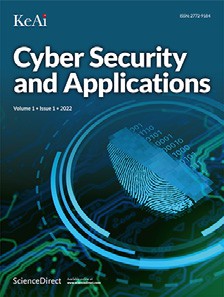
[Cyber Security and Applications 1 (2023) 100006](https://doi.org/10.1016/j.csa.2022.100006)

Contents lists available at [ScienceDirect](http://www.ScienceDirect.com/)

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[☆](#_bookmark2)

Weiwei Yan[a](#_bookmark0),[∗](#_bookmark3), Sai Ji [a](#_bookmark0),[b](#_bookmark1)

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 a b s t r a c t

*Keywords:*

DSSE

Smart devices Storage cost Forward security

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.

# 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]](#_bookmark44). However, sensitive information is easy to leak, and data outsourced to cloud may bring search leak- age. Besides, smart devices are often unable to perform a large number of complex operations due to limited hardware resources. Searchable symmetric encryption (SSE) [[26]](#_bookmark32) 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 down- loading all the ciphertext from the cloud service platform (CSP) [[24]](#_bookmark30). Compared with Public key Encryption with Keyword Search (PEKS) [[2]](#_bookmark22), 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 oper- ation. Dynamic searchable symmetric encryption (DSSE) appears [[3]](#_bookmark23). In recent years, attacks against DSSE have become more frequent [[7,14]](#_bookmark24). Stefanov et al. [[27]](#_bookmark34) defines the forward security that the leaked infor-

mation under update operation would corrupts the existing index in the CSP. Although many scholars have proposed schemes with forward se- curity, 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 sta- tistical inference attack can still recover the keywords [[19]](#_bookmark43). The strong forward update privacy and poor forward update privacy, defined by Li et al. [[19]](#_bookmark43), can resist statistical inference attack. Their scheme is based on poor forward update privacy and achieve eﬃcient deletion opera- tion, 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]](#_bookmark43). Although the ex- isting 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 over- head 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.](https://doi.org/10.13039/501100004608) [U1836115](#_bookmark2)[, No.](https://doi.org/10.13039/501100004608) [61922045](#_bookmark2)[, No.](https://doi.org/10.13039/501100004608) [61877034](#_bookmark2)[, the Natural Science Foundation of Jiangsu Province under Grant No.](https://doi.org/10.13039/501100004608) [BK20181408](#_bookmark2)[, the CICAEET fund, and the PAPD fund.](https://doi.org/10.13039/501100004608)

∗ Corresponding author.

*E-mail addresses:* [20201221055@nuist.edu.cn](mailto:20201221055@nuist.edu.cn), [yanyunyanwei9140@163.com](mailto:yanyunyanwei9140@163.com) (W. Yan), [jisai@nuist.edu.cn](mailto: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

2772-9184/© 2022 The Authors. Published by Elsevier B.V. on behalf of KeAi Communications Co., Ltd. This is an open access article under the CC BY-NC-ND license (<http://creativecommons.org/licenses/by-nc-nd/4.0/>)

**Table 1**

Comparison with Existing DSSE Schemes.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Bost [[3]](#_bookmark23)  Bost et al. [[5]](#_bookmark25) Kim et al. [[18]](#_bookmark42) | O(*𝑏𝑤* )  O((*𝑏𝑤* - *𝑑𝑤* )  O(*𝑏𝑤* ) | O(1)  O(1)  O(1) | ×  ×  × | ×  √  √ | ×  ×  × | O(m) O(m))  O(m)) |
| Cai et al. [[6]](#_bookmark26) | O(*𝑚*+2*𝑘𝑡* )/O(1+2*𝑘𝑡* ) | O(1+2*𝑘𝑡* ) | × | × | × | O(*𝑚*) |

search computation update computation F.UP B.P verifiable client storage

He et al. [[13]](#_bookmark38) O(*𝑏𝑤* + *𝐶𝑙𝑒𝑛*) O(Clen) × √ × O(1)

Li et al. [[19]](#_bookmark43) O(*𝑏𝑤* ) O(1) √ √ × O(*𝑚* + *𝑛*) Ge et al. [[10]](#_bookmark33) O(*𝑛*) O(*𝑚*) × × √ O(*𝑚* + *𝑛*) Ours O(*𝑏𝑤* ) O(1) √ √ √ O(1)

of keywords while n is the number of documents. *𝑏𝑤* is the number of entries that matching keyword F.UP represents forward update privacy and B.P represents backward privacy. m represents the number

w. *𝑑𝑤* is the number of deleted entries matching w. Under Cai et al.’s [[6]](#_bookmark26)scheme, based on whether

complexity is O(*𝑁* +2*𝑘𝑡* ) and O(1+2*𝑘𝑡* ) respectively. *𝑘𝑡* Represents the cost of conducting a transaction. the keywords retrieved have been searched before, Cai et al. [[6]](#_bookmark26) is executed in two cases, computation

*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 secu- rity with the constant client storage. The reason for this is that clients cannot transfer their locally storage to the CSP, which could bring se- rious leak problems. To deal with this question, we propose a clever solution. We adopt currently hot technology-blockchain [[1]](#_bookmark27). 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 ser- vice 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 eﬃ- cient retrieval with very low local storage overhead, as well as detect the malicious behavior of CSP and service peers.

* 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 re- sist replay attack from CSP and from service peers at the same time. Besides, each entry who joins our protocol can monitor the illegal be- havior 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](#_bookmark4). The list given below details our

contributions.

* 1. *Our contributions*
     + **We propose a novel index structure called RC-II which achieves search control.** Traditional DSSE schemes only allow the CSP to re- trieve 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 up- date 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 guar- antees 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](#_bookmark5) overviews some related works and [Section 3](#_bookmark6) explains some background information. Sys- tem model and threat model are given in [Section 4](#_bookmark9). [Section 5](#_bookmark13) 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](#_bookmark14) and [7](#_bookmark18), we give the security analysis and conduct experiments to test the eﬃciency of our design. Finally, we elaborate the conclusions in [Section 8](#_bookmark19).

# Related work

In 2000, SSE was first proposed by Song et al. [[26]](#_bookmark32) as a new type of cryptographic primitive. This scheme realizes the function of key- word 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 eﬃ- ciency was proposed by Kamara et al. [[16]](#_bookmark40). Clients update encrypted in- dex 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]](#_bookmark39) successfully solved these problems. Cai et al. [[6]](#_bookmark26) 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 key- word. Zuo et al. [[29]](#_bookmark38) added the function of range retrieval to the scheme, which greatly improved the eﬃciency of ciphertext retrieval while en- suring 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]](#_bookmark33) 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 at- tack [[28]](#_bookmark36). Therefore, DSSE needs to separate the existing index from the current updated data. Stefanov et al. [[27]](#_bookmark34) 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]](#_bookmark23) 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]](#_bookmark34) first mentioned the concept of backward security, but did not gave a specific definition. Bost et al. [[5]](#_bookmark25) 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]](#_bookmark41) proposed two DSSE schemes that sup- port geometric range retrieval, of which the first supports forward secu- rity and the second supports backward security. However, both of them need homomorphic encryption which has poor retrieval performance. Liu et al. [[21]](#_bookmark45) 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) technol- ogy, which is relatively low in practicability, and the high local storage will also bring a lot of overhead. He et al. [[13]](#_bookmark38) proposed a new chain- type storage structure called fishbone chain which greatly reduces the local storage overhead. But this scheme suffers from the maximum num- ber 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]](#_bookmark43) which will cause the scheme to be vulnerable to statistical inference attack [[7,14,25]](#_bookmark24) 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 guar- antee the security of DSSE. Meanwhile, as the client transfers files and index to the CSP, the client’s local storage overhead increases linearly.

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* → *𝑘***)**: User inputs a security parameter *𝑢* and outputs

a symmetric key *𝑘*.

1. **Enc(***𝑖𝑡*, *𝑘* → *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.
2. **SrchTokenGen(***𝑤*, *𝑘* → *𝑠𝑡𝑤* **)**: The user first selects the keyword w that he wants to search, and then generates a search token *𝑠𝑡𝑤* with the keyword *𝑤* and the symmetric key *𝐾*, and then sends the stw to the CSP.
3. **Search(***𝐼𝑇* , *𝑠𝑡𝑤* → *𝐼𝐷*(*𝑤*)**)**: The algorithm is executed by a CSP. Af-

ter receiving the user’s search token *𝑠𝑡𝑤* , 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.

1. **UpTokenGen(***𝑤*, *𝑘*, *𝑖𝑑* → *𝑢𝑡𝑤* **)**: The algorithm is executed by the

data owner who generates the update token *𝑢𝑡𝑤* by inputing the identi- fier of the files which he wants to update and then sent the files and *𝑢𝑡𝑤*

to the CSP.

1. **Update(***𝑢𝑡𝑤* → *𝐼𝑇* **)**: After receiving the *𝑢𝑡𝑤* , the CSP update the index IT by the *𝑢𝑡𝑤* .

*3.2. Blockchain*

Blockchain originated with Bitcoin [[12]](#_bookmark37). In essence, blockchain is a shared, immutable distributed ledger technology that facilitates the pro- cess of recording transactions and tracking assets across business net- works. 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]](#_bookmark46). 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 se- curity 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*

Among the set hash, *𝑀𝑆𝑒𝑡* − *𝑀𝑢* − *𝐻𝑎𝑠ℎ* is collision-resistant. A hash (Multi) set hash function maps the data of a set to a string [[8]](#_bookmark29).

(*𝐻 ,* ≡ *𝐻 ,* +*𝐻,* −*𝐻* ) if for all *𝑆 ⊂* 𝕊, the hash function meet the following function is defined as quadruple of probabilistic polynomial algorithms

conditions.

This puts a lot of storage burden on smart devices. Therefore, how to reduce the storage cost of DSSE must be considered.

*𝐻* (*𝑆*) ≡

*𝐻 𝐻* (*𝑆*) (1)

# Preliminaries

∀x ∈ 𝕊∖*𝑆, 𝐻* (*𝑆* ∪ {x} ≡ *𝐻 𝐻𝑆* + *𝐻 𝐻* {x} (2)

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

∀x ∈ *𝑆 , 𝐻* (*𝑆* ∖{x} ≡

*𝐻 𝐻 𝑆* −

*𝐻 𝐻* {x} (3)

* 1. *DSSE*

DSSE generally includes 7 algorithms (Setup, KeyGen, Enc, SrchTo- kenGen, 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 con- taining a keyword, the owner sends a search token containing the key- word 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

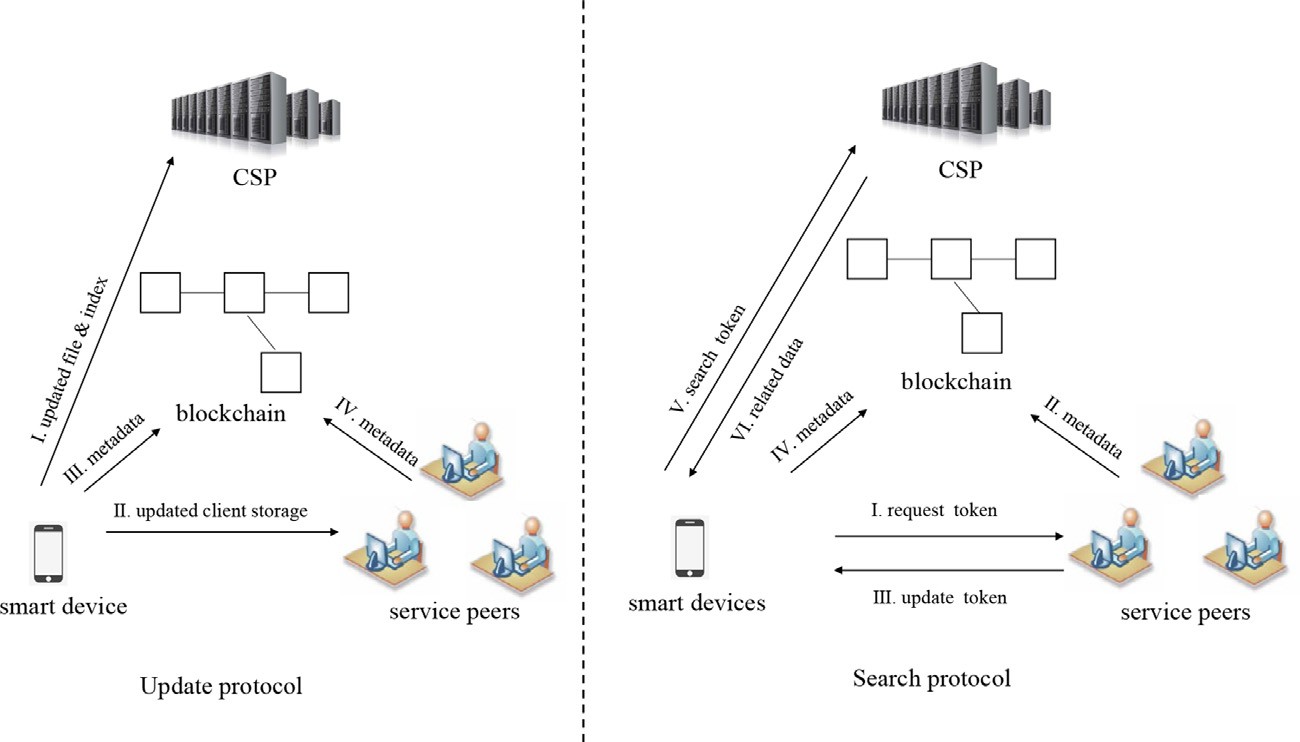
the operation of adding data *𝑥* to the set, and [(3)](#_bookmark8) present the operation Equations [(2)](#_bookmark7) and [(3)](#_bookmark8) present two operations for the set hash, [(2)](#_bookmark7) present

of deleting data x to the set. In this paper, we apply set hash function to data validation.

# System model and threat model

* 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 eﬃ- cient search service. CSP provides storage and retrieval services. It needs to receive data sent from clients, update index, provide retrieval ser- vices and return the corresponding results after receiving a search to- ken from clients. Service peers are individual nodes who receive finan- cial 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 trans- actions 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 auto- matically transfer the requester’s money to the service peer after he con- firms the result. These are advantages that trust third party (TTP) cannot provide.

The system flow is shown in the [Fig 1](#_bookmark10). 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 gen- erates 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

* 1. *Adaptive security*

The standard security model of SSE called adaptive security was first proposed by Curtmola et al. [[9]](#_bookmark31). By using the leakage function to de- scribe the specific leakage of the SSE scheme, it is proved that in addi- tion to the leakage function, the SSE scheme with adaptive security does

are defined as follows, where *𝐴* is the attacker and *𝑆* is the simulator. not leak any information. To be specific, a real game and an ideal game RealΠ(u): The attacker will honestly execute the scheme. The at-

*𝐴*

tacker 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 algo- rithm to generate the *𝑠𝑡𝑞* , then inputs *𝑠𝑡𝑞* , runs the search algorithm and returns the results to *𝐴*. Finally, *𝐴* outputs a bit *𝑏*.

*𝐼* dealΠ : In an ideal scenario, the adversary interacts with simulator

S to obtain the results of the algorithm.The adversary *𝐴* sends an setup

*𝐴,𝑆*

result through the leak function. Then adversary *𝐴* adaptively selects request to simulator S, and the simulator S returns the corresponding 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

|Pr[Re *𝑎𝑙*Π(*𝑢*) = 1] − Pr[*𝐼* dealΠ = 1]| ≤ *𝑛𝑒𝑔𝑙*(*𝑢*)*,*

some of the requester’s assets. He can verify the returned data within a | *𝐴*

*𝐴,𝑆* |

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 ver- ify 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 sav- ing computation and storage overhead and blame the error to the ser- vice 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 knowl- edge. 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.

where *𝑢* is the security parameter of the scheme and negl(.) is a negligi-

ble function.

* 1. *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]](#_bookmark43) 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 infer- ence attacks. Li et al. [[19]](#_bookmark43) thus proposed the definition of forward up- date privacy. Specifically, there are two kinds - strong forward update privacy and weak forward update privacy. Strong forward update pri- vacy 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 conve- nience, 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-

leakage function *𝐿𝑆𝑒𝑎𝑟𝑐ℎ* can be written as: secure DSSE scheme achieves weak forward update privacy if the search

*𝐿𝑆𝑒𝑎𝑟𝑐ℎ*(*𝑤*) = *𝐿*′ (*𝑇 𝑖𝑚𝑒𝐷𝐵*(*𝑤* )) (4)

*𝑖*

*𝑇 𝑖𝑚𝑒𝐷𝐵*(*𝑤𝑖* ) = *𝑠𝑤* where *𝑠𝑤* 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.

* 1. *Backward security*

There are three levels of backward security from high to low (Type-I to Type-III) [[5]](#_bookmark25). 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.

backward-private iff the search and update leakage functions *𝐿𝑆𝑒𝑎𝑟𝑐ℎ* **Definition 3.** An L-adaptive-secure DSSE scheme achieves Type-II and *𝐿𝑈𝑝𝑑𝑎𝑡𝑒* can be written as:

*𝐿𝑆𝑒𝑎𝑟𝑐ℎ*(*𝑤*) = *𝐿*′′ (*𝑇 𝑖𝑚𝑒𝐷𝐵*(*𝑤*)*, 𝑈𝑝𝑑𝑎𝑡𝑒𝑠*(*𝑤*)) (5)

*𝐿𝑈𝑝𝑑𝑎𝑡𝑒*(*𝑜𝑝, 𝑤, 𝑖𝑛𝑑*) = *𝐿*′ (*𝑜𝑝, 𝑤*)*,* (6)

where *𝐿*′ and *𝐿*′′ are stateless. *𝑈𝑝𝑑𝑎𝑡𝑒𝑠*(*𝑤*) denotes the update time and time of updates on *𝑤*.

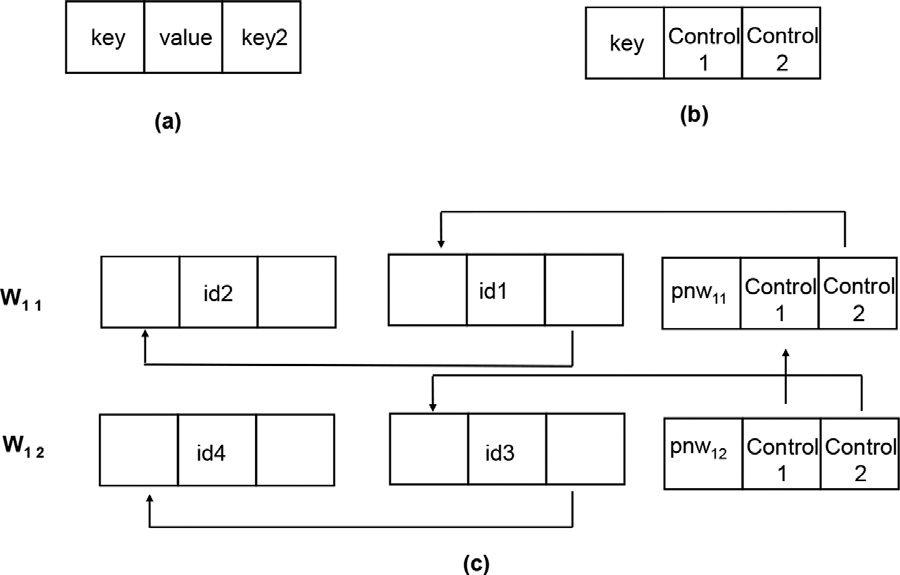
# A Secure and efficient DSSE scheme combing on and off-chain

* 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 eﬃ- cient search under constant client storage overhead, but their ideas are not applicable in our scheme. In fact, the idea of constant client stor- age 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 an- other approach to achieve the same goal, relying on the recently popu- lar technology-blockchain. Requester transfers local storage overhead to the service peers off the blockchain who wants to make profits by pro- viding 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.

corresponding to the keyword *𝑤*, the service peers maybe choose not to (ii) **un-updated data:** When the requester sends the updated data

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 con- tract 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 ac- tivates 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 de- vice 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 trans- ferred to the peer. Through the collaboration of on and off-blockchain, the rights of all participating clients will not be infringed.

* 1. *Our index structure- RC-II*

To achieve eﬃcient search eﬃciency, we put forward our index structure in [Fig 2](#_bookmark11). It is based on a classic inverted index structure [[9]](#_bookmark31). In our index, each chain corresponds to one keyword. Each chain has a con- trol node as its head node and several data nodes. As shown in [Fig 2](#_bookmark11)(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 *𝑘𝑒𝑦*2 field to the

CSP, and the CSP can quickly traverse the whole chain.

However, just by a reverted index, much information can still be re- vealed under file injection attack [[28]](#_bookmark36). 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. Re- quester can decide whether to retrieve latest chain or all of them by giving different search token. Our control node is shown in [Fig 2](#_bookmark11)(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 *𝑐𝑜𝑛𝑡𝑟𝑜𝑙*2 filed

has the same function as the *𝑘𝑒𝑦*2 field in data nodes. The *𝑐𝑜𝑛𝑡𝑟𝑜𝑙*1 field

stores an encrypted connected data which presents two symmetric keys

*𝑘*1 and *𝑘*2 and a pseudo-random number connects the key field in the previous chain’s head node. *𝑘*1 is used to decrypt the *𝑐𝑜𝑛𝑡𝑟𝑜𝑙*2 field of this node and *𝑘*2 is used to decrypt the *𝑐𝑜𝑛𝑡𝑟𝑜𝑙*1 field of the previous

chain’s head node.

**Algorithm 1** Setup.

**Input:** a security parameter *𝜆*

**Output:** three keys: *𝑘𝑓* , *𝑘𝑐* , *𝑘𝑠* , empty encrtpted index EDB and *𝑚𝑎𝑝* for

retrieval and validation

1: **Smart Device**:

2: *𝑘* , *𝑘* , *𝑘* $ {0*,* 1}*𝜆*

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

detailed as follows. *𝐹* :(0*,* 1)*𝜆* × (0*,* 1)∗ → (0*,* 1)*𝜆* is a secure pseudo-random add and delete operations. The protocol description and pseudocode are function (PRF). *𝐻* is a *𝑀𝑆𝑒𝑡* − *𝑀𝑢* − *𝐻𝑎𝑠ℎ* function [[4]](#_bookmark28). (Enc, Dec) can

be any symmetric encryption algorithm that satisfies IND-CPA (indis- tinguishability against the chosen plaintext attack) with the encryption

key *𝑘𝑓* . Besides, the smart device and the service peer have identities

service peer as *𝑆* and consider a smart device only interacting with one associated with their account. For the sake of description, we denote

service peer.

***Setup.*** In the setup phase, the smart device randomly generates *𝑘𝑓* ,

*𝑘* and *𝑘* as the symmetric key required by the scheme. At the same

*𝑓 𝑐*

*𝑠* ←←←

*𝑐 𝑠*

3: *𝐸𝐷𝐵* ← *𝜙*

4: *𝑚𝑎𝑝* ← *𝜙*

5: send *𝐸𝐷𝐵* to CSP

6: send *𝑚𝑎𝑝* to service peer

the keyword *𝑤*1 twice and get two chains. The chain *𝑤*11 was previously The whole index structure is shown in [Fig 2](#_bookmark11)(c). Suppose we update retrieved. And if requester only want to search the chain *𝑤*12, he gives the *𝑝𝑛𝑤*12 and the decryption key *𝑘*1 of the *𝑐𝑜𝑛𝑡𝑟𝑜𝑙*2 field to the CSP. In this way, the CSP can only traverse the chain *𝑤*12. And if requester

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

Then he first needs to send the search token *𝑈𝑠𝑒𝑡* of the keyword to wants to update and generates keyword-identifier pairs for these files. 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

*𝑉𝑤* ← *𝐻* (*𝑈* [*𝑘𝑒𝑦𝑤* ]) (7)

wants to search all the chain of *𝑤*1 , he can give the *𝑝𝑛𝑤*12 and the *𝑘*2 of the *𝑐𝑜𝑛𝑡𝑟𝑜𝑙*2 field to the CSP. The CSP can firstly decrypt the control2 field and get the *𝑘*1, *𝑝𝑛𝑤*11 and the decryption key *𝑘* of the *𝑐𝑜𝑛𝑡𝑟𝑜𝑙*2 field in chain *𝑤*11 . By using *𝑝𝑛𝑤*12 and *𝑘*1, 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.

* 1. *Storage structure*

The smart device needs to record three symmetric keys *𝑘𝑓* , *𝑘𝑐* and

*𝑘𝑠* and a map *𝑀𝑎𝑝*, where *𝑀𝑎𝑝* is the data sent to the service peers and does not need to be stored locally. *𝑘𝑐* is used to encrypt files and identi- fies on the cloud, *𝑘𝑐* is used to encrypt indexes on the cloud, and *𝑘𝑠* 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*, *𝑘𝑐𝑡𝑟* , *𝑘𝑤* and rt, which will be used to encrypt and decrypt the

control nodes of index on the CSP. The flag field contains a boolean vari-

the keyword w has been retrieved. The c field contains *𝑐𝑡𝑟*, *𝐻𝑐𝑡𝑟* (*𝑖𝑛𝑑*) able ‘true’ or ‘false’ which means whether the latest chain representing and *𝐻* (*𝑖𝑛𝑑*). ctr records the number of chains corresponding w, while

*𝐻𝑐𝑡𝑟* (*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

*𝑖 𝑖*

The obtained *𝑉𝑤𝑖* 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 *𝑉𝑊* for the *𝑈* [*𝑤*]. After that, it stores the

Finally, it sends the *𝑈* [*𝑤*] to S and the *𝐸𝐷𝐵* to CSP to complete an update summary data into the blockchain to facilitate subsequent verification.

operation.

Dec) to encrypt the identifier ind and update operation *𝑜𝑝* correspond- In our scheme, device uses symmetric encryption algorithm (Enc, ing to each keyword in *𝑈* [*𝑤*]. Therefore, when the requester performs

retrieval operation, the server cannot get any information about the doc- ument except the document corresponding to the keyword, which rep- resents that the scheme can implement type-II backward security.

*𝑠𝑡𝑤* ← *𝐹* ((*𝑈* [*𝑤𝑖* ]*.*[*𝑐*][0])*, 𝐹* (*𝑘𝑠 , 𝑤*) (8)

*𝑠𝑡𝑤* ∗← *𝐹* (*𝑘𝑠 , 𝑤*)*.* (9)

***Veriﬁcation-I***. Upon receiving the data sent by smart device, the ser- vice 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 ter- minated 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

sending an un-updated *𝑈* [*𝑊* ] received data during retrieval operation. 3. Prevents service peer from

ates the following two retrieval tokens *𝑠𝑡𝑤* and *𝑠𝑡𝑤* ∗ for the keyword ***Search***. To perform the retrieval operation, the smart device gener-

that requester wants to retrieve.

**Algorithm 2** Update.

**Input:** a set of documents DOC and the actions *𝑜𝑝* that need to be per-

formed on them such as deleting or adding

**Output:** the updated index *𝐸𝐷𝐵* for CSP and *𝑈𝑤* for service peer

1: **Smart Device**:

2: *𝑈𝑠𝑒𝑡* ← *𝜙*

3: **for** each *𝑤𝑖* **do**

4: *𝑘𝑒𝑦𝑤* = *𝐹* (*𝑘𝑠 , 𝑤𝑖* )

5: *𝑈𝑠𝑒𝑡* = *𝑘𝑒𝑦𝑤* ∪ *𝑈𝑠𝑒𝑡*

*𝑖*

*𝑖*

6: **end for**

**Algorithm 3** Verification-I.

**Input:** *𝑉𝑤* 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 *𝑘𝑒𝑦𝑤𝑖* in *𝑤* **do**

3: *𝑉𝑤* ← *𝐻* (*𝑈* [*𝑘𝑒𝑦𝑤* ])

*𝑖 𝑖*

# 7: Service peer:

||

8: **for** each *𝑘𝑒𝑦𝑤𝑖* in *𝑈𝑠𝑒𝑡* **do**

9: *𝑈* [*𝑘𝑒𝑦𝑤* ] = lookup(*𝑚𝑎𝑝*, *𝑘𝑒𝑦𝑤* )

10: delete (*𝑘𝑒𝑦𝑤* , *𝑈* [*𝑘𝑒𝑦𝑤* ]) in map

*𝑖 𝑖*

*𝑖 𝑖*

11: **end for**

12: send all the *𝑈* [*𝑘𝑒𝑦𝑤* ] to device

*𝑖*

13: **Smart Device:**

14: **for** each *𝑘𝑒𝑦𝑤𝑖* **do**

15: *𝑉𝑤* ← *𝐻* (*𝑈* [*𝑘𝑒𝑦𝑤* ])

4: *𝑉𝑤𝑖𝑐* ← *𝐻* (*𝑘𝑒𝑦𝑤𝑖 𝑈* [*𝑤𝑖* ]*.*[*𝑐*][0])

5: **end for**

6: *𝑉𝑤* ← *𝐻* (*𝑈* (*𝑤*))

7: send *𝑉𝑤* and all the *𝑉𝑤𝑖* to the blockchain

8: **Service peer**:

9: *𝑉* ∗ ← *𝐻* (*𝑈* (*𝑤*))

10: send *𝑉* ∗ to the blockchain

*𝑤*

11: **if** *𝑉* ∗ ≠ *𝑉𝑤* **then**

*𝑤*

*𝑤*

12: terminate

*𝑖 𝑖*

16: check if it is stored in blockchain, if not terminate the protocol

17: **end for**

18: **for** each (*𝑖𝑛𝑑*, *𝑊𝑖𝑛𝑑* ) **do**

|  |  |
| --- | --- |
| 19: | *𝑤𝑖* ← *𝐹* (*𝑘𝑠 , 𝑊𝑖𝑛𝑑* ) |
| 20: | **if** *𝑈* [*𝑤𝑖* ] = *𝜙* or *𝑈* [*𝑤𝑖* ]*.*[*𝑓𝑙𝑎𝑔*] = *𝑡𝑟𝑢𝑒* **then** |
| 21: | **if** *𝑈* [*𝑤𝑖* ] = *𝜙* **then** |
| 22: | *𝑈* [*𝑤𝑖* ]*.*[*𝑐*][0] ← 0 |
| 23: | *𝑈* [*𝑤𝑖* ]*.*[*𝑐*][2] ← 0 |
| 24: | **end if** |
| 25: | *𝑐𝑡𝑟* ← *𝑈* [*𝑤𝑖* ]*.*[*𝑐*][0] + 1 |
| 26: | *𝐻𝑐𝑡𝑟* (*𝑖𝑛𝑑*) ← *𝐻* (*𝑖𝑛𝑑*) |
| 27: | *𝐻* (*𝑖𝑛𝑑*) ← *𝑈* [*𝑤𝑖* ]*.*[*𝑐*][2] + *𝐻 𝐻* (*𝑖𝑛𝑑*) |
| 28: | *𝑟𝑡*, *𝑘*, *𝑘*1, *𝑘𝑤* ← {0*,* 1}*𝜆* |
| 29: | *𝑘𝑒𝑦* ← *𝐹* (*𝑐𝑡𝑟, 𝐹* (*𝑘𝑐 , 𝑊𝑖𝑛𝑑* )) |
| 30: | **if** *𝑐𝑡𝑟* = 0 **then** |
| 31: | *𝑣*1 ← *𝑘*1 |
| 32: | **else** |
| 33:  34: | *𝑣*1 ← (*𝐹* (*𝑐𝑡𝑟* − 1*, 𝐹* (*𝑘𝑐 , 𝑤*))||*𝑈* [*𝑤𝑖* ]*.*[*𝑘𝑒𝑦*][0]||*𝑘*1 )  **end if** |
| 35: | *𝑣𝑎𝑙𝑢𝑒*[0] ← *𝐻* (*𝑘𝑤* ) *⊕ 𝑣*1 |
| 36:  37: | *𝑣𝑎𝑙𝑢𝑒*[1] ← *𝐻* (*𝑘*1) *⊕* (*𝑟𝑡*||*𝑘*||*𝐸𝑛𝑐*(*𝑖𝑛𝑑*||*𝑜𝑝*))  *𝑈* [*𝑤𝑖* ]*.*[*𝑘𝑒𝑦*] ← (*𝑘𝑤, 𝑘*1 *, 𝑘, 𝑟𝑡*) |
| 38: | *𝑈* [*𝑤𝑖* ]*.*[*𝑓𝑙𝑎𝑔*] ← *𝑓𝑎𝑙𝑠𝑒* |
| 39: | *𝑈* [*𝑤𝑖* ]*.*[*𝑐*] ← (*𝑐𝑡𝑟, 𝐻𝑐𝑡𝑟* (*𝑖𝑛𝑑*)*, 𝐻* (*𝑖𝑛𝑑*)) |
| 40: | *𝐸𝐷𝐵* ← (*𝑘𝑒𝑦, 𝑣𝑎𝑙𝑢𝑒*) ∪ *𝐸𝐷𝐵* |
| 41: | **else** |
| 42: | *𝑟𝑡*, *𝑘* ← {0*,* 1}*𝜆* |
| 43: | *𝑘𝑒𝑦* ← *𝑈* [*𝑤𝑖* ]*.*[*𝑘𝑒𝑦*][3] |
| 44: | *𝑣𝑎𝑙𝑢𝑒*[0] ← *𝐸𝑛𝑐*(*𝑖𝑛𝑑*||*𝑜𝑝*) |
| 45: | *𝑣𝑎𝑙𝑢𝑒*[1] ← *𝐻* (*𝑈* [*𝑤𝑖* ]*.*[*𝑘𝑒𝑦*][2]) *⊕* (*𝑟𝑡*||*𝑘*) |
| 46: | *𝑈* [*𝑤𝑖* ]*.*[*𝑘𝑒𝑦*][1] ← *𝑘*, *𝑈* [*𝑤𝑖* ]*.*[*𝑘𝑒𝑦*][3] ← *𝑟𝑡* |
| 47: | *𝑈* [*𝑤𝑖* ]*.*[*𝑐*][1] ← *𝑈* [*𝑤𝑖* ]*.*[*𝑐*][1] + *𝐻 𝐻* (*𝑖𝑛𝑑*) |
| 48: | *𝐸𝐷𝐵* ← (*𝑘𝑒𝑦, 𝑣𝑎𝑙𝑢𝑒*) ∪ *𝐸𝐷𝐵* |
| 49: | **end if** |

13: **else**

14: store the *𝑈* [*𝑤*] in map

15: **end if**

uses *𝑠𝑡𝑤* and *𝑘* to find the corresponding chain and returns the value[0] field of each node except control node to the device.

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

′

*𝑉 𝑡𝑤* ← *𝐻* (*𝐷𝐵*(*𝑤*)) (10)

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. Af-

ter receiving *𝑠𝑡𝑤* ∗, S will return the corresponding data *𝑈* [*𝑤*] to smart

blockchain. Besides, device needs to check if *𝐻* (*𝑘𝑒𝑦𝑤𝑖 𝑈* [*𝑤𝑖* ]*.*[*𝑐*][0] + 1) device. The device still firstly verifies the correctness of the data by

||

on the blockchain to prevent replay attack. When the data is considered

and send (*𝑠𝑡𝑤 , 𝑜𝑝, 𝑘*)as the search token to CSP via smart device.The CSP correct and latest, requester determines the chain he wants to search

Because the smart device has stored a similar validation data *𝑉 𝑡𝑤* in U[w] in advance. As long as the *𝑉 𝑡𝑤* is consistent with the *𝑉 𝑡𝑤* ′, 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.

# 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](#_bookmark13) the potential pitfalls to our solution. Next, we will verify that the security of our scheme is suﬃcient from two aspects.

* 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, con- sidering 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 af- ter 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 con- taining the keyword by themselves via smart device. The data used to generate the search token is securely stored in the service peers, and in- formation 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 doc- ument identifier to requester

1: **Smart Device**:

2: *𝑠𝑡𝑤* ∗← *𝐹* (*𝑘𝑠 , 𝑤*)

3: send *𝑠𝑡𝑤* ∗ to S

4: **Service peer**:

5: *𝑈* [*𝑤𝑖* ] ← *𝑙𝑜𝑜𝑘𝑢𝑝*(*𝑚𝑎𝑝, 𝑠𝑡𝑤* ∗) 6: delete (*𝑠𝑡𝑤* ∗, *𝑈* [*𝑤𝑖* ]) in map 7: return *𝑈* [*𝑤𝑖* ] to client

8: **Client**:

9: *𝑉* (*𝑤*) ← *𝐻* (*𝑈* [*𝑤𝑖* ])

10: check if the *𝑉* (*𝑤*) on the blockchain

11: check if *𝐻* (*𝑘𝑒𝑦𝑤𝑖 𝑈* [*𝑤𝑖* ]*.*[*𝑐*][0]) and *𝐻* (*𝑘𝑒𝑦𝑤𝑖 𝑈* [*𝑤𝑖* ]*.*[*𝑐*][0] + 1) on the

|| ||

blockchain

12: *𝑜𝑝* ← }*𝑎𝑙𝑙*′*𝑜𝑟*}*𝑜𝑛𝑒*′

13: *𝑘* ← *𝑈* [*𝑤𝑖* ]*.*[*𝑘𝑒𝑦*][0]*𝑜𝑟𝑈* [*𝑤𝑖* ]*.*[*𝑘𝑒𝑦*][1]

14: *𝑠𝑡𝑤* ← *𝐹* ((*𝑈* [*𝑤𝑖* ]*.*[*𝑐*][0])*, 𝐹* (*𝑘𝑠 , 𝑤*)

15: send (*𝑠𝑡𝑤* , *𝑜𝑝*, *𝑘*) to CSP

16: **CSP**:

17: *𝑘𝑒𝑦* ← *𝑠𝑡𝑤*

18: **if** *𝑜𝑝* =′ *𝑎𝑙𝑙*′ **then**

19: **while** *𝑘𝑒𝑦* ≠ *⊥* **do**

20: *𝑣𝑎𝑙𝑢𝑒* ← *𝑙𝑜𝑜𝑘𝑢𝑝*(*𝐸𝐷𝐵, 𝑘𝑒𝑦*)

21: (*𝑘𝑒𝑦 𝑘𝑤 𝑘*) ← *𝐻* (*𝑘*) *⊕ 𝑣𝑎𝑙𝑢𝑒*[0]

|| ||

22: *𝑠𝑡𝑠𝑒𝑡* ← (*𝑘𝑒𝑦, 𝑘*) ∪ *𝑠𝑡𝑠𝑒𝑡*

23: (*𝑟𝑡 𝑘*1 *𝑚𝑠𝑔*) ← *𝐻* (*𝑘𝑤*) ∪ *𝑣𝑎𝑙𝑢𝑒*[1]

|| ||

24: *𝑟𝑒𝑠𝑢𝑙𝑡* ← *𝑚𝑠𝑔* ∪ *𝑟𝑒𝑠𝑢𝑙𝑡*

25: **end while**

26: **else**

27: *𝑠𝑡𝑠𝑒𝑡* ← (*𝑘𝑒𝑦, 𝑘*)

28: **end if**

29: **for** each (*𝑘𝑒𝑦, 𝑘*)*𝑖𝑛𝑠𝑡𝑠𝑒𝑡* **do**

30: *𝑣𝑎𝑙𝑢𝑒* ← *𝑙𝑜𝑜𝑘𝑢𝑝*(*𝐸𝐷𝐵, 𝑘𝑒𝑦*)

31: **while** *𝑣𝑎𝑙𝑢𝑒* ≠ *⊥* **do**

32: (*𝑟𝑡 𝑘*1) ← (*𝑣𝑎𝑙𝑢𝑒 ⊕ 𝐻* (*𝑘*))

||

33: *𝑟𝑒𝑠𝑢𝑙𝑡* ← *𝑣𝑎𝑙𝑢𝑒*[0] ∪ *𝑟𝑒𝑠𝑢𝑙𝑡*

34: *𝑘𝑒𝑦* ← *𝑟𝑡*

35: *𝑘* ← *𝑘*1

36: *𝑣𝑎𝑙𝑢𝑒* ← *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* [*𝑤𝑖* ]

**Output:** smart device decrypts the *𝐸𝑛𝑐*(*𝑖𝑛𝑑 𝑜𝑝*) and gets the *𝐷𝐵*(*𝑤𝑖* ),

||

it is equal to *𝐻* (*𝑖𝑛𝑑*) which is stored in *𝑈* [*𝑤𝑖* ]*.*[*𝑐*][2] then it uses set hash to get a hash value of the results and checks if

1: **Smart Device**:

2: *𝑉 𝑡𝑤* ′ ← 0

3: **for** each *𝐸𝑛𝑐*(*𝑖𝑛𝑑 𝑜𝑝*) in result **do**

||

4: (*𝑖𝑛𝑑, 𝑜𝑝*) ← *𝐷𝑒𝑐*(*𝐸𝑛𝑐*(*𝑖𝑛𝑑 𝑜𝑝*))

||

5: **if** *𝑂𝑃* =′ *𝑎𝑑𝑑*′ **then**

6: *𝑉 𝑡𝑤* ′ ← *𝐻 𝐻* (*𝑖𝑛𝑑*) + *𝑉 𝑡𝑤* ′

7: **end if**

8: **end for**

9: *𝑉 𝑡𝑤* ← *𝑈* [*𝑤𝑖* ]*.*[*𝑐*][2]

10: **if** *𝑉 𝑡𝑤* ′ = *𝑉 𝑡𝑤* **then**

***Adaptive security***. Our scheme is the first to achieve forward update privacy and backward security with constant client storage by the coop- eration of on-chain and off-chain. The adaptive security of our scheme is proven in [Theorem 1](#_bookmark15).

**Theorem 1.** *Let 𝜆 be the security parameter. Assume 𝐿*′ *is stateless. Define*

*𝐿𝐷𝑆𝑆𝐸* =*(𝐿𝑆𝑒𝑎𝑟𝑐ℎ, 𝐿𝑈𝑝𝑑𝑎𝑡𝑒), where*

*𝐷𝑆𝑆𝐸 𝐷𝑆𝑆𝐸*

*𝐿𝑆𝑒𝑎𝑟𝑐ℎ*(*𝑤*) = (*𝐿*′′ (*𝑠𝑝*(*𝑤*)*, 𝑇 𝑖𝑚𝑒𝐷𝐵*(*𝑤*)*, 𝑈𝑝𝑑𝑎𝑡𝑒𝑠*(*𝑤*))*,*

*𝐿𝑈𝑝𝑑𝑎𝑡𝑒*(*𝑤, 𝑜𝑝, 𝑖𝑛𝑑*) = *𝐿*′ (*𝑤*)*.*

We believe that our scheme is £-adaptive-secure in the random ora- cle 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 theo- rem. 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[*𝐷𝑆𝑆𝐸𝑅𝑒𝑎𝑙𝐷𝑆𝑆𝐸* (*𝜆*) = 1] = Pr[*𝐺𝑎𝑚𝑒*0 = 1]*.*

*𝐴*

**Game1.** In this game, to simulate the operation of pseudo-random functions, system maintain a table to record the results of keyword re- quest. In Update protocol, adversary A sends a token request to the sys- tem. 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 *𝛽*1 such that:

Pr[Game0=1]-Pr[Game1=1] ≤ *𝐴𝑑𝑣𝑝𝑟𝑓* (*𝜆*)

*𝐹 ,𝛽*1

**Game2.** In this game, to generate the ciphertext of identifier, sys- tem encrypt the string of “0”s rather than use the identifier. The oth- ers 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 *𝛽*2 such that:

Pr[Game1=1]-Pr[Game2=1] ≤ *𝐴𝑑𝑣𝐸𝑛𝑐*(*𝜆*)

*𝛽*2

**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 *𝛽*3 such that

Pr[Game2=1]-Pr[Game3=1] ≤ *𝐴𝑑𝑣ℎ𝑎𝑠ℎ* (*𝜆*)

*𝐻,𝛽*3

**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[*𝐷𝑆𝑆𝐸𝑅𝑒𝑎𝑙𝐴* (*𝜆*) = 1] − Pr[*𝐷𝑆𝑆𝐸𝐼𝑑𝑒𝑎𝑙𝐴* (*𝜆*) = 1] ≤ *𝐴𝑑𝑣𝑝𝑟𝑓* (*𝜆*)

*𝐹 ,𝛽*1

11: complete a correct search operation

+ *𝐴𝑑𝑣𝐸𝑛𝑐*(*𝜆*) + *𝐴𝑑𝑣ℎ𝑎𝑠ℎ* (*𝜆*)

12: **else**

*𝛽*2

*𝐻,𝛽*3

13: the result is wrong and call for appeals of the result

14: **end if**

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 |

* 1. *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 re- late it to other index data, which guarantees weak forward update pri- vacy. 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](#_bookmark15) and [Definition 3](#_bookmark12), we can come to the following conclusion:

**Corollary 1.** *Our scheme satisfies weak forward update privacy and back-* *ward privacy.*

* 1. *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 meta- data they upload on the chain, while the monitoring of malicious be- havior 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.

# 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 encryp- tion scheme we used for encrypting identifier is AES. The computer is Intel(R) Core (TM) i5-10210U and 8GB of RAM that running on win- dows 11. We will prove the reliability of this scheme applied to smart devices from storage cost, computation cost and communication cost respectively.

* 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

tion is displayed in [Table 2](#_bookmark16). Among them, *𝑛* represents the number of smart devices, we divide our dataset into 3 sets. The detailed informa- documents, *𝑚* is on behalf of the number of keywords extracted from the documents and *𝑁* is the number of keyword-document pairs.

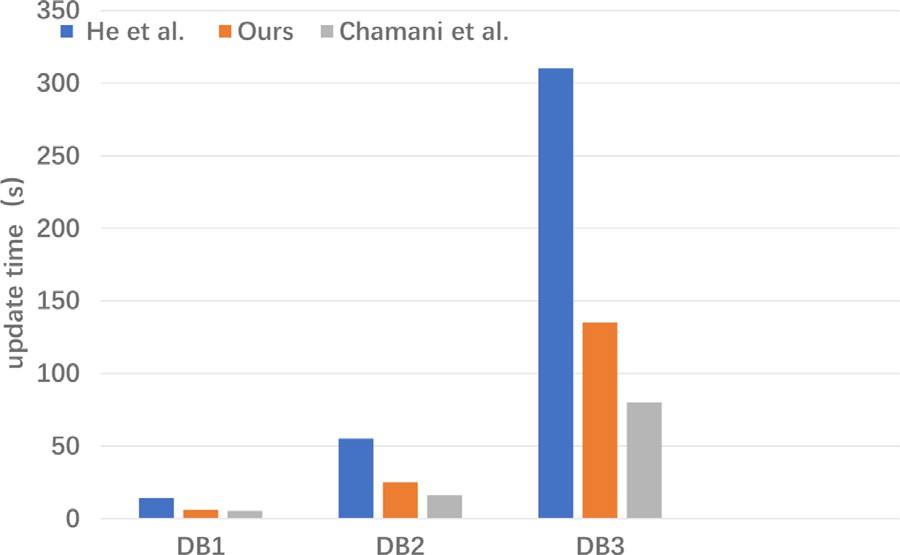
* 1. *Storage cost*

The Local storage resource usage of the protocol is an important indi- cator 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]](#_bookmark25) | 203KB | 610KB | 2,711KB |
| Li et al. [[19]](#_bookmark43) | 1,071KB | 3,207KB | 14,271KB |
| He et al. [[13]](#_bookmark38) | 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](#_bookmark16), local storage overhead of Bost et al. [[5]](#_bookmark25) and Li et al. [[19]](#_bookmark43) increases linearly as the number of keywords increases. It is worth noting that although the scheme of Li et al. [[19]](#_bookmark43) can achieve

threshold *𝑃* which means the maximum number of blocks in the chain as forward search privacy, their local storage overhead is affected by the

well as the number of sub-keywords. In our experiment, we set the num- ber of sub-keywords to 1 which means that their local storage overhead will be even heavier in practical. He et al. [[13]](#_bookmark38) 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.

* 1. *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]](#_bookmark38) and Ghareh et al.’s scheme [[11]](#_bookmark35) to verify the performance of our protocol in search and update operation.

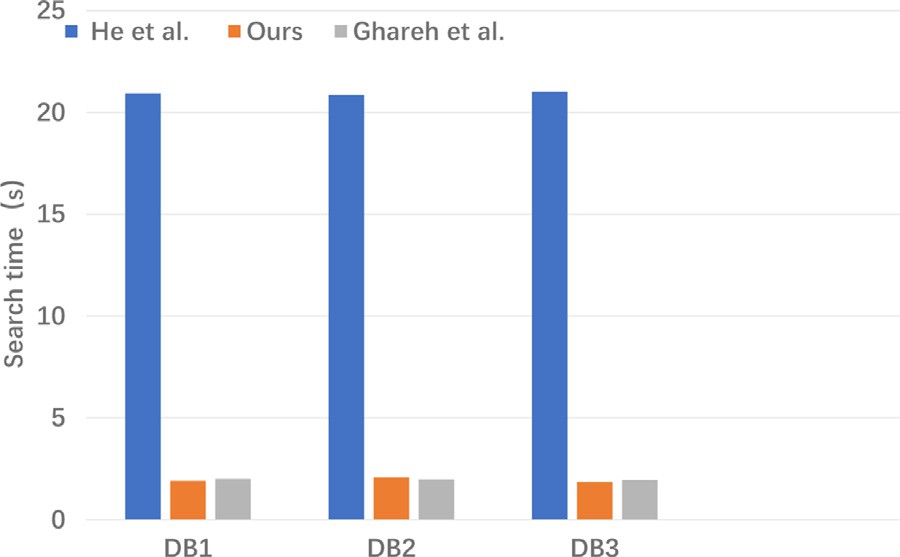
we choose *𝑐𝑙𝑒𝑛* for 2000 which means He et al.’s protocol can support [Figure 3](#_bookmark17) shows the update time on 3 different databases. Specifically,

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 fig- ure, it is obviously that the update time increases linearly as the num- ber 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 computa- tion. 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](#_bookmark20), 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.

* 1. *Communication costs*

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

[Table 4](#_bookmark21) gives the communication costs of clients updating the database to different objects. It is important to note that our commu- nication 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 sin- gle update of DB3. Encrypted index is transferred to the CSP. It is a data volume that increases linearly as the number of keywords increases. Un- der 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 up- loads 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 over- head 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 communi-

cation overhead, we give [Table 5](#_bookmark20). 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.

# 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 opera- tion of the protocol. The smart device does not need to record any data related to keywords locally, and the retrieval operation has low compu- tation 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.](http://refhub.elsevier.com/S2772-9184(22)00006-6/sbref0001)
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.](http://refhub.elsevier.com/S2772-9184(22)00006-6/sbref0002)
3. [R. Bost, Forward secure searchable encryption, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016,](http://refhub.elsevier.com/S2772-9184(22)00006-6/sbref0003)

[pp. 1143–1154.](http://refhub.elsevier.com/S2772-9184(22)00006-6/sbref0003)

1. [R. Bost, P.-A. Fouque, D. Pointcheval, Verifiable dynamic symmetric searchable en- cryption: optimality and forward security, IACR Cryptol. ePrint Arch. 2016 (2016) 62.](http://refhub.elsevier.com/S2772-9184(22)00006-6/sbref0004)
2. [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,](http://refhub.elsevier.com/S2772-9184(22)00006-6/sbref0005)

[pp. 1465–1482.](http://refhub.elsevier.com/S2772-9184(22)00006-6/sbref0005)

1. [C. Cai, J. Weng, X. Yuan, C. Wang, Enabling reliable keyword search in encrypted decentralized storage with fairness, IEEE Trans. Dependable Secure. Comput. (2018).](http://refhub.elsevier.com/S2772-9184(22)00006-6/sbref0006)
2. [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.](http://refhub.elsevier.com/S2772-9184(22)00006-6/sbref0007)
3. [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.](http://refhub.elsevier.com/S2772-9184(22)00006-6/sbref0008)
4. [R. Curtmola, J. Garay, S. Kamara, R. Ostrovsky, Searchable symmetric encryption: improved definitions and eﬃcient constructions, J. Comput. Secur. 19 (5) (2011) 895–934.](http://refhub.elsevier.com/S2772-9184(22)00006-6/sbref0009)
5. [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).](http://refhub.elsevier.com/S2772-9184(22)00006-6/sbref0010)
6. [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.](http://refhub.elsevier.com/S2772-9184(22)00006-6/sbref0011)
7. [P. Golle, I. Mironov, Uncheatable distributed computations, in: Cryptographers’ Track at the RSA Conference, Springer, 2001, pp. 425–440.](http://refhub.elsevier.com/S2772-9184(22)00006-6/sbref0012)
8. [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.](http://refhub.elsevier.com/S2772-9184(22)00006-6/sbref0013)
9. [M.S. Islam, M. Kuzu, M. Kantarcioglu, Access pattern disclosure on searchable en- cryption: ramification, attack and mitigation, in: Ndss, vol. 20, Citeseer, 2012, p. 12.](http://refhub.elsevier.com/S2772-9184(22)00006-6/sbref0014)
10. [S. Kamara, C. Papamanthou, Parallel and dynamic searchable symmetric encryption, in: International Conference on Financial Cryptography and Data Security, Springer, 2013, pp. 258–274.](http://refhub.elsevier.com/S2772-9184(22)00006-6/sbref0015)
11. [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.](http://refhub.elsevier.com/S2772-9184(22)00006-6/sbref0016)
12. [S.K. Kermanshahi, S.-F. Sun, J.K. Liu, R. Steinfeld, S. Nepal, W.F. Lau, M. Au, Geo- metric range search on encrypted data with forward/backward security, IEEE Trans. Dependable Secure. Comput. (2020).](http://refhub.elsevier.com/S2772-9184(22)00006-6/sbref0017)
13. [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,](http://refhub.elsevier.com/S2772-9184(22)00006-6/sbref0018)

[pp. 1449–1463.](http://refhub.elsevier.com/S2772-9184(22)00006-6/sbref0018)

1. [J. Li, Y. Huang, Y. Wei, S. Lv, Z. Liu, C. Dong, W. Lou, Searchable symmetric encryp- tion with forward search privacy, IEEE Trans. Dependable Secure. Comput. (2019).](http://refhub.elsevier.com/S2772-9184(22)00006-6/sbref0019)
2. [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.](http://refhub.elsevier.com/S2772-9184(22)00006-6/sbref0020)
3. [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. Depend- able Secure Comput. (2020).](http://refhub.elsevier.com/S2772-9184(22)00006-6/sbref0021)
4. [Y. Miao, J. Ma, Q. Jiang, X. Li, A.K. Sangaiah, Verifiable keyword search over en- crypted cloud data in smart city, Comput. Electr. Eng. 65 (2018) 90–101.](http://refhub.elsevier.com/S2772-9184(22)00006-6/sbref0022)
5. [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, Com- munication and Networking Technologies (ICCCNT), IEEE, 2018, pp. 1–4.](http://refhub.elsevier.com/S2772-9184(22)00006-6/sbref0023)
6. [M. Naveed, The fallacy of composition of oblivious ram and searchable encryption, IACR Cryptol. ePrint Arch. 2015 (2015) 668.](http://refhub.elsevier.com/S2772-9184(22)00006-6/sbref0024)
7. [M. Naveed, S. Kamara, C.V. Wright, Inference attacks on property-preserving en- crypted databases, in: Proceedings of the 22nd ACM SIGSAC Conference on Com- puter and Communications Security, 2015, pp. 644–655.](http://refhub.elsevier.com/S2772-9184(22)00006-6/sbref0025)
8. [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.](http://refhub.elsevier.com/S2772-9184(22)00006-6/sbref0026)
9. [E. Stefanov, C. Papamanthou, E. Shi, Practical dynamic searchable encryption with small leakage, in: NDSS, vol. 71, 2014, pp. 72–75.](http://refhub.elsevier.com/S2772-9184(22)00006-6/sbref0027)
10. [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 Sympo- sium ({USENIX} Security 16), 2016, pp. 707–720.](http://refhub.elsevier.com/S2772-9184(22)00006-6/sbref0028)
11. [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).](http://refhub.elsevier.com/S2772-9184(22)00006-6/sbref0029)

**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 in- terests 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 In- vestigator of three NSF projects.