Towards Achieving Keyword Search over Dynamic Encrypted Cloud Data with Symmetric-Key Based Verification

Xinrui Ge, Jia Yu, Hanlin Zhang, Chengyu Hu, Zengpeng Li, Zhan Qin, Rong Hao

***Abstract*—Verifiable Searchable Symmetric Encryption, as an important cloud security technique, allows users to retrieve the encrypted data from the cloud through keywords and verify the validity of the returned results. Dynamic update for cloud data is one of the most common and fundamental requirements for data owners in such schemes. To the best of our knowledge, the existing verifiable SSE schemes supporting data dynamic update are all based on asymmetric-key cryptography verification, which involves time-consuming operations. The overhead of verification may become a significant burden due to the sheer amount of cloud data. Therefore, how to achieve keyword search over dynamic encrypted cloud data with efficient verification is a critical unsolved problem. To address this problem, we explore achieving keyword search over dynamic encrypted cloud data with symmetric-key based verification and propose a practical scheme in this paper. In order to support the efficient verification of dynamic data, we design a novel Accumulative Authentication Tag (AAT) based on the symmetric-key cryptography to generate an authentication tag for each keyword. Benefiting from the accumulation property of our designed AAT, the authentication tag can be conveniently updated when dynamic operations on cloud data occur. In order to achieve efficient data update, we design a new secure index composed by a search table ST based on the orthogonal list and a verification list VL containing AATs. Owing to the connectivity and the flexibility of ST, the update efficiency can be significantly improved. The security analysis and the performance evaluation results show that the proposed scheme is secure and efficient.**

***Index Terms*—searchable symmetric encryption, encrypted cloud data, verification, data dynamic, symmetric-key cryptography**

I. Introduction

X.R. Ge is with the College of Computer Science and Technology, Qingdao University, Qingdao 266071, China, with State Key Laboratory of Cryptology, Beijing 100878, China. E-mail:gxr193819@163.com.

(*Corresponding author: J. Yu*) J. Yu is with the College of Computer Science and Technology, Qingdao University, Qingdao 266071, China, with State Key Laboratory of Cryptology, Beijing 100878, China, and with State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China. E-mail:qduyujia@gmail.com.

H. L. Zhang is with the College of Computer Science and Technology, Qingdao University Qingdao 266071, China.

C. Y. Hu is with the School of Software, Shandong University, Jinan 250100, China.

Z. P. Li is with the the College of Computer Science and Technology, Qingdao University, Qingdao 266071, China.

Z. Qin is with the Electrical and Computer Engineering Department of the Texas University at San Antonio, USA.

R. Hao is with the College of Computer Science and Technology, Qingdao University Qingdao 266071, China.

SEARCHABLE Symmetric Encryption(SSE) is a practical way for users to securely retrieve the interested ciphertexts from the encrypted cloud data through keywords. It has become a hot research topic in cloud computing security and numerous SSE schemes have been proposed. Nonetheless, most of them only consider realizing keyword search over static encrypted cloud data. In practice, the data stored on the cloud server might often need to be updated(added, deleted or modified) by data owners. Therefore, it is necessary to design SSE schemes supporting dynamic update for cloud data. Kamara et al. [1] proposed a SSE scheme supporting data dynamic update. This scheme designs a search table by extending the inverted index to realize the sublinear search, and adopts a search array and a deletion array with other free storage spaces to achieve data dynamics. Guo et al. [2] proposed a dynamic SSE scheme, in which an inverted index is used to record the locations of keywords. The update table and the update list make the scheme support data dynamics. In addition, some other dynamic keyword search schemes [3–5], which adopt tree-based (i.e. KBB tree, KRB tree and B^+ tree) index structure, have also been proposed.

All of the above schemes do not consider the verification of the returned search results from the cloud server. In practice, the cloud server may return invalid results to the data user for saving computational resources or the software/hardware malfunctions. Therefore, the data user should be able to check the authenticity of the returned search results. Kurosawa et al. [6] introduced two verifiable dynamic SSE schemes. The first scheme, which adopts the Message Authentication Code(MAC) to verify the search results, works fine with static cloud data. However, when the data is updated, the data user cannot verify whether the returned results are newly updated or not. If the cloud server returns a result including a pair of non-updated file and MAC, it can pass the verification. So it is unable to defend against the replay attack [6]. In order to solve this problem, the second scheme uses the timestamp functionality of the RSA accumulator to obtain the verifiability of search results. It generates accumulators for all files and for all index vector bits, which are kept by the data owner. If the cloud server returns the non-updated results, the data owner can detect them with the newest accumulators. The follow-up schemes [7–9] utilize RSA accumulator to achieve the verification for search results and the dynamic update for cloud data. Scheme [10] leverages bilinear-map accumulator to achieve the result verification and the data dynamics. However, RSA accumulator and bilinear-map accumulator are both based on

asymmetric-key cryptography involving time-consuming operations. The overhead of verification will become a significant computation burden for resource-constrained devices in these schemes. Therefore, it is a critical unsolved problem to realize efficient verification for dynamic SSE schemes.

In this paper, we explore how to achieve keyword search over dynamic encrypted cloud data with symmetric-key based verification. The contributions of this paper can be summarized as follows:

- In order to support the efficient verification of dynamic data, we design a novel symmetric-key based Accumulative Authentication Tag (**AAT**) to generate an authentication tag for each keyword. Benefiting from the accumulation property of our designed **AAT**, the authentication tag can be conveniently updated when dynamic operations on cloud data occur. The proposed **AAT** is collision resistant, that is, it is computational-difficult for any adversary to find different messages with the same tag. It also can resist the replay attack to prevent the cloud server from returning the old data that actually has been updated.
- In order to realize efficient data update, we design a new secure index composed by a search table ST and a verification list VL. ST is based on the orthogonal list and VL is a singly linked list. For each keyword, we construct a linked list with the same length aiming at hiding the frequency of each keyword. When performing modification operations, the cloud server can fleetly find the index nodes related to the modified files. When some files need to be added or deleted, the secure index can be conveniently enlarged or reduced. Owing to the connectivity and flexibility of ST, the update efficiency can be significantly improved.
- Based on the above technique and structure, we design the first keyword search scheme over dynamic encrypted cloud data with symmetric-key based verification. We give the security analysis of the proposed scheme and conduct the performance comparison with other work in terms of the search token generation efficiency, verification efficiency and update efficiency. The results shows that the proposed scheme is secure and efficient.

Organization. The rest of this paper is organized as follows. We summarize the related work in Section 2. We formally define the problem formulation including system model, design goals, and definitions in Section 3. The construction of our scheme is presented in Section 4. We give the security analysis in Section 5. In Section 6, we evaluate the proposed scheme through extensive experiments and give the performance comparison with other work. After that, we conclude this paper in Section 7.

II. Related work

In recent years, cloud computing has been applied to securely perform various tasks, such as healthcare monitoring [11], deep packet inspection [12] and key updates [13]. Cloud storage is universally viewed as one of the most important services of cloud computing. Although cloud storage provides great benefit to users, it brings some new security challenges. Firstly, users may worry about whether their data is intactly stored in the cloud because the cloud data is out of their physical control. In order to solve this problem, some cloud storage auditing schemes [14–16] are proposed to check the integrity of cloud data. In addition, users usually need to encrypt the data for keeping the privacy before outsource them to the cloud. It makes performing keyword search over encrypted cloud data become a new challenge. In order to address this issue, searchable encryption is proposed, which allows users to selectively retrieve cipher documents stored in the cloud by keyword-based search. Compared with searchable public key encryption [17, 18], searchable symmetric encryption draws more attention owing to its high efficiency.

Static SSE. Song et al. [19] firstly proposed the searchable symmetric encryption scheme, in which a special two-layered encryption structure is constructed to encrypt each keyword. Goh et al. [20] proposed a keyword search scheme over encrypted cloud data based on the Bloom filter. Curtmola et al. [21] proposed two efficient keyword search schemes(SSE-1 and SSE-2) over encrypted cloud data. These schemes can realize sublinear search, that is, the search cost is proportional to the number of the files matching the queried keyword. Cao et al. [22] proposed a privacy-preserving multi-keyword ranked search scheme over encrypted cloud data by utilizing the similarity measure of "Coordinate matching" and "inner product similarity". In addition, some other static SSE schemes, such as semantic search scheme [23, 24], similarity search scheme [25–28], ranked keyword search schemes [29–32], central keyword-based semantic extension search scheme [33], and keyword search scheme supporting deduplication [34], have also been proposed.

Dynamic SSE. In order to support data dynamic update, some dynamic SSE schemes [1–4, 35–38] have been proposed. Kamara et al. [1] proposed a dynamic SSE scheme by extending the inverted index approach. This scheme can achieve sublinear search and CKA2-security. Subsequently, they proposed another dynamic SSE scheme [4] based on *keyword red-black* tree index structure. This scheme supports parallel keyword search as well as parallel addition and deletion of files. Naveed et al. [38] presented a dynamic SSE scheme via blind storage. Blind storage allows a data owner to store files on a cloud server in such a way that the cloud server does not learn the number of files. Xia et al. [3] proposed a dynamic keyword search scheme over encrypted cloud data based on the tree-based index structure, which can support multi-keyword rank. Guo et al. [2] proposed a dynamic SSE scheme based on the inverted index. It enables the data user to search several phrases in a query request. Also, their proposed scheme supports the sorting of the search results.

Verifiable SSE. In order to prevent the cloud server from returning the invalid search results, Chai et al. [39] firstly proposed the verifiable keyword search scheme over encrypted cloud data. In order to make the scheme with the "verifiable searchability", the cloud server is required to provide a proof along with the returned results. Kurosawa et al. [40] have shown how to construct a universally composable (UC)-secure verifiable SSE scheme. Jiang et al. [41] proposed a verifiable multi-keyword ranked search scheme over encrypted cloud

data. A special data structure named QSet is constructed to achieve efficient keyword search in this scheme. In order to support dynamic data update for verifiable SSE schemes, Sun et al. [10] proposed a verifiable dynamic conjunctive keywords search scheme based on the bilinear-map accumulator and the accumulation tree. Zhu et al. [9] introduced a verifiable and dynamic fuzzy keyword search scheme based on the inverted index. Liu et al. [7] presented a verifiable dynamic keyword search scheme supporting the search results rank. This scheme and scheme [9] both leverage RSA accumulator to realize the results verification and the data dynamics. The verification techniques used in above verifiable and dynamic schemes are all based on asymmetric-key cryptography, which involves time-consuming operations. As a result, the verification efficiency is very low in these schemes.

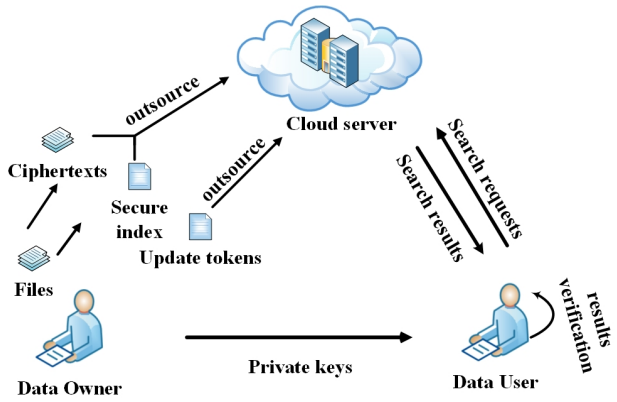


Fig. 1: System model

III. Problem Formulation

A. System Model

As shown in Fig.1, the system model consists of three entities: data owner, data user and cloud server.

- Data owner: He encrypts his plain files and constructs a secure index with private keys. He uploads the ciphertexts and the secure index to the cloud server. When the data owner wants to update files, he generates the update tokens locally and sends them to the cloud server.
- Data user: He is authorized by the data owner who shares the private keys with him. When he wants to search the files containing the interested keywords, he sends the search requests to the cloud server. After the data user receives the search results from the cloud server, he can verify the validity of the results.
- Cloud server: It stores the ciphertexts and the secure index from the data owner. Upon receiving the search requests from the data user, it performs search operation over the secure index, and returns the search results. In addition, upon receiving the update information from the data owner, it updates the secure index and the related ciphertexts.

In this model, the data owner and the data user are assumed to be always trusted. That is, the data owner honestly encrypts files and builds a secure index. The data user honestly generates the search request for the queried keyword. The cloud server is regarded as an untrusted entity. It is allowed to learn which encrypted files contain the queried keyword by performing the search operation. However, it might try to learn more valuable information from the encrypted files, the secure index, and the search trapdoors. For example, it might try to find which files contain two queried keywords and which keywords have changed in the modified file. Besides, the cloud server may return invalid or non-updated search results to the data user for saving computation cost or other reasons.

B. Design Goals

To address the above challenges, we aim at making the proposed scheme meet the following goals.

- *Supporting efficient keyword search over encrypted cloud data.* The scheme should retrieve all matching files for the search token and also should achieve sublinear search efficiency.
- *Supporting efficient data dynamic update.* The scheme should support efficient data dynamic update, such as modification, addition or deletion.
- *Supporting efficient search results verification.* The user should be able to verify the correctness of the search results from the cloud server. The verification should be based on symmetric-key cryptography and not involve any complex operation.
- *Achieving privacy preserving.* The scheme should guarantee that the cloud server cannot learn any useful information from the stored ciphertexts and secure index beyond primitive authorizations. For example, the scheme should not expose which keywords appear at high-frequency and which keywords appear at low-frequency. When a file is added or deleted, the number of keywords in it should not be leaked to the cloud server. The primitive authorizations may include the information of encrypted files (i.e. files sizes and contents), access pattern (i.e. the search results) and the repetition of search tokens (i.e. the preciously queried trapdoors).
- *Achieving replay attack resistance.* When the data needs to be updated, the cloud server might not perform the update operation for saving computation cost or software/hardware failures. As a result, it might return the non-updated results to the data user. The scheme should be able to verify whether the returned results are newly updated or not.

C. Definitions

1) Scheme Definition:

Definition 1 (Verifiable and Dynamic SSE (VDSSE) scheme): A verifiable and dynamic SSE scheme includes eight polynomial-time algorithms i.e. Setup, IndexBuild, GenToken, Search, Verify, Dec, UpToken and Update. These algorithms are defined as follows.

- $K \leftarrow$ ***Setup*** (1^λ) is the probabilistic key generation algorithm run by the data owner. It takes a random secure parameter λ as input, and outputs a private key set K.

- $(\mathcal{I}, \mathcal{C}) \leftarrow$ ***IndexBuild*** $(K, \mathcal{F}, \mathcal{W})$ is the probabilistic index building algorithm run by the data owner. It takes the private key set K, the file set $\mathcal{F}$ and the keyword set $\mathcal{W}$ as input, and outputs a secure index $\mathcal{I}$ and a ciphertext collection $\mathcal{C}$.
- $T_w \leftarrow$ ***GenToken*** (K, w) is the (possibly probabilistic) trapdoor generation algorithm run by the data user. It takes the private key set K and the queried keyword w as input, and outputs the trapdoor T_w.
- $(\text{AAT}_S, C(w)) \leftarrow$***Search***$(T_w, \mathcal{I}, \mathcal{C})$ is the deterministic search algorithm run by the cloud server. It takes the trapdoor T_w, the secure index $\mathcal{I}$ and the ciphertext set $\mathcal{C}$ as input, and outputs a ciphertext set $C(w)$ and an authentication tag AAT_S.
- $(accept, reject) \leftarrow$ ***Verify***$(K, T_w, C(w), \text{AAT}_S)$ is the deterministic verification algorithm run by the data user. It takes the private key set K, the trapdoor T_w, the set $C(w)$ and the authentication tag AAT_S as input, and outputs "$accept$" or "$reject$".
- $F(w) \leftarrow$ ***Dec*** $(K, C(w))$ is the deterministic decryption algorithm run by the data user. It takes the private key set K and the set $C(w)$ as input, and outputs a plaintext set $F(w)$.
- $\tau \leftarrow$ ***UpToken*** $(K, F, (F'))$ is the (possibly probabilistic) update tokens generation algorithm run by the data owner. When modifying a file, it takes as input the original file F, the new file F' and the private key set K, and outputs the modify token $\tau_m = (\tau_{m_0}, \tau_{m_1}, ..., \tau_{m_{|\mathcal{W}|}}, \tau_{m_{|\mathcal{W}|+1}})$, where τ_{m_0} is the identifier of the modified file, $\tau_{m_i} = \{< modval >, \text{"}mod\text{"}\}(1 \leq i \leq |\mathcal{W}|)$ and $\tau_{m_{|\mathcal{W}|+1}} = C'$. When adding a file, it takes as input a new file F and the private key set K, and outputs the add token $\tau_a = (\tau_{a_0}, \tau_{a_1}, ..., \tau_{a_{|\mathcal{W}|}}, \tau_{a_{|\mathcal{W}|+1}})$, where τ_{a_0} is the identifier of the added file, $\tau_{a_i} = \{< addval >, \text{"}add\text{"}\}(1 \leq i \leq |\mathcal{W}|)$ and $\tau_{a_{|\mathcal{W}|+1}} = C$. When deleting a file, it takes as input the deleted file F and the private key set K, and outputs the delete token $\tau_d = (\tau_{d_0}, \tau_{d_1}, ..., \tau_{d_{|\mathcal{W}|}}, \tau_{d_{|\mathcal{W}|+1}})$, where τ_{d_0} is the identifier of the deleted file, $\tau_{d_i} = \{< delval >, \text{"}del\text{"}\}(1 \leq i \leq |\mathcal{W}|)$ and $\tau_{d_{|\mathcal{W}|+1}} = C$.
- $(\mathcal{I}', \mathcal{C}') \leftarrow$ ***Update*** $(\tau, \mathcal{I}, \mathcal{C})$ is the deterministic update algorithm run by the cloud server. It takes as input the update token τ, the secure index $\mathcal{I}$, and the ciphertext collection $\mathcal{C}$. It outputs a new secure index $\mathcal{I}'$, and a new ciphertext collection $\mathcal{C}'$.

2) Security Definitions:

Definition 2 (Update reliability): A verifiable and dynamic SSE scheme guarantees the update reliability, that is, resists the replay attack if for any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, the probability of successfully giving the non-updated search results that can pass the verification is negligible for any $(\mathcal{F}, \mathcal{W}, \mathcal{I})$ and trapdoor T_w.

Specifically, as the cloud server is untrusted, it may not update the secure index and the ciphertext collection when the data owner sends the update request. When the cloud server receives the search request from the data user, it may return the non-updated data to the data user. The data user should be able to detect whether the results are the latest ones. Given an updated and valid $C(w)$ and an authentication tag AAT_S for a search token, the adversary wins if he can give a non-updated value ($C'(w)$, AAT'_S) that passes the ***Verify*** algorithm.

Definition 3 (Verifiability): A verifiable and dynamic SSE scheme satisfies verifiability if for any PPT adversary $\mathcal{A}$, the probability of successfully forging the search results is negligible for any $(\mathcal{F}, \mathcal{W}, \mathcal{I})$ and trapdoor T_w.

Specifically, because the cloud server is untrusted, it may return incorrect search results to the data user. The data user should be able to detect such misbehavior to guarantee the validity of search results. Given a valid $C(w)$ and an authentication tag AAT_S for a search token, the adversary wins if he can give a forgery ($C'(w)$, AAT'_S) that passes the ***Verify*** algorithm.

Definition 4 (CKA2-security): Let Π = (Setup, IndexBuild, GenToken, Search, Verify, Dec, UpToken and Update) be a verifiable and dynamic SSE scheme and $\mathcal{L} = \{\mathcal{L}_{setup}, \mathcal{L}_{search}, \mathcal{L}_{update}\}$ be a leakage function set. Let $\mathcal{A}$ be an adversary and $\mathcal{S}$ be a simulator. For $\mathcal{A}$ and $\mathcal{S}$, we define the following experiments:

- $\textbf{Real}_{\mathcal{A}}(\lambda)$: the challenger runs ***Setup***(1^λ) to generate the private key set K. $\mathcal{A}$ outputs the file set $\mathcal{F}$ and the keyword set $\mathcal{W}$. The challenger computes $(\mathcal{I}, \mathcal{C}) \leftarrow$ ***IndexBuild***$(K, \mathcal{F}, \mathcal{W})$, and sends $\mathcal{I}, \mathcal{C}$ to $\mathcal{A}$. $\mathcal{A}$ makes a polynomial number of adaptive queries. For each query, $\mathcal{A}$ receives a search token $T_w \leftarrow$ ***GenToken*** (K, w), or a modify token τ_m, or an add token τ_a, or a delete token τ_d from the challenger, where $\tau_m, \tau_a, \tau_d \leftarrow$***UpToken*** $(K, F, (F'))$. Finally, $\mathcal{A}$ returns a bit b that is output by the experiment.
- $\textbf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda)$: $\mathcal{A}$ outputs the file set $\mathcal{F}$ and the keyword set $\mathcal{W}$. Given $\mathcal{L}_{setup}(\mathcal{F}, \mathcal{W})$, $\mathcal{S}$ generates a pair $(\mathcal{I}', \mathcal{C}')$ and sends it to $\mathcal{A}$. $\mathcal{A}$ makes a polynomial number of adaptive queries. For each query, $\mathcal{S}$ is given $\mathcal{L}_{search}(w)$, or $\mathcal{L}_{update}(F)$. Meanwhile, $\mathcal{S}$ simulates and sends a search token T'_w, or a modify token τ'_m, or an add token τ'_a, or a delete token τ'_d to $\mathcal{A}$. Finally, $\mathcal{A}$ returns a bit b that is output by the experiment.

We say Π is $(\mathcal{L}_{setup}, \mathcal{L}_{search}, \mathcal{L}_{update})$-secure against adaptive chosen-keyword attacks(CKA2) if for any PPT adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that

$$|\Pr[\textbf{Real}_{\mathcal{A}}(\lambda) = 1] - \Pr[\textbf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda) = 1]| \leq \text{negl}(\lambda),$$

where negl stands for a negligible function.

IV. Verifiable and Dynamic SSE (VDSSE) scheme

For convenience, we summarize some common notations throughout the paper shown in Table.1.

A. Technical Overview

1) Accumulative Authentication Tag: Some existing verifiable literatures [41–44] adopt the Message Authentication Code(MAC) to achieve verification for search results. However, in dynamic SSE scenarios, MAC cannot be used to verify whether the results returned from the cloud are newly updated or not, that is, it cannot resist the replay attack [6]. To address this problem, in this paper, we design a novel and efficient

TABLE I: **Some Frequently used Symbols and Descriptions**

Notations	Descriptions
N	The number of files
n	The number of keywords
$\mathcal{W}$	The keyword set
w_i	The i-th keyword in $\mathcal{W}$
$\mathcal{F}$	The file set
F_j	The j-th file in $\mathcal{F}$
$\mathcal{C}$	The encrypted file set
C_j	The j-th encrypted file in $\mathcal{C}$
$F(w_i)$	The plain file set containing w_i
$C(w_i)$ or S_i	The encrypted file set containing w_i
ST	The search table
VL	The verification list
$\mathcal{I}$	The secure index
T_{w_i}	The search token(trapdoor) of w_i
τ_a	The add token
τ_m	The modify token
τ_d	The delete token
L_{w_i}	The row list related to w_i
L_{f_j}	The column list related to F_j
w_{ij}	The 0/1 bit denoting whether F_j contains w_i
v_j	The update time for file F_j
V	The global update number
Ew_{ij}	The ciphertext of (w_{ij}, v_j)
AAT_{S_i}	The authentication tag for w_i
p	The large prime

Accumulative Authentication Tag(**AAT**) based on symmetric-key cryptography to realize the results verification supporting data update.

Firstly, we interpret one variable and some functions used in **AAT** including the global update number V, the pseudo-random permutation function(PRP) π and the pseudo-random functions(PRFs) f and P. V is used to denote the total update times for all files and kept by the data owner. When a file is updated, V is updated to $V+1$. With the current V, the data user can verify whether the search results are newly updated or non-updated to resist the replay attack. $\pi : \{0,1\}^* \times K_{prp} \rightarrow \{0,1\}^l$ is used to generate the keyword permutation $\pi_{K_{prp}}(w)$ with the key K_{prp}, where $l = max\{|w|| w \in \mathcal{W}\}$. $f : \{0,1\}^* \times K_{prf} \rightarrow \{0,1\}^n$ is used to encrypt the keyword permutation $\pi_{K_{prp}}(w_i)$ and the global update number V with the key K_{prf}, that is, $f_{K_{prf}}(\pi_{K_{prp}}(w))$ and $f_{K_{prf}}(V)$. $P : \{0,1\}^* \times K_p \rightarrow \{0,1\}^{logN}$ is used to generate the random value α_j $(1 \leq j \leq N)$ for each file F_j, that is, $\alpha_j \leftarrow P_{K_p}(j)$. The above π, f and P are all polynomial-time computable functions, which cannot be distinguished from random functions by any probabilistic polynomial-time adversary.

To simplify the description, we assume all files have the same size[1], and write $\pi(m), f(m)$ and $P(m)$ instead of $\pi_{K_{prp}}(m), f_{K_{prf}}(m)$ and $P_{K_p}(m)$. Let S be the encrypted file set containing the keyword w and Δ be the identifier set of encrypted files in S. For each encrypted file C_j in S, we divide it into b blocks $M_{jt} \in \mathbb{Z}_p (1 \leq t \leq b)$. For keyword w and the current global update number V, Accumulative Authentication Tag(**AAT**) is defined as follows:

$$\text{AAT}_S = f(\pi(w)) + f(V) \ + \sum_{j \in \Delta} \sum_{t=1}^{b} \alpha_j M_{jt},$$

where $\alpha_j \leftarrow P(j)$.

AAT has the following properties.

Accumulation. AAT can accumulate the files containing a keyword to generate one authentication tag.

Firstly, assume there is only one encrypted file C_{k_1} in S for keyword w and C_{k_1} is composed by $M_{k_1 t} \in \mathbb{Z}_p (1 \leq t \leq b)$. For keyword w and the current global update number V, **AAT** is denoted as $\text{AAT}_S = f(\pi(w)) + f(V) + \sum_{t=1}^{b} \alpha_{k_1} M_{k_1 t}$.

Then assume there are total two encrypted files C_{k_1} and C_{k_2} in S. C_{k_1} is composed by $M_{k_1 t} \in \mathbb{Z}_p$ and C_{k_2} is composed by $M_{k_2 t} \in \mathbb{Z}_p$. Now, **AAT** is denoted as $\text{AAT}_S = f(\pi(w)) + f(V) + \sum_{t=1}^{b} \alpha_{k_1} M_{k_1 t} + \sum_{t=1}^{b} \alpha_{k_2} M_{k_2 t}$, that is, $\text{AAT}_S = \text{AAT}_{S'} + \sum_{t=1}^{b} \alpha_{k_2} M_{k_2 t}$, where $S' = S/\{C_{k_2}\}$.

Similarly, assume there are total z encrypted files $C_{k_1}, C_{k_2}, ..., C_{k_z}$ in S. Then, **AAT** is denoted as $\text{AAT}_S = f(\pi(w)) + f(V) + \sum_{j \in \Delta} \sum_{t=1}^{b} \alpha_j M_{jt} = f(\pi(w)) + f(V) + \sum_{t=1}^{b} \alpha_{k_1} M_{k_1 t} + \cdots \sum_{t=1}^{b} \alpha_{k_z} M_{k_z t}$, that is, $\text{AAT}_S = \text{AAT}_{S'} + \sum_{t=1}^{b} \alpha_{k_z} M_{k_z t}$, where $S' = S/\{C_{k_z}\}$.

Update. Owing to the accumulation property, **AAT** can support data update conveniently. For keyword w, assume there are z encrypted files $C_{k_1}, C_{k_2}, ..., C_{k_z}$ in S. Let Δ be the identifier set of files in S and $M_{k_j t} \in \mathbb{Z}_p (1 \leq t \leq b)$ be the t-th block of C_{k_j} $(1 \leq j \leq z)$. When the data is updated, V will be updated to $V+1$. The data update includes the following three cases.

Case 1: The file F_k needs to be modified to F'_k.

- If both F_k and F'_k contain w, then update AAT_S = $\text{AAT}_S + f(V+1) - f(V) + \sum_{t=1}^{b} \alpha_k M'_{kt} - \sum_{t=1}^{b} \alpha_k M_{kt}$, where $M'_{kt} \in \mathbb{Z}_p$ is the t-th block of C'_k. Modify C_k to C'_k in S.
- If F_k contains w while F'_k does not, then update AAT_S = $\text{AAT}_S + f(V+1) - f(V) - \sum_{t=1}^{b} \alpha_k M_{kt}$. Delete C_k from S and delete k from Δ.
- If F_k does not contain w while F'_k does, then update AAT_S = $\text{AAT}_S + f(V+1) - f(V) + \sum_{t=1}^{b} \alpha_k M'_{kt}$. Add C'_k into S and add k into Δ.
- If both F_k and F'_k do not contain w, then update AAT_S = $\text{AAT}_S + f(V+1) - f(V)$.

Case 2: The file $F_{k_{z+1}}$ needs to be added.

- If $F_{k_{z+1}}$ does not contain w, then update AAT_S = $\text{AAT}_S + f(V+1) - f(V)$.
- If $F_{k_{z+1}}$ contains w, then update AAT_S = $\text{AAT}_S + f(V+1) - f(V) + \sum_{t=1}^{b} \alpha_{k_{z+1}} M_{(k_{z+1})t}$. Add $C_{k_{z+1}}$ to S and add its identifier to Δ.

Case 3: The file F_k needs to be deleted.

- If F_k does not contain w, then update $\text{AAT}_S = \text{AAT}_S + f(V+1) - f(V)$.
- If F_k contains w, then update $\text{AAT}_S = \text{AAT}_S + f(V+1) - f(V) - \sum_{t=1}^{b} \alpha_k M_{kt}$. Delete C_k from S and delete k from Δ.

[1]Actually, our technique can easily support the situation that the files have different sizes.

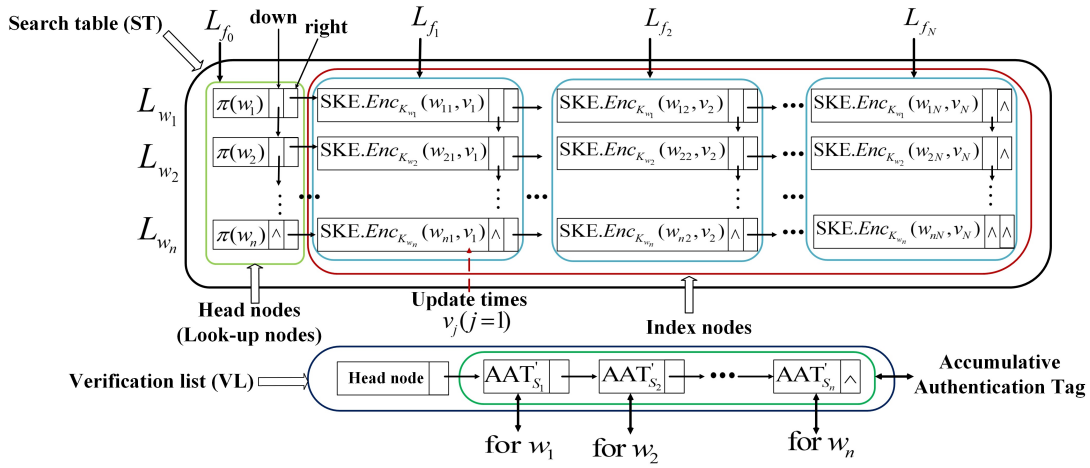


Fig. 2: The secure index structure

Collision resistance. For keyword w, assume the correct encrypted file set is $S = \{C_{k_1}, ..., C_{k_z}\}$ and the correct tag is AAT_S. We will prove that if an adversary $\mathcal{A}$ can forge a set S' ($S' \neq S$) satisfying $\text{AAT}_{S'} = \text{AAT}_S$, then $\mathcal{A}$ breaks the security of PRF P.

Let $S' = \{C'_{k_1}, ..., C'_{k_{z'}}\}$. Without loss of generality, assume $z' \leq z$. Let Δ be the identifier set of files in S and Λ be the identifier set of the same files between S and S'. If $k_j \in \Lambda$, $C'_{k_j} = C_{k_j}$; otherwise, $C'_{k_j} \neq C_{k_j}$ $(1 \leq j \leq z')$ or $C_{k_j} \in S$ $(z' < j \leq z)$. Since AAT_S - $\text{AAT}_{S'}$ = 0, we have $\sum_{k_j \in \Delta/\Lambda} \sum_{t=1}^{b} \alpha_{k_j} M''_{k_j t} = 0$, where $M''_{k_j t} = M_{k_j t} - M'_{k_j t}$ $(1 \leq j \leq z')$ and $M''_{k_j t} = M_{k_j t}$ $(j > z')$.

For simplicity, assume $\Lambda = \{k_1, k_2, ..., k_r\}$ $(r \leq z')$ and $\Delta/\Lambda = \{k_{r+1}, ..., k_z\}$. We have

$$\alpha_{k_{r+1}} \sum_{t=1}^{b} M''_{k_{r+1} t} + \cdots + \alpha_{k_z} \sum_{t=1}^{b} M''_{k_z t} = 0 \quad (1)$$

$\mathcal{A}$ is allowed to query P oracle up to $z-1$ times. Without loss of generality, assume $\mathcal{A}$ has queried $\alpha_{k_1}, ..., \alpha_{k_{z-1}}$ from P oracle. According to equation (1), $\mathcal{A}$ can compute

$$\alpha_{k_z} = \frac{-(\alpha_{k_{r+1}} \sum_{t=1}^{b} M''_{k_{r+1} t} + \cdots + \alpha_{k_{z-1}} \sum_{t=1}^{b} M''_{k_{z-1} t})}{\sum_{t=1}^{b} M''_{k_z t}}. \quad (2)$$

Note that, the denominator $\sum_{t=1}^{b} M''_{k_z t}$ is zero only with negligible probability. It means the adversary $\mathcal{A}$ can give a valid forgery $P_{K_p}(k_z)$ without knowing the key K_p with high probability, which is contradictory to the security of PRF P. Therefore, our proposed **AAT** satisfies collision resistance.

2) The Secure Index Structure: In our work, we design a new secure index consisting of a search table ST and a verification list VL.

Let SKE = (Gen, Enc, Dec) be a symmetric encryption scheme. Algorithm Gen is used to generate the private keys. Algorithm $Enc(Dec)$ is used to encrypt(decrypt) files and some information in the secure index. Let $H : \{0,1\}^* \times K_h \rightarrow \mathbb{Z}_q$ be an HMAC with the key K_h. We use it to generate the key K_{w_i} for each keyword w_i by computing $K_{w_i} \leftarrow H_{K_h}(w_i)$. For simplicity, we write $H(m)$ instead of $H_{K_h}(m)$. We encrypt the index vector bit w_{ij} and the update times v_j with the key K_{w_i} by computing $Ew_{ij} \leftarrow \text{SKE}.Enc_{K_{w_i}}(w_{ij}, v_j)$, where w_{ij} denotes whether the file F_j contains the keyword w_i and v_j represents the update times for F_j. If F_j contains w_i, $w_{ij} = 1$; otherwise, $w_{ij} = 0$. Initially, set $v_j = 1$.

ST is based on the orthogonal list. It consists of n row lists and $N+1$ column lists. Each row list is related to each keyword. The i-th row list is denoted as $L_{w_i}(1 \leq i \leq n)$, which consists of one head node and N index nodes. Each column list is related to each file. The j-th column list is denoted by $L_{f_j}(0 \leq j \leq N)$. The head node, also named look-up node, records the location information of each keyword. Let $\pi(w_i)$ be the value of the head node(look-up node), which helps the cloud server locate the position of the queried keyword. Each index node stores Ew_{ij}. When a file is modified, the state of some keywords might change, i.e, some are in the previous file but not in the modified file, or some are not in the previous file but in the modified file. To prevent the cloud server from knowing which keywords have changed between the new file and the previous file, we will update all index nodes in the related column list. So we introduce the update times v_j and make it increase with the file update. All the values of index nodes in the related column list will be modified. As a result, the cloud server will not learn which keywords have changed in the new file. The privacy of the updated files can be preserved in this way.

VL is a singly linked list containing one head node and n verification nodes. The data field of the head node is empty and the pointer field stores the address of the next node. The verification node stores the authentication tag AAT_S generated by **AAT**.

We show the detailed secure index structure in Fig.2.

B. Construction of VDSSE scheme

In this section, we give the overview of core algorithms and the detailed description of the proposed scheme, and illustrate

some simple examples to articulate the update operations.

1) Overview of core algorithms: In algorithm ***IndexBuild***, the data owner builds the secure index $\mathcal{I}$ =(ST,VL). In ST, each row list $L_{w_i}(1 \leq i \leq n)$ is associated with one keyword w_i. The head node of each row list stores the keyword permutation $\pi(w_i)$ as the address of this list. All head nodes are linked to the first column list L_{f_0}, which are used as look-up nodes for cloud server. The index node of each row list stores the ciphertext Ew_{ij} = SKE.$Enc_{K_{w_i}}(w_{ij}, v_j)$ $(1 \leq j \leq N)$ related to an index vector bit w_{ij} and the update times v_j. All index nodes in the same column are linked to the column list L_{f_j}, which corresponds to the file F_j. For each keyword w_i, the authentication tag AAT_{S_i} stored in the index node of VL is computed based on the accumulation property of **AAT**. When the data user would like to search files containing the interested keyword, he generates the trapdoor through algorithm ***GenToken***. The cloud server can perform search operation through algorithm ***Search***. In algorithm ***Verify***, the data user computes the authentication tag for returned ciphertexts, and checks whether these ciphertexts are correct according to the authentication tag.

In algorithm ***UpToken***, the data owner generates update tokens for the updated files. Each token is composed by $n + 2$ elements. The first element denotes the identifier of the updated file and the last element denotes the ciphertext of the updated file. Each middle element includes the values of updated index nodes in ST and the update value of AAT in VL. Owing to the connectivity and the flexibility of orthogonal list, the cloud server can update the secure index efficiently in algorithm ***Update***. In **modify operation**, the cloud server replaces the value of each index node related to the updated file with the new one in ST and updates AAT value in VL according to the update value. In **add operation**, the cloud server adds a new column list in ST and updates AAT value in VL according to the update value. In **delete operation**, the column list related to this file in ST is deleted directly. The cloud server only needs to update AAT value in VL. Owing to the accumulation and the update property of AAT, the values of index nodes in VL can be conveniently updated.

2) Scheme construction: The detailed scheme description are as follows.

Setup(1^λ): Generate a private key set $K = \{K_{enc}, K_{prf}, K_{prp}, K_h, K_p\}$ by inputting a secure parameter λ.

IndexBuild$(K, \mathcal{F}, \mathcal{W})$**:**

1) Initialization:
   a) scan $\mathcal{F}$ to extract all different keywords and build $\mathcal{W}$;
   b) initialize the update times v_j to 1 for each file F_j and the global update number V to 1;
   c) compute $\mathcal{C}$ =SKE.$Enc_{K_{enc}}(\mathcal{F})$ and $\alpha_j = P(j)$ for each encrypted file $C_j \in \mathcal{C}$ $(1 \leq j \leq N)$;
   d) divide each C_j into b blocks $M_{jt} \in \mathbb{Z}_p$ $(1 \leq t \leq b)$.
2) Build ST:
   a) for each keyword $w_i \in \mathcal{W}$ $(i = 1 \to n)$:
      - construct a linked list L_{w_i} with one head node and N index nodes;
      - compute $\pi(w_i)$ as the value of head node and $K_{w_i} = H(w_i)$;
      - for $j = 1 \to N$:
        - if F_j contains w_i, then set $w_{ij} = 1$; else, set $w_{ij} = 0$;
        - compute Ew_{ij} =SKE.$Enc_{K_{w_i}}(w_{ij}, v_j)$ as the value of the j-th index node.
   b) Chain all nodes in the same column list together to obtain a column list L_{f_j}.
3) Build VL:
   a) construct a singly linked list L' with a head node storing the address of the first node;
   b) for each keyword $w_i \in \mathcal{W}$ $(i = 1 \to n)$:
      - let S_i be the encrypted file set of containing w_i;
      - initialize $\mathrm{AAT}_{S_i} = f(\pi(w_i)) + f(V)$;
      - for $j = 1 \to N$:
        - if $C_j \in S_i$, then compute AAT_{S_i} =$\mathrm{AAT}_{S_i} + \sum_{t=1}^{b} \alpha_j M_{jt}$.
      - let AAT_{S_i} be the value of the i-th node related to w_i.
4) Output the secure index $\mathcal{I} = (\mathrm{ST}, \mathrm{VL})$ and the ciphertext collection $\mathcal{C}$.

GenToken(K, w_k)**:**

- Output the trapdoor $T_{w_k} = (\pi(w_k), K_{w_k})$, where $K_{w_k} \leftarrow H(w_k)$.

Search$(T_{w_k}, \mathcal{I}, \mathcal{C})$**:**

1) Parse $T_{w_k} = (\eta, \theta)$ and find the related row list L_{w_k} according to η;
2) Decrypt each index node SKE.$Enc_{K_{w_i}}(w_{ij}, v_j)$ with θ to obtain (w_{ij}, v_j);
   - if $w_{ij} = 1$, then add C_j to the collection $C(w_k)$;
3) Take out AAT_{S_k} from VL according to η;
4) Output $C(w_k)$ and AAT_{S_k}.

Verify$(K, T_{w_k}, C(w_k), \mathrm{AAT}_{S_k})$:

1) Parse T_{w_k} as $T_{w_k} = (\eta, \theta)$ and initializes $\mathrm{AAT}'_{S_k} = f(\eta) + f(V)$;
2) for each encrypted file $C_j \in C(w_k)$:
   - divide C_j into b blocks $M_{jt} \in \mathbb{Z}_p$ $(1 \leq t \leq b)$;
   - compute $\alpha_j = P(j)$ and AAT'_{S_k} =$\mathrm{AAT}'_{S_k} + \sum_{t=1}^{b} \alpha_j M_{jt}$.
3) if AAT'_{S_k} =AAT_{S_k}, then return "$accept$"; else, return "$reject$".

Dec$(K, C(w_k))$**:**

- Computes F =SKE.$Dec_{K_{enc}}(C)$ with the key K_{enc} for each encrypted file C in $C(w_k)$.

UpToken$(K, F_k, (F'_k))$**:**

1. **Modify token τ_m (modify F_k to F'_k):**
   a) Compute the ciphertext C'_k =SKE.$Enc_{K_{enc}}(F'_k)$ and the random value $\alpha_k = P(k)$, and set the update times $v_k = v_k + 1$;
   b) Divide C'_k into b blocks $M'_{kt} \in \mathbb{Z}_p$ $(1 \le t \le b)$;
   c) Set $\tau_{m_0} = k$;
   d) for each keyword $w_i \in \mathcal{W}$ $(i = 1 \to n)$:
      - compute $K_{w_i} = H(w_i)$ and the middle modify token is $\tau_{m_i} = \{< \tau_{m_i}.modval[1], \tau_{m_i}.modval[2] >, \text{“}mod\text{”}\}$;
      - if $w_i \in F'_k$ and $w_i \in F_k$, then set $w'_{ik} = 1$, and compute $\tau_{m_i}.modval[1]$ =SKE.$Enc_{K_{w_i}}(w'_{ik}, v_k)$ and $\tau_{m_i}.modval[2] = f(V+1) - f(V) + \sum_{t=1}^{b} \alpha_k M'_{kt} - \sum_{t=1}^{b} \alpha_k M_{kt}$.
      - else if $w_i \in F'_k$ and $w_i \notin F_k$, then set $w'_{ik} = 1$, and compute $\tau_{m_i}.modval[1]$ =SKE.$Enc_{K_{w_i}}(w'_{ik}, v_k)$ and $\tau_{m_i}.modval[2] = \sum_{t=1}^{b} \alpha_k M'_{kt} + f(V+1) - f(V)$.
      - else if $w_i \notin F'_k$ and $w_i \in F_k$, then set $w'_{ik} = 0$, and compute $\tau_{m_i}.modval[1]$ =SKE.$Enc_{K_{w_i}}(w'_{ik}, v_k)$ and $\tau_{m_i}.modval[2] = f(V+1) - f(V) - \sum_{t=1}^{b} \alpha_k M_{kt}$.
      - else if $w_i \notin F'_k$ and $w_i \notin F_k$, then set $w'_{ik} = 0$, and compute $\tau_{m_i}.modval[1]$ =SKE.$Enc_{K_{w_i}}(w'_{ik}, v_k)$ and $\tau_{m_i}.modval[2] = f(V+1) - f(V)$.
   e) Set $\tau_{m_{n+1}} = C_k$;
   f) Update $V = V + 1$;
   g) Output the modify token $\tau_m = (\tau_{m_0}, ..., \tau_{m_{n+1}})$;
2. **Add token τ_a (add F_{N+1}):**
   a) Compute the ciphertext C_{N+1} =SKE.$Enc_{K_{enc}}(F_{N+1})$ and the random value $\alpha_{N+1} = P(N+1)$, and set the update times $v_{N+1} = 1$;
   b) Divide C_{N+1} to b blocks $M_{(N+1)t} \in \mathbb{Z}_p$ $(1 \le t \le b)$;
   c) Set $\tau_{a_0} = N + 1$;
   d) for each keyword $w_i \in \mathcal{W}$ $(i = 1 \to n)$:
      - compute $K_{w_i} = H(w_i)$ and the middle add token is $\tau_{a_i} = \{< \tau_{a_i}.addval[1], \tau_{a_i}.addval[2] >, \text{“}add\text{”}\}$;
      - if $w_i \in F_{N+1}$, then set $w_{i(N+1)} = 1$ and compute $\tau_{a_i}.addval[1]$ =SKE.$Enc_{K_{w_i}}(w_{i(N+1)}, v_{N+1})$ and $\tau_{a_i}.addval[2] = \sum_{t=1}^{b} \alpha_{N+1} M_{(N+1)t} + f(V+1) - f(V)$.
      - else, set $w_{i(N+1)} = 0$ and compute $\tau_{a_i}.addval[1]$ =SKE.$Enc_{K_{w_i}}(w_{i(N+1)}, v_{N+1})$ and $\tau_{a_i}.addval[2] = f(V+1) - f(V)$.
   e) Set $\tau_{a_{n+1}} = C_{N+1}$;
   f) Update $V = V + 1$;
   g) Output the add token $\tau_a = (\tau_{a_0}, \tau_{a_1}, ..., \tau_{a_{n+1}})$.
3. **Delete token τ_d (delete F_k):**
   a) Compute the ciphertext C_k =SKE.$Enc_{K_{enc}}(F_k)$ and the random value $\alpha_k = P(k)$;
   b) Divide C_k into b blocks $M_{kt} \in \mathbb{Z}_p$ $(1 \le t \le b)$;
   c) Set $\tau_{d_0} = k$;
   d) for each keyword $w_i \in \mathcal{W}$ $(i = 1 \to n)$:
      - the middle delete token is $\tau_{d_i} = \{\tau_{d_i}.delval, \text{“}del\text{”}\}$;
      - if $w_i \in F_k$, then set $\tau_{d_i}.delval = f(V+1) - f(V) - \sum_{t=1}^{b} \alpha_k M_{kt}$.
      - else, set $\tau_{d_i}.delval = f(V+1) - f(V)$.
   e) Set $\tau_{d_{n+1}} = C_k$;
   f) Update $V = V + 1$;
   g) Output the delete token $\tau_d = (\tau_{d_0}, \tau_{d_1}, ..., \tau_{d_{n+1}})$.

Update$(\tau, \mathcal{I}, \mathcal{C})$:

1) **Modify operation(modify F_k to F'_k, $\tau = \tau_m$):**
   a) for $j = 1 \to k$ $(k = \tau_{m_0})$:
      - find the k-th index node of L_{w_1}, that is the first node of the column list L_{f_k}.
   b) for each node in L_{f_k} $(i = 1 \to n)$:
      - replace the value of each node with $\tau_{m_i}.modval[1]$.
   c) for each node in VL $(i = 1 \to n)$:
      - update AAT$_{S_i}$ =AAT$_{S_i} + \tau_{m_i}.modval[2]$.
   d) Replace C_k with $\tau_{m_{n+1}}$ in the ciphertext collection;
   e) Output the new secure index $\mathcal{I}'$ and the new ciphertext collection $\mathcal{C}'$.
2) **Add operation (add F_{N+1}, $\tau = \tau_a$):**
   a) for $j = 1 \to N$ $(N = \tau_{a_0} - 1)$:
      - find the last (the N-th) node of L_{w_1}, that is the first node of L_{f_N}.
   b) for each node in L_{f_N} $(i = 1 \to n)$:
      - add a new node with the value of $\tau_{a_i}.addval[1]$ right to each node in L_{f_N};
      - set the right point field of the new node to $NULL$.
   c) Chain all of the new nodes to get the new column list $L_{f_{N+1}}$;
   d) for each node in VL $(i = 1 \to n)$:
      - update AAT$_{S_i}$ =AAT$_{S_i} + \tau_{a_i}.addval[2]$.
   e) Add the new file $\tau_{a_{n+1}}$ to the ciphertext collection.
   f) Output the new secure index $\mathcal{I}'$ and the new ciphertext collection $\mathcal{C}'$.
3) **Delete operation (delete F_k, $\tau = \tau_d$):**
   a) if $k = N$ $(k = \tau_{d_0})$, then find the $(N-1)$-th node of L_{w_1}, that is the first node of column list $L_{f_{N-1}}$.
      - for each node in $L_{f_{N-1}}$ $(i = 1 \to n)$:
        – set the right point field of each node to $NULL$, that is disconnect $L_{f_{N-1}}$ with L_{f_N};
        – free L_{f_N}.
   b) if $k \neq N$, then find the $(k-1)$-th node of L_{w_1}, that is the first node of column list $L_{f_{k-1}}$.
      - for each node in $L_{f_{k-1}}$ $(i = 1 \to n)$:
        – chain each node in $L_{f_{k-1}}$ with the related node in $L_{f_{k+1}}$, that is disconnect $L_{f_{k-1}}$ with L_{f_k} and L_{f_k} with $L_{f_{k+1}}$;
        – free L_{f_k}.
   c) for each node in VL $(i = 1 \to n)$:
      - update AAT$_{S_i}$ =AAT$_{S_i} + \tau_{d_i}.delval$.
   d) Delete the file $\tau_{d_{n+1}} = C_k$ from the ciphertext collection;
   e) Output the new secure index $\mathcal{I}'$ and the new ciphertext collection $\mathcal{C}'$.

3) Examples for update operations: In this section, we give some simple examples to illustrate the update operations. Assume there are three files F_1, F_2 and F_3 in the file set $\mathcal{F}$ and three keywords w_1, w_2 and w_3 in the key set $\mathcal{W}$.

F_1 contains keywords w_1 and w_3, F_2 contains keywords w_1 and w_2, and F_3 contains keywords w_1, w_2 and w_3. So $S_1 = \{C_1, C_2, C_3\}$, $S_2 = \{C_2, C_3\}$ and $S_3 = \{C_1, C_3\}$. We have $w_{11} = 1, w_{12} = 1$, $w_{13} = 1$, $w_{21} = 0$, $w_{22} = 1$, $w_{23} = 1$, $w_{31} = 1$, $w_{32} = 0$ and $w_{33} = 1$. The update times v_1, v_2 and v_3 are all initialized to 1. ST contains 3 row lists $(L_{w_1}, ..., L_{w_3})$ and 4 column lists $(L_{f_0}, ..., L_{f_3})$. The first node of each row list L_{w_i} $(1 \le i \le 3)$ stores $\pi(w_i)$. The index node stores SKE.$Enc_{K_{w_i}}(w_{ij}, v_j)$ $(1 \le j \le 3)$, where

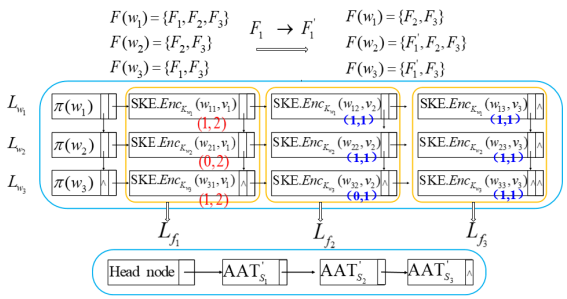


Fig. 3: Modifying a file

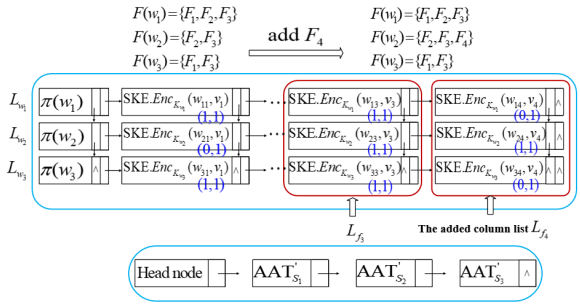


Fig. 4: Adding a file

$K_{w_i} = H(w_i)$. VL is a singly linked list composed by one head node and three index nodes. Each index node in VL stores the following the authentication tag, $\text{AAT}_{S_1} = f(\pi(w_1)) + f(V) + \sum_{t=1}^{b} \alpha_1 M_{1t} + \sum_{t=1}^{b} \alpha_2 M_{2t} + \sum_{t=1}^{b} \alpha_3 M_{3t}$, $\text{AAT}_{S_2} = f(\pi(w_2)) + f(V) + \sum_{t=1}^{b} \alpha_2 M_{2t} + \sum_{t=1}^{b} \alpha_3 M_{3t}$, and $\text{AAT}_{S_3} = f(\pi(w_3)) + f(V) + \sum_{t=1}^{b} \alpha_1 M_{1t} + \sum_{t=1}^{b} \alpha_3 M_{3t}$, where $\alpha_j = P(j)$, V is initialized to 1 and $M_{jt} \in \mathbb{Z}_p$ is the t-th block of C_j. We now describe the details of the modifying operation, the adding operation and the deleting operation.

Assume the data owner modifies the file F_1 to F_1', which contains keywords w_2, w_3. The keyword w_1 is modified to w_2. The data owner only needs to update $v_1 = 2$ and set $w_{11} = 0$, $w_{21} = 1$. v_2, v_3 and w_{31} do not need to be modified. Meanwhile, he modifies $S_1 = \{C_2, C_3\}$ and $S_2 = \{C_1', C_2, C_3\}$, and sets $\tau_{m_0} = 1$ and $\tau_{m_4} = C_1'$. Then he computes $\tau_{m_i}.modval[1]$ = SKE.$Enc_{K_{w_i}}(w_{i1}, v_1)$ $(1 \le i \le 3)$ and some tag value $\tau_{m_i}.modval[2]$, that is, $\tau_{m_1}.modval[2] = f(V+1) - f(V) - \sum_{t=1}^{b} \alpha_1 M_{1t}$, $\tau_{m_2}.modval[2] = \sum_{t=1}^{b} \alpha_1 M_{1t}' + f(V+1) - f(V)$, and $\tau_{m_3}.modval[2] = \sum_{t=1}^{b} \alpha_1 M_{1t}' - \sum_{t=1}^{b} \alpha_1 M_{1t} + f(V+1) - f(V)$, where $M_{1t}' \in \mathbb{Z}_p$ is the t-th block of C_1'. Then he sets $V = V + 1$ and sends the modify token to the cloud server.

Owing to the connectivity of orthogonal list, the cloud server can fleetly find the modified column list L_{f_1}. It does not need to search through each row list from scratch, just needs to find the first index node of L_{f_1} along with L_{w_1} and updates all nodes in L_{f_1}. It replaces the value of each node in L_{f_1} with $\tau_{m_i}.modval[1]$ $(1 \le i \le 3)$ and replaces C_1 with τ_{m_4}. It updates the authentication tag in VL by adding the related tag value $\tau_{m_i}.modval[2]$, that is, AAT_{S_1}' = $\text{AAT}_{S_1} + \tau_{m_1}.modval[2]$, AAT_{S_2}' = $\text{AAT}_{S_2} + \tau_{m_2}.modval[2]$, and AAT_{S_3}' = $\text{AAT}_{S_3} + \tau_{m_3}.modval[2]$. This example is shown in Fig.3.

Assume the data owner adds a new file F_4 containing keywords w_2 to the file collection. The data owner sets $v_4 = 1$, $w_{14} = 0$, $w_{24} = 1$, $w_{34} = 0$, $\tau_{a_0} = 4$ and $\tau_{a_4} = C_4$, and modifies $S_2 = \{C_2, C_3, C_4\}$. He computes $\tau_{a_i}.modval[1]$ = SKE.$Enc_{K_{w_i}}(w_{i4}, v_4)$ $(1 \le i \le 3)$, and the related tag value $\tau_{a_i}.modval[2]$, that is, $\tau_{a_1}.modval[2] = \tau_{a_3}.modval[2] = f(V+1) - f(V)$, and $\tau_{a_2}.modval[2] = \sum_{t=1}^{b} \alpha_4 M_{4t} + f(V+1) - f(V)$, where $\alpha_4 = P(4)$ and $M_{4t} \in \mathbb{Z}_p$ is the t-th block of C_4. Then he sets $V = V + 1$ and sends the add token to the cloud server.

Owing to the flexibility of orthogonal list, the cloud server only needs to find the third index node of L_{w_1}, that is, the first node of L_{f_3}. It adds a new node next to each node of L_{f_3} from the first node to the last node, and adds τ_{a_4} to the ciphertext set. The cloud server computes the new tag value in VL as AAT_{S_1}'(or AAT_{S_3}') =AAT_{S_1}(or AAT_{S_3}) + $\tau_{a_1}.modval[2]$, and AAT_{S_2}' =AAT_{S_2} + $\tau_{a_2}.modval[2]$. This example is shown in Fig.4.

Assume the data owner deletes a file F_2 containing keywords w_1 and w_2 from the file collection. The data owner sets $\tau_{d_0} = k$ and $\tau_{d_4} = C_2$, and computes the related tag value $\tau_{d_i}.delval(1 \le i \le 3)$, that is, $\tau_{d_1}.delval = \tau_{d_2}.delval = f(V+1) - f(V) - \sum_{t=1}^{b} \alpha_2 M_{2t}$, and $\tau_{d_3}.delval = f(V+1) - f(V)$. He updates $V = V + 1$ and gives the delete token to the cloud server.

The cloud server firstly finds the first index node of L_{w_1}, that is, the first node of L_{f_1}. Then it chains L_{f_1} with L_{f_3} and deletes L_{f_2}. Finally, it updates the tag $\text{AAT}_{S_i}(1 \le i \le 3)$ in VL, that is, AAT_{S_1}'(or AAT_{S_2}') =AAT_{S_1}(or AAT_{S_2})+$\tau_{d_1}.delval$, and AAT_{S_3}' =AAT_{S_3} + $\tau_{d_3}.delval$. This example is shown in Fig.5.

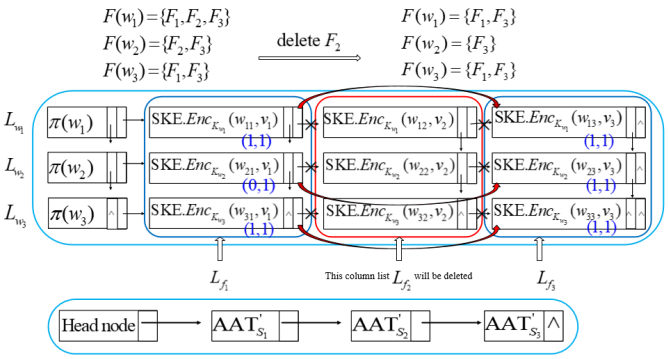


Fig. 5: Deleting a file

V. Security Analysis

In this section, we analyze the security of our proposed scheme in term of update reliability, verifiability and CKA2-security.

Theorem 1: The proposed VDSSE scheme satisfies update reliability.

Proof. Assume a *PPT* adversary $\mathcal{A}$ does not update the secure index or the ciphertext collection when the data owner requires to update some files. We will prove that if $\mathcal{A}$ returns the non-updated result, the verification algorithm will output "$reject$".

Assume the returned result is $(C'(w),\mathrm{AAT}_{S'})$, the corresponding global update number is V' and Δ' is the identifier set of files in $C'(w)$. $C'_j \in C'(w)$ is composed by $M'_{jt} \in \mathbb{Z}_p$ $(1 \le t \le b)$. We have

$$\mathrm{AAT}_{S'} = f(\pi(w)) + f(V') + \sum_{j\in\Delta'}\sum_{t=1}^{b}\alpha_j M'_{jt} \tag{3}$$

Now the secure index and the ciphertext collection are updated x $(x \ge 1)$ times again. So the current global update number is $V = V' + x$. The verified $\mathrm{AAT}'_{S'}$ is computed by V, that is,

$$\mathrm{AAT}'_{S'} = f(\pi(w)) + f(V) + \sum_{j\in\Delta'}\sum_{t=1}^{b}\alpha_j M'_{jt} \tag{4}$$

If the non-updated result passes the verification, the adversary $\mathcal{A}$ can get $f(V) = f(V')$ by subtracting equation (4) from equation (3). It means the adversary $\mathcal{A}$ can find a V and a V' satisfying $f(V) = f(V')$, which is contradictory to the security of PRF f. Therefore, our proposed scheme satisfies the update reliability.

Theorem 2: The proposed VDSSE scheme satisfies verifiability.

Proof. Assume that $\mathcal{A}$ is a *PPT* adversary who can give a forgery $(C'(w),\mathrm{AAT}_{S'})$ $(S' = C'(w))$ such that the verification algorithm **Verify**$(K,\mathrm{AAT}_{S'},C'(w),T_w)$ outputs "$accept$". Assume the correct search result is $(C(w),\mathrm{AAT}_S)$ $(S = C(w))$. We will prove that there is no such an adversary $\mathcal{A}$ who can give a forgery satisfying $(C'(w),\mathrm{AAT}_{S'}) = (C(w),\mathrm{AAT}_S)$.

Let Δ be the identifier set of files in $C(w)$ and $C_j \in C(w)$ be composed by $M_{jt} \in \mathbb{Z}_p$ $(1 \le t \le b)$. Similarly, let Δ' be the identifier set of files in $C'(w)$ and $C'_j \in C'(w)$ be composed by $M'_{jt} \in \mathbb{Z}_p$ $(1 \le t \le b)$. Assume Λ is the identifier set of the same files between $C(w)$ and $C'(w)$, and $|S'| \le |S|$.

We consider the following three cases:

Case 1: $C'(w) = C(w)$ and $\mathrm{AAT}_{S'} \ne \mathrm{AAT}_S$. We have $\mathrm{AAT}_{S'} = f(\pi(w)) + f(V) + \sum_{j\in\Delta'}\sum_{t=1}^{b}\alpha_j M'_{jt}$ and $\mathrm{AAT}_S = f(\pi(w)) + f(V) + \sum_{j\in\Delta}\sum_{t=1}^{b}\alpha_j M_{jt}$. Since $C'(w) = C(w)$, we have $\mathrm{AAT}_{S'} = \mathrm{AAT}_S$. This is contradictory to $\mathrm{AAT}_{S'} \ne \mathrm{AAT}_S$. So this case does not hold.

Case 2: $C'(w) \ne C(w)$ and $\mathrm{AAT}_{S'} = \mathrm{AAT}_S$. It means $\mathcal{A}$ can find a collision for **AAT**. It is contradictory to the collision resistance of **AAT** described in section 4.1.1. Therefore, this case also does not hold.

Case 3: $C'(w) \ne C(w)$ and $\mathrm{AAT}_{S'} \ne \mathrm{AAT}_S$. We have $\mathrm{AAT}_{S'} = f(\pi(w)) + f(V) + \sum_{j\in\Delta'}\sum_{t=1}^{b}\alpha_j M'_{jt}$ and $\mathrm{AAT}_S = f(\pi(w)) + f(V) + \sum_{j\in\Delta}\sum_{t=1}^{b}\alpha_j M_{jt}$. For simplicity, assume $\Lambda = \{k_1, k_2, ..., k_r\}$ $(r \le |S'|)$ and $\Delta/\Lambda = \{k_{r+1}, ..., k_{|S|}\}$. Let $d = \mathrm{AAT}_S - \mathrm{AAT}_{S'}$. We have

$$d = \alpha_{k_{r+1}}\sum_{t=1}^{b}M''_{k_{r+1}t} + \cdots + \alpha_{k_{|S|}}\sum_{t=1}^{b}M''_{k_{|S|}t}, \tag{5}$$

where $M''_{k_jt} = M_{k_jt} - M'_{k_jt}(1 \le j \le |S'|)$ and $M''_{k_jt} = M_{k_jt}(j > |S'|)$. $\mathcal{A}$ is allowed to query P oracle up to $|S|-1$ times. Without loss of generality, assume $\mathcal{A}$ has queried $\alpha_{k_1}, ..., \alpha_{k_{|S|-1}}$ from P oracle. According to equation (5), $\mathcal{A}$ can compute

$$\alpha_{k_{|S|}} = \frac{d - (\alpha_{k_{r+1}}\sum_{t=1}^{b}M''_{k_{r+1}t} + \cdots + \alpha_{k_{|S|-1}}\sum_{t=1}^{b}M''_{k_{|S|-1}t})}{\sum_{t=1}^{b}M''_{k_{|S|}t}}. \tag{6}$$

Note that, the denominator $\sum_{t=1}^{b}M''_{k_{|S|}t}$ is zero only with negligible probability. It means the adversary $\mathcal{A}$ can compute $P_{K_p}(k_{|S|})$ without knowing the key K_p with high probability, which is contradictory to the security of PRF P. Therefore, the proposed VDSSE scheme satisfies the verifiability.

Theorem 3: If PRFs f, π, P are all pseudo-random and cryptographic primitives and SKE scheme is secure against chosen-plaintext attack (CPA-secure), then our scheme is $\mathcal{L} = \{\mathcal{L}_{setup}, \mathcal{L}_{search}, \mathcal{L}_{update}\}$-secure against adaptive chosen-keyword attacks(CKA2) in the standard model.

Proof. We describe a polynomial-time simulator $\mathcal{S}$ such that for any PPT adversary $\mathcal{A}$, the outputs of **Real**$_{\mathcal{A}}(\lambda)$ and **Ideal**$_{\mathcal{A},\mathcal{S}}(\lambda)$ are computationally indistinguishable. In **Real**$_{\mathcal{A}}(\lambda)$ experiment, $\mathcal{A}$ receives $\{\mathcal{I}, \mathcal{C}, \tau = \{T_w, \tau_m, \tau_a, \tau_d\}\}$ from the challenger. In **Ideal**$_{\mathcal{A},\mathcal{S}}(\lambda)$ experiment, the simulator $\mathcal{S}$ generates a simulated secure index $\mathcal{I}'$=(ST$'$,VL$'$), a simulated ciphertexts $\mathcal{C}'$, and the simulated tokens $\tau' = \{T'_w, \tau'_m, \tau'_a, \tau'_d\}$ for the information $\mathcal{L}_{setup}, \mathcal{L}_{search}, \mathcal{L}_{update}$. We prove that $\mathcal{A}$ cannot distinguish between experiments **Real** and **Ideal**. In other words, $\mathcal{A}$ cannot distinguish between $\{\mathcal{I}, \mathcal{C}, \tau\}$ and $\{\mathcal{I}', \mathcal{C}', \tau'\}$.

Simulating $\mathcal{I}'$: The simulated secure index $\mathcal{I}'$ can be constructed similarly to a real secure index, except that node information is replaced by ciphertext of the zero string (of appropriate length) and the outputs of PRFs are replaced by random values.

$\mathcal{S}$ initializes $|\mathcal{W}|$ lists $L_{w_i(1\le i\le|\mathcal{W}|)}$ with $|\mathcal{C}|+1$ nodes for ST$'$. $\mathcal{S}$ randomly selects $|\mathcal{W}||\mathcal{C}|$ bit strings $Ew'_{ij(1\le j\le|\mathcal{C}|)}$ for each node of L_{w_i}. $\mathcal{S}$ initializes VL$'$ with $|\mathcal{W}|+1$ nodes. $\mathcal{S}$ sets $\mathcal{I}'$ = (ST$'$,VL$'$), where ST$'$ $= Ew'_{ij}$ and VL$'$ = AAT'_{S_i}.

The CPA-security of the encryption algorithms and the pseudo-randomness of the PRFs will guarantee that the simulated secure index $\mathcal{I}'$ = (ST$'$,VL$'$) is indistinguishable from $\mathcal{I}$ = (ST,VL) for $\mathcal{A}$.

Simulating $\mathcal{C}'$: The simulated cipher files $\mathcal{C}' = \{C'_1, ..., C'_{|\mathcal{C}|}\}$ are simulated in the same manner as the simulated secure index (i.e., replacing the cipher files by ciphertexts of the all zero strings) and the CPA-security of the encryption algorithms guarantees indistinguishability.

Given the leakage function $\mathcal{L}_{setup}(\mathcal{F}, \mathcal{W})$, $\mathcal{S}$ randomly selects a bit string $(C'_j)_{(1\le j\le|\mathcal{C}|)}$ of length $|C_j|$. As the encryp-

tion algorithms are CPA-secure, $\mathcal{C}'$ and $\mathcal{C}$ are indistinguishable to $\mathcal{A}$.

Simulating τ': $\mathcal{S}$ computes the search token T'_w, the modify token τ'_m, the add token τ'_a and the deleted token τ'_d according to $\mathcal{L}_{search}$, $\mathcal{L}_{update}$. We give the analysis from the following four cases.

Case 1: T'_w is a search token.

If T'_w is the first search query, $\mathcal{S}$ randomly selects a $\pi'(w)$ that has not been selected before. For each query w, there exists a key K'_w corresponding to it. $\mathcal{S}$ computes $K'_w \leftarrow H(w)$, and sets $T'_w = (\pi'(w), K'_w)$.

If the k-th query T'_w is the same as a prior query, $\mathcal{S}$ sets $T'_w = (\pi'(w)_k, (K'_w)_k)$.

Case 2: τ'_m is a modify token.

$\mathcal{S}$ first randomly selects a bit string Ew'_{ij} $(1 \leq i \leq |\mathcal{W}|)$, a bit string C'_j of length $|F_j|$, and a bit string $\widetilde{C'_j}$ of length $|F'_j|$. Then $\mathcal{S}$ divides $\widetilde{C'_j}$ and C'_j into b blocks, and uses $\widetilde{M'_{jt}}$ and M'_{jt} to denote the t-th block. Finally, $\mathcal{S}$ computes $\alpha'_j \leftarrow P(j)$, and sets $\tau'_{m.modval} =< Ew'_{ij}, \sum_{t=1}^{b} \alpha'_j \widetilde{M'_{jt}} + f(V+1) - f(V) >$ or $< Ew'_{ij}, f(V+1) - f(V) - \sum_{t=1}^{b} \alpha'_j M'_{jt} >$ or $< Ew'_{ij}, \sum_{t=1}^{b} \alpha'_j \widetilde{M'_{jt}} + f(V+1) - f(V) - \sum_{t=1}^{b} \alpha'_j M'_{jt} >$ or $< Ew'_{ij}, f(V+1) - f(V) >$.

Case 3: τ'_a is an add token.

$\mathcal{S}$ first randomly selects a bit string Ew'_{ir} $(1 \leq i \leq |\mathcal{W}|)$, and a bit string C'_r of length $|F'_r|$, where r denotes the identifier of the added file F'_r. Then $\mathcal{S}$ divides C'_r into b blocks, and uses M'_{rt} to denote the t-th block. Finally, $\mathcal{S}$ computes $\alpha'_r \leftarrow P(r)$, and sets $\tau'_{a.addval} =< Ew'_{ir}, \sum_{t=1}^{b} \alpha'_r M'_{rt} + f(V+1) - f(V) >$ or $< Ew'_{ir}, f(V+1) - f(V) >$.

Case 4: τ'_d is a delete token.

$\mathcal{S}$ first randomly selects a bit string C'_j of length $|F'_j|$. Then $\mathcal{S}$ divides C'_j into b blocks, and uses M'_{jt} to denote the t-th block. Finally, $\mathcal{S}$ computes $\alpha'_j \leftarrow P(j)$, and sets $\tau'_{d.delval} =< f(V+1) - f(V) - \sum_{t=1}^{b} \alpha'_j M'_{jt} >$ or $< f(V+1) - f(V) >$.

In such a way, $\mathcal{S}$ simulates the correct search/update tokens which have the same result as that in the experiment **Real**. Therefore, $\mathcal{A}$ cannot distinguish between τ and τ'.

In conclusion, $\mathcal{A}$ cannot distinguish between the result in the experiment **Real** and the result in the experiment **Ideal**. That is,

$$|\Pr[\mathbf{Real}_{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda) = 1]| \leq \mathrm{negl}(\lambda).$$

Therefore, our proposed scheme satisfies CKA2-security.

VI. Performance and Experiments

In this section, we analyze the performance of the proposed VDSSE scheme. Experiments are implemented using C++ language on a Linux OS equipped with 2.4GHz Inter(R) Core(TM) i5 CPU and 4GB RAM, and constructed on a real-world data set [45].

Index construction efficiency. To evaluate the efficiency of our proposed schemes, we conduct the experiments for building ST and building VL. Fig.6 shows the time cost of building the secure index when the number of files is set to 10000 and the number of keywords varies from 1000 to 10000. We can observe that the time cost grows linearly with the number of keywords. In our scheme, as the row list number in ST equals to the number of keywords, the increase of the row list number leads to the increase of ST construction time cost. As the node number in VL equals to the number of keywords, the increase of the node number leads to the increase of VL construction time. Fig.7 shows the time cost of building the secure index when the number of keywords is set to 10000 and the number of files varies from 1000 to 10000. We can observe that the time cost of building index grows linearly with the number of files. In our scheme, as the column list number in ST equals to the number of files, the increase of the column list number leads to the increase of ST construction time cost. The time cost of computing tags in VL is related to the number of files.

Update token generation efficiency. Fig.8 shows that the generation time for update tokens, i.e., modify token, add token and delete token, is almost linear with the number of keywords. Since the computation for modify token generation is more than that for add token generation and that for delete token generation, the time cost of modify token generation is a little longer.

Search efficiency. The cloud server respectively performs search operations in ST and VL. In Fig.9, we vary the number of files from 1000 to 10000 and set the number of keywords to 10000. We can observe that the search time cost in ST is nearly linear with the number of files. In Fig.10, we vary the number of keywords from 1000 to 10000 and set the number of files to 10000. As the search time is mainly related to the time of locating the queried keyword, its cost is nearly linear with the number of keywords. Fig.11 shows that the search time cost in VL is nearly linear with the number of keywords. As the node number in VL equals to the number of keywords, the increase of node number leads to the increase of search time.

Because the secure index is built offline and building the index is just a one-time operation by the user, we did not compare its efficiency in our scheme with that in another scheme. We compare the performance of our scheme with that of scheme [10] in terms of search token generation efficiency, verification efficiency and update efficiency. The reason that we choose scheme [10] as a benchmark is mainly because it is generally viewed as the most typical verifiable SSE scheme.

Search token generation efficiency. We compare the search token generation time cost of our scheme with that of scheme [10] shown in Fig.12. The time cost of search token generation is almost linear with the number of files in Sun's scheme [10], while that is almost constant in our VDSSE scheme. Because the bit string length in search token equals to the number of files in scheme [10], the increase of the bit string length leads to the increase of search token generation cost. In contrast, the cost of search token generation is irrelevant to the number of files in our scheme.

Verification efficiency. We compare the verification time cost of our scheme with that of scheme [10] shown in Fig.13. We can see that the verification efficiency in our scheme is much higher than that in scheme [10]. The scheme [10]

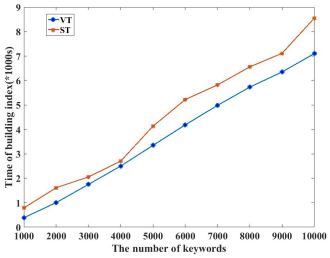


Fig. 6: Secure index construction time cost

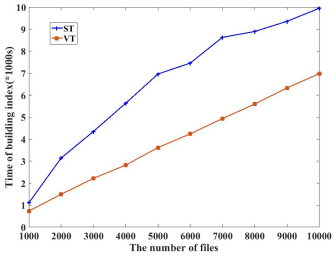


Fig. 7: Secure index construction time cost

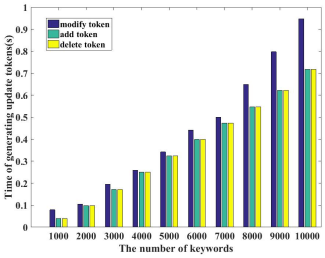


Fig. 8: Modify token generation time cost

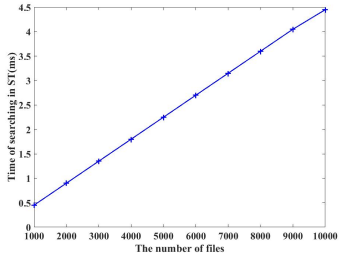


Fig. 9: Search time cost in ST

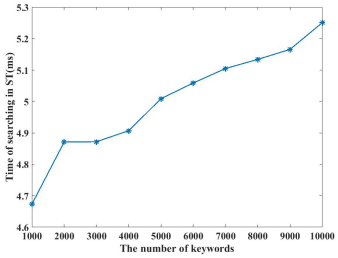


Fig. 10: Search time cost in ST

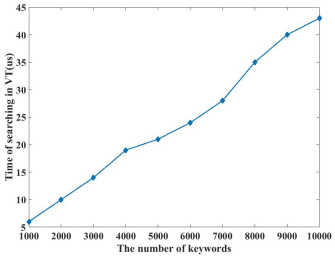


Fig. 11: Search time cost in VL

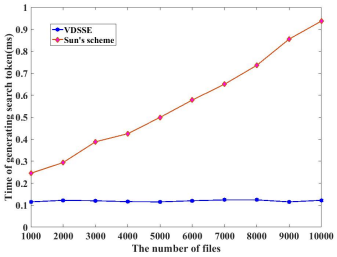


Fig. 12: The time cost of search token generation

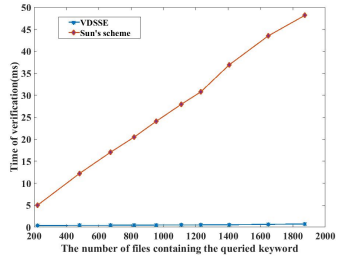


Fig. 13: The time cost of verification

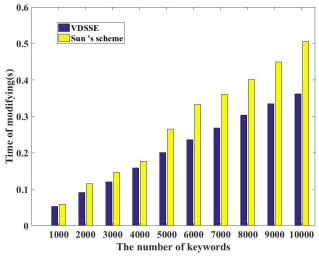


Fig. 14: Modifying time cost

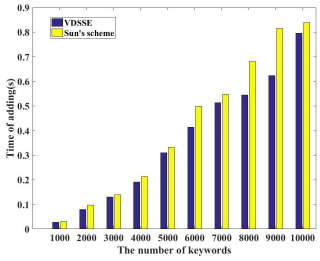


Fig. 15: Adding time cost

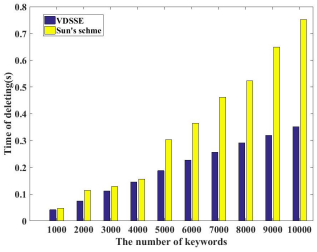


Fig. 16: Deleting time cost

utilizes the bilinear-map accumulator to verify the search results. The bilinear-map accumulator is based on asymmetric-key cryptography involving time-consuming operations. In contrast, we design a novel accumulative authentication tag based on symmetric-key cryptography to achieve verification in our scheme. As shown in Fig.13, when the number of files containing the queried keyword is about 200, the verification time is about 0.36ms in our scheme, while it is about 5ms in scheme [10]. When the number of files containing the queried keyword is about 2000, the verification time is about 0.75ms in our scheme, while it is nearly 50ms in scheme [10].

Update efficiency. We compare the update time cost of our scheme with that of scheme [10] shown in Fig.14, Fig.15 and Fig.16. As the node number in the updated column list equals to the number of keywords, the increase of the node number leads to the increase of update time. Therefore, the update time is proportional to the number of keywords. We can observe that the update efficiency in our scheme is clearly higher than that in scheme [10]. The main reason for the promotion of update efficiency is that the cloud server only needs to find the updated column list in the first row list, while does not search each row list from scratch in our scheme.

VII. Discussion

In this section, we discuss how to prevent the cloud server from learning the number of keywords and how to add a new keyword in the original keyword set. We will give potential solutions to these problems.

1) **Hiding the number of keywords.** In our VDSSE scheme, we construct a row list for each keyword. The cloud server can know the number of keywords by the number of row lists. If we consider to hide the number of keywords, we can extend the secure index by padding dummy value [21]. Assume that each keyword is at most l bits and the number of keywords is at most 2^l, $n \leq 2^l$. We will enlarge the number of row lists to $2 \cdot 2^l$ to ensure that the number of linked lists is larger than that of keywords. We select $2 \cdot 2^l - n$ dummy values dum_k(the short of "dummy") and $2 \cdot 2^l - n$ random 0/1 bit denoted by pw_{kj} $(1 \leq k \leq 2 \cdot 2^l - n,\ 1 \leq j \leq N)$. To update the index nodes by the keyword order, we pad these $2 \cdot 2^l - n$ dummy values into the last remaining positions.

The value of each index node is set SKE.$Enc(pw_{kj}, v_j)$ generated with the key $K_{w_{dum_k}}$. Meanwhile, we also need to enlarge VL to $2 \cdot 2^l$. The remaining $2 \cdot 2^l - n$ positions are padded with dum_k.

2) **Adding a new keyword.** We have extracted all different keywords from the original files before constructing the secure index. When a new file is added, the file might contain a new keyword except the original keywords. In this case, we need to add the new keyword to the keyword set and update the secure index. We assume the added file is F_{N+1} and the new keyword is w_{n+1}. Obviously, w_{n+1} only appears in F_{N+1}. So we know that $w_{(n+1),1} = ... = w_{(n+1),N} = 0$ and $w_{(n+1)(N+1)} = 1$. We can generate $K_{w_{n+1}} \leftarrow H(w_{n+1})$ and $\alpha_j \leftarrow P(j)$, and update the global update number $V = V + 1$. The cloud server updates the value of the $(n+1)$-th linked list (the above generated) with SKE.$Enc(w_{(n+1)j}, v_j)$ using the key $K_{w_{n+1}}$, and also updates the $(n+1)$-th verification node in the verification list with AAT$_{S_{n+1}}$ = $f(\pi(w_{n+1})) + f(V) + \sum_{t=1}^{b} \alpha_{N+1} M_{(N+1)t}$.

We give a simple example with 3 keywords and 3 files to show the detailed structure with padding and with adding a new keyword in Fig.17. There are 2^{l+1} row lists in ST and 2^{l+1} nodes in VL. As shown in Fig.17, 3 row lists are padded with the real values and $2^{l+1} - 3$ row lists are padded with dummy values in ST. As same as ST, 3 nodes are padded with real values and $2^{l+1} - 3$ nodes are padded with dummy values in VL. When a new file containing a new keyword is added, the cloud server replaces the values of nodes in the 3-th row list of ST with the add token, and updates the value of the 3-th node in VL.

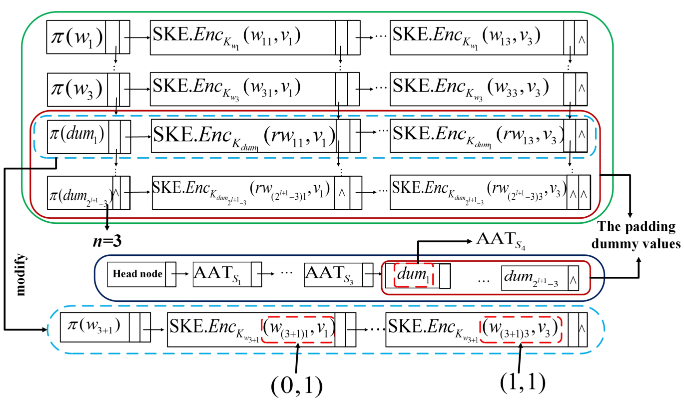


Fig. 17: The secure index with padding dummy values and adding a new keyword

VIII. Conclusion

In this paper, we explore realizing keyword search over dynamic encrypted cloud data with symmetric-key based verification. In order to support the efficient verification of dynamic data, we design a novel Accumulative Authentication Tag (**AAT**) based on symmetric-key cryptography to generate an accumulative authentication tag for each keyword. Moreover, a new secure index based on the orthogonal list and the single linked list is designed to improve the updated efficiency. The security analysis and the performance evaluation show that the proposed scheme is secure and efficient.

Acknowledgments

This research is supported by National Natural Science Foundation of China (61572267,61572412,61602275), National Development Foundation of Cryptography (MMJJ20170118,MMJJ20170126), the Open Project of Co-Innovation Center for Information Supply and Assurance Technology, Anhui University, the Open Project of the State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences(2017-MS-21,2019-MS).

References

[1] S. Kamara, C. Papamanthou and T. Roeder, "Dynamic searchable symmetric encryption," presented at *ACM Conference on Computer and Communications Security,* pp. 965-976, 2012.
[2] C. Guo, X. Chen, Y. M. Jie, Z. J. Fu, M. C. Li and B. Feng, "Dynamic Multi-phrase Ranked Search over Encrypted Data with Symmetric Searchable Encryption," in *IEEE Transations on Services Computing,* vol. 99, No. 1939, pp. 1-1, 2017.
[3] Z. H. Xia, X. H. Wang, X. M. Sun and Q. Wang, "A Secure and Dynamic Multi-Keyword Ranked Search Scheme over Encrypted Cloud Data," in *IEEE Transactions on Parallel and Distributed Systems,* vol. 27, No. 2, pp. 340-352, 2016.
[4] S. Kamara and C. Papamanthou, "Parallel and Dynamic Searchable Symmetric Encryption," presented at the *International Conference on Financial Cryptography and Data Security,* pp. 258-274, 2013.
[5] J. B. Yan, Y. Q. Zhang and X. F. Liu, "Secure multi-keyword search supporting dynamic update and ranked retrieval," in *China Communication,* vol. 13, No. 10, pp. 209-221, 2016.
[6] K. Kurosawa and Y. Ohtaki, "How to Update Documents Verifiably in Searchable Symmetric Encryption," presented at *International Conference on Cryptology and Network Security,* pp. 309-328, 2013.
[7] Q. Liu, X. H. Nie, X. H. Liu, T. Peng and J. Wu, "Verifiable Ranked Search over dynamic encrypted data in cloud computing," presented at the *IEEE/ACM International Symposium on Quality of Service,* pp. 1-6, 2017.
[8] X. H. Nie, Q. Liu, X. H. Liu, T. Peng and Y. P. Lin, "Dynamic Verifiable Search Over Encrypted Data in Untrusted Clouds," presented at the *International Conference Algorithm and Architectures for Parallel Processing,* pp. 557-571, 2016.
[9] X. Y. Zhu, Q. Liu and G. J. Wang, "A Novel Verifiable and Dynamic Fuzzy Keyword Search Scheme over Encrypted Data in Cloud Computing," presented at the *IEEE Trustcom/BigDataSE/ISPA,* pp. 845-851, 2017.
[10] W. H. Sun, X. F. Liu, W. J. Lou. Y. T. Hou and H. Li, "Catch you if you lie to me: Efficient verifiable conjunctive keyword search over large dynamic encrypted cloud

data," presented at the *IEEE Conference on Computer Communications(INFOCOM),* pp. 2110-2118, 2015.

[11] C. Wang, B. S. Zhang, K. Ren, J. M. Roveda, C. W. Chen and Z. Xu, "A privacy-aware cloud-assisted healthcare monitoring system via compressive sensing," presented at *INFOCOM, 2014 Proceedings IEEE,* pp. 2130-2138, 2014.

[12] X. L. Yuan, X. Y. Wang, J. Lin and C. Wang, "Privacy-preserving deep packet inspection in outsourced middleboxes," presented at *The 35th Annual IEEE Internatioal conference on computer communications,* pp. 1-9, 2016.

[13] J. Yu, K. Ren and C. Wang, "Enabling Cloud Storage Auditing With Verifiable Outsourcing of Key Updates," in *IEEE Transactions on Information Forensics and Security,* vol. 11, No. 6, pp. 1362-1375, 2016.

[14] J. Yu, K. Ren and C. Wang, "Enabling Cloud Storage Auditing With Key-Exposure Resistance," in *IEEE Transactions on Information Forensics and Security,* vol. 10, No. 6, pp. 1167-1179, 2015.

[15] Y. Zhang, J. Yu, R. Hao, C. Wang and K. Ren, "Enabling Efficient User Revocation in Identity-based Cloud Storage Auditing for Shared Big Data," in *IEEE Transactions on Dependable and Secure Computing,* DOI Bookmark: 10.1109/TDSC.2018.2829880, 2018.

[16] H. Shacham and B. Waters, "Compact Proofs of Retrievability," presented at *ASIACRYPT 2008: Advances in Cryptology,* pp. 90-107, 2008.

[17] Y. B. Miao, J. F. Ma, X. M. Liu, X. H. Li, Q. Jiang and J. W. Zhang, "Attribute-based keyword search over hierarchical data in cloud computing," in *IEEE Transactions on Services Computing,* doi:10.1109/TSC.2017.2757467, 2017.

[18] Y. B. Miao, J. F. Ma, X. M. Liu, J. Weng, H. W. Li and H. Li, "Lightweight fine-grained search over encrypted data in fog computing," in *IEEE Transactions on Services Computing,* DOI: 10.1109/TSC.2018.2823309, 2018.

[19] D. X. Song, D. Wagner and A. Perrig, "Practical Techniques for Searches on Encrypted Data," presented at *IEEE Symposium on Security and Privacy,* pp. 44-55, 2000.

[20] E. J. Goh, "Secure Indexes," presented at *Technical Report 2003/216, IACR ePrint Cryptography Archive,* pp. 1-19, 2003.

[21] R. Curtmola, J. Gary, S. Kamara and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," presented at the *ACM Conference on Computer and Communications Security,* pp. 79-88, 2006.

[22] N. Cao, C. Wang, M. Li, K. Ren and W. J. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," presented at the *INFOCOM, 2011 Proceedings IEEE,* pp. 829-827, 2011.

[23] Z. J. Fu, X. M. Sun, S. Ji and G. W. Xie, "Towards Efficient Content-aware Search over Encrypted Outsourced Data in Cloud," presented at *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications,* pp. 1-9, 2016.

[24] Z. J. Fu, L. L. Xia, X. M. Sun, Alex X. Liu and G. W. Xie, "Semantic Aware Searching over Encrypted Data for Cloud Computing," in *IEEE Transactions on Information Forensics and Security,* vol. 13, No. 9, pp. 2359-2371, 2018.

[25] X. L. Yuan, X. Y. Wang, C. Wang, A. Squicciarini and K. Ren, "Enabling Privacy-preserving Image-centric Social Discovery," presented at *2014 IEEE 34th International Conference on Distributed Computing Systems (ICDCS),* pp. 222-233, 2014.

[26] C. Wang, Q. Wang and K. Ren,"Towards secure and effective utilization over encrypted cloud data," presented at *2011 31st International Conference on Distributed Computing Systems Workshops, ICDCSW'11,* pp. 282-286, 2011.

[27] X. R. Ge, J. Yu, C. Y. Hu, H. L. Zhang and R. Hao, "Enabling Efficient Verifiable Fuzzy Keyword Search Over Encrypted Data in Cloud Computing," in *IEEE Access,* vol. 6, pp. 45725-45739, 2018.

[28] X. L. Yuan, H. Cui, X. Wang and C. Wang, "Enabling Privacy-Assured Similarity Retrieval over Millions of Encrypted Records," presented at *European Symposium on Research in Computer Security,* pp. 40-60, 2015.

[29] W. H. Sun, B. Wang, N. Cao, M. Li, W. J. Lou, Y. Thomas. Hou and H. Li, "Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," presented at *ACM Sigsac Symposium on Information, Computer and Communications Security,* pp. 71-82, 2013.

[30] C. Wang, N. Cao, J. Li and K. Ren, "Secure Ranked Keyword Search over Encrypted Cloud Data," presented at the *IEEE International Conference on Distributed Computing Systems,* pp. 253-262, 2010.

[31] W. Zhang, S. Xiao, Y. P. Lin, T. Zhou and S. W. Zhou, "Secure Ranked Multi-keyword Search for Multiple Data Owners in Cloud Computing," presented at the *IEEE International Conference on Parallel and Distributed Systems,* pp. 276-286, 2017.

[32] Z. J. Fu, X. M. Sun, Nigel Linge and L. Zhou, "Achieving effective cloud search services: multi-keyword ranked search over encrypted cloud data supporting synonym query," in *IEEE Transactions on Consumer Electronics,* vol. 60, No. 1, pp. 164-172, 2014.

[33] Z. J. Fu, X. L. Wu, Q. Wang and K. Ren, "Enabling Central Keyword-based Semantic Extension Search over Encrypted Outsourced Data," in *IEEE Transactions on Information Forensics and Security,* vol. 12, No. 12, pp. 2986-2997, 2017.

[34] J. Li, X. F. Chen, F. Xhafa and L. Barolli, "Secure deduplication storage systems supporting keyword search," in *Journal of Computer and System Sciences,* vol. 81, No. 8, pp. 1532-1541, 2015.

[35] C. Liu, L. H. Zhu and J. J. Chen, "Efficient Searchable Symmetric Encryption for Storing Multiple Source Dynamic Social Data on Cloud," in *Journal of Network and Computer Applications,* vol. 75, No. 3, pp. 451-458, 2016.

[36] C. David, J. Joseph, J. Stanislaw, J. Charanjit, K. Hugo, R. Marcel and S. Michael, "Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation," presented at the *Network and Distributed*

System Security Symposium, pp. 1-16, 2014.

[37] E. Stefanov, C. Papamanthou and E. Shi, "Practical Dynamic Searchable Encryption with Small Leakage," presented at the *Network and Distributed System Security Symposium,* pp. 1-15, 2014.

[38] M. Naveed, M. Prabhakaran and C. A. Gunter, "Dynamic Searchable Encryption via Blind Storage," presented at the *IEEE Symposium on Security and Privacy,* pp. 639-654, 2014.

[39] Q. Chai and G. Gong, "Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers," presented at the *IEEE International Conference on Communications,* pp. 917-922, 2012.

[40] K. Kurosawa and Y. Ohtaki, "UC-Secure Searchable Symmetric Encryption," presented at the *International Conference on Financial Cryptography and Data Security,* pp. 285-298, 2012.

[41] X. X. Jiang, J. Yu, J. B. Yan and R. Hao, "Enabling efficient and verifiable multi-keyword ranked search over encrypted cloud data," in *Information Sciences,* vol. 403-404, pp. 22-41, 2017.

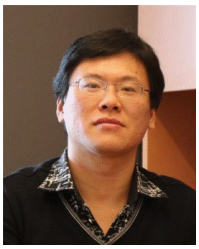
[42] F. Chen, T. Xiang, X. W. Fu and W. Yu, "User Differentiated Verifiable File Search on the Cloud," in *IEEE Transactions on Services Computing*, PP(99):1-1, 2016.

[43] Z. G. Wan and R. H. Deng, "VPSearch: Achieving Verifiability for Privacy-Preserving Multi-Keyword Search over Encrypted Cloud Data," in *IEEE Transactions on Dependable and Secure Computing,* vol. 15, No. 6, pp. 1083-1095, 2016.

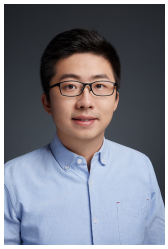
[44] R. Bost, P.A. Fouque and D. Pointcheval, "Verifiable Dynamic Symmetric Searchable Encryption Optimality and Forward Security," in *Cryptology ePrint Archive, Report 2016/062 (2016).* 2016.

[45] C. William, "Enron Email Dataset,"[Online]. Available: http://www.cs.cmu.edu/ ./enron/.

Xinrui Ge received the B.E. degree in information security from Qingdao University in 2016. She is currently pursuing the masters degree in computer science and technology with Qingdao University. Her research interests include cloud computing security and searchable encryption.



Jia Yu received the B.S. and M.S. degrees from the School of Computer Science and Technology, Shandong University, in 2000 and 2003, respectively, and the Ph.D. degree from the Institute of Network Security, Shandong University, in 2006. He was a Visiting Professor with the Department of Computer Science and Engineering, The State University of New York at Buffalo, Buffalo, NY, USA, from 2013 to 2014. He is currently a Professor with the College of Computer Science and Technology, Qingdao University. His research interests include cloud computing security, key evolving cryptography, digital signature, and network security.

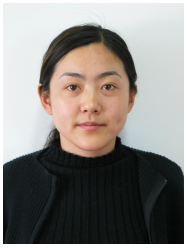


Hanlin Zhang received the B.S. degree in software engineering from Qingdao University in 2010 and the M.S. degree in applied information technology and the Ph.D. degree in information technology from Towson University,Towson, MD, USA, in 2011 and 2016, respectively. He is currently with the College of Computer Science and Technology, Qingdao University, as an Assistant Professor. His research interests include information security, cloud security, mobile security, network security, cloud computing security, key evolving cryptography, digital signature, and network security.

Chengyu Hu received the Ph.D. degree from Shandong University in 2008. He is currently a Lecturer with the School of Software, Shandong University. His main research interests include cloud system security, public key cryptography, and leakage-resilient cryptography.

Zengpeng Li received his PhD degree from Harbin Engineering University. He is currently a Lecturer of Qingdao University. He has worked on lattice based cryptography, password based cryptography, and cryptographic protocol.

Zhan Qin is an assistant professor in the Electrical and Computer Engineering Department of the Texas University at San Antonio, USA. He obtained PhD from the State University of New York at Buffalo in 2017. His current research interests focus on Data Privacy, Secure Computation, Crowdsourcing Data Security and Smart Grid Security.



Rong Hao received the masters degree from the Institute of Network Security, Shandong University.She is currently with the College of Computer Science and Technology, Qingdao University. Her research interest is information security.