Fuzzy Searching Encryption with Complex Wild-Cards Queries on Encrypted Database

He Chen[1], Xiuxia Tian[2(✉)], and Cheqing Jin[1]

[1] School of Data Science and Engineering, East China Normal University, Shanghai, China
watch_ch@163.com, cqjin@dase.ecnu.edu.cn
[2] College of Computer Science and Technology, Shanghai University of Electric Power, Shanghai, China
xxtian@fudan.edu.cn

Abstract. Achieving fuzzy searching encryption (FSE) can greatly enrich the basic function over cipher-texts, especially on encrypted database (like CryptDB). However, most proposed schemes base on centralized inverted indexes which cannot handle complicated queries with wild-cards. In this paper, we present a well-designed FSE schema through Locality-Sensitive-Hashing and Bloom-Filter algorithms to generate two types of auxiliary columns respectively. Furthermore, an adaptive rewriting method is described to satisfy queries with wild-cards, such as percent and underscore. Besides, security enhanced improvements are provided to avoid extra messages leakage. The extensive experiments show effectiveness and feasibility of our work.

Keywords: Fuzzy searching encryption · Wild-cards searching CryptDB

1 Introduction

Cloud database is a prevalent paradigm for data outsourcing. In consideration of data security and commercial privacy, both individuals and enterprises prefer outsourcing them in encrypted form. CryptDB [21] is a typical outsourced encrypted database (OEDB) which supports executing SQL statements on cipher-texts. Its transparency essentially relies on the design of splitting attributions and rewriting queries on proxy middle-ware. Under this proxy-based encrypted framework, several auxiliary columns are extended with different encryptions and query semantics are preserved through modifying or appending SQL statements.

Supported by the National Key Research and Development Program of China (No. 2016YFB1000905), NSFC (Nos. 61772327, 61532021, U1501252, U1401256 and 61402180), Project of Shanghai Science and Technology Committee Grant (No. 15110500700).

Y. Cai et al. (Eds.): APWeb-WAIM 2018, LNCS 10988, pp. 3–18, 2018.
https://doi.org/10.1007/978-3-319-96893-3_1

To enrich basic functions on cipher-texts, searchable symmetric encryption (SSE) is proposed for keyword searching with encrypted inverted indexes [4,13,22,24], and then dynamic SSE (DSSE) achieves alterations on various centralized indexes to enhance applicability [2,11,12,14]. Besides, the studies about exact searching with boolean expressions are extended in this field to increase accuracy [3,10]. Furthermore, the researches of similar searching among documents or words are widely discussed through introducing locality sensitive hashing algorithms [1,7–9,15,18,19,23,25–27]. However, these proposed schemes are not applicable to OEDB scenario because of the centralized index design and cannot handle complex fuzzy searching with wild-cards.

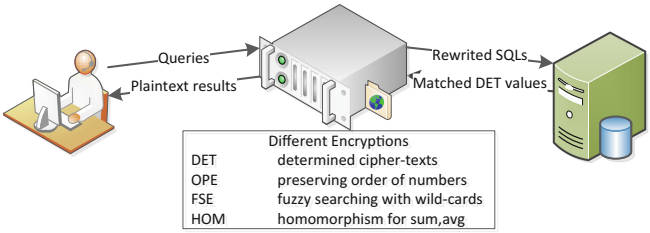

Fig. 1. The client-proxy-database framework synthesizes various encryptions together, such as the determined encryption (DET) preserves symmetric character for en/decryption, the order-preserving encryption (OPE) persists order among numeric values, the fuzzy searching encryption (FSE) handles queries on text, and the homomorphic encryption (HOM) achieves aggregation computing.

Therefore, it is meaningful and necessary to achieve fuzzy searching encryption over outsourced encrypted database. As shown in Fig. 1, the specific framework accomplishes transparency and homomorphism by rewriting SQL statements on auxiliary columns. In this paper, we focus on resolving the functionality of 'like' queries with wild-cards ('%' and '_'). Our contributions are summarized as follows:

- We propose a fuzzy searching encryption with complex wild-cards queries on encrypted database which extends extra functionality for the client-proxy-database framework like CryptDB.
- We present an adaptive rewriting method to handle different query cases on two types of auxiliary columns. The formal column works for similar searching by locality sensitive hashing and the latter multiple columns work for maximum substring matching by designed bloom-filter vectors.
- We evaluate the efficiency, correctness rate and space overhead by adjusting the parameters in auxiliary columns. Besides, security enhanced improvements are provided to avoid extra messages leakage. The extensive experiments also indicate the effectiveness and feasibility of our work.

The rest of paper is organized as follows. Section 2 discusses the related work and Sect. 3 introduces some basic concepts and definitions. Section 4 describes our schema including initialization of auxiliary columns, adaptive rewriting queries and security enhanced improvements. Section 5 presents the experiments and a brief conclusion is given in Sect. 6.

2 Related Work

In recent years, many proposed schemes have been attempting to achieve fuzzy searching encryption with helps of similarity [1,8,9,15,23,25–27]. The most of them introduce locality sensitive hashing (LSH) to map similar items together and bloom-filter to change the method of measuring. Wang et al.'s work [23] was one of the first works to present fuzzy searching. They encode every words in each file into same large bloom-filter space as a vector and evaluate similarity of target queries by computing the inner product for top-k results among vectors. Kuzu et al.'s work [15] generates similar feature vectors by embedding keyword strings into the Euclidean space which approximately preserves the relative edit distance. Fu et al.'s work [8] proposes an efficient multi-keyword fuzzy ranked search schema which is suitable for common spelling mistakes. It benefits from counting uni-gram among keywords and transvection sorting to obtain ranked candidates. Wang et al.'s work [26] generates a high-dimensional feature vector by LSH to support large-scale similarity search over encrypted feature-rich multimedia data. It stores encrypted inverted file identifier vectors as indexes while mapping similar objects into same or neighbor keyword-buckets by LSH based on Euclidean distance. In contrast to sparse vectors from bi-gram mapping, their work eliminates the sparsity and promotes the correctness as well. However, there are many problems in existing schemes including the insufficient metric conversion, the coarse-grained similarity comparison, the extreme dependency of assistant programs and the neglect about wild-card queries.

Meanwhile, the proposal of CryptDB [21] has attracted world-wide attention because they provide a practical way to combine various attribution-preserving encryptions over encrypted database. Then many analogous researches [5,16,17,20] study its security definitions, feasible frameworks, extensible functions and optimizations. Chen et al. [5] consider these encrypted database as a client-proxy-database framework and presents symmetric column for en/decryption and auxiliary columns for supporting executions. This framework helps execute SQL statements directly over cipher-texts through appending auxiliary columns with different encryptions. It also benefits from the transparency of en/decryption processes and combines various functional encryptions together. Therefore, it is meaningful to achieve efficient fuzzy searching with complex wild-cards queries on proxy-based encrypted database.

3 Preliminaries

3.1 Basic Concepts

A. N-gram. In the fields of computational linguistics and probability, the n-gram method is proposed for measurement by generating a contiguous sequence of items from given strings. Essentially, it converts texts to fragments sets for vectorization while preserving some connotative connections. As shown in Table 1, various n-gram methods are utilized to preserve different implicit inner relation from origin strings.

Table 1. Various n-gram forms in our scheme

N-gram methods	Value	Description
String	secure	The original keyword
Counting uni-gram [8]	s1, e1, c1, u1, r1, e2	Preserve repetitions
Bi-gram	#s, se, ec, cu, ur, re, e#	Preserve adjacent letters
Tri-gram	sec, ecu, cur, ure	Preserve triple adjacent letters
Prefix and suffix	@s, e@	Beginning and ending of sentence

In general, bi-gram is the most common converting method which maintains the connotative information between adjacent letters. However, each change of single letter will double influence bi-gram results and cause reduction of matching probability. The counting uni-gram preserves repetitions and benefits on letter-confused comparison cases, such as misspelling of a letter, missing or adding a letter and reversing the order of two letters. However, it reduces the degree of constraint along with increasing false positives. The tri-gram is a more strict method which only suits the specific scene like existing judgment. The prefix and suffix preserve the beginning and ending of data to meet edge-searching.

B. Bloom-Filter. The Bloom-filter is a compact structure reflecting whether specific elements exist in prepared union. In our schema, we introduce this algorithm to judge existence about maximized substring fragments and represent the sparse vector through decimal numbers in separated columns. Given words fragments set $\mathcal{S} = \{e_1, \ldots, e_{\#e}\}$, a bloom-filter maps each element e_i into a same l-bit sparse array by k independent hash functions. Positive answer is provided only if all bits of matched positions are true.

C. Locality Sensitive Hashing. The locality sensitive hashing (LSH) algorithm helps reduce the dimension of high-dimensional data. In our schema, we introduce this algorithm to map similar items together with high probability. Besides, the specific manifestation of the algorithm is different under different measurement standards. However, there is no available method for levenshtein distance among text. So that a common practice is converting texts to fragment sets with n-gram methods.

Definition 1 (Locality sensitive hashing)**.** *Given a distance metric function D, a hash function family $\mathcal{H} = \{h_i : \{0,1\}^d \rightarrow \{0,1\}^t | i = 1, \ldots, M\}$ is (r_1, r_2, p_1, p_2)-sensitive if for any $s, t \in \{0,1\}^d$ and any $h \in \mathcal{H}$ satisfies:*

if $D(s,t) \leq r_1$ then $Pr[h_i(p) = h_i(q)] \geq p_1$;

if $D(s,t) \geq r_2$ then $Pr[h_i(p) = h_i(q)] \leq p_2$.

For nearest neighbor searching, $p_1 > p_2$ and $r_1 < r_2$ is needed. Practically, feasible permutations are generated through surjective hashing functions with our security parameter λ. And the minhash algorithm helps map fragment sets of every separated words which achieves similar searching.

3.2 Functional Model

Let $D = (d_1, \ldots, d_{\#D})$ be sensitive row data (each line contains some words respectively, as $d_i = \bigcup_{j=1}^{|d_i|} w_j^i$) and $C = \{c_{det}, c_{lsh}, c_{bf}\}$ be the corresponding cipher-texts. Two types of indexing methods are enforced: the first one achieves similar searching among words through dimension reductions with locality sensitive hashing (let m be the dimension of LSH, n be the tolerance and $\mathcal{L}$ represents its conversion); the last one achieves maximum substring matching through bit operation with bloom-filter (let l be the length of vector space, k be the amount of hashing functions and $\mathcal{B}$ represents its conversion). We consider LSH tokens set $T_i = \mathcal{L}_m^n(\bigcup_{j=1}^{|d_i|} \mathcal{G}_{ss}(w_j^i))$ be the elementary ciphers for c_{lsh}, and BF vector $V_i = \mathcal{B}_l^k(\bigcup_{j=1}^{|d_i|} \mathcal{G}_{msm}(w_j^i))$ be the ciphers of whole continuous sequence for c_{bf}. Besides, $\mathcal{G}$ represents n-gram methods for similar searching or maximum substring matching.

Definition 2 (Fuzzy searching encryption)**.** *A proxy-based encrypted database implements fully fuzzy searching with rewriting SQL statements through the following polynomial-time algorithms:*

$(K_{det}, \mathcal{L}_m^n, \mathcal{B}_l^k) \leftarrow \text{KeyGen}(\lambda, m, n, l, k)$: Given security parameter λ, dimension m of LSH and tolerance n, vector length l of BF and hash amount k, it outputs a primary key K_{det} for determining encryption, $\mathcal{L}_m^n$ for LSH, $\mathcal{B}_l^k$ for BF. The security parameter λ helps initialize the hash functions and randomization processes.

$(c_{det}, T_i, V_i) \leftarrow \text{Index}(d_i, \mathcal{L}_m^n, \mathcal{B}_l^k)$: Given the LSH function $\mathcal{L}_m^n$ and the BF function $\mathcal{B}_l^k$, the plain-text d_i is encrypted to determined cipher-texts c_{det}, ciphers T_i for similar searching and ciphers V_i for maximum substring matching respectively.

$(c_{det}||T_i||V_i) \leftarrow \text{Trapdoor}(expression)$: Given the query expression analyzed from 'like' clause, the adaptive rewriting method help generate representing elements out of different considerations with wild-cards condition. The determined cipher-texts would return in next step over encrypted database and K_{det} helps decryption.

As shown in definition of fuzzy searching encryption, we mainly emphasize transformation processes like building, indexing and executing. There exist other functional methods such as updating, deleting to achieve dynamically of our schema. It is applicable for outsourced encrypted database through rewriting SQL statements including 'create', 'insert', 'select' and so on.

3.3 Security Notions

Our security definition follows the widely-accepted security frameworks in this field [6,12,15,22]. It is summarized in fuzzy query over encrypted database that the overall security relies on the cryptographic assurance of indexes and trapdoors. In our schema, we store extra functional ciphers as indexes and rewrite queries as trapdoors. The security guarantee means there is no additional information leaked other than the functional results of fuzzy query.

4 Proposed Fuzzy Searching Encryption

4.1 Two Types of Functional Auxiliary Columns

The multiple-attributions-splitting design in cloud database synthesizes various encryptions to preserve query semantics. As shown in Table 2, two types of auxiliary columns (c-LSH and c-BF) are appended on cloud database along with a symmetrical determined column (DET).

Table 2. Storage pattern of multiple functional columns in database

c_{det}	c_{lsh}(m = 4, wid = 2)	$c_{bf}(1)$	...	$c_{bf}(\lceil \frac{l}{32} \rceil)$
0x1234 ("I love apple")	19030024, 01000409, 00020412	1077036627	...	1957741388
0x3456 ("lave banana")	01000409, 00020303	1079642851	...	625017556
0x5678 ("I love coconut")	19030024, 01000409, 06000700	1626500087	...	1687169793

This schema aims at handling queries with wild-cards on cipher-texts. So that several appended columns could store different functional ciphers with various encryptions, such as determination (DET) of data for equality, locality sensitive hashing (LSH) of words fragments for similar searching, bloom-filter (BF) among lines for maximum substring matching.

A. c-LSH. The c-LSH column, which stores the locality sensitive hashing values of each sentence, represents a message digest after dimensionality reduction. It

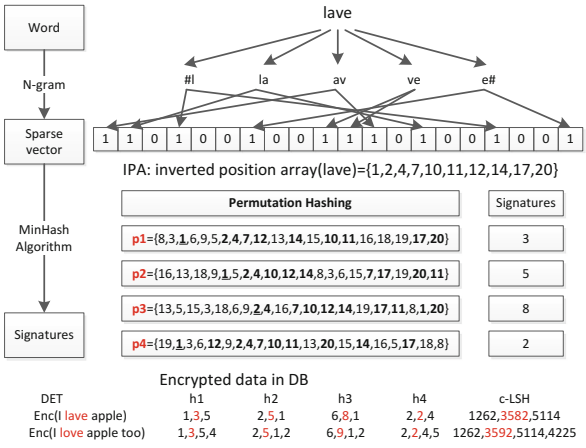

Fig. 2. A sample with bi-gram method (counting uni-gram as well) to show transforming process: (1) split sentences in line to multiple words; (2) transport a word to fragments with n-gram and build inverted position array; (3) execute dimension reduction with LSH and get m features; (4) link features to a token for each word; (5) combine tokens in line with comma.

helps map similar items together with probability which equals to the jaccard distance between their inverted position arrays (IPA for short).

During transforming process, n-gram methods are utilized (such as bi-gram and counting uni-gram) for dividing texts into fragments and finally to sparse vectors (IPA for short). As shown in Fig. 2, the transforming process maps every rows to separate signature collections by steps. This process changes measurement from levenshtein distance on texts to jaccard similarity on IPAs. So that the particular minhash algorithm could reduce the dimensions of numeric features for each subject (words). Finally, each word is converted to a linked sequence as a token and the c-LSH stores tokens set with comma to represent data of whole line.

B. c-BF. The multiple auxiliary c-BF columns, which represents macroscopic bloom-filter spaces for each row, are implemented on several 'bigint' (32-bit) columns. The database will return the DET ciphers where all c-BF columns cover the target sequences through native bit arithmetic operation '&'. Briefly, these columns are proposed for maximum substring matching which is a supplement to the c-LSH column above.

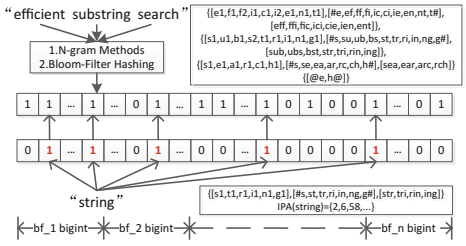

Fig. 3. The maximum substring matching over c-BF vectors which are stored in multiple 'bigint' auxiliary columns separately. After mapping fragments from whole sentence to vectors, queries execute with bit matching.

During mapping process, we respectively generate vectors for each row through bloom-filter hashing with following n-gram methods: bi-gram, tri-gram, prefix and suffix. These auxiliary columns are designed for substring matching so that the implicit information need be maximally persisted from origin strings. Through matching fragments between target bit vector and stored separated 'bigint' numbers, we could obtain all matched rows as shown in Fig. 3.

To meet application scenarios of inextensible cloud database, we accomplish operations completely through rewriting SQL statements by native bit arithmetic operation over multiple auxiliary columns, such as *select m_det from t where m_bf0&1=1 and m_bf1&3=3*. We experiment the connection between length of the sparse vector and correct rate of maximum substring matching in Sect. 5.

4.2 Adaptive Rewriting Method over Queries with Wild-Cards

In SQL, wild-card characters are used in 'like' expression: the percent sign '%' matches zero or more characters and the underscore '_' matches a single character. Usually the former symbol is a coarse-grained comparable delimiter and the latter could be tolerated by locality sensitive hashing in slightly different cases. So we construct an adaptive rewriting method over queries with wild-cards as shown in Fig. 4.

We consider three basic cases according to the number of percent signs to meet indivisible string fragments. Furthermore in every basic case, we also divide three sub-cases according to the number of underscore to benefit from different auxiliary columns. Besides, each query text is considered as whole word and substring while experiment exhibits the optimal selection.

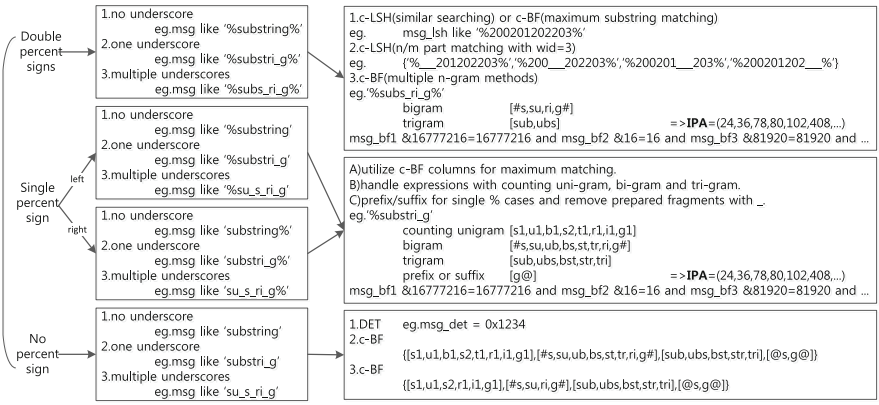

Fig. 4. Adaptive rewriting method over queries with wild-cards. We consider percent sign as a coarse-grained separator and few underscore could be tolerated according to similarity.

Firstly, the double percent signs case means that user attempts finding rows which contains the given string. Because the LSH function could tolerate small differences naturally, the sub-case with no underscore could accomplish similar searching among whole words. We achieve the one underscore sub-case with part matching method. This clever trick helps adjust fineness of similar searching as shown in Fig. 5. The multiple underscores sub-case is achieved by maximum substring matching on c-BF columns with bloom-filter.

Secondly, the single percent sign case need to consider prefix and suffix. The occurs of this type of queries reflect more detailed information and we match them all as substrings with maximum degree of constraint through various N-gram forms on c-BF column. Meanwhile, the prefix and suffix help preserve beginning and ending information of whole sentences in row. During splitting process, every fragments with underscore would be abandoned and the rest part would be mapped to the sparse bloom filter space which represented by IPA.

Thirdly, in the last no percent sign case, the user might already obtain most of target information and attempt to match specific patterns with underscore. Besides no underscore sub-case could be treated as determining equality operation, the maximum substring matching on c-BF column could meet the rest sub-cases' requirements.

Additionally, the tolerance parameter n is proposed as a flexible handler under the dimension m of locality sensitive hashing auxiliary column. Briefly, every features of word are set as fixed-length numbers which is filled by zero in basic scheme. As a linked string with all m features, the token could be converted to different variants where some feature parts replacing with underscores. We joint every possible cases together for database searching with keyword 'or' through a called bubble function as shown in Fig. 5.

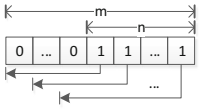

Fig. 5. The part matching method represents the adjustable fineness in c-LSH column with the tolerance parameter n and the LSH dimension m. For instance, let $m = 4, n = 3$ and the target feature set be $\{1, 2, 3, 4\}$, therefore the candidate set is $\{\_234, 1\_34, 12\_4, 123\_\}$ by this method.

The adaptive rewriting method helps generate trapdoor queries to meet the wild-cards fuzzy searching encryption in database through similar searching on c-LSH column and maximum substring matching on c-BF columns.

4.3 LSH-Based Security Improvements

The security of our schema relies on three parts. The symmetric cryptography algorithm guarantees the security on determining column and the divided bloom-filter vectors are presented by unidentifiable hashing ciphers. However, the content in c-LSH column might leak some extra information such as sizes and sequences of plain-texts. We present three improvements to enhance security and an integrated algorithm as followed.

A. Linking Features Without Padding
In basic scheme, we pad each feature with zero by the upper limit *wid* which benefits selecting process. To enhance security, we cancel the zero padding before linking features to a token. Meanwhile, the part matching method is also changed to an analogous bi-gram form. For instance, a secure enhanced part matching method is *select m_det where m_lsh like '%ab%' or m_lsh like '%bc%'* where *a*,*b*,*c* are multiple features of a word. We discuss the validity with experiments.

B. Modifying Sequences of Tokens
Each line of c-LSH auxiliary column stores a tokens set for whole sentence. Therefore, the sequences of tokens might exhibit the relevancy among words

to malicious attacker. To overcome this leakage, we modify the sequences randomly by hashing permutations. Additionally, we implement the permutation function with $\mathcal{P} : y = a * x + b \mod c$ where a is relatively-prime to c. This improvement protects the relation between invisible words and specific tokens. Since the matching only demands on existing rather than order, so this sequence modification helps for security protection.

C. Appending Confusing Tokens

The tokens sets in row leak the size of words. Appending tokens is a practical way for security, but what kind of token content should be added is the target of our discussion. The first way is appending repeated tokens from itself. It is simple and effective, but it only improves limited security. The second way is appending a little random tokens. Because of sparsity and randomization, few random tokens might not change the matching results. The third way is appending tokens combined from separated features among this tokens set. This way also influences the matching precision and increases proportion of false positive. Actually, these ways help greatly enhance security despite of disturbances.

D. Integrated Security Enhanced Algorithm

We present an integrated algorithm for security enhancement which combines all above implementations. As shown in Algorithm 1, this algorithm transforms the tokens set in each row to an security enhanced one. It helps prevent information leakage from c-LSH column.

Algorithm 1. Security Enhanced Improvements

Input: $token_m[\#word_i]$ which represents the m-dimensional tokens set of line i, wid be width of feature with zero padding, $amount$ be the lower bound for appending tokens

Output: an optimal security enhanced set $e\_token[amount]$

Let t represent token and each t can be split into m features by wid;
Generate a permutation function with $\mathcal{F} : y = a * x + b \mod c$ where $c = amount$ and $(a, c) = 1$;
Let $c = 0$ be the count for permutation;
foreach t *in* $token_m[\#word_i]$ **do**
    Generate a temporary string et;
    for *int j=0; j<m; j++* **do**
        Remove the zero prefix of $t.substr(j * wid, (j + 1) * wid)$;
        Link it to et;
    $e\_token[\mathcal{F}(c++)]=et$;
while $c < amount$ **do**
    Generate a temporary string et; **for** *int k=0; k<m; k++* **do**
        Get a feature $token_m[random()].substr(k * wid, (k + 1) * wid)$;
        Remove the zero prefix and link it to et;
    $e\_token[\mathcal{F}(c++)]=et$;
return $e\_token[amount]$;

5 Performance Evaluation

In this section, we evaluate the performance of our work. Firstly, we discuss the effect of different n-gram methods about matching accuracy in c-LSH column. Secondly, we discuss the effect of bloom-filter length on collision degree and space usage of maximum substring matching. Thirdly, we discuss performance of adaptive rewriting method. Finally, we compare execution efficiency and space occupancy among efficient proposed schemes. The proposed scheme is implemented in Core i5-4460 3.20 GHz PC with 16 GB memory, and the used datasets include 2000 TOEFL words, the leaked user data of CSDN and the reuters news.

Manifestations of Different N-gram Methods on c-LSH Column. Utilizing bi-gram and counting uni-gram, we achieve similar searching on c-LSH column by introducing minhash algorithm based on the jaccard distance of fragments set. Intuitively, every change of character would greatly influence the corresponding fragments union over bi-gram method. So we introduce the counting uni-gram method to balance this excessiveness relativity. In this experiment, we evaluate the performance of these two n-gram methods and the combined one respectively.

The dataset we used is a 2000 TOEFL words set and we construct three variants of them to reveal the efficiency about LSH-based similar searching under different N-gram methods. The ways getting variants include appending a letter in the middle or in one side for every words, such as 'word' into 'words','wosrd','sword'. We calculate the average matched rows to reflect the searching results.

As shown in Fig. 6, we choose $m = 4, 6, 8$ to reveal matched numbers through part matching method with n. And the accuracy rate has a big promotion when n is larger than half of m. Besides, the combined method performs well when $m \geq 6$. It is reflected about the variation trend of accuracy that the amount of false positive reduces while the correct items remain unchanged.

The Bloom-Filter Length on c-BF Columns. In second experiment, we valuate collision accuracy and space occupancy under impacts of bloom-filter length and hashing function amount on c-BF columns when executing maximum substring matching. In detail, we attempt to find out an appropriate setting about the number of hashing function and the vector length of our bloom-filter structures.

The dataset we used is a leaked accounts set about CSDN, one of the most famous technical forum websites in China, and contains user name, password and e-mail. To guarantee effectiveness and avoid collisions, we change the vector length and keep the sparsity in several degrees such as half, quarter, one-sixth and one-eighth. Meanwhile different amount of hashing functions in bloom-filter influence accuracy and collision.

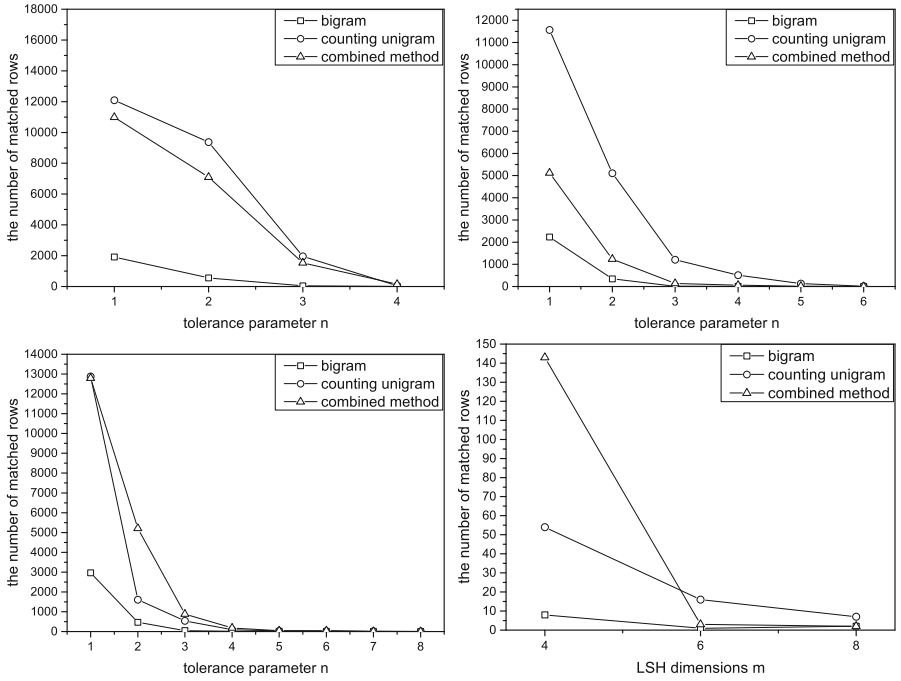

Fig. 6. The matching size under the tolerance parameter n and the LSH dimension m over variants of TOEFL words set. The first three graphs show the performance of different n-gram methods about part-matching respectively. The last graph shows the LSH dimension only complete-matching when $m = n$.

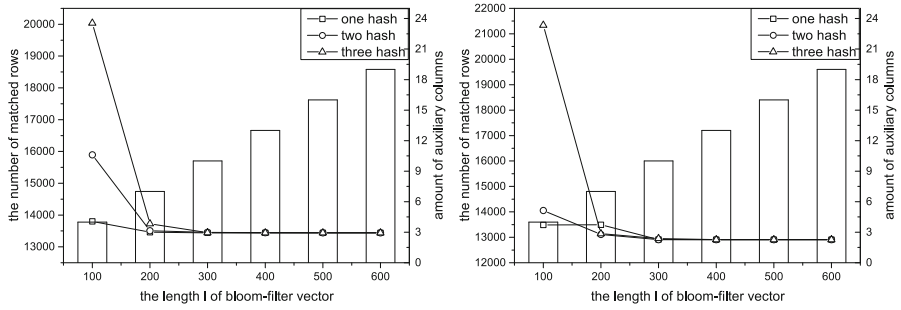

Fig. 7. The experiments show performance of maximum substring searching under different bloom-filter length l and different hashing amount k. We utilize fifty thousand rows of leaked CSDN account data and set several degrees of sparsity about bloom-filter vector while each row contains 50 characters. The left graph shows matching sizes of substring '163.com' on $\lceil \frac{l}{32} \rceil$ auxiliary columns when we build indexes under different length of bloom-filter. And the right graph represents 'qq.com'.

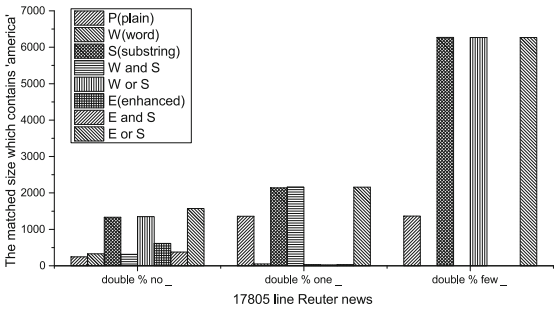

Fig. 8. The experiment shows the performance of adaptive rewriting method under different combinations, and reveals the most qualified modes for each fuzzy searching cases. Some expressions are used, such as '%america%', '%am_rica%', '%am_ri_a%'.

Because the bloom-filter length l corresponds to the amount of c-BF columns, this experiment discuss relations between matching accuracy and space occupancy under different amount of bloom-filter hashing functions. As shown in Fig. 7, the amount of matching size drops rapidly in the first place and then gets stable when sparsity is close to one-sixth.

The Performance of Adaptive Rewriting Method. This experiment aims at verifying effectiveness of the adaptive rewriting method. After auxiliary columns storing values as indexes, the 'like' clauses with wild-cards are analyzed by an adaptive rewriting method and rewritten to trapdoors. In this experiment, we consider the content of expression as a word or substring for comparison, and execute different types of queries with basic and security enhanced schemes respectively.

The dataset we used is Reuters-21578 news of 1987 [28]. In this experiment, we mainly discuss the double '%' cases because the other single '%' and no '%' cases carry out analogous steps. The only difference is that these cases additionally consider the prefix and suffix.

As shown in Fig. 8, we compare the matched size under different combinations. We also execute the origin SQL statements on extra stored plain-text column for contrast. It helps find the best combination modes under various wild-cards cases. We accomplish this experiment with the sparsity of c-BF columns being one-sixth and the dimension of c-LSH column being six. The graph shows that 'W and S' is fit for double '%' no '_' and double '%' one '_' cases while 'S' is fit for double '%' few '_' case. Besides, we discuss the performance of LSH-based security enhanced method and the graph confirms its feasibility.

Performance Comparison Among Proposed Schemes. In this section, we compare the efficiency of proposed schemes about inserting and selecting data. In general, the inserting process involves generating indexing values in auxiliary columns, and the selecting process involves decrypting determined cipher-texts.

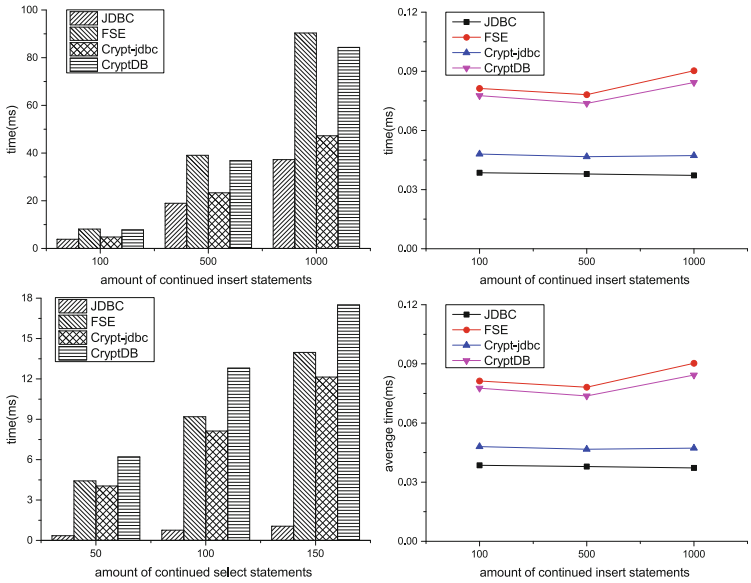

Fig. 9. This experiment show the execution efficiency among proposed schemes.

As shown in Fig. 9, our schema verifies this point and performs well comparing to normal JDBC, Crypt-jdbc and CryptDB.

6 Conclusion

This paper investigates the problem about fuzzy searching encryption with complex wild-cards queries on proxy-based encrypted database, then gives a practical schema with two types of auxiliary columns and rewriting SQL statements. Besides, security enhanced implementations and extensive experiments show the effectiveness. In future, the serialization and compression of functional ciphertexts would be studied to reduce space overhead.

References

1. Boldyreva, A., Chenette, N.: Efficient fuzzy search on encrypted data. In: Cid, C., Rechberger, C. (eds.) FSE 2014. LNCS, vol. 8540, pp. 613–633. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46706-0_31
2. Cash, D., Jaeger, J., Jarecki, S., Jutla, C., Krawczyk, H., Rou, M.C., Steiner, M.: Dynamic searchable encryption in very-large databases: data structures and implementation. In: Network and Distributed System Security Symposium (2014)
3. Cash, D., Jarecki, S., Jutla, C., Krawczyk, H., Roşu, M.-C., Steiner, M.: Highly-scalable searchable symmetric encryption with support for boolean queries. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 353–373. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_20

4. Chase, M., Kamara, S.: Structured encryption and controlled disclosure. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 577–594. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_33
5. Chen, H., Tian, X., Yuan, P., Jin, C.: Crypt-JDBC model: optimization of onion encryption algorithm. J. Front. Comput. Sci. Technol. **11**(8), 1246–1257 (2017)
6. Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: ACM Conference on Computer and Communications Security, pp. 79–88 (2006)
7. Fan, K., Yin, J., Wang, J., Li, H., Yang, Y.: Multi-keyword fuzzy and sortable ciphertext retrieval scheme for big data. In: 2017 IEEE Global Communications Conference, GLOBECOM 2017, pp. 1–6. IEEE (2017)
8. Fu, Z., Wu, X., Guan, C., Sun, X., Ren, K.: Toward efficient multi-keyword fuzzy search over encrypted outsourced data with accuracy improvement. IEEE Trans. Inf. Forensics Secur. **11**(12), 2706–2716 (2017)
9. Hahn, F., Kerschbaum, F.: Searchable encryption with secure and efficient updates. In: ACM SIGSAC Conference on Computer and Communications Security, pp. 310–320 (2014)
10. Jho, N.S., Chang, K.Y., Hong, D., Seo, C.: Symmetric searchable encryption with efficient range query using multi-layered linked chains. J. Supercomput. **72**(11), 1–14 (2016)
11. Kamara, S., Papamanthou, C.: Parallel and dynamic searchable symmetric encryption. In: Sadeghi, A.-R. (ed.) FC 2013. LNCS, vol. 7859, pp. 258–274. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39884-1_22
12. Kamara, S., Papamanthou, C., Roeder, T.: Dynamic searchable symmetric encryption. In: ACM Conference on Computer and Communications Security, pp. 965–976 (2012)
13. Kurosawa, K., Ohtaki, Y.: UC-secure searchable symmetric encryption. In: Keromytis, A.D. (ed.) FC 2012. LNCS, vol. 7397, pp. 285–298. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32946-3_21
14. Kurosawa, K., Sasaki, K., Ohta, K., Yoneyama, K.: UC-secure dynamic searchable symmetric encryption scheme. In: Ogawa, K., Yoshioka, K. (eds.) IWSEC 2016. LNCS, vol. 9836, pp. 73–90. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-44524-3_5
15. Kuzu, M., Islam, M.S., Kantarcioglu, M.: Efficient similarity search over encrypted data. In: IEEE International Conference on Data Engineering, pp. 1156–1167 (2012)
16. Lesani, M.: MrCrypt: static analysis for secure cloud computations. ACM SIGPLAN Not. **48**(10), 271–286 (2013)
17. Li, J., Liu, Z., Chen, X., Xhafa, F., Tan, X., Wong, D.S.: L-EncDB: a lightweight framework for privacy-preserving data queries in cloud computing. Knowl.-Based Syst. **79**, 18–26 (2015)
18. Liu, Z., Li, J., Li, J., Jia, C., Yang, J., Yuan, K.: SQL-based fuzzy query mechanism over encrypted database. Int. J. Data Wareh. Min. (IJDWM) **10**(4), 71–87 (2014)
19. Liu, Z., Ma, H., Li, J., Jia, C., Li, J., Yuan, K.: Secure storage and fuzzy query over encrypted databases. In: Lopez, J., Huang, X., Sandhu, R. (eds.) NSS 2013. LNCS, vol. 7873, pp. 439–450. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38631-2_32
20. Popa, R.A., Li, F.H., Zeldovich, N.: An ideal-security protocol for order-preserving encoding. In: IEEE Symposium on Security and Privacy, pp. 463–477 (2013)

21. Popa, R.A., Redfield, C.M.S., Zeldovich, N., Balakrishnan, H.: CryptDB: protecting confidentiality with encrypted query processing. In: ACM Symposium on Operating Systems Principles, SOSP 2011, Cascais, Portugal, October, pp. 85–100 (2011)
22. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: IEEE Symposium on Security and Privacy, p. 44 (2000)
23. Wang, B., Yu, S., Lou, W., Hou, Y.T.: Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud. In: 2014 Proceedings of IEEE INFOCOM, pp. 2112–2120 (2014)
24. Wang, C., Cao, N., Li, J., Ren, K.: Secure ranked keyword search over encrypted cloud data. In: IEEE International Conference on Distributed Computing Systems, pp. 253–262 (2010)
25. Wang, J., Ma, H., Tang, Q., Li, J., Zhu, H., Ma, S., Chen, X.: Efficient verifiable fuzzy keyword search over encrypted data in cloud computing. Comput. Sci. Inf. Syst. **10**(2), 667–684 (2013)
26. Wang, Q., He, M., Du, M., Chow, S.S.M., Lai, R.W.F., Zou, Q.: Searchable encryption over feature-rich data. IEEE Trans. Dependable Secur. Comput. **PP**(99), 1 (2016)
27. Wei, X., Zhang, H.: Verifiable multi-keyword fuzzy search over encrypted data in the cloud. In: International Conference on Advanced Materials and Information Technology Processing (2016)
28. Wiki: Reuters. http://www.research.att.com/~lewis