electronics

MDPI

Article

Verifiable and Searchable Symmetric Encryption Scheme Based on the Public Key Cryptosystem

Gangqiang Duan and Shuai Li *

School of Information Engineering, Ningxia University, Yinchuan 750021, China; 12022131924@stu.nxu.edu.cn
* Correspondence: lis198707@gmail.com

Abstract: With the rapid development of Internet of Things technology and cloud computing technology, all industries need to outsource massive data to third-party clouds for storage in order to reduce storage and computing costs. Verifiable and dynamic searchable symmetric encryption is a very important cloud security technology, which supports the dynamic update of private data and allows users to perform search operations on the cloud server and verify the legitimacy of the returned results. Therefore, how to realize the dynamic search of encrypted cloud data and the effective verification of the results returned by the cloud server is a key problem to be solved. To solve this problem, we propose a verifiable dynamic encryption scheme (v-PADSSE) based on the public key cryptosystem. In order to achieve efficient and correct data updating, the scheme designs verification information (VI) for each keyword and constructs a verification list (VL) to store it. When dynamic update operations are performed on the cloud data, it is easy to quickly update the security index through obtaining the latest verification information in the VL. The safety and performance evaluation of the v-PADSSE scheme proved that the scheme is safe and effective.

Keywords: Internet of Things; cloud computing; verifiable; searchable; symmetric encryption; public key cryptosystem

Citation: Duan, G.; Li, S. Verifiable and Searchable Symmetric Encryption Scheme Based on the Public Key Cryptosystem. *Electronics* **2023**, *12*, 3965. https://doi.org/10.3390/electronics12183965

Academic Editors: Yu-an Tan, Qikun Zhang, Yuanzhang Li and Xiao Yu

Received: 30 July 2023
Revised: 4 September 2023
Accepted: 7 September 2023
Published: 20 September 2023

1. Introduction

With the development of IoT technology, in order to achieve industrial informatization, more and more IoT devices are being connected, and the amount of data is becoming larger and larger. In order to save on storage and computing costs, enterprises choose to outsource massive amounts of data to cloud servers. However, while enjoying the convenience brought by the cloud, the security of the data has become crucial. To protect data privacy, sensitive private data need to be encrypted before being outsourced to the IoT cloud [1]. Song et al. [2] proposed searchable symmetric encryption (SSE). SSE is an encryption scheme that allows users to store encrypted data on a third-party cloud server and search the encrypted cloud data through the trapdoor generated by the keywords. However, most SSE schemes only consider keyword search operations on statically encrypted cloud data, which is inconsistent with the real-time and dynamic update requirements for enterprise data. Moreover, some studies have shown that conventional SSE schemes are vulnerable to leakage abuse attacks [3], file injection attacks [4], and technical attacks [5]. In order to realize the dynamic update (add, delete, or modify) operations of encrypted data stored on cloud servers, some SSE schemes supporting the dynamic update operations of private data have been proposed [6–9]. Kamara et al. [6] proposed an SSE scheme that supports dynamic data updating. The scheme realizes a sublinear search via extending an inverted index and uses a search array and delete array combined with other storage space to realize the dynamic update of the data. Subsequently, they proposed another method based on the keyword red–black tree index structure [7] to support parallel keyword search and parallel data insertion and deletion. Guo et al. [8] proposed a dynamic SSE scheme based on an inverted index. The scheme records the keyword position by the inverted index and

realizes the data dynamics through updating the index. Xia et al. [9] proposed a dynamic keyword search scheme for encrypted cloud data based on the tree index structure that supports multi-keyword sorting.

The above DSSE schemes do not consider the correctness and integrity verification of the returned matching result of the cloud server. In practice, the cloud server may return un-updated or incorrect matching results to the user in order to save computing resources. Therefore, users need to verify the results returned by the cloud server to ensure the correctness and integrity of the returned results. Some schemes [10–12] use the timestamp function of the RSA accumulator to verify the search results, which generates accumulator bits for all files and indexes, which can be saved by the data owner. If the cloud server returns an un-updated result, the user can check it with the latest accumulator. The RSA accumulator [13] can synthesize a large number of data into a fixed-size value to achieve member authentication, which can effectively reduce communication overhead. The RSA accumulator is applied to the compressed prefix tree structure to realize both efficient retrieval and result verification. However, the RSA accumulator is based on asymmetric cryptosystems, and the computational costs and verification costs are high. Some research teams have proposed verifiable schemes based on message authentication code (MAC) [14–16], but in DSSE application scenarios, MAC cannot verify whether the results returned by the cloud server are the latest, that is, it cannot resist replay attacks [17]. Ge et al. [18] proposed a verifiable DSSE scheme based on cumulative authentication tags (AATs), which generates authentication tags for keywords and verifies the returned results through recording the number of global updates and the number of updates of a single file containing those keywords. Each update operation consumes only one label, which is highly efficient. However, the pseudo-random permutation and pseudo-random function are used in this scheme to replace and encrypt the keywords and global update times, which leads to key management problems. Relational authentication tags (RALs) [19] are used to verify the relationship of query keywords in documents, and audit certificates can be generated without exposing sensitive information. However, the program requires third-party auditors to be involved in the search process. Therefore, how to effectively verify the correctness and integrity of the returned results is an urgent problem to be solved.

According to the research on the above schemes, the verification of search results in most schemes is not comprehensive and also involves key management problems. Therefore, how to effectively verify the correctness and integrity of search results as well as the security management of encryption keys are the problems we should focus on and solve.

In this paper, we explore how to use the public key cryptosystem in the DSSE scheme to verify the correctness and integrity of the result returned by the cloud server and to manage the encryption key effectively and securely.

The contributions of this paper can be summarized as follows:

(1) In order to efficiently realize the index update and index lookup, we constructed a bitmap index to store the relationship between the keywords and encrypted files. A verification list (VL) was used to store the latest verification information of files containing keywords so that we can quickly obtain the latest verification information from the VL to perform the secure index update.

(2) In order to support the effective verification of dynamic data, we designed public-key-based cumulative verification information (VI), which is stored in the bitmap index. When the encrypted cloud data are dynamically updated, the verification information can be easily updated. In addition, the verification information contains the corresponding keywords' information, which makes the verification information of various keywords different. Moreover, replay attacks can be resisted by the VI, that is, through verifying whether the returned result is up to date.

(3) In order to achieve forward security, the scheme places the node information in the bitmap index to avoid statistical attacks. When we need to search or update private data,

the cloud server will return or change the whole column's data so that malicious cloud software cannot obtain the relationship of the keywords and index.

(4) Based on the above description, we design a verifiable DSSE scheme based on the public key cryptosystem. The security, verification efficiency, and updating efficiency of the scheme are analyzed and explored. The results show that the scheme is safe and effective.

Organization: The rest of the paper is organized as follows. We summarize the related work in Section 2. In Section 3, the formulas and algorithms involved in the scheme are defined, including model construction, design objectives, etc. In Section 4, we describe the construction of the scheme and the execution of the algorithm. The security analysis of the v-PADSSE scheme is given in Section 5. In Section 6, the implementation efficiency and updating efficiency of the program are analyzed and evaluated.

2. Related Work

With the development and application of the Internet of Things and cloud computing technology, many industries have chosen to outsource data to third-party clouds for storage. While cloud storage brings convenience to enterprises, it also brings new security challenges. Users cannot directly control the data stored in the cloud, so it is impossible to determine whether the data stored are complete and correct. To solve the problem of data verification, the research community has proposed some cloud storage verification schemes [20–22] to audit and verify data in the cloud. In addition, before uploading private data to the cloud for storage, users need to encrypt it to prevent it from being accessed directly by cloud providers. However, in this case, how users perform search operations on encrypted cloud data is also an important problem to be solved. To solve the above problems, the research community proposes searchable symmetric encryption (SSE), which allows users to perform search operations directly on the ciphertext. Compared with the searchable encryption scheme of the public key encryption system [23,24], the efficiency of the SSE scheme has received more attention from the industry.

Dynamic SSE. Searchable encryption can be divided into two categories: symmetric key encryption [25] and public key encryption [26]. Song et al. [2] first proposed a searchable encryption scheme that encrypts each keyword through constructing a special two-layer encryption structure. Some static SSE schemes, such as semantic search schemes [27] and ranked keyword search schemes [28,29], are also proposed. However, in practice, industrial data are dynamically updated in real time, and the static SSE scheme does not support the dynamic update of encrypted cloud data, so it cannot meet the requirements of cloud storage data encryption at this stage. In order to support the dynamic update of encrypted data, Kamara et al. [6] proposed a dynamic SSE scheme through constructing an extended inverted index to achieve sublinear search efficiency and CKA-2 security. Scheme [30] proposed a dynamic SSE scheme which allows data owners to store privacy files in a way that the cloud server does not know the number of files through constructing a blind storage system on the cloud server. Guo et al. [9] proposed a DSSE scheme based on the inverted index, which allows data users to search multiple phrases in a query request, and the scheme supports the ordering of search results. In recent years, a number of cloud-assisted schemes have been proposed for searchable encryption [31]. Scheme [32] utilized the searchable encryption technologies of keyword range search and multi-keyword search. Since the cloud is untrustworthy, scheme [32] used Bloom filters and message verification codes to classify health information, filter out fake data, and check data integrity. In order to verify whether the cloud faithfully performs the search operation, a multi-user verifiable searchable symmetric encryption is proposed in scheme [25]. Authorized users can search the data, verify the authenticity of the search results, and improve the accuracy of the search results. Since the access rights of authorized users are always valid, it is not secure. In order to automatically revoke a user's access, the time key was introduced in [33]. At the beginning of encryption, the key is encapsulated in ciphertext, which means that all users, including the data owner, are bound by the time period. Later, Yang et al. [34] proposed a conjunctional keyword search with the function of specifying testers and

enabling timed proxy re-encryption. It utilizes a time server to generate time tokens for users. In addition, it implements time-controlled access revocation to prevent authorized users from accessing future EHRs. Scheme [35] proposed timed-release computational secret sharing and threshold encryption which used a time-release function instead of a time server to reduce overhead. Scheme [36] proposed 0-encoding and 1-encoding to generate the time key. However, the retrieval efficiency of this work is low. In order to improve search efficiency, scheme [37] with hidden data structures was proposed in the literature. The user expected to find more ciphertexts in one step. However, scheme [37] reduced the number of computation-intensive operations without searching for at least two matching ciphertexts in just one step. This work cannot meet the need for a quick search and prevent authorized users from accessing future data. While all of the above work enables cloud-based search, there is still a challenge: the cloud is not a fully trusted entity and can collude with other entities to gain access to users' private information.

Verifiable SSE. In practice, cloud servers are semi-trusted entities [38] that may return incorrect or un-updated results to the data user in order to save on computing overhead. Miao et al. [39] constructed the verifiable SE framework (VSEF), which can withstand internal KGA and achieve verifiable searchability. Wu et al. [40] proposed a new authentication data structure based on homomorphic encryption and showed how to apply it to verify the correctness and integrity of search results. However, the verification proof in their scheme is generated by the cloud server, which can forge the proof to pass the verification when the cloud server is seen as an adversary. To avoid this, Chai et al. [41] first proposed a verifiable keyword search scheme for encrypting cloud data, using hash functions to generate proof of document identity. Jiang et al. [14] proposed a verifiable multi-keyword ranked search scheme based on encrypted cloud data, which realized an efficient keyword search through constructing the special data structure QSet. Yang et al. [42] designed a forward-privacy VDSSE scheme with Bloom filters and message authentication codes to allow verification and support dynamic updates of outsourced data. Zhang et al. [43] proposed a verifiable data structure based on a multi-set hash function, which guarantees forward security and realizes effective verifiable data updates. Gao et al. [19] used relational authentication tags (RALs) to verify the relationship of the query keywords in the document, which can generate audit certificates without exposing sensitive information. However, the program requires third-party auditors to be involved in the search process. Merkle hash trees [44] are used to validate data elements in large databases. Through adding data elements to the leaf node of the tree, the tree structure is constructed layer by layer from the leaf node to the root node, and finally the unique root node is obtained. A change to any element in the data set will make the root node change. A Merkle Patricia tree is proposed in GSSE [45] to reduce the storage overhead of index structures in schemes based on Merkle hash trees. It reduces storage space through reducing the depth of the tree. However, in the above two scenarios, the proof provided by the cloud server to the DU is larger in scale, which brings more communication overhead. Chen et al. [46] extend the Merkle hash tree [47] to a searchable index tree to achieve efficient result verification, where search time grows sublinearly with the size of the data set, and verification is more efficient than the accumulator structure. In addition, verifiable DSSE has been implemented by schemes [45,48], but they either support a single keyword match search or use two rounds of communication in a single-user setup to achieve result verification. The RSA accumulator [13] can aggregate a large amount of data into a fixed value to achieve member verification, which can effectively reduce communication overhead. The RSA accumulator is applied to the compressed prefix tree structure to realize the combination of efficient retrieval and result verification. Schemes [10,12,13] all use an RSA accumulator to realize result verification for dynamic data. Most of the above VDSSE schemes are based on asymmetric key cryptography, and the results returned by the cloud server are verified using the public key signature.

Forward secure SSE. Forward privacy protection requires that update operations (insert or delete) performed by the data owner cannot be associated with previously performed search operations. Because the secret key is used for the deterministic encryption

of private data in the DSSE scheme, it is easy for untrusted servers to obtain repeated queries and other information, which leads to information leakage (such as the number of keyword queries, etc.). If ORAM is introduced into the scheme, such problems will be avoided, but the communication cost and calculation cost are high, which causes the calculation and execution efficiency of DSSE to be exchanged through allowing some information to be leaked in actual use. However, such leaks are often attacked in different ways [49,50]. Bost et al. [51] proposed to use a one-way trapdoor replacement to eliminate the correlation between the latest trapdoor and the previous trapdoor, that is, the latest trapdoor can search all encrypted documents, but the previous trapdoor cannot match the latest encrypted document. Cao et al. [52] used the KNN method to construct the security index and trapdoor. This method is used to encode indexes and trapdoors so that even if the keyword is the same, the encoding is different. In this way, the cloud server can avoid obtaining the number of keyword queries and the association between keywords and encrypted data based on data user query operations, thereby protecting forward privacy. Li et al. [53] used partitioning and pointer hiding technology to partition the secure index and extracted sub-keywords according to the original keywords as the keywords of partition search, then encrypted and hid the index block, which only needed to save the index table header identification and encryption key locally. Since the search token information is calculated using subkeywords, it is difficult for subsequent query keywords to be directly associated with the newly added encrypted document.

3. Security Model and Related Definitions

3.1. Security Model

In the design scheme, there are four entities that need to be involved, namely: the data owner (DO), the data user (DU), the cloud server (CS), and the key distribution center (KGC). The system security architecture is shown in Figure 1.

- Data owner: This entity encrypts private files and secure indexes with a symmetric key and encrypts verification information with the data user's public key. After the encryption is finished, the ciphertext and index are uploaded to the cloud server. When the data owner wants to update the privacy data, the update token needs to be generated locally and then sent to the cloud server for data updating. Upon receiving the VI request from the data user, the data owner returns the number of files N and the total number of file updates V that contain the keyword w.

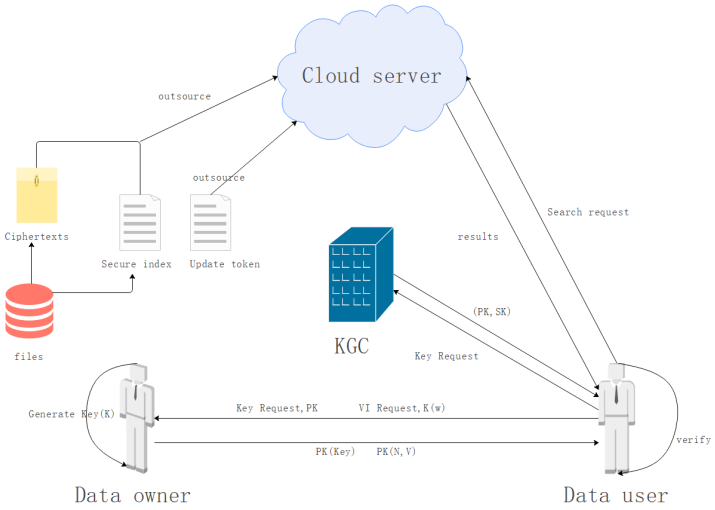

Figure 1. System security architecture.

- Data user: The entity shares the encrypted private key with the data owner. When he wants to perform a search operation containing keywords, he needs to generate a trapdoor locally, then send the trapdoor to the cloud server for searching, and apply

to the data owner for the latest verification information of the keyword. The w of VI indicates the keyword for which the user wants to perform the search operation. Upon receiving the returned results from the cloud server, the correctness and integrity of the results are verified according to the verification information.

- Cloud server: This entity stores the ciphertext and security index information uploaded by the data owner. When it receives a search request, it performs the search operation on the security index and returns the corresponding matching results and verification information. When it receives an update request, it performs an update operation on the security index and the corresponding ciphertext.
- Key Distribution Center (KGC): This entity is primarily used to generate keys. Upon receiving a key request from the data user, the entity returns a key pair (PK, SK) to the corresponding user.

In the system model, both the data owner and data user must be trusted entities. The data owner honestly encrypts the private files and builds the secure index. The data user honestly generates trapdoors for the desired keywords and sends them to the cloud server. The cloud server is an untrusted entity that allows search operations to record the correspondence between search keywords and encrypted files, and it may return incorrect or un-updated results to the data user in order to save computing overhead. The key distribution center is a trusted entity that honestly generates the key pair requested by the data user and sends it to the requesting user.

3.2. Design Goals

Based on the above model architecture, to achieve a verifiable DSSE scheme, we design a scheme which needs to meet the following objectives:

- Support keyword search over the encrypted cloud data: The scheme needs to match all ciphertexts containing the corresponding keywords according to the search token and demonstrate high query efficiency.
- Support efficient dynamic data updates: The scheme needs to support the dynamic update of encrypted data and secure indexes, such as dynamic addition, dynamic deletion, and dynamic modification.
- Support search result verification: The scheme needs to support data users in efficiently verifying the correctness and integrity of the matching results returned by the cloud server, and the verification does not involve any complex operations.
- Privacy protection.: Due to the scheme being based on the public key cryptosystem, the public key cannot be used to encrypt private information directly. The private information is encrypted using the symmetric key, and the asymmetric key is used to encrypt the verification information. In addition, the scheme should hide the encrypted file containing information about keyword quantity and keyword search frequency.
- Replay attack resistance: To save on computing or storage overhead, the cloud server directly returns the un-updated results to the data user. The scheme should enable data users to verify the returned results to determine whether the returned results are up to date.

3.3. Algorithm Definition

The related algorithms in the v-PADSSE scheme we designed are KeyGen, PSKeyGen, IndexBuild, Building VL, GenToken, Search, Verify, Decrypt, UpdateToken, and Update. These algorithms are defined as follows:

- $K \leftarrow KeyGen(1^{\lambda_1})$. The data owner outputs the key (K) through using the secure random parameter λ_1 as the input.
- $(PK, SK) \leftarrow PSKeyGen(\lambda_2)$. The KGC outputs the key pair (PK,SK), using the secure random parameter λ_2 as input.

- $(I, C) \leftarrow IndexBuild(K, PK, F, W, N, V, v_i, flag)$. When building a secure index, encrypt the file F and the keywords W with K. Use the public key (PK) to encrypt the number of files (N) and the total number of updates (V) containing the keywords, the number of updates (v_i) per file, and the flag bit of whether the file contains the keyword ($flag$), where the random number generator function $rand()$ is used to generate random numbers ($randA$ for odd numbers, $randB$ for even numbers). If the file contains the keyword, the $flag$ is odd. If not, the $flag$ is an even number. Calculate the file F using the SHA-3 hash algorithm (later replaced with the symbol "H"), and finally, output security index I and ciphertext C.
- $T_w \leftarrow GenToken(K, w)$. The data user executes the trapdoor generation algorithm. Take key K and keyword w as inputs, and output trapdoor T_w.
- $(VI, C(w)) \leftarrow Search(T_w, I, C)$. The output is the verification information VI, matching ciphertext set $C(w)$. The VI contains the update times of the file (v_i) matching the trapdoor, the flag of whether the ciphertext contains the keyword, and the result $H(F)$ after hashing the plaintext file using T_w, I and C as input.
- $(Y, N) \leftarrow Verify(VI, SK, T_w, C(w))$. Output the result of verifying (Y or N) through using VI, $PK(N, V)$, SK, T_w, and $C(w)$ as inputs.
- $F(w) \leftarrow Decrypt(K, C(w))$. Take K and $C(w)$ as inputs, and output plaintext file F.
- $\tau \leftarrow UpdateToken(K, PK, F, \{w, flag\}, v_i)$. Update token information includes the update operation type, the newly updated file F, the document identifier F_{id}, the number of updates per file v_i, $H(F)$ (the hash result of F), the set of keywords w contained in the file, and $flag$. When the add operation is performed, the VL is matched according to the keyword set contained in file F, $V = V + 1$ and $N = N + 1$ are calculated in the matched node, and the document ID (F_{id}) and update number $v_i = 1$ of file F are added to the node. At this point, the update token contains the addition of F_{id}, v_i, the keyword set w_i contained in the file F, and $H(F)$. If F contains the keyword, $flag = randA$; otherwise, $randB$. When the delete operation is performed, the verification list VL is matched according to the keyword set contained in F; then, $N = N - 1$ and $V = V - v_i$ are calculated in the matched nodes, and F_id and its update times v_i in the nodes are deleted, so that $v_i = 0$ and $flag = randB$. When the modify operation is performed, the verification list VL is matched according to the keyword set contained in F. In the matching node, perform $V = V + 1$ and $v_i = v_i + 1$; N and $flag$ remain unchanged, and the new file is hashed to $H(F)$. After the above verification information is modified, the public key is used to encrypt this information except for $H(F)$.
- $(I', C') \leftarrow Update(\tau)$. The cloud server executes the update algorithm. Match according to F_id contained in τ, and replace the nodes' value in the column. According to the update token τ, The cloud server generates a new index item I' and ciphertext C'.

3.4. Security Definition

- Updated Reliability: A verifiable DSSE scheme first needs to ensure that the cloud server performs reliable update operations, that is, replay attack resistance. Since the cloud server is not trusted after it receives an update request from the data owner, it may not perform the corresponding update operation according to the update token content, that is, it will not update the security index and ciphertext collection. After receiving a search request from the data user, the un-updated data are returned to the data user, and the data user should verify that the returned results are up to date. If the opponent obtains the latest authentication information VI' and valid ciphertext $C(w)'$, and the forged information can pass the verification algorithm, the opponent wins.
- Verifiability: If the probability of the opponent successfully forging search results is negligible, the v-PADSSE scheme is considered verifiable. Due to the unreliability of the cloud server, it may return incorrect or incomplete results to the data user. Data users should be able to detect the improper behavior of the cloud server using verifi-

cation algorithms to ensure the correctness and integrity of the returned results. If the opponent obtained the latest verification information VI' and the valid ciphertext set $C(w)'$ and can forge the authentication information to pass the verification algorithm, the opponent wins.

4. v-PADSSE Scheme Construction and Algorithm Description

We have summarized some common symbols used in the design of the v-PADSSE scheme, as shown in Table 1.

Table 1. Common symbols and descriptions

Symbol	Description
N	the number of files containing keywords
n	the number of keywords
F	plaintext file set
W	keywords set
w_i	the i-th keyword in the keywords set
V	the total number of updates to the file containing keyword w
v_i	the number of updates to per file containing the keyword w_i
I	secure index
C	ciphertext set
T_w	the search trapdoor of keyword w
C(w)	ciphertext set containing keyword w
VI	verification information set
F(w)	plaintext file set containing keyword w
τ	update token
VL	verify list
I′	updated security index
C′	updated ciphertext set
docId	document identification
H	SHA-3 hash algorithm
flag	indicates whether the file contains the keyword
rand()	random number generation function
randA	odd numbers generated by rand()
randB	even numbers generated by rand()

4.1. Overview of the v-PADSSE Scheme

In order to solve the problem of correctness and integrity verification of the results returned by the cloud server, this paper designs a DSSE scheme based on public key verification (v-PADSSE). In this scheme, verification information is added to the security index and encrypted using the public key of the data user, so that the user can verify the returned results. Below, the construction of the v-PADSSE scheme is described in detail.

When constructing VI, the v-PADSSE scheme needs to include v_i, $flag$, and $H(F)$. The VI needs to be encrypted with the user's public key. Assume that the encryption function is $PK(VI)$. To prevent the cloud server from collecting statistics on the correlation between keywords and updated files, all index nodes in the column of the document representation of the updated file must be updated so that the update operation can hide the correlation between the ciphertext and the keywords. Therefore, the verification information $VI = PK(v_i) + PK(flag) + H(F)$. In addition, the data owner creates a verification list (VL) locally, which stores the latest verification information of each keyword, including the number of files containing keyword N, the total number of updates of files containing keyword V, the document identification of files containing keyword id, and its single set of file update times v_i so that the latest update token can be generated directly when the update operation is performed. For different update operations (such as modify, add, and delete), VI needs to be performed in different operations. VI is calculated as follows:

- Add new file F'.

When the data owner obtains the latest VI and performs the add operation, the number of updates of a single file is initialized to $v_i = 1$. If the newly added file F' contains the keyword w in the VL, the corresponding node in the VL needs to execute $N = N + 1$, $V = V + 1$, then add the document id of the new file F and the number of updates v_i to the node. If it does not, a new node needs to be added to the VL, where $N = 1$, $V = 1$, $v_i = 1$, and $VI = PK(v_i) + PK(flag = randA) + H(F')$.

- Modify file F to F'.

When file F needs to be updated to a new file F' (both F and F' contain the keyword w), the data owner updates VL with the latest verification information for the keyword w in the corresponding node and executes $V = V + 1$ and $v_i = v_i + 1$. $VI' = VI - pk(v_i) + PK(v_i + 1) - H(F) + H(F')$.

- Delete file F.

File F contains keyword w, and the data owner updates $N = N - 1$, $V = V - v_i$, and $v_i = 0$ in the VL in the node where keyword w resides. In this case, the latest verification information $VI' = VI - PK(N) + PK(N - 1) - PK(v_i) + PK(v_i = 0) - PK(flag = randA) + PK(flag = randB)$. Additionally, the document id and v_i of file F are removed from the VL.

After the data owner performs different update operations, the corresponding update token τ is generated and sent to the cloud server, which updates the security index and ciphertext according to the update token information.

4.2. Secure Index Structure

In v-PADSSE, the data owner constructs the security index through using the bitmap index and constructs VL locally. The data owner generates the symmetric encryption key through executing the algorithm *KeyGen*, and the data user's key pair (PK, SK) is generated via KGC executing the *PSKeyGen* algorithm.

First, the data user publicly releases the public key, and the data owner, after obtaining the user's public key, uses the symmetric private key K to encrypt the privacy files and keywords and uses PK to encrypt the verification information corresponding to the keywords. When the security index is firstly constructed, the data owner needs to initialize VI through initializing N to the number of files containing the keyword w, $V = \sum_{i=1}^{n} v_i$, $v_i = 1$ ($1 \leq i \leq N$). When the security index is constructed, the column header contains the keyword w_i, and the row header contains the document ID $docId$. Middle node information includes v_i (the number of updates to the file containing the keyword w_i), $flag$ (indicating whether the document contains the keyword), and $H(F)$ (the result of the hash operation of the plaintext file). The security index structure is shown in Figure 2.

keyWord / docId	id1	id2	id3	...
$K(w_1)$	$PK(v_1),PK(flag_{11}),H(F_1)$	$PK(v_2),PK(flag_{12}),H(F_2)$	$PK(v_3),PK(flag_{13}),H(F_3)$	...
$K(w_2)$	$PK(v_1),PK(flag_{21}),H(F_1)$	$PK(v_2),PK(flag_{22}),H(F_2)$	$PK(v_3),PK(flag_{23}),H(F_3)$	...
$K(w_3)$	$PK(v_1),PK(flag_{31}),H(F_1)$	$PK(v_2),PK(flag_{32}),H(F_2)$	$PK(v_3),PK(flag_{33}),H(F_3)$	...
$K(w_4)$	$PK(v_1),PK(flag_{41}),H(F_1)$	$PK(v_2),PK(flag_{42}),H(F_2)$	$PK(v_3),PK(flag_{43}),H(F_3)$	...
...	...	...	...	...

Figure 2. Security index structure.

The number of rows in the secure index is determined by the number of keywords, and each row is associated with a keyword. The number of column nodes is determined by the number of privacy files. When the data owner needs to perform an update operation, PK(v_i), $PK(flag)$, and $H(F)$ in all nodes in the whole column are modified according to the updated document identifier.

The data owner needs to obtain the latest verification information of the corresponding keyword when generating the update token. Therefore, VL is designed in the scheme and is owned by the data owner. The latest VI of keywords contained in each privacy file must be recorded in VL. When updating, the data owner modifies the VI in VL to ensure that the VL and VI in the security index are updated simultaneously. The structure of the VL is shown in Figure 3, where N indicates the number of files containing the keyword, V indicates the total number of updates to files containing the keyword, and $\{id, v_i\}$ indicates the set composed of the document identification of the file containing the keyword and the number of updates to the file.

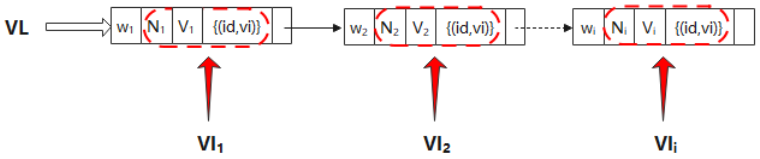

Figure 3. Structure of VL.

4.3. Algorithm Description

In this section, we give the execution steps of the core algorithm of the v-PADSSE scheme and explain the related functions in detail.

In v-PADSSE, the core algorithms involved are *IndexBuild* (Algorithm 1), *Building VL* (Algorithm 2), *GenToken*, *Search* (Algorithm 3), *Verify* (Algorithm 4), *UpdateToken* (Algorithm 5), and *Update* (Algorithm 6). The *IndexBuild* Algorithm 1 is used by the data owners to construct the secure index with bitmaps. Among them, the header node stores keyword and document identification, and the middle node stores keyword-related verification information. After constructing the secure index, data users upload the secure index and encrypted files to the cloud server. Data users use the *GenToken* algorithm to generate a trapdoor and send it to the cloud server. The cloud server executes the *Search* Algorithm 3, performs the matching query on the security index according to the trapdoor, and returns the matching results and verification information to the data user. After receiving the results, the user executes the *Verify* Algorithm 4 to verify the correctness and integrity of the results. If not verified, refuse. To update the privacy file, the data owner obtains the verification information related to the keyword contained in the file from VL, runs the *UpdateToken* Algorithm 5 to generate the corresponding update token, and sends it to the cloud server. The cloud server executes the *Update* Algorithm 6 based on the updated token information to update the security index and related ciphertext. Below, we will give a detailed explanation of the above core algorithm execution process.

- Initialization parameters:
  - (1) Obtain the keyword set $\{w\}$ contained in the plaintext file and save it in the keyword set $W = \{w_1, w_2, \ldots, w_n\}$, where n is the number of keywords.
  - (2) Obtain the number of files containing keyword w, $N = F(w).num$, the total number of updates to files containing keyword w, $V = N$, the set $\{id, v_i = 1\}$ consisting of the document ID and the number of updates per file, and whether the file contains the $flag$ of the keyword. $(PK(v[N]), PK(flag)) \leftarrow EncryptPK(v[N], flag)$ through using the public key.
  - (3) Encrypt keyword w and plaintext file F using the symmetric key algorithm, compute $(Kw, C) \leftarrow EncryptK(w, F)$, and hash file F to obtain $H(F)$.
- Building secure index:

  For each privacy file, document identification $docId$ and keyword set $\{w_i\} \in W(1 \leq i \leq n)$ are used to construct the bitmap index. The functions and parameters required to construct the security index are described as follows:
  - (1) Create the header node $BuildHeadNode(K_w, docId)$, where K_w is the keyword set after symmetric key K encryption, and $docId$ is the private file identification set.

(2) Create an intermediate node $BuildMiddleNode(K_w, docId, PK(v_i), PK(flag), H(F))$; K_w and $docId$ indicate where verification information is stored in the bitmap index, $PK(v_i)$ is the number of file updates containing the keyword w, and $PK(flag)$ indicates whether the privacy file corresponding to $docId$ contains keyword w. If yes, $flag = randA$; if no, $flag = randB$. $H(F)$ is the result of hashing the privacy file F.

The algorithm process is as follows:

Algorithm 1 $(I, C) \leftarrow IndexBuild(K, PK, F, W, N, V, v_i, flag)$

DO:
N=F.num();V=F.num();v[N]=$\{1, 1, \ldots, 1\}$;
$(K_w, C) \leftarrow EncryptK(w, F)$;
$PK(v_i), PK(flag) \leftarrow EncryptPK(v_i, flag)$;
$H(F) \leftarrow Hash(F)$;
// Assuming a total of n keywords and n privacy files, the following procedure is executed $n * n$ times.
for $i = 0; i < n;$ i++ **do**
  for $j = 0; j < n;$ j++ **do**
    $BuildHeadNode(K_w, docId)$;
    $BuildMiddleNode(K_w, docId, PK(v_i), PK(flag), H(F))$;
  end for
end for
//The above process of creating head nodes and middle nodes together forms the bitmap index, and assigns the $flag$ bit according to whether the keyword is contained in the privacy file, and generates the security index I.
//The data owner sends the generated security index I and ciphertext C to the cloud server.
Send to CS(I,C);

- Building verification list: VL

The verification list is constructed and stored locally by the data owner. The VL is a single linked list, and the linked list node needs to contain the verification information of each keyword. The creation process is as follows.
The VL header node does not store any information, only the address of the first keyword.
$BuildHeadNode(*firstKeyWord)$:$*firstKeyWord$ indicates the address of the first keyword.
$BuildListNode(w_i, N_i, V_i, \{id, v_i\}, *nextKeyWord)$. The middle node of the linked list stores the keyword w_i, the number of files N_i containing w_i, and the total number of updates to files V_i containing w_i. A set of the document identification of the privacy file containing w_i and the number of updates v_i to the file, and a pointer to the next keyword address $*nextKeyWord$ are also included.

Algorithm 2 Building VL

//Because there are n keywords, the creation of the intermediate index node needs to be executed n times.
for $i = 0; i < n;$ i++ **do**
  $BuildListNode(w_i, N_i, V_i, \{id, v_i\}, *nextKeyWord)$;
end for

The Search algorithm process is as follows:

Algorithm 3 $(verifyInfo, C(w)) \leftarrow Search(T_w, I, C)$

DO:
//Data users execute trapdoor generation algorithm *GenToken*, use symmetric key K to encrypt keyword information, generate trapdoor T_w, and send it to the cloud server. It is also sent to the data owner through the channel to obtain the latest verification information of the keyword.
$T_w \leftarrow GenToken(K, w)$;
Send (T_w) to CS and DO;
CS:
if IndexSearch(T_w) = null **then**
  Then return null;
else
  //Obtain keyword verification information.
  verifyInfo = GetVerifyInfo(PK(v_i),PK(flag),H(F));
  C(w) = search(docId);
end if
//Return the search results and VI to the data user.
Return (verifyInfo,C(w));

The Verity algorithm process is as follows:

Algorithm 4 $(Y, N) \leftarrow Verify(verifyInfo, SK, T_w, C(w))$

DU:
//The user uses the private key SK to decrypt the verification information.
$(v_i, flag) \leftarrow DecSK(PK(v_i), PK(flag))$;
//Decrypt the latest verification information returned by the data owner.
$getNewVerifyInfo(T_w, PK(N_w), PK(V_w))$;
$DecSK(PK(N_w), PK(V_w))$;
//Check whether the keyword is contained in the privacy file according to the $flag$. If yes, proceed with the execution. If not, the privacy file will not be decrypted.
if flag% 2=0 **then**
  Delete;
  //The verification information returned by the data owner compares with that returned by the cloud server. If the verification information is correct, the ciphertext is accepted and decrypted. If not, the ciphertext is rejected.
else
  if $N_w = C(w)$. num and $V_w = \sum v_i$ and H(Decrypt(K,C(w)))=H(F) **then**
    Return Y;
  else
    return N;
  end if
end if

The UpdateToken algorithm process is as follows:

Algorithm 5 $\tau \leftarrow UpdateToken(K, PK, F, \{w, flag\}, v_i)$

DO:
Add:
//The data owner needs to add the privacy file F, he firstly obtains the keyword set $\{w\}$ and document identification $docIdF$ in F and then encrypts F and keyword $\{w\}$ with K, v_i is the number of updates corresponding to the privacy file.
$(\{K_w\}, C) \leftarrow EncryptK(\{w\}, F)$;

Algorithm 5 *Cont.*

//The VL is matched according to the keywords contained in F. If $\{w_i\}$ contained in F already exists, add $(docIdF, 1)$ to the $\{id, v_i\}$ in the matched node, and do $N = N + 1$, $V = V + 1$ in this node. If not, add a new node $(w, N = 1, V = 1, \{docIdF, 1\})$ to VL.
//Assume there are k keywords in file F, it needs to be executed k times.
for $i = 0; i < k;$ i++ **do**
  if search(w_i) is true **then**
    N = N + 1;V = V + 1;
    Add(docIdF,1);
  else
    $BuildListNode(w_i, N_i = 1, V_i = 1, \{docIdF, 1\}, *nextKeyWord)$;
  end if
end for
//The updated verification information is encrypted using public key PK, and generates the added token τ_{add}. If w is contained by the file F, the value of $flag$ is $randA$ in node $(w, docIdF)$, If not, the $flag$'s value is $randB$.
$PK(v_i), PK(flag) \leftarrow EncryptPK(v_i, flag)$;
$H(F) \leftarrow Hash(F)$;
$\tau_{add} = (''add'', \{PK(v_i), PK(flag), H(F), K_w, docIdF\}, C)$;
$Send(\tau) toCS$;
Delete:
//If the data owner needs to delete file F, he obtains the set of keyword $\{wi\}(1 \leq i \leq k)$ in file F and uses K to encrypt the keywords and the deleted file F. The document identification of the file F is $docIdF$.
$(\{K_w\}, C) \leftarrow EncryptK(\{w\}, F)$;
//Search for $\{w_i\}$ in VL, and update N, V and the set $\{id, v_i\}$ in the corresponding node according to the keyword w_i. This procedure takes k times.
for $i = 0; i < k;$ i++ **do**
  if search(w) is true **then**
    N=N-1;V=V-v[docIdF];
    //Removes $\{docIdF, v\}$ from the document identification $\{id, vi\}$ set in matched node;
    Delete $\{docIdF, v\}$;
  else
    Return error;
  end if
end for
//The updated verification information is encrypted using the public key, and the deleted token τ_{del} is generated and sent to the cloud server.
$PK(v_i), PK(flag) \leftarrow EncryptPK(v_i = 0, flag = randB)$;
$H(F) \leftarrow Hash(F)$;
$\tau_{del} = (''delete'', \{PK(v_i), PK(flag), H(F), K_w, docIdF\}, C)$;
Send(τ) to CS;
Modify:
//If the data owner needs to modify the privacy file, he uses K to encrypt the modified file F and keywords contained in F.
$(\{K_w\}, C) \leftarrow EncryptK(\{w\}, F)$;
//Update VI in VL according to the keywords contained in file F. After updating, the verification information is encrypted using the public key, and the modified token τ_{mod} is generated and sent to the cloud server.
//Since the modified file F contains k keywords $\{w_i\}(1 \leq i \leq k)$, the following procedure needs to be performed k times.

Algorithm 5 *Cont.*

for $i = 0; i < k;$ i++ **do**
  if search(w) is true **then**
    V=V+1;v[docIdF]=v[docIdF]+1;
  end if
end for
$PK(v[docIdF]) \leftarrow EncryptPK(v[docIdF])$;
$H(F) \leftarrow Hash(F)$;
//During the modification, the $flag$ remains unchanged.
$\tau_{mod} = (\text{"}modify\text{"}, \{PK(v[docIdF]), PK(flag), H(F), K_w, docIdF\}, C)$;
Send(τ) to CS;

Algorithm 6 $(I', C') \leftarrow Update(\tau)$

CS:
//After receiving the updated token from the data owner, the cloud server performs operations on the security index I and ciphertext C according to the token.
if τ.operate = "add" **then**
  //Add a new column $docIdF$ to the bitmap index, or add a new row if the keyword contained in F does not exist in the bitmap index.
  BuildHeadNode(docIdF);
  //Assume there are n keywords in the bitmap index, that is, n rows, which need to be executed n times.
  for $i = 0; i < n;$ i++ **do**
    $BuildMiddleNode(K_w, docIdF, PK(v_i), PK(flag), H(F))$;
  end for
  if K_{wi} does not exist in the bitmap index **then**
    $BuildHeadNode(K_{wi})$;
    //Assume there are n document identifications in bitmap, the following procedure needs to be performed n times.
    for $i = 0; i < n;$ i++ **do**
      $BuildMiddleNode(K_w, id, PK(v_i), PK(flag), H(F))$;
    end for
  end if
  addFile(C');
else if τ.operate = "delete" **then**
  //Delete All nodes in the bitmap index if the value of column head node is $deleteId$.
  deleteColumn(deleteId);
  deleteFile(C');
else if τ.operate = "modify" **then**
  //Assume there are n keywords in the bitmap index, all nodes need to be changed if the value of column head node is $modifyId$.
  for $i = 0; i < n;$ i++ **do**
    $ChangeNode(K_{wi}, modifyId, PK(v_i), PK(flag), H(F))$;
  end for
  changeFile(C,C');
end if

In conclusion, when the cloud server performs the update operation, the updated column needs to be modified. At this time, the existence of the $flag$ and the correct update of the $flag$ will not affect the verification result, even if the keyword w which is not contained in F changes. Moreover, since the update operation involves the change of an entire column in the secure index, it is also a good way to hide the correlation between the keywords and the updated file.

4.4. Comparison

In this section, we compare our scheme with Σ*οφος* [51], Ge's scheme [18], Gao's scheme [19], and Zhang's scheme [43]. All of those schemes can ensure the verifiability of search results. Assume there are n files and m keywords in total. For simplification, we assume that each search returns n files. We neglect the communication costs and only compare the computation overhead in different phases of these schemes. Table 2 shows the results of the comparison.

Table 2. Performance comparison.

Schemes	FS	Update	Search	Verify
Σ*οφος* [51]	✓	$O(mn)$	$O(m)$	
Zhang's scheme [43]	✓	$O(mn)$	$O(m)$	$O(n)$
Gao's scheme [19]	✓	$O(mn)$	$O(m)$	$O(n)$
Ge's scheme [18]	✓	$O(n)$	$O(m)$	$O(n)$
our scheme	✓	$O(n)$	$O(m)$	$O(n)$

As can be seen from Table 2, the efficiency of our scheme is close to Zhang's scheme [43], Gao's scheme [19], and Ge's scheme [18] in terms of search and verify operations. However, in the update process, our scheme and Ge's scheme [18] are better than others; the time complexity of the two schemes is O(n).

5. Security Analysis

In this section, we will analyze the security of the v-PADSSE scheme in two aspects: update reliability and verifiability.

5.1. Update Reliability Analysis

Due to the cloud server being unreliable in v-PADSSE, it may not update the security index and ciphertext after receiving the update request from the data owner in order to save computing or storage resources. We are going to prove that the Verify algorithm outputs "N" when the cloud server returns un-updated results.

Assume that the result returned by the cloud server is $(VI', C'(w))$, and the correct result and verification information is $(VI, C(w))$. The number of files containing the keyword w is N', and the total number of updates to files containing the keyword w is V'. In addition, the scheme proposes that when the data user performs a query, it will apply to the data owner for the latest verification information $PK(N)$, $PK(V)$ for the keyword. At this time, we will consider the following three scenarios to prove the reliability of the v-PADSSE.

(1) VI = VI', C(w) $\neq$ C'(w)

If the cloud server updates only the security index but not the ciphertext, the returned result is $(VI, C'(w))$.

$VI = \{PK(v_i), PK(flag), H(F)\}$;

$PK(V) = PK(\sum v[1, \ldots, N])$;

But the return verification information contains $H(F)$. At this time, we decrypt the return ciphertext $C'(w)$ to get F'. If you want to pass the verification, then $H(F) = H(F')$; that is, $F = F'$. If the plaintext is the same, the result $C(w) = C'(w)$ after encryption with the same key, which is inconsistent with the assumption that $C(w) \neq C'(w)$. The above calculation shows that if only the security index is updated without the ciphertext, the Verify Algorithm 4 cannot output 'Y' when the data user performs verification.

(2) VI $\neq$ VI', C(w) = C'(w)

In this case, the cloud server only updates the ciphertext but not the verification information in the security index.

$VI = \{PK(v_i), PK(flag), H(F)\}$;

$VI' = \{PK(v_i'), PK(flag'), H(F')\}$;

If we want the Verify Algorithm 4 to output 'Y', that means

$\{PK(v_i), PK(flag), H(F)\} = \{PK(v_i'), PK(flag'), H(F')\}$.

Since the verification information in the security index is not updated, if $PK(v_i) \neq PK(v_i')$ and the rest are equal, the total update times $V = \sum v[1, \ldots, N']$ will output 'N' when verifying the returned results. If $PK(flag) \neq PK(flag')$ and the rest are the same, the number of returned results is not equal to N, and the Verify Algorithm 4 will output 'N'. If the number of $flag = randA$ is the same as the number of $flag' = randA$ in the returned VI, it is also necessary to ensure that the v_i corresponding to the two are the same, which indicates that the attacker needs to obtain the verification information from the data owner, but VL is private to the data owner, and the probability of information leakage can be almost ignored. If $H(F) \neq H(F')$ and the rest are the same, in this case $F \neq F'$, the Verify Algorithm 4 will output 'N'.

(3) VI $\neq$ VI', C(w) $\neq$ C'(w)

Assume that the cloud server does not update the security index and ciphertext after receiving the update token from the data owner. If the data user performs a Search operation (Algorithm 3), the cloud server returns the un-updated results to the user. In the system model architecture (Figure 1), before performing the Search operation, the data user needs to send the latest VI request for the searched keyword to the data owner. The data owner searches VL and returns the latest VI of the keyword to the data user. After receiving the VI, the data user uses the Verify Algorithm 4 to compare the un-updated VI with the latest. If any inconsistency is found, the Verify Algorithm 4 directly outputs 'N'.

The above shows that if the cloud server does not update the security index and ciphertext to save computing or storage resources, our scheme can verify the verification information and return results through the verification algorithm to find the un-updated situation in time. Therefore, the v-PADSSE scheme proposed by us meets the updated reliability.

5.2. Verifiability Analysis

In this section, we assume that the attacker can forge $(C'(w), VI')$ so that the returned results pass the Verify Algorithm 4. Assuming that the correct results and verification information are $(C(w), VI)$, we will compare the forged information with the real information to prove that the probability of the attacker passing the Verify Algorithm 4 through forging verification information is negligible.

We will consider the following three scenarios to demonstrate the verifiability of the v-PADSSE.

(1) VI = VI', C(w) $\neq$ C'(w)

Attackers forge $VI' = PK(flag) + PK(VI) + H(F')$, while proper verification information $VI = PK(flag) + PK(VI) + H(F)$. This makes:

$PK(flag') + PK(v_i') + H(F') = PK(flag) + PK(v_i) + H(F)$;

In this case, because N, V, $flag$, and v_i are encrypted using the user's public key, the cost of forgery is relatively small. At this point, we can consider:

$H(F') = H(F)$;

According to the properties of the hash function, $F' = F$ is certain. In this case, $C'(w) = C(w)$, which is not consistent with the assumption. Therefore, the probability of an attacker passing the Verify Algorithm 4 in this way is almost negligible.

(2) VI $\neq$ VI', C(w) = C'(w)

The attacker forges the ciphertext $C'(w)$ to be consistent with the correct ciphertext. According to the design of the v-PADSSE, the verification information contains $H(F)$. At this point, we can consider:

$PK(v_i') + PK(flag') \neq PK(v_i) + PK(flag)$;

According to the above situation, $PK(V) = PK(\sum v_i) = PK(\sum v_i')$; when $PK(flag') \neq PK(flag)$, if the number of $flag = randA$ is not equal to the number of $flag' = randA$, the known probability of information leakage of VL can be ignored. In this case, the probability

of the number of returned results being N is negligible, and the Verify Algorithm 4 will output 'N'. Therefore, the probability of the above situation passing the Verify Algorithm 4 can also be ignored.

(3) VI $\neq$ VI', C(w) $\neq$ C'(w)

In this case, the data user will spend a communication after sending the trapdoor to the data owner to request the latest verification information N, V of the keyword. Therefore, under this assumption, as long as any of the verification information is different, or the encrypted files returned are different, $H(F)$ is inconsistent, which will make the Verify Algorithm 4 output 'N'. Therefore, the probability of the attacker passing the verification can be ignored under this condition.

The above three scenarios show that if a malicious attacker forges ciphertext or verification information, our scheme can also determine which information is forged and give feedback. Therefore, our scheme satisfies verifiability.

6. Performance and Experiments

In this section, we will analyze the performance of the proposed v-PADSSE scheme. The basic logic of the experiment was written in C++, and the running environment was Windows 10 equipped with a 2.40 GHz 12th Gen Intel(R) Core(TM) i7 CPU and 4.0 GB RAM.

Index construction efficiency. We evaluated the bitmap index proposed in the scheme and the verification list construction efficiency. Figure 4 shows the time spent to build the security index and verification list when the number of keywords is set to 10,000 and the number of privacy files changes from 1000 to 10,000. In the scheme, the security index adopts the form of a bitmap index, the number of rows is the number of keywords, and the number of columns is the number of document identifiers of privacy files. When the number of rows in the bitmap index is fixed and the number of private files increases, the number of columns in the bitmap also needs to increase, and the time cost of building a secure index also increases. Figure 5 shows the time spent to construct the security index and verification list when the number of privacy files is 10,000 and the number of keywords contained in the privacy files changes from 1000 to 10,000. When the number of secure index columns is fixed, the increase in the number of keywords leads to an increase in the number of rows in the bitmap index, and the time cost of building the secure index Algorithm 1 also increases. During secure index construction, the number of nodes is related to the number of keywords and privacy files. Therefore, when the number of privacy files or keywords increases, the number of columns or rows of the bitmap index will also increase, and the time cost of building a security index will also increase. Since the number of nodes in VL is only related to the number of keywords contained in the privacy file, when the number of keywords increases, the number of nodes in VL increases, and the time cost of building VL (Algorithm 2) increases at the same time.

Update token generation efficiency. Figure 6 shows the time cost of generating update tokens (Algorithm 5) (modify token, delete token, and add token) in the scheme. Since the generation of the update token involves the document identification and the number of keywords contained in the privacy file after the document identification of the modified file is determined, the latest update times of the file need to be obtained from VL. Therefore, the generation efficiency of update tokens is linearly related to the number of nodes in VL, and the higher the number, the longer the token generation time. However, since the added token may involve increasing the number of VL nodes, the generation time will be slightly longer.

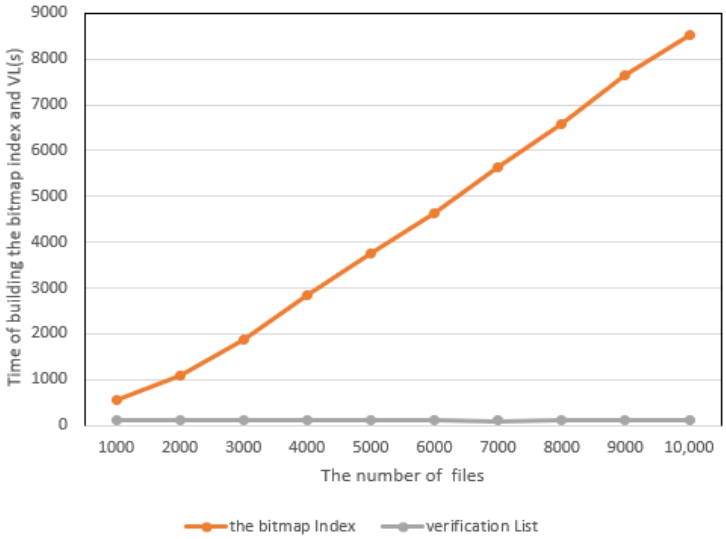

Figure 4. Security index and VL construction time cost.

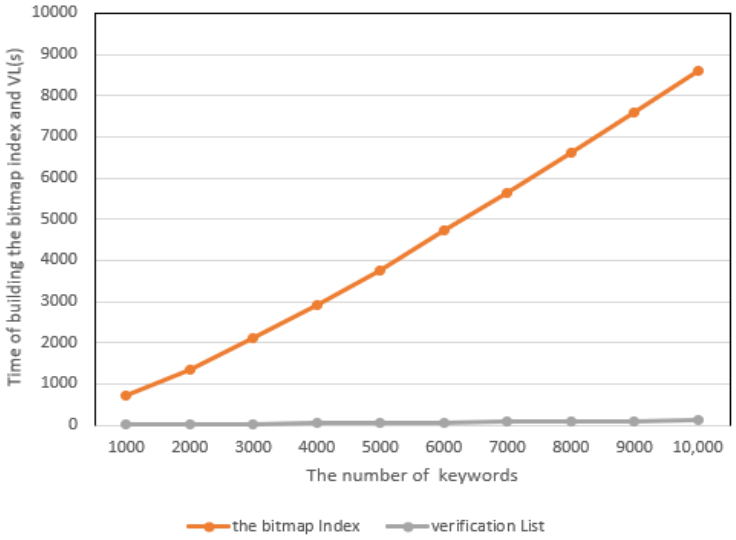

Figure 5. Security index and VL construction time cost.

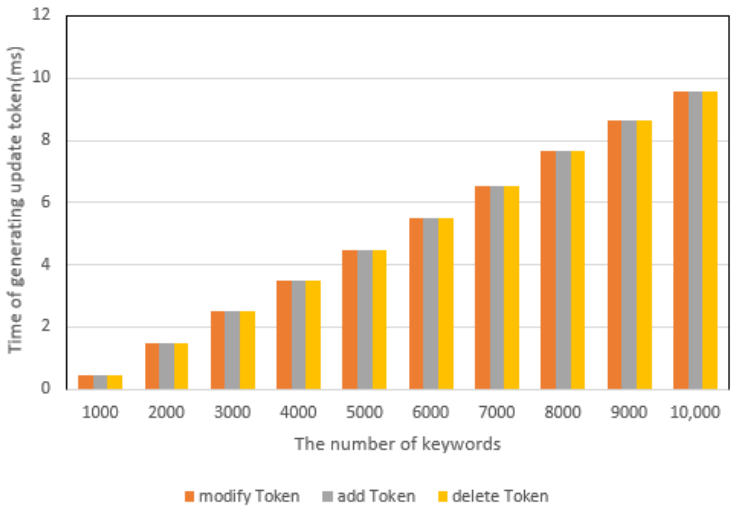

Figure 6. The update token generation time cost.

Search efficiency analysis. The VL obtains the latest verification information and generates an update token. After receiving the update token, the cloud server searches for it in the security index. Figure 7 shows the time cost of performing a search operation in the security index when the number of keywords is 10,000 and the number of private files changes from 1000 to 10,000. It can be seen that when the number of rows in the security index is fixed, that is, the number of keywords is fixed, the time cost of searching the index increases linearly with the increase in the number of columns, that is, the number of privacy files. Figure 8 shows the time cost of searching (Algorithm 3) the VL and the secure index when the number of privacy files is 10,000 and the number of keywords changes from 1000 to 10,000. Since the number of nodes in VL is equal to the number of keywords, the search time also increases linearly when the number of keywords increases. When the number of columns in the security index is fixed and the number of rows in the bitmap index increases as the number of keywords increases, the search time cost increases.

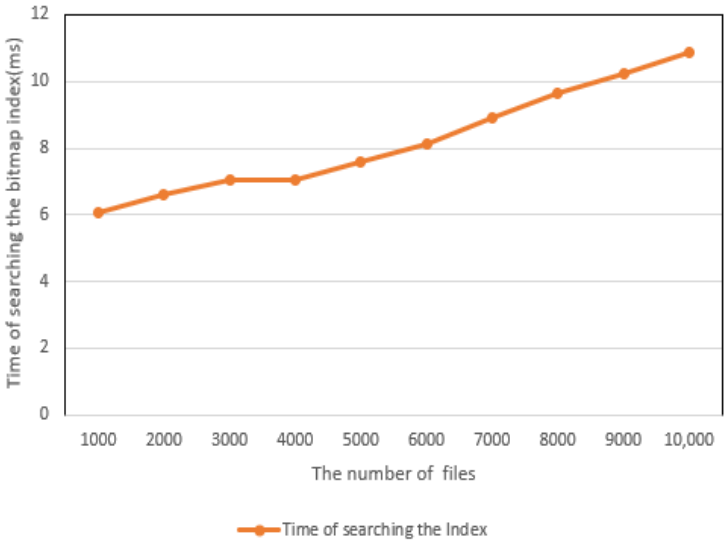

Figure 7. Search secure index time cost.

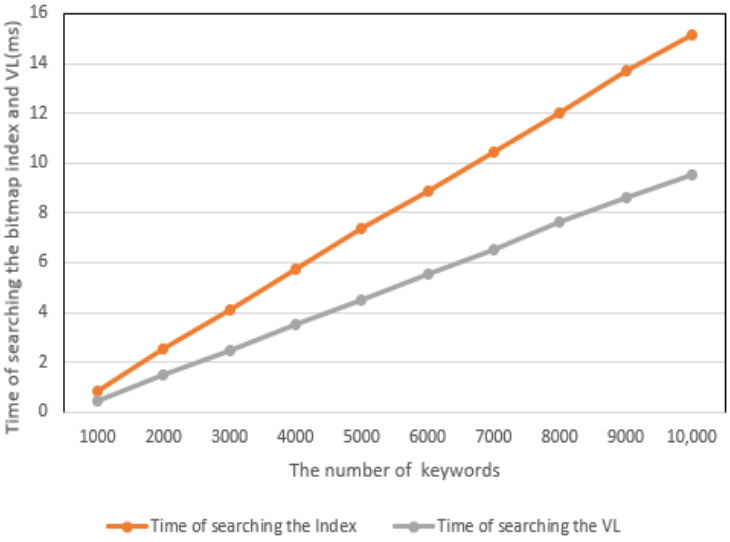

Figure 8. Search secure index and VL time cost.

In this section, we compare this scheme with what is generally regarded as the most typical verifiable SSE scheme [54] in terms of verification efficiency and update efficiency.

Verify efficiency analysis. We made a comparative analysis of the verification (Algorithm 4) time cost of our scheme and scheme [54]. As can be seen from Figure 9,

the verification time cost of our scheme is lower than scheme [54]. Scheme [54] used a bilinear mapping accumulator to verify search results, which is based on asymmetric key encryption. Our scheme is based on a VL; the number of file updates is stored in the VL, which is more efficient than the accumulator. As shown in Figure 9, when the number of privacy files containing search keywords is 200, the verification time cost of scheme [54] is roughly 5 ms, and the verification time cost of our scheme is 0.3725 ms. When the number of search keywords is 2000, the verification time cost of scheme [54] is about 48 ms, and the verification time cost of our scheme is about 9.2437 ms. As can be seen from Figure 10, the number of CPU clock cycles of our scheme is lower than that of scheme [54]. Therefore, the verification efficiency of our scheme is higher than that of scheme [54].

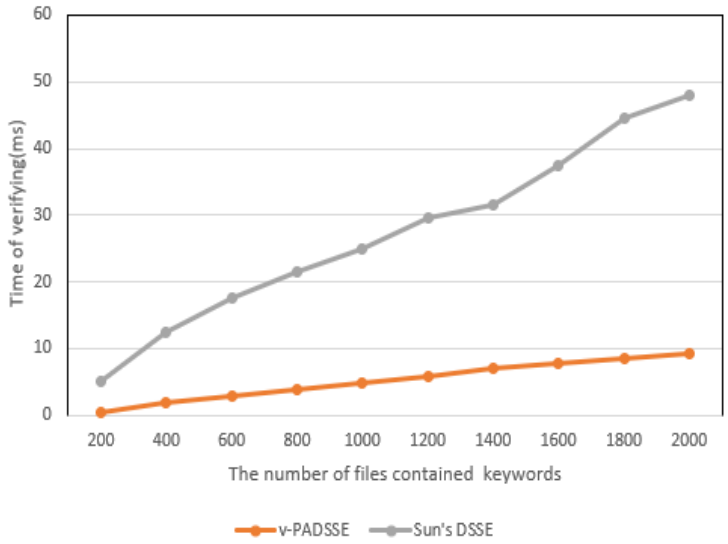

Figure 9. Verification efficiency comparison.

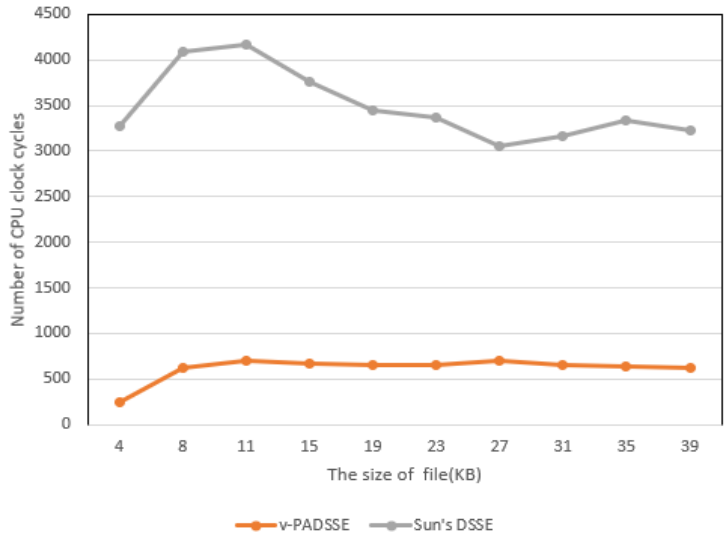

Figure 10. CPU clock cycles of per data comparison.

Update efficiency analysis. After receiving the update token, the cloud server needs to perform the corresponding Update operation (Algorithm 6) on the security index. As can be seen from Figures 11–13, the number of columns or rows in the security index needs to be increased due to the add and modify operation, and the time cost is slightly larger than the delete operation. The cloud server deletes the corresponding column in the security index when it performs the delete operation, and the time cost is lower. As can be seen from Figures 11–13, the update efficiency of our scheme is better than scheme [54].

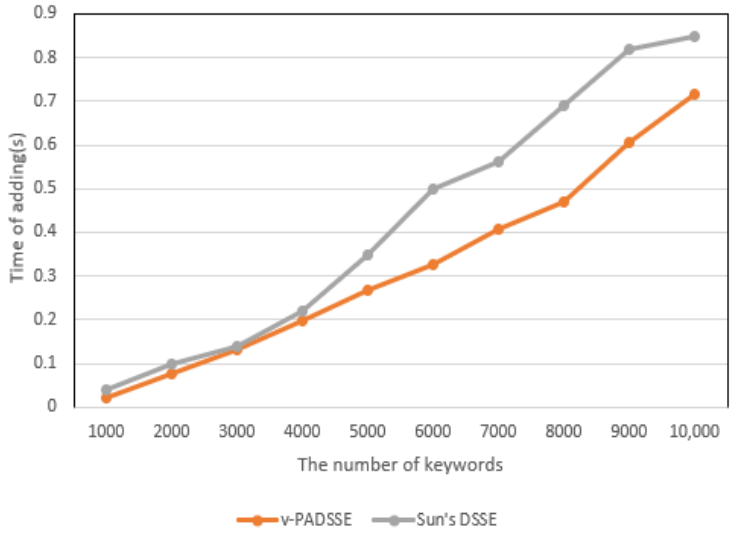

Figure 11. The comparison of add operation time cost.

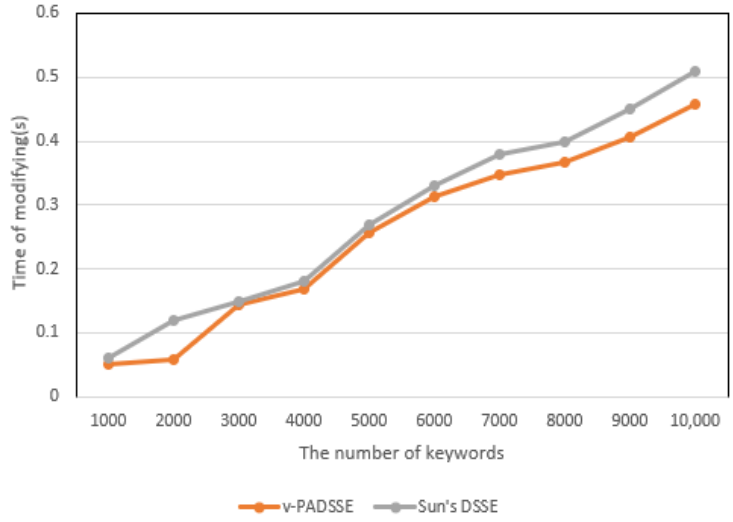

Figure 12. The comparison of modify operation time cost.

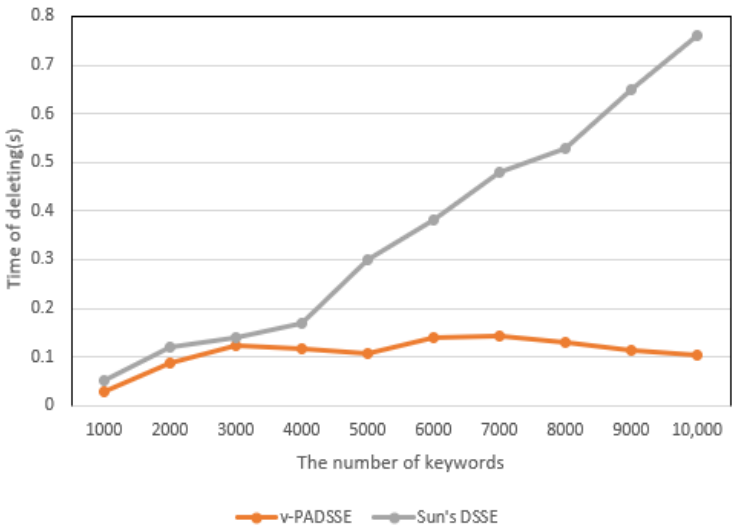

Figure 13. The comparison of delete operation time cost.

7. Discussion

In this section, we analyze the advantages and disadvantages of schemes [18,19,43,51], as shown in Table 3. Σ*οφος* [51], Zhang's scheme [43], and Gao's scheme [19] realize the dynamic update and searchability of data through constructing an inverted index. If the update file contains many keywords, the update efficiency is relatively low. None of the above four schemes involve key management securely. Σ*οφος* [51] uses a one-way trap gate to realize forward security, but the calculation cost is high. Zhang's scheme [43] is improved on the basis of Σ*οφος* [51], using random states to achieve forward safety and improve efficiency. However, the correctness of the returned results is not verified; that is, the update reliability proposed in this scheme is not satisfied. Gao's scheme [19] requires third-party TPA to verify the integrity of search results, which requires TPA to honestly implement the verification algorithm, which requires a trap gate, data block number, RAL, and authenticator to perform related calculations, which is relatively complex. Both Ge's scheme [18] and our scheme used a bitmap to construct the security index, which has high updating efficiency. However, the accumulated authentication tags (AATs) in Ge's scheme [18] contain ciphertext data blocks, which consume additional storage resources. In our scheme, the verification process does not involve complex operations, and the verification information is simple, which makes the verification efficiency high, but the scheme also needs to consume communication resources once more. In conclusion, compared with the above schemes, our scheme is relatively efficient in the process of searching, updating, and verifying.

Table 3. Advantages and disadvantages.

Schemes	Advantages	Disadvantages
Σ*οφος* [51]	Effectively implement forward security.	The calculation of trapdoor replacement is costly.
Zhang's scheme [43]	The multi-level hash function is used to replace the one-way permutation function, and the update efficiency is improved significantly.	The correctness of the returned result is not verified.
Gao's scheme [19]	The verification process effectively hides the relation between the keywords and the encrypted files to avoid statistical attacks.	The verification process is complex, and TPA must perform the verification process honestly.
Ge's scheme [18]	The verification of the returned results consumes only one AAT resource, which has high efficiency.	AAT contains ciphertext data blocks, which consume additional storage resources.
our scheme	The verification procedure is simple and does not involve complex operations, which has high efficiency.	The verification process consumes an additional communication resource.

8. Conclusions

In this paper, we first studied the research status of the DSSE scheme and analyzed the advantages and disadvantages of different schemes. Since most schemes do not involve key management, we proposed a verifiable DSSE scheme based on the public key cryptosystem which can realize secure key management. In Section 3, we defined the security model, design goals, core algorithm, and security analysis of the V-PDSSE scheme. In Section 4, we described the bitmap index and verification list construction of the scheme in detail and explained the core algorithm steps of the scheme. Finally, we compared the time complexity with schemes [18,19,43,51] to prove that the implementation efficiency of our scheme is high. In Section 5, a security analysis was carried out on the reliability and verifiability of our scheme to prove that our scheme meets the security requirements. In Section 6, we tested the efficiency of the core algorithms and compared the efficiency of scheme [54] in security index construction, verification list construction, searching, search result verification, and updating. The results show that our scheme has high efficiency and strong feasibility. In Section 7, we analyzed the advantages and disadvantages of schemes [18,19,43,51].

Compared with previous schemes, the functional design of this scheme is more comprehensive. The verification process does not involve complex operations, the verification information structure is simple, and the execution efficiency is high. Our scheme can solve the problems of dynamic searchability, forward security, integrity, and correct verification of search results and key management well. In future work, given the rapid development of quantum computers and verifiable DSSE schemes, how to deal with quantum attacks effectively will become a key direction of our research.

Author Contributions: Writing—original draft preparation, G.D.; writing—review and editing, S.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported by the Natural Science Foundation of Ningxia (No. 2021AAC03068) and the Key R&D Program of Ningxia (No. 2021BEG03071).

Data Availability Statement: Not applicable.

Acknowledgments: The authors would like to thank the editor and the anonymous reviewers for their valuable comments and suggestions that improved the quality of this paper.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Xu, L.; Xu, C.G.; Liu, Z.Y.; Wang, Y.L.; Wang, J.F. Enabling comparable search over encrypted data for IoT with privacy-preserving. *Comput. Mater. Contin.* **2019**, *60*, 675–690. [CrossRef]
2. 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.
3. Islam, M.S.; Kuzu, M.; Kantarcioglu, M. *Access Pattern Disclosure on Searchable Encryption: Ramification, Attack and Mitigation*; Proc.NDSS: New York, NY, USA, 2012; pp. 1–15.
4. Zhang, Y.P.; Katz, J.; Papamanthou, C. All your queries are belong to us: The power of file-injection attacks on searchable encryption. *IACR Cryptol. Eprint Arch.* **2016**, *172*, 707–720.
5. Cash, D.; Grubbs, P.; Perry, J.; Ristenpart, T. Leakage-abuse attacks against searchable encryption. In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'15), Denver, CO, USA, 12–16 October 2015; pp. 668–679.
6. Kamara, S.; Papamanthou, C.; Roeder, T. Dynamic searchable symmetric encryption. In Proceedings of the ACM Conference on Computer and Communications Security (CCS 2012), Raleigh, NC, USA, 16–18 October 2012; pp. 965–976.
7. Kamara, S.; Papamanthou, C. Parallel and dynamic searchable symmetric encryption. In Proceedings of the 17th International Conference on Financial Cryptography and Data Security (FC 2013), Okinawa, Japan, 1–5 April 2013; Sadeghi, A.-R., Ed.; Springer: Berlin/Heidelberg, Germany, 2013; pp. 258–274.
8. Guo, C.; Chen, X.; Jie, Y.M.; Fu, Z.J.; Li, M.C.; Feng, B. Dynamic multi-phrase ranked search over encrypted data with symmetric searchable encryption. *IEEE Trans. Serv. Comput.* **2017**, *13*, 1034–1044. [CrossRef]
9. Xia, Z.H.; Wang, X.H.; Sun, X.M.; Wang, Q. A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data. *IEEE Trans. Parallel Distrib. Syst.* **2016**, *27*, 340–352. [CrossRef]
10. Liu, Q.; Nie, X.H.; Liu, X.H.; Peng, T.; Wu, J. Verifiable ranked search over dynamic encrypted data in cloud computing. In Proceedings of the 2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS), Vilanova i la Geltrú, Spain, 14–16 June 2017; pp. 1–6.
11. Nie, X.H.; Liu, Q.; Liu, X.H.; Peng, T.; Lin, Y.P. Dynamic verifiable search over encrypted data in untrusted clouds. In *Algorithms and Architectures for Parallel Processing*; Springer: Cham, Switzerland, 2016; pp. 557–571.
12. Zhu, X.Y.; Liu, Q.; Wang, G.J. A novel verifiable and dynamic fuzzy keyword search scheme over encrypted data in cloud computing. In Proceedings of the 2016 IEEE Trustcom/BigDataSE/ISPA, Tianjin, China, 23–26 August 2016; pp. 845–851.
13. Liu, Q.; Tian, Y.; Wu, J.; Peng, T.; Wang, G. Enabling verifiable and dynamic ranked search over outsourced data. *IEEE Trans. Serv. Comput.* **2022**, *15*, 69–82. [CrossRef]
14. Jiang, X.X.; Yu, J.; Yan, J.B.; Hao, R. Enabling efficient and verifiable multi-keyword ranked search over encrypted cloud data. *Inf. Sci.* **2017**, *403–404*, 22–41. [CrossRef]
15. Chen, F.; Xiang, T.; Fu, X.W.; Yu, W. User differentiated verifiable file search on the cloud. *IEEE Trans. Serv. Comput.* **2018**, *11*, 948–961. [CrossRef]
16. Wan, Z.G.; Deng, R.H. VPSearch: Achieving verifiability for privacy preserving multi-keyword search over encrypted cloud data. *IEEE Trans. Depend. Sec. Comput.* **2018**, *15*, 1083–1095. [CrossRef]
17. Kurosawa, K.; Ohtaki, Y. How to update documents verifiably in searchable symmetric encryption. In *Cryptology and Network Security*; Springer: Cham, Switzerland, 2013; pp. 309–328.

18. Ge, X.; Yu, J.; Zhang, H.; Hu, C.; Li, Z.; Qin, Z.; Hao, R. Towards achieving keyword search over dynamic encrypted cloud data with symmetric-key based verification. *IEEE Trans. Dependable Secur. Comput.* **2021**, *18*, 490–504. [CrossRef]
19. Gao, X.; Yu, J.; Chang, Y.; Wang, H.; Fan, J. Checking only when it is necessary: Enabling integrity auditing based on the keyword with sensitive information privacy for encrypted cloud data. *IEEE Trans. Dependable Secur. Comput.* **2021**, *19*, 3774–3789. [CrossRef]
20. Yu, J.; Ren, K.; Wang, C. Enabling cloud storage auditing with key-exposure resistance. *IEEE Trans. Inf. Forensics Secur.* **2015**, *10*, 1167–1179.
21. Zhang, Y.; Yu, J.; Hao, R.; Wang, C.; Ren, K. Enabling efficient user revocation in identity-based cloud storage auditing for shared big data. *IEEE Trans. Dependable Secur. Comput.* **2018**, *17*, 608–619. [CrossRef]
22. Shacham, H.; Waters, B. Compact Proofs of Retrievability. In Proceedings of the 14th Annual International Conference on the Theory and Application of Cryptology & Information Security, Melbourne, Australia, 7–11 December 2008; pp. 90–107.
23. Miao, Y.B.; Ma, J.F.; Liu, X.M.; Li, X.H.; Jiang, Q.; Zhang, J.W. Attribute-based keyword search over hierarchical data in cloud computing. *IEEE Trans. Serv. Comput.* **2017**, *13, 985–998*. [CrossRef]
24. Miao, Y.B.; Ma, J.F.; Liu, X.M.; Weng, J.; Li, H.W.; Li, H. Lightweight fine-grained search over encrypted data in fog computing. *IEEE Trans. Serv. Comput.* **2018**, *12*, 772–785. [CrossRef]
25. Liu, X.; Yang, G.; Mu, Y.; Deng, R.H. Multi-user verifable searchable symmetric encryption for cloud storage. *IEEE Trans. Dependable Secur. Comput.* **2020**, *17*, 1322–1332. [CrossRef]
26. Xu, P.; Wu, Q.; Wang, W.; Susilo, W.; Domingo-Ferrer, J.; Jin, H. Generating searchable public-key ciphertexts with hidden structures for fast keyword search. *IEEE Trans. Inf. Forensics Secur.* **2015**, *10*, 1993–2006.
27. Fu, Z.J.; Xia, L.L.; Sun, X.M.; Liu, A.X.; Xie, G.W. Semantic aware searching over encrypted data for cloud computing. *IEEE Trans. Inf. Forensics Secur.* **2018**, *13*, 2359–2371. [CrossRef]
28. Zhang, W.; Xiao, S.; Lin, Y.P.; Zhou, T.; Zhou, S.W. Secure ranked multi-keyword search for multiple data owners in cloud computing. In Proceedings of the 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Atlanta, GA, USA, 23–26 June 2014; pp. 276–286.
29. Fu, Z.J.; Sun, X.M.; Linge, N.; Zhou, L. Achieving effective cloud search services: Multi-keyword ranked search over encrypted cloud data supporting synonym query. *IEEE Trans. Consum. Electron.* **2014**, *60*, 164–172. [CrossRef]
30. 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.
31. Zhang, R.; Xue, R.; Liu, L. Searchable encryption for healthcare clouds: A survey. *IEEE Trans. Serv. Comput.* **2018**, *11*, 978–996. [CrossRef]
32. Xu, C.; Wang, N.; Zhu, L.; Sharif, K.; Zhang, C. Achieving searchable and privacy-preserving data sharing for cloud-assisted E-healthcare system. *IEEE Internet Oftings J.* **2019**, *6*, 8345–8356. [CrossRef]
33. Emura, K.; Miyaji, A.; Omote, K. A timed-release proxy re-encryption scheme. *IEICE-Trans. Fundam. Electron. Commun. Comput. Sci.* **2011**, *E94-A*, 1682–1695. [CrossRef]
34. Yang, Y.; Ma, M. Conjunctive keyword search with designated tester and timing enabled proxy Re-encryption function for E-health clouds. *IEEE Trans. Inf. Forensics Secur.* **2016**, *11*, 746–759. [CrossRef]
35. Watanabe, Y.; Shikata, J. Timed-release computational secret sharing and threshold encryption. *Des. Codes Cryptogr.* **2018**, *86*, 17–54. [CrossRef]
36. Emura, K.; Hayashi, T.; Ishida, A. Group signatures with timebound keys revisited: A new model, an efcient construction, and its implementation. *IEEE Trans. Dependable Secur. Comput.* **2020**, *17*, 292–305. [CrossRef]
37. Xu, P.; He, S.; Wang, W.; Susilo, W.; Jin, H. Lightweight searchable public-key encryption for cloud-assisted wireless sensor networks. *IEEE Trans. Ind. Inform.* **2018**, *14*, 3712–3723. [CrossRef]
38. Xiong, H.; Wang, Y.; Li, W.; Chen, C.M. Flexible, efficient, and secure access delegation in cloud computing. *ACM Trans. Manag. Inf. Syst. (TMIS)* **2019**, *10*, 2. [CrossRef]
39. Miao, Y.; Tong, Q.; Deng, R.H.; Choo, K.K.R.; Liu, X.; Li, H. Verifiable searchable encryption framework against insider keyword-guessing attack in cloud storage. *IEEE Trans. Cloud Comput.* **2020**, *10*, 835–848. [CrossRef]
40. Wu, D.N.; Gan, Q.Q.; Wang, X.M. Verifiable public key encryption with keyword search based on homomorphic encryption in multi-user setting. *IEEE Access* **2018**, *6*, 42445–42453. [CrossRef]
41. Chai, Q.; Gong, G. Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers. In Proceedings of the 2012 IEEE International Conference on Communications (ICC), Ottawa, ON, Canada, 10–15 June 2012; pp. 917–922.
42. Yang, L.; Zheng, Q.; Fan, X. Rspp: A reliable, searchable and privacy-preserving e-healthcare system for cloud-assisted body area networks. In Proceedings of the IEEE INFOCOM 2017-IEEE Conference on Computer Communications, Atlanta, GA, USA, 1–4 May 2017; pp. 1–9.
43. Zhang, Z.; Wang, J.; Wang, Y.; Su, Y.; Chen, X. Towards efficient verifiable forward secure searchable symmetric encryption. In *Computer Security-ESORICS 2019*; Springer International Publishing: Cham, Switzerland, 2019; Volume 11736, pp. 304–321.
44. Deepa, N.; Perumal, P. Hybrid context aware recommendation system for e-health care by merkle hash tree from cloud using evolutionary algorithm. *Soft. Comput.* **2020**, *24*, 7149–7161. [CrossRef]
45. Zhu, J.; Li, Q.; Wang, C.; Yuan, X.; Wang, Q.; Ren, K. Enabling generic, verifiable, and secure data search in cloud services. *IEEE Trans. Parallel Distrib. Syst.* **2018**, *29*, 1721–1735. [CrossRef]

46. Chen, C.; Zhu, X.; Shen, P.; Hu, J.; Guo, S.; Tari, Z.; Zomaya, A.Y. An efficient privacy-preserving ranked keyword search method. *IEEE Trans. Parallel Distrib. Syst.* **2015**, *27*, 951–963. [CrossRef]
47. Cheng, H.; Wang, H.; Liu, X.; Fang, Y.; Wang, M.; Zhang, X. Person re-identification over encrypted outsourced surveillance videos. *IEEE Trans. Dependable Secur. Comput.* **2019**, *18*, 1456–1473. [CrossRef]
48. Guo, Y.; Zhang, C.; Jia, X. Verifiable and forward-secure encrypted search using blockchain techniques. In Proceedings of the 2020 IEEE International Conference on Communications (ICC), Dublin, Ireland, 7–11 June 2020; pp. 1–7.
49. Wang, Q.; He, M.; Du, M.; Chow, S.; Lai, R.; Zou, Q. Searchable encryption over feature-rich data. *IEEE Trans. Depend. Sec. Comput.* **2018**, *15*, 496–510. [CrossRef]
50. Du, M.; Wang, Q.; He, M.; Weng, J. Privacy-preserving index-ing and query processing for secure dynamic cloud storage. *IEEE Trans. Inf. Forensics Secur.* **2018**, *13*, 2320–2332. [CrossRef]
51. Bost, R. $\Sigma o \phi o \varsigma$: forward secure searchable encryption. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS 2016), Vienna, Austria, 24–28 October 2016; Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S., Eds.; ACM: New York, NY, USA, 2016; pp. 1143–1154.
52. Cao, N.; Wang, C.; Li, M.; Ren, K.; Lou, W. Privacy-preserving multi-keyword ranked search over encrypted cloud data. *IEEE Trans. Parallel Distrib. Syst.* **2014**, *25*, 222–233. [CrossRef]
53. Li, J.; Huang, Y.; Wei, Y.; Lv, S.; Liu, Z.; Dong, C.; Lou, W. Searchable Symmetric Encryption with Forward Search Privacy. *IEEE Trans. Dependable Secur. Comput.* **2021**, *18*, 464–465. [CrossRef]
54. Sun, W.H.; Liu, X.F.; Hou, W.J.L.Y.T.; Li, H. Catch you if you lie to me: Efficient verifiable conjunctive keyword search over large dynamic encrypted cloud data. In Proceedings of the 2015 IEEE Conference on Computer Communications (INFOCOM), Hong Kong, China, 26 April–1 May 2015; pp. 2110–2118.