Forward Private Multi-Client Searchable Encryption with Efficient Access Control in Cloud Storage

Jinjiang Yang*, Feng Liu*, Xinyi Luo*, Jianan Hong†, Jian Li*, Kaiping Xue*‡
*School of Cyber Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China
†School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai 200241, China
‡Corresponding author, kpxue@ustc.edu.cn

***Abstract*—Through Searchable Symmetric Encryption (SSE), a user can make search over encrypted documents that are stored on an untrusted cloud server. Multi-client SSE schemes require that one client can search documents contributed by other clients and upload documents. Nevertheless, existing multi-client SSE schemes implement the fine-grained access control with high complexity. Although fine-grained access control adapts to complex scenarios, it is not necessary anytime and may cause heavy costs over computation in SSE schemes. Moreover, it is crucial to support documents updating and forward privacy. To combat that, we design a multi-client SSE scheme with efficient access control over dynamic encrypted documents. Specifically, we first modify Symmetric Hidden Vector Encryption (SHVE) and utilize Bloom filter to implement the access control, which reduces much of computation overhead. We then employ Oblivious Dynamic Cross-Tag (ODXT) protocol to preserve the forward privacy of our scheme. Finally, the corresponding security and experimental evaluation demonstrate both security and practicality of our scheme, respectively.**

***Index Terms*—Cloud Computing, Searchable Encryption, Access Control, Multiple Clients, Forward Privacy**

I. Introduction

With the rapid advent of cloud computing [1], individuals and enterprises are able to outsource storage and computations to a cloud. Meanwhile, to prevent privacy leakage on outsourced data, data owners can encrypt data before uploading it to the cloud. Nevertheless, traditional encryption will result in extremely low data availability. For instance, queries on outsourced data will not be possible.

Searchable Symmetric Encryption (SSE) offers a potential solution to the problem above, so that a client can search encrypted documents securely. Till now, works on SSE are mainly focused on two scenarios: one is that a client uploads documents to the cloud and only herself can search these encrypted documents, e.g., [2]–[4], and the other is that one client is supposed to upload documents while other multiple clients can search them from the cloud, e.g., [5]–[8]. However, in many realistic scenarios, clients should not only search documents but also upload their own documents by themselves. For example, cases like collaborate e-health or project executions require a crowd of members to upload relevant data, and meanwhile, according to their roles, members should search and access documents contributed by other members. This paper names such scenarios as *Multi-Client* scenarios.

978-1-6654-3540-6/22/$31.00 © 2022 IEEE

Although some researchers gave solutions for such scenarios such as [9], there are still two pressing issues remaining to be addressed. Firstly, we need a more lightweight approach to achieve access control for multiple clients. Current access control methods in multi-client SSE schemes are fine-grained and not suitable for all cases due to their complexity and heavy computational overhead. Secondly, multi-client scenarios must be dynamic. Therefore, to achieve secure uploading, forward privacy [10] for the upload process is also necessary. Nevertheless, to the best of our knowledge, it has received little discussion in multi-client works.

To address the above two issues, we construct a novel multi-client SSE scheme with efficient access control and forward privacy preservation. Notably, our access control is symmetric and based on privilege levels. Compared with attribute-based schemes, our proposed scheme aims at a simpler and faster multi-client SSE scenario based on privilege levels, rather than those for complex access structure.In particular, the main contributions of our model can be summarized as follows:

- We propose a multi-client searchable symmetric encryption scheme with efficient access control by utilizing the modified Symmetric Hidden Vector Encryption [4] and Bloom filter [11], both of which use symmetric keys. In this way, our scheme has much less computation costs.
- By employing Oblivious Dynamic Cross-Tag (ODXT) protocol [12], our proposed scheme is able to provide forward privacy for uploading documents. So our multi-client scheme is forward private in a dynamic setting.
- We theoretically analyse forward privacy and clients security to prove that our scheme is secure in multi-client scenarios. Moreover, we conduct extensive experiments to demonstrate that our design is efficient in terms of computation and storage overhead.

The rest of our paper is organized as follows. In Section II, we introduce the related work. And then in Section III, we give the problem statement. Bloom filter, the modified Symmetric Hidden Vector Encryption algorithm, and ODXT model are briefly described in Section IV. Section V illustrates our proposed scheme in detail. Afterwards, system analysis and performance evaluation are provided in Section VI. Finally, in Section VII, we have a summary of this paper.

II. Related Work

For a long time, multi-client scenarios in SSE schemes have not been taken seriously. Recently, one study [9] has attempted to design a multi-client SSE scheme based on ABE and OXT [3]. In their scheme, clients can use their own attributes to set access policies for documents and then upload them to the server. Meanwhile, other clients can locate a document only if their own attributes satisfy its access policy.

However, there are two limitations in [9]. Firstly, the utilization of ABE brings in unbearable overhead, while it is not necessary to deploy such a complex access control approach for most cases (Privilege level is a more common method for access control). Secondly, it allows any client to upload documents anytime, but the corresponding security issue, i.e., forward privacy, is not considered. Forward privacy was first claimed by Stefanov *et al*. [10], which indicates that in an uploading-supporting SSE scheme, a newly inserted document should not be matched by previous search tokens. Once an SSE scheme lacks forward privacy, the file injection attack [13] would rapidly devastate the encrypted system.

TABLE I
Comparison with Various Schemes

Schemes	Multiple Users	Multiple Clients	Fine-Grained Access Control	Efficient Access Control	Forward Privacy
[5]	√	×	√	×	-
[6]	√	×	-	×	-
[7]	√	×	-	×	-
[9]	√	√	√	×	×
Ours	√	√	×	√	√

Overall, we present a comparison between our proposed scheme and works in multi-user/multi-client scenarios. As listed in Table I, no previous work supports both multiple clients and forward privacy. Moreover, only the scheme of Zhang *et al*. [9] holds multiple clients. But, as discussed above, its fine-grained access control is not necessary in our scenarios and will cause heavy computation burden.

III. Problem Statement

A. System Model

There are three types of entities in our SSE system as shown in Fig. 1: Manager, Cloud Server and Clients.

- Manager: The manager is a trusted party that maintains the system. It selects and distributes the access tokens for every client.
- Clients: A client outsources documents and searches documents contributed by other clients. Every client obtains his/her access tokens through assignment of the manager, and uses them to set access policies when uploading documents. If a client attempts to conduct a query, he/she can only obtain those documents whose access policies are satisfied by his/her access tokens.
- Cloud Server: Cloud Server runs the algorithms and provides powerful computing and storage capacity. When a client uploads a document to the server, the server would process outsourced ciphertext with its corresponding index. Also, it will perform the search processing and return precise ciphertext results if a client sends a search token to Cloud Server.

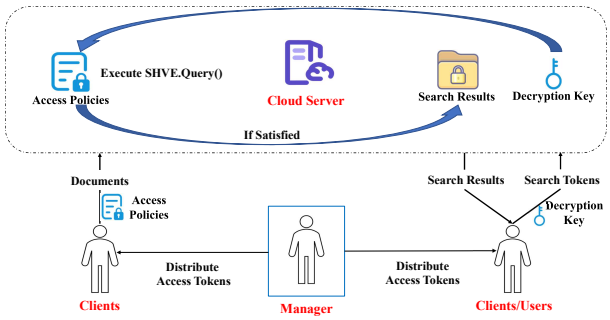


Fig. 1. System Model

B. Security Assumption

In the proposed scheme, we assume that Cloud Server is honest-but-curious. In this model, Cloud Server will execute the protocols and procedures honestly. Meanwhile, it is curious and tries to collect and analyze uploaded documents from clients in order to obtain some additional privacy information.

Among all clients, except the manager, there may be colluded ones in order to search or access the documents beyond their own permission. Moreover, now that it is clients who generate Bloom filters used in our scheme, they may maliciously add values into Bloom filters.

C. Design Goals

Our proposed scheme intends to achieve efficient conjunctive queries in dynamic multi-client scenarios. Therefore, there are several goals for security and performance in our design.

- Sub-linear conjunctive queries. In our system, a client can issue sub-linear conjunctive queries over keywords that have been stored in the encrypted database.
- Multiple clients. All clients can outsource their documents to Cloud Server and search outsourced documents contributed by other clients according to the corresponding access tokens.
- Light computational costs. Our scheme utilizes merely symmetric encryption to implement access control. We argue that our proposed scheme will have less computation costs than the previous schemes.
- Forward privacy preservation. Motivated by ODXT, we construct a counter to preserve forward privacy, so that Cloud Server cannot match the connection between newly uploaded documents and previously sent search tokens.

IV. Preliminaries

A. Bloom Filter

Bloom filter, a probabilistic structure, was first proposed by Bloom [11] in 1970. Because of its efficiency on retrieving, Bloom filter has been widely used to determine whether certain

element belongs to a collection. Bloom filter is initialized as an m-bit bit array with all bits set as 0. Given a set $S = \{a_1, a_2, \ldots, a_n\}$ and l independent hash functions: $H = \{h_j | h_j : S \to [1, m], 1 \leq j \leq l\}$, Bloom filter sets the $h_j(a_i)$-th bit in the array to 1. When testing whether an element a is in S, we calculate $h_j(a), 1 \leq j \leq l$ to obtain l positions. If any of the l positions equals to 0, then $a \notin S$; otherwise, it means that $a \in S$ or a yields a false positive.

In this paper, we make little modification to Bloom filter as there will be security issues if we use Bloom filter as always.

B. Modified Symmetric Hidden Vector Encryption

Symmetric Hidden Vetor Encryption (SHVE) scheme was first proposed by Lai *et al.* [4] to encrypt vector data. In our proposed scheme, we slightly modify SHVE protocol to make it be able to carry a payload message, which is not implemented in [4]. In our framework, Σ is a finite set of attributes. Typically, Σ is a finite field $\mathbb{Z}_p$.

The modified SHVE utilizes a pseudorandom function (PRF) $F_0 : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^\lambda$ and a symmetric encryption scheme (Sym.Enc, Sym.Dec). The detail of construction can be defined as the following four Probabilistic Polynomial-Time (PPT) algorithms:

- $\mathsf{SHVE.Setup}(1^\lambda) \to msk$: On input a security parameter λ, the algorithm uniformly samples a master secret key $msk \xleftarrow{\$} \{0,1\}^\lambda$ and outputs a message space $\mathcal{M}$.
- $\mathsf{SHVE.Enc}(msk, \mu \in \mathcal{M}, y \in \Sigma^m) \to c$: On input a vector $y = (y_1, ..., y_m)$ and the msk, let $C = \{l_j \in [m] | y_{l_j} \neq 0\}$ be the set of all locations in y that do not equal to 0, and $l_1 < l_2 < ... < l_{|C|}$.

  The algorithm samples a key $K \xleftarrow{\$} \{0,1\}^\lambda$ and constructs ciphertext c as follows:

  $$d_0 = \oplus_{j \in [|C|]}(F_0(msk, y_{l_j} || l_j)) \oplus K,$$
  $$d_1 = Sym.Enc(K, \mu).$$

  Then, it outputs the ciphertext $c = \{d_0, d_1, C\}$.
- $\mathsf{SHVE.KeyGen}(msk, x \in \Sigma^m) \to s$: On input a vector $x = (x_1, ..., x_m)$ and the master secret key msk, for each $l \in [m]$, the algorithm calculates $s_l = F_0(msk, x_l || l)$. And then, output the decryption key $s = (\{s_l\}_{l \in [m]})$.
- $\mathsf{SHVE.Query}(c, s)$: On input ciphertext c and a decryption key s, the algorithm then calculates:

  $$K' = (\oplus_{j \in |C|} s_j) \oplus d_0,$$
  $$\mu' = Sym.Dec(K', d_1).$$

  If the values of those locations in C are equal between vector x and y, then the obtained μ' is also equal to μ; otherwise, we will obtain that $\mu' \notin \mathcal{M}$.

C. Oblivious Dynamic Cross-Tag

Oblivious Dynamic Cross-Tag (ODXT) [12] is a conjunctive SSE scheme, which preserves forward privacy while uploading documents. An ODXT scheme formally consists of the following three algorithms:

- $\mathsf{ODXT.Setup}(1^\lambda)$: Taking a security parameter λ as input, this algorithm outputs a set of keys sk, an initialized counter st that records the counter number of updating for each keyword, and an empty database EDB.
- $\mathsf{ODXT.Update}(\mathsf{sk}, \mathsf{st}, \mathsf{op}, (id, w); \mathsf{EDB})$: On input the secret keys sk, the counter st, an operation $op \in \{add, del\}$, the updating information (id, w), and encrypted database EDB, the algorithm will update the corresponding section in EDB according to the information it obtains.
- $\mathsf{ODXT.Search}(\mathsf{sk}, \mathsf{st}, q = (w_1, w_2, ..., w_n); \mathsf{EDB})$: With the secret keys sk and counter st, this algorithm finds those documents that satisfy the set of keywords q in EDB. Finally, it returns these documents to the client.

V. The Proposed Scheme

A. Overview

As we mentioned before, our multi-client SSE scheme is designed for dynamic scenarios where fine-grained access control is not essential because of its heavy computation costs. Therefore, in this paper, we design an efficient authorization method to implement access control. Our method is conducted by Bloom filter and modified SHVE, which leads to a much less computation cost than fine-grained methods.

In our paper, the system is composed of a manager, Cloud Server and clients. The manager determines all privilege levels in the group and which level a client belongs to. Then, the manager generates access tokens according to the privilege levels, and distributes corresponding access tokens to clients. If a client desires to upload documents, he/she first selects access tokens to be inserted into a Bloom filter BF_1. Subsequently, the client uses $\mathsf{SHVE.Enc}(\cdot)$ to encrypt BF_1, leading to an encrypted result c. Afterwards, the encrypted document is uploaded to Cloud Server with corresponding c.

While a client attempts to search documents on the server, he/she also chooses access tokens to obtain a Bloom filter BF_2. Through $\mathsf{SHVE.KeyGen}(\cdot)$, the client gains a decryption key s and sends it to the server with a keywords set. Cloud Server finds those documents and exploits $\mathsf{SHVE.Query}(\cdot)$ to test whether the decryption key s can decrypt corresponding c. If so, the server returns that document to the client.

The complete procedure of our scheme can be divided into 4 steps: $Setup$, $Upload$, $TokenGen$, $Search$.

B. Setup Step

In this step, the manager divides the whole group's privileges into ℓ levels: $level_1, level_2, \ldots, level_\ell$. According to the privilege levels, the manager generates ℓ encrypted access tokens: $token_1, token_2, \ldots, token_\ell$. Later, he/she determines which level each client belongs to and distributes corresponding access tokens to every client. For instance, a client at $level_3$ will be distributed $token_1, token_2$ and $token_3$.

Meanwhile, the manager invokes $\mathsf{ODXT.Setup}(1^\lambda)$ and $\mathsf{SHVE.Setup}(1^\lambda)$ to generate secret keys sk, an initialized counter st, a master secret key msk, a message space $\mathcal{M}$, and an empty database EDB. Afterwards, (sk, msk) is sent to clients with corresponding access tokens, and EDB is uploaded to the server. As for st, all clients share and maintain it.

Algorithm 1: Upload

Input: Client's access tokens $token_1, \ldots, token_{n_1}$;
Encrypted database $\mathsf{EDB} = (TSet, XSet)$;
Client's keys $\mathsf{sk} = (K_T, K_X, K_Y, K_Z)$;
Keyword set $w = (w_1, w_2, \ldots, w_n)$;
The master secret key msk;
Document identifier id;
Counter $\mathsf{st} = \mathsf{UpdCnt}$.

Client:

1 Initialize $BF_1 \leftarrow 0^m$.
2 **for** $i \leftarrow 1$ *to* n_1 **do**
3 **for** $j \leftarrow 1$ *to* k **do**
4 Set $BF_1[h_j(token_i)] = h_j(token_i)$.
5 **end**
6 **end**
7 **for** $i \leftarrow 1$ *to* n **do**
8 **if** $\mathsf{UpdCnt}[w_i]$ *is NULL* **then**
9 $\mathsf{UpdCnt}[w_i] = 0$.
10 **end**
11 Set $\mathsf{UpdCnt}[w_i] = \mathsf{UpdCnt}[w_i] + 1$;
12 Set $addr = F(K_T, w_i || \mathsf{UpdCnt}[w_i] || 0)$;
13 Set $e = id \oplus F(K_T, w_i || \mathsf{UpdCnt}[w_i] || 1)$;
14 Compute $z = F_P(K_Z, w_i || \mathsf{UpdCnt}[w_i])$;
15 Set $\alpha = F_P(K_Y, id) \cdot z^{-1}$;
16 Set $\mu = (e, \alpha)$;
17 Compute $c = \mathsf{SHVE.Enc}(msk, \mu, BF_1)$;
18 Compute $xtag = g^{F_P(K_X, w_i) \cdot F_P(K_Y, id)}$;
19 Send $(addr, c, xtag)$ to server.
20 **end**
21 **end**

Server:

22 Set $TSet[addr] = c$ and $XSet[xtag] = 1$.
23 **end**

C. Upload Step

The algorithm is shown in Algorithm 1, where F and F_p are PRFs. In this step, a client first chooses access tokens $token_1, \ldots, token_{n_1}$ and maps them into a bloom filter BF_1. Later, the client generates encrypted index values of document identifier id and keywords set $w = (w_1, w_2, \ldots, w_n)$ with algorithm $\mathsf{SHVE.Enc}(\cdot)$. The details of encryption are shown in line 7-18. Afterwards, the server executes the addition operation in EDB according to what the client has sent.

D. TokenGen Step

In this step, a client is supposed to generate a search token and send it to Cloud Server. Firstly, with access tokens $token_1, \ldots, token_{n_2}$, the client calculates a decryption key s by employing $\mathsf{SHVE.KeyGen}(\cdot)$. To perform a search query $Q = (w_1, w_2, \ldots, w_q)$, the client gets $\mathsf{UpdCnt}[w_1]$ from st. Then he/she generates stokenList and xtokenLists, and sends them along with the decryption key s as $search\_token$ to Cloud Server. The details are as shown in Algorithm 2.

E. Search Step

As shown in Algorithm 3, while receiving $search\_token$ from a client, Cloud Server first locates where those documents are in EDB according to stokenList. From these locations, the server obtains corresponding encrypted index values. Let c be

Algorithm 2: TokenGen

Input: Client's access tokens $token_1, \ldots, token_{n_2}$;
Client's keys $\mathsf{sk} = (K_T, K_X, K_Y, K_Z)$;
Keyword set $w = (w_1, w_2, \ldots, w_q)$;
The master secret key msk;
Counter $\mathsf{st} = \mathsf{UpdCnt}$.

Client:

1 Initialize $BF_2 \leftarrow 0^m$.
2 **for** $i \leftarrow 1$ *to* n_2 **do**
3 **for** $j \leftarrow 1$ *to* k **do**
4 Set $BF_2[h_j(token_i)] = h_j(token_i)$.
5 **end**
6 **end**
7 Compute $s = \mathsf{SHVE.KeyGen}(msk, BF_2)$.
8 From UpdCnt identify the keyword with least updates (assume it to be w_1).
9 Initialize $\mathsf{stokenList}, \mathsf{xtokenList}_1, \ldots, \mathsf{xtokenList}_{\mathsf{UpdCnt}[w_1]}$ to empty lists.
10 **for** $j \leftarrow 1$ *to* $\mathsf{UpdCnt}[w_1]$ **do**
11 Set $search\_addr_j = F(K_T, w_1 || j || 0)$;
12 Set $\mathsf{stokenList}[j] = search\_addr_j$;
13 **for** $i \leftarrow 2$ *to* q **do**
14 Set $\mathsf{xtoken}_{j,i} = g^{F_P(K_X, w_i) \cdot F_P(K_Z, w_1 || j)}$;
15 Set $\mathsf{xtokenList}_j[i] = \mathsf{xtoken}_{j,i}$.
16 **end**
17 **end**
18 **Return** $Search\_token = (s, \mathsf{stokenList}, \mathsf{xtokenList}_1, \ldots, \mathsf{xtokenList}_{\mathsf{UpdCnt}[w_1]})$.
19 **end**

Algorithm 3: Search

Input: Encrypted database $\mathsf{EDB} = (TSet, XSet)$;
Client's $search\_token$.

Server:

1 Initialize returnList to an empty list.
2 **for** $j \leftarrow 1$ *to* $|\mathsf{stokenList}|$ **do**
3 Set $counter_j = 1$;
4 Set $c_j = TSet[\mathsf{stokenList}[j]]$;
5 Compute $\mu = \mathsf{SHVE.Query}(c_j, s)$;
6 **if** $\mu \in \mathcal{M}$ **then**
7 Set $(e_j, \alpha_j) = \mu$;
8 **for** $i \leftarrow 2$ *to* q **do**
9 Set $\mathsf{xtoken}_{j,i} = \mathsf{xtokenList}_j[i]$;
10 Compute $xtag_{j,i} = \mathsf{xtoken}_{j,i}^{\alpha_j}$;
11 **if** $XSet[xtag_{j,i}] = 1$ **then**
12 $counter_j = counter_j + 1$.
13 **end**
14 **end**
15 **if** $counter_j = q$ **then**
16 $\mathsf{returnList} = \mathsf{returnList} \cup \{(j, e_j)\}$.
17 **end**
18 **end**
19 **end**
20 **Return** returnList.
21 **end**

Client:

22 **for** $i \leftarrow 1$ *to* $|\mathsf{returnList}|$ **do**
23 Set $(j, e_j) = \mathsf{returnList}[i]$.
24 Compute $id = e_j \oplus F(K_T, w_i || j || 1)$.
25 Output document identifier id.
26 **end**
27 **end**

one of them. Later, the server runs $\mathsf{SHVE.Query}(c, s)$ and gets the result μ to check whether $\mu \in \mathcal{M}$. If so, it implies that the authorization of the client is enough for accessing this document. We set an example of the access control process in Fig. 2. Then the server checks whether that document involves all keywords the client wants to search, with corresponding $\mathsf{xtokenList}$. Only if both conditions are satisfied, an encrypted value involving id can be returned to the client. Finally, the client can decrypt that value and gain the document identifier.

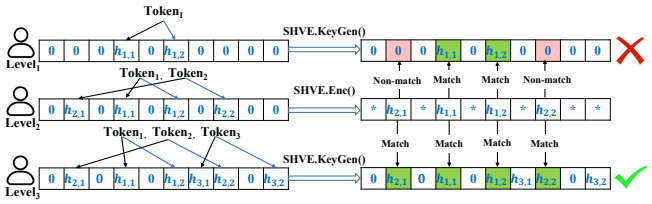


Fig. 2. An Example of Our Access Control Process

VI. System Analysis

A. Correctness

Here, we prove that values the server computes from the client's $\mathsf{xtoken}_{j,i}$ are equal to those values which have been stored in $XSet$. We have

$$\begin{aligned}&\mathsf{xtoken}_{j,i}^{\alpha_j}\\&= (g^{F_P(K_X,w_i)\cdot F_P(K_Z,w_1||j)})^{F_P(K_Y,id)\cdot[F_P(K_Z,w_1||j)]^{-1}}\\&= g^{F_P(K_X,w_i)\cdot F_P(K_Y,id)} = \mathsf{xtag}.\end{aligned}$$

The correctness of the equation above implies that a document whose identifier is id does include the keyword w_i. And that is exactly what we desire to implement. Therefore, the correctness of our search step holds.

B. Security Analysis

Forward Privacy. We have clearly described the definition and significance of forward privacy in Section II. Here, we show how forward privacy is guaranteed in our scheme. Assuming that at certain moment a client uploads a document including w_1, the uploaded ciphertext involves a counter value $\mathsf{UpdCnt}[w_1] = \mathsf{UpdCnt}[w_1] + 1$. Nevertheless, counter values that all previously generated search tokens involve are not updated, so these values will never match the newly uploaded document. Thus, our scheme is forward private.

Security Against Clients. As stated in our security model, some clients might collude with each other and clients may maliciously add values into Bloom filters.

To address the first issue, we make sure that distributed access tokens are continuous beginning with $token_1$, e.g., $token_1, token_2, token_3$. In this way, supposing that the token collections of two clients are respectively A and B, there must be either $A \subset B$ $(B \subset A)$ or $A = B$. Therefore, collusion is useless for at least one of them.

As for the second issue, when a client is going to map tokens into a Bloom filter, he/she is supposed to insert the hash values into corresponding positions, instead of 1. Thus, even if clients make malicious addition, there is no influence on calculating at all.

C. Performance Analysis

In the followings, we give a performance evaluation of our proposed scheme in terms of time and storage costs. All procedures are conducted in Linux operation system with an Intel Core i5-10400 CPU of 4.30 GHz and 16GB RAM. Moreover, we use the elliptic curve "MNT159" to implement the scheme in [9] (MCSE for short hereafter) for comparison[1]. Every experiment is conducted 100 times and we take average values as the final results.

Setup Computational Costs. Fig. 3 shows time costs in setup step of MCSE and our proposed scheme related to the number of access tokens. To be noted, for the sake of simplicity, we regard attributes in ABE as access tokens in the experiments, and "#keyword = 10" means that there are 10 keywords in every documents. Now that our scheme does not involve ABE, time costs hold much less than MCSE. Although we enlarge our Bloom filters according to access tokens, our time costs still maintain almost invisible growth when the number of access tokens increases.

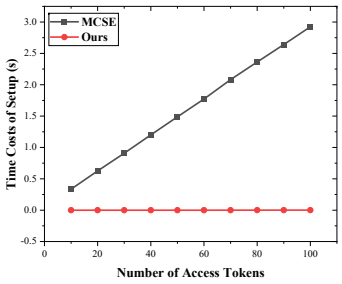
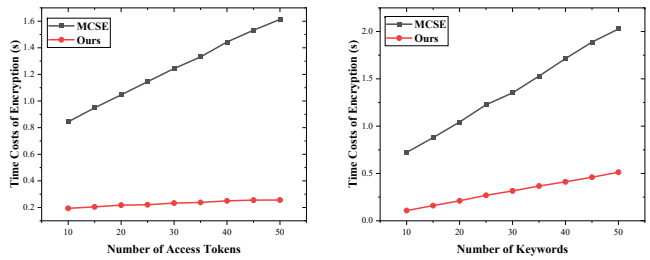


Fig. 3. Time Cost of Setup When #Keyword = 10

Encryption Efficiency. Here, Fig. 4 shows encryption efficiency of both schemes. Just like setup computational costs, more access tokens merely lead to slight influence over our scheme, but significantly increase the encryption time of MCSE. Although the number of keywords has obvious effects on both schemes, encryption of MCSE is clearly more affected. Ultimately, MCSE conducts several bilinear pairing operations for encryption. And, meanwhile, both two schemes need three exponentiation operations to encrypt one keyword.

Client's Costs over Search Tokens. We take time and storage costs of generating tokens to measure the overhead of one client. Here, the number of documents involving the first keyword w_1, i.e., $\mathsf{UpdCnt}[w_1]$, is fixed to 20. In Fig. 5, we record time and storage overhead under different keywords and access tokens. Obviously, both kinds of overhead in our proposed scheme are less compared to MCSE. When access tokens increase, overhead in our scheme remains nearly constant as the length of Bloom filters has little effect on this

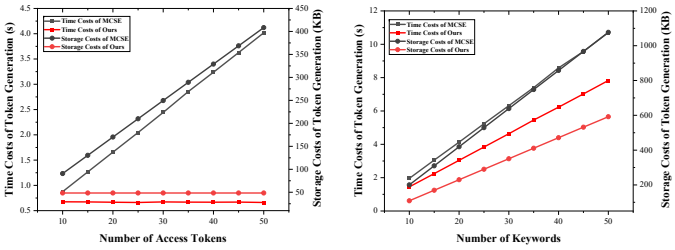
[1]There is no scheme in the exact same scenario as ours, so we have no choice but compare our scheme with an attribute-based SSE scheme in a similar multi-client scenario.



(a) Fixed #Keyword = 20 (b) Fixed #Tokens = 20

Fig. 4. Efficiency of Encryption

step. However, in Fig. 5b, we can see that the gap between two schemes is not as large as before when the number of keywords increases. It is easy to explain: when generating search tokens, MCSE needn't conduct bilinear pairing operations. Moreover, though its exponentiation operations are more complicated than those in our scheme, both schemes perform the same number of operations.



(a) Fixed #Keyword = 5 (b) Fixed #Tokens = 10

Fig. 5. Time and Storage Costs of Token Generation

Search Time. We present our results for search time in Table II. Our scheme certainly achieves higher time efficiency and roughly twice savings as MCSE. However, in MCSE, clients need another round of communication to get decryption keys for what they have received from the server, which is not presented in Table II. Therefore, the real processing time to get decrypted documents in our scheme is much less than that in MCSE.

TABLE II
SEARCH TIME IN TWO SCHEMES (S)

Number of Keywords and Access Tokens		MCSE	Ours
#Keywords=5	#Tokens=10	0.606849700	0.325721067
	#Tokens=20	0.607580033	0.328395567
	#Tokens=30	0.611077133	0.322247667
#Tokens=10	#Keywords=15	2.113462667	1.118631567
	#Keywords=25	3.700582933	1.924311633
	#Keywords=35	5.140796400	2.726146800

VII. CONCLUSION

In this paper, we proposed a searchable symmetric encryption scheme with efficient access control for dynamic multi-client scenarios. Firstly, we pointed out the overhead issues for fine-grained access control in symmetric scenarios. Later, we presented a scheme to achieve higher efficiency by utilizing Bloom filter and modified Symmetric Hidden Vector Encryption. Moreover, we claimed the significance of forward privacy in multi-client scenarios and, through the analysis of security, our proposed scheme indeed holds the forward privacy preservation. Finally, extensive experiments also indicate that the computation, storage and communication overhead in our scheme are lightweight, which implies our proposed scheme is practical in realistic multi-client scenarios.

ACKNOWLEDGMENT

The work is supported in part by the National Natural Science Foundation of China (NSFC) under Grant No. 61972371, and Youth Innovation Promotion Association of the Chinese Academy of Sciences (CAS) under Grant No. Y202093.

REFERENCES

[1] B. Varghese and R. Buyya, "Next generation cloud computing: New trends and research directions," *Future Generation Computer Systems*, vol. 79, pp. 849–861, 2018.

[2] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proceedings of the 2000 IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2000, pp. 44–55.

[3] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for boolean queries," in *Proceedings of the 33rd Annual Cryptology Conference (CRYPTO)*. Springer, 2013, pp. 353–373.

[4] S. Lai, S. Patranabis, A. Sakzad, J. K. Liu, D. Mukhopadhyay, R. Steinfeld, S.-F. Sun, D. Liu, and C. Zuo, "Result pattern hiding searchable encryption for conjunctive queries," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2018, pp. 745–762.

[5] S.-F. Sun, J. K. Liu, A. Sakzad, R. Steinfeld, and T. H. Yuen, "An efficient non-interactive multi-client searchable encryption with support for boolean queries," in *Proceedings of the 21st European Symposium on Research in Computer Security (ESORICS)*. Springer, 2016, pp. 154–172.

[6] L. Du, K. Li, Q. Liu, Z. Wu, and S. Zhang, "Dynamic multi-client searchable symmetric encryption with support for boolean queries," *Information Sciences*, vol. 506, pp. 234–257, 2020.

[7] S.-F. Sun, C. Zuo, J. K. Liu, A. Sakzad, R. Steinfeld, T. H. Yuen, X. Yuan, and D. Gu, "Non-interactive multi-client searchable encryption: Realization and implementation," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 1, pp. 452–467, 2022.

[8] H. Ling, K. Xue, D. Wei, and R. Li, "An efficient multi-user multi-keyword fuzzy search scheme over encrypted cloud storage," *Journal of University of Science and Technology of China*, vol. 51, no. 7, pp. 562–576, 2021.

[9] K. Zhang, M. Wen, R. Lu, and K. Chen, "Multi-client sub-linear boolean keyword searching for encrypted cloud storage with owner-enforced authorization," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 6, pp. 2875–2887, 2020.

[10] E. Stefanov, C. Papamanthou, and E. Shi, "Practical dynamic searchable encryption with small leakage," in *Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS)*. ISOC, 2014, pp. 72–75.

[11] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[12] S. Patranabis and D. Mukhopadhyay, "Forward and backward private conjunctive searchable symmetric encryption," in *Proceedings of the 28th Annual Network and Distributed System Security Symposium (NDSS)*. ISOC, 2021.

[13] Y. Zhang, J. Katz, and C. Papamanthou, "All your queries are belong to us: The power of file-injection attacks on searchable encryption," in *Proceedings of the 25th USENIX Security Symposium (USENIX Security)*. USENIX Association, 2016, pp. 707–720.