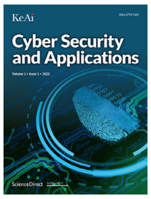
Cyber Security and Applications 1 (2023) 100006

Contents lists available at ScienceDirect

Cyber Security and Applications

journal homepage: http://www.keaipublishing.com/en/journals/cyber-security-and-applications/



A secure and efficient DSSE scheme with constant storage costs in smart devices☆

Weiwei Yan[a,*], Sai Ji[a,b]

[a] School of computer and software, Nanjing University of Information, Science and Technology, Nanjing, 210000, China
[b] Suqian University, Suqian, 223800, China

A R T I C L E I N F O

Keywords:
DSSE
Smart devices
Storage cost
Forward security

A B S T R A C T

With the continuous development of the Internet of Things (IOT) and cloud computing, smart devices are playing an increasingly important role in users' daily life. Dynamic searchable symmetric encryption (DSSE) schemes are popular on smart devices because of their efficient retrieval performance and low computational overhead. Traditional DSSE with forward update privacy and backward security can resist file injection attack and statistical inference attack. However, it is high cost and not suitable for smart devices due to large local storage and low storage capacity. To achieve forward update privacy, we design a novel index structure called RC-II (Inverted Index with Retrieve Control) for search control which improves the security of DSSE. Besides, we combine on and off-chain to decrease client's local storage to a constant. Specifically, we transfer a significant amount of local overhead to the service peers off blockchain. We solve the trust problem between the client and the service by putting authentication data on blockchain. Compared with the state-of-the-art schemes, our scheme has a constant client storage overhead and an excellent retrieval performance which provides guarantee for smart devices under IOT environment.

1. Introduction

A large number of smart devices are playing an important role in users' daily lives with the development of Internet of Things (IOT) and cloud computing. Smart devices have wide applications such as sensitive data storage and data search [20,22]. However, sensitive information is easy to leak, and data outsourced to cloud may bring search leakage. Besides, smart devices are often unable to perform a large number of complex operations due to limited hardware resources. Searchable symmetric encryption (SSE) [26] schemes use lightweight operations to ensure data security and search privacy which are fit with the smart devices for IOT outsourced data.

SSE allows users to retrieve encrypted data directly without downloading all the ciphertext from the cloud service platform (CSP) [24]. Compared with Public key Encryption with Keyword Search (PEKS) [2], SSE is more efficient and have broad application scenario because it does not need complex operations such as bilinear pair operation. These make it more suitable for smart devices. SSE needs to support add/delete operation. Dynamic searchable symmetric encryption (DSSE) appears [3]. In recent years, attacks against DSSE have become more frequent [7,14]. Stefanov et al. [27] defines the forward security that the leaked information under update operation would corrupts the existing index in the CSP. Although many scholars have proposed schemes with forward security, the latest research shows that schemes with forward security will expose the relationship between updated data and existing data in the index after the next retrieval operation which means attacks such as statistical inference attack can still recover the keywords [19]. The strong forward update privacy and poor forward update privacy, defined by Li et al. [19], can resist statistical inference attack. Their scheme is based on poor forward update privacy and achieve efficient deletion operation, but the local storage overhead is too large. For example, In order to generate the wikipedia data index of one day, the user needs to incur storage overhead of close to 1 GB on the local side [19]. Although the existing scheme realizes the retrieval function on smart devices, the high storage overhead is still a difficult problem to solve. To better match the performance of smart devices, how to balance local storage overhead and cloud retrieval efficiency while ensuring security is still a hot target of current research.

Under the previous DSSE scheme, the client can only retrieve one chain or all the chains corresponding to one keyword. Although it is safer to retrieve only one chain, the corresponding performance will be reduced (because you need to send more tokens). The performance of re-

☆ This work is supported by the National Natural Science Foundation of China under Grants No. U1836115, No. 61922045, No. 61877034, the Natural Science Foundation of Jiangsu Province under Grant No. BK20181408, the CICAEET fund, and the PAPD fund.

* Corresponding author.
E-mail addresses: 20201221055@nuist.edu.cn, yanyunyanwei9140@163.com (W. Yan), jisai@nuist.edu.cn (S. Ji).

https://doi.org/10.1016/j.csa.2022.100006
Received 4 March 2022; Received in revised form 12 June 2022; Accepted 15 August 2022
Available online 23 August 2022

Table 1
Comparison with Existing DSSE Schemes.

	search computation	update computation	F.UP	B.P	verifiable	client storage
Bost [3]	O(b_w)	O(1)	×	×	×	O(m)
Bost et al. [5]	O((b_w- d_w)	O(1)	×	√	×	O(m))
Kim et al. [18]	O(b_w)	O(1)	×	√	×	O(m))
Cai et al. [6]	O($m+2k_t$)/O($1+2k_t$)	O($1+2k_t$)	×	×	×	O(m)
He et al. [13]	O(b_w + $Clen$)	O(Clen)	×	√	×	O(1)
Li et al. [19]	O(b_w)	O(1)	√	√	×	O(m + n)
Ge et al. [10]	O(n)	O(m)	×	×	√	O(m + n)
Ours	O(b_w)	O(1)	√	√	√	O(1)

F.UP represents forward update privacy and B.P represents backward privacy. m represents the number of keywords while n is the number of documents. b_w is the number of entries that matching keyword w. d_w is the number of deleted entries matching w. Under Cai et al.'s [6]scheme, based on whether the keywords retrieved have been searched before, Cai et al. [6] is executed in two cases, computation complexity is O($N+2k_t$) and O($1+2k_t$) respectively. k_t Represents the cost of conducting a transaction. $Clen$ is a constant which represents the maximum of update times.

trieving multiple chains is superb, the corresponding security cannot be guaranteed. We design a reverse index structure with retrieval control called RC-II. Clients can send different tokens to the server to retrieve one or more chains, which ensures both security and efficiency. Most of the previous research failed to achieve the forward and backward security with the constant client storage. The reason for this is that clients cannot transfer their locally storage to the CSP, which could bring serious leak problems. To deal with this question, we propose a clever solution. We adopt currently hot technology-blockchain [1]. By setting up smart contacts on blockchains with service peers who are individual clients and want to get rewards by providing storage and computation resources, we can easily transfer client's local storage overhead to service peers off chain. The nodes on chain are precious and only a small amount of data can be stored there. Therefore, we upload the data used for verification to the chain and put encrypted files and index off chain. Specifically, we upload files and index to the CSP. And in order to save the client's local storage overhead, we package this part of data to the service peers and let the service peers bear this overhead. At the same time, considering the replay attack, that is, the service peers return the un-updated data, we need to introduce the update counter in the local overhead part to prevent the malicious behavior of service peers. By the collaboration of on chain and off chain, our scheme can achieve efficient retrieval with very low local storage overhead, as well as detect the malicious behavior of CSP and service peers.

1.1. Motivations

DSSE allows users to ensure their search privacy when performing retrieval operations using smart devices. The current attack on DSSE requires SSE to meet more security requirements. However, the extra storage and computing overhead brought by the requirements makes it difficult to use in smart devices. Specifically, forward update privacy which guarantees protocol's ability to resist statistical inference attack requires the separation of the updated index from the existing index during retrieval. To achieve this feature, protocols need to record more data for each keyword on the local side. When a large amount of data is updated, this storage overhead affects the performance of smart devices.

We propose a DSSE scheme with constant client storage. It can also achieve forward and backward security when updating index. Compared with previous research, our scheme can verify the returned data and resist replay attack from CSP and from service peers at the same time. Besides, each entry who joins our protocol can monitor the illegal behavior of others. All of these ensures our scheme can achieve correct execution and low storage cost. This provides a strong prerequisite for the application of this solution for IOT outsourced data. The comparison with previous works is given in Table 1. The list given below details our contributions.

1.2. Our contributions

- **We propose a novel index structure called RC-II which achieves search control.** Traditional DSSE schemes only allow the CSP to retrieve one or all chains corresponding to one keyword and it will bring efficient and security problems. Search control means that client can choose to let the CSP retrieve one chain or all chains by providing different search token, which improves client's control over index on the CSP during search operation.
- **Based on RC-II, we propose a scheme with poor forward update privacy and backward security.**Both strong and poor forward update privacy ensures the separation of updated data from the index in the CSP, even during the next search. To obtain a good tradeoff between security and performanc, our scheme meets poor forward update privacy. Our scheme can also resist replay attack.
- **To meet the needs of smart devices, we combine on and off-blockchain to decrease client's local storage cost to a constant.** Specifically, the client transfers the storage overhead to the service peers and he only needs to record three symmetric keys to realize complete retrieval and update operations. Besides, each party can monitor the legitimacy of others in our system. These are the guarantees of the proper implementation of our protocol.
- We implement our scheme in Python and conduct experiments to test its latency in encrypted search and update processes, as well as communication costs. The results indicate that our scheme has excellent retrieval performance and very low local overhead while providing adaptive security.

The rest of the paper is structured as follows. Section 2 overviews some related works and Section 3 explains some background information. System model and threat model are given in Section 4. Section 5 introduces the combination of on and off-chain and the designed index control node RC-II. After that, we present our scheme. In Section 6 and 7, we give the security analysis and conduct experiments to test the efficiency of our design. Finally, we elaborate the conclusions in Section 8.

2. Related work

In 2000, SSE was first proposed by Song et al. [26] as a new type of cryptographic primitive. This scheme realizes the function of keyword search in ciphertext based on symmetric encryption algorithm. SSE needs to support dynamic data update: the addition and deletion operations. In 2012, the first DSSE scheme with sublinear retrieval efficiency was proposed by Kamara et al. [16]. Clients update encrypted index by sending an update token to the CSP. But their scheme reveals the hash value of the keyword during update operation. By increasing the spatial complexity, Kamara and Papamanthou [15] successfully solved these problems. Cai et al. [6] applied DSSE scheme to distributed system.

While their scheme adopts sequential index and the retrieval efficiency is sublinear with the keyword-identifier pairs when searching a new keyword. Zuo et al. [29] added the function of range retrieval to the scheme, which greatly improved the efficiency of ciphertext retrieval while ensuring the forward and backward security of the scheme. By building a binary tree at the local side, their scheme can achieve efficient search. However, the homomorphic encryption algorithm adopt in this scheme has high computational overhead and is not suitable for smart devices. In order to ensure the reliability of the retrieval results, Ge et al. [10] put the validation information into the index, and their scheme needs only one round of communication overhead. While the scheme will send the symmetric key corresponding to the keyword client wants to search, the security of the scheme will become worse and worse with the increase of retrieval times.

Important information is often leaked during data updating under several attacks. Previous studies have shown that the CSP can recover about 70 percent of the client's encrypted keyword by file injection attack [28]. Therefore, DSSE needs to separate the existing index from the current updated data. Stefanov et al. [27] first gave the definition of forward security and its concrete construction. However, their scheme suffers from communication and computing costs. In order to reduce them, Bost [3] eliminates the need for interaction between the CSP and the client when updating data. The DSSE scheme has the problem that the deleted documents can still be retrieved when the documents are deleted. Stefanov et al. [27] first mentioned the concept of backward security, but did not gave a specific definition. Bost et al. [5] first gave the standard definition and concrete construction of backward security. However, their schemes are limited by the impact of keyword-document pairs. Kermanshahi et al. [17] proposed two DSSE schemes that support geometric range retrieval, of which the first supports forward security and the second supports backward security. However, both of them need homomorphic encryption which has poor retrieval performance. Liu et al. [21] protected DSSE scheme from pattern leakage. The scheme shares data to the CSP through Shamir Secret Sharing which has higher security. However, the scheme adopts oblivious RAM(ORAM) technology, which is relatively low in practicability, and the high local storage will also bring a lot of overhead. He et al. [13] proposed a new chain-type storage structure called fishbone chain which greatly reduces the local storage overhead. But this scheme suffers from the maximum number of updates, only periodic updates are supported.

Recent researches show that the traditional forward security will link updated index with existing indexes in the CSP in the next retrieval [19] which will cause the scheme to be vulnerable to statistical inference attack [7,14,25] and will still cause the leak of the retrieved keyword. Therefore, how to ensure that the updated index and the keywords are still separated from each other after the next retrieval is the key to guarantee the security of DSSE. Meanwhile, as the client transfers files and index to the CSP, the client's local storage overhead increases linearly. This puts a lot of storage burden on smart devices. Therefore, how to reduce the storage cost of DSSE must be considered.

3. Preliminaries

In this section, We introduce the preparations required for this paper, including DSSE and set hash function.

3.1. DSSE

DSSE generally includes 7 algorithms (Setup, KeyGen, Enc, SrchTokenGen, Search, UpTokenGen, Update). DSSE typically have two actors, the data owner and the CSP. The data owner sends encrypted data and index to the CSP. When the owner wants to retrieve the documents containing a keyword, the owner sends a search token containing the keyword information to the CSP. The CSP uses the search token to traverse the index and return the search results that meet the keyword. when the owner needs to update the data, he generates the index of the updated data as the update token. Send the file and the corresponding update token to the server. The CSP uses the update token to update the index on the cloud, completing an update operation. A formal description of the algorithm is given below.

1) **Setup**: This algorithm is executed by the data owner. For all files, users extract their keywords and construct a index based on the keyword-document pair.

2) **KeyGen**($1^u \rightarrow k$): User inputs a security parameter u and outputs a symmetric key k.

3) **Enc**($it,\ k \rightarrow IT$): The user uses the symmetric key K to encrypt the index to generate an encrypted index IT. The encrypted file and the encrypted index IT are then sent to the CSP.

4) **SrchTokenGen**($w,\ k \rightarrow st_w$): The user first selects the keyword w that he wants to search, and then generates a search token st_w with the keyword w and the symmetric key K, and then sends the stw to the CSP.

5) **Search**($IT,\ st_w \rightarrow ID(w)$): The algorithm is executed by a CSP. After receiving the user's search token st_w, the CSP will search in the index and output all the document identifiers that conform to this keyword. Then the CSP will send the documents corresponding to these document identifiers to the user.

6) **UpTokenGen**($w,\ k,\ id \rightarrow ut_w$): The algorithm is executed by the data owner who generates the update token ut_w by inputing the identifier of the files which he wants to update and then sent the files and ut_w to the CSP.

7) **Update**($ut_w \rightarrow IT$): After receiving the ut_w, the CSP update the index IT by the ut_w.

3.2. Blockchain

Blockchain originated with Bitcoin [12]. In essence, blockchain is a shared, immutable distributed ledger technology that facilitates the process of recording transactions and tracking assets across business networks. Each block is linked by a random hash that contains the hash value of the previous block. It has three characteristics: transparency, liveness and eventual consensus. Users can create smart contracts with users on the chain to make transactions [23]. The content of the smart contract is enforced, and the record of the transaction is stored on the blockchain, which is open and transparent to all users, ensuring the security of the transaction. Blockchain has been playing an important role in many fields such as logistics, railways and government work.

3.3. (Multi) set hash function

(Multi) set hash function maps the data of a set to a string [8]. Among the set hash, $MSet-Mu-Hash$ is collision-resistant. A hash function is defined as quadruple of probabilistic polynomial algorithms $(H, \equiv H, +H, -H)$ if for all $S \subset \mathbb{S}$, the hash function meet the following conditions.

$$H(S) \equiv {}_H H(S) \tag{1}$$

$$\forall \mathrm{x} \in \mathbb{S} \backslash S, H(S \cup \{\mathrm{x}\} \equiv {}_H HS +{}_H H\{\mathrm{x}\} \tag{2}$$

$$\forall \mathrm{x} \in S, H(S \backslash \{\mathrm{x}\} \equiv {}_H HS -{}_H H\{\mathrm{x}\} \tag{3}$$

Equations (2) and (3) present two operations for the set hash, (2) present the operation of adding data x to the set, and (3) present the operation of deleting data x to the set. In this paper, we apply set hash function to data validation.

4. System model and threat model

4.1. System model

In our system, there are four main participants: service requester with smart device, CSP, service peer and blockchain. Service requester

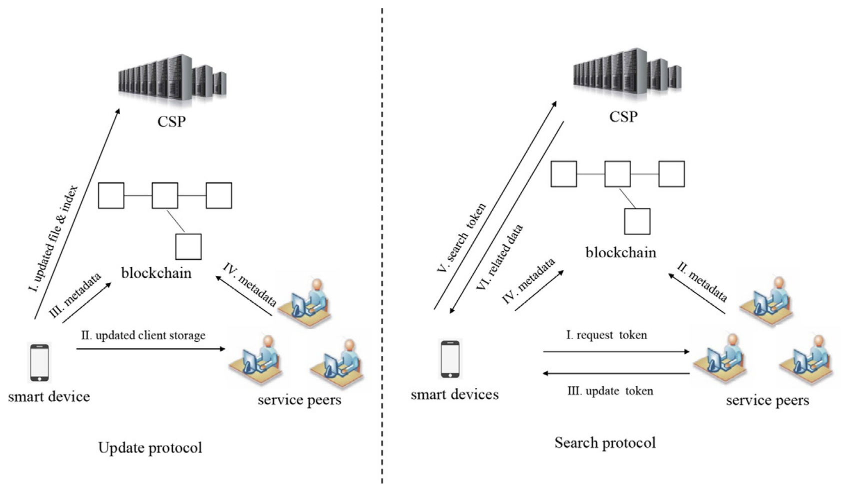


Fig. 1. System Model.

is intended to upload their documents via smart devices and enjoy efficient search service. CSP provides storage and retrieval services. It needs to receive data sent from clients, update index, provide retrieval services and return the corresponding results after receiving a search token from clients. Service peers are individual nodes who receive financial rewards by providing storage and computing services. Specifically, they take on some of the data that devices use to complete the retrieval operation. Blockchain in this paper is mainly used for recording transactions between smart devices and service peers and storing some data for verification. Besides, service requester and service peers will make a smart contract on chain. Smart contract is mandatory and will automatically transfer the requester's money to the service peer after he confirms the result. These are advantages that trust third party (TTP) cannot provide.

The system flow is shown in the Fig 1. The service requester sends document and index via smart device to the CSP, transfers the data which helps to complete retrieval operation to the service peer and generates metadata on the chain. After receiving the request, service peer generates metadata for the received data and upload it to the blockchain. During search phase, requester first uses device and makes a request to the peer and generates metadata on the chain. Based on the data sent by the requester, the peer returns the corresponding result and generates metadata on the blockchain. After receiving the data, the smart device verifies the validity of the returned data against the on-chain data. In the meantime, the smart contract will set a verification period and freeze some of the requester's assets. He can verify the returned data within a certain period of time with smart devices. If the results is correct and the money in his account will be automatically transferred to the peer account. Afterwards, the requester can make a search operation using the correct returned data. Besides, the verification data are stored in the service peer. When the CSP returns the data, the requester will also verify this part of the data and complete a retrieval operation if and only if he confirms that the result is correct.

In this scenario, we believe all the entries may be malicious. 1) CSP may return some error/un-updated results for the sake of saving computation and storage overhead and blame the error to the service peers. At the same time, it will also be curious about the data stored by the smart devices. 2) For service peers, we also consider them malicious. Specifically, they will return some incorrect data to avoid some consumption of computing resources. Since requester send the data for retrieval to the service peers, it is not desirable for him to let the service peers privately retrieve the data and acquire knowledge. 3) For the requester, we consider they maybe deny receipt of data from a service peer and refuse payment. In our scheme, by combing the on and off-chain, we can monitor the malicious behavior of all the participants.

4.2. Adaptive security

The standard security model of SSE called adaptive security was first proposed by Curtmola et al. [9]. By using the leakage function to describe the specific leakage of the SSE scheme, it is proved that in addition to the leakage function, the SSE scheme with adaptive security does not leak any information. To be specific, a real game and an ideal game are defined as follows, where A is the attacker and S is the simulator.

$\mathrm{Real}_A^\Pi(\mathrm{u})$: The attacker will honestly execute the scheme. The attacker first runs the setup algorithm to generate the encrypted index IT and send IT to the adversary A. Adversary A then adaptively selects several queries q. The attacker runs the search token generation algorithm to generate the st_q, then inputs st_q, runs the search algorithm and returns the results to A. Finally, A outputs a bit b.

$I\mathrm{deal}_{A,S}^\Pi$: In an ideal scenario, the adversary interacts with simulator S to obtain the results of the algorithm.The adversary A sends an setup request to simulator S, and the simulator S returns the corresponding result through the leak function. Then adversary A adaptively selects several query q, and the simulator records the query $q_{[i]}$ and runs the search algorithm. Finally, S returns the results and adversary A outputs a bit b.

Definition 1. A scheme Π has L-adaptive-secure, if for any PPT attacker A, there exists a PPT simulator S, such that

$$\left|\Pr[\mathrm{Re}\,al_A^\Pi(u)=1]-\Pr[I\mathrm{deal}_{A,S}^\Pi=1]\right|\le negl(u),$$

where u is the security parameter of the scheme and negl(.) is a negligible function.

4.3. Forward update privacy

Traditional forward security does not affect the client's existing data in the cloud when updated data is compromised. However, Li et al. [19] points out that after the next retrieval, the updated data is still associated with an existing index in the CSP. Therefore, the scheme is vulnerable to statistical inference attacks, where the CSP can perform a query list to record the same query. If we can guarantee that updated data remains un-linkable to the index after a search query, it will be more difficult for CSP to recover clients' keyword by statistical inference attacks. Li et al. [19] thus proposed the definition of forward update privacy. Specifically, there are two kinds - strong forward update privacy and weak forward update privacy. Strong forward update privacy means that search protocol reveal nothing to the adversary. It is a very powerful notion, but it requires an expensive technology similar to ORAM or PIR to achieve and is not practical. For the sake of convenience, we will only elaborate on the definition of weak forward update security.

Definition 2. (Weak forward update security):Let $G(w)$ denote a set of sub keywords that corresponding to one keyword. An L-adaptive-secure DSSE scheme achieves weak forward update privacy if the search leakage function L^{Search} can be written as:

$$L^{Search}(w) = L^{'}(TimeDB(w_i)) \tag{4}$$

$TimeDB(w_i) = s_w$ where s_w is a constant.

Under this security definition, the protocol is resistant to statistical attacks and has better practicability. The scheme in this paper will be proposed with this forward security as the objective.

4.4. Backward security

There are three levels of backward security from high to low (Type-I to Type-III) [5]. Backward security limits the client's current updated data, which the CSP can use for subsequent retrieval. Specifically, a scheme has backward security if the server cannot retrieve a document that the client has deleted using the retrieval token. We will present the common used Type-II. Our scheme can also achieve Type-II backward security by encrypting identifiers of the document.

Type-II leakage : Type-II discloses the update time of the keywords and the corresponding document identifiers under subsequent updates of all keywords (but not the content). That is, all the identifies of the updated data has been encrypted.

Definition 3. An L-adaptive-secure DSSE scheme achieves Type-II backward-private iff the search and update leakage functions L^{Search} and L^{Update} can be written as:

$$L^{Search}(w) = L^{''}(TimeDB(w), Updates(w)) \tag{5}$$

$$L^{Update}(op, w, ind) = L^{'}(op, w), \tag{6}$$

where $L^{'}$ and $L^{''}$ are stateless. $Updates(w)$ denotes the update time and time of updates on w.

5. A Secure and efficient DSSE scheme combing on and off-chain

5.1. The cooperation of on and off-chain

As the security provided by the scheme increases, the smart device needs to record more data to complete the search operation. In such a scenario, the local storage overhead is also a point of concern for it. This storage overhead can be a burden due to the hardware limitations of smart devices. Previous DSSE schemes have been able to achieve efficient search under constant client storage overhead, but their ideas are not applicable in our scheme. In fact, the idea of constant client storage is bound to lead to a link between chains in some way that will be reflected after the next search operation. In this paper, we take another approach to achieve the same goal, relying on the recently popular technology-blockchain. Requester transfers local storage overhead to the service peers off the blockchain who wants to make profits by providing storage and computational resources. Requester can obtain the corresponding keyword data by requesting the service peers. The local end does not need to record any keyword data.

However, after the requester transfers local data to the service peers, the credibility of the service peers must be taken into consideration. Indeed, the service peers will return some wrong/un-updated data to the client for reducing their storage and computation overhead. Therefore, verification of the returned results is a point that must be considered. In particular, the client uploads each data digest on the chain, and then uploads the data to the service peers off the chain. A digest of the data is a collision-resistant hash that maps the uploaded data to a pseudo-random string. The clients rely on the cooperation of on-chain and off-chain to verify the returned result.

In our scheme, we argue that CSP can also be malicious because it can return a wrong data and blame it on service peers even though

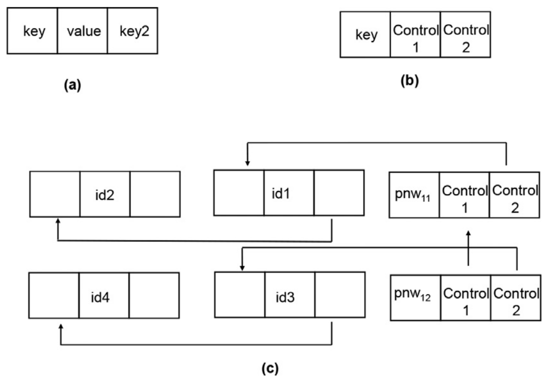


Fig. 2. Index Structure.

clients can verify the search results. Therefore, we mainly investigate the cases of all the wrong data types returned by the service peers. When we think the data returned by the service peers is correct, if there is still a problem in the final search result, it must be the CSP's problem. Here we list the possible malicious behavior of the service peers. In this way, we think we can identify all kinds of malicious behavior from CSP and from service peers and finally make our solutions more secure and sophisticated.

(i) **incorrect data:** The service peers modify some part of the data and sends it to the client.

(ii) **un-updated data:** When the requester sends the updated data corresponding to the keyword w, the service peers maybe choose not to update the data in order to saving computation and storage resources.

(iii) **the service peers performing the retrieval operation:** Since the requester sends locally stored data to the service peers, it is not desirable for us to let the peers to be able to use the data to accomplish search operations.

Besides, we consider that requesters are also malicious actors who deny receiving the results/considering they have received incorrect data and refuse to pay their service peers. Therefore, we include requesters in our monitoring scope. Specifically, we will first set up a smart contract between the smart device and the service peer. Not only should the contract contain the protocol, but also the addresses of both parties and a mutually agreed payment lock-in time. When the device sends a request to the service peer and receives the result, the smart contract activates a lock-in payment and freezes part of the requester's savings. If the requester disagrees with the result, the lock time will be suspended and subsequent checks will be made by non-peer clients. Because device and peers store metadata on the chain, all processes are open and transparent. The specific verification process is not discussed in detail in this article, but once the requester's feedback is rejected, the lock time will be turned on and the subsequent money will be automatically transferred to the peer. Through the collaboration of on and off-blockchain, the rights of all participating clients will not be infringed.

5.2. Our index structure- RC-II

To achieve efficient search efficiency, we put forward our index structure in Fig 2. It is based on a classic inverted index structure [9]. In our index, each chain corresponds to one keyword. Each chain has a control node as its head node and several data nodes. As shown in Fig 2(a), a date node has 3 fields. The key filed records a pseudo-random string and the value field stores an encrypted document identifier. The key2 field records an encrypted connected data which presents a symmetric key and a pseudo-random number for recording the next data node's key field. By using this index, we can easily connect two data nodes. The data owner only needs to give the first data node's pseudo-random number of the key field and the decryption key of the $key2$ field to the CSP, and the CSP can quickly traverse the whole chain.

However, just by a reverted index, much information can still be revealed under file injection attack [28]. The smart device's next retrieval operation will still connect the updated chain to the previously retrieved chain. To separate the update operation from the search operation, we modify the head node of each chain and call the modified head node the control node. The node has the function of retrieval control. Requester can decide whether to retrieve latest chain or all of them by giving different search token. Our control node is shown in Fig 2(b). To be specific, a control node still has 3 fields. The key field stores a pseudo-random string associated with the keyword w. The $control2$ filed has the same function as the $key2$ field in data nodes. The $control1$ field stores an encrypted connected data which presents two symmetric keys $k1$ and $k2$ and a pseudo-random number connects the key field in the previous chain's head node. $k1$ is used to decrypt the $control2$ field of this node and $k2$ is used to decrypt the $control1$ field of the previous chain's head node.

Algorithm 1 Setup.

```
Input: a security parameter $\lambda$
Output: three keys: $k_f$, $k_c$, $k_s$, empty encrtpted index EDB and $map$ for retrieval and validation
1: Smart Device:
2: $k_f, k_c, k_s \xleftarrow{\$} \{0,1\}^\lambda$
3: $EDB \leftarrow \phi$
4: $map \leftarrow \phi$
5: send $EDB$ to CSP
6: send $map$ to service peer
```

The whole index structure is shown in Fig 2(c). Suppose we update the keyword w_1 twice and get two chains. The chain w_{11} was previously retrieved. And if requester only want to search the chain w_{12}, he gives the pnw_{12} and the decryption key $k1$ of the $control2$ field to the CSP. In this way, the CSP can only traverse the chain w_{12}. And if requester wants to search all the chain of w_1, he can give the pnw_{12} and the $k2$ of the $control2$ field to the CSP. The CSP can firstly decrypt the control2 field and get the $k1$, pnw_{11} and the decryption key k of the $control2$ field in chain w_{11}. By using pnw_{12} and $k1$, the CSP can traverse chain w_{12}. By using pnw_{11} and k, the CSP can traverse chain w_{11}. The requester can control the retrieval of CSP through different symmetric keys recorded on the local side to separate the retrieval and update operations. This can significantly improve the forward update privacy of the scheme. Even if the requester retrieves the second chain, no information about the first chain is revealed.

5.3. Storage structure

The smart device needs to record three symmetric keys k_f, k_c and k_s and a map Map, where Map is the data sent to the service peers and does not need to be stored locally. k_c is used to encrypt files and identifies on the cloud, k_c is used to encrypt indexes on the cloud, and k_s is a symmetric key used to encrypt data sent to service peers. A Map is like a dictionary that stores one key field and three value fields. The key field stores pseudo-random strings containing w, and the three value fields are key field, flag field, and c field respectively. The key field store 4 keys: k, k_{ctr}, k_w and rt, which will be used to encrypt and decrypt the control nodes of index on the CSP. The flag field contains a boolean variable 'true' or 'false' which means whether the latest chain representing the keyword w has been retrieved. The c field contains ctr, $H_{ctr}(ind)$ and $H(ind)$. ctr records the number of chains corresponding w, while $H_{ctr}(ind)$ and $H(ind)$ record the data used to validate the results from CSP.

For the two service bodies off the chain, the CSP and service peers, they need to store the following data. Specifically, CSP needs to store the encrypted index and encrypted files sent by the client, as well as receive the updated data sent by the client. The service peers need to store the map sent by the requester. In our scheme, the smart device sends the index and files to the CSP, transferring the local storage overhead to the service peers. Through the two service bodies off the chain, our scheme can reduce the storage overhead of the smart device in a clever way. The smart device only needs to record three symmetric keys to complete the retrieval and update operation and verify the returned results by relying on the cooperation of on and off-chain.

5.4. Scheme details

We rely on five protocols to implement our scheme (setup, update, verification-I, search and verification-II). Our update protocol includes add and delete operations. The protocol description and pseudocode are detailed as follows. $F:(0,1)^\lambda \times (0,1)^* \to (0,1)^\lambda$ is a secure pseudo-random function (PRF). H is a $MSet - Mu - Hash$ function [4]. (Enc, Dec) can be any symmetric encryption algorithm that satisfies IND-CPA (indistinguishability against the chosen plaintext attack) with the encryption key k_f. Besides, the smart device and the service peer have identities associated with their account. For the sake of description, we denote service peer as S and consider a smart device only interacting with one service peer.

Setup. In the setup phase, the smart device randomly generates k_f, k_c and k_s as the symmetric key required by the scheme. At the same time the client initializes map and an empty encrypted database EDB. Then it sends the map to the service peer S and the EDB to the CSP.

Update. In the update phase, requester firstly choose the files that he wants to update and generates keyword-identifier pairs for these files. Then he first needs to send the search token $Uset$ of the keyword to the service peer via smart device, the service peer S uses $Uset$ to find matching data in map and return it to the client. After receiving the results, the smart device first needs to verify the correctness of the data. Specifically, it needs to do the following for each keyword

$$V_{w_i} \leftarrow H(U[key_{w_i}]) \tag{7}$$

The obtained V_{w_i} will be searched in the blockchain. When the same data is found, the data returned by S is considered legitimate. The smart device then uses $U[w]$ to generate updated chained data blocks for each keyword where op corresponds to a string representing a delete or add operation. At the same time, it updates the corresponding $U[w]$ and generates a metadata data V_W for the $U[w]$. After that, it stores the summary data into the blockchain to facilitate subsequent verification. Finally, it sends the $U[w]$ to S and the EDB to CSP to complete an update operation.

In our scheme, device uses symmetric encryption algorithm (Enc, Dec) to encrypt the identifier ind and update operation op corresponding to each keyword in $U[w]$. Therefore, when the requester performs retrieval operation, the server cannot get any information about the document except the document corresponding to the keyword, which represents that the scheme can implement type-II backward security.

$$st_w \leftarrow F((U[w_i].[c][0]), F(k_s, w) \tag{8}$$

$$st_w * \leftarrow F(k_s, w). \tag{9}$$

Verification-I. Upon receiving the data sent by smart device, the service peer S needs to generate a digest of the received data and send it to the chain. There are three reasons for this section:1.Determine whether the service peer has received the correct data (the protocol will be terminated if the digest sent by the smart device is inconsistent with the digest sent by the peer) 2. Help the device verify the validity of the received data during retrieval operation. 3. Prevents service peer from sending an un-updated $U[W]$

Search. To perform the retrieval operation, the smart device generates the following two retrieval tokens st_w and st_w* for the keyword that requester wants to retrieve.

Algorithm 2 Update.

```
Input: a set of documents DOC and the actions $op$ that need to be performed on them such as deleting or adding
Output: the updated index $EDB$ for CSP and $U_w$ for service peer
1:  Smart Device:
2:  $Uset \leftarrow \phi$
3:  for each $w_i$ do
4:      $key_{w_i} = F(k_s, w_i)$
5:      $Uset = key_{w_i} \cup Uset$
6:  end for
7:  Service peer:
8:  for each $key_{w_i}$ in $Uset$ do
9:      $U[key_{w_i}] = \text{lookup}(map, key_{w_i})$
10:     delete $(key_{w_i}, U[key_{w_i}])$ in map
11: end for
12: send all the $U[key_{w_i}]$ to device
13: Smart Device:
14: for each $key_{w_i}$ do
15:     $V_{w_i} \leftarrow H(U[key_{w_i}])$
16:     check if it is stored in blockchain, if not terminate the protocol
17: end for
18: for each $(ind, W_{ind})$ do
19:     $w_i \leftarrow F(k_s, W_{ind})$
20:     if $U[w_i] = \phi$ or $U[w_i].[flag] = true$ then
21:         if $U[w_i] = \phi$ then
22:             $U[w_i].[c][0] \leftarrow 0$
23:             $U[w_i].[c][2] \leftarrow 0$
24:         end if
25:         $ctr \leftarrow U[w_i].[c][0] + 1$
26:         $H_{ctr}(ind) \leftarrow H(ind)$
27:         $H(ind) \leftarrow U[w_i].[c][2] +_H H(ind)$
28:         $rt, k, k_1, k_w \leftarrow \{0,1\}^\lambda$
29:         $key \leftarrow F(ctr, F(k_c, W_{ind}))$
30:         if $ctr = 0$ then
31:             $v1 \leftarrow k_1$
32:         else
33:             $v1 \leftarrow (F(ctr-1, F(k_c, w))||U[w_i].[key][0]||k_1)$
34:         end if
35:         $value[0] \leftarrow H(k_w) \oplus v1$
36:         $value[1] \leftarrow H(k_1) \oplus (rt||k||Enc(ind||op))$
37:         $U[w_i].[key] \leftarrow (k_w, k_1, k, rt)$
38:         $U[w_i].[flag] \leftarrow false$
39:         $U[w_i].[c] \leftarrow (ctr, H_{ctr}(ind), H(ind))$
40:         $EDB \leftarrow (key, value) \cup EDB$
41:     else
42:         $rt, k \leftarrow \{0,1\}^\lambda$
43:         $key \leftarrow U[w_i].[key][3]$
44:         $value[0] \leftarrow Enc(ind||op)$
45:         $value[1] \leftarrow H(U[w_i].[key][2]) \oplus (rt||k)$
46:         $U[w_i].[key][1] \leftarrow k$, $U[w_i].[key][3] \leftarrow rt$
47:         $U[w_i].[c][1] \leftarrow U[w_i].[c][1] +_H H(ind)$
48:         $EDB \leftarrow (key, value) \cup EDB$
49:     end if
50: end for
```

We choose two different symmetric keys because we don't want S to be able to perform the retrieval operation without permission. After receiving st_w*, S will return the corresponding data $U[w]$ to smart device. The device still firstly verifies the correctness of the data by blockchain. Besides, device needs to check if $H(key_{w_i}||U[w_i].[c][0]+1)$ on the blockchain to prevent replay attack. When the data is considered correct and latest, requester determines the chain he wants to search and send (st_w, op, k)as the search token to CSP via smart device.The CSP uses st_w and k to find the corresponding chain and returns the value[0] field of each node except control node to the device.

Algorithm 3 Verification-I.

```
Input: $V_w$ and $V_w^*$
Output: Upon receiving the U[w] sent by the client, the service peer needs to generate a summary of the data on the chain to indicate that it has received the U[w]. The verification-I function is used to verify that two data digests are equivalent, and if they are not, the protocol is terminated.
1:  Client::
2:  for each $key_{w_i}$ in $w$ do
3:      $V_{w_i} \leftarrow H(U[key_{w_i}])$
4:      $V_{w_{i_c}} \leftarrow H(key_{w_i}||U[w_i].[c][0])$
5:  end for
6:  $V_w \leftarrow H(U(w))$
7:  send $V_w$ and all the $V_{w_i}$ to the blockchain
8:  Service peer:
9:  $V_w^* \leftarrow H(U(w))$
10: send $V_w^*$ to the blockchain
11: if $V_w^* \neq V_w$ then
12:     terminate
13: else
14:     store the $U[w]$ in map
15: end if
```

Verification-II. After receiving the encrypted result from CSP. Smart device should first decrypt the ciphertext and get the identifier corresponding keyword w. Then it needs to verify the correctness of identifiers returned from CSP. Specifically, the device uses the following formula to aggregate all identifiers into a pseudo-random string

$$Vt_w' \leftarrow H(DB(w)) \tag{10}$$

Because the smart device has stored a similar validation data Vt_w in U[w] in advance. As long as the Vt_w is consistent with the Vt_w', the returned result is regarded as right. After that, it updates the $U[w]$, generates the corresponding metadata on the chain, and sends the $U[w]$ to S to complete a retrieval operation.

6. Security analysis

In order to make the requester enjoy more convenient, we make a bold attempt. We combine CSP and service peer to achieve our search function which will certainly bring lots of security risks. We discussed in Section 5 the potential pitfalls to our solution. Next, we will verify that the security of our scheme is sufficient from two aspects.

6.1. Confidentiality

In our scheme, all the files and encrypted index are stored in CSP while the data used for search operation are stored in service peers. To Safeguard data confidentiality, the files are encrypted. Besides, considering the service peer might to perform search operations without requester's permission. We use different keys to encrypt the keyword so that the service peer cannot perform retrieval operations and guess the keywords the requester is searching for.

At the same time, considering that the CSP may attack through file injection and link the injected file with the requester's existing index after the next retrieval, we cleverly set up control nodes for each chain in the index design, and requester can select a chain or all the chains containing the keyword by themselves via smart device. The data used to generate the search token is securely stored in the service peers, and information such as whether each chain of the keyword has been retrieved is also recorded. The smart device only needs to record three symmetric keys locally to achieve secure and efficient retrieval operations.

Algorithm 4 Search.

```
Input: the keyword w and the operation(search one chain or all the chains of w) the client wants to take
Output: CSP complete search operation and returns the encrypted document identifier to requester
 1: Smart Device:
 2: $st_w * \leftarrow F(k_s, w)$
 3: send $st_w *$ to S
 4: Service peer:
 5: $U[w_i] \leftarrow lookup(map, st_w *)$
 6: delete $(st_w *, U[w_i])$ in map
 7: return $U[w_i]$ to client
 8: Client:
 9: $V(w) \leftarrow H(U[w_i])$
10: check if the $V(w)$ on the blockchain
11: check if $H(key_{w_i}||U[w_i].[c][0])$ and $H(key_{w_i}||U[w_i].[c][0]+1)$ on the blockchain
12: $op \leftarrow \}all'or\}one'$
13: $k \leftarrow U[w_i].[key][0] or U[w_i].[key][1]$
14: $st_w \leftarrow F((U[w_i].[c][0]), F(k_s, w)$
15: send $(st_w, op, k)$ to CSP
16: CSP:
17: $key \leftarrow st_w$
18: if $op =' all'$ then
19:     while $key \neq \perp$ do
20:         $value \leftarrow lookup(EDB, key)$
21:         $(key||k_w||k) \leftarrow H(k) \oplus value[0]$
22:         $st_{set} \leftarrow (key, k) \cup st_{set}$
23:         $(rt||k_1||msg) \leftarrow H(k_{w)} \cup value[1]$
24:         $result \leftarrow msg \cup result$
25:     end while
26: else
27:     $st_{set} \leftarrow (key, k)$
28: end if
29: for each $(key, k) inst_{set}$ do
30:     $value \leftarrow lookup(EDB, key)$
31:     while $value \neq \perp$ do
32:         $(rt||k_1) \leftarrow (value \oplus H(k))$
33:         $result \leftarrow value[0] \cup result$
34:         $key \leftarrow rt$
35:         $k \leftarrow k_1$
36:         $value \leftarrow lookup(EDB, key)$
37:     end while
38: end for
39: send result to Smart Device
```

Algorithm 5 Verification-II.

```
Input: the returned search results and the $U[w_i]$
Output: smart device decrypts the $Enc(ind||op)$ and gets the $DB(w_i)$, then it uses set hash to get a hash value of the results and checks if it is equal to $H(ind)$ which is stored in $U[w_i].[c][2]$
 1: Smart Device:
 2: $Vt_w{}' \leftarrow 0$
 3: for each $Enc(ind||op)$ in result do
 4:     $(ind, op) \leftarrow Dec(Enc(ind||op))$
 5:     if $OP =' add'$ then
 6:         $Vt_w{}' \leftarrow {}_H H(ind) + Vt_w{}'$
 7:     end if
 8: end for
 9: $Vt_w \leftarrow U[w_i].[c][2]$
10: if $Vt_w{}' = Vt_w$ then
11:     complete a correct search operation
12: else
13:     the result is wrong and call for appeals of the result
14: end if
```

Adaptive security. Our scheme is the first to achieve forward update privacy and backward security with constant client storage by the cooperation of on-chain and off-chain. The adaptive security of our scheme is proven in Theorem 1.

Theorem 1. *Let λ be the security parameter. Assume L' is stateless. Define L_{DSSE}=(L^{Search}_{DSSE}, L^{Update}_{DSSE}), where*

$$L^{Search}(w) = (L''(sp(w), TimeDB(w), Updates(w)),$$

$$L^{Update}(w, op, ind) = L'(w).$$

We believe that our scheme is £-adaptive-secure in the random oracle model if F is a pseudo-random function, (Enc, Dec) is secure against chosen-plaintext attack (CPA-secure) and H is a cryptographic hash function. The proof is as follows.

Proof. We derive some games from real world game to prove the theorem. Specifically, the first game Game0 is the same with the real world DSSE game, while the last game Game3 is exactly the same with the ideal world DSSE game. □

Game0. This game is the real world DSSE security game DSSEReal. Therefore, we have

$$\Pr[DSSEReal^{DSSE}_A(\lambda) = 1] = \Pr[Game_0 = 1].$$

Game1. In this game, to simulate the operation of pseudo-random functions, system maintain a table to record the results of keyword request. In Update protocol, adversary A sends a token request to the system. System chooses a random string in response and keeps the string in the table used for the next same keyword request. The ability of an adversary to distinguish Game0 from Game1 depends on the ability to distinguish pseudo-random from random functions. Hence, we have an efficient adversary $\beta 1$ such that:

$$\Pr[\text{Game0}=1]\text{-}\Pr[\text{Game1}=1] \leq Adv^{prf}_{F,\beta 1}(\lambda)$$

Game2. In this game, to generate the ciphertext of identifier, system encrypt the string of “0”s rather than use the identifier. The others are the same as in Game1. The advantage of distinguishing Game1 and Game2 is considered to be the advantage of breaking the (Enc,Dec) scheme. Hence, we have an adversary $\beta 2$ such that:

$$\Pr[\text{Game1}=1]\text{-}\Pr[\text{Game2}=1] \leq Adv^{Enc}_{\beta 2}(\lambda)$$

Game3. In this game, system picks random strings in update protocol instead of using hash function to generate a data for validation. As long as the hash function we apply is safe which means it is pseudo random, the advantage of the opponent distinguishing between the two games can be ignored. Thus, we can build a adversary $\beta 3$ such that

$$\Pr[\text{Game2}=1]\text{-}\Pr[\text{Game3}=1] \leq Adv^{hash}_{H,\beta 3}(\lambda)$$

Simulator. Game3 and DSSEIdeal are identical. In update protocol, the challenge pick random strings instead of hash and F. Besides, the challenge can replace the identifier with the encryption of “0”s in the EDB. In search protocol, the challenger can answer the search query using the Hash table and Key table.

Conclusion. Combining all games, if hash function H and F is pseudo random and (Enc, Dec) is IND-CPA secure, our scheme can achieve adaptive security such that

$$\Pr[DSSEReal_A(\lambda) = 1] - \Pr[DSSEIdeal_A(\lambda) = 1] \leq Adv^{prf}_{F,\beta 1}(\lambda) + Adv^{Enc}_{\beta 2}(\lambda) + Adv^{hash}_{H,\beta 3}(\lambda)$$

Our protocol satisfies adaptive security, that is, our protocol does not disclose any information except the leakage function.

Table 2
Database Sizes.

	DB1	DB2	DB3
n	537	2024	8539
m	3025	9066	40,342
N	7712	35,466	199,244

6.2. Forward update privacy and backward privacy

In our index structure, we design the search control node to achieve user control of retrieval operations. In search phase, the user can choose to retrieve only the most recently updated chain, and the CSP cannot relate it to other index data, which guarantees weak forward update privacy. For backward security, we use a symmetric encryption algorithm that has proven to be secure. By symmetrically encrypting the identifier information during the update phase, we ensure that the index will not reveal any information even if it is deleted.

Obviously, due to Theorems 1, Definition 2 and Definition 3, we can come to the following conclusion:

Corollary 1. *Our scheme satisfies weak forward update privacy and backward privacy.*

6.3. Correctness

In order to monitor the malicious behavior of the three entities in this system, we use the set hash in our protocol. Specifically, malicious behavior detection from device and service peers relies on the metadata they upload on the chain, while the monitoring of malicious behavior in CSP depends on the validation data stored in the service peer. We hold the opinion that if the set hash we use is secure enough, the probability that an adversary returns an error and passes validation is negligible. Our scheme can be performed correctly, and even if there is malicious behavior, we can detect it and take appropriate measures to punish them.

7. Results and discussion

In order to demonstrate the applicability of this scheme to smart devices, we evaluate our scheme and show the advantages of this scheme compared with other schemes. The experiments are performed in Python 3. We use HMAC to realize the pseudo-random function. The set hash function was implemented in SHA-256. The symmetric encryption scheme we used for encrypting identifier is AES. The computer is Intel(R) Core (TM) i5-10210U and 8GB of RAM that running on windows 11. We will prove the reliability of this scheme applied to smart devices from storage cost, computation cost and communication cost respectively.

7.1. Dataset

Enron email dataset is used to help us complete our experiments. We extract 517,401 plain-text files and its size is 1.32 GB. We use RAKE to extract keywords. To better verify the performance of this protocol on smart devices, we divide our dataset into 3 sets. The detailed information is displayed in Table 2. Among them, n represents the number of documents, m is on behalf of the number of keywords extracted from the documents and N is the number of keyword-document pairs.

7.2. Storage cost

The Local storage resource usage of the protocol is an important indicator of its success in smart devices. We give the local storage overhead of ours as well as others.

Table 3
Local Storage Costs in different database.

	DB1	DB2	DB3
Bost et al. [5]	203KB	610KB	2,711KB
Li et al. [19]	1,071KB	3,207KB	14,271KB
He et al. [13]	3KB	3KB	3KB
Ours	3KB	3KB	3KB

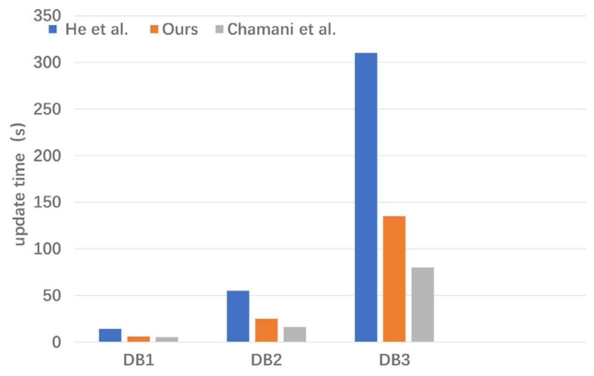


Fig. 3. Comparison of different schemes in the update process of different databases.

As shown in Table 3, local storage overhead of Bost et al. [5] and Li et al. [19] increases linearly as the number of keywords increases. It is worth noting that although the scheme of Li et al. [19] can achieve forward search privacy, their local storage overhead is affected by the threshold P which means the maximum number of blocks in the chain as well as the number of sub-keywords. In our experiment, we set the number of sub-keywords to 1 which means that their local storage overhead will be even heavier in practical. He et al. [13] proposed a fish-bone structure which decreases the local storage costs to a constant. But their schemes suffer from the maximum number of updates. Besides, client need to provide more computing resources under their protocol(which will be explained in Section 8.3).

It is important to note that although the table shows that the storage overhead for each scenario is still acceptable, this is only for small data sets. When the data of requester is larger (i.e. wikimedia with 50GB data per day), the storage overhead is obviously not negligible. And finding the information that corresponds to keywords in this scale of local data is a challenge for smart devices with low computing resources.

7.3. Computation cost

Update and search time is a measure of the usefulness of DSSE. In this section, we discuss the computation costs of ours with He et al.'s scheme [13] and Ghareh et al.'s scheme [11] to verify the performance of our protocol in search and update operation.

Figure 3 shows the update time on 3 different databases. Specifically, we choose $clen$ for 2000 which means He et al.'s protocol can support at most 2000 times of updates. After that, requester needs to download all the files from CSP and re-index these encrypted files. From the figure, it is obviously that the update time increases linearly as the number of keyword file pairs increases. The update time of He et al.'s is far more than the other two because theirs needs to perform at most $clen$ times of hash to complete the update of one keyword-document pair. Ghareh et al.'s scheme shows better performance than us because theirs require less computation to calculate each keyword-document pair. In order to achieve forward update privacy, we design a retrieval control node in our scheme, resulting in an increase in the amount of computation. Overall, our scheme achieves a better tradeoff in terms of security and performance during the update phase.

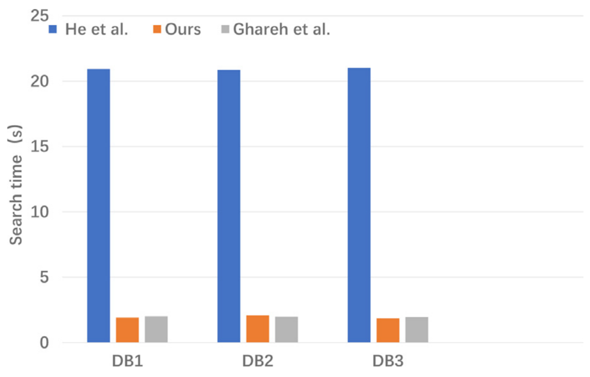


Fig. 4. Comparison of completing 10,000 times search operation of different schemes in different databases.

Table 4
Communication costs in different databases.

	DB1	DB2	DB3
blockchain	195KB	585KB	2,601KB
CSP	786KB	3,220KB	17,866KB
service peer	1,159KB	3,471KB	15,444KB
total	2,140KB	7,276KB	41,511KB

Since the retrieval time of the three schemes is very small under the three databases, in order to measure the difference of retrieval time cost more intuitively, we perform 10,000 retrieval operations from the dataset, each of which is performed by selecting a keyword randomly from the keyword list. As shown in Fig 4, the size of the database has almost negligible effect on the retrieval time. Our scheme shows good retrieval performance as well as Ghareh et al.'s because the client only needs to perform one hash operation for each retrieval operation. In He et al.'s scheme, smart device needs to calculate clen times (usually not small) of hash to complete one search operation which creates a large computing burden for smart devices.

7.4. Communication costs

On smart devices with limited resources, the performance of transferring a large amount of data at one time is poor. In terms of communication cost, our scheme also shows good performance.

Table 4 gives the communication costs of clients updating the database to different objects. It is important to note that our communication overhead here does not include the encrypted files, because in practice the size of encrypted files are incalculable. Firstly, smart device should transfer data for validation to the on-chain node. The on-chain nodes are more precious, so we choose to transmit a summary of the data, which is still an acceptable bandwidth cost even with a large single update of DB3. Encrypted index is transferred to the CSP. It is a data volume that increases linearly as the number of keywords increases. Under smart devices, users can control the number of files updated each time to reduce the data volume. According to our test, when the number of files updated by users is less than 800, the communication cost of this part is less than 1MB. Besides, this is a once and for all job which means that the client can enjoy continuous retrieval operations as long as uploads the index for one time. As for the service peer, we use it to hold some data for retrieval. Unlike indexes on the cloud, this portion of overhead is incurred each time a user updates a file. Although it increases linearly with the number of keywords, the data in the table shows that this overhead can still be applied to smart devices when users perform updates with small databases.

In order to further analyze the impact of file quantity on communication overhead, we give Table 5. There are two things we can learn from this table. The first one is that the communication overhead per

Table 5
Communication costs of updating one file.

	DB1	DB2	DB3
blockchain	0.36KB	0.29KB	0.30KB
CSP	1.46KB	1.59KB	2.09KB
service peer	2.16KB	1.71KB	1.81KB
total	3.98KB	3.59KB	4.2KB

file does not increase with the size of the database. Another one is that whether updating a file's index or generating a summary of its data, its communication overhead is negligible. The above two characteristics prove the credibility of the proposed scheme for smart devices.

8. Conclusion

We design a secure and efficient keyword search scheme in smart devices in this paper. Our protocol demonstrated good performance by storing encrypted files and index in the CSP and transferring complex local storage to off-chain service peers. At the same time, in order to detect the malicious behavior of service peers, CSP and requsters, we store the data summary to the on-chain node to realize the stable operation of the protocol. The smart device does not need to record any data related to keywords locally, and the retrieval operation has low computation and high efficiency. All these provide a strong guarantee for the implementation of the scheme on smart devices.

However, the following problems still exist in this scheme. Firstly, this scheme requires multiple rounds of communication in both retrieval and update phases. Although in the performance analysis section we verified that this overhead is acceptable, the challenge remains to reduce the number of communication rounds without compromising security and local storage overhead. In addition, this scheme is only applicable to single user, single keyword retrieval scenarios. An open and challenging problem is to design securer and more efficient DSSE with properties of multi-user and multi-keyword setting.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Weiwei Yan: Conceptualization, Methodology, Software. **Sai Ji:** Data curation, Writing – original draft.

References

[1] M. Ali, J. Nelson, R. Shea, M.J. Freedman, Blockstack: a global naming and storage system secured by blockchains, in: 2016 {USENIX} Annual Technical Conference ({USENIX}{ATC} 16), 2016, pp. 181–194.
[2] D. Boneh, G. Di Crescenzo, R. Ostrovsky, G. Persiano, Public key encryption with keyword search, in: International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2004, pp. 506–522.
[3] R. Bost, Forward secure searchable encryption, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 1143–1154.
[4] R. Bost, P.-A. Fouque, D. Pointcheval, Verifiable dynamic symmetric searchable encryption: optimality and forward security, IACR Cryptol. ePrint Arch. 2016 (2016) 62.
[5] R. Bost, B. Minaud, O. Ohrimenko, Forward and backward private searchable encryption from constrained cryptographic primitives, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 1465–1482.
[6] C. Cai, J. Weng, X. Yuan, C. Wang, Enabling reliable keyword search in encrypted decentralized storage with fairness, IEEE Trans. Dependable Secure. Comput. (2018).
[7] D. Cash, P. Grubbs, J. Perry, T. Ristenpart, Leakage-abuse attacks against searchable encryption, in: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, 2015, pp. 668–679.

[8] D. Clarke, S. Devadas, M. Van Dijk, B. Gassend, G.E. Suh, Incremental multiset hash functions and their application to memory integrity checking, in: International Conference on the Theory and Application of Cryptology and Information Security, Springer, 2003, pp. 188–207.

[9] R. Curtmola, J. Garay, S. Kamara, R. Ostrovsky, Searchable symmetric encryption: improved definitions and efficient constructions, J. Comput. Secur. 19 (5) (2011) 895–934.

[10] X. Ge, J. Yu, H. Zhang, C. Hu, Z. Li, Z. Qin, R. Hao, Towards achieving keyword search over dynamic encrypted cloud data with symmetric-key based verification, IEEE Trans. Dependable Secure. Comput. (2019).

[11] J. Ghareh Chamani, D. Papadopoulos, C. Papamanthou, R. Jalili, New constructions for forward and backward private symmetric searchable encryption, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018, pp. 1038–1055.

[12] P. Golle, I. Mironov, Uncheatable distributed computations, in: Cryptographers' Track at the RSA Conference, Springer, 2001, pp. 425–440.

[13] K. He, J. Chen, Q. Zhou, R. Du, Y. Xiang, Secure dynamic searchable symmetric encryption with constant client storage cost, IEEE Trans. Inf. Forensics Secur. 16 (2020) 1538–1549.

[14] M.S. Islam, M. Kuzu, M. Kantarcioglu, Access pattern disclosure on searchable encryption: ramification, attack and mitigation, in: Ndss, vol. 20, Citeseer, 2012, p. 12.

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

[16] S. Kamara, C. Papamanthou, T. Roeder, Dynamic searchable symmetric encryption, in: Proceedings of the 2012 ACM Conference on Computer and Communications Security, 2012, pp. 965–976.

[17] S.K. Kermanshahi, S.-F. Sun, J.K. Liu, R. Steinfeld, S. Nepal, W.F. Lau, M. Au, Geometric range search on encrypted data with forward/backward security, IEEE Trans. Dependable Secure. Comput. (2020).

[18] K.S. Kim, M. Kim, D. Lee, J.H. Park, W.-H. Kim, Forward secure dynamic searchable symmetric encryption with efficient updates, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 1449–1463.

[19] J. Li, Y. Huang, Y. Wei, S. Lv, Z. Liu, C. Dong, W. Lou, Searchable symmetric encryption with forward search privacy, IEEE Trans. Dependable Secure. Comput. (2019).

[20] Y. Liu, J. Yu, M. Yang, W. Hou, H. Wang, Towards fully verifiable forward secure privacy preserving keyword search for IoT outsourced data, Future Gener. Comput. Syst. 128 (2022) 178–191.

[21] Z. Liu, Y. Huang, X. Song, B. Li, J. Li, Y. Yuan, C. Dong, Eurus: towards an efficient searchable symmetric encryption with size pattern protection, IEEE Trans. Dependable Secure Comput. (2020).

[22] Y. Miao, J. Ma, Q. Jiang, X. Li, A.K. Sangaiah, Verifiable keyword search over encrypted cloud data in smart city, Comput. Electr. Eng. 65 (2018) 90–101.

[23] B.K. Mohanta, S.S. Panda, D. Jena, An overview of smart contract and use cases in blockchain technology, in: 2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT), IEEE, 2018, pp. 1–4.

[24] M. Naveed, The fallacy of composition of oblivious ram and searchable encryption, IACR Cryptol. ePrint Arch. 2015 (2015) 668.

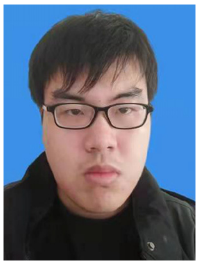
[25] M. Naveed, S. Kamara, C.V. Wright, Inference attacks on property-preserving encrypted databases, in: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, 2015, pp. 644–655.

[26] D.X. Song, D. Wagner, A. Perrig, Practical techniques for searches on encrypted data, in: Proceeding 2000 IEEE Symposium on Security and Privacy: S&P 2000, IEEE, 2000, pp. 44–55.

[27] E. Stefanov, C. Papamanthou, E. Shi, Practical dynamic searchable encryption with small leakage, in: NDSS, vol. 71, 2014, pp. 72–75.

[28] Y. Zhang, J. Katz, C. Papamanthou, All your queries are belong to us: the power of file-injection attacks on searchable encryption, in: 25th {USENIX} Security Symposium ({USENIX} Security 16), 2016, pp. 707–720.

[29] C. Zuo, S. Sun, J.K. Liu, J. Shao, J. Pieprzyk, L. Xu, Forward and backward private DSSE for range queries, IEEE Trans. Dependable Secure. Comput. (2020).



Weiwei Yan is now working towards his master degree in Nanjing University of Information Science & Technology (NUIST), Nanjing, China, in 2022. His research interets are in the areas of applied cryptography, cloud computing, keyword search.

Sai Ji received his BS degree from the Nanjing University of Information Science & Technology (NUIST), Nanjing, China, in 1999, and his MS degree from the Nanjing Aeronautics and Astronautics University (NUAA), Nanjing, China, in 2006. He works as an Associate Professor at the NUIST. His research interests are in the areas of computer measurement and control, structural health monitoring, and WSNs. He has published more than 20 journal/conference papers. He is a Principle Investigator of three NSF projects.