EURASIP Journal on Wireless Communications and Networking

RESEARCH Open Access

A blockchain-assisted framework for secure and reliable data sharing in distributed systems

Yu Guo[1], Shenling Wang[1*] and Jianhui Huang[2]

*Correspondence: slwang@bnu.edu.cn
[1] School of Artificial Intelligence, Beijing Normal University, Beijing, China
Full list of author information is available at the end of the article

Abstract

The explosive growth of big data is pushing forward the paradigm of cloud-based data store today. Among other, distributed storage systems are widely adopted due to their superior performance and continuous availability. However, due to the potentially wide attacking surfaces of the public cloud, outsourcing data store inevitably raises new concerns on user privacy exposure and unauthorized data access. Besides, directly introducing a centralized third-party authority for query authorization management does not work because it still can be compromised. In this paper, we propose a blockchain-assisted framework that can support trustworthy data sharing services. In particular, data owners allow to outsource their sensitive data to distributed systems in encrypted form. By leveraging smart contracts of blockchain, a data owner can distribute secret keys for authorized users without extra round interaction to generate the permitted search tokens. Meanwhile, such blockchain-assisted framework naturally solves the trust issues of query authorization. Besides, we devise a secure local index framework to support encrypted keyword search with forward privacy and mitigate blockchain overhead. To validate our design, we implement the prototype and deploy it at Amazon Cloud. Extensive experiments demonstrate the security, efficiency, and effectiveness of the blockchain-assisted design.

Keywords: Dynamic searchable encryption, Distributed data storage, Forward security, Smart contract, Blockchain

1 Introduction

To accommodate the explosive growth of big data, distributed data stores have become the main solution for many public cloud services. Toward such a trend, many emerging database systems such as Redis [1], RAMCloud [2], and DynamoDB [3] are being increasingly deployed at the public cloud, due to their strength of performance, scalability, and fault tolerance.

Despite being promising, outsourcing data processing in the public cloud would also raise new challenges for its privacy and support for flexible types of data operations. Public cloud might be vulnerable to security breaches, and privacy concerns are becoming more serious with recent incidents of massive data disclosures [4]. Although standard encryption technology could guarantee data privacy, it would explicitly invalidate textual search functions over encrypted data. Accordingly, in the literature, there have been recent endeavors on investigating how to enhance data privacy while preserving data operation privileges, such as searchable symmetric encryption (SSE) for encrypted keyword search [5–7]. By leveraging these cryptographic primitives, a line of work on encrypted database systems have been proposed and implemented [8–10]. Yet, most of them focus on centralized database servers, which are not specifically designed for distributed systems.

In the literature, only a few recent works [11–14] have started to study secure data retrieval in distributed systems. These works can be viewed as valuable start points in the design space, but there are still some challenging issues to be solved. First, most of exiting solutions only focus on a single-user setting that allows encrypted queries from a single user holding the secret key. However, in practical database systems, there will be multi-users accessing the database. To this end, existing solutions either introduce a third-party authority for query authorization management or leveraging the data owner to generate query token for each authorized user [15]. However, the former will be suffered from the problem of untrusted authority (e.g., Denial-of-service attack), while the latter is difficult to realize in reality because the data owner must be online consistently. Second, update operation needs to be treated carefully for encrypted database systems. Simply adding new records or removing idle records will allow the cloud server to learn the associations between newly updated entries and those search results. Recent leakage-abuse attacks [16] have shown that this additional leakage can be exploited to learn sensitive information about the query requests or data contents. Thus, it is desirable to devise a new system framework for secure and reliable data sharing.

In this work, we design and implement a blockchain-assisted framework for secure and reliable data sharing in distributed database systems. Getting rid of a central third-party authority, outsourcing query authorization to the smart contract of blockchain yields a reliable data sharing service, and no longer need the data owner to be always online for token generation. Besides, we devise a forward-privacy index construction to handle the problem of update operation in encrypted data storage. Specifically, we store the encrypted state of each keyword at the smart contract and ask the data owner to use fresh random masks generated from the latest state to encrypt newly added entries. Thus, the cloud server cannot learn whether a newly added entry contains a keyword used in a previous search token. Besides, our proposed index construction can support efficient data deletion. We implement the system prototype and provide a formal security analysis. Extensive experiments demonstrate the practicality of the design. In summary, our contributions are listed as follows:

- We propose a blockchain-assisted framework that enables secure and reliable data sharing in a multi-user setting. A data owner can efficiently authorize multiple users to access the encrypted data store, or revoking users' authorization.
- Our proposed index construction can protect against persistent adversaries and guarantees forward security. Specifically, it can support secure delete operation efficiently, unlike the other forward-privacy schemes that only support insertion-only operations.
- We implement our system prototype and deploy it on Amazon Cloud. The experimental results show that it preserves linear scalability of distributed data stores with

respect to their performance. And the performance is significantly improved when comparing to previous works.

2 Overview

2.1 System Architecture

Figure 1 presents our system architecture, containing four entities: the data owner, data users, the blockchain, and a number of distributed data stores. Our design serves the client who wishes to outsource their sensitive data to the public cloud and offers secure query services to authorized users. Specifically, the data owner performs data encryption, encrypted index construction, data update operation, as well as users' authorization. The owner holds the master key which is used to derive different private keys for the function above. The rule of the blockchain is to trace the state information of each keyword and maintain authorized users' information. Meanwhile, it maintains a small-sized consistent hashing ring to route the encrypted documents across all distributed data stores. A data node in the cloud handles query requests from authorized users. It processes search token over its secure local indexes and returns the matched results to the user.

From a high-level point of view, our system includes the following procedure to provide secure and efficient data sharing services. Initially, the data owner builds the encrypted index and stores it with corresponding encrypted documents to the distributed database. Meanwhile, the data owner needs to upload the encrypted state to the blockchain for query token generation. After user registration, the authorized user can obtain his query key from the data owner via the blockchain. To enable the secure query with forward security, the user first generates the query token by using his query key and sends it to the blockchain. After that, the blockchain computes the latest trapdoor based on the received token and the keyword state. Finally, the data node processes the

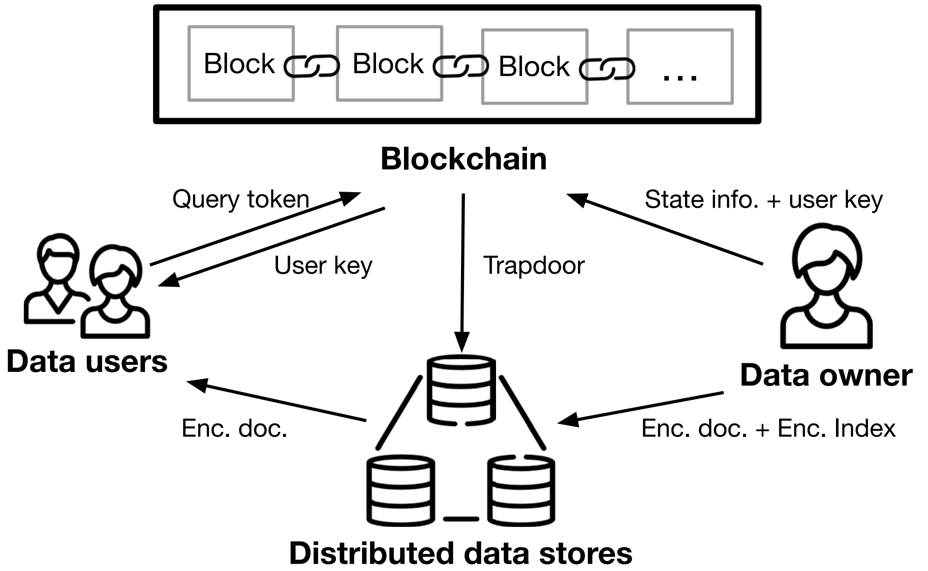


Fig. 1 System architecture

token over the encrypted local index and returns the documents that contain the exactly matched keywords.

In our architecture, we resort to the smart contract of blockchain to achieve secure and efficient data sharing. It can not only assist with data insertion with forward privacy guarantees, but also enforce access control policy for data sharing. Besides, after one time interaction, authorized users can always generate the search token by using smart contract without the interaction with the data owner any more. Thus, there is no need for the data owner to stay online.

2.2 Threat model

Consistent with prior studies on search over encrypted data [6, 10], our security goal is to protect data owners' database. Specifically, we consider the threats from semi-honest (aka honest-but-curious) adversaries on the data servers, who faithfully follows the prescribed protocols but may intend to learn the information of data and queries by observing the query token, accessed index entries, and encrypted results. The attackers cannot access the private keys stored at the users, but they could dump the entire contents of the datasets from the server. We consider that data users are secure and trusted. It will not expose the keys to cloud servers, and private keys are securely stored on the client side.

2.3 Design goals

We note that the blockchain actually is a trusted platform for correctness and availability, but does not provide protection of users' privacy. Therefore, our design goal is to provide the strongest possible protection on the data privacy while simultaneously maintaining the service efficiency and quality. Our design goals are listed as follows:

Data confidentiality It should ensure strong protection of data owner's files, state information, and users' query content during the service flow. It is the most basic security feature in general searchable encryption schemes.

Forward security Forward security is a strong security requirement for dynamic SSE schemes, which requires newly inserted entries are unlinkable to previous query results.

Multi-client support After user registration, authorized users can execute permitted query processing by using separate keys without extra round interaction with the data owner.

Query efficiency The complexity of the proposed secure query protocol should be sublinear, and the query latency and bandwidth should be bounded.

3 Background knowledge

3.1 Cryptographic primitives

Searchable symmetric encryption A searchable symmetric encryption scheme is a set of three polynomial time algorithms $\Pi = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$: The key generation algorithm KGen takes a security parameter k as input and outputs a secret key K; the encryption algorithm Enc takes a key K and a value $v \in \{0, 1\}^*$ as inputs and outputs a ciphertext $v^* \in \{0, 1\}^*$; The decryption algorithm Dec takes a key K and a ciphertext v^* as inputs and returns v.

Pseudo-random function (PRF) Define a pseudo-random function $F : \mathcal{K} \times X \rightarrow R$, if for all probabilistic polynomial-time distinguishers Y, $|Pr[Y^{F(k,\cdot)} = 1 | k \leftarrow \mathcal{K}] - Pr[Y^{g} = 1 | g \leftarrow \{\mathsf{Func} : X \rightarrow R\}]| < negl(k)$, where $negl(k)$ is a negligible function in k.

Bilinear pairings Let G_1, G_2 and G_T be three bilinear groups of prime order p, with generators $g_1 \in G_1$ and $g_2 \in G_2$, respectively. A bilinear pairing is a map $\hat{e} : G_1 \times G_2 \rightarrow G_T$ with the three properties: 1) Bilinearity: for all $u \in G_1$, $v \in G_2$ and $a, b \in Z_p$, $\hat{e}(u^a, v^b) = \hat{e}(u, v)^{ab}$. 2) Non-degeneracy: $\hat{e}(g_1, g_2) \neq 1$. 3) Computability: $\hat{e}(u, v)$ can be efficiently computed for any $u \in G_1$, $v \in G_2$.

3.2 Blockchain and KV store

Blockchain In general, the blockchain [17–27] can be treated as a distributed data store that records all the transactions that have occurred in the peer-to-peer network. All participants in the network hold the same copy of the data record, and there is no central authority or single node can control the entire network. Compared to the original Bitcoin system [28], a new blockchain framework [29] is proposed, which allows users to create, deploy, and run smart contracts [29] (predefined computer program) on the blockchain. Once the contract is deployed, it can be automatically executed according to the agreed logic of smart contracts and verified to demonstrate the effectiveness of the contract operation [30]. In this work, we use the blockchain as a trusted platform for key sharing, permission grant, and trapdoor generation.

Encrypted key-value store We follow the construction of encrypted key-value stores proposed in [11], where the document can be stored as an encrypted key-value pair. Assume that the data owner has a document f to be outsourced in the data node, and it has a unique document identifier id. Then, the document f is encrypted with the above symmetric encryption scheme Enc, and its identifier id is protected with PRF (i.e., Pseudo-random function) P. Specifically, each key-value pair is defined as: $\langle k, v \rangle = \langle P(k_{id}, id), Enc(k_f, f) \rangle$, where k_{id} and k_f are the private keys (Table 1).

4 The proposed system

In this section, we present our blockchain-assisted design to support multi-client queries in an encrypted KV store with forward privacy. Encrypted search protocol, on-chain query authorization and secure update protocol are also presented in this section.

Table 1 Glossary

Acronym	Definition
$G1, G2$	Pseudo-random functions
h	Hash function
$w, \{id_1, .., id_n\}$	Keyword-file IDs pairs
α, β	Encrypted Keyword-file IDs pairs
$\{I_1, .., I_m\}$	Encrypted indexes
$\hat{e}(u, v)$	Bilinear pairing operation
S	On-chain keyword state table
U	On-chain authority table

Algorithm 1 : Build keyword-match index

```
Input: Private key $k_o$; secure PRFs $\{G1, G2\}$; hash function $h : \{0,1\}^* \rightarrow g_1$; keyword $w$ from
    documents IDs $\{id_1, .., id_n\}$.
Output: Encrypted index $\{I_1, .., I_m\}$.
Derive $\{k_w, k_\alpha\}$ from $k_o$;
Initialize a hash table $S$ to maintain keyword state;
Generate $\gamma \in g_2$;
$t_w \leftarrow \hat{e}(h(w), \gamma)^{k_w}$;
$j \leftarrow route(t_w)$; // $j \in \{1, m\}$ is node ID
$\alpha_0 \leftarrow \perp$; // all matched entries have been found
$\alpha_n \leftarrow G1_{k_\alpha}(t_w, n), k_\beta^n \leftarrow KGen(n)$;
for $i \in \{n, 1\}$ do
    $\alpha_{i-1} \leftarrow G1_{k_\alpha}(t_w, i-1), k_\beta^{i-1} \leftarrow KGen(i-1)$;
    $\beta_i \leftarrow G2_{k_\beta^i}(\alpha_{i-1}||k_\beta^{i-1}, id_i)$;
    $I_j.put(\alpha_i, \beta_i)$;
end for
$S.put(t_w, n, k_\beta^n)$; // $S$ is stored on smart contract
deploy $\{k_\alpha$, S$\}$ to smart contract;
```

4.1 Encrypted Index design

The detailed algorithm to index file IDs $\{id_1, .., id_n\}$ for a given keyword w is shown in Algorithm 1. This procedure is executed at the data owner side. First, the data owner generates a query token t_w via computing $\hat{e}(h(w), \gamma)^{k_w}$, where $h(w) \in g_1$ and $\gamma \in g_2$. Then, the owner finds the target node j for t_w based on the position on the consistent hashing ring. After that, for the ith entry id_i, it generates encrypted index pairs via secure PRF, i.e., $\alpha_i = G1_{k_\alpha}(t_w, i)$ and $\beta_i = G2_{k_\beta^i}(\alpha_{i-1}||k_\beta^{i-1}, id_i)$, where α_{i-1} is the address of previous index entry and k_β^{i-1} is the corresponding encryption key. Finally, the owner sends the state table S and index pairs $\langle\alpha, \beta\rangle$ to the smart contract and the data node, respectively.

The encrypted index above holds the security notion of SSE. The index size is known to the data node. Without querying, no other information about the underlying content is learned. This property is achieved by embedding the unique keyword state into the index entry. Thus, the associations between keywords in different document IDs can be well protected.

Algorithm 2 : Multi-client authorization

```
Input: Private key $k_w$; $\gamma \in g_2$.
Output: Authorize/Revoke user $u$
$\underline{Register.User}$
    Initialize a hash table $U$ to maintain user's key;
    Generate a query key $k_u^1$;
    $k_u^2 \leftarrow k_w / k_u^1$;
    $U.put(u, \gamma^{k_u^2})$; // $U$ is stored on smart contract
    Send $k_u^1$ to authorized user $u$;
    Deploy $U$ to smart contract;
$\underline{Revoke.User}$
    $U.delete(u, \gamma^{k_u^2})$;
```

Algorithm 3 : Secure keyword query protocol

```
Input: Private keys $k_\alpha$; user $u$'s query key $k_u^1$; hash function $h$; hash tables $\{U, S\}$; query keyword $w$.
Output: Matched results $\{id_1, .., id_n\}$.
$\underline{User.Token}$
  $t_w^u \leftarrow h(w)^{k_u^1}$;
  Send $\{u, t_w^u\}$ to smart contract;
$\underline{Blockchain.Token}$
  if $U.find(u) \neq null$ then
      $\gamma^{k_u^2} \leftarrow U.find(u)$;
      $t_w \leftarrow \hat{e}(t_w^u, \gamma^{k_u^2})$;
      $n, k_\beta^n \leftarrow S.find(t_w)$;
      $\alpha_n \leftarrow G1_{k_\alpha}(t_w, n)$;
      $j \leftarrow route(t_w)$;
      Send $\{\alpha_n,\ k_\beta^n\}$ to $node_j$;
  else
      Return "access denied";
  end if
$\underline{Node.Query}$
  for $i \in \{n, 1\}$ do
      $\beta_i \leftarrow Node_j.find(\alpha_i)$;
      $\alpha_{i-1}, k_\beta^{i-1}, id_i \leftarrow Dec_{k_\beta^i}(\beta_i)$;
  end for
  Fetch encrypted documents via $\{id_n, .., id_1\}$;
```

4.2 Multi-client authorization

To register a new user u, the data owner first generates a pair of query keys $\{k_u^1, k_u^2\}$, where $k_u^1 \times k_u^2 = k_w$. Then, the query key k_u^1 and the authorized user table U are sent to the user and the smart contract, respectively, as shown in Algorithm 2. Correspondingly, it also presents how to revoke an authorized user. Given the user id u, the smart contract just needs to remove the entry $(u, \gamma^{k_u^2})$ from U. After that, the user u can no longer query the encrypted data because the smart contract cannot generate the query token. By introducing the access permission table U at the smart contract, our design enforces the access control without the extensive interaction between owner and authorized users.

4.3 Secure keyword-match protocol

Based on the index construction, we present secure query protocol in details in Algorithm 3. Given a query keyword w, the authorized user u wants to find all document IDs containing the keyword. First, the data user generates the keyword token t_w^u by using its query key k_u^1, where $t_w^u = h(w)^{k_u^1}$. After receiving the user id u and token t_w^u, the smart contract first checks its access permission at table U and then computes the query token t_w via bilinear pairing, i.e., $\hat{e}(h(w)^{k_u^1}, \gamma^{k_u^2}) = \hat{e}(h(w), \gamma)^{k_w} = t_w$. After that, the smart contract generates the token α_n by embedding the latest state n securely via secure PRF, i.e., $\alpha_n = G1_{k_\alpha}(t_w, n)$. Given the current token α_n for keyword w, the data node can retrieve all IDs from the chaining index. In particular, each matched entry is unmasked via decryption to get the document ID and the next entry address till no entry is returned.

During the query procedure, the keyword and document IDs are strongly protected. Each node only learns the query token, accessed index entries, and the encrypted result set. Note that an authorized user only needs to spend $O(1)$ time to generate the token for a keyword, and each data node spends $O(n)$ time to fetch document IDs in parallel, where n is the number of documents matching the query condition.

Algorithm 4 : Secure record insertion protocol

```
Input: Private key $k_o$; secure PRFs {G1, G2}; $\gamma \in g_2$; hash function $h : \{0,1\}^* \rightarrow g_1$; newly added
    document's ID $id_{new}$ for keyword $w$.
Output: Newly added index entry $\langle \alpha_{n'}, \beta_{n'} \rangle$.
 1: Derive $\{k_w, k_\alpha\}$ from $k_o$;
 2: $t_w \leftarrow \hat{e}(h(w), \gamma)^{k_w}$;
 3: $j \leftarrow route(t_w)$; // $j \in \{1, m\}$ is node ID
 4: $n, k_\beta^n \leftarrow S.find(t_w)$;
 5: $\alpha_n \leftarrow G1_{k_\alpha}(t_w, n)$;
 6: $n++$;
 7: $\alpha_{n'} \leftarrow G1_{k_\alpha}(t_w, n), k_\beta^{n'} \leftarrow KGen(n)$; //$k_\beta^{n'}$ is a fresh key
 8: $\beta_{n'} \leftarrow G2_{k_\beta^{n'}}(\alpha_n || k_\beta^n, id_{new})$;
 9: $S.update(t_w, n, k_\beta^{n'})$;
10: $I_j.put(\alpha_{n'}, \beta_{n'})$;
```

4.4 Secure record insertion

To enable dynamic search over encrypted data with forward security, we integrate the latest state value for a given keyword into the newly added index entry. Thus, the data node cannot learn whether the newly added index contains the keyword queried before. We now present the details of our proposed insertion protocol in Algorithm 4. Given the keyword w and the newly added document ID id_{new}, the data owner first obtains the current state n from table S and then updates it to build the newly added index entry, as shown from Line 4 to line 6 in Algorithm 4. Meanwhile, the previous index address α_n and encryption key k_β^n are re-masked with id_{new} by using a fresh key $k_\beta^{n'}$, i.e., $\beta_{n'} = G2_{k_\beta^{n'}}(\alpha_n || k_\beta^n, id_{new})$. Since the newly added index entry is generated from the latest state and a fresh key, the association between the searched keyword and the newly added document is fully protected. Formal security analysis will later be conducted in "Security analysis" section.

Algorithm 5 : Secure record deletion protocol

```
Input: Private key $k_o$; secure PRFs {G1, G2}; $\gamma \in g_2$; hash function $h : \{0,1\}^* \rightarrow g_1$; deleted
    document's ID $id_{del}$ for keyword $w$.
Output: Updated index.
  Derive $\{k_w, k_\alpha\}$ from $k_o$;
  $t_w \leftarrow \hat{e}(h(w), \gamma)^{k_w}$;
  $n, k_\beta^n \leftarrow S.find(t_w)$;
  $\alpha_n \leftarrow G1_{k_\alpha}(t_w, n)$;
  $node_j \leftarrow route(t_w)$;
  Send $\{\alpha_n, k_\beta^n, id_{del}\}$ to $node_j$;
$\underline{Node.Delete}$
  $\beta_n \leftarrow Node_j.find(\alpha_n)$;
  $\alpha_{n-1}, k_\beta^{n-1}, id_n \leftarrow Dec_{k_\beta^n}(\beta_n)$;
  if $id_n == id_{del}$ then
    $S.update(t_w, n\text{- -}, k_\beta^{n-1})$;
    Delete index entry $\langle \alpha_n, \beta_n \rangle$;
  else
    for $i \in \{n-1, 1\}$ do
      $\beta_i \leftarrow Node_j.find(\alpha_i)$; //shown in Algorithm 3
      $\alpha_{i-1}, k_\beta^{i-1}, id_i \leftarrow Dec_{k_\beta^i}(\beta_i)$;
      while $id_i == id_{del}$ do
        $\beta_{i+1} \leftarrow G2_{k_\beta^{i+1}}(\alpha_{i-1} || k_\beta^{i-1}, id_{i+1})$;
        Delete index entry $\langle \alpha_i, \beta_i \rangle$;
      end while
    end for
  end if
```

4.5 Secure record deletion

The corresponding record deletion protocol following the index construction is presented in Algorithm 5. The core idea of the deletion algorithm is to reconnect the secure index chain after removing the deleted entry. Specifically, a data node first executes the secure query protocol as shown in Algorithm 3 and locates all matched entries over the encrypted index chain. Then, each matched entry is unmasked to obtain the underlying document ID. After ID checking and deletion, the data node will reconnect the index chain by re-encrypting the previous entry β_{i+1} with the next entry's contents. If the document ID id_n of the first index entry α_n matches the deleted ID id_{del}, the smart contract also needs to update the state and the private key.

4.6 Encrypted keyword search example

To better understand the encrypted keyword search protocol of our blockchain-assisted framework, Fig. 2 uses an example to show how it works to search all file IDs matching the keyword "war." Specifically, the example of the MongoDB query is $db.col.find(\text{``}movie'' : \text{``}war'')$, which can select from "col'' collection all documents where the keyword "$movie''$ equals "war''. The authorized user with ID "UID01" first generates the query token $tk = h(\text{``}war'')^{k1}$ based on the query condition and sends it to the smart contact as a query transaction. Upon receiving the token from the client, the smart contract first checks its access permission via the user table and generates the keyword token $tk_{war} = e(tk, r^{k2})$. After that, the smart contract creates the latest index entry $\alpha_{20} = G1(tk_{war}, 20)$ with the keyword state 20 and sends it to the corresponding data node. Each data node processes these tokens in parallel. Specifically, all matched entries α are located via the index chain with the tokens and the document IDs id are revealed after decryption. Finally, the encrypted documents are returned to the client from the data node.

5 Security analysis

In this section, we will conduct rigorous security analysis of our proposed scheme. Specifically, we evaluate the security strength of secure keyword-match queries. Then, we discuss how our scheme can achieve forward security during the update operation.

5.1 Security on encrypted keyword search

The keyword-match index design is built on the framework of SSE scheme proposed in [5]. Once the data owner uploads the encrypted index to the data server, the size of indexes will be learned. During the query procedure, there will be the leakage of access pattern and query pattern. Explicitly, the access pattern indicates the search results; the query pattern is the repeated query tokens. Following the security notion of SSE, we first define the leakage functions for exact-match index initialization as follows:

$$\mathcal{L}_1^{kwd}(\mathbf{K}) = (\{Z_i\}_m, \langle|\alpha|, |\beta|\rangle)$$

where $\mathbf{K}$ is the set of keywords, m is the number of data nodes, Z_i is the node i's keyword-match index size, and $\langle|\alpha|, |\beta|\rangle$ are the index lengths of key-value pairs. After processing a keyword search request, we define the following leakage functions:

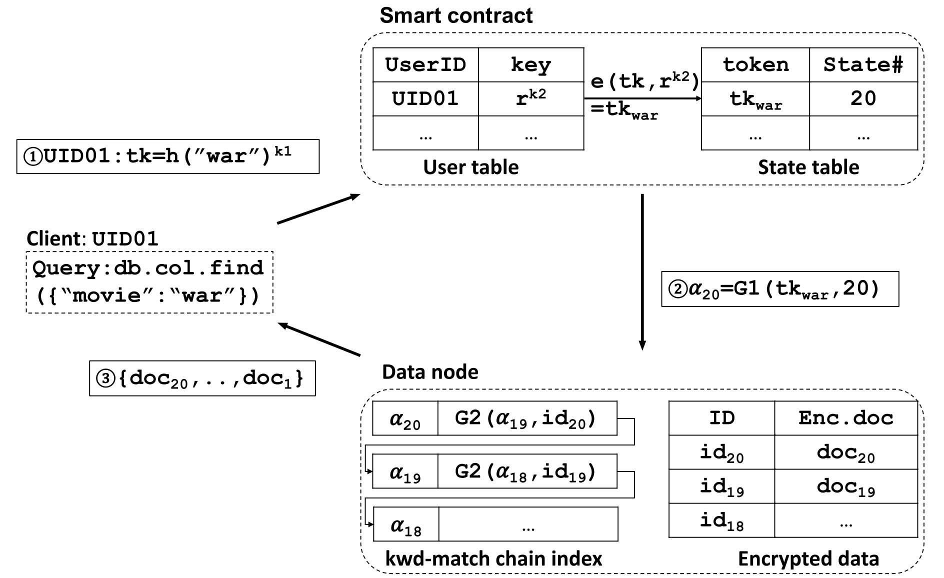


Fig. 2 Blockchain-assisted keyword search illustration

$$\mathcal{L}_2^{kwd}(K) = (t_K, \{\langle\alpha, \beta\rangle, id\}_n)$$

where K is the query keyword, t_K is the query token, and $\{\langle\alpha, \beta\rangle, id\}_n$ are n query results including the accessed index pairs and corresponding encrypted document IDs. In addition, we also define the leakage $\mathcal{L}_3^{kwd}$ to maintain repeated requests as follows:

$$\mathcal{L}_3^{kwd}(\mathbf{Q}) = (M_{q\times q})$$

where $\mathbf{Q}$ is q number of keyword search requests. $M_{q\times q}$ is the symmetric bit matrix that maintains the repeated requests. Each element in the $M_{q\times q}$ is initialized as 0. For $i, j \in [1, q]$, the elements of matrix $M_{i,j}$ and $M_{j,i}$ are equal to 1 if two tokens $t_i = t_j$. Given above leakage definitions, we provide the simulation-based security definition of the keyword-match scheme as follows:

Definition 1 Let $\Pi_{\mathsf{kwd}} = (\mathsf{KGen}, \mathsf{Build}_{\mathsf{kwd}}, \mathsf{Query}_{\mathsf{kwd}})$ be our secure keyword-match query scheme, and let $\mathcal{L}_1^{kwd}$, $\mathcal{L}_2^{kwd}$ and $\mathcal{L}_3^{kwd}$ be the leakage functions. Given a probabilistic polynomial time (PPT) adversary $\mathcal{A}$ and a PPT simulator $\mathcal{S}$, define the following probabilistic games $\mathbf{Real}_{\mathcal{A}}(k)$ and $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(k)$:

$\mathbf{Real}_{\mathcal{A}}(k)$: The data owner calls $\mathsf{KGen}(1^k)$ to get a private key K. $\mathcal{A}$ selects a dataset $\mathbf{D}$ and asks the owner to build $\{I_1^{kwd}, \cdots, I_m^{kwd}\}$ via $\mathsf{Build}_{\mathsf{kwd}}$. Then, $\mathcal{A}$ adaptively conducts a polynomial number of q queries with the tokens and ciphertexts generated by the owner. Finally, $\mathcal{A}$ returns a bit as the output.

$\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(k)$: $\mathcal{A}$ selects $\mathbf{D}$, and $\mathcal{S}$ builds $\{I_1'^{kwd}, \cdots, I_m'^{kwd}\}$ for $\mathcal{A}$ based on $\mathcal{L}_1^{kwd}$. Then, $\mathcal{A}$ adaptively performs a polynomial number of q queries. From $\mathcal{L}_2^{kwd}$ and $\mathcal{L}_3^{kwd}$ in each query, $\mathcal{S}$ generates the simulated tokens and ciphertexts, which are processed over $\{I_1'^{kwd}, \cdots, I_m'^{kwd}\}$. Finally, $\mathcal{A}$ returns a bit as the output.

Π_{kwd} is adaptively secure with $(\mathcal{L}_1^{kwd}, \mathcal{L}_2^{kwd}, \mathcal{L}_3^{kwd})$ if for all PPT adversaries $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that: $Pr[\mathbf{Real}_{\mathcal{A}}(k) = 1] - Pr[\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(k) = 1] \leq negl(k)$, where $negl(k)$ is a negligible function in k.

Theorem 1 Π_{kwd} *is adaptively secure with* $(\mathcal{L}_1^{kwd}, \mathcal{L}_2^{kwd}, \mathcal{L}_3^{kwd})$ *leakages under the random-oracle model if* $G1$, $G2$, *and* h *are secure PRFs.*

Proof Given $\mathcal{L}_1^{kwd}$, the simulator $\mathcal{S}$ simulates the encrypted keyword-match indexes $\{I_1'^{kwd}, \cdots, I_m'^{kwd}\}$ for m nodes, which have the same size Z as the real encrypted indexes. Each simulated entry contains $|\alpha|$-bit and $|\beta|$-bit random string as a key-value pair, which is indistinguishable from the real encrypted index entry.

From $\mathcal{L}_2^{kwd}$, $\mathcal{S}$ can simulate the first query token and results. On the simulated index, $\mathcal{S}$ randomly selects n entries, which are the same as the query request over the real one, and assigns the resulting id to the simulated entries. The random masked key-value pair can be simulated as $\alpha_i' = G1'(t', n), \beta' = G2'(\alpha_{i-1}', id)$, where $i \in \{1, n\}$ and t' is a random string as the simulated token, and id is identical to the one in the real keyword-match queries. In particular, we use random oracles $\{G1', G2'\}$ as PRFs $\{G1, G2\}$. From $\mathcal{L}_3^{kwd}$, S updates $M_{1,1} = 1$ in a matrix $M_{q \times q}$.

In the subsequent jth queries ($j \in \{2, q\}$), if the query appears repeatedly, $\mathcal{S}$ will choose the same tokens simulated before, and return the repeated matching results. Meanwhile, it will update the corresponding element in $M_{1,j}'$ and $M_{j,1}'$ to be "1." Otherwise, $\mathcal{S}$ will generate simulate tokens and operate random oracle to get the results as shown in the first query procedure.

Due to the pseudo-randomness of secure PRF, $\mathcal{A}$ cannot differentiate the outputs of the simulated experiment from the real one. □

5.2 Forward security analysis

As described in Sect. 4, we combine keyword state information stored on table S on the smart contract and a chaining index table stored on the cloud server to preserve our scheme to achieve forward security. Because the search trapdoor of keyword w is generated from the latest state of S associated with w, and this state updates once a new keyword/document pair (w, id) is added to the database. Meanwhile, each newly added entry needs to be encrypted by using fresh random masks generated from the latest state information. Cloud server does not know which already searched/updated keyword that current document contains. And it does not know newly updated search trapdoor of keyword w until next query of keyword w. Based on the construction of the chain-based index, the cloud server can recover neither the matched document id embedded with newly added key-value pair without the updated search trapdoor, nor learn whether the newly added entry is generated from the same keyword as that of those previously added entries without knowing the newly updated state information.

6 Experimental evaluation

6.1 Prototype implementation

We implement the proposed system prototype in C++ and perform the evaluation on Amazon Web Services. We create the AWS "C5.xlarge" instances with 4 vcores (3 GHz Intel Xeon® Platinum 8124M), and 8 GB RAM. In this experiment, we generate a Redis (v3.2.0) cluster that consists of 9 AWS "C5.xlarge" instances as data nodes of the database server and 4 AWS "C5.xlarge" instances as the multi-client of data applications. All instances are installed on Ubuntu server 14.04. We use Apache Thrift (*v*0.9.2) to implement the remote procedure call (RPC).

For cryptographic primitives, we use OpenSSL to implement the symmetric encryption via AES-128 and the pseudo-random function via HMAC-256. Our keyword-match indexes are integrated into the implementation of the distributed index framework proposed in [12]. In total, the prototype consists of more than 8500 lines of C++ code.

6.2 Performance evaluation

In our experimental evaluation, we target several practical aspects including initialization time, memory cost, query performance, and bandwidth overhead.

Index evaluation: We first assess the space consumption of keyword-match index (*kwd*-match) in Table 2. For the keyword-match index, we use AES-128 encryption algorithm to generate building blocks. Thus, the size of each key-value pair $\langle \alpha, \beta \rangle$ is 256 bits. As shown in Table 2, the index size of keyword-match increases linearly from 4.88MB (20K index entries) to 19.53MB (80K index entries).

Figure 3 presents the time cost of building the encrypted indexes at the client side. The time cost increases linearly with the number of index entries. For instance, it takes around 1.2s to generate $40K$ index entries, which is roughly half of the time cost when encrypting $80K$ keyword indexes.

Query evaluation To assess the system efficiency and security overhead, we measure the query throughput, the process latency under different workload, and the cost of record insertion and deletion. In this evaluation, we preload totally 160K data records to assess the practicality of our design for large-scale deployment.

To evaluate the scalability of our system, we first report the throughput for encrypted keyword match. By using different numbers of data nodes, we capture the total number of handled queries for a duration of 100s to obtain the throughput when each of the nodes is fully loaded. As shown in Fig. 4, we can find that the total number of index entries processed per second increases with the number of cores. When there are nine nodes at the cloud server, the keyword-match queries can achieve up to 157K entries per second. The overhead comes from the cost of secure PRF during keyword-match comparison. The results confirm that our design performs satisfactorily at scale.

To gain a deeper understanding on the query performance of our proposed design, we further evaluate the query latency for encrypted keyword-match. It worth to note that our encrypted index chain can map duplicates to single chain reference and locates them all in a scan. Overall, our evaluation shows that the query latency decreases with the increasing number of nodes. As shown in Fig. 5, the query latency of keyword-match decreases from about 2.4s to 0.8s as the number of nodes increases from 3 to 9 when returning 32K data records.

Figure 6 also compares the keyword-match query performance with the scheme proposed in [31] denoted as Sophos when returning a fixed number of results. Our scheme achieves better performance than Sophos because their construction requires heavy cryptographic primitives and RSA encryption, which incurs considerable burden for query performance. Overall, we can confirm that our design benefits from the local index framework and can effectively process queries in parallel.

In this experiment, we also evaluate the incremental scalability by measuring the time cost for keyword-match index insertion. We note that the time cost includes the network transmission cost for each newly added entry; thus, it is much higher than the index building time as shown in Fig. 3. When the number of newly added entries is 32K, Fig. 7 shows that it just takes around 6.6s to add these index entries to the encrypted index chain. Meanwhile, we also evaluate the efficiency of delete algorithm

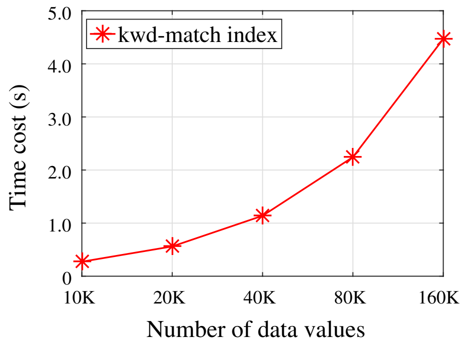


Fig. 3 Keyword-match index build

Table 2 Space consumption of encrypted index

# Entries	20K	40K	80K	160K
Keyword index	4.88 (MB)	9.77 (MB)	19.53 (MB)	39.06 (MB)

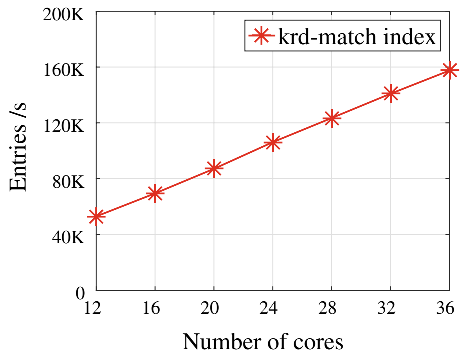


Fig. 4 Keyword-match throughput

in Fig. 8. As mentioned in Sect. 4, the process of delete operation is exactly the same as the insert operation so that the efficiency of the add and delete algorithm is almost the same. Specifically, it only takes 7.4s to delete 32K records.

Bandwidth evaluation Recall that the distributed index framework requires the client to generate query tokens for each node. To understand the bandwidth overhead, Fig. 9 shows the ratio between the query token size and result size. The result indicates that the bandwidth ratio of keyword-match decreases gradually with the increased size of results. When there are 50 nodes at the cloud side, the bandwidth ratio drops from about 2.50% to approximately 0.16% when the number of retrieved result values rises from 2K to 32K. On the other hand, the result shows that the increasing number of nodes can render a rise in the bandwidth. The ratio of 8K result size increases from about 0.125% to 0.625% as the number of nodes increases from 10 to 50. Nevertheless, the bandwidth overhead is still negligible to the size of results.

7 Related work

7.1 Searchable symmetric encryption

Keyword search over encrypted data has been an active research area in the past decade. With encrypted search, it allows an untrusted server to conduct a secure comparison between the encrypted indexes and the trapdoors without leaking the searched keywords [32–38]. In 2000, Song et al. first introduced the notion of searchable encryption [39]. In [40], the security notion of Searchable Symmetric Encryption (SSE) is formalized. The notion of dynamic SSE was further formalized in [7]. To improve I/O efficiency, Cash et al. [5] developed a dynamic scheme optimized for large datasets. However, the existing dynamic SSE schemes would leak the updated keyword in the newly added documents. In [41], Stefanov et al. presented the notion of forward security and proposed an ORAM-like index by using a hierarchical structure, but the overhead of ORAM is too high for a practical use of the SSE scheme. The motivation for studying forward security came from file injection attacks on SSE by Zhang et al. [42]. Bost et al. formally defined forward security in [31] and designed an insertion-only SSE scheme with optimal search and update complexity, based on asymmetric cryptography (i.e., trapdoor permutations). But, this construction requires heavy public key encryption operations, which leads to performance

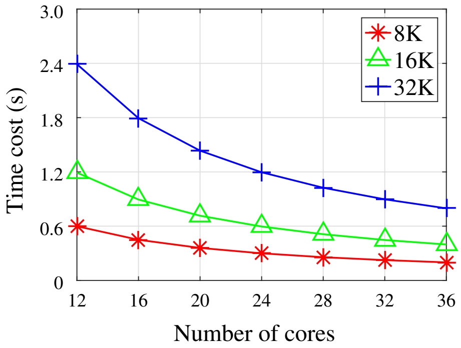
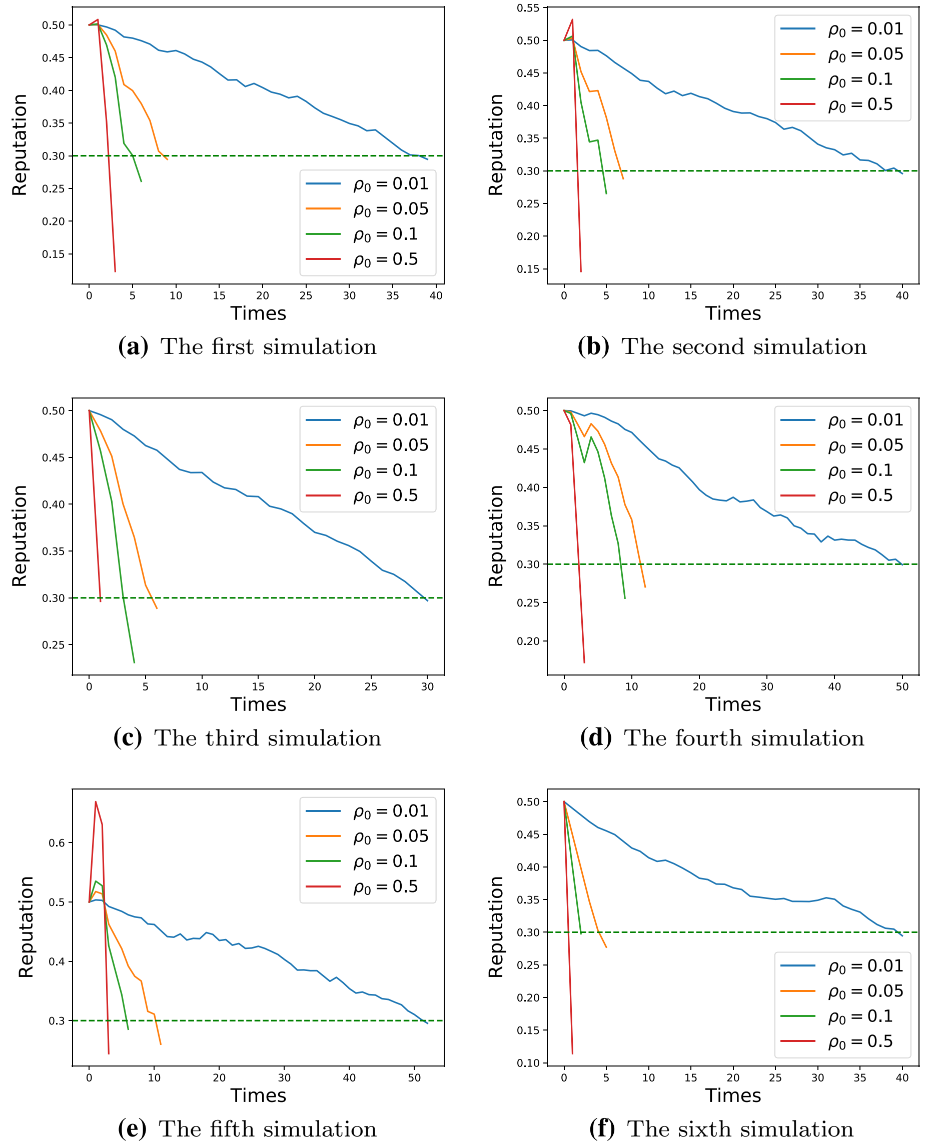


Fig. 5 Keyword-match latency



(a) The first simulation **(b)** The second simulation

(c) The third simulation **(d)** The fourth simulation

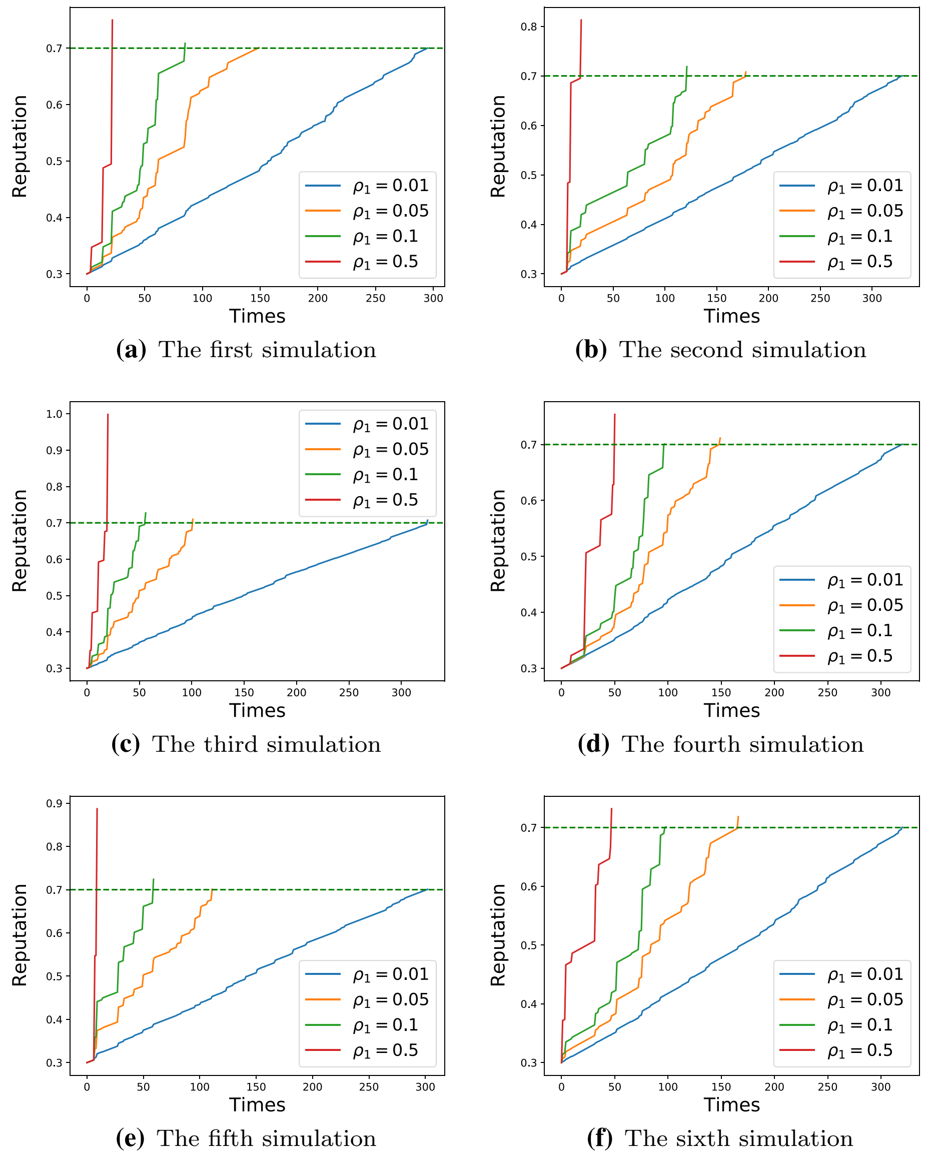
(e) The fifth simulation **(f)** The sixth simulation

Fig. 6 Query latency comparison

degradation. Moreover, the above work on encrypted search are mostly designed for centralized systems.

7.2 Multi-client access in searchable encryption

In [40], Curtmola et al. proposed the first construction for multi-user SSE based on broadcast encryption. Jarecki et al. [43] leveraged oblivious PRF to enhance the access policies. In [44], Sun proposed for Boolean queries make existing multi-client query protocols non-interactive so as to reduce the communication overhead. The schemes [45, 46] considered the multi-client setting in distributed key-value stores. Unfortunately, neither designs can achieve forward security.



(a) The first simulation

(b) The second simulation

(c) The third simulation

(d) The fourth simulation

(e) The fifth simulation

(f) The sixth simulation

Fig. 7 Secure insertion latency

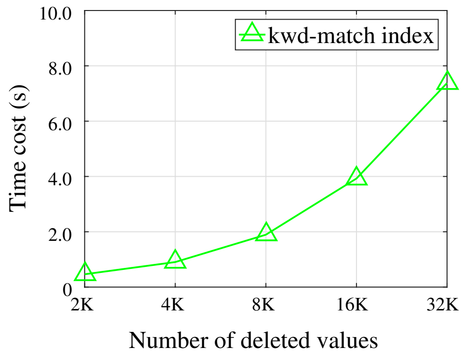


Fig. 8 Secure deletion latency

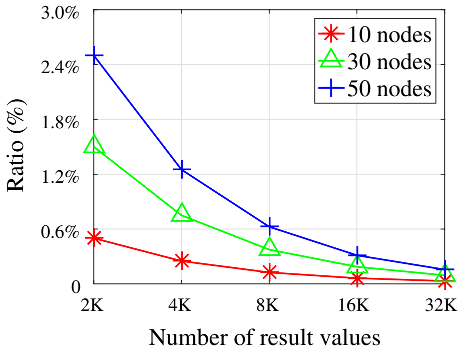


Fig. 9 Query token bandwidth overhead

8 Results and discussion

Our blockchain-assisted secure data sharing framework has three advantages:

It is secure Since users' data and file indexes are all encrypted, both blockchain nodes and storage server are not able to obtain any information from the stored data, search queries, or search results.

It is efficient Data indexes are collocated with the data and stored at storage server, which makes the blockchain lightweighted and the search operations more efficient. Besides, by leveraging the smart contract to construct query tokens, a data owner can authorize query permission without extra round interaction.

It is fairness Query authorization is maintained at the smart contract, which ensures the data sharing services non-deniable without involving any third-party authority.

As future work, we plan to explore advanced searchable encryption schemes to support other SQL query services, such as range queries and join operations. Meanwhile, we leave how to detect malicious data owner who submit invalid data to intentionally disrupt the system as our future work.

9 Conclusion

In this paper, we present a completely new system architecture enabling secure multi-client queries in distributed database systems. We propose to leverage the smart contract of blockchain as a trusted party for secure query authorization and integrate dynamic SSE scheme with bilinear pairings, achieving forward privacy for the update operation. Extensive experiments show that it preserves advantages in existing distributed database systems such as high throughput, low latency, incremental scalability, and fine availability.

Abbreviations

PRF:: Pseudo-random function.

Acknowledgements

This work was supported by the Fundamental Research Funds for the Central Universities under Grants 310421108 and National Key R&D Program of China (No. 2019YFB2102600).

Author's contribution

YG initiated this project and designed the framework. SW and JH have been involved in drafting the manuscript. All authors read and approved the final manuscript.

Funding
Not applicable.

Availability of data and materials
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Author details
[1]School of Artificial Intelligence, Beijing Normal University, Beijing, China. [2]Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China.

Received: 16 December 2020 Accepted: 2 August 2021
Published online: 19 August 2021

References

1. Redis: An Advanced Key-Value Cache and Store. Online at http://redis.io/ (2015)
2. J. Ousterhout, A. Gopalan, A. Gupta, A. Kejriwal, C. Lee, B. Montazeri, D. Ongaro, S.J. Park, H. Qin, M. Rosenblum et al., The RAMCloud storage system. ACM TOCS **33**(3), 7 (2015)
3. G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, W. Vogels, Dynamo: Amazon's highly available key-value store. ACM SIGOPS Oper. Syst. Rev. **41**(6), 205–220 (2007)
4. Information is Beautiful: World's Biggest Data Breaches. Online at http://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/ (2016)
5. D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Rosu, M. Steiner, Dynamic searchable encryption in very large databases: Data structures and implementation. In: Proc. of NDSS (2014)
6. R. Curtmola, J.A. Garay, S. Kamara, R. Ostrovsky, Searchable symmetric encryption: improved definitions and efficient constructions. J. Comput. Secur. **19**(5), 895–934 (2011)
7. S. Kamara, C. Papamanthou, T. Roeder, Dynamic searchable symmetric encryption. In: Proceedings of ACM CCS (2012)
8. V. Pappas, B. Vo, F. Krell, S. Choi, V. Kolesnikov, A. Keromytis, T. Malkin, Blind Seer: A Scalable Private DBMS. In: Proceedings of IEEE S&P (2014)
9. S. Kamara, T. Moataz, SQL on Structurally-Encrypted Databases. Cryptology ePrint Archive, Report 2016/453. http://eprint.iacr.org/2016/453 (2016)
10. R.A. Popa, C. Redfield, N. Zeldovich, H. Balakrishnan, CryptDB: protecting confidentiality with encrypted query processing. In: Proceedings of ACM SOSP (2011)
11. X. Yuan, X. Wang, C. Wang, C. Qian, J. Lin, Building an encrypted, distributed, and searchable key-value store. In: Proceedings of ACM AsiaCCS (2016)
12. X. Yuan, Y. Guo, X. Wang, C. Wang, B. Li, X. Jia, Enckv: An encrypted key-value store with rich queries. In: Proceedings of ACM AsiaCCS (2017)
13. R. Poddar, T. Boelter, R.A. Popa, Arx: A strongly encrypted database system. Cryptology ePrint Archive, Report 2016/591 (2016)
14. Y. Guo, X. Yuan, X. Wang, C. Wang, B. Li, X. Jia, Enabling encrypted rich queries in distributed key-value stores. IEEE TPDS **30**(7), 1283–1297 (2018)
15. Q. Wang, Y. Guo, H. Huang, X. Jia, Multi-user forward secure dynamic searchable symmetric encryption. In: Proceedings of Conference on Network and System Security, pp. 125–140 (2018)
16. Z. Yupeng, K. Jonathan, P. Charalampos, All your queries are belong to us: The power of file-injection attacks on searchable encryption. In: Proceedings of USENIX (2016)
17. G. Wood, Ethereum: A secure decentralised generalised transaction ledger. In: Ethereum Project Yellow Paper (2014)
18. G. Wang, Z.J. Shi, M. Nixon, S. Han Sok: Sharding on blockchain. In: Proceedings of the 1st ACM Conference on Advances in Financial Technologies, pp. 41–61 (2019)
19. G. Wang, M. Nixon: Randchain: Practical scalable decentralized randomness attested by blockchain. In: 2020 IEEE International Conference on Blockchain (Blockchain) (2020). IEEE
20. H. Shi, S. Wang, Q. Hu, X. Cheng, J. Zhang, J. Yu, Fee-free pooled mining for countering pool-hopping attack in blockchain. IEEE TDSC (2020)
21. H. Shi, S. Wang, Y. Xiao, Queuing without patience: A novel transaction seletion mechanism in blockchain for iot enhancement. IEEE IoT-J (2018)
22. G. Wang, M. Nixon, M. Boudreaux, Toward cloud-assisted industrial iot platform for large-scale continuous condition monitoring. Proc. IEEE **107**(6), 1193–1205 (2019)
23. G. Wang, Z. Shi, M. Nixon, S. Han, Chainsplitter: Towards blockchain-based industrial iot architecture for supporting hierarchical storage. In: 2019 IEEE International Conference on Blockchain (Blockchain), pp. 166–175 (2019). IEEE
24. H. Shi, S. Wang, Q. Hu, X. Cheng, J. Zhang, J. Yu, Fee-free pooled mining for countering pool-hopping attack in blockchain. IEEE TDSC (2020)
25. H. Shi, S. Wang, Y. Xiao, Queuing without patience: A novel transaction seletion mechanism in blockchain for iot enhancement. IEEE IoT-J **7**(9), 7941–7948 (2020)
26. S. Wang, C. Wang, Q. Hu, Corking by forking: Vunerability analysis of blockchain. In: INFOCOM (2019). IEEE
27. Q. Hu, S. Wang, X. Cheng, A game theoretic analysis on block withholding attacks using the zero-determinant strategy. In: IWQoS (2019). IEEE
28. S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system. Online at https://bitco.in/pdf/bitcoin.pdf

29. Ethereum: Ethereum blockchain app platform. Online at https://www.ethereum.org
30. G. Wang, Z.J. Shi, M. Nixon, S. Han, Smchain: A scalable blockchain protocol for secure metering systems in distributed industrial plants. In: Proceedings of the International Conference on Internet of Things Design and Implementation, pp. 249–254 (2019)
31. R. Bost: Sophos - Forward Secure Searchable Encryption. Cryptology ePrint Archive, Report 2016/728 (2016)
32. Y. Guo, H. Xie, Y. Miao, C. Wang, X. Jia, Fedcrowd: A federated and privacy-preserving crowdsourcing platform on blockchain. IEEE TSC (2020)
33. H. Xie, Y. Guo, X. Jia, A privacy-preserving online ride-hailing system without involving a third trusted server. IEEE TIFS (2021)
34. Y. Guo, M. Wang, C. Wang, X. Yuan, X. Jia, Privacy-preserving packet header checking over in-the-cloud middleboxes. IEEE IoT-J **7**(6), 5359–5370 (2020)
35. R. Kui, Y. Guo, J. Li, X. Jia, C. Wang, Y. Zhou, S. Wang, N. Cao, F. Li, Hybridx: New hybrid index for volume-hiding range queries in data outsourcing services. In: Proceedings of IEEE ICDCS (2020)
36. Y. Guo, C. Zhang, X. Jia, Verifiable and forward-secure encrypted search using blockchain techniques. In: Proceedings of IEEE ICC (2020)
37. C. Zhang, Y. Guo, X. Jia, C. Wang, H. Du, Enabling proxy-free privacy-preserving and federated crowdsourcing by using blockchain. IEEE IoT-J (2020)
38. Z. Chen, Y. Guo, H. Du, X. Jia, :Pfcrowd: Privacy-preserving and federated crowdsourcing framework by using blockchain. In: Proceedings of IEEE IWQoS (2020)
39. D. Song, D. Wagner, A. Perrig, Practical techniques for searches on encrypted data. In: Proceedings of IEEE S&P (2000)
40. R. Curtmola, J. Garay, S. Kamara, R. Ostrovsky, Searchable symmetric encryption: improved definitions and efficient constructions. In: Proceedings of ACM CCS (2006)
41. E. Stefanov, C. Papamanthou, E. Shi, Practical dynamic searchable symmetric encryption with small leakage. In: Proceedings of NDSS (2014)
42. J.K. Yupeng Zhang, C. Papamanthou, All your queries are belong to us: The power of file-injection attacks on searchable encryption. In: Proceedings of USENIX Security (2016)
43. S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, M. Steiner, Outsourced symmetric private information retrieval. In: Proceedings of ACM CCS (2013)
44. S.-F. Sun, J.K. Liu, A. Sakzad, R. Steinfeld, T.H. Yuen, :An efficient non-interactive multi-client searchable encryption with support for Boolean queries. In: Proceedings ESORICS (2016)
45. W. Lin, X. Yuan, B. Li, C. Wang Multi-client searchable encryption over distributed key-value stores. In: Proceedings of SMARTCOMP (2017)
46. X. Yuan, X. Yuan, B. Li, C. Wang, Secure multi-client data access with boolean queries in distributed key-value stores. In: Proceedings of CNS (2017)

Publisher's note