Privacy-preserving and Efficient Multi-keyword Search Over Encrypted Data on Blockchain

Shan Jiang*, Jiannong Cao*, Julie A. McCann†, Yanni Yang*, Yang Liu‡, Xiaoqing Wang‡, Yuming Deng‡
*The Hong Kong Polytechnic University, Hong Kong, China
†Department of Computing, Imperial College London, London, UK
‡Alibaba Group Holding Limited, Hangzhou, China
{cssjiang, csjcao, csynyang}@comp.polyu.edu.hk, jamm@imperial.ac.uk, {lionel.ly, robin.wxq, yuming.dym}@alibaba-inc.com

***Abstract*—Recent research has demonstrated searchable blockchains that not only provide reliable search over encrypted distributed storage systems but ensure privacy is preserved. Yet, current solutions focus on single-keyword search over encrypted data on the blockchain. To extend such approaches to multi-keyword scenarios, they essentially perform a single-keyword search for multiple times and take the intersection of the results. However, such extensions suffer from privacy and efficiency issues. In particular, the service peers, which process the search requests, will be aware of the intermediate results, which include the data associated with each of the encrypted keywords. Moreover, these multiple traversals incur long delays in performing the search requests one after another with an extra cost in calculating the intersection of multiple sets. Finally, the service peers will charge the data owner a lot for writing the vast intermediate results to the smart contract. In this paper, we propose a bloom filter-enabled multi-keyword search protocol with enhanced efficiency as well as privacy preservation. In the protocol, a low-frequency keyword selected by a bloom filter will be used to filter the database when performing a multi-keyword search operation. Because the keyword is of low frequency, the majority of the data will be excluded from the result, which reduces the computational cost significantly. Moreover, we propose to use pseudorandom tags to facilitate completing each search operation in only one round. In this way, no intermediate results are generated, and the privacy is preserved. Finally, we implement the protocol in a local simulated blockchain network and conduct extensive experiments. The results indicate that our multi-keyword search protocol outperforms the traditional method with an average of 14.67% less time delay and 59.96% less financial cost.**

***Index Terms*—Blockchain, symmetric searchable encryption, smart contract**

I. Introduction

Nowadays, enterprises tend to store their data in data centers rather than locally due to the increasing demands for storage and computation resource [1]. Because there can be sensitive information, e.g., trade-union membership and health-related records, and even the data center and be malicious, the data is typically encrypted before outsourcing. The encryption, in turn, hinders data utilization, e.g., frequent search operations. Therefore, we need to bring out the technology of searchable symmetric encryption (SSE).

SSE is a technique enabling searching over encrypted data, in which the data owners encrypt not only the data but the search requests as well [2]. By this means, the data center knows nearly nothing about the data. However, the data center is assumed to be technically curious but honest in this field [3]. In practice, the data center has the potential to be malicious and deviate from the predefined protocol, e.g., to return only part of the result to save computational resources. Hence, reliability issues arise.

To deal with the reliability issue, the research community has proposed verifiable searchable encryption, in which the data center attaches some flag bits to the result [4]. Upon receiving the result from the data center, the data owners can decode the flag bits to verify the correctness of the result. However, it requires noticeable computational resources for the data owners to decode the flag bits [5]. It is preferable to outsource as many computational tasks as possible to the end devices, especially those with limited battery.

Recently, blockchain technology [6] [7] shows its potential to solve the reliability issue. In these schemes, the blockchain, a distributed ledger maintained by a trustless peer-to-peer network, serves as the data center. The encrypted data with indexes, which is stored in the blockchain and smart contract [8], is used to implement the functions of data storage and data search. Since all the operations are completed by all the nodes in the network, the correctness of the result can be guaranteed as long as the majority of the nodes are honest [9].

Existing blockchain-based searchable encryption schemes [10] [11] focus only on single-keyword search. They can be extended to multi-keyword scenarios by performing a single-keyword search for multiple times and taking the intersection of the results. However, such extensions suffer from privacy and efficiency issues. In particular, the intermediate results, i.e., the data associated with each individual keyword, will be exposed to the service peers. Such data leakage raises the privacy issue. Moreover, the service peers have to handle the single-keyword search requests one after another. Since some keywords can appear in the majority or even all the data, the computational cost to handle such keywords multiple times seems a substantial burden. The large amount of intermediate results also leads to a significant financial cost since they are written to the smart contract by the service peers. After the results for all the keywords are calculated, the service peers have to calculate the intersection of the results, which demands an extra computational cost.

In this paper, we design a privacy-preserving and efficient data management system with the functions of database setup, dynamic update, and multi-keyword search. The original database to be outsourced is defined as a set of identifier-keyword pairs. That is, there are several keywords associated with each of the identifiers. After setting the database up, the data owner can add or delete some identifier-keyword pairs dynamically. Meanwhile, the data owner can query all the identifiers that are associated with a set of keywords. The data center is a blockchain network composed of multiple service peers which rent out their computational resources to earn monetary rewards. Inside the blockchain, smart contracts are deployed to fulfill the data services. Because smart contracts are automated programs executed by all the service peers, the data services can be provided with reliability. Furthermore, SSE is employed in smart contracts to preserve privacy. Finally, we propose a bloom filter-enabled multi-keyword search protocol, which reduces the time delay and financial cost remarkably. Next, we discuss the challenges in designing our system and the proposed approaches to overcome them.

The first challenge is to set up an encrypted database without violating the cost limit rule in smart contracts [12]. That is, the service peers contribute their computational resources to maintain the state of the smart contracts. Such work is not free since each operation in the smart contract, e.g., adding two numbers and storage to the local disks, takes certain costs. To guarantee the validity of a smart contract, the cost for a single transaction is bounded, which is called the cost limit rule. In the setup phase, a large number of encrypted data is outsourced to the service peers. In particular, for each identifier-keyword pair, a reversed and encrypted keyword-identifier pair and a tag to support multi-keyword search will be generated in our algorithm. To prevent the setup operation from violating the cost limit rule, we devise a way to estimate the number of bytes that will be generated for each identifier-keyword pair and calculate the number of encrypted data that can be contained in a single transaction. Then, we slice the encrypted database based on the calculation to comply with the cost limit rule. Finally, the encrypted keyword-identifier pairs and the tags are randomly shuffled to prevent data leakage.

The second challenge lies in designing a time- and finance-efficient protocol for multi-keyword search. As discussed previously, the existing approaches are inefficient in terms of time delay and financial cost due to the large number of intermediate results. In this paper, we design our multi-keyword search protocol based on the insight that the appearance times of the majority of the keywords are low in the database. First, we generate a tag for each identifier-keyword pair in the setup phase. Meanwhile, we build a bloom filter to record all the high-frequency keywords in the database. In the multi-keyword search phase, we use the bloom filter to find a low-frequency keyword from the search request and use it to filter the database. Note that most of the keywords will be excluded from the result since the selected keyword is of low frequency. Finally, we use the tag of each identifier-keyword pair to check whether each candidate identifier meets the search request.

II. Privacy-preserving and Efficient Data Management via Blockchain

A. System Overview

There are two actors in the blockchain-based data management system, i.e., *service peer* and *data owner*. The service peers are individual nodes in the blockchain network who are renting out computational resources to earn monetary rewards. Data owners are the service requesters who want to outsource their data and later enjoy content update and search services.

There are four kinds of actions that can happen between the two actors, i.e., *setup*, *addition*, *deletion*, and *search*. The formal definition of each action is described as follows.

- *Setup*. The data owner outsources a database $W = \{(id_i, w_i) | i = 1, 2, \cdots, l\}$, a list of l identifier-keyword pairs to the service peers. Each $id_i \in \{0,1\}^\mu$ is a string of certain length while each $w_i \in \{0,1\}^*$ is a string of uncertain length. In the later sections, we will use l, m, and n to notate the number of identifier-keyword pairs, keywords, and identifiers respectively.
- *Addition*. The data owner adds a set of identifier-keyword pairs $\{id, W_a\}$ to the database W, where W_a is a set of keywords.
- *Deletion*. The data owner deletes a set of identifier-keyword pairs $\{id, W_d\}$ from the database W, where W_a is a set of keywords.
- *Search*. The data owner sends $W_s = \{w_1, w_2, \cdots, w_k\}$ to the service peers to find out all the identifiers ids such that there exists $w \in W_s$ and $(id, w) \in W$.

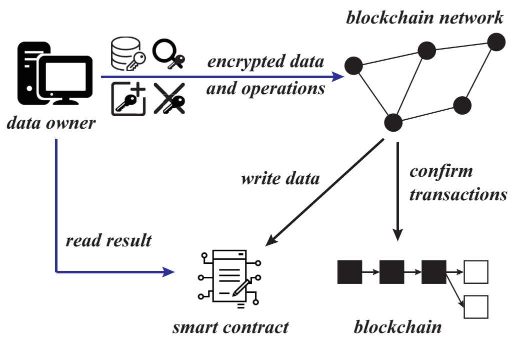

Fig. 1. System Overview

The flowchart of the actions is demonstrated in Fig. 1. When the data owners perform the operations of setup, addition, deletion, or search, they will send one or more transactions containing the encrypted data and operations to the service peers. The service peers process the transactions and pack the transactions into the blockchain. After the transaction is confirmed into the blockchain, the service peers will perform the operations which write data to the smart contract. Finally, the data owners can get the results according to the state of the smart contract.

We design privacy-preserving and reliable protocols to fulfill the four actions described in the following subsections[1]. Be-

[1]The protocols of addition and deletion are omitted due to page limit.

Algorithm 1 Initialization

```
Service Peers on Initializing the Smart Contract:
1: Allocate a dictionary $D_{ori}$
2: Allocate two sets $S_{del}$ and $S_{tag}$
3: Allocate two lists $Flag$ and $Result$
4: Set balance to be $B$, the money deposited by data owner
```

fore the illustration of the four actions, we first demonstrate the initialization of the smart contract as shown in Alg. 1. The smart contract stores five variables, a dictionary D_{ori}, two sets two sets S_{del} and S_{tag}, and two lists $Flag$ and $Result$ for future usage, in which D_{ori} is to store encrypted keyword-identifier pairs, S_{del} and $Flag$ are to support dynamic update of the database, S_{tag} is to support multi-keyword search, and $Result$ is used for storage of the search result. Finally, the balance of the smart contract is set to be B, which is the money deposited by the data owner. The operations cannot cost more than B in the future.

B. Database Setup

After initializing the smart contract, the data owner can set the database up as shown in Alg. 2. The data owner aims to store a set of identifier-keyword pairs. First, the data owner generates four secret keys K, K^+, K^-, and K^T. The four secret keys are all of size λ, which is an adjustable security parameter. Then, for each keyword w, derives two keys K_1 and K_2 are derived from a predefined pseudorandom function (PRF) [13] F and the secret key K. The key K_1 is to derive pseudorandom labels for the keywords while the key K_2 is to encrypt the identifiers. Using the keyword w, we can get all the identifiers that are associated with w, which is notated as a set W_w. Afterward, we iterate the identifiers id over W_w. For each id, the PRF F is applied to a counter c using the key K_1 to generate a pseudorandom label l. Meanwhile, we use K_2 to encrypt id using K_2 and get the encrypted identifier d. Then, we add the keyword-identifier pair (l, d) to a list L. To summarize, we reverse each identifier-keyword pair to an encrypted keyword-identifier pair and store the result in a list L. Besides this, we generate a unique tag for each identifier-keyword pair (id, w) by applying the PRF F over w and id sequentially using the secret key K^T. The tags are accumulated into the list L_{tag}. Note that the tags are used for the multi-keyword search.

The data owner has to send the database to the service peers after encryption. However, the encrypted database has to be sliced before sending due to the cost limit rule in the smart contract. Generally, each operation in smart contract takes a specific cost, and there is a cost limit for each transaction sent to the smart contract. As a result, the number of data that can be attached for each transaction is limited. In our protocol, the data generated for each identifier-keyword pair is designed to be bounded, i.e., a keyword-identifier pair and a tag. The size of the keyword-identifier pair and the tag will be fixed, e.g., 512 bits and 256 bits respectively, if the PRF F is fixed, e.g., HMAC-SHA256 [14].

Algorithm 2 Setup

```
Data Owner on Setting $W$ up:
1: $K \leftarrow \{0,1\}^\lambda$
2: $K^+ \leftarrow \{0,1\}^\lambda$
3: $K^- \leftarrow \{0,1\}^\lambda$
4: $K^T \leftarrow \{0,1\}^\lambda$
5: Allocate two lists $L$ and $L_{tag}$
6: Allocate two local dictionaries $D_{count}$ and $D_{key}$
7: $count \leftarrow 0$
8: for each keyword $w \in W$ do
9:     $K_1 || K_2 \leftarrow F(K, w)$
10:    $K^T_w \leftarrow F(K^T, w)$
11:    $c \leftarrow 0$
12:    for each $id \in W_w$ do
13:        $l \leftarrow F(K_1, c)$
14:        $d \leftarrow Enc(K_2, id)$
15:        $c \leftarrow c + 1$
16:        $tag \leftarrow F(K^T_w, id)$
17:        Append $(l, d)$ to $L$
18:        Append $tag$ to $L_{tag}$
19:        if $count \geq \delta$ then
20:            Shuffle $L$ and $L_{tag}$ randomly
21:            Send $(\textsc{Setup}, L, L_{tag})$ to the service peer
22:            $count \leftarrow 0$
23:            Empty $L$ and $L_{tag}$
24:        end if
25:    end for
26:    $cw \leftarrow Get(D_{key}, w)$
27:    if $cw = \perp$ then $cw \leftarrow 0$ end if
28:    $Set(D_{key}, w, cw + c)$
29: end for
30: Sort all the keywords $w \in W$ according to $Get(D_{key}, w)$
    in descending order to get a list of keywords $W_s$
31: Allocate a list $bf$ of $\alpha$ 0/1 bits initialized to be all 0
32: for each keyword $w$ in the first $\beta$ percentage of $W_s$ do
33:    $bf \leftarrow (H(w) \mid bf)$
34: end for
35: Send $(\textsc{Setup}, L, L_{tag})$ to the service peer
36: Store $K$, $K^+$, $K^-$, $K^T$, $bf$, and $D_{count}$ locally
Service Peers on Receiving $(\textsc{Setup}, L, L_{tag})$:
37: Add all the elements $(l_i, d_i)$ in $L$ to the dictionary $D_{ori}$
    with $l$ as the key and $d$ as the value
38: Add all the elements in $L_{tag}$ to the set $S_{tag}$
```

As a result, we can calculate the maximum number δ of identifier-keyword pairs that can be handled in one transaction. Assume that the PRF F digests message into δ_f bits and the number of bits that can be stored in a single transaction to be δ_t. Then, the value of δ should be $\lfloor \delta_t / (3\delta_f) \rfloor$. In this paper, at most 10KB can be stored in a transaction and HMAC-SHA256 is used as the PRF. As a result, δ_t equals $80,000$, δ_f equals 256, and δ is calculated to be 104. When the counter reaches δ, we shuffle the lists L and L_{tag}, and send a transaction of setup containing them to the service peer. The reason for shuffling

L and L_{tag} is to prevent the service peers from inferring any information related to the data. For example, L and L_{tag} are in the same order with regard to each identifier-keyword pair. After sending each setup transaction, the counter will be reset to be 0 and that lists L and L_{tag} will be emptied. Note that there are additional operations related to the bloom filter bf, which will be explained in the later subsections.

From the perspective of the service peers, they are receiving several transactions of setup together with two lists L and L_{tag}. For each transaction, they enumerate the elements (l_i, d_i) in the list L and add it into the dictionary D_{ori} with l_i as the key and d_i as the vale. Afterward, they store all the elements in L_{tag} to the set S_{tag} in the smart contract. We can see that the data is stored in the smart contract with unordered data structures, i.e., dictionary and set. Therefore, the setup protocol is immune to the order of the transactions received, which is a notable feature.

C. Multi-keyword Search

In the above subsections, we demonstrate the protocols of setup, addition, and deletion, which enables dynamic, reliable, and privacy-preserving storage and update of the data. In this subsection, we demonstrate the protocol for multi-keyword search upon the encrypted database.

To begin with our approach, we introduce the way to build the bloom filter which includes the keywords that frequently appear in the database. In Alg. 2, we create a dictionary D_{key} to count the appearance time of each keyword, where the appearance time of a keyword w is defined to be:

$$F(w, W) = |\{(id_i, w)|(id_i, w) \in W\}|$$

A keyword is defined to be high-frequency in a database W if it is in the first β percentage when sorting $\{w|w \in W\}$ in non-increasing order according to the appearance times. A keyword is defined to be low-frequency in W if it is not high-frequency in W.

For each high-frequency keyword w in W, we use a hash function H to hash w into an α-bit $0/1$ string $H(w)$ and apply bitwise OR operation to bf using $H(w)$. In this way, bf is a bloom filter containing all the keywords of high frequency. Note that α and β are parameters that should be fine-tuned to make the bloom filter efficient. A large value of α increases the storage burden for the data owner while decreases the false positive rate when judging whether a keyword of high frequency. On the other hand, a large value of β increases the false positive rate while reduces the cost if true positive. In this paper, we set α and β to be $8,000$ and 10% respectively, which is enough to handle a database of up to 9.1M identifier-keyword pairs.

After setting the bloom filter up, the data owner can use it to find an arbitrary low-frequency keyword among the k keywords in the search request. If the hash value of a keyword does not equal to the result of applying bitwise AND operation to itself with bf, then the keyword must be low-frequency. If such a low-frequency keyword is found, we swap it with the first keyword in the multi-keyword search request;

Algorithm 3 Search

```
Data Owner on Searching $(w_1, w_2, \cdots, w_k)$ upon $W$:
1: for $k \leftarrow 1$ to $k$ do
2:     $h \leftarrow H(w_i)$
3:     if $(h \,\&\, bf) \neq h$ then
4:         Swap $w_1$ and $w_i$
5:         Break
6:     end if
7: end for
8: $K_1||K_2 \leftarrow F(K, w_1)$
9: $K_1^+||K_2^+ \leftarrow F(K^+, w_1)$
10: $K_1^- \leftarrow F(K^-, w_1)$
11: for $i \leftarrow 2$ to $k$ do $K_i^T \leftarrow F(K^T, w_i)$ end for
12: Send $(\text{SEARCH}, K_1, K_2, K_1^+, K_2^+, K_1^-, K_2^T, \cdots, K_k^T)$ to
    the service peer
Service Peers on Receiving $(\text{SEARCH}, K_1, K_2, K_1^+, K_2^+, K_1^-, K_2^T, \cdots, K_k^T)$:
13: $res \leftarrow \emptyset$
14: for $c \leftarrow 0$ to $\infty$ do
15:     $l \leftarrow F(K_1, c)$
16:     $d \leftarrow Get(D_{ori}, l)$
17:     if $d = \perp$ then Break end if
18:     $res \leftarrow res \cup \{Dec(K_2, d)\}$
19: end for
20: for $c \leftarrow 0$ to $\infty$ do
21:     $l \leftarrow F(K_1^+, c)$
22:     $d \leftarrow Get(D_{ori}, l)$
23:     if $d = \perp$ then Break end if
24:     $res \leftarrow res \cup \{Dec(K_2^+, d)\}$
25: end for
26: for each $id \in res$ do
27:     $delid \leftarrow F(K_1^-, id)$
28:     if $delid \in S_{del}$ then $res \leftarrow res \setminus \{id\}$
29:     else for $i \leftarrow 2$ to $k$ do
30:         if $F(K_i^T, id) \notin S_{tag}$ then
31:             $res \leftarrow res \setminus \{id\}$
32:             Break
33:         end if
34:     end for end if
35: end for
36: $Result \leftarrow res$
```

otherwise, there is no low-frequency keyword among the k keywords, and the first keyword will remain high-frequency. Note that we will not make the first keyword to be high-frequency if it is low-frequency before the operation since no true negative judgment happens a bloom filter.

Now, it comes to the phase of generating encrypted search request by the data owner. The data owner will take the secret keys K, K^+, and K^- to generate three pseudorandom labels K_1, K_1^+, and K_1^- and two symmetric keys K_2 and K_2^+ for the first keyword w_1. Meanwhile, a tag will be generated for each keyword k_i, where i ranges from 2 to k, using the secret key K^T. In this way, an encrypted search request containing three pseudorandom labels, two symmetric keys, and $k-1$ tags will be generated and sent to the service peers.

On receiving a search request from the data owner, the service peers will start with the first keyword and get a set of candidate identifiers. In particular, they traverse D_{ori} in the smart contract using the pseudorandom K_1 and K_1^+ in sequence. Then, the service peers accumulate all the counters that exist in the key field of D_{ori} after encryption using K_1 or K_1^+. Afterward, the service peers add the identifiers after decryption corresponding to each of the accumulated counters using the corresponding symmetric key. Finally, we deal with the deletion set and the other $k-1$ keywords at the same time. For each of the identifiers id after decryption, we check whether the encrypted result of id using the pseudorandom deletion label K_1^- is in the set of S_{del} and whether any of the $k-1$ tags after applying PRF F to id is not in the tag list S_{tag}. If any of the two conditions hold, id will be excluded from the result.

Finally, we analyze the time delay and financial cost for the traditional method and our method. In the traditional method, it takes $O(l)$ and generates $O(n)$ identifiers for each single-keyword search request. Then, it takes an extra $O(k \cdot n \cdot \log n)$ time to calculate the intersection of k sets, each of which is of size $O(n)$. Since the writing operations dominate the financial cost for a smart contract [15], we approximate the financial cost as the number of identifiers in the intermediate and final results. Hence, the time delay and financial overhead are $O(k \cdot l + k \cdot n \cdot \log n)$ and $O(k \cdot n)$ respectively. In terms of our method, it takes $O(l)$ to use the first keyword to filter the database. Then, $\beta \cdot n$ identifiers will be generated and verified through $k-1$ tags, which takes $O(k \cdot \beta \cdot n)$ time. Therefore, the time overhead for our approach is $O(l + k \cdot \beta \cdot n)$. In the experiments, we figure that β can be as small as 10%, which reduce the time complexity remarkably. The financial cost overhead is $O(n)$ since only the final result will be written to the smart contract.

III. Experimental Result

We implement the operations of database setup, dynamic update, and multi-keyword search in python 2.7 with the PRF implemented by HMAC-SHA256 in the PyCryptodome package and the bloom filter implemented by the pybloom package [16]. We run both the data owner and the service peers on laptops running Ubuntu 16.04.5 with 16GB RAM and two Intel i7-6500U cores. The service peers form a local simulated blockchain network, accept the request from the data owner, and run the smart contract. To focus on the performance of our protocol, we set the block generation time to be 0, which means the influence of the complex network topology is not taken into consideration.

After setting up the experimental environment, we conduct extensive experiments on the Eron email dataset [17], which consists of 517K emails. The original database is generated from the dataset as follows. Each email is treated as a new identifier id, and each word after lowercasing in the email is treated as a keyword associated with id. By this means, we get a database consisting of 517K identifiers, 622K distinct keywords, and 9.1M identifier-keyword pairs.

A. Single-keyword Search

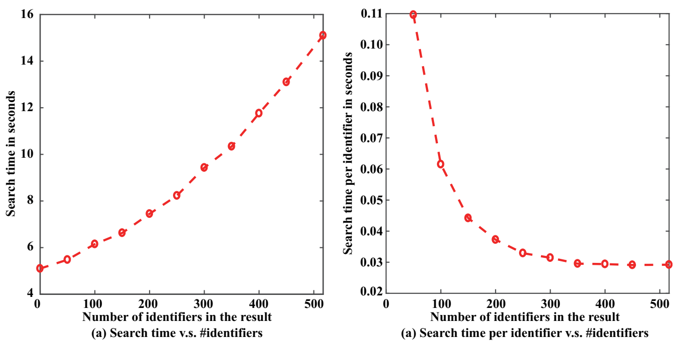

Fig. 2. Single-keyword search

In the single-keyword search operation, the time consumption at the data owner side can be neglected since only several operations of symmetric encryption are involved. At the service peer side, it needs to traverse the dictionary D_{ori} twice and write the data to the local state. We conduct experiments on searching keywords with various appearance times. The result is shown in Fig. 2. We can see that 5.12s is needed when there is no matched identifier, which is the time to traverse the dictionary D_{ori}. Moreover, it takes 15.10s when all the identifiers are associated with the keyword. The search time per identifier decreases as the number of identifiers in the result increases. The reason is that a large number of identifiers can average the time to traverse the dictionary.

B. Multi-keyword Search

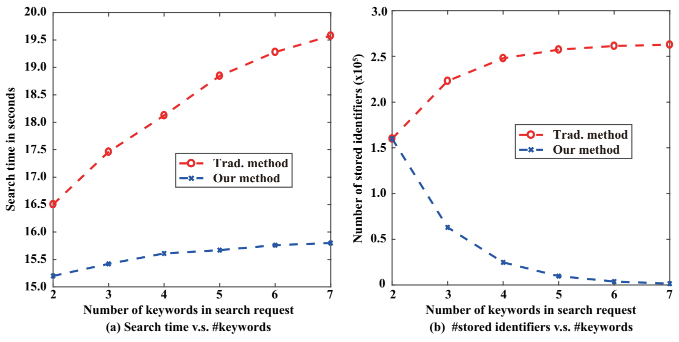

Fig. 3. Time evaluation for multi-keyword search

We conduct experiments for multi-keyword search over traditional method and our method for the number of keywords ranging from 2 to 7. The traditional method, or the intersection method, is to apply single-keyword search multiple times and take the intersection of the results as the final result. In our method, we set β to be 10 since no more than 0.05% keywords appears in at least 10% identifiers after analysis. We run the experiments for 50 times and the comparison results in terms of time and financial overhead are shown in Fig. 3. Note that extreme cases, e.g., all the keywords are of high frequencies, are included in the experiments because the keywords are randomly generated.

In terms of time overhead, the intersection method is significantly affected by the number of keywords since the intersections of more sets should be calculated when the number of keywords increases. For our method, the time to execute a multi-keyword search does not vary too much as the number of keywords increases. The reason is that there will be few candidate identifiers after filtering the database using the first keyword. Moreover, only one operation of tag comparison will be added to the computational burden in case of one more keyword. On average, our method outperforms the intersection method by 14.67% in terms of time efficiency.

In terms of financial overhead, the major financial cost for search operation lies in data storage because data storage dominates the cost compared to other operations. Therefore, we treat the number of identifiers that are written to the state of the smart contract as the financial cost. For the intersection method, the number of stored identifiers increases about 60% when the number of keywords increases from 2 to 5 and remains nearly unchanged for 5 and more keywords. The reason is that there will be few identifiers with the same 5 or more keywords. The financial cost of our method decreases when the number of keywords increases since we call the smart contract and write the result only once. Besides, the number of eligible identifiers decreases when the number of filtering keywords increase. On average, our method outperforms the intersection method by 59.96% concerning financial cost.

IV. Related Work

Searchable symmetric encryption is a technique to enable privacy-preserving and secure search over encrypted data between client and server [2]. The research community has been devoted this area for enabling dynamic operations [18], supporting boolean queries [3], and extending to graph database [19]. However, none of these research works considers dishonest servers. Verifiable searchable encryption has the same target as SSE while considering the dishonest servers. The researchers enable the client-side verification by introducing verifiable hash table [5]. Nevertheless, the client is assumed to be honest in their works. Moreover, the client takes non-negligible efforts to verify the results from the servers. In recent years, the research community introduces blockchain to searchable encryption to solve the dishonesty issues of both the client and server [9] [10] [11] [20]. However, they focus on the financial fairness between the miners in blockchain and the clients and suffer from privacy and efficiency issues when extended to multi-keyword search.

V. Conclusion

In this paper, we propose a blockchain-based data management system with functions of privacy-preserving and efficient database setup, dynamic update and multi-keyword search. The technique to divide the encrypted database into several pieces in the protocols is general for other blockchain applications. The key contribution lies in enabling multi-keyword search over encrypted database on blockchain and improving its efficiency in terms of time delay and financial cost. To do so, we propose to use a bloom filter to find out a low-frequency keyword and filter the encrypted database using the keyword, which significantly narrows down the searching space. The future work can be tuning the parameters in the bloom filter to further enhance efficiency.

VI. Acknowledgment

This work is supported by Alibaba Innovative Research (AIR) Program with project number H-ZG6N and Hong Kong RGC Research Impact Fund (RIF) with project number R5034-18.

References

[1] B. Hayes, "Cloud computing," *Communications of the ACM*, vol. 51, no. 7, pp. 9–11, 2008.
[2] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *IEEE S&P*, 2000, pp. 44–55.
[3] S. Kamara and T. Moataz, "Boolean searchable symmetric encryption with worst-case sub-linear complexity," in *Springer EUROCRYPT*, 2017, pp. 94–124.
[4] X. Liu, G. Yang, Y. Mu, and R. Deng, "Multi-user verifiable searchable symmetric encryption for cloud storage," *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2018.
[5] R. Bost, "$\sum o\varphi o\varsigma$: Forward secure searchable encryption," in *ACM CCS*, 2016, pp. 1143–1154.
[6] S. Nakamoto *et al.*, "Bitcoin: A peer-to-peer electronic cash system," *Bitcoin Project White Paper*, pp. 1–9, 2008.
[7] S. Jiang, J. Cao, H. Wu, Y. Yang, M. Ma, and J. He, "Blochie: a blockchain-based platform for healthcare information exchange," in *IEEE SMARTCOMP*, 2018, pp. 49–56.
[8] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, pp. 1–32, 2014.
[9] S. Hu, C. Cai, Q. Wang, C. Wang, X. Luo, and K. Ren, "Searching an encrypted cloud meets blockchain: A decentralized, reliable and fair realization," in *IEEE INFOCOM*, 2018, pp. 792–800.
[10] Y. Zhang, R. Deng, X. Liu, and D. Zheng, "Outsourcing service fair payment based on blockchain and its applications in cloud computing," *IEEE Transactions on Services Computing (TSC)*, 2018.
[11] C. Cai, J. Weng, X. Yuan, and C. Wang, "Enabling reliable keyword search in encrypted decentralized storage with fairness," *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2018.
[12] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on ethereum smart contracts (sok)," in *Springer POST*, 2017, pp. 164–186.
[13] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *Journal of the ACM (JACM)*, vol. 33, no. 4, pp. 792–807, 1986.
[14] H. Krawczyk, M. Bellare, and R. Canetti, "Hmac: Keyed-hashing for message authentication," *RFC*, vol. 2104, pp. 1–11, 1997.
[15] X. Li, P. Jiang, T. Chen, X. Luo, and Q. Wen, "A survey on the security of blockchain systems," *Future Generation Computer Systems (FGCS)*, 2017.
[16] P. S. Almeida, C. Baquero, N. Preguiça, and D. Hutchison, "Scalable bloom filters," *Information Processing Letters*, vol. 101, no. 6, pp. 255–261, 2007.
[17] B. Klimt and Y. Yang, "The enron corpus: A new dataset for email classification research," in *Springer ECML*, 2004, pp. 217–226.
[18] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *ACM CCS*, 2012, pp. 965–976.
[19] X. Meng, S. Kamara, K. Nissim, and G. Kollios, "Grecs: Graph encryption for approximate shortest distance queries," in *ACM CCS*, 2015, pp. 504–517.
[20] L. Chen, W.-K. Lee, C.-C. Chang, K.-K. R. Choo, and N. Zhang, "Blockchain based searchable encryption for electronic health record sharing," *Future Generation Computer Systems (FGCS)*, 2019.