as the output of the row x and column key $ck' = \langle m, r \rangle$. Therefore, an n column data can be transformed into a $1 \times n$ row data, without resulting much more cost. Thus the sorting operation can be achieved.

C. Framework

In our method, we firstly transform strings into fingerprint vectors V by a fingerprint extraction algorithm called $\mathsf{FVE}(\cdot)$ and then encrypt the values in SDB system.

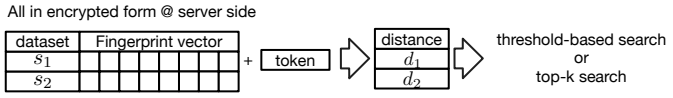


Fig. 2. search procedure

As shown in the figure 2, given a query **q**, the remote server executes the data interoperability protocols to compute similarity scores tween the each string pattern $sp \in q$ and each index strings ($s_i \in D$) based on Weighted Euclidean Distance using encrypted database. Weighted Euclidean Distance serves as a metric which can compare the the matching degree between two fingerprint vectors: $Score(s_i, sp) = dis(V_{s_i}, V_{sp}) = dis(H_1V_{s_i}, H_2V_{sp}) = \sum_{i=1}^{d}(h_{1i}V_{s_i}[j] - h_{2i}V_{sp}[j])^2$, where d is the size of fingerprint vectors, H_1 and H_2 are two weight vectors to specify which elements in the vectors need to be compared with and how much weight they have (see Section IV-B for details). When $m > 1$ we adopt threshold-based search as pruning strategies to select the conjunctive answer set. When $m = 1$, we support both threshold-based search and top-k search. Formally, our secure pattern matching system supporting boolean query consists of the following polynomial-time algorithms:

$sk \leftarrow \mathsf{KeyGen}(1^\lambda)$: is a probabilistic key generation algorithm run by the client. It takes as input a security parameter λ and outputs the secret key sk.

$V \leftarrow \mathsf{FVE}(sk, str)$: is an algorithm run by the client. It takes an input string str and outputs a fingerprint vector V. str can be either an index string or a query pattern.

$\Delta \leftarrow \mathsf{Enc}(sk, D, \{V_1, \cdots, V_{\#D}\})$: is an algorithm run by the client. It takes as input a secret key sk and a string collection D, and outputs an encrypted database Δ, which contains a set of ciphertexts **c** with respective encrypted fingerprint vectors $\mathbf{V}_e$.

$s \leftarrow \mathsf{Dec}(sk, c)$: is a deterministic algorithm run by the client. It takes as input a ciphertext c, and outputs a string s.

$\delta \leftarrow \mathsf{Preprocess}(sk, D)$: is a protocol run on both the client side and the server side to generate auxiliary data structure for the follow-up functions, including:

- $\mathsf{Est.preprocess}(D)$: is a protocol run on both the client side to generate auxiliary data structure for $\mathsf{EstK}(\cdot)$.
- $\mathsf{BQS.Setup}(\lambda)$: is a protocol run on both the client side and the server side to generate auxiliary data structure for the boolean query function.

$c_q \leftarrow \mathsf{Search}(sk, q, \Delta, \delta)$: is an interactive protocol run between the client and the server. The client side takes as input a query **q** while the server side takes as input the encrypted database Δ. Upon completion of the protocol, the client obtains a sequence of ciphertexts c_q. Also, there are several sub-protocols included in Search:

- $\mathsf{SDB.operator}(sk, \Delta) \rightarrow \Delta$: denotes the set of protocols in SDB system that take the key sk and encrypted database Δ as input and output the computation results.
- $k \leftarrow \mathsf{EstK}(q, \delta)$: is an algorithm to estimate the answer set size for a certain Top-k query. It takes as input the query q together with the auxiliary information and output a parameter k.
- $TK_i \leftarrow \mathsf{BQS.GetToken}(sk, \delta, \eta)$: is an algorithm run on the client side. It takes as input a key sk, the threshold η and auxiliary data structure δ, and output a trapdoor TK_i.
- $c_i \leftarrow \mathsf{BQS.Query}(TK_i, \Delta, \delta)$: is a algorithm run by the server. It takes as input the trapdoor TK_i for $i \in [1, n]$, the encrypted database and auxiliary data structure δ and output encrypted index string if it satisfy the query.

III. Building Blocks

A. Support of Boolean Query

Suppose that the boolean query involves m substring patterns $sp_1, \cdots, sp_m$. For each index string s_i, the server has computed the encrypted scores $\widehat{x}_{ij}$ between s_i and sp_j for all $j = 1, 2, \ldots, m$. The threshold is denoted as η. For example, when $m = 2$, our main idea is that for each index string s_i, let the server compute the similarity scores, x_{i1} and x_{i2}, for sp_1 and sp_2 respectively. The problem is to decide if "$x_{i1} \leq \eta$ AND $x_{i2} \leq \eta$". Note that different thresholds can be set for different sp. The problem of boolean query can be considered as a multi-dimensional range query.

There are limited studies of range query over encrypted data. [26] address the multi-dimensional range query problem. [27] constructed a scheme named hidden vector encryption that handles range queries as well as conjunctions. Both of them were cast in the concept of predicate encryption (PE) which stem from attribute-based encryption (ABE). In details, they defined a certain range as an attribute and encrypt the message using corresponding attributes. Whenever the attribute satisfies the query, the message can be decrypted correctly. We adopt the technique in [27] to implement conjunctive range query. However in our problem the distances are computed in encrypted form on the remote server, DO cannot attribute the encrypted scores into ranges. Thus their method cannot directly be applied in our case. The difference is that in our scheme, we have to give the work load of attributing and encryption to server side while protecting the sensitive information. Since the scheme is a public key encryption and each attribute is in binary representation, in setup phase, the server can prepare multiple cipthertexts for each bit (0 or 1) of the attributes which is a one time initialization. To avoid leaking results of individual condition of the conjunction, client use random numbers to hide η in the token and generates newly-designed tokens to obliviously select correct attributes. Our boolean query system (BQS) consists of three parts: BQS.Setup, BQS.GetToken and BQS.Query.

$(\delta) \leftarrow \mathsf{BQS.Setup}(\lambda)$

@Client:

The setup algorithm first chooses random primes p, q and creates bilinear groups $\mathbb{G}$ and $\mathbb{G}_T$ of composite order $N = pq$.

e is a function $e : \mathbb{G}^2 \to \mathbb{G}_T$ satisfying the bilinear[1]and non-degenerate[2]properties. Next it picks random elements.

$$(u_1, h_1, w_1), \cdots, (u_l, h_l, w_l) \in \mathbb{G}_p^3, g, v \in \mathbb{G}_p, g_q \in \mathbb{G}_q$$

and an exponent $\alpha \in \mathbb{Z}_p$. l is a number related to the maximum number of pattern strings in a boolean query. In practice, user seldom includes too many patterns at a time (normally, $m \leq 5$), we can reserve a relative larger l. It keeps all these as the secret key SK and include it into sk.

It then chooses $3l + 1$ random blinding factors in $\mathbb{G}_q$:

$$(R_{u,1}, R_{h,1}, R_{w,1}), \cdots, (R_{u,l}, R_{h,l}, R_{w,l}) \in \mathbb{G}_q, R_v \in \mathbb{G}_q$$

For the public key, PK, it publishes the description of the group $\mathbb{G}$ and the values

$$g_q, V = vR_v, A = e(g, v)^\alpha,$$
$$\begin{pmatrix} U_1 = u_1 R_{u,1}, & H_1 = h_1 R_{h,1}, & W_1 = w_1 R_{w,1} \\ & \vdots & \\ U_l = u_l R_{u,l}, & H_l = h_l R_{h,l}, & W_l = w_l R_{w,l} \end{pmatrix}$$

The identity space $\mathcal{ID} = \{id_{s_1}, id_{s_2}, \cdots, id_{s_n}\}$ is set to be a subset of $\mathbb{G}_T$ of size less than $N^{1/4}$.

@Server:

For each s_i, id_i is the identity of s_i, choose a random $s \in \mathbb{Z}_n$ and random $Z, (Z_{1,1}, Z_{1,2}), \cdots, (Z_{l,1}, Z_{l,2}) \in \mathbb{G}_q$, output the ciphertext: $C = (C' = id_i \cdot A^s, C_0 = V^s Z, C'')$,

$$C'' = \begin{pmatrix} C_{1,1}^0, & C_{1,1}^1, & C_{1,2} = W_1^s Z_{1,2} \\ \vdots & \vdots & \vdots \\ C_{l,1}^0, & C_{l,1}^1, & C_{l,2} = W_l^s Z_{l,2} \end{pmatrix}$$

where for $i \in [1, l], b \in \{0, 1\}$, $C_{i,1}^b = (U_i^b H_i)^s Z_{i,1}$.

$TK_i \leftarrow \mathsf{BQS.GetToken}(SK, \eta, \delta)$ **@Client**:

For $j \in [1, m]$, the client generates $t - 1$ random integers $y_1, \cdots, y_{t-1}$. Note that as long as the system is setup, the similarity distance x_{ij} usually have a range: $[X_{min}, X_{max}]$ and $\eta \in [X_{min}, X_{max}]$. For each $j \in [1, m]$, concatenate $c_j = (y_1, \cdots, \eta, \cdots, y_{t-1})$ together and finally obtains

$$\begin{aligned} T_i &= (c_1 || \cdots || c_m) \\ &= (y_1^1, \cdots, \eta, \cdots, y_{t-1}^1 || \cdots || y_1^m, \cdots, \eta, \cdots, y_{t-1}^m) \end{aligned}$$

For each value $T_i[k] \in T_i$ if $X_{min} < T_i[k] < \eta$, set $\sigma_i[k] = *$, if $T_i[k] < X_{min}$, set $\sigma_i[k] = 0$, otherwise, $T_i[k] \geq \eta$, set $\sigma_i[k] = 1$. Encrypt values in T_i, get $\widehat{T}_i = \mathsf{SDB.Enc}(T_i)$.

Let S be the set of all indexes j such that $\sigma_i[j] \neq *$. To generate a token for the queries, client chooses random $(r_{j,1}, r_{j,2}) \in \mathbb{Z}_p^2$ for all $j \in S$ and output:

$$TK_i = (S, \widehat{T}_i, K_0 = g^\alpha \prod_{j \in S} (u_j^{\sigma_i[j]} h_j)^{r_{j,1}} w_j^{r_{j,2}},$$
$$\forall j \in S, \sigma_i[j] : K_{j,1} = v^{r_{j,1}}, K_{j,2} = v^{r_{j,2}})$$

$c_i \leftarrow \mathsf{BQS.Query}(TK_i, \Delta, \delta)$ **@Server**.

[1](Bilinear) $\forall u, v \in \mathbb{G}, \forall a, b \in \mathbb{Z}$, $e(u^a, v^b) = e(u, v)^{ab}$

[2](Non-degenrate) $\exists g \in \mathbb{G}$ such that $e(g, g)$ has order n in $\mathbb{G}_T$

After receiving the query token for id_i, the server do: First, for each index string s_i in candidate set and sp_j in query set, initialize a I_i of size mt. Using secure operations in SDB to obtain the encrypted score $\widehat{x}_{ij}$. For s_i, sp_j, use SDB.Comparison operation to reveal the relationship between x_{ij} and $T_i[k], k \in [(j-1)t + 1, jt]$. If $x_{ij} \leq T_i[k]$, set $I_i[k] = 1$, otherwise, set $I_i[k] = 0$

Finally, compute

$$id_i \leftarrow C' / \left(e(C_0, K_0) / \prod_{j \in S} e(C_{j,1}^{I_i[j]}, K_{j,1}) e(C_{j,2}, K_{j,2}) \right)$$

If $id_i \notin \mathcal{ID}$, output $\perp$. Otherwise, output c_i.

Lemma III.1. *With the notation as above, and assuming $|\mathcal{ID}| < N^{1/4}$, whenever $(x_{i1} \leq \eta) \wedge \cdots \wedge (x_{im} \leq \eta)$, id_i is correctly output.*

The proof of Lemma III.1 is given in Figure 3.

B. Answer Set Size Estimation:

As discussed before, for single query, top-k based search are in common use in many scenario. It is our newly designed sorting protocol for SDB that realizes this goal. What has never been considered is how to choose a better parameter k for the top-k search for different queries. The most accurate and intuitive method to get the ideal answer set size is to store a number N_{sp} for every substring during the system setup period, where N_{sp} denotes the number of strings containing sp in the dataset. Knowing N_{sp} in advance, user can choose a better parameter top-k for different queries. However, from Figure 1, the number of different n-grams (when $n \geq 4$) requires a substantial amount of memory and even larger than the number of keywords from which they are derived from when $n \geq 5$. In order to decrease the storage and management costs, our main idea is to maintain the statistic information for all the 3-grams of the dataset and use it to predicate the occurrences of any given query with length > 3 using the intersection size of the 3-grams it contains. For example, to estimate the number of index strings contain $sp =$'tion', we split it into two 3-gram: {tio, ion}. We approximate $|A_{tion}|$ by utilizing the intersection size of $|A_{tio} \cap A_{ion}|$.

Definition III.1. *(Approximate k) Let A_{str} denote the set of identifiers of index strings which contain str. Given a substring query str, $k = \lceil x \times (|\bigcap_{g \in L_{str}^3} A_g| + 1) \rceil$, where L_{str}^3 is the 3-gram list of str and x is a scale parameter.*

Remark: x is a scale parameter to allow looser restriction for higher $recall$ or tighter one for higher $accuracy$. In our experiments, we test $accuracy$ and $recall$ for different $x \in [0.1, 1]$, and users can determine whether he wants higher $accuracy$ or $recall$ according to his own requirement.

To estimate the intersection size, we adopt MinHash technique proposed in [21]. The resemblance between two sets X and Y is defined as $\frac{|X \cap Y|}{|X \cup Y|}$. Min-wise independent permutation is a well-known Monte Carlo technique that estimate set resemblance. The estimator estimates the answer by repeatedly

$I_j[i] = 1$ or 0,and if $\sigma_j[i] = I_j[i]$ for all $i \in S$, we compute:

$$\begin{aligned} & C' / \left(\frac{e(C_0, K_0)}{\prod_{i \in S} e(C_{i,1}, K_i, 1) e(C_{i,2}, K_{i,2})} \right) \\ = \; & C' / \left(\frac{e(v^s, g^\alpha \prod_{i \in S} (u_i h_i)^{r_{i,1}} w_i^{r_{i,2}}) e(R_v Z, g^\alpha \prod_{i \in S} (u_i h_i)^{r_{i,1}} w_i^{r_{i,2}})}{\prod_{i \in S} e((u_i R_{u,i} h_i R_{h,i})^s, v^{r_{i,1}}) e(Z_{i,1}, v^{r_{i,1}}) e((w_i R_{w,i})^s, v^{r_{i,2}}) e(Z_{i,2}, v^{r_{i,2}})} \right) \end{aligned}$$

because $e(R_v Z, g^\alpha \prod_{i \in S} (u_i h_i)^{r_{i,1}} w_i^{r_{i,2}}) = 1$

$$\begin{aligned} & C' / \left(\frac{e(v^s, g^\alpha \prod_{i \in S} (u_i h_i)^{r_{i,1}} w_i^{r_{i,2}}) e(R_v Z, g^\alpha \prod_{i \in S} (u_i h_i)^{r_{i,1}} w_i^{r_{i,2}})}{\prod_{i \in S} e((u_i R_{u,i} h_i R_{h,i})^s, v^{r_{i,1}}) e(Z_{i,1}, v^{r_{i,1}}) e((w_i R_{w,i})^s, v^{r_{i,2}}) e(Z_{i,2}, v^{r_{i,2}})} \right) \\ = \; & C' / \left(\frac{e(v^s, g^\alpha \prod_{i \in S} (u_i h_i)^{r_{i,1}} w_i^{r_{i,2}})}{\prod_{i \in S} e((u_i h_i)^s, v^{r_{i,1}}) e(w_i^s, v^{r_{i,2}})} \right) \\ = \; & C' / \left(\frac{e(v^s, g^\alpha \prod_{i \in S} (u_i h_i)^{r_{i,1}})}{\prod_{i \in S} e((u_i h_i)^s, v^{r_{i,1}}) e(w_i, v)^s} \right) \\ = \; & C' / \left(e(v^s, g^\alpha) \right) \\ = \; & id \cdot e(v, g)^{s\alpha} / e(v^s, g^\alpha) \\ = \; & id \end{aligned}$$

Fig. 3. Proof of Lemma III.1

assigning random rank to the universe and keeping the minimal rank of a set. Each assignment of a random rank to the universe is a Pseudo-random permutation. The minimum values from the permuted ranks of a set obtained from each permutation are called the signature vector and can also be used to estimate the set resemblance.

Lemma III.2. *Let A_j be the largest among $\{A_1, \cdots, A_n\}$. Given the resemblance γ between A_j and $A_1 \cup \cdots \cup A_n$ together with the resemblance ρ between $A_1 \cap \cdots \cap A_n$ and $A_1 \cup \cdots \cup A_n$, we estimate $|\bigcap_{i=1}^n A_i| = \frac{\rho |A_j|}{\gamma}$.*

Proof.

$$\gamma = \frac{|A_j|}{|A_1 \cup \cdots \cup A_n|}, \rho = \frac{|A_1 \cap \cdots \cap A_n|}{|A_1 \cup \cdots \cup A_n|}$$
$$|\bigcap_{i=1}^n A_i| = \rho \times |A_1 \cup \cdots \cup A_n| = \rho \times \frac{\gamma}{|A_j|}$$

□

The algorithm, Est.preprocess($\cdot$), generates a signature vector for each 3-gram set. EstK($\cdot$) uses the signature vectors of $A_1, \cdots, A_n$ to estimate set resemblance and finally the size of the intersection $|\bigcap_{i=1}^n A_i|$.

Est.preprocess(D):
D is the dataset and the procedure generates a set of permutations on id of dataset D: $\Pi = \{\pi_1, \cdots, \pi_{|\Pi|}\}$. Let L denote the set containing all the different 3-grams in D.
For each 3-gram $g \in L$:

- $A_g \subset D$ is the set of strings that contain g as a substring;
- Let $min(\pi_i(A_g)) = min\{\pi_i(x) | x \in A_g\}$;
- $sig_{A_g} = [min(\pi_1(A_g)), \cdots, min(\pi_{|\Pi|}(A_g))]$;
- Output sig_{A_g} and include it into δ .

$N_{str} \leftarrow$ EstK($str, sig_{A_1}[i], \cdots, sig_{A_n}[i]$): For $i \in [1, |\Pi|]$: $sig_{A_1 \cup \cdots \cup A_n}[i] = min\{sig_{A_1}[i], \cdots, sig_{A_n}[i]\}$;
Let $A_j \in \{A_1, \cdots, A_n\}$ be the one whose size is the biggest. Obtain γ' as the estimation of the resemblance γ: $\gamma' = \frac{|i | sig_{A_j}[i] = sig_{A_1 \cup \cdots \cup A_n}[i], 1 \le i \le |\Pi|]|}{|\Pi|}$ and ρ' as the estimation of the resemblance ρ: $\rho' = \frac{|i | sig_{A_1}[i] = \cdots = sig_{A_n}[i], 1 \le i \le |\Pi|]|}{|\Pi|}$.
Then compute the estimation $|A_1 \cap \cdots \cap A_n| = \rho |A_1 \cup \cdots \cup A_n| = \frac{\rho' |A_j|}{\gamma'}$.
Output $k = \lceil x \times (|\bigcap_{i=1}^n A_i| + 1) \rceil$.

IV. Our Construction

A. System setup phase

$sk \leftarrow$ KeyGen(1^λ): Given a security parameter λ, the algorithm outputs a set of secret keys $sk = (sk_f, sk_s)$. sk_f is the key for a secret PRF f used in FVE($\cdot$). sk_s is a symmetric key to encrypted the original strings in dataset.
$V \leftarrow$ FVE(str, sk): For each $s_i \in D$, the algorithm first calls FVE(sk, s_i) to generate a fingerprint vector V_i.
$\Delta \leftarrow$ Enc($sk, D, V_1, \cdots, V_n$) :

1) For each string s_i in dataset, set each entry to $V_i[j] = \alpha + V_i[j]$, where α is a random element.
2) Each entrie $V_i[j]$ of V_i will be further encrypted by running SDB.Gen and SDB.Enc. V_i is encrypted into $\widehat{V}_i$
3) For $i \in [1, n]$, let $c_i =$ SKE(s_i, sk_s), where SKE($\cdot$) is a symmetric encryption.
4) Output $\Delta = (\mathbf{c} = \{c_1, \cdots, c_n\}, \mathbf{V}_e = \{\widehat{V}_1, \cdots, \widehat{V}_n\})$.

$s \leftarrow$ Dec(sk, c): Return $s_i =$ SKE(c_i, sk_s).
$\delta \leftarrow$ Preprocess(sk, D):

- Est.preprocess(D): Generate auxiliary data structure for EstK($\cdot$).
- BQS.Setup(λ): Setup the boolean query system.

B. Query phase

$\mathbf{c_q} \leftarrow \mathsf{Search}(sk, \mathbf{q} = \{sp_1, \cdots, sp_m\}, \Delta)$:

@Client: For each sp in $\mathbf{q}$:

1) Call $\mathsf{FVE}(sk, sp_i)$ to generate the fingerprint vector V_{sp} for sp.

2) Note that all the n-grams of sp are mapped into the nonzero elements of V_{sp}. We use weight vectors to indicate the non-zero positions in the fingerprint vectors. Let $N_e = \{n_1, n_2, \cdots, n_c\}$ denote the positions of nonzero elements in V_{sp}. Let H_1 and H_2 be two vectors with size equals to $||V_{sp}||$. For $i \in [1, d]$, d is the size of V_{cnt}, the matrixes H_1 and H_2 are designed as follows:

$$H_1[i] = \begin{cases} \omega & if\ i \in N_e \\ 0 & otherwise \end{cases}$$
$$H_2[i] = \beta \cdot H_1[i]$$

where ω is a weight parameter that is used to select a certain nonzero element in the fingerprint vector. According to [12], β is the average value of all the non-zero elements in all fingerprint vectors. For $j \in N_e$, set $V_{sp}[j] = \frac{\alpha}{\beta} + V_{sp}[j]$. For the three types of query:

- If it is a substring query $\%sp\%$ which is not position-sensitive, for $i \in N_e$, DO computes $token_{sp}[i] = V_{cnt}[i]$
- If it is a prefix matching $sp\%$, which is position-sensitive, for $i \in N_e$, DO computes $token_{sp}[i] = V_{cnt}[i] + V_{\mathsf{prefixpos}}[i]$
- If it is a suffix matching $\%sp$, which is position-sensitive, for $i \in N_e$, DO computes $token_{sp}[i] = V_{cnt}[i] + V_{\mathsf{suffixpos}}[i]$

3) In order to reduce the communication and computation costs, DO extracts the non-zero elements as $token'_{sp}[i]$. For each index string s_i, DO encrypts $\widehat{token}_{sp}$= $\mathsf{SDB.Enc}(token'_{sp})$.

4) DO sends the trapdoor $TD_i = \langle \widehat{token}_{sp}, H_1, H_2 \rangle$ to the SP.

5) If $m > 1$ which means it is a boolean query, client calls $\mathsf{BQS.GetToken}(sk, \delta, \eta)$ to obtain TK_i and sends it to SP. If $m = 1$, user can either choose a threshold, output a threshold η or top-k results for output. Moreover, client can run $\mathsf{EstK(sp)}$ to select a k according to the user's requirement of $Accuracy$ and $Recall$.

@Server: Compute the similarity distance between index strings and the query by $\mathsf{SDB.Operator}(sk, \Delta, TD_i)$. From the weighted vector H_1 and H_2, SP can obtain the positions N_e that count in the process of score computation. Taking the prefix query for example:

1)For each sp in $\mathbf{q}$ and each index string s_i, compute the encrypted distance using the multiplication, addition, subtraction operators.

2) After computing all the similarity distances: If $m > 1$, for each s_i, SP calls $\mathsf{BQS.Query}(TK_i, \Delta, \delta)$ to determine if s_i satisfies the query. If $m = 1$, SP performs SDB.Comparison protocol to verify whether the similarity score is within the threshold η. For top-k results, SP runs our designed sorting and returns the k best matching strings as the results.

V. Security Analysis

In this paper, we consider an honest-but-curious cloud server in our threat model. We follow the widely-accepted framework [7] to analyze the security and formalize the security model as follows:

Definition V.1. *(History H): Given a string collection D, a k-query history over D is a tuple $H = (D, Q)$ that includes the document collection D and a vector of k string patterns $Q = \{q_1, \cdots, q_k\}$.*

In substring matching problem, for two queries q_1 and q_2, if q_1 is a substring of q_2, the result set of q_2 is a subset of result set of q_1, which is an inevitable leakage. Thus, we give the following search pattern definition for secure pattern matching problem.

Definition V.2. *(Search Pattern π): Given a search object q at time t, N_e is the position information list of q (which means an n-gram of q was hashed in this bucket.), For $g \in N_e$ the search pattern $\pi(D, g)$ is defined by a binary vector of length t with a '1' at location i if the search at time $i \leq t$ contains an n-gram which is hashed into g; '0' otherwise.*

Definition V.3. *(Access Pattern AP): Given a search object q at time t, the access pattern is defined by $AP(D, q) = \{id(\boldsymbol{c}_q)\}$.*

[CQA-security] Let $\mathcal{L}_1$, $\mathcal{L}_2$ be leakage functions for setup and query respectively. A secure pattern matching (SPM) scheme is said to be secure against non-adaptively chosen query attack (CQA), if for any PPT adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that for the following probabilistic experiments $\mathbf{Real}_{\mathcal{A}}^{SPM}(\lambda)$ and $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}^{SPM}(\lambda)$, we have:

$$|\mathsf{Pr}[\mathcal{D}(O, st_{\mathcal{A}}) = 1 : (O, st_{\mathcal{A}}) \leftarrow \mathbf{Real}_{\mathcal{A}}^{SPM}(\lambda)] -$$
$$\mathsf{Pr}[\mathcal{D}(O, st_{\mathcal{A}}) = 1 : (O, st_{\mathcal{A}}) \leftarrow \mathbf{Ideal}_{\mathcal{A},\mathcal{S}}^{SPM}(\lambda)| \leq \mathsf{negl}(\lambda)$$

$\mathbf{Real}_{\mathcal{A}}^{SPM}(\lambda)$: The challenger runs $\mathsf{KeyGen}(1^\lambda)$ to generate the key sk. $\mathcal{A}$ outputs D and receives Δ from the challenger. $\mathcal{A}$ makes a polynomial number of adaptive queries q. For query q, the challenger acts as a client and runs Search with $\mathcal{A}$ acting as an honest server. Finally, $\mathcal{A}$ returns a bit b output by the experiment.

$\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}^{SPM}(\lambda)$: $\mathcal{A}$ outputs D and sends it to $\mathcal{S}$. Given $\mathcal{L}_1$, $\mathcal{S}$ generates and sends Δ to $\mathcal{A}$. $\mathcal{A}$ makes a polynomial number of adaptive queries q. For each query q, $\mathcal{S}$ is given $\mathcal{L}_2$, and simulates a client who runs Search with $\mathcal{A}$ acting as an honest server. Finally, $\mathcal{A}$ returns a bit b output by the experiment. Here, $\mathcal{L}_1$, $\mathcal{L}_2$ are stateful leakage functions.

For our construction, we define the leakage function as follows:

Definition V.4. *(Leakage function $\mathcal{L}_1$ for setup): Given a string collection $D = \{s_1, \cdots, s_n\}$, $\mathcal{L}_1 = \{|s_i|\}_{i=1}^n$, where $|\cdot|$ denotes the length of the string.*

Definition V.5. *(Leakage function $\mathcal{L}_2$ for query): Given a string collection D, a search object q,*

$\mathcal{L}_2(D,Q) = \{\pi(D,Q), AP(D,Q), \boldsymbol{TD}\}$, ***T*** *is the set of query trapdoors generated from* Q.

The proofs for different kinds of schemes: substring matching, prefix matching and suffix matching are similar. Thus, we will only prove for the substring matching scheme.

Theorem V.1. *If* h_{sk} *is a pseudo-random function, and if* SKE *and* SDB.Enc *are CPA-secure, HVE is selectively secure, then SPM is CQA secure.*

Proof. We argue its security by describing a polynomial-time simulator $\mathcal{S}$, such that for any PPT adversary $\mathcal{A}$, the outputs of $\mathbf{Real}_{\mathcal{A}}^{SPM}(\lambda)$ and $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}^{SPM}(\lambda)$ are indistinguishable. Consider the simulator $\mathcal{S}$ that given the leakage of a history H, generates a string $O' = (\{\widehat{V}_1', \cdots, \widehat{V}_n'\}, \mathbf{c}', \mathbf{TD}')$ as follows:

[Simulating Setup] $\mathcal{S}$ is given $\mathcal{L}_1 = \{|s_i|\}_{i=1}^n$ and $\mathcal{L}_2 = \{\pi(D,Q), AP(D,Q), \mathbf{TD}\}$.

- (Simulating $\{\widehat{V}_1', \cdots, \widehat{V}_n'\}$) For $i \in [1,n]$, let V_i' be a vectors of length $|V_i|$. According to the token history, for each q_j and $j \in [1,k]$, $T_i = \langle \widehat{token}_{q_j}, H_1, H_2 \rangle$ obtained from leakage $\mathcal{L}_2$, for each non-zero position z in H_1, if $s_i \in AP(D,q_j)$, $\mathcal{S}$ set $V_i'[z] = \alpha' + r$, where r is randomly chosen from $[1, CT]$, otherwise set α'; for zero positions in H_1, $\mathcal{S}$ randomly set them to $\alpha' + e$, where e is randomly chosen from $[0, CT]$;. Following SDB.Enc, $\mathcal{S}$ chooses random row-id and column key to obtain $\widehat{V}_i' \leftarrow V_i'$ and stores the keys.
- (Simulating $\mathbf{c}'$) Generate $sk_s' \leftarrow \mathsf{SKE.Gen}(1^\lambda)$ and $c_i' \leftarrow \mathsf{SKE.Enc}(\{0,1\}^{|s_i|}, sk_s')$.
- Output $(\{\widehat{V}_1', \cdots, \widehat{V}_n'\}, \mathbf{c}')$.

[Simulating Queries] $\mathcal{S}$ is given $\mathcal{L}_2 = \{\pi(D,Q), AP(D,Q), \mathbf{TD}\}$.

- (Simulating $\mathbf{TD}'$) For query $q \in Q$, let V_q' be a vectors of length $|V_q|$. According to the token history, for $i \in [1,n]$, $TD_i = \langle \widehat{token}_q, H_1, H_2 \rangle$ obtained from leakage $\mathcal{L}_2$, for each non-zero position z in H_1, $\mathcal{S}$, set $V_q'[z] = \alpha' + r$, where r is randomly chosen from $[1, CT]$. $\mathcal{S}$ uses random column key and the row-id chosen in previous to obtain $\widehat{token}_q'$. Finally, set $H_1' = H_1, H_2' = H_2$.
- (Simulating $\mathbf{TK}$') $\mathcal{S}$ randomly generates SK and PK, to simulate TK_{q_i}' for every $q_i \in Q$, for $j \in [1,m]$, the client generates $t-1$ random integers $y_1, \cdots, y_{t-1}$, together with the threshold $\eta \in [X_{min}, X_{max}]$. For each $j \in [1,m]$, TK_{ij}'.
- Output $\mathbf{TD}'$ and $\mathbf{TK}'$.

We now claim that no polynomial-size distinguisher $\mathcal{D}$ can distinguish between the distribution O' and O.

- ($\{\widehat{V}_1, \cdots, \widehat{V}_n\}$ and $\{\widehat{V}_1', \cdots, \widehat{V}_n'\}$) The CPA-security of SDB.Enc guarantees that $\widehat{V}_i'$ cannot be distinguished from $\widehat{V}_i$.
- ($\mathbf{c}$ and $\mathbf{c}'$) Recall that $\mathbf{c}'$ is SKE encryption, without encryption key, the PCPA-security of SKE will guarantee that $\mathbf{c}'$ and $\mathbf{c}$ are indistinguishable.
- ($\mathbf{TD}$ and $\mathbf{TD}'$) For different index string s_i and $q_j \in Q$ each token TD_{ij}' in $\mathbf{TD}'$ consists of three parts: $\widehat{token}_{q_j}'$, H_1' and H_2'. H_1', H_2' are the same with H_1, H_2, thus cannot be distinguished, and the CPA-security of SDB.Enc guarantees the indistinguishability of $\widehat{token}_{q_j}'$ and $\widehat{token}_{q_j}$.
- ($\mathbf{TK}$ and $\mathbf{TK}'$) It has been proved in [27] that HVE is selectively secure under the composite 3-party Diffie-Hellman assumption and the bilinear Diffie-Hellman assumption. In GetToken phase, the randomly generated numbers and the CPA-secure SDB encryption guarantee the indistinguishability of the token. Thus, the probability that the distinguisher $\mathcal{D}$ can distinguish between $\mathbf{TK}$ and $\mathbf{TK}'$ is $\mathrm{negl}(n)$.

In conclusion, the output $O' = (\{\widehat{V}_1', \cdots, \widehat{V}_n'\}, \mathbf{c}', \mathbf{TD}')$ of $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}^{SPM}(\lambda)$ and the output $O = (\{\widehat{V}_1, \cdots, \widehat{V}_n\}, \mathbf{c}, \mathbf{TD})$ of $\mathbf{Real}_{\mathcal{A}}^{SPM}(\lambda)$ are indistinguishable, thus complete the proof. □

VI. Performance Evaluation

We conduct experiments to evaluate the effectiveness and efficiency of our constructions. The experiments are performed on a machine with an Intel core i5-3570 3.40GHz processor 16GB of DRAM. All the algorithms are programmed using python. We report search accuracy, recall, encryption time and search time. We randomly extract 100000 plaintext keywords from an English words dictionary[3]. The dataset contains 100000 strings, the average length is 9.4577, the minimum length is 3 and the maximum length is 25.

A. Evaluation Metrics

For different query string lengths, we randomly pick substrings and extract the corresponding fingerprint vectors as the search objects. Following [12], we test substrings with length varying from 3 to 5. All performance measures are averaged over these queries. For each search substring, the *ideal answer set* is defined to be all the index strings that contain the query substring and the *actural answer set* is defined to be all the index strings returned by our scheme. Given a search query Q, we use $I(Q)$ to denote the set of ideal answers in which the index strings actually contain the query substring. Assume $A(Q)$ is the set of actual answers returned by our pattern matching scheme. $Accuracy$ and $Recall$ are two common metrics in evaluation of data retrieval system. They are defined as:

$$Accuracy = \frac{|A(Q) \cap I(Q)|}{|A(Q)|}, \quad Recall = \frac{|A(Q) \cap I(Q)|}{|I(Q)|}$$

In the ideal cases, the values of $Accuracy$ and $Recall$ are 1.0.

B. Evaluation Results

Accuracy and efficiency: Table I shows the search accuracy and time costs under different fingerprint vector sizes. We randomly pick 500 substrings with length 4. The table shows that we can rely on a larger vector size to increase the accuracy, for the reason that the number of false positives introduced by

[3]https://github.com/dwyl/english-words

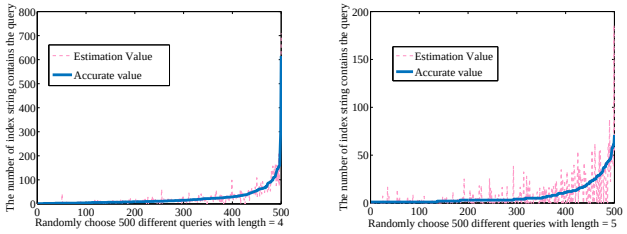
Fingerprint Vector size	*Accuracy*	FVE time(s)	Encryption time(s)	Total setup time (s)
26-64-128	0.8908	15.2842	112.5163	127.8005
26-64-256	0.9298	21.7716	177.5533	199.3249
26-64-512	0.9540	34.0799	308.9223	343.0022
26-128-128	0.92000	18.5684	144.7111	163.2795
26-128-256	0.94288	24.6240	210.3956	235.0196
26-128-512	0.95999	37.3588	341.7645	379.1233

TABLE I
THE SEARCH ACCURACY AND TIME COSTS UNDER DIFFERENT VECTOR SIZE

Single query			
pattern type	length	*Accuracy*	*Recall*
substring matching	3	0.8410	1.0
	4	0.9261	1.0
	5	0.9180	1.0
prefix matching	3	1.0	1.0
	4	0.9875	1.0
	5	0.9896	1.0

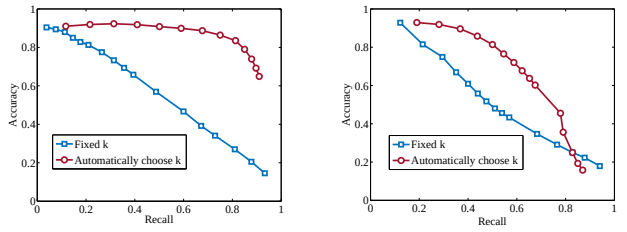
Boolean query		
#patterns	*Accuracy*	*Recall*
1	0.9764	0.9979
2	0.9764	0.9920
3	0.9902	1.0
4	1.0	1.0

TABLE II
Accuracy AND *Recall* FOR DIFFERENT TYPES OF QUERY



(a) Substring length = 4. (b) Substring length = 5.

Fig. 4. The relationship between the accurate value and the estimation value.



(a) Substring length = 4. (b) Substring length = 5.

Fig. 5. The relationship between search *Accuracy* and *Recall*. for top-k substring search.

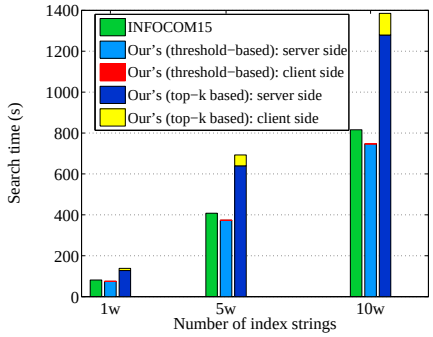


Fig. 6. Search efficiency: search time cost under different numbers of index strings.

hash collision becomes smaller. Moreover, a longer vector will increase the fingerprint vector extraction time and encryption time, but the setup of the system is a one-time step. Thus, for a specific dataset, by choosing a vector size carefully, we can obtain a balance between accuracy and efficiency.

Table II shows the main results of different types of queries including: substring and prefix threshold based single query with pattern length ranging from 3 to 5. We conclude that the threshold-based scheme can retrieve the correct results with high *Accuracy* and *Recall* at the same time so that we can rely on it to achieve effective boolean query. We also illustrate the search performance of boolean query in Table II.

Figure 6 gives the search costs under different sizes of datasets. For the time cost of our scheme. The time costs are mainly caused by the repeatedly comparison operator. From the figure, we can see that the workload on the client side is very small and most of the computation are outsourced to the remote server.

k value estimation: It is reported in [21] that in practice, the number of permutations $|\Pi|$ does not need to be big for good estimation. Chen *et al.*used 50 permutations when the size of universe $|U|$ is about 2^{17}. In our work, we adopt 50 pseudorandom permutations. We randomly pick 500 substrings with length-4 and length-5 respectively and use the statistical information of the 3-gram of the queried substrings to predicate the number of answers. Figure 4 shows the estimation performance. It can be seen from the figure that the estimation values are close to the true values. The accuracy of the estimation decreases when the length of the substring becomes longer because longer substrings require more 3-grams to do set intersection, which increases the accumulated errors. How to improve the prediction accuracy for longer strings would be one of our future work.

Based on the estimation shown in Figure 4, we show the results of $Accuracy$ and $Recall$ in Figure 5. Theoretically, higher $Recall$ will inevitably bring more false positives which will decrease the $Accuracy$. For the fix-k experiments, we set different k to obtain the relationship between $Accuracy$ and $Recall$. The experiment has shown that with the increase in $Recall$, the $Accuracy$ decreases sharply for fix-k solutions while the results of our approach show a much better accuracy than the approach in [12] whose k is fixed for all queries. The average times for estimating length-4 and length-5 queries are $0.2538s(s)$ and $0.4138(s)$ respectively. The overhead costed by the estimation is small and can be neglected.

VII. Conclusions

In this work, we propose a secure pattern matching scheme for searching strings in the secure database system SDB. To the best of our knowledge, we are the first to investigate the problem of secure pattern matching that supports boolean query. For single query, we design a new operator - sorting for SDB to support top-k similarity search and propose a selective estimation technique to enable the system to compute k automatically for each query. Thorough security analysis and experiment results over real data, we show that our methods is not only secure but also provide high search quality in terms of $recall$ and $accuracy$.

Acknowledgement

This project is in part supported by RGC funding, Hong Kong (Project No.: CityU C1008-16G).

References

[1] I. Hang, F. Kerschbaum, and E. Damiani, "Enki: access control for encrypted query processing," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 183–196.

[2] M. I. Sarfraz, M. Nabeel, J. Cao, and E. Bertino, "Dbmask: fine-grained access control on encrypted relational databases," in *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*. ACM, 2015, pp. 1–11.

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

[4] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich, "Processing analytical queries over encrypted data," in *Proceedings of the VLDB Endowment*, vol. 6, no. 5. VLDB Endowment, 2013, pp. 289–300.

[5] W. K. Wong, B. Kao, D. W. L. Cheung, R. Li, and S. M. Yiu, "Secure query processing with data interoperability in a cloud database environment," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014, pp. 1395–1406.

[6] E.-J. Goh *et al.*, "Secure indexes." *IACR Cryptology ePrint Archive*, vol. 2003, p. 216, 2003.

[7] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," *Journal of Computer Security*, vol. 19, no. 5, pp. 895–934, 2011.

[8] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 965–976.

[9] F. Hahn and F. Kerschbaum, "Searchable encryption with secure and efficient updates," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 310–320.

[10] S. Kamara and C. Papamanthou, "Parallel and dynamic searchable symmetric encryption," in *International Conference on Financial Cryptography and Data Security*. Springer, 2013, pp. 258–274.

[11] M. Chase and E. Shen, "Substring-searchable symmetric encryption," *Proceedings on Privacy Enhancing Technologies*, vol. 2015, no. 2, pp. 263–281, 2015.

[12] D. Wang, X. Jia, C. Wang, K. Yang, S. Fu, and M. Xu, "Generalized pattern matching string search on encrypted data in cloud systems," in *Computer Communications (INFOCOM), 2015 IEEE Conference on*. IEEE, 2015, pp. 2101–2109.

[13] T. Zhang, X. Wang, and S. S. Chow, "Privacy-preserving multi-pattern matching," in *International Conference on Security and Privacy in Communication Systems*. Springer, 2016, pp. 199–218.

[14] W. K. Wong, D. W.-l. Cheung, B. Kao, and N. Mamoulis, "Secure knn computation on encrypted databases," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. ACM, 2009, pp. 139–152.

[15] Q. Wang, M. He, M. Du, S. S. Chow, R. W. Lai, and Q. Zou, "Searchable encryption over feature-rich data," *IEEE Transactions on Dependable and Secure Computing*, 2016.

[16] Z. Xia, X. Wang, X. Sun, and Q. Wang, "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 2, pp. 340–352, 2016.

[17] B. Wang, S. Yu, W. Lou, and Y. T. Hou, "Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud," in *INFOCOM, 2014 Proceedings IEEE*. IEEE, 2014, pp. 2112–2120.

[18] Y. Kim and K. Shim, "Efficient top-k algorithms for approximate substring matching," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. ACM, 2013, pp. 385–396.

[19] I. F. Ilyas, W. G. Aref, and A. K. Elmagarmid, "Supporting top-k join queries in relational databases," *The VLDB JournalThe International Journal on Very Large Data Bases*, vol. 13, no. 3, pp. 207–221, 2004.

[20] R. Vernica and C. Li, "Efficient top-k algorithms for fuzzy search in string collections," in *Proceedings of the First International Workshop on Keyword Search on Structured Data*. ACM, 2009, pp. 9–14.

[21] Z. Chen, N. Koudas, F. Korn, and S. Muthukrishnan, "Selectively estimation for boolean queries," in *Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 2000, pp. 216–225.

[22] W. B. Cavnar, J. M. Trenkle *et al.*, "N-gram-based text categorization," *Ann Arbor MI*, vol. 48113, no. 2, pp. 161–175, 1994.

[23] W. Wang, J. Qin, C. Xiao, X. Lin, and H. T. Shen, "Vchunkjoin: An efficient algorithm for edit similarity joins," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 8, pp. 1916–1929, 2013.

[24] C. Y. Suen, "N-gram statistics for natural language understanding and text processing," *IEEE transactions on pattern analysis and machine intelligence*, no. 2, pp. 164–172, 1979.

[25] A. Behm, S. Ji, C. Li, and J. Lu, "Space-constrained gram-based indexing for efficient approximate string search," in *2009 IEEE 25th International Conference on Data Engineering*. IEEE, 2009, pp. 604–615.

[26] E. Shi, J. Bethencourt, T. H. Chan, D. Song, and A. Perrig, "Multi-dimensional range query over encrypted data," in *Security and Privacy, 2007. SP'07. IEEE Symposium on*. IEEE, 2007, pp. 350–364.

[27] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Theory of cryptography*. Springer, 2007, pp. 535–554.