2022 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress

Achieving Privacy-preserving Data Sharing for Dual Clouds

Xingqi Luo
Beijing Institute of Technology
Beijing, China
Email: 1120182901@bit.edu.cn

Haotian Wang
University of Pennsylvania
Philadelphia, U.S.A.
E-mail: albertht@sas.upenn.edu

Jinyang Dong✉
Defense Innovation Institute
Chinese Academy of Military Science
Beijing, China
Email: tobev5@163.com

Chuan Zhang
Beijing Institute of Technology
Beijing, China
Email: chuanz@bit.edu.cn

Tong Wu
Beijing Institute of Technology
Beijing, China
Yangtze Delta Region Academy of Beijing Institute of Technology
Jiaxing, China
Email: tongw@bit.edu.cn

***Abstract*—With the advent of the era of Internet of Things (IoT), the increasing data volume leads to storage outsourcing as a new trend for enterprises and individuals. However, data breaches frequently occur, bringing significant challenges to the privacy protection of the outsourced data management system. There is an urgent need for efficient and secure data sharing schemes for the outsourced data management infrastructure, such as the cloud. Therefore, this paper designs a dual-server-based data sharing scheme with data privacy and high efficiency for the cloud, enabling the internal members to exchange their data efficiently and securely. Dual servers guarantee that none of the servers can get complete data independently by adopting secure two-party computation. In our proposed scheme, if the data is destroyed when sending it to the user, the data will not be restored. To prevent the malicious deletion, the data owner adds a random number to verify the identity during the uploading procedure. To ensure data security, the data is transmitted in ciphertext throughout the process by using searchable encryption. Finally, the black-box leakage analysis and theoretical performance evaluation demonstrate that our proposed data sharing scheme provides solid security and high efficiency in practice.**

***Index Terms*—data sharing, dual server, secure two-party computation, searchable encryption**

1. Introduction

As the developement of Internet technologies such as IoT, there is a large amount of data generated by the related applications every day. With the advent of the era of IoT, more and more companies choose to upload massive internal data to the cloud storage, which brings convenience to enterprises along with the risk of data leakage.

In recent years, the data leakage occurs frequently in practice. For example, in 2016, Yahoo announced two major data breaches, which affected more than 1.5 billion account data and reduced its acquisition price by 350 million dollars [1]. In 2018, Cambridge Analytica, in partnership with Trump's election team and the Brexit campaign, collected the profiles of millions of American voters on Facebook, which is one of the most significant data breaches of Facebook [2]. In 2020, the federal and state governments in the United States experienced severe data breaches, affecting more than 1,000 organizations worldwide and launching supply chain attacks against VMware, Microsoft, and other commonly used software [3]. Therefore, it is desirable to design a data storage and sharing scheme for the internal members to protect data security in such outsourced system.

1.1. Related Work and Challenges

In recent years, with the improvement of people's awareness of security and privacy, a large amount of works have focused on data sharing schemes with privacy protection. Wu et al. [4] proposed a data sharing scheme in the electronic medical scenario, using blockchain to ensure data integrity in the process of sharing, and introducing data masking technology to solve the problem of privacy disclosure. But their work will reduce the accuracy of some data. Yang et al. [5] designed a data sharing scheme to ensure data privacy by combining blockchain with attribute-based encryption (ABE) [6]. Tang et al. [7] implemented efficient and privacy-protecting data sharing schemes using ABE and fog nodes. However, ABE is a time-cost cryptographic primitive, which will cause a heavy burden on user side. Shen et al. [8] use a smart contract mechanism to control data access rights and ensure data security sharing. The data sharing scheme proposed by Lu et al. [9] uses the integrity mechanism based on an algebraic signature to ensure data integrity and access control to ensure data privacy. The above two blockchain-based data sharing schemes require third-party trusted institutions to issue keys, resulting in key escrow

978-1-6654-5417-9/22/$31.00 ©2022 IEEE

DOI 10.1109/iThings-GreenCom-CPSCom-SmartData-Cybermatics55523.2022.00059

problems. To acheive the anonymity of data sharing, Lai et al. [10] introduced a data sharing scheme use traceable ring signature and blockchain to secure data sharing. The data sharing mechanism proposed by Wang et al. [11] provides secure data sharing by using blockchain to monitor participants' behaviors, non-interactive zero-knowledge proof, and dynamic pseudo-identity policies to hide the identity of data providers. Zhang et al. [12] proposed a secure multi-hop re-encryption scheme, which continuously hides the re-encryption process to ensure that sensitive information is not leaked, but the data sharing model is one-to-one. Liang et al. [13] proposed a searchable symmetric encryption-based decision tree classification scheme that can effectively protect user privacy. The works of He et al. [14] and Wang et al. [15] were plagued by a large amount of calculation, causeing low search efficiency.

In summary, the existing schemes have some issues, such as reduced data accuracy, low query efficiency, limited in one-to-one sharing model, and the third-party key escrow. Therefore, it is a certain gap on researching data sharing schemes for building an efficient data sharing scheme without lossing the data accuracy and thrid-party issue in one-to-many model. In this paper, we introduce a novel one-to-many data sharing scheme to realize efficient queries, and use double servers to eliminate third-party issue in typical data sharing schemes.

1.2. Contributions

In this paper, we design a locally oriented data storage sharing scheme for internal members to protect data security based on the above situation. The contribution points are as follows:

- Our proposed scheme adopts dual servers to prevent a single server from obtaining complete data. We suggest the secure two-party computation to conduct the communication and data exchange bwtween two server, which prevents the data leakage from the server sides.
- In our proposed scheme, data is transmitted in the form of ciphertext among different parties. We introduce the searchable encryption to ensure the data security during the sharing procedure.
- To prevent malicious deletion of data, our proposed scheme adopts verification mechanism. We introduce random numbers into the index vector to ensure that other users will not maliciously tamper with the data.
- To further strengthen system security, our proposed scheme is to update the key regularly. The user updates retrial vector as well, corresponding to updated key.

Organization. The rest of this paper is organized as follows. In Section 2, we reviews the related cryptographic primitives. Then, we introduce the system model, threat model, and design goals in Section 3. The concrete construction is described in Section 4. In Section 5, we analyze the security of our proposed scheme. Finally, the theoretical analysis of our scheme is given in Section 6.

2. Preliminary

This section reviews some cryptographic primitives, such as secure two-party computation, secret sharing, and searchable encryption.

2.1. Secure Two-party Computation

Secure two-party computation was proposed by Yao in 1982 [16], originally developed to solve the millionaire problem [16].

In this part, we briefly review the typical secure two-party computation in [16]. The parties will communicate with each other as follows. Firstly, the function to be calculated is converted into a computing circuit. Then the one party involved in the calculation constructs the garbled circuit and sends the garbled table to the other party, which communicates through the daze transmission protocol [17]. Then the two parties convert the private value into the garbled value and input it into the garbled circuit. The result is obtained through the garbled circuit and output.

The current example of DDH hypothesis-based construction in [17] is widely used as an instansiation of Yao's scheme.

2.2. Secret Sharing

In this part, we briefly review Shamir's (t, n) threshold secret-sharing scheme. The details are as follows:

- $SS.Share(S, t, n) \rightarrow (s_1, s_2, ..., s_n)$: Split secret S into n pieces, and at least t pieces are needed to restore secret S.
- $SS.Recover(s_1, s_2, ..., s_t) \rightarrow S$: Restore secret S with t secret shares.

2.3. Searchable Encryption

Searchable encryption was proposed in 2000 [18]. Hidden vector encryption (HVE) [19] is a searchable encryption mechanism that supports coalescence, equality, comparison, and subset query of encrypted data. In our proposed scheme, we will adopt a hidden vector encryption scheme based on symmetric encryption (SHVE) [20] to implement the searchable encryption. The details of SHVE are as follows:

- $SHVE.Setup(1^\lambda) \rightarrow (msk, M)$: On input the security parameter λ, it randomly generates a master secret key $msk \xleftarrow{\$} (0,1)^\lambda$. It then defines the payload message space M, then it outputs (msk, M).
- $SHVE.KeyGen(msk, v) \rightarrow s$: Input the master secret key msk and predicate vector $v = \{v_1, v_2, ..., v_d\}$, and the algorithm returns the decryption key s.

- $SHVE.Enc(msk, \mu \in M, x) \rightarrow c$: On input the master secret key msk, a message μ, and index vector $x = \{x_1, x_2, ..., x_d\}$, and it returns ciphertext c associated with (μ, x).
- $SHVE.Query(c, s) \rightarrow \mu/\perp$: On input the ciphertext c corresponding to the index vector x and the decryption key s corresponding to the predicate vector v. If $P_v^{\mathrm{SHVE}} = 1$, it returns μ; otherwise, it returns $\perp$. For each $v \in \Sigma_*^d$, $x \in \Sigma^d$,

$$P_v^{\mathrm{SHVE}}(x) = \begin{cases} 1 & \forall 1 \leq i \leq d, (v_i = x_i \text{ or} v_i = *) \\ 0 & \text{otherwise} \end{cases}$$

3. Definition and Threat Model

This section introduces the system model, threat model and desgin goals of our proposed scheme, respectively.

3.1. System Model

This system involves four steps, inlcuding data upload, data download, date deletion, and key update. The system model for our scheme consists of two entities, as the server and the user, as shown in figure 1.

- **Server**: The system consists of two servers, S_1 and S_2. The server is responsible for processing the data uploaded by users, encrypting, retrieving, and decrypting the data through secure two-party computing. In addition, the server is also responsible for regularly updating the master key of the system, which is conducive to improving system security and reducing the risk of data breaches.
- **User**: In our scheme, user U_i can upload and delete his/her own data or download the data. As a data owner, U_i can upload and delete his/her data. When the system needs to update the key, the user is to update the retrieval vector of the files. Otherwise, the corresponding file will be deleted. As a data requester, U_i generates a query vector to retrieve data he/she needed.

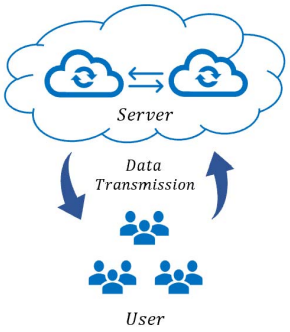

Figure 1. The system model.

3.2. Threat Model

In our scheme, we assume that user U_i is honest but curious (HBC). Users will strictly implement data upload, delete, query, key update and the retrieval vector update, but always keep curious about other's data and make inferences based on their known data.

We assume that the two servers are also honest but curious. The servers will perform each operation according to the steps specified in the scheme. They will be curious about the data they get and deduce it. In addition, server S_1 and server S_2 are not collusive.

Suppose there is a probabilistic polynomial-time (PPT) HBC adversary. There are two types of attack to be considered in our proposed scheme:

1) The first type of attack is corrupting the server. In our scheme, the adversary can corrupt no more than one server. It can retrieve all the data received by the server and try to retrieve the data uploaded by the user.
2) The second type of attack is corrupting the user. The adversary can retrieve all the data owned and received by the user and try to crack the key.

3.3. Design Goal

This section analyzes the privacy requirements that the system needs to meet. As it is a data-sharing scheme for regional internal systems, the following privacy requirements should be ensured:

- Prevent leakage of uploaded data files. Data files are internal information and may involve internal organization secrets. If data files are disclosed to external organizations in plaintext or directly obtained by a third-party server, data leakage may cause severe losses.
- Prevent the encryption key from being cracked. Any participant in the system cannot fully grasp the data encryption key of the system; that is, a single server cannot obtain the complete encryption key, and users cannot obtain the key by any means to prevent corrupt participants from destroying the system.

4. Privacy-preserving Data Sharing for Dual Clouds

In this section, we first introduce the main ideas of the scheme and then explain the functions and technical details of each sub-protocol.

4.1. Workflow

The scheme consists of five steps, namely **Init**, **Upload**, **Download**, **Delete**, and **Update**. Their functions are to initialize the system, upload data, download data, delete data, and update keys, respectively.

1) Both servers run **InitGlobal** to generate the system's master key and an encrypted database. Both servers jointly maintain the database and have copies of it.

2) The user U_i and both servers run **Upload** together. The user divides the data and the corresponding index vector into two parts and sends them to S_1 and S_2. Servers restore and encrypt data to the database through secure two-party computation.

3) U_i runs **Download** with both servers. U_i generates the query vector and sends it in two to S_1 and S_2, respectively. S_1 and S_2 calculate the recovery query vector on the security side, generate the corresponding decryption key, query the ciphertext, send the ciphertext and the corresponding decryption key to the user, and the user recovers the data. The decryption key is valid only for specific ciphertext. Users cannot use the decryption key to decrypt the encrypted data that does not belong to the query scope.

4) U_i runs **Delete** with both servers. U_i uploads the corresponding retrieval vector and random string to the servers. The servers use the retrieval vector to find the corresponding data and confirm whether U_i is the data holder through the random string. If so, the data will be deleted.

5) The system periodically runs **Update** to update keys.

4.2. Details of the Scheme

In this section, we describe the implementation steps of each sub-protocol in technical detail.

Initialization. The protocol is used to initialize the system and generate the master key of the system and an empty encrypted database. The detailed steps are shown in the figure 2.

Upload. The protocol is used to upload data to the encrypted database EDB, which is run by the user U_i and S_1 and S_2. The detailed steps are shown in the figure 3.

Download. This protocol is used to download the required data and is run by both the user and the two servers. The detailed steps are shown in the figure 4.

Delete Data. U_i can delete data from the system by executing the Delete protocol with both servers. The detailed steps are shown in the figure 5.

Update Key. To improve system security and reduce the possibility of password cracking, the system periodically runs the Update protocol to replace the encryption key. The technical details are shown in the figure 6.

5. Security Analysis

In this section, we prove the security of our proposed scheme by black-box leakage analysis with an ideal/real-world paradigm [21]. Assume that the scenario is run in PPT environment $\mathcal{Z}$, with two servers, S_1 and S_2, $n = poly(\lambda)$ users as participants, and adversary $\mathcal{A}$. Suppose $\mathcal{A}'$ is a simulation of $\mathcal{A}$ in the ideal environment.

Real-world execution. At the outset, environment $\mathcal{Z}$ selects a string $str \in \{0,1\}^*$ as input and selects a participant set I, who are externally/internally corrupted by $\mathcal{A}$. At the end of the above process, S_1 and S_2 execute $InitGlobal$ to initialize the system.

After initialization, $\mathcal{Z}$ adaptively selects a polynomial number of commands $(comm_1, \cdots, comm_k)$, where $comm_j = (U_j, OP_j)$, U_j is the execution object of the command,OP_j is the operation that U_j needs to perform, that is, upload data $(Upload, (X_j, File_j))$, download data $(Download, q_j)$, delete data $(Delete, X_j)$, and update key $(Update, \mathcal{X})$. When the command is received, user U_j runs the corresponding protocol with both servers to get the result.

After each command is executed, if a result is returned, the results are sent to the environment $\mathcal{Z}$. After all commands have been executed, $\mathcal{A}$ sends any message to $\mathcal{Z}$, which outputs 1 bit, $Real_{\mathcal{Z},\mathcal{A}}(2\lambda)$.

Ideal execution. Firstly, we define the leakage profile to simulate a real-world data leakage, namely $\Lambda = (\mathcal{L}_{init}, \mathcal{L}_{up}, \mathcal{L}_{down}, \mathcal{L}_{de}, \mathcal{L}_{update})$. The dataset EDB_s is a simulation of a real-world encrypted database EDB. Then we define the function F_{Ideal} as follows.

- $InitGlobal_s$: When receiving the initialization command, F_{Ideal} sends the message "initGlobal Completed" to S_1 and S_2, and $\mathcal{L}_{init}(1^{2\lambda})$to $\mathcal{A}'$.
- $Upload_s$: When receiving the upload command $(Upload, (X_j, File_j))$ from U_j, store $(X_j, File_j)$ in EDB_s, send the message "j Upload completed" to S_1 and S_2, and send $\mathcal{L}_{up}(EDB_s, File_j)$to $\mathcal{A}'$.
- $Download_s$: When receiving the upload command $(Download, q_j)$ from U_j, search for all data matching q_j and return the matching data set F to U_j. Send the message "j Download completed" to S_1 and S_2. If the server is corrupted, send $(j, \mathcal{L}_{down}(EDB_s, q_j))$ to A', if the user is corrupted, send $(j, q_j, F, \mathcal{L}_{down}(EDB_s, q_j))$ to $\mathcal{A}'$.
- $Delete_s$: When receiving the delete command $(Delete, X_j)$ from U_j, check whether X_j contains wildcard $*$. If so, send the message "j delete fail" to S_1 and S_2. If not, the data matching X_j is searched and deleted, and the message "j delete completed" is sent to S_1 and S_2. If the server is corrupted, send $(j, \mathcal{L}_{de}(EDB_s, X_j))$ to A', if the user is corrupted, send $(j, X_j, \mathcal{L}_{de}(EDB_s, X_j))$to $\mathcal{A}'$.
- $Update_s$: When receiving the update command $(Update, \mathcal{X})$, send the message "update completed" to S_1 and S_2. If the server is corrupted, send $\mathcal{L}_{update}(1^{2\lambda}, EDB_s, \mathcal{X})$ to A'; if the user is corrupted and is a data owner, send $(j, \mathbf{X}_j, \mathcal{L}_{update}(1^{2\lambda}, EDB_s, \mathcal{X}))$ to $\mathcal{A}'$, where, j is the corrupted user id, and $\mathbf{X}_j$ is the index vector uploaded by the user.

The procedure in the ideal environment is the same as that in the real-world environment. After all commands have been executed, $\mathcal{A}'$ sends any message to $\mathcal{Z}$, which outputs 1 bit. We define an abstract leakage profile for subsequent

InitGlobal:

S_1 and S_2 perform secure two-party computation, compute $(MSK_1, MSK_2, EDB) \leftarrow f(r_1, r_2)$, $r_1 \xleftarrow{\$} (0,1)^{2\lambda}, r_2 \xleftarrow{\$} (0,1)^{2\lambda}$. λ is the security parameter. S_1 and S_2 store r_1 and r_2 respectively. $f(r_1, r_2)$:

1) Compute $MSK \leftarrow SHVE.Setup(r_1 \oplus r_2)$.
2) Compute $(MSK_1, MSK_2) \leftarrow SS.Share(MSK, 2, 2)$.
3) Create an empty encrypted database EDB.
4) Output (EDB, MSK_1) to S_1, (EDB, MSK_2) to S_2.

Figure 2. The details of InitGlobal.

Upload:

Servers S_1, S_2, and user U_i do the following steps to upload data:

1) U_i computes
$$((\mathbf{X}_{i1}, File_{i1}), (\mathbf{X}_{i2}, File_{i2})) \leftarrow SS.Share((\mathbf{X}_i, File_i), 2, 2).$$
$\mathbf{X}_i = \mathbf{x}_i || r_i$, where $\mathbf{x}_i$ is the index vector corresponding to $File_i$ and $r_i \xleftarrow{\$} (0,1)^{\lambda}$ is a random string used to verify identity, both generated by U_i.
2) U_i sends $(\mathbf{x}_{i1}, File_{i1})$ to S_1, $(\mathbf{x}_{i1}, File_{i1})$ to S_2.
3) S_1 and S_2 perform secure two-party computation, compute:
$$EFile_i \leftarrow f((\mathbf{x}_{i1}, File_{i1}, MSK_1), (\mathbf{x}_{i2}, File_{i2}, MSK_2)).$$
$f((\mathbf{x}_{i1}, File_{i1}, MSK_1), (\mathbf{x}_{i2}, File_{i2}, MSK_2))$:
   a) Compute $(\mathbf{x}_i, File_i, MSK) \leftarrow SS.Recover((\mathbf{x}_{i1}, File_{i1}, MSK_1), (\mathbf{x}_{i2}, File_{i2}, MSK_2))$
   b) Compute $EFile_i \leftarrow SHVE.Enc(MSK, File_i, \mathbf{x}_i)$

Figure 3. The details of Upload.

Download:

User U_i and the two servers perform the following steps to retrieve the required data securely:

1) User U_i generates query vector $\mathbf{q}_i$ and computes $(q_{i1}, q_{i2}) \leftarrow SS.Share(q_i, 2, 2)$, sends q_{i1} to S_1 and q_{i2} to S_2.
2) S_1 and S_2 perform secure two-party computation, compute
$$((tk_1, EF_1), (tk_2, EF_2)) \leftarrow f((MSK_1, q_{i1}), (MSK_2, q_{i2})).$$
$f((MSK_1, q_{i1}), (MSK_2, q_{i2}))$:
   a) Compute $(MSK, q_i) \leftarrow SS.Recover((MSK_1, q_{i1}), (MSK_2, q_{i2}))$.
   b) Compute $tk \leftarrow SHVE.KeyGen(MSK, q_i || *^{\lambda})$, where $*$ is a wildcard.
   c) For $\forall EFile_i \in EDB$, compute $flag \leftarrow SHVE.Query(EFile_i, tk)$, if $flag \neq \perp$, add $EFile_i$ into EF_i.
   d) Compute $(tk_1, tk_2) \leftarrow SS.Share(tk, 2, 2; r_1 \oplus r_2)$, $(EF_1, EF_2) \leftarrow SS.Share(EF, 2, 2; r_1 \oplus r_2)$
   e) Output (tk_1, EF_1) to S_1 and (tk_2, EF_2) to S_2.
3) S_1 sends (tk_1, EF_1) to U_i and S_2 sends (tk_2, EF_2) to U_i.
4) U_i computes $tk \leftarrow SS.Recover(tk_1, tk_2), EF \leftarrow SS.Recover(EF_1, EF_2)$ and $F \leftarrow SHVE.Query(EF, tk)$.

Figure 4. The details of Download.

proof, as

$$\begin{aligned}\Lambda &= (\mathcal{L}_{init}, \mathcal{L}_{up}, \mathcal{L}_{down}, \mathcal{L}_{de}, \mathcal{L}_{update}) \\ &= (\mathcal{L}^I_{initial}, \mathcal{L}^I_{add}, \mathcal{L}^I_{query}, \mathcal{L}^I_{delete}, (\mathcal{L}^I_{query}, \mathcal{L}^I_{add}, \mathcal{L}^I_{delete})) \\ &= (patt^I_{initial}, patt^I_{add}, patt^I_{query}, (patt^I_{query}, patt^I_{add}, \\ &\quad patt^I_{delete})).\end{aligned}$$

I means Ideal World, $\mathcal{L}$ means leakage profile, and $patt$ means leakage pattern.

Theorem 1. *Our proposed scheme is λ-secure if for all PPT adversary $\mathcal{A}$, there exists a PPT adversary $\mathcal{A}'$ such that in all PPT independent environment $\mathcal{Z}$, for all $str \in \{0,1\}^*$,*

$$|\Pr[Real_{\mathcal{Z},A}(\lambda) = 1] - \Pr[Ideal^{\Lambda}_{\mathcal{Z},A'}(\lambda) = 1]| \leq negl(\lambda)$$

Proof. First, we simulate the view of adversary $\mathcal{A}$ ideally and then prove that there is no difference between the view of the adversary in the real world and the view of the adversary in the ideal world, thus proving that if SS is λ-security, our scheme is also λ-security. $\mathcal{A}'$ is a simulator of $\mathcal{A}$ in the ideal world. $\mathcal{S}_I$ is a simulator of ideal environment satisfying λ-security. Let us talk about it by case.

Delete:
User U_i and the two servers perform the following steps to delete related data securely:

1) User U_i computes $(\mathbf{X}_{i1}, \mathbf{X}_{i2}) \leftarrow SS.Share(\mathbf{X}_i, 2, 2)$, where $\mathbf{X}_i = \mathbf{x}_i || r_i$, $\mathbf{x}_i$ is the index vector of the data to be deleted, and r_i is the random string uploaded for verification when the user uploads data.
2) U_i sends $\mathbf{X}_{i1}$ to S_1 and $\mathbf{X}_{i2}$ to S_2.
3) S_1 and S_2 perform secure two-party computation, compute
   a) Compute $(\mathbf{X}_i, MSK) \leftarrow SS.Recover((\mathbf{X}_{i1}, MSK_1), (\mathbf{X}_{i2}, MSK_2))$.
   b) Compute $tk_{Delete} \leftarrow SHVE.KeyGen(MSK, \mathbf{X}_i)$, and make sure there is no wildcard $*$ in $\mathbf{X}_i$; otherwise, end the delete operation as U_i is not the owner of the data to be deleted.
   c) For $\forall EFile_i \in EDB$, compute $flag \leftarrow SHVE.Query(EFile_i, tk_{Delete})$, if $flag \neq \perp$, delete $EFile_i$.
4) S_1 sends message "Delete Completed" to U_i.

Figure 5. The details of Delete.

Update:
Both servers and all data owner users execute the protocol to update the key as follows:

1) Two servers issue an update request to the data owners, and the data owners sends the data index vector to S_1 and S_2, assuming the index set is $\mathcal{X} = \{\mathbf{X}_1, ..., \mathbf{X}_m\}$ (if the user does not want to save a file, the corresponding index vector of the file can not be uploaded, which is equivalent to deleting the file). Users compute $(\mathcal{X}_1, \mathcal{X}_2) \leftarrow SS.Share(\mathcal{X}, 2, 2)$, and send $\mathcal{X}_1$ to S_1 and $\mathcal{X}_2$ to S_2.
2) S_1 and S_2 perform secure two-party computation, compute

$$(MSK_1, MSK_2, EDB') \leftarrow f((\mathcal{X}_1, r_1', MSK_1), (\mathcal{X}_2, r_2', MSK_2)), r_1' \xleftarrow{\$} (0,1)^{2\lambda}, r_2' \xleftarrow{\$} (0,1)^{2\lambda}.$$

   S_1 and S_2 store r_1' and r_2' respectively. $f((\mathcal{X}_1, r_1', MSK_1), (\mathcal{X}_2, r_2', MSK_2))$:
   a) Compute $MSK' \leftarrow SHVE.Setup(r_1' \oplus r_2')$.
   b) Compute $\mathcal{X} \leftarrow SS.Recover(\mathcal{X}_1, \mathcal{X}_2)$, $MSK \leftarrow SS.Recover(MSK_1, MSK_2)$.
   c) For $\forall \mathbf{x}_i \in \mathcal{X}$, compute

$$tk \leftarrow SHVE.KeyGen(\mathbf{x}_i, MSK),$$

$$File_i \leftarrow SHVE.Query(EFile_i, tk),$$

$$EFile_i' \leftarrow SHVE.Enc(MSK', File_i, \mathbf{x}_i), File_i \neq \perp .$$

   d) Compute $(MSK_1', MSK_2') \leftarrow SS.Share(MSK', 2, 2)$.
   e) Output (EDB', MSK_1') to S_1, (EDB', MSK_2') to S_2.
3) S_1 deletes r_1 and EDB. S_2 deletes r_2 and EDB.

Figure 6. The details of Update.

Server Corruption. The server participates in the execution of all sub-protocols, so $\mathcal{A}'$ can access all the information in the process. The simulation is as follows:

- Simulation of InitGlobal: Simulate a secure two-party computation, compute

$$EDB \leftarrow \mathcal{S}_I(\mathcal{L}^I_{initial}(1^{2\lambda})),$$

$$(MSK_1, MSK_2) \leftarrow SS.Share(0^{2\lambda}, 2, 2).$$

  Send MSK_1 to S_1 and MSK_2 to S_2.
- Simulation of Upload: Simulate a secure two-party computation, compute

$$EFile_i \leftarrow \mathcal{S}_I(\mathcal{L}^I_{add}(MSK, File_i, \mathbf{x}_i).$$

  Send $EFile_i$ to S_1 and S_2.
- Simulation of Download: Simulate a secure two-party computation, compute

$$(tk, EF) \leftarrow \mathcal{S}_I(\mathcal{L}^I_{query}(MSK, q_i)),$$

$$(tk_1, tk_2) \leftarrow SS.Share(tk, 2, 2),$$

$$(EF_1, EF_2) \leftarrow SS.Share(EF, 2, 2).$$

  Send (tk_1, EF_1) to S_1 and (tk_2, EF_2) to S_2.
- Simulation of Delete: Simulate a secure two-party computation, compute

$$tk_{Delete} \leftarrow \mathcal{S}_I(\mathcal{L}^I_{delete}(MSK, \mathbf{X}_i))$$

- Simulation of Update: Compute

$$(MSK_1', MSK_2') \leftarrow SS.Share(1^{2\lambda}, 2, 2).$$

  Simulate a secure two-party computation, compute

$$EDB' \leftarrow \mathcal{S}_I(\mathcal{L}_{update}(\mathcal{X}), MSK, MSK', EDB).$$

  Send EDB' to S_1 and S_2.

Next, we use the following game sequence to prove that $\mathcal{A}$'s view of the ideal world is no different from that of the real world.

- $G(0)$: Run the $Real_{\mathcal{Z},\mathcal{A}}(2\lambda)$ experiment.
- $G(1)$: Simulate secure two-party computation, with the rest aligned with $G(0)$. Obviously, the adversary's view will not be affected by this change.
- $G(2)$: Replace $r_1 \oplus r_2$ with the 2λ-bit random string r in $InitGlobal$. The rest is the same as $G(1)$. Since $\mathcal{A}'$ corrupts at most one server, and the distribution probability of r and $r_1 \oplus r_2$ is the same, $\mathcal{A}'$ cannot tell the difference between them, ensuring that the change does not affect the view of $\mathcal{A}'$.
- $G(3)$: Replace $r_1 \oplus r_2$ with the 2λ-bit random string r in $Download$. The rest is the same as $G(2)$. Similarly, the change does not affect the view of $\mathcal{A}'$.
- $G(4)$: Replace $r'_1 \oplus r'_2$ with the 2λ-bit random string r' in $Update$. The rest is the same as $G(3)$. Similarly, the change does not affect the view of $\mathcal{A}'$.
- $G(5)$: Replace the encrypted database EDB with the following simulated database:
  - In $InitGlobal$, replace MSK and EDB with $1^{2\lambda}$,
    $$EDB \leftarrow \mathcal{S}_I(\mathcal{L}^I_{initial}(1^{2\lambda})).$$
  - In $Upload$, replace $EFile_i$ with
    $$EFile_i \leftarrow \mathcal{S}_I(\mathcal{L}^I_{add}(MSK, File_i, \mathbf{x}_i).$$
  - In $Download$, replace (tk, EF) with
    $$tk_{Delete} \leftarrow \mathcal{S}_I(\mathcal{L}^I_{delete}(MSK, \mathbf{X}_i)).$$
    $$(tk, EF) \leftarrow \mathcal{S}_I(\mathcal{L}^I_{query}(MSK, q_i)),$$
  - In $Delete$, replace tk_{Delete} with
    $$tk_{Delete} \leftarrow \mathcal{S}_I(\mathcal{L}^I_{delete}(MSK, \mathbf{X}_i)).$$
  - In $Update$, relapce EDB' and MSK' with
    $$EDB' \leftarrow \mathcal{S}_I(\mathcal{L}_{update}(\mathcal{X}), MSK, MSK', EDB),$$
    and $1^{2\lambda}$.

  The rest is the same as $G(4)$. The replacement of MSK does not affect $\mathcal{A}'$'s view. In addition, the λ-security of ideal environment ensures that all simulated upload, download, delete and update operations are indistinguishable from real-world environment. Thus, $G(5)$ is equivalent to F_{Ideal}.

User Corruption. If the user is not a data owner, the user only participates in the Download protocol. $\mathcal{A}'$ can only obtain the data obtained by the user. Emulation can be done during each $Download$ by simulating secure two-party computation, computing $(tk, EF) \leftarrow \mathcal{S}_I(\mathcal{L}^I_{query}(MSK, q_i))$, $(tk_1, tk_2) \leftarrow SS.Share(tk, 2, 2)$ and $(EF_1, EF_2) \leftarrow SS.Share(EF, 2, 2)$, and sending $((tk_1, tk_2), (EF_1, EF_2))$ to U_i. $\mathcal{A}'$ can get (q_i, tk, EF) from U_i.

If the user is a data owner, the simulation is as follows:

- Simulation of Upload: Sample and store $X_i \xleftarrow{\$} \{0,1\}^{2\lambda}$ as the data index vector of U_i. $\mathcal{A}'$ can get $(File_i, \mathbf{X}_i)$ from U_i.
- Simulation of Delete: $\mathcal{A}'$ can get $\mathbf{X}_i$ from U_i. U_i sends $\mathbf{X}_i$ to S_1 and S_2.
- Simulation of Update: U_i sends $\mathbf{X}_i$ to S_1 and S_2. $\mathcal{A}'$ can get $\mathbf{X}_i$ from U_i.

Next, we use the following game sequence to prove that $\mathcal{A}$'s view of the ideal world is no different from that of the real world.

- $G(0)$ and $G(1)$ are the same as in the server corruption case.
- $G(2)$: Compute
  $$(tk', EF') \leftarrow \mathcal{S}_I(\mathcal{L}^I_{query}(MSK, q_i)),$$
  $$(tk'_1, tk'_2) \leftarrow SS.Share(tk, 2, 2),$$
  $$(EF'_1, EF_{,2}) \leftarrow SS.Share(EF, 2, 2).$$
  In $Download$, use (tk'_1, tk'_2) and (EF'_1, EF'_2) instead of (tk_1, tk_2) and (EF_1, EF_2). The rest is the same as $G(1)$. Since I is λ-security, this guarantees that the above changes will not affect the view of $\mathcal{A}'$.
- $G(3)$: Compute
  $$\mathcal{S}_I(\mathcal{L}^I_{delete}(MSK, \mathbf{X}_i))$$
  instead of $Delete$. The rest is the same as $G(2)$. Since I is λ-security, this guarantees that the above changes will not affect the view of $\mathcal{A}'$.
- $G(4)$: Compute
  $$(MSK'_1, MSK'_2) \leftarrow SS.Share(1^{2\lambda}, 2, 2).$$
  $$EDB' \leftarrow \mathcal{S}_I(\mathcal{L}_{update}(\mathcal{X}), MSK, MSK', EDB),$$
  to replace that in $Update$. The rest is the same as $G(3)$. Since I is λ-security, this guarantees that the above changes will not affect the view of $\mathcal{A}'$. Thus, $G(4)$ is equivalent to F_{Ideal}.

□

Notation	Meaning
λ	security parameter
n	number of users
m	number of multiplications over group $\mathbb{G}$
m'	number of non-wildcard elements in a bloom filter [22]
T_M	time of a multiplication
T_E	time of an exponential calculation
T_p	time of a power ooperation
T_P	time of a bilinear pair operation
T_{PRF}	time taken to compute a pseudo-random function
T_{XOR}	time taken to perform an exclusive-or operation over λ
T_{Enc}	time taken to compute a ciphertext
T_{Dec}	time taken to decrypt a ciphertext

TABLE 1. NOTATIONS FOR THEORETICAL ANALYSIS

6. Performance Analysis

We first give a list of notations needed in this section for our theoretical analysis in Table 1.

We use theoretical analysis to evaluate our scheme, evaluate the efficiency of data encryption and decryption, comparing with [10] and [12]. The analysis results are shown in the table 2. According to the time consumption of SHVE's algorithms [20], we can deduce the time cost of our scheme.

Scheme	Data Upload	Data Download
Proposed	$(m)T_{PRF}$	$O(m') + 2\lambda +$ $2((m')T_{XOR} + T_{Dec})$
[10]	$(4n-1)T_M +$ $(4n+6)T_E + T_{Enc}$	$(2n)T_E + 4T_P +$ $2T_{Enc} + T_{Dec}$
[12]	$T_P + nT_p + T_M$	$2(T_P + T_M + 2T_p)$

TABLE 2. TIME COST OF DATA UPLOAD AND DOWNLOAD.

Data upload and data download are the two most frequently used sub-protocols in this solution. It can be seen from Table 2, that the time cost of this scheme in data uploading and downloading is better than [10] and [12]. In our scheme, the time spent uploading and downloading data is independent of the number of users. When there are numerous users, the overhead of [10] and [12] is considerable. Exponential operation and bilinear pair operation are two time-consuming operations. The schemes proposed by [10] and [12] require multiple exponential operations and bilinear pair operations, but our scheme does not use exponential operation, which leads to relatively small time cost.

7. Conclusion

In this paper, we combine secret sharing, secure two-party computation, with searchable encryption to design a localized internal-oriented data sharing scheme, which is secure under certain leakage circumstances. In our proposed scheme, the data privacy holds during the entire sharing procedure. Additionally, our proposed scheme prevents the malicious deletion by adopting verification mechanism. To further strengthen system security, our proposed scheme supports the key update and retrial vector update. In the future, we will consider to improve its architecture and functions and design for other scenarios, such as smart medical care and resident information management.

Acknowledgments

This work is supported by the China Postdoctoral Science Foundation (Grant Nos. 2021TQ0041, 2021TQ0042, 2021M700435), and the National Natural Science Foundation of China (Grant No. 62102027).

References

[1] L. Cheng, F. Liu, and D. Yao, "Enterprise data breach: causes, challenges, prevention, and future directions," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 7, no. 5, p. e1211, 2017.

[2] C. Cadwalladr and E. Graham-Harrison, "Revealed: 50 million facebook profiles harvested for cambridge analytica in major data breach," *The guardian*, vol. 17, p. 22, 2018.

[3] D. V. S. Kaja, Y. Fatima, and A. B. Mailewa, "Data integrity attacks in cloud computing: A review of identifying and protecting techniques," *Journal homepage: www. ijrpr. com ISSN*, vol. 2582, p. 7421, 2022.

[4] S. Wu and J. Du, "Electronic medical record security sharing model based on blockchain," in *Proceedings of the 3rd International Conference on Cryptography, Security and Privacy*, pp. 13–17, 2019.

[5] X. Yang, T. Li, X. Pei, L. Wen, and C. Wang, "Medical data sharing scheme based on attribute cryptosystem and blockchain technology," *IEEE Access*, vol. 8, pp. 45468–45476, 2020.

[6] A. Sahai and B. Waters, "Fuzzy identity-based encryption," *International Conference on Theory Applications of Cryptographic Techniques*, 2005.

[7] W. Tang, J. Ren, K. Zhang, D. Zhang, Y. Zhang, and X. Shen, "Efficient and privacy-preserving fog-assisted health data sharing scheme," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 6, pp. 1–23, 2019.

[8] B. Shen, J. Guo, and Y. Yang, "Medchain: Efficient healthcare data sharing via blockchain," *Applied sciences*, vol. 9, no. 6, p. 1207, 2019.

[9] X. Lu, Z. Pan, and H. Xian, "An efficient and secure data sharing scheme for mobile devices in cloud computing," *Journal of Cloud Computing*, vol. 9, no. 1, pp. 1–13, 2020.

[10] C. Lai, Z. Ma, R. Guo, and D. Zheng, "Secure medical data sharing scheme based on traceable ring signature and blockchain," *Peer-to-Peer Networking and Applications*, pp. 1–15, 2022.

[11] C. Wang, S. Wang, X. Cheng, Y. He, K. Xiao, and S. Fan, "A privacy and efficiency-oriented data sharing mechanism for iots," *IEEE Transactions on Big Data*, 2022.

[12] M. Zhang, Y. Jiang, Y. Mu, and W. Susilo, "Obfuscating re-encryption algorithm with flexible and controllable multi-hop on untrusted outsourcing server," *IEEE Access*, vol. 5, pp. 26419–26434, 2017.

[13] J. Liang, Z. Qin, S. Xiao, L. Ou, and X. Lin, "Efficient and secure decision tree classification for cloud-assisted online diagnosis services," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 4, pp. 1632–1644, 2021.

[14] M. He and D. Xu, "Electronic medical record sharing scheme combining blockchain and searchable encryption," *Computer Engineering and Applications*, vol. 57, no. 21, p. 8, 2021.

[15] Q. Wang, C. Lai, R. Lu, and D. Zheng, "Searchable encryption with autonomous path delegation function and its application in healthcare cloud," *IEEE Transactions on Cloud Computing*, 2021.

[16] A. C. Yao, "Protocols for secure computations," in *23rd annual symposium on foundations of computer science (sfcs 1982)*, pp. 160–164, IEEE, 1982.

[17] C. Peikert, V. Vaikuntanathan, and B. Waters, "A framework for efficient and composable oblivious transfer," in *Annual international cryptology conference*, pp. 554–571, Springer, 2008.

[18] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proceeding 2000 IEEE symposium on security and privacy. S&P 2000*, pp. 44–55, IEEE, 2000.

[19] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Theory of cryptography conference*, pp. 535–554, Springer, 2007.

[20] 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*, pp. 745–762, 2018.

[21] R. Canetti, "Security and composition of multiparty cryptographic protocols," *Journal of CRYPTOLOGY*, vol. 13, no. 1, pp. 143–202, 2000.

[22] 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.