information

MDPI

Article

Multi-User Searchable Symmetric Encryption with Dynamic Updates for Cloud Computing

Chen Guo [1], Xingbing Fu [2,3], Yaojun Mao [1], Guohua Wu [2,*], Fagen Li [4] and Ting Wu [2]

1 School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou 310018, China; gc_sxdt@163.com (C.G.); lslqmyj@163.com (Y.M.)
2 School of Cyberspace, Hangzhou Dianzi University, Hangzhou 310018, China; uestcfuxb@126.com (X.F.); wuting@hdu.edu.cn (T.W.)
3 Guangxi Key Laboratory of Cryptography and Information Security, Guilin 541004, China
4 School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China; fagenli@uestc.edu.cn
* Correspondence: wgh@hdu.edu.cn

Received: 2 September 2018; Accepted: 26 September 2018; Published: 28 September 2018

Abstract: With the advent of cloud computing, more and more users begin to outsource encrypted files to cloud servers to provide convenient access and obtain security guarantees. Searchable encryption (SE) allows a user to search the encrypted files without leaking information related to the contents of the files. Searchable symmetric encryption (SSE) is an important branch of SE. Most of the existing SSE schemes considered single-user settings, which cannot meet the requirements for data sharing. In this work, we propose a multi-user searchable symmetric encryption scheme with dynamic updates. This scheme is applicable to the usage scenario where one data owner encrypts sensitive files and shares them among multiple users, and it allows secure and efficient searches/updates. We use key distribution and re-encryption to achieve multi-user access while avoiding a series of issues caused by key sharing. Our scheme is constructed based on the index structure where a bit matrix is combined with two static hash tables, pseudorandom functions and hash functions. Our scheme is proven secure in the random oracle model.

Keywords: cloud computing; multi-user; searchable symmetric encryption; wireless sensor

1. Introduction

In recent years, more and more users have chosen to outsource files to cloud servers with the popularization of mobile devices (e.g., wireless sensors) and the development of cloud computing, since this can alleviate the local storage pressures and achieve convenient access to data. However, the cloud server is semi-trusted, and users' files may contain sensitive information. To ensure personal privacy and data security, users will encrypt files and outsource the ciphertexts to the cloud server. This is a problem about how to retrieve the ciphertexts stored on the cloud server.

Searchable encryption (SE) provides the possibility of solving such problems, and it is currently being considered for cloud computing and wireless sensor networks. In an SE mechanism, a user (or sensor) first encrypts files with the SE algorithm and then stores the ciphertexts on the cloud server. For the search, the corresponding search token generated by the user (sensor) is sent to the cloud server. With the token, the cloud server executes relevant retrieval operations and returns the matching ciphertexts. The user (sensor) decrypts the ciphertexts to obtain the required files. In the above process, the files are stored in the encrypted form, which reveals no information about the contents of the files. Therefore, SE not only ensures data confidentiality, but also utilizes the powerful computing power of the cloud server. Research on SE mainly include public key encryption with keyword search (PEKS) and searchable symmetric encryption (SSE), which correspond to public-key

cryptography and symmetric-key cryptography, respectively. SSE was first proposed in 2000 [1], then it became one of the important research directions in SE due to its fast computing speeds and small calculation overheads.

A practical SSE scheme should have desirable properties such as dynamism, high efficiency and security. The dynamic scheme is a scheme in which one can update the encrypted file collection. Common dynamic operations include insertion, modification and deletion. In recent years, there have been many dynamic searchable symmetric encryption (DSSE) schemes [2–9] addressing this issue. Efficiency is the focus of all SSE schemes, because overhead is one of the decisive factors in judging whether a scheme is practical or not. There are many factors affecting efficiency, which can be roughly divided into two types: computational complexity and communication complexity. For computational complexity, an SSE scheme should provide fast searches and updates. In addition, due to the rapid development of multi-core processors, a scheme can be parallelizable to improve efficiency. Non-interactive operations are beneficial to reduce communication complexity. A fully-secure SE scheme should meet two requirements as follows: the first is that no information about the contents of the files can be derived from the ciphertexts; the second requirement is that no information be leaked in the retrieval process. It is difficult for SSE to guarantee the above two points. Since the first SSE scheme [1] was proposed, the security definitions have been continuously improved [10–13]. Two security models proposed by Curtmola et al. [12] are widely used as the standard security models for SSE. We use the adaptive model called semantic security (indistinguishability) against adaptive chosen-keyword attacks (IND-CKA2). In addition, the research on security may involve multiple aspects such as trust management [14], data deduplication [15], access control [16] and cloud auditing [17].

Since its introduction, SSE has become more and more important in the cloud environment. A basic function of the cloud platform is data sharing, which enables multiple users to access the files shared by the data owner. Existing SSE schemes mostly focus on single-user access, which means that only one data owner is allowed to access the ciphertexts. Therefore, to solve the problem, multi-user searchable encryption [12] was proposed, which enables a group of users to search and decrypt all the encrypted files stored on the cloud. In early relevant studies, researchers wanted to realize multi-user management and data decryption by sharing the search key and the decrypt key, respectively. However, the users who leave the group still possess the keys, which can cause serious damage if they leak the keys. The keys need to be updated and the files need to be re-encrypted each time the group members change, which brings about significant overheads.

1.1. Related Work

In 2000, Song et al. [1] proposed the first practical SSE scheme, which achieved the target by sequentially scanning ciphertexts. This scheme is vulnerable to adversaries' statistical attacks, and its search complexity is linear in the overall size of files. To increase search efficiency, Goh [10] proposed a secure index scheme, which introduced the Bloom filter [18] as the index of a file. The Bloom filter is the binary data structure that can efficiently and quickly determine whether an element belongs to a collection, but it may draw wrong judgments. With the help of the indexes, the search complexity is linear in the number of files. In addition, Goh introduced a security definition for his scheme, namely IND1-CKA. IND1-CKA ensures the security of indexes, so that attackers do not obtain useful information from the indexes. Unlike the forward index used by Goh, Curtmola et al. [12] proposed a scheme based on the inverted index. The scheme creates an index for each keyword and thus achieves the sublinear and optimized search time. However, it is difficult for the inverted indexes to obtain dynamic updates.

In 2010, van Liesdonk et al. [2] proposed two SSE schemes supporting dynamic updates. Their one scheme requires interactive searches and updates, which increases the amount of data transfers while making processing delay larger. The other scheme is non-interactive, but the overhead grows with update operations. Based on the inverted index, Kamara et al. [3] presented a dynamic scheme, which

achieved optimized search time. However, the scheme will leak information related to search tokens and does not support parallel processing. Subsequently, Kamara and Papamanthou [4] presented a dynamic scheme with a red-black tree as the index. This scheme was parallelizable, and the search complexity was logarithmic in the number of files. However, the scheme has interactive update operations. Recently, there have been several schemes aiming at improving the security performance of DSSE. Stefanov et al. [6] presented a DSSE scheme that supported forward privacy, which had small information leakage. Unlike any of the above schemes, Naveed et al. [7] presented a DSSE scheme based on blind storage. In the scheme, the server can only store and transmit data and cannot perform search operations, so it has better security. In order to achieve security against a malicious server, with the notion of universal composability (UC) [19], Kurosawa and Ohtaki [20] proposed a UC-secure SSE scheme. Then, Kurosawa and Ohtaki [5] proposed a UC-secure DSSE scheme, and the scheme was verifiable. On this basis, Kurosawa et al. [8] achieved better efficiency and security.

In 2006, Curtmola et al. [12] first presented multi-user searchable encryption. Their approach combined broadcast encryption [21] with single-user searchable symmetric encryption. Legitimate users access data with the shared key. Since the keys are the same, there are many problems in practical applications. Unlike key sharing, key distribution [22–24] can provide different keys, which is more suitable for data sharing in multi-user settings. In 2008, Bao et al. [25] designed a PEKSscheme that supports multi-user access, which avoids the use of the same keys. Their scheme uses a bilinear map and hash functions to re-encrypt the search tokens, but a bilinear map requires high overheads. Therefore, Dong et al. [26,27] successively presented two multi-user searchable encryption schemes based on RSA and Elgamal. Those are two PEKS schemes based on proxy cryptography [28]. Subsequent schemes [29–32] have made further studies on multi-user searchable encryption.

To show the advantages of our scheme, we give the comparisons between our scheme and other schemes in Tables 1 and 2.

Table 1. Comparison of different dynamic searchable symmetric encryption (DSSE) schemes. Let n be the maximum number of files, n' the number of files that contain a search keyword, m the maximum number of keywords, m' the number of unique keywords in a file, ι the number of update operations and p the number of parallel processors, respectively.

Schemes	Parallelizable	No Interactive Updates	Update Time	Search Time
van Liesdonk et al. [2]	no	yes	$\iota \cdot O(m')$	$O(n)$
Kamara et al. [3]	no	yes	$O(m')$	$O(n')$
Kamara and Papamanthou [4]	yes	no	$O(\frac{m}{p} \log n)$	$O(\frac{n'}{p} \log n)$
Stefanov et al. [6]	yes	no	$\iota \cdot O(\frac{m'}{p} \log n)$	$O(\frac{n'}{p})$
Kurosawa et al. [8]	yes	no	$O(\frac{m}{p} \log n)$	$O(\frac{n}{p})$
Ours	yes	yes	$O(\frac{m}{p})$	$O(\frac{n}{p})$

Table 2. Feature comparison of multi-user searchable encryption schemes.

Schemes	Type	No Key Sharing	No Bilinear Operation
Curtmola et al. [12]	SSE	no	yes
Bao et al. [25]	PEKS	yes	no
Dong et al. [26]	PEKS	yes	no
Dong et al. [27]	PEKS	yes	yes
Nair and Rajasree [32]	SSE	yes	no
Ours	SSE	yes	yes

1.2. Our Contributions

In this work, we propose a multi-user searchable symmetric encryption scheme, which also achieves efficient updates for encrypted files. The scheme aims at the application scenario where a single data owner encrypts files and shares them among multiple users, and we use key distribution

rather than key sharing to authorize users. Each authorized user has its own search key and decrypt key. The search and decrypt operations can be performed, combined with the complementary keys stored on the cloud server. Our index structure is the combination of a bit matrix and two static hash tables. Such a structure can support secure and efficient searches/updates. In addition, our scheme is parallelizable. We make the contributions as follows:

(1) Our scheme avoids key sharing. For the search process, a search token generated by the search key is processed via re-encryption technology to obtain a new token, which can be used to search in the encrypted index. For the decryption process, we encrypt the key used for encrypting data and upload it to the cloud server. When the ciphertexts are decrypted, with re-encryption technology, the encrypted key is converted into the key that a user can employ.
(2) Our scheme enables efficient searches. To improve efficiency, we use pseudo-random functions and hash functions instead of a bilinear map, which has low efficiency (a bilinear map is often used in multi-user searchable encryption schemes). Searching for a keyword takes $O(n/p)$ parallel time.
(3) Our scheme enables efficient updates. For the update process, the data owner simply sends an update token to the cloud server. Updating a file takes $O(m/p)$ parallel time.
(4) Our scheme meets the security requirements for query privacy, search unforgeability and revocability.

The remainder of this paper is structured as follows. We show some notations and define our system and security requirements in Section 2. Our construction is presented in Section 3. We provide the relevant security analysis and the performance analysis in Section 4 and Section 5, respectively. Finally, we give the conclusion in Section 6.

2. Preliminaries

2.1. Notations

A symmetric encryption scheme is a triplet that contains polynomial-time algorithms (Gen, Enc, Dec). The algorithm $Gen(1^k)$ takes a security parameter k and outputs a secret key K; $Enc(K, f)$ takes a secret key K and a file f and outputs a ciphertext c; $Dec(K, c)$ takes a secret key K and a ciphertext c and outputs a file f. Let ϖ be a symmetric encryption scheme and ε be another symmetric encryption scheme. Note that ϖ and ε are secure (indistinguishable) against chosen-plaintext attacks (IND-CPA). Table 3 gives some notations used in our scheme.

In our scheme, UL is the authorized user list stored on the cloud server, and the entries of the list are tuples (ui, a_{si}, d_{si}), where ui denotes a user identifier and a_{si} and d_{si} denote the user's complementary keys. (a_{ui}, d_{ui}) belongs to a user ui, where a_{ui} and d_{ui} denote the user's search key and decrypt key, respectively. We use a pseudorandom function $P : \{0,1\}^k \times \{w_1, ..., w_m\} \rightarrow \mathbb{Z}_q$, another pseudorandom function $R : \{0,1\}^k \times \{0,1\}^* \rightarrow \{0,1\}^k$ and a collision-resistant hash function $H : G \rightarrow \{0,1\}^k$. $q \in \{w, f_{id}\}$ is the query operation received by the cloud server, where $q = w$ and $q = f_{id}$ denote the search query for keyword w and the update query for file f_{id}, respectively. We write $Q_t = (q_1, ..., q_t)$ to represent a collection of t queries, let $U_q = (u_{q1}, ..., u_{qt})$ be the set of users who make a search query $q = w$, and let $W_t = (w_1, ..., w_t)$ be the set of queried keywords and $\Gamma_t = (\Upsilon_1, ..., \Upsilon_t)$ the set of t replies.

Table 3. Notations.

Notations	Meaning
$\{0,1\}^k$	the set of all k-bit binary strings
$\{0,1\}^*$	the set of all finite binary strings
$y \leftarrow S$	an algorithm S outputs y
$\|b\|$	the bit length of a string b
$\|B\|$	the cardinality of a set B
$y \xleftarrow{R} B$	the element y is randomly and uniformly selected from a set B
w	a unique keyword
f_{id}	a file with identifier id
m	the maximum number of keywords
n	the maximum number of files
δ	an index
γ	the encrypted index
G	a cyclic group of order q
g	a generator of G
$\mathbb{Z}_\mathsf{q}$	an additive group (modulo q)
$\Omega = (w_1, ..., w_m)$	the set of all m keywords
$F = (f_{id_1}, ..., f_{id_n})$	the set of n files
$C = (c_{id_1}, ..., c_{id_n})$	the collection of n corresponding ciphertexts

2.2. Architecture

Figure 1 shows the architecture of our scheme. The data owner outsources his/her own files to the cloud. To assure data security, the files must be stored in the encrypted form. The data owner authorizes a group of users to access the encrypted files stored on the cloud server, and he/she is responsible for updating the encrypted files and managing the group. Each user in the group generates search tokens and decrypts the ciphertexts with his/her own unique search key and decrypt key, respectively. The cloud server executes the retrieval operations with the search token and returns the results.

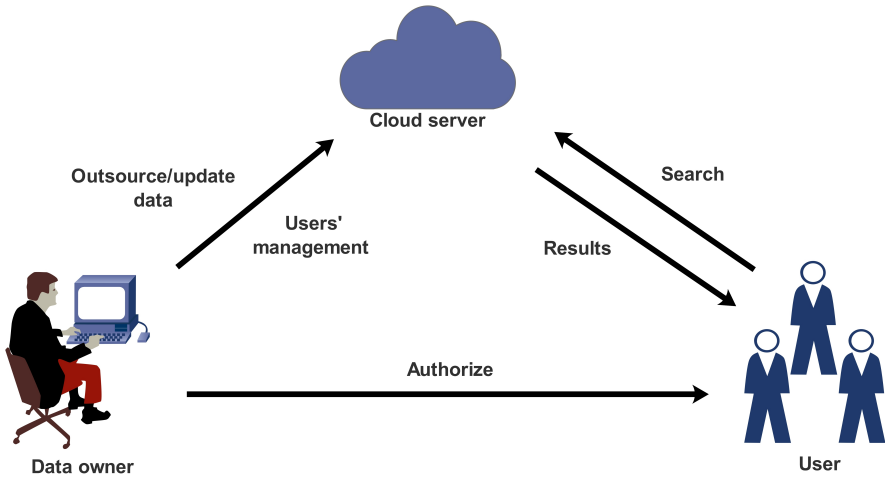

Figure 1. Architecture of multi-user searchable symmetric encryption with dynamic updates.

Definition 1. *A multi-user searchable symmetric encryption scheme is a tuple of polynomial-time algorithms* $(Gen, AddUser, Enc, KeyEnc, SrchToken, Search, KeyDec, Dec, UpdToken, Update, RevokeUser)$ *such that:*

1. $(Params, MK) \leftarrow Gen(1^k)$*: Given a security parameter* k*, the data owner generates the master public parameters* $Params$ *and a master secret key* $MK = (a, d, s_1, s_2, K)$*, where* K *is the secret key for* $\mathcal{O}$*.*

2. $UL \leftarrow AddUser(ui, MK)$: *It is run by the data owner to authorize a new user ui. Using a user identifier ui and a master secret key MK, the data owner produces two key pairs* (a_{si}, a_{ui}) *and* (d_{si}, d_{ui}). *Then,* (ui, a_{si}, d_{si}) *is sent securely to the cloud server, and* (a_{ui}, d_{ui}, s_1) *is sent securely to the user ui. The cloud server updates its authorized user list* $UL = UL \cup (ui, a_{si}, d_{si})$.
3. $(C, \gamma) \leftarrow Enc(MK, F, \delta)$: *Given a master secret key MK, the files F and an index* δ, *the data owner generates the ciphertexts C and the encrypted index* γ. *Then, C and* γ *are uploaded to the cloud server.*
4. $K'' \leftarrow KeyEnc(d, K)$: *It is run by the data owner to encrypt the secret key K. It outputs the encrypted secret key* K'' *for the cloud server.*
5. $T_{ui}(w) \leftarrow SrchToken(a_{ui}, w)$: *Given the user's search key* a_{ui} *and a keyword w, a user ui generates a corresponding search token* $T_{ui}(w)$.
6. $C_w \leftarrow Search(T_{ui}(w), \gamma)$: *Given a search token* $T_{ui}(w)$ *and an encrypted index* γ, *the cloud server returns the results* C_w, *which contain the keyword w.*
7. $K \leftarrow KeyDec(d_{si}, d_{ui}, K'')$: *It is run by the cloud server and a user ui. It takes the user's decrypt key pair* (d_{si}, d_{ui}) *and the encrypted secret key* K'' *as input. It outputs the secret key K for the user ui.*
8. $f_j \leftarrow Dec(K, c_j)$: *Given the secret key K and a ciphertext* c_j, *a user ui gets the file* f_j.
9. $T_f \leftarrow UpdToken(MK, \beta, f_{id})$: *Given a master secret key MK, the type* β *and a file* f_{id}, *the data owner generates an update token* T_f.
10. $(C', \gamma') \leftarrow Update(T_f, C, \gamma)$: *Given an update token* T_f, *the ciphertexts C and an encrypted index* γ, *the cloud server generates the new ciphertexts* C' *and the new encrypted index* γ'.
11. $UL \leftarrow RevokeUser(ui)$: *Given a user identifier ui, the cloud server updates its authorized user list* $UL = UL \setminus (ui, a_{si}, d_{si})$.

Correctness: A multi-user searchable symmetric encryption scheme is correct if for all security parameters k, for all parameters *Params* and keys *MK* from $Gen(1^k)$, for all F, for all tuples (C, γ) from $Enc(MK, F, \delta)$ and for all successive search/update operations on γ, a legitimate user *ui* always gets the correct files F_w, which contain the search keyword w.

2.3. Security Requirements

In our scheme, the cloud server is considered to be semi-trusted, which means that it will gather as much information as possible while complying with the protocol. In addition, we do not consider the case of user-server collusion. Ideally, a secure scheme should leak no information about the plaintexts and queries to malicious attackers. However, a practical scheme will inevitably leak the search pattern and access pattern.

Definition 2. *Search pattern* $\mathrm{P}(\delta, q, t)$: *Given a search query* $q = w$ *at time t, the search pattern is a binary vector of length t with a one at location i if the search at time* $i \leq t$ *was for w; and zero otherwise. The search pattern determines whether the same keyword was searched in the past.*

Definition 3. *Access pattern* $\Delta(F, \delta, w, t)$: *Given a search query* $q = w$ *at time t, the access pattern is the identifiers from* F_w *at time t.*

We define the following leakage functions [13] for our scheme.

Definition 4. *Leakage functions* (ℓ_1, ℓ_2):

1. $\ell_1(F, \delta)$: *It takes the files F (containing their identifiers) and the index* δ *as input.* ℓ_1 *outputs the maximum number of keywords m, the maximum number of files n, the identifiers id of files and the size of each file* $|f_{id}|$. *Specifically,* $\ell_1(F, \delta) = (m, n, id_1, ..., id_n, |f_{id_1}|, ..., |f_{id_n}|)$.
2. $\ell_2(F, \delta, w, t)$: *It takes as input the files F, the index* δ *and a queried keyword w at time t.* ℓ_2 *outputs the search pattern* $\mathrm{P}(\delta, q, t)$, *the access pattern* $\Delta(F, \delta, w, t)$ *and the number of authorized users* $|UL|$. *Specifically,* $\ell_2(F, \delta, w, t) = (\mathrm{P}(\delta, q, t), \Delta(F, \delta, w, t), |UL|)$.

Definition 5. *View of an adversary V_t: Given t queries, the view of an adversary is the transcript of the interactions. Specifically, $V_t = (C, \gamma, id_1, ..., id_n, Q_t, U_q, \Gamma_t, UL)$.*

We first consider the security requirement for query privacy. Apart from the information derived from the view, the adversary should obtain no extra information. We define query privacy as a simulation-based game between an adversary and the challenger. The view of an adversary V_t is from the interactions with the challenger in a real situation, and V_t^* is from the interactions with a simulator in the ideal status. The scheme achieves query privacy if V_t^* and V_t are computationally indistinguishable. Now, we use the notions of dynamic IND-CKA2 [3,4] and give our security definition for query privacy.

Definition 6. *Let MSSEbe a multi-user searchable symmetric encryption scheme as defined in Definition 1. Consider the following probabilistic experiments, where $\mathcal{A}$ is a stateful adversary and S is a stateful simulator:*

$Real_{\mathcal{A}}(k)$: The challenger generates Params and MK by $Gen(1^k)$. Then, $\mathcal{A}$ generates (F, δ) and obtains $(C, \gamma) \leftarrow Enc(MK, F, \delta)$ from the challenger. $\mathcal{A}$ makes a polynomial number of adaptive queries $q \in \{w, f_{id}\}$. If $q = w$, then $\mathcal{A}$ obtains a search token $T_{ui}(w) \leftarrow SrchToken(a_{ui}, w)$ from the challenger. If $q = f_{id}$ with the type β, the challenger generates an update token $T_f \leftarrow UpdToken(MK, \beta, f_{id})$ for $\mathcal{A}$. Eventually, $\mathcal{A}$ returns a bit b as the output of the experiment.

$Ideal_{\mathcal{A},S}(k)$: Given $\ell_1(F, \delta)$, S produces (C^*, γ^*) for $\mathcal{A}$. $\mathcal{A}$ makes a polynomial number of adaptive queries $q^* \in \{w, f_{id}\}$. For each adaptive query, S is given $\ell_2(F, \delta, w, t)$. If $q^* = w$, S generates a search token $T_{ui}^*(w)$. If $q^* = f_{id}$ with the type β, then S returns an update token T_f^*. Eventually, $\mathcal{A}$ returns a bit b as the output of the experiment.

We say that MSSE is (ℓ_1, ℓ_2)-secure against adaptive dynamic chosen-keyword attacks if for all probabilistic polynomial-time (PPT) adversaries $\mathcal{A}$, there exists a PPT simulator S such that:

$$|Pr[Real_{\mathcal{A}}(k) = 1] - Pr[Ideal_{\mathcal{A},S}(k) = 1]| \leq neg(k).$$

Compared with the previous SSE, the multi-user scheme should not only achieve query privacy, but also meet the security requirements for search unforgeability and revocability.

In the multi-user scheme, each user *ui* makes search queries by his/her unique search key. We require that the malicious user $\mathcal{A}_U$ or the cloud server $\mathcal{A}_S$ cannot produce a valid search token $T_{ui}(w)$ on behalf of *ui*. This property is referred to as search unforgeability. For a user *ui*, his/her valid search queries are defined as $Q_{ui} = \{T_{ui}(w) | T_{ui}(w) \leftarrow SrchToken(a_{ui}, w), w \in \Omega\}$. Thus, search unforgeability is defined as that for each authorized user *ui*, adversaries cannot produce $T_{ui}(w) \in Q_{ui}$ without the search key a_{ui}.

For the multi-user application, it is a basic requirement to revoke the access authorities of users as needed. The revoked user cannot access the encrypted files stored on the cloud, which implies he/she is incapable of distinguishing the indexes. Therefore, revocability is defined based on index indistinguishability.

3. Proposed Scheme

3.1. Index Structure

The index δ is an $m \times n$ matrix, and I is another $m \times n$ matrix. We store the encrypted $\delta[i, j]$ in $I[i, j]$, where $\delta[i, j] \in \{0, 1\}$, $I[i, j] \in \{0, 1\}$ for $i = 1, ..., m$ and $j = 1, ..., n$. Two static hash tables are used to uniquely map each keyword-file pair (w, f_{id}) to the indices (i, j) in δ and I. A hash table is composed of tuples $(key, value)$. The *key* is a *k*-bit binary string, and *key* indicates the location in the table. The *value* in the hash table can be accessed in $O(1)$ time. We use a static hash table α_w, which has the tuple (λ_{w_x}, i), where $\lambda_{w_x} = H(h^{\varphi_{w_x}})$, $\varphi_{w_x} = P_{s_1}(w_x)$ for keyword w_x, $i \in \{1, ..., m\}$, $x \in \{1, ..., m\}$.

The access operations can be represented as $i \leftarrow \alpha_w(\lambda_{w_x})$. We use another static hash table α_f, which has the tuple (λ_{f_y}, j), where $\lambda_{f_y} = R_{s_2}(id_y)$ for the identifier id_y of file f_{id_y}, $j \in \{1, ..., n\}$, $y \in \{1, ..., n\}$. Similarly, the access operations can be represented as $j \leftarrow \alpha_f(\lambda_{f_y})$. Our encrypted index γ includes the matrix I and the hash table α_w. Figure 2 shows the index structure.

Note that we write $I[i, *]$ and $I[*, j]$ to represent all elements in the i-th row and the j-th column of the matrix I, respectively. I^T denotes the transpose of the matrix I.

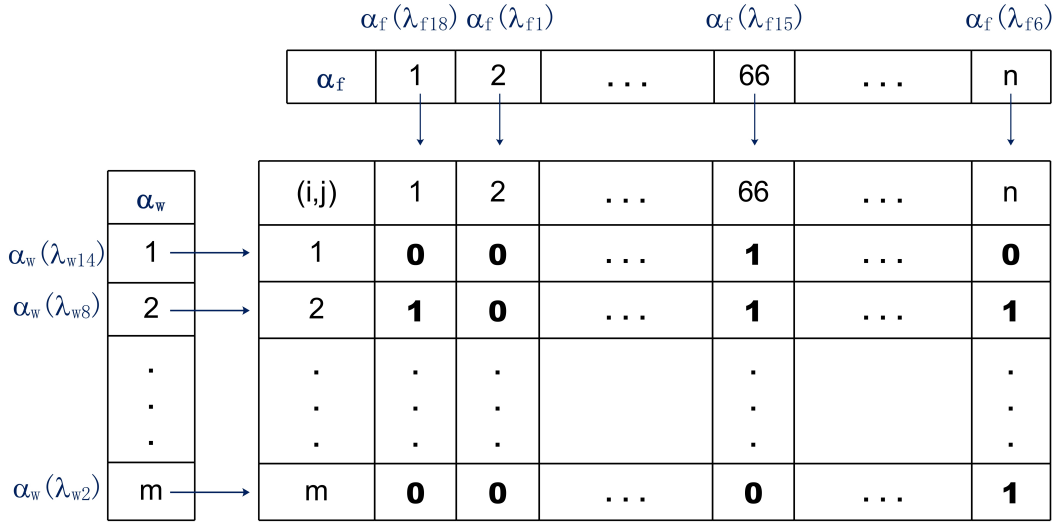

Figure 2. The index structure.

3.2. Concrete Scheme

The construction of our scheme is as follows:

- $(Params, MK) \leftarrow Gen(1^k)$: Given a security parameter k, the data owner generates the master public parameters $Params = (G, g, \mathsf{q}, h = g^a, P, R, H)$ and a master secret key $MK = (a \in \mathbb{Z}_\mathsf{q}, d \in \mathbb{Z}_\mathsf{q}, s_1, s_2, K)$, where $K = g^\mu$, $g^\mu \xleftarrow{R} G$.
- $UL \leftarrow AddUser(ui, MK)$: Given a user identifier ui and a master secret key MK, the data owner generates two key pairs $(a_{si} \xleftarrow{R} \mathbb{Z}_\mathsf{q}, a_{ui} = a - a_{si})$ and $(d_{si} \xleftarrow{R} \mathbb{Z}_\mathsf{q}, d_{ui} \cdot d_{si} = d)$. Then, (ui, a_{si}, d_{si}) is sent securely to the cloud server, and (a_{ui}, d_{ui}, s_1) is sent securely to the user ui. The cloud server updates its authorized user list $UL = UL \cup (ui, a_{si}, d_{si})$. The data owner keeps the master secret key MK.
- $(C, \gamma) \leftarrow Enc(MK, F, \delta)$: The data owner generates the ciphertexts C and an encrypted index γ as follows:

  (1) Initialize two matrices δ and I; all elements are set to zero. Extract all distinct keywords $(w_1, ..., w_{m'})$ from the files $F = (f_{id_1}, ..., f_{id_{n'}})$, where $m' \leq m$ and $n' \leq n$.

  (2) Construct the index δ for $x = 1, ..., m'$ and $y = 1, ..., n'$:

  (a) $\varphi_{w_x} = P_{s_1}(w_x)$, $\lambda_{w_x} = H(h^{\varphi_{w_x}})$, $i \leftarrow \alpha_w(\lambda_{w_x})$, $\lambda_{f_y} = R_{s_2}(id_y)$ and $j \leftarrow \alpha_f(\lambda_{f_y})$.

  (b) If w_x appears in f_{id_y}, set $\delta[i, j] = 1$.

  (3) Encrypt the files F for $y = 1, ..., n'$: $c_{id_y} \leftarrow \varpi.Enc(K, f_{id_y})$, and set $C = C \cup (c_{id_y}, j)$.

  (4) Encrypt the index δ for $x = 1, ..., m$: $\varphi_{w_x} = P_{s_1}(w_x)$, $\lambda_{w_x} = H(h^{\varphi_{w_x}})$, $i \leftarrow \alpha_w(\lambda_{w_x})$, $I[i, *] \leftarrow \varepsilon.Enc(h^{\varphi_{w_x}}, \delta[i, *])$.

(5) Then, (C, γ) are uploaded to the cloud server, where $\gamma = (I, \alpha_w)$. The data owner keeps (α_w, α_f).

- $K'' \leftarrow KeyEnc(d, K)$: The data owner generates the encrypted secret key $K'' = K^d = g^{\mu d}$. Then, K'' is sent to the cloud server.
- $T_{ui}(w) \leftarrow SrchToken(a_{ui}, w)$: With a random number $r \xleftarrow{R} \mathbb{Z}_q$, a user ui produces a search token $T_{ui}(w) = (\tau_1, \tau_2)$, where $\tau_1 = g^{-r} g^{\varphi_w}$, $\tau_2 = h^r g^{-a_{ui} r} g^{a_{ui} \varphi_w}$, $\varphi_w = P_{s_1}(w)$ for the given keyword w.
- $C_w \leftarrow Search(T_{ui}(w), \gamma)$: On receiving a search token $T_{ui}(w) = (\tau_1, \tau_2)$, the cloud server generates the results C_w as follows:

(1) If (ui, a_{si}) can be found in authorized user list UL, compute $\tau_w = \tau_1^{a_{si}} \cdot \tau_2 = h^{\varphi_w}$, $\lambda_{w_x} = H(\tau_w), i \leftarrow \alpha_w(\lambda_{w_x})$. If not, output *Error*.

(2) For $j = 1, ..., n$, compute $I'[i, j] \leftarrow \varepsilon.Dec(\tau_w, I[i, j])$, if $I'[i, j] = 1$, set $C_w = C_w \cup c_j$, where c_j denotes the ciphertext associated with j.

(3) Output C_w. The cloud server returns C_w to the user ui.

- $K \leftarrow KeyDec(d_{si}, d_{ui}, K'')$: Given the encrypted secret key K'', the cloud server finds the corresponding (ui, d_{si}) for a user ui and computes $K' = (K'')^{d_{si}^{-1}} = g^{\mu d_{ui}}$. For K', the user ui uses the decrypt key d_{ui} to get $K = (K')^{d_{ui}^{-1}} = g^{\mu}$.
- $f_j \leftarrow Dec(K, c_j)$: A user ui gets the file as $f_j \leftarrow \varpi.Dec(K, c_j)$.
- $T_f \leftarrow UpdToken(MK, \beta, f_{id})$: The data owner generates an update token T_f for a file f_{id} as follows:

(1) Initialize two arrays $\bar{\delta}[i]$ and $\bar{I}[i]$ for $i = 1, ..., m$; all elements are set to zero. Compute $\lambda_{f_y} = R_{s_2}(id), j \leftarrow \alpha_f(\lambda_{f_y})$.

(2) If the type β is insertion or modification:

a) Extract all distinct keywords $(w_1, ..., w_\beta)$ from the file f_{id}.

b) For $x = 1, ..., \beta$, compute $\varphi_{w_x} = P_{s_1}(w_x)$, $\lambda_{w_x} = H(h^{\varphi_{w_x}}), i \leftarrow \alpha_w(\lambda_{w_x}), \bar{\delta}[i] = 1$.

c) $c_{id_y} \leftarrow \varpi.Enc(K, f_{id}), c = (c_{id_y}, j)$.

d) For $x = 1, ..., m$, compute $\varphi_{w_x} = P_{s_1}(w_x)$, $\lambda_{w_x} = H(h^{\varphi_{w_x}})$, $i \leftarrow \alpha_w(\lambda_{w_x})$, $\bar{I}[i] \leftarrow \varepsilon.Enc(h^{\varphi_{w_x}}, \bar{\delta}[i])$.

(3) If the type β is deletion: for $x = 1, ..., m$, compute $\varphi_{w_x} = P_{s_1}(w_x)$, $\lambda_{w_x} = H(h^{\varphi_{w_x}})$, $i \leftarrow \alpha_w(\lambda_{w_x})$, $\bar{I}[i] \leftarrow \varepsilon.Enc(h^{\varphi_{w_x}}, \bar{\delta}[i])$.

(4) Output $T_f = (c, j, \bar{I})$. Then, T_f is sent to the cloud server.

- $(C', \gamma') \leftarrow Update(T_f, C, \gamma)$: On receiving an update token $T_f = (c, j, \bar{I})$, the cloud server performs the update operation as follows:

(1) $I[*, j] = (\bar{I})^T$.

(2) Output the new ciphertexts C' and the new encrypted index $\gamma' = (I, \alpha_w)$.

- $UL \leftarrow RevokeUser(ui)$: Given a user identifier ui, the cloud server updates its authorized user list $UL = UL \setminus (ui, a_{si}, d_{si})$.

3.3. Correctness

Theorem 1. *Our scheme as described above is correct.*

Proof. Given an encrypted index $\gamma = (I, \alpha_w)$, then $I[i,j] \leftarrow \varepsilon.Enc(h^{\varphi_w}, \delta[i,j])$ and $i \leftarrow \alpha_w(H(h^{\varphi_w}))$, $\varphi_w = P_{s_1}(w)$. With the search key a_{ui}, a user ui generates a search token $T_{ui}(w') = (\tau_1, \tau_2)$ for a keyword w'. The cloud server uses the complementary key a_{si} to compute $\tau_{w'} = \tau_1^{a_{si}} \cdot \tau_2 = h^{\varphi_{w'}}$. We use a collision-resistant hash function H. If $w' = w$, then $H(h^{\varphi_{w'}}) = H(h^{\varphi_w})$. With $i \leftarrow \alpha_w(H(h^{\varphi_{w'}}))$, the cloud server gets the decrypted index $I'[i,j] \leftarrow \varepsilon.Dec(h^{\varphi_{w'}}, I[i,j])$. Thus, the algorithm $Search(T_{ui}(w), \gamma)$ generates the correct results $C_{w'}$ by properly decrypting row i. Finally, with the decrypt key pair (d_{si}, d_{ui}), the user ui gets the secret key K to decrypt $C_{w'}$. □

4. Security Analysis

4.1. Query Privacy

Theorem 2. *Our scheme as described above is (ℓ_1, ℓ_2)-secure in the random oracle model according to Definition 6.*

Proof. During the interactions with the challenger, an adversary $\mathcal{A}$ generates a view V_t, and a simulator S can also simulate a view V_t^* by using the information that will be allowed to leak. Now, we show that V_t^* is computationally indistinguishable from V_t.

For the real view of $\mathcal{A}$, $V_t = (C, \gamma, id_1, ..., id_n, Q_t, U_q, \Gamma_t, UL)$, S simulates the view $V_t^* = (C^*, \gamma^*, id_1, ..., id_n, Q_t^*, U_q^*, \Gamma_t^*, UL^*)$ in the ideal situation.

For $t = 0$, given $\ell_1(F, \delta) = (m, n, id_1, ..., id_{n'}, |f_{id_1}|, ..., |f_{id_{n'}}|)$, $V_t^* = (C^*, \gamma^*, id_1, ..., id_{n'}, UL^*)$ is generated as follows. S simulates the ciphertexts C^* by using the symmetric encryption ϖ. To generate the encrypted index $\gamma^* = (I^*, \alpha_w^*)$, S first constructs the hash table α_f^* and the index δ^*, where all elements of δ^* are randomly set to zero or one. With m randomly selected keys φ_w^*, S constructs the hash table α_w^* and encrypts δ^* for $j = 1, ..., n$: $I^*[\alpha_w^*(H(h^{\varphi_w^*})), j] \leftarrow \varepsilon.Enc(h^{\varphi_w^*}, \delta^*[\alpha_w^*(H(h^{\varphi_w^*})), j])$. To construct the authorized user list UL^* based on the number of authorized users $|UL|$, S generates a random user identifier and the complementary keys $a_{si}^* \xleftarrow{R} \mathbb{Z}_q$, $d_{si}^* \xleftarrow{R} \mathbb{Z}_q$ for each user. The security of the ciphertexts is based on symmetric encryption, so C^* is computationally indistinguishable from C. Symmetric encryption and pseudorandom functions ensure the indistinguishability between γ^* and γ. During the construction of the authorized user list, the complementary keys are randomly assigned to each user, so UL^* and UL are also computationally indistinguishable.

For $t > 0$, given $\ell_2(F, \delta, w, t) = (\mathrm{P}(\delta, q, t), \Delta(F, \delta, w, t), |UL|)$, S simulates the view $V_t^* = (C^*, \gamma^*, id_1, ..., id_n, Q_t^*, U_q^*, \Gamma_t^*, UL^*)$ as follows. Note that all queries in Q_t are issued by different users, and they may make a search query for the same keyword.

- UL^*: The entries of the list are tuples $(ui^*, a_{si}^*, d_{si}^*)$. For $i = 1, ..., |UL|$, S selects the complementary keys $a_{si}^* \xleftarrow{R} \mathbb{Z}_q$ and $d_{si}^* \xleftarrow{R} \mathbb{Z}_q$ for the user ui^*. Set $a^* = a_{s1}^* + ... + a_{s|UL|}^*$, $d^* = d_{s1}^* \times ... \times d_{s|UL|}^*$. Because the complementary keys are randomly assigned to each user, UL^* and UL are computationally indistinguishable.
- C^* and γ^*: Refer to the case of $t = 0$. C^* and C are computationally indistinguishable, and γ^* is computationally indistinguishable from γ.
- Q_t^* and U_q^*: The query operation $q^* \in \{w, f_{id}\}$ includes the search query $q^* = w$ and the update query $q^* = f_{id}$. For $q^* = w$, S randomly selects a user ui^* with its search key $a_{ui}^* = a^* - a_{si}^*$ and its random number $r^* \xleftarrow{R} \mathbb{Z}_q$. With the corresponding key φ_w^*, S generates a search token $T_{ui}^*(w) = (\tau_1, \tau_2)$, where $\tau_1 = g^{-r^*} g^{\varphi_w^*}$, $\tau_2 = h^{r^*} g^{-a_{ui}^* r^*} g^{a_{ui}^* \varphi_w^*}$. The randomly selected key φ_w^* and pseudorandom function $\varphi_w = P_{s_1}(w)$ are computationally indistinguishable, so $T_{ui}^*(w)$ is computationally indistinguishable from $T_{ui}(w)$. The user ui^* is randomly selected to generate $T_{ui}^*(w)$, so U_q^* and U_q are also computationally indistinguishable. For $q^* = f_{id}$, if the type β

is insertion or modification, S simulates the ciphertext c^* by using the symmetric encryption ϖ. With m randomly selected keys φ_w^*, S constructs the array $\bar{\delta}^*[i]$ for $i = 1, ..., m$ and encrypts $\bar{\delta}^*$: $\bar{I}^*[\alpha_w^*(H(h^{\varphi_w^*}))] \leftarrow \varepsilon.Enc(h^{\varphi_w^*}, \bar{\delta}^*[\alpha_w^*(H(h^{\varphi_w^*}))])$. Then, S generates an update token $T_f^* = (c^*, j, \bar{I}^*)$, where j can be derived from α_f^* and f_{id}. Therefore, we can see that T_f^* is computationally indistinguishable from T_f.

- Γ_t^*: For $q^* = w$, if w appears in $\Delta(F, \delta, w, t)$, it outputs the corresponding results. Otherwise, $\mathcal{A}$ performs the algorithm $Search(T_{ui}^*(w), \gamma^*)$ to generate the corresponding results. Therefore, Γ_t^* and Γ_t are computationally indistinguishable.

In conclusion, V_t^* is computationally indistinguishable from V_t. Therefore, for all PPT adversaries $\mathcal{A}$, the outputs of $Real_{\mathcal{A}}(k)$ and of $Ideal_{\mathcal{A},S}(k)$ are negligibly close:

$$|Pr[Real_{\mathcal{A}}(k) = 1] - Pr[Ideal_{\mathcal{A},S}(k) = 1]| \leq neg(k). \quad \square$$

4.2. Search Unforgeability

Theorem 3. *Our scheme as described above achieves search unforgeability.*

Proof. For the malicious user $\mathcal{A}_U$: Consider the search token $T_{ui}(w) = (\tau_1, \tau_2)$, where $\tau_1 = g^{-r} g^{\varphi_w}$, $\tau_2 = h^r g^{-a_{ui} r} g^{a_{ui} \varphi_w} = h^r g^{a_{ui}(\varphi_w - r)}$, $\varphi_w = P_{s_1}(w)$, $r \xleftarrow{R} \mathbb{Z}_q$ and all authorized users know $(g, h = g^a, s_1, P)$. For a user ui, if $\mathcal{A}_U$ wants to generate $T_{ui}(w) \in Q_{ui}$ without the search key a_{ui}, then $\mathcal{A}_U$ has to compute the discrete logarithm for $\tau_2 = h^r g^{-a_{ui} r} g^{a_{ui} \varphi_w} = h^r g^{a_{ui}(\varphi_w - r)}$. We can consider the equation: $y = g^x \ mod \ p$. Given (y, g, p), it is very hard to obtain x in polynomial time. That means $\mathcal{A}_U$ cannot generate $T_{ui}(w) \in Q_{ui}$ without the search key a_{ui}.

For the cloud server $\mathcal{A}_S$: $\mathcal{A}_S$ can make a search query $q = w$ with $\tau_w = \tau_1^{a_{si}} \cdot \tau_2 = h^{\varphi_w}$, and $\mathcal{A}_S$ knows $(a_{si}, g, h = g^a, P)$. However, without s_1, $\mathcal{A}_S$ would not compute $\varphi_w = P_{s_1}(w)$. Therefore, if $\mathcal{A}_S$ wants to make a search query $q = w$, $\mathcal{A}_S$ has to generate a search token $T_{ui}(w) = (\tau_1, \tau_2)$, where $\tau_1 = g^{-r} g^{\varphi_w}$, $\tau_2 = h^r g^{-a_{ui} r} g^{a_{ui} \varphi_w} = g^{a_{si} r} g^{a_{ui} \varphi_w}$. That means $\mathcal{A}_S$ needs to compute the discrete logarithm for τ_1 and τ_2. Now, there is no proper algorithm to find the discrete logarithm. Hence, $\mathcal{A}_S$ cannot generate $T_{ui}(w) \in Q_{ui}$ without the search key a_{ui} and s_1. $\square$

4.3. Revocability

Theorem 4. *Our scheme as described above achieves revocability.*

Proof. In the algorithm $Enc(MK, F, \delta)$, the index δ is encrypted: $I[i, j] \leftarrow \varepsilon.Enc(h^{\varphi_w}, \delta[i, j])$, where $i \leftarrow \alpha_w(H(h^{\varphi_w}))$. An authorized user ui does not have the hash table α_w, so for keyword w, he/she cannot find the corresponding position in I without the assistance of the cloud server. Hence, if the cloud server deletes (ui, a_{si}, d_{si}) from UL, the revoked user ui is incapable of distinguishing keywords w_1 and w_2 in I. $\square$

5. Performance Evaluation

Let M be the length of modulus q, L the length of the outputs for the hash function, E a symmetric encryption operation, e a modular exponentiation, b a bilinear operation, v an inversion operation, n the maximum number of files, m the maximum number of keywords and p the number of parallel processors, respectively. We compare our scheme with several multi-user searchable encryption schemes in Table 4.

Table 4. Performance comparison of several multi-user searchable encryption schemes.

Schemes	Size			Time	
	Public Parameters	**Secret Key**	**Search Token**	**Encrypt**	**Search**
Bao et al. [25]	$2M$	$2M$	M	$E+2e+b$	$b+nE$
Dong et al. [26]	$5M$	$2M$	M	$E+4e$	$e+nb$
Dong et al. [27]	$4M+2L$	$2M$	$2M$	$2E+4e$	$e+nv$
Ours	$4M+3L$	$5M$	$2M$	$2E+e$	$e+nE$

Compared to the schemes in [25,26] that require bilinear operations, which have high overheads, our scheme has better search efficiency. For the encryption process, our scheme takes less time than [27]. From the above table, the search time of all schemes is asymptotically $O(n)$. However, our scheme is parallelizable because it involves the operations for independent vector positions. Thus, the search time for a keyword is $O(n/p)$ parallel time, and the update time for a file is $O(m/p)$ parallel time.

6. Conclusions

Many previous SSE schemes are limited to single-user settings, which cannot meet the needs of data sharing. For this reason, we presented a multi-user searchable symmetric encryption scheme that uses key distribution and re-encryption to achieve multi-user access. In our scheme, each user performs search operations via his/her unique search key, and others are prevented from searching on behalf of a user. We use a simple and practical index structure, namely the combination of a bit matrix and two static hash tables. With the index structure, our scheme allows efficient searches and updates. Specifically, the search time for a keyword is $O(n/p)$ parallel time, and the update time for a file is $O(m/p)$ parallel time. Our scheme obtains revocation operations. For security, we prove that our scheme is IND-CKA2-secure.

Our scheme only supports single-keyword search. In our future work, we will consider how to make our scheme support multi-keyword search, which can achieve expressive search operations in multi-user settings. In addition, we will consider the verifiability of search results.

Author Contributions: C.G. and X.F. conceived of the scheme. C.G. and G.W. mainly wrote the paper. Y.M., G.W., F.L. and T.W. revised the paper.

Funding: This work was supported by Guangxi Key Laboratory of Cryptography and Information Security (Grant No. GCIS201718); the Department of Education of Zhejiang Province of China (Grant No. Y201636547); the Fund of the Lab of Security Insurance of Cyberspace, Sichuan Province (Grant No. szjj2017-055); the Key Research Project of Zhejiang Province (Grant No. 2017C01062); the cybersecurity discipline of Hangzhou Dianzi University (Grant No. GK168800225075).

Acknowledgments: The authors thank the anonymous reviewers for their helpful comments and suggestions.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Song, D.X.; Wagner, D.; Perrig, A. Practical Techniques for Searches on Encrypted Data. In Proceedings of the 2000 IEEE Symposium on Security and Privacy, Berkeley, CA, USA, 14–17 May 2000; pp. 44–55.
2. van Liesdonk, P.; Sedghi, S.; Doumen, J.; Hartel, P.; Jonker, W. Computationally Efficient Searchable Symmetric Encryption. In Proceedings of the 7th VLDB Workshop on Secure Data Management, Singapore, 17 September 2010; pp. 87–100.
3. Kamara, S.; Papamanthou, C.; Roeder, T. Dynamic Searchable Symmetric Encryption. In Proceedings of the ACM Conference on Computer and Communications Security, Raleigh, NC, USA, 16–18 October 2012; pp. 965–976.
4. Kamara, S.; Papamanthou, C. Parallel and Dynamic Searchable Symmetric Encryption. In Proceedings of the 17th International Conference on Financial Cryptography and Data Security, Okinawa, Japan, 1–5 April 2013; pp. 258–274.

5. Kurosawa, K.; Ohtaki, Y. How to Update Documents Verifiably in Searchable Symmetric Encryption. In Proceedings of the 12th International Conference on Cryptology and Network Security, Paraty, Brazil, 20–22 November 2013; pp. 309–328.
6. Stefanov, E.; Papamanthou, C.; Shi, E. Practical Dynamic Searchable Encryption with Small Leakage. In Proceedings of the 21st Annual Network and Distributed System Security Symposium, San Diego, CA, USA, 23–26 February 2014.
7. Naveed, M.; Prabhakaran, M.; Gunter, C.A. Dynamic Searchable Encryption via Blind Storage. In Proceedings of the 2014 IEEE Symposium on Security and Privacy, Berkeley, CA, USA, 18–21 May 2014; pp. 639–654.
8. Kurosawa, K.; Sasaki, K.; Ohta, K.; Yoneyama, K. UC-Secure Dynamic Searchable Symmetric Encryption Scheme. In Proceedings of the 11th International Workshop on Security, Tokyo, Japan, 12–14 September 2016; pp. 73–90.
9. Xu, P.; Liang, S.; Wang, W.; Susilo, W.; Wu, Q.; Jin, H. Dynamic Searchable Symmetric Encryption with Physical Deletion and Small Leakage. In Proceedings of the 22nd Australasian Conference on Information Security and Privacy, Auckland, New Zealand, 3–5 July 2017; pp. 207–226.
10. Goh, E.J. Secure Indexes. *IACR Cryptol. ePrint Arch.* **2003**, *2003*, 216.
11. Chang, Y.C.; Mitzenmacher, M. Privacy Preserving Keyword Searches on Remote Encrypted Data. In Proceedings of the 3rd International Conference on Applied Cryptography and Network Security, New York, NY, USA, 7–10 June 2005; pp. 442–455.
12. Curtmola, R.; Garay, J.; Kamara, S.; Ostrovsky, R. Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions. In Proceedings of the 13th ACM Conference on Computer and Communications Security, Alexandria, VA, USA, 30 October–3 November 2006; pp. 79–88.
13. Chase, M.; Kamara, S. Structured Encryption and Controlled Disclosure. In Proceedings of the 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, 5–9 December 2010; pp. 577–594.
14. Javanmardi, S.; Shojafar, M.; Shariatmadari, S.; Ahrabi, S.S. FR TRUST: A Fuzzy Reputation Based Model for Trust Management in Semantic P2P Grids. *IJGUC* **2015**, *6*, 57–66. [CrossRef]
15. Pooranian, Z.; Chen, K.C.; Yu, C.M.; Conti, M. RARE: Defeating Side Channels Based on Data-Deduplication in Cloud Storage. In Proceedings of the 37th IEEE INFOCOM, Honolulu, HI, USA, 15–19 April 2018; pp. 444–449.
16. Fu, X.; Nie, X.; Wu, T.; Li, F. Large Universe Attribute Based Access Control with Efficient Decryption in Cloud Storage System. *J. Syst. Softw.* **2018**, *135*, 157–164. [CrossRef]
17. Zhang, X.; Wang, H.; Xu, C. Identity-Based Key-Exposure Resilient Cloud Storage Public Auditing Scheme from Lattices. *Inf. Sci.* **2019**, *472*, 223–234. [CrossRef]
18. Bloom, B.H. Space/Time Trade-Offs in Hash Coding with Allowable Errors. *Commun. ACM* **1970**, *13*, 422–426. [CrossRef]
19. Canetti, R. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In Proceedings of the 42nd Annual Symposium on Foundations of Computer Science, Las Vegas, NV, USA, 14–17 October 2001; pp. 136–145.
20. Kurosawa, K.; Ohtaki, Y. UC-Secure Searchable Symmetric Encryption. In Proceedings of the 16th International Conference on Financial Cryptography and Data Security, Kralendijk, Bonaire, 27 February–2 March 2012; pp. 285–298.
21. Fiat, A.; Naor, M. Broadcast Encryption. In Proceedings of the 13th Annual International Cryptology Conference, Santa Barbara, CA, USA, 22–26 August 1993; pp. 480–491.
22. Barman, S.; Chattopadhyay, S.; Samanta, D. An Approach to Cryptographic Key Distribution Through Fingerprint Based Key Distribution Center. In Proceedings of the 2014 International Conference on Advances in Computing, Communications and Informatics, Delhi, India, 24–27 September 2014; pp. 1629–1635.
23. Pecori, R. A Comparison Analysis of Trust-Adaptive Approaches to Deliver Signed Public Keys in P2P Systems. In Proceedings of the 7th International Conference on New Technologies, Mobility and Security, Paris, France, 27–29 July 2015; pp. 1–5.
24. Parakh, A.; Verma, P.; Subramaniam, M. Improving Efficiency of Quantum Key Distribution with Probabilistic Measurements. *IJSN* **2016**, *11*, 37–47. [CrossRef]

25. Bao, F.; Deng, R.H.; Ding, X.; Yang, Y. Private Query on Encrypted Data in Multi-User Settings. In Proceedings of the 4th International Conference on Information Security Practice and Experience, Sydney, Australia, 21–23 April 2008; pp. 71–85.
26. Dong, C.; Russello, G.; Dulay, N. Shared and Searchable Encrypted Data for Untrusted Servers. In Proceedings of the 22nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security, London, UK, 13–16 July 2008; pp. 127–143.
27. Dong, C.; Russello, G.; Dulay, N. Shared and Searchable Encrypted Data for Untrusted Servers. *JCS* **2011**, *19*, 367–397. [CrossRef]
28. Blaze, M.; Bleumer, G.; Strauss, M. Divertible Protocols and Atomic Proxy Cryptography. In Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, 31 May–4 June 1998; pp. 127–144.
29. Yang, Y.; Lu, H.; Weng, J. Multi-User Private Keyword Search for Cloud Computing. In Proceedings of the IEEE 3rd International Conference on Cloud Computing Technology and Science, Athens, Greece, 29 November–1 December 2011; pp. 264–271.
30. Zhao, F.; Nishide, T.; Sakurai, K. Multi-User Keyword Search Scheme for Secure Data Sharing with Fine-Grained Access Control. In Proceedings of the 14th International Conference on Information Security and Cryptology, Seoul, Korea, 30 November–2 December 2011; pp. 406–418.
31. Rane, D.D.; Ghorpade, V.R. Multi-User Multi-Keyword Privacy Preserving Ranked Based Search Over Encrypted Cloud Data. In Proceedings of the International Conference on Pervasive Computing, Pune, India, 8–10 January 2015; pp. 1–4.
32. Nair, M.S.; Rajasree, M.S. Fine-Grained Search and Access Control in Multi-User Searchable Encryption without Shared Keys. *J. Inf. Secur. Appl.* **2018**, *41*, 124–133. [CrossRef]