Dynamic searchable symmetric encryption schemes with forward and backward security

Ke Huang, Xiaolei Dong, Zhenfu Cao and Jiachen Shen*
Shanghai Key Laboratory of Trustworthy Computer, East China Normal University, Shanghai 200062, China

*Email: jcshen@sei.denu.edu.cn

Abstract. Dynamic searchable symmetric encryption (DSSE) is helpful to users with limited storage. In DSSE, the users are able to perform search and update queries on the ciphertext which stored on the cloud server. However, DSSE may suffer from file-injection attacks and leak information after deletion. In order to solve these problems, DSSE schemes with forward security and/or backward security have been proposed. In this paper, we propose two efficient DSSE schemes. The first one supports forward security using binary tree as the basic data structure. The second one uses puncturable encryption to achieve both forward and backward privacy.

1. Introduction
With the development of cloud technology, when local storage resources are limited, user can consider storing files on the cloud server. However, since the external cloud server is untrusted, the file needed to be encrypted and uploaded to the cloud server. In order to guarantee the searchable of these encrypted files, searchable encryption is proposed. In short, searchable encryption ensure that user can search on encrypted files which stored on a cloud server. Dynamic searchable encryption allows the server to dynamically update the stored data without changing the confidentiality of the data and without losing the searchable feature. But most of the current dynamic searchable encryption schemes have the risk of being attacked by file injection. The adversary injects new files obtained some keywords into the encrypted database and then do some query operations. By that way, the adversary may learn the user's private information. Similarly, when deleting files, it will also cause some leakage of private information. Therefore, for the addition and deletion, forward security and backward security are proposed respectively to protect the confidentiality of files.

1.1. Contribution
We will propose two DSSE constructions in this paper. The first one is forward security with basic data structure binary tree. The second one achieves both forward privacy and backward privacy. In more details, our main contributions are as follows:

Binary tree. We use binary tree as a basic data structure to store encrypted database to improve query efficiency, include search queries and update queries; change the structure of leaf node on thr tree to satisfy our DSSE scheme.

Forward privacy. By improving the composition of the search token base on Bost and combine the data structure of binary tree, we construct an efficien DSSE scheme with forward security.
Backward privacy. Based on our forward security DSSE scheme, we use puncturable encryption to achieve backward security by adding a tag.

1.2. Related work

Searchable Encryption

Symmetric searchable encryption (SSE) was first introduced by Song et al. [1]. Curtmola et al. [2] gave the modern definitions of SSE and corresponding security model. They designed the first reversed-index-based SSE construction. Chase and Kamara [3] focused on multi-maps, which was a base structure of their scheme. After that, SSE is a particular case of structured encryption.

However, the above schemes cannot support data update. To solve this problem, some Dynamic symmetric searchable encryption (DSSE) schemes have been proposed. Kamara and Papamanthou [4] designed the first sublinear dynamic scheme. Cash et al. [5] constructed a dynamic scheme optimized for large datasets.

Many DSSEs could suffer file-injection attack proposed by Zhang et al. [6]. In file-injection attack, the adversary can infer some information through client queries by inject some new documents into encrypted database. So, the need for guaranteeing forward security which protects security of new added parts raised. Similarly, we have backward security which protects security of new added parts and later deleted. The notions of forward and backward security were first introduced by Stefanov et al. [7]. The authors present an ORAM-inspired SSE construction. However, this was only designed to improve the performance of the scheme, rather than its security and the overhead of ORAM is too high for a practical SSE scheme [8]. Forward and backward security formal definitions for DSSE were given by Bost [9] and Bost et al. [10]. In [9], Bost proposed a DSSE scheme only support forward security. Later on, Bost et al. [10] proposed a DSSE scheme support both forward and backward security.

Puncturable encryption

In [11], Green and Miers introduced a new form of encryption named puncturable encryption. Puncturable encryption is a way to provide forward secrecy in asynchronous messaging systems. Their scheme modifies the secret key every time a message is received, and modified key has decryption capability only for special messages. That means old messages remain safe. They also mentioned secure deletion as another application of their work.

In this paper, we will use puncturable encryption to securely delete entries in an encrypted database.

2. Preliminaries

2.1. Dynamic symmetric searchable encryption

We follow the database model given in the paper [10]. A database DB is a collection of keyword/index pairs. W is a set of keywords and ind stand for index of document. \( DB = \left\langle ind_i, w_i \right\rangle \), \( ind_i \in (0, 1)^* \) and \( w_i \subseteq (0, 1)^* \).

\( D \) is the number of documents in DB and \( W \) is the number of keywords and \( N \) is the number of index/keyword pairs. DB(w) represents the set of indices of documents which contain a keyword w.

A DSSE scheme consists of an algorithm Setup and two protocols Search and Update as described below.

EDB ← Setup(DB, \( 1^\lambda \)): Input a security parameter \( 1^\lambda \) and a database DB. The algorithm outputs an encrypted database EDB

(index, \( \bot \)) ← Search(q, \( \sigma \), EDB): The protocol is executed between a client and a server. The client asks for documents with given keyword. The server returns a collection of file indices of requested documents.

(\( \sigma \), EDB) ← Update(\( \sigma \), op, in, EDB): The protocol runs between a client and a server. According to the op = (add, del), client send different tokens to server and the server do update on EDB.

2.2. Forward and backward privacy

Forward security (or forward privacy) is that an SSE scheme won’t leak what keywords are associated
whit the keyword/document pairs while doing update queries.

Backward security (or backward privacy) is that an SSE scheme can’t get that document index when doing a search query after a keyword/document pair is deleted.

Forward privacy and Backward privacy were informally defined in [7]. In this paper, we use the formal definition of [10].

3. Our schemes

3.1. Data structure binary tree

3.1.1. Binary tree structure

![Binary tree structure](image)

**Figure 1.** Basic data structure binary tree.

In our construction, a perfect binary tree is used as the basic data structure. As shows in figure 1. The value of a leaf node represents a keyword. But not every leaf node is real, as indicated by the dotted line in figure 1. In addition, a list is stored in the leaf node, which stores the file indices associated with that keyword.

3.1.2. Binary tree assignment and update

We define the following construction, search and update operations to manipulate the base data structure of our DSSE. As shown in the Algorithm 1.

**Algorithm 1. Our Binary Tree**

```
TreeCon (m)
Input number of keywords m
Output complete binary tree PBT
1: Construct a PBT with \( \log (m) \) + 1 levels.
2: Set the top m leaf nodes to be real from left to right.
3: return PBT

TreeAssign (ca, PBT)
Input complete binary tree PBT, an initial counter ca
Output assigned binary tree ABT
1: TreeAssignSub (ca, PBT, root)
2: return ABT

TreeAssignSub (ca, PBT)
Input a node n of PBT, an integer ca
Output Assigned binary tree ABT
1: if n.left ≠ ⊥ then
2:  TreeAssignSub (ca, n.left)
3:  end if
4:  n.value = ca
5:  ca = ca + 1
6: if n.right ≠ ⊥ then
7:  Create a new root node root,
8:  ABT.parent = root,
9:  ABT.parent.value = root,
10:  TreeAssignSub (ca, root)
11:  ABT.root = root,
12:  ABT.root.value = ca + 1

TreeSearch (root, v)
Input Assigned binary tree ABT, a value v
Output a node n of ABT
1: if v = root.value then
2:  return n = root
3:  else if v < root.value then
4:    TreeSearch (root.left, v)
5:  else
6:    TreeSearch (root.right, v)
7:  end if
8:  return n

TreeUpdate (add, v, ABT)
Input add, number of keywords m, value v, ABT
Output updated ABT
1: if ABT = ⊥ then
2:  Create a node.
3:  Associate value v = 0 to this node.
4:  Set ABT to this node.
5:  else if ABT.root.value = m-1 then
6:    m = m + 1
7:    return
8:  else
9:  Execute line 12.
10:  else
11:    end if
12:  return ABT
```
3.2. DSSE with forward privacy
In this section, we apply our binary tree to the Bost[9] scheme to improve the efficiency of search and update queries.

Now we are ready to present the DSSE scheme with forward security. The scheme is described in Algorithm 2.

Setup(\(\lambda\)): For a security parameter \(\lambda\), the algorithm outputs \((PK, SK, K, W, m)\).

Search(w, \(\sigma\), EDB): The protocol is executed between a client and a server. The client asks for documents with keyword w. The server returns a collection of file indices of requested documents.

Update(add, w, \(\sigma\), m, EDB): The protocol is performed jointly by a client and a server. The client wishes to add a keyword w together with a file index \(ind\) to EDB.

Algorithm 2. DSSE with forward security

---

3.3. DSSE with forward and backward privacy
In this section, we apply PPKE [11] to scheme in section 4.2. So now we have a DSSE scheme with both forward and backward security. For details, we add a tag to the file index. When deleting a pair of keyword/index, the server modifies the tag and the client puncture the SK of PPKE. After that, that keyword/index pair couldn't be decrypted.

The scheme is described in Algorithm 3.

Setup(\(\lambda\)): For a security parameter \(\lambda\), the algorithm outputs \((PK, SK, K, W, m)\).

Search(w, \(\sigma\), EDB): The protocol is executed between a client and a server. The client asks for documents with keyword w. The server returns a collection of file indices of requested documents. But only certain indices (not been deleted) will be returned.

Update(op, w, \(\sigma\), m, EDB): The protocol is performed jointly by a client and server. Op = (add, delete). If op = add, the client wishes to add a keyword w together with a file index \(ind\) to EDB. There
are three cases same as the TreeUpdate. If op = delete, the client wishes to delete a keyword/index pair in the EDB.

### Algorithm 3. DSSE with forward and backward security

| Setup(\(\ell\)) |
|------------------|
| Client:          |
| Input: security parameter \(1^w\) |
| Output: (PK, SK, K, W, m) |
| 1: \(K, K_{nap} \leftarrow \{0, 1\}^*\) |
| 2: (SK, PK) \leftarrow KeyGen(\(1^\ell\)) |
| 3: W \leftarrow empty map |
| 4: return (PK, SK, K, K_{nap}, W) |
| Search (w, o, EDB) |
| Client:          |
| Input w, o, EDB |
| Output (K_c, ST_c, c, v) |
| 1: \(K_c \leftarrow F_g(w)\) |
| 2: (ST_c, c, order, sko, PPK) \leftarrow W[w] |
| 3: if (ST_c, c, order, sko, PPK) = \(\perp\) then return \(\emptyset\) |
| 5: end if |
| 6: v = \(2^r\) order |
| 7: Send (K_{nap}, ST_c, c, v) to the server. |
| Server:          |
| Input (K_c, ST_c, c, v), EDB |
| Output \(\{\emptyset\}\) |
| 8: Upon receiving (K_c, ST_c, c, v) |
| 9: node \(n = \text{TreeSearch}(2v)\) |
| 10: \(e = n.\text{list}\) |
| 11: for \(i = c\) to \(0\) do |
| 12: (ST_{c+1} \leftarrow \pi_{pre}(ST_c)) |
| 13: end for |
| 14: for \(j = 0\) to \(c\) do |
| 15: \(e^* = \text{PPKE.Encrypt}(e)\) |
| 16: \(ind \leftarrow e^* \oplus H(K_c, ST_i)\) |
| 17: Output the ind |
| 18: end for |
| Update(op, w, ind, o, EDB) |
| Client:          |
| Input op = (add, del), keyword/index pair \(w\) and \(ind\), State of EDB o |
| Output e |

#### 4. Conclusion

In this paper, we give two DSSE schemes. The first one is forward security with basic data structure binary tree. This basic scheme is improved by [9] and the perfect binary tree is used to improve the efficiency of search and update queries. However, this scheme only meets the requirements of forward security. To address these problems, we propose the second DSSE construction that uses puncturable encryption [11]. It achieves both the forward and backward security. But the above schemes just support the queries on single-keyword. In the future, we would like to construct a DSSE scheme support multi-keywords.

### Acknowledgments

This work was supported in part by National Nature Science Foundation of China (Grant No.61632012, 61672239, and U1509219) and in part by “the Fundamental Research Funds for the Central Universities”. Zhenfu Cao and Jiachen Shen shared the corresponding authors.
References

[1] D. X. Song, D. Wagner, A. Perrig, Practical techniques for searches on encrypted data. IEEE S&P 2000, pp.44–55 (2000)

[2] R. Curtmola, J. Garay, S. Kamara, R. Ostrovsky, Searchable symmetric encryption: improved definitions and efficient constructions. ACM ASIACCS 2006, pp.79–88 (2006)

[3] M. Chase, S. Kamara, Structured encryption and controlled disclosure. ASIACRYPT 2010, 6477, pp.577–594 (2010)

[4] S. Kamara, C. Papamanthou, Parallel and dynamic searchable symmetric encryption. A.R. Sadeghi (ed.) FC 2013, 7859, pp.258–274 (2013)

[5] D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M. C. Rosu, M. Steiner, Dynamic searchable encryption in very-large databases: Data structures and implementation. NDSS 2014 (2014)

[6] Y. Zhang, J. Katz, C. Papamanthou, The fallacy of composition of oblivious RAM and searchable encryption. USENIX Security Symposium, pp.707–720 (2016)

[7] E. Stefanov, C. Papamanthou, E. Shi, Practical dynamic searchable encryption with small leakage. NDSS, 71, pp.72–75 (2014)

[8] M. Naveed, The fallacy of composition of oblivious RAM and searchable encryption. Cryptology ePrint Archive Report 2015/668 (2015)

[9] R. Bost, Σοφος: forward secure searchable encryption. ACM SIGSAC CCS2016, pp.1143–1154 (2016)

[10] R. Bost, B. Minaud, O. Ohrimenko, Forward and backward private searchable encryption from constrained cryptographic primitives. ACM SIGSAC CCS2017, pp.1465–1482 (2017)

[11] M. D. Green, I. Miers, Forward secure asynchronous messaging from puncturable encryption. IEEE S&P 2015, pp.305–320 (2015).