Variable Length Indistinguishable Binary Tree For Keyword Searching Over Encrypted Data

Heyu Wang, Qin Jiang, Hui He, and Yong Qi

School of Electrical and Information Engineering
Xi'an Jiaotong University
Xi'an 710071, China
E-mail:why615347842@stu.xjtu.edu.cn,qjiang16@stu.xjtu.edu.cn
huihe@mail.xjtu.edu.cn,qiy@mail.xjtu.edu.cn

Abstract. We consider the problem of privacy-preserving keyword search on the clouds. In this paper, we propose a new searchable encryption scheme that satisfies efficient and secure query processing. In particular, our scheme utilizes a novel data structure called *Indistinguishable Bloom Filter(IBF)* to construct the first *variable-length Indistinguishable Binary Tree(vIBtree)* which can compress storage of the data structure and support efficient private keyword search. Meanwhile, we propose a corresponding trapdoor algorithm for *vIBtree*. Further more, we analyze that our scheme is secure under the adaptive IND-CKA security model. We implemented our scheme in C language. Our experiments show that *vIBtree* can speedup the construction time and reduce the storage overhead for the data structure over known, state of the art alternatives.

Keywords: Searchable Encryption(SE) · Indistinguishable Bloom Filter(IBF)Binary Tree(IBtree)computing

1 Introduction

1.1 Motivation

The cloud has significantly matured in the last several years. It centralizes unlimited resources and delivers elastic services to end-users. For reasons of cost and convenience, users often store their data on remote cloud servers that may offer better connectivity. However, privacy has become the key concern as data owners may not fully trust public clouds, including concerns about data security and user's privacy. The untrusted server tends to contribute to privacy and security issues. For example, a Google engineer broke into Gmail and Google Voice accounts of several children in 2010.

As a solution, the data is encrypted before outsourcing to a cloud service[2] [3][19]. However, data encryption makes it hard to retrieve data selectively from the server. The usual searching analogy could not get through the encrypted data. To address this issue, many Searchable Encryption(SE) schemes have been proposed, which usually consists of the following steps: A data owner generates

its encrypted document and the encrypted index. The encrypted document and the encrypted index are then outsourced to the cloud. To search a keyword, a data user generates a so-called trapdoor. With the trapdoor, the cloud can search on the encrypted index and return related documents.

Unfortunately, it is insufficient to protect the data using encryption, there are side channels for the adversary to get valid information. For instance, by adaptively select the queries sent to the server, the attacker can infer the content of the queries. Moreover, it is unwise to store the data structures without considering the storage as a large space can increase the attack suffice for the server and maintenance cost for the clients.

1.2 Threat Model

We assume that a server is semi-honest. The attackers on the system can observe the interaction between the client and the server and can not prevent the schemes executing correctly but attempt to obtain valid information such as the content of the user queries, and the data items. We also assume that the user communicates with the server through a secure channel and the user is not compromised. In our implementation, the security of cryptographic primitives is also assumed.

1.3 Security Goal

Let $VBT = (VBT\_Setup, VBT\_Enc, VBT\_Tok, VBT\_Query, VBT\_Dec)$ be searchable scheme, k be the security parameter, ζ ={**Gen**^{sym}, **Enc**^{sym}, **Dec**^{sym}} be the IND-CPA encryption scheme. We now describe our VBT scheme as follows:

- $(SK)\leftarrow$ **VBT_Setup**(1^k, S). Take k and S=(D_1, D_2,..., D_n) as input. Use input k to execute **Gen**^{sym} two times and output SK_I, SK_v. SK_v is used for encrypt the data items. SK_I is used to encrypt the variable bloom filer tree.In addition, we set SK= (SK_I, SK_v)
- $(E\_IV, E\_S) \leftarrow$ **VBT_Enc**(SK, IV, S). Take SK, inverted index IV ,document collection S=(D_1, D_2,...,D_n) as input. Use input SK_I, SK_v execute Enc^{sym} to encrypt the IV, S and output E_IV, E_S respectively.
- $t \leftarrow$ **VBT_Tok**(SK_I, R). Use input SK_I and R=(key_1 and key_2 and key_3 and ... and key_m) to calculate t using **Enc**^{sym}.
- C_Reuslt$\leftarrow$ **VBT_Query**(t, E_IV). Use input search token t, and encrypted inverted index E_IV to perform query operations and output the encrypted ID of the documents satisfying the search token t.
- $result \leftarrow$ **VBT_Dec**(SK_I, C_Reuslt).$Useinput SK_I$ and E_Reuslt to calculate the search result $result$ using **Dec**^{sym}.

Similar to [1], we assume that the level security of VBT is adaptive $IND-CKA$ security model. More specifically, the main challenge for VBT is to construct an index I which is adaptive $IND-CKA$. In particular, the challenge is to proof that there exists a simulator S which can simulate I not only for a set of known queries but also for the unknown queries.

1.4 Our Contributions

In this paper, we study the fundamental problem of processing keyword queries in a privacy-preserving manner. Our main contributions are summarized as follows.

First, we propose an SSE based keyword query scheme that satisfies requirements of efficient query processing and adaptive security. We utilize a data structure IBF to construct the first variable length IBtree as shown in Figure 4.

Second, we extend the trapdoor computation algorithm for fixed-length IBtree to supporting searching over our variable-length IBtree. Data users who share the same secrete keys with data owner can generate a trapdoor according to our algorithm, by sending the trapdoor to the cloud they can perform a keyword search over the encrypted data items in $O(logn)$ time, where n is the total number of data items inserted into our variable-length IBtree. The index reveals no information about its contents without valid trapdoors.

Third, we implement both our scheme and prior fixed-length IBtree in scheme [1] in C language.[1] We evaluate our scheme with the fixed-length scheme on three aspects: IBtree construction time, IBtree memory usage and keyword searching time. Experimental results show that our scheme significantly speeds up IBtee construction time and reduces IBtree memory overhead compared with the state of the art.

2 Related Work

Song et al. developed a Searchable Symmetric Key Encryption (SSKE) scheme [4] that allows a user, given a trapdoor for a word, to test if a ciphertext block contains the word. Boneh et al. developed Searchable Public Key Encryption (SPKE) [5], which is the public key equivalent of SSKE. But their schme is inefficient because the server performs a linear scan through all data on every query. In [6], Goh presents an efficient scheme for keyword search over encrypted data using Bloom filters. Determining whether a document contains a keyword can be done securely in constant time. However, the scheme does not support secure conjunctive search. Curtmola et al. [7] proposed the first sub-linear scheme, which is more suitable for a static document set than a dynamic one. [8].

A significant amount of work has been done in privacy preserving keyword queries[9]-[18]. In [9], Qi et al. proposed a compressed and private data sharing framework that provides efficient and private data management. In [10], S.Kamara et al. proposed a SSE scheme that satisfies sublinear search time, security against adaptive chosen keyword attacks, compact indexes and the ability to add and delete files efficiently. In [11] and [12], they used deterministic encryption to enable search on encrypted data with existing database and search techniques. Jiang et al. [21]proposed oblivious data structure to support efficient boolean queries using Intel SGX. However it focuses on access pattern leakage and does not consider the adaptive security for the data structure which is

[1] https://github.com/HeyuWang/Indistinguishable-bloom-filters

important for the private search. Li et al. [1] proposed a data structure *Indistinguishable Bloom Filter* for storing index elements, and utilized it to construct a fixed-length *Indistinguishable Binary Tree(IBtree)*. However, the cost of a fixed-length IBtree scheme in terms of build tree time and memory usage increases quickly with the number of leaf nodes.

3 Background

3.1 Notations and Terminology

This section explains the notation and terminology used in this paper.
D=$\{d_1, d_2, ..., d_n\}$—A collection of n data items.
W_i=$\{\omega_1,\omega_2,\cdots,\omega_n\}$—A keyword set of a given data item d_i.
$HMAC_{K_i}(.)$ [22] —A keyed hash function which is used to hash a keyword ω_i, SHA-256 is used in our HMAC. The outcome are 256 bits.
$SHA1(.)$—A hash function that is used to determine the "chosen" cell position in an IBF, the outcome are 160 bits.
B: An array B of m twins which stands for the IBF.

3.2 System Model

In this paper, we consider a cloud-data-sharing system consisting of three entities, i.e., data owners, cloud server and data users. In particular, we consider Searchable Symmetric Encryption(SSE) schemes as symmetric encryption based privacy-preserving schemes are significantly more efficient than asymmetric ones. In SSE, a data owner builds a *secure index* I for a data set D and encrypts each data item $d_i \in D$ into $((d_i)_K)$ using a secret key K that is shared between the data owner and data users. Then the data owner outsources the secure index I along with the set of encrypted data $\{(d_1)_K,(d_2)_K,\cdots,(d_n)_K\}$ to the cloud. Given a keyword query q, the data user generates a *trapdoor* t_q for q and sends t_q to the cloud. Base on t_q and the secure index I, the cloud can determine which encrypted data items satisfy q without knowing the content of data and query. In this process, the cloud should not be able to infer privacy information about data items and queries such as data content, query content, and the statistical properties of attribute values. The index I should leak no information about the data items in D.

3.3 Indistinguishable Bloom Filter(IBF)

To achieve adaptive security and privacy-preserving query scheme, we adopt a new data structure called IBF. Li *et al.*[1] proposed Indistinguishable Bloom Filter(IBF) in 2017. As figure 1 shows, an IBF is an array B of m twins. Each twin consists of two cells where each cell stores one bit, and the two bits remain opposite. Figure 1 shows an initialized IBF with 11 twins(22 bits) without any encrypted keyword added. Two cells of each twin are not equivalent; one of them

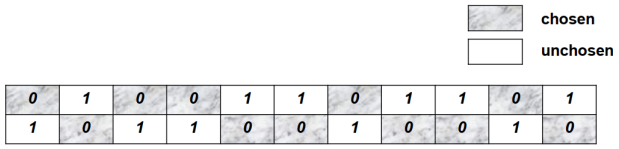

Fig. 1. An initialized IBF without any keyword inside with 11 twins.

is labelled with "chosen", which is the bit used to store information. The other is the "unchosen" cell that is used to mask information inside this twin.

As shown in figure 2, after inserting the keyword inside the IBF, the bit in "chosen" cell of each twin is one, and the bit in "unchosen" cell of each twin is 0, the other unmapped position keeps the "chosen" cell 0 and "unchosen" cell 1. Compared with BF, IBF chooses a random cell to store the information that the position is mapped by associating a random number with it, which also eliminates the correlation among different IBFs. IBF construction algorithm can be found in [1].

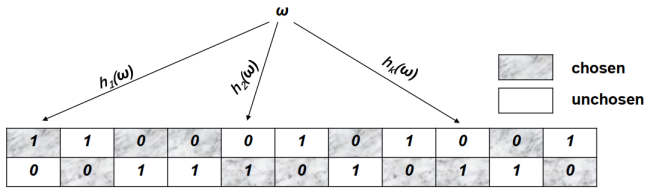

Fig. 2. An IBF which has been inserted.

False Positive Rate of Indistinguishable Bloom Filter : According to [6], after using k hash functions to insert n distinct elements into an array of size m, the probability that bit i in the array is 0 is $(1-(1/m))^{kn} \approx e^{-kn/m}$. Therefore, the probability of a false positive is $(1-(1-(1/m))^{kn})^k \approx (1 - e^{-kn/m})^k$. Since m and n are typically fixed parameters, we compute the value of k that minimizes the false positive rate. The derivative of the right hand side of the equation with respect to k gives a global minimum of $k = (ln2)(m/n)$, with a false positive rate of $(1/2)^k$. We can round k to the nearest integer.

3.4 Fixed-Length Indistinguishable Binary Tree

In a fixed-length indistinguishable binary tree, each terminal and nonterminal node is an IBF of the identical size. We briefly described their IBtree construction algorithm here. Details about corresponding trapdoor algorithm and query process can be found in [1]. Li et al. construct an IBtree in two stages. First, they insert each keyword into a separate leaf IBF node. Second, they construct each nonterminal IBF node by merging its left and right children.

As shown in figure 3, they first insert $W(d_1)$, $W(d_2)$, $W(d_3)$ into the terminal IBF nodes 4, 5, 6 respectively. Then they construct IBF node 2 through the union of node 4 and node 5. They repeat the similar merging process until the root node has been constructed.

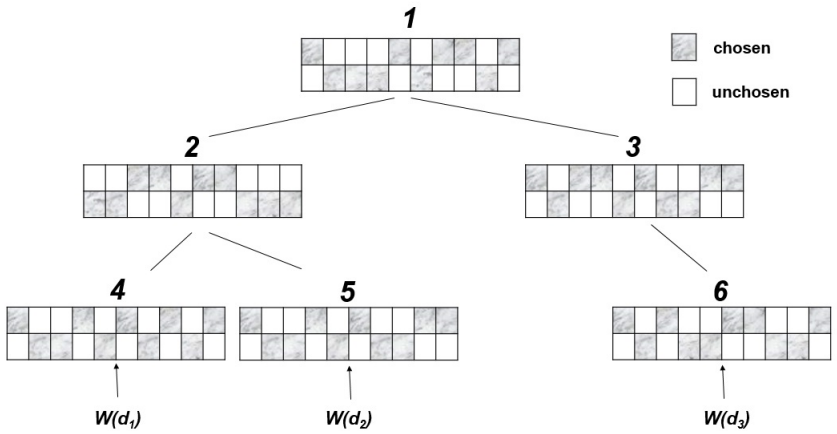

Fig. 3. A Fixed length IBtree.

4 Variable Length IBtree Scheme

4.1 Variable Length IBtree

A variable-length IBtree is a binary tree. Each node is a constructed IBF. A children IBF node is only half the size of its parent node and stores approximately half the keywords of its parent. Figure 4 shows a simple variable-length IBtree a root node size of 12 twins.

Variable Length IBtree construction: In a fixed-length IBtree, each IBF node occupies the same memory as the root IBF node. Thus the memory required at each level increases exponentially, which is extremely a waste of storage because each child IBF node represents a keyword set that is approximately half of its parent. Besides, twins need to be initialized also increases exponentially. The initialization stage requires three hash operations for each twin. Therefore the fixed-length IBtree spend a tremendous amount of time on the initializing stage. Considering these two severe disadvantages, we propose to construct a variable-length IBtree to overcome these two shortcomings. Our variable-length IBtree construction steps are listed below.

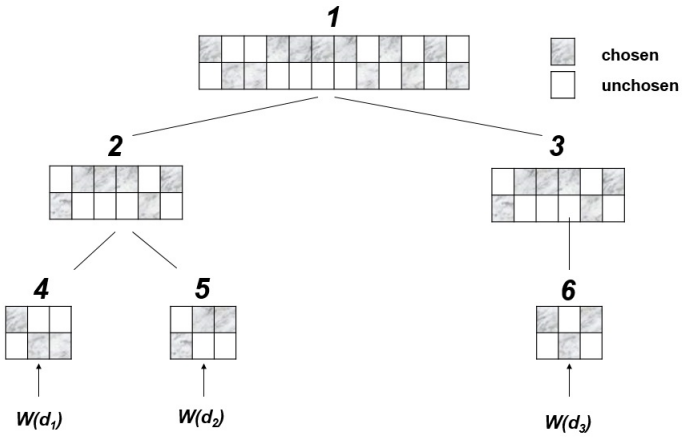

Fig. 4. Variable length IBtree.

Algorithm 1: VBT_Setup

Input: n encrypted data documents $E_k(d_i)(1 \leq i \leq n)$ and corresponding data indices (W_i), where $W_i = \{\omega_1,\omega_2,\cdots,\omega_n\}$
Output: a constructed variable length IBtree
initialization stage :
for *each node in the variable length IBtree* **do**
  1.*allocate memory*.
  2. *initialize it*.
terminal nodes insertion stage :
for *each terminal node* **do**
  insert keywords set individually into it.
nonterminal nodes insertion stage:;
for *each nonterminal node* **do**
  insert all keywords added to its left children and right children (if has) into it directly.

4.2 Trapdoor Algorithm For Variable-Length IBtree

We need a trapdoor algorithm to enable encrypted searching over the variable-length IBtree. In a fixed-length IBtree, t_q=$\{(h_1(\omega_i),\ h_{k+1}(h_1(\omega_i))),\ (h_2(\omega_i),\ h_{k+1}(h_2(\omega_i))), \cdots, (h_k(\omega_i), h_{k+1}(h_k(\omega_i)))\}$ where $h_j(\omega_i)$=HMAC$_{K_j}(\omega_i)\%m(1 \leq j \leq k+1)$. We are not allowed to use the trapdoor algorithm for fixed-length IBtree here. Because in our variable-length IBtree, the different level has different twins, which means m is different. We extend original fixed length IBtree trapdoor algorithm, supposing the variable-length IBtree has h levels, then data users need to calculate h trapdoors for each level. Denoting n is the number of data items stored in the leaf node of variable-length IBtree, $above(n)$ is the

number of all nonterminal nodes above terminal nodes. Then the height of the tree is $h = (log_2(n + above(n)) + 1)$(if we label the height of root node with 1). Thus, our variable length trapdoor t_q^j consists of $h * k$ pairs. For each level j we calculate corresponding k pairs trapdoor $t_q^j(\omega_i)$=$\{(h_1^j(\omega_i),\ h_{k+1}^j(h_1^j(\omega_i))),\ (h_2^j(\omega_i),\ h_{k+1}^j(h_2^j(\omega_i))),\ \cdots,\ (h_k^j(\omega_i),\ h_{k+1}^j(h_k^j(\omega_i)))\}(1 \le i \le n, 1 \le j \le h)$, i is used to denote different keyword ω and j is used to denote different level in the variable-length IBtree. We show our variable-length trapdoor algorithm below.

Algorithm 2: VBT_Tok

Input: a keyword ω_i
Output: a trapdoor t_q^j of $h * k$ pairs
Get the height of the variable length IBtree h **by communicating with cloud server;**
for *each height* $j(1 \le j \le h)$ **do**
  $t_q^j(\omega_i)$=**{**$(h_1^j(\omega_i),\ h_{k+1}^j(h_1(\omega_i)))$**,** $(h_2^j(\omega_i),\ h_{k+1}^j(h_2^j(\omega_i)))$**,** $\cdots$**,**
  $(h_k^j(\omega_i),\ h_{k+1}^j(h_k^j(\omega_i)))$**}**$(1 \le i \le n)$

4.3 Query Processing

After receiving a trapdoor of $h * k$ pairs for a specified keyword from data user, cloud server can search over the variable length IBtree with algorithm listed below. Cloud server also perform searching in a top-down manner, in each level $j(1 \le j \le h)$, cloud server using corresponding trapdoor $t_q^j(\omega_i)$=$\{(h_1^j(\omega_i),\ h_{k+1}^j(h_1^j(\omega_i))),\ (h_2^j(\omega_i),\ h_{k+1}^j(h_2^j(\omega_i))),\ \cdots,\ (h_k^j(\omega_i),\ h_{k+1}^j(h_k^j(\omega_i)))\}(1 \le i \le n)$ to match the specified keyword ω_i, if a match is found at root node, then it's child can be searched according to the corresponding trapdoor. For each ordered pair of hashes in t_q^j, we use the first hash value to locate the twin that the corresponding keyword is hashed to, the second hash value locates the "chosen" cell in the twin. If the cloud determines that t_q matches no data items, the query processing terminates. Otherwise, the cloud processes t_q^j in each level in a top-down manner.

Algorithm 3: VBT_Query

Input: h trapdoors for a specified keyword ω_i
Output: if a match is found, then return relevant encrypted document.
  otherwise return nothing
for *each height* $j(1 \le j \le h)$ **do**
  if *match is found in either a node* **then**
    if *is leaf* **then**
      return encrypted document
    else
      keep searching
  else
    break

4.4 Advantages and Disadvantages of our scheme

Disadvantages of Variable Length IBtree A fixed-length indeed has some advantages over our variable-length IBtree scheme. In the fixed-length scheme, data users only need to generate one trapdoor for a specified keyword ω_i, while in our scheme, data users need to generate h trapdoors, each trapdoor for each level where $h = \lceil log(n) \rceil + 1$. The overhead brought by increased trapdoors is negligible in nowadays's network bandwidth. For example, when we insert 10000 data documents into the leaf nodes of our variable-length IBtree, we only need to generate 12 more trapdoors for the other IBF node to perform the matching operation.

Advantages of Variable Length IBtree Our variable-length IBtree scheme significantly reduces IBtree construction time and IBtree memory usage. For example, if we use 10000 data items with an average keyword 100 and 7 hash functions. Thus according to $k = ln2(m/n)$, the root IBF size for both scheme is 2.4MB. Therefore, a fixed-length IBtree cost approximately 46GB. However, our variable-length IBtree only takes 36MB. Our scheme reduces memory usage by more than 1000 times in this situation. Besides, a fixed-length IBtree spend tremendous more time than our scheme in the initializing stage.

5 Security analysis

To give the security analysis, we firstly define the leakage functions based on the threat model in section 1.

L_{enc}(D): Given the document data set D, this function outputs the number of n documents, the max keywords m in the document in S, the document identifiers $ID = \{id_1, id_2, ..., id_n\}$, the number of nodes in the created index and the size of each node of the index.

L_{que}(D,T,R,t): Given the document set D, the index tree T, the search query R and given point in time t, this function outputs the search pattern at time t which contains the information about whether R is the same as queries performed before time t or not and the access pattern at time t which contains the information about which documents contain R.

Theorem 2 (Security) : The variable IBtree is IND-CAK (L_{enc}, L_{que})-security against an adaptive adversary.

Proof (Sketch). Given the security of the hash function used in the scheme, we can construct a simulator S as in [1] to build a simulated index. More specifically, the simulator can generate a trapdoor based on the query sent by the probabilistic polynomial time adversary. When receiving the search token, the simulator uses the simulated index to produce the query result. The search token and the query result are indistinguishable to the adversary as the pseudo-random function and CPA-secure encryption algorithm. Thus, our scheme is adaptive IND-CKA security.

6 EXPERIMENTAL EVALUATION

We realized both fixed-length IBtree scheme and variable-length IBtree scheme in C language. We choose a real-world data set "America NSF Research Award Abstract set[23]". We evaluate both schemes on three aspects of IBtree memory usage, IBtree construction time and keyword searching time.

6.1 Data sets

We choose a real-world data set that is the America NSF Research Award Abstract set, which consists of 129K abstracts describing NSF awards from 1990-2003. We extract keywords from each abstract. The number of keywords in a document ranges from 15 to 454, and the average number of keywords in a document is 99.27. We use the real word data set to generate 1 data set, which contains 10 document sets with a size range of 500 to 5000. Our experiments are deployed on the generated data set.

6.2 Implementation Details

Our experiment platform is equipped with 3.20GHz Intel Core i7-8700 CPU and 16 GB RAM. The operating system running on the server is Ubuntu(version:18.03), and the kernel is Linux Ubuntu 5.3.0-51-generic. In our experiments, we use SHA256 as the hash function of HMAC. To minimize the false positive rate, we use three hash functions, and we set the root node size m according to $k = ln2(m/n)$ for different collections of data items.

6.3 Experiment Results

Experiment 1 (IBtree construction time) : We test IBtree construction time with data items ranging from 500 to 5000 and show the results in Figure 5. One can observe that the construction time consumed by a fixed-length IBtree scheme increases exponentially. Experimental results show that our variable-length scheme significantly reduces IBtree construction time. For example, for data sets of 5K, the construction time of fixed-length IBtree scheme and variable-length IBtree scheme is 29 hours and 10 minutes respectively.

Experiment 2 (IBtree memory usage) : The evaluated results of IBtree memory usage are shown in Figure 6. With data items ranging from 500 to 5K, the memory usage of fixed-length scheme and variable-length scheme grows from 102 MB to 10 GB, and from 1 MB to 14 MB, respectively. One can observe clearly that the memory occupied by the fixed-length IBtree increases exponentially. However, the memory usage of the variable length IBtree is proportional to the levels of the IBtree, as we described in section 4.4 Experimental evaluation shows that the more the number of terminal nodes we put, the larger the memory of our scheme saved compared with the fixed-length scheme.

Experiment 3 (keyword searching time) : Experimental results show that the keyword query processing time is in the millisecond scale. The time

consumed in keyword searching in a fixed-length IBtree scheme is an order of magnitude the same as our variable-length IBtree scheme as a hash operation is O(1) time. Figure 7 shows the single keyword searching time in two schemes.

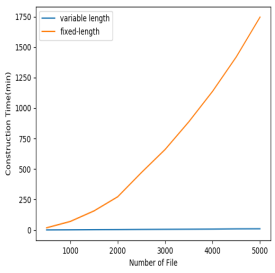

Fig. 5. Construction Time

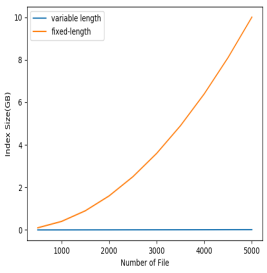

Fig. 6. Index Size

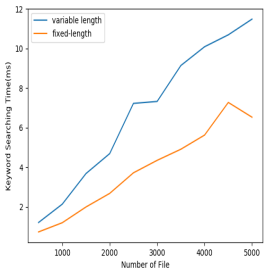

Fig. 7. Searching Time

7 Conclusion And Future Work

In this paper, we first propose a variable-length IBtree scheme to enable the keyword searching over the encrypted data in a popular client and cloud paradigm based on the data structure IBF. We extend the original trapdoor algorithm for a fixed length IBtree to get our trapdoor algorithm. In IBtree construction algorithm, much time is consumed by initializing IBFs. Therefore, our scheme theoretically cost less time and occupy less memory. Experimental results show our scheme has significantly reduced the IBtree construction time and memory usage. Keyword searching time of our scheme is an order of magnitude the same as the fixed-length IBtree scheme. Future work will be done to enable range queries through a variable-length IBtree over encrypted data.

Acknowledgement

This work is in part supported by the National Natural Science Foundation of China(No.61672421).

References

1. Li R , Liu A X . Adaptively Secure Conjunctive Query Processing over Encrypted Data for Cloud Computing[C] 2017 IEEE 33rd International Conference on Data Engineering (ICDE). IEEE, 2017.
2. S. K. Pasupuleti, S. Ramalingam, and R. Buyya, 'An efficient and secure privacy-preserving approach for outsourced data of resource constrained mobile devices in cloud computing', J. Netw. Comput. Appl., vol. 64, pp. 12–22, 2016.

3. Saiyu Qi, Yuanqing Zheng, Mo Li, Yunhao Liu, Jinli Qiu, "Scalable Industry Data Access Control in RFID-Enabled Supply Chain", IEEE/ACM Transactions on networking (ToN), 2016, 24 (6), 3551 – 3564
4. D. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In Proceedings of IEEE Symposium on Security and Privacy, pages 44–55. IEEE, May 2000.
5. D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. Public-key encryption with keyword search. In C. Cachin, editor, Proceedings of Eurocrypt 2004, LNCS. Springer-Verlag, May 2004.
6. Goh E J. Secure Indexes. International Association for Cryptologic Research Cryptology ePrint Archive, 2003: 216.
7. Curtmola R, Garay J, Kamara S, et al. Searchable symmetric encryption: Improved definitions and efficient constructions. Journal of Computer Security, 2011, 19(5):79-88.
8. Bösch C, Hartel P, Jonker W, et al. A survey of provably secure searchable encryption[J]. ACM Computing Surveys (CSUR), 2015, 47(2): 18.
9. Saiyu Qi, Youshui Lu, Yuanqing Zheng, Yumo Li, Xiaofeng Chen, "Cpds: Enabling Compressed and Private Data Sharing for Industrial IoT over Blockchain"IEEE Transactions on on Industrial Informatics, 28 May 2020DOI: 10.1109/TII.2020.2998166, PP: 1-12
10. S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In ACM CCS, 2012.
11. M. Bellare, A. Boldyreva, and A. ONeill. Deterministic and efficiently searchable encryption. In Proceedings of the CRYPTO, 2007.
12. G. Amanatidis, A. Boldyreva, and A. O'Neill. Provably-secure schemes for basic query support in outsourced databases. In Proc. Working Conference on Data and Applications Security (DBSEC), pages 14–30, 2007.
13. R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In ACM CCS, 2006.
14. L. Ballard, S. Kamara, and F. Monrose. Achieving efficient conjunctive keyword searches over encrypted data. In Information and Communications Security, pages 414–426. 2005.
15. E. Damiani, S. C. Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Balancing confidentiality and efficiency in untrusted relational dbmss. In CCS, pages 93–102, 2003.
16. P. Golle, J. Staddon, and B. Waters. Secure conjunctive keyword search over encrypted data. In Applied Cryptography and Network Security. 2004.
17. M. Kantarcioglu and C. Clifton. Security issues in querying encrypted data. In DBSec, 2005.
18. D. Park, K. Kim, and P. Lee. Public key encryption with conjunctive field keyword search. In Information Security Applications, pages 73–86. 2005.
19. Saiyu Qi*, Yuanqing Zheng, "Crypt-DAC: Cryptographically Enforced Dynamic Access Control in the Cloud"IEEE Transactions on Dependable and Secure Computing, 29 March 2019DOI: 10.1109/TDSC.2019.2908164, Page(s):1-13
20. B. Bloom. Space/time trade-offs in hash coding with allowable errors. Communications of the ACM, 13(7):422–426, Jul 1970.
21. Q. Jiang, Y. Qi, S. Qi, W. Zhao and Y. Lu. Pbsx: A practical private boolean search using Intel SGX. Information science. pages 172-194, 2020.
22. https://en.wikipedia.org/wiki/HMAC.
23. K. Bache and M. Lichman. UCI machine learning repository, http://archive.ics.uci.edu/ml, 2013.