Integrity Preserving Multi-keyword Searchable Encryption for Cloud Computing

Fucai Zhou[1(✉)], Yuxi Li[1], Alex X. Liu[2], Muqing Lin[3], and Zifeng Xu[1]

[1] Software College, Northeastern University, Shenyang 110819, Liaoning, China
fczhou@mail.neu.edu.cn
[2] The Department of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824, USA
[3] The College of Information Science and Engineering, Northeastern University, Shenyang 110819, Liaoning, China

Abstract. Searchable symmetric encryption is an efficient way to perform keyword search over encrypted data in cloud storage. However, most existing methods do not take into account the integrity verification of the search result. Moreover, existing methods can only verify the integrity of single-keyword search results, which cannot meet the requirements of multi-keyword conjunctive search. To address this problem, we proposed a multi-keyword searchable encryption scheme with an authentication mechanism that can efficiently verify the integrity of search results. The proposed scheme is based on the searchable symmetric encryption and adopts the bilinear map accumulator to prove the correctness of set operations. It supports multiple keywords as input for conjunctive search and gives the server the ability to prove the integrity of the search result to the user. Formal proofs show that the proposed scheme is unforgeable and adaptive secure against chosen-keyword attacks. To the best of our knowledge, this is the first work that can authenticate the multi-keyword search result over encrypted data.

Keywords: Conjunctive keyword search · Integrity authentication · Searchable encryption · Secure cloud storage

1 Introduction

Cloud computing is an innovative Internet-based computing paradigm that enables cloud users to move out their data and applications to a remote cloud in order to deploy scalable and elastic services on demand without having to provision a data center. However, while cloud computing has many advantages, it has not been widely used. According to a survey lunched by Twin Strata in 2015, only 38 % of organizations would like to put their inactive data stored in public cloud; about 24 % of users were using cloud storage for data backup, archiving and disaster recovery. This shows that the issue of data security [1, 2] is one of the major obstacles to the promotion of cloud

L. Chen and J. Han (Eds.): ProvSec 2016, LNCS 10005, pp. 153–172, 2016.
DOI: 10.1007/978-3-319-47422-9_9

storage. Since the user's data is outsourced to distributed cloud servers, the service provider can easily access the data.

To prevent data from being maliciously accessed by cloud providers, data owners tend to encrypt their private data before outsourcing to the cloud, and they only share the decryption key to other authorized users. Although this method can protect the privacy of the data, it brings the data retrieve problems. This limitation has motivated many researches on advanced searchable encryption schemes that enable searching on the encrypted data while protecting the confidentiality of the data and queries.

The solution of searchable encryption that first proposed by Song et al. [3] provides a way to perform efficient keyword searches over encrypted data. Promoted by Song's pioneering work, many efforts have been devoted to construct more efficient searchable symmetric encryption (SSE) schemes, such as [4–8] and [9]. A SSE scheme allows users to encrypt their data using symmetric encryption, and then uses files and keywords to create the encrypted index for further searches. When the user wants to retrieve some files, he needs to choose a keyword and use it to generate a search request. After that, the server uses this special request to search over its internal data structure. At last, the server finds all the files related to that keyword and returns the file collection to the user. Besides performing successful searches, the privacy feature of the SSE also ensures that, given encrypted files, encrypted indexes and a series of search requests, the server cannot learn any useful information about the files and the keywords.

The solutions above are single-keyword oriented, which are inefficient in practice since the searches may return a very large number of files, such as when searching in a remote-stored email archive. The works in [10–12] and [13] extend the search primitive to the multi-keyword conjunctive search, which avoid this limitation and are more practical for real world scenarios.

To the best of our knowledge, few works consider the searchable encryption and the search authentication together. Kamara et al. [14] presented a cryptographic cloud storage system which combines an adaptive secure searchable symmetric encryption scheme with a search authenticate mechanism to allow the user to verify the integrity of the search result. They used a simple Merkle tree structure [15] and a pre-computed basis to authenticate the given dataset. Kurosawa and Ohtaki [16] introduced the definition of UC-security and proposed a verifiable SSE scheme that allows the user to detect search result's integrity.

Our Contribution. In this paper, we present a dynamic integrity preserving multi-keyword searchable encryption scheme, enabling search authentication in multi-keyword searchable encryption schemes to fulfill the practical needs. We reduce the multi-keyword search (MSE) problem to the single-keyword case by performing a search for each individual keyword and doing the intersection between each resultant file sets to get the final result. To lower the communication overhead during a search, the intersection of each keyword's search result is computed at the server side. The only thing that the user needs to do is to receive the final result and verify its integrity.

Thus, our approach should meet the following requirements: (1) the server is able to take multiple keywords as input, and give the final result directly; (2) for the server that

honestly executes the search algorithm, a valid proof can be formed and pass the verification; no one can generate a valid proof for a maliciously modified search result and still pass the verification. Theoretical basis of proposed solution is inspired by the authenticated data structure in [17] to verify set operations on out sourced sets.

We use dynamic SSE to realize the single-keyword search and use Merkle tree as the base data structure to prove the correctness of the intersection. Based on them, our scheme maintains the adaptive chosen-keyword security and is unforgeable against adaptive adversaries.

2 Definition and Security Model

2.1 Definitions

We consider the scenario that consists of two types of entities. One of them is the user that owns the data, and the other is the cloud storage provider, as known as the server, which provides storage services to the user. The dynamic MSE scheme allows a user to encrypt his data and outsource the encrypted data to the server. After uploading the encrypted data, the user only needs to store a secret key and an authenticated data state, regardless of the file number and size, i.e., the user's storage overhead is constant size. User can later generate search requests using single or multiple keywords and submit to the server. Given a search request, the server searches over the encrypted data and returns the set of encrypted files and a corresponding proof. The correctness of this search result can be verified by the user, using this result and proof. User can also dynamically update the file set on demand after the first uploading. The main system architecture is showed in Fig. 1.

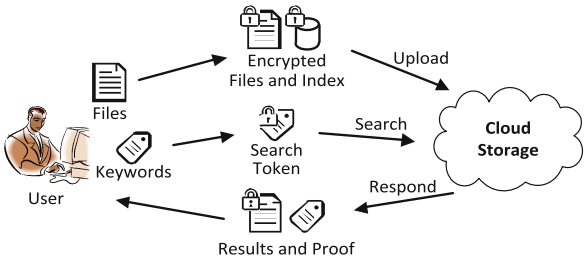

Fig. 1. Integrity preserving search over encrypted data

While using multiple keywords in a search, we define the search result to be the intersection of the sets generated by searching for each individual keyword. Concretely speaking, the question we discussed in this paper is the conjunctive keyword search. We use "token" to describe the request sent by user. Since our scheme is dynamic, there are two additional tokens, the add token and the delete token. The formal definition of our scheme is defined as follows.

Definition 1. *A dynamic MSE scheme is a tuple of eight polynomial-time algorithms and protocols* MSE = (Gen, Setup, SrchToken, Search, Verify, Dec, Add/Update, Del/Update) *such that:*

$K \leftarrow \text{Gen}(1^k)$*: is a probabilistic algorithm run by the user that takes a security parameter* 1^k *as input, outputs a secret key* K.

$(\gamma, \mathbf{c}, st, \alpha) \leftarrow \text{Setup}(K, \delta, \mathbf{f})$*: is a probabilistic algorithm run by the user that takes the secret key* K*, an index* δ *and a set of files* $\mathbf{f}$ *as input, outputs an encrypted index* γ*, a set of ciphertexts* $\mathbf{c}$*, a data state* st *and an authenticated structure* α.

$\tau_s \leftarrow \text{SrchToken}(K, W)$*: is a deterministic algorithm run by the user that takes as input the secret key* K *and a set of words* W*, outputs search token* τ_s.

$(\mathbf{I}_W, \pi) \leftarrow \text{Search}(\alpha, \gamma, \mathbf{c}, \tau_s)$*: is a deterministic algorithm run by the server that takes as input the authenticated structure* α*, the encrypted index* γ*, the set of ciphertexts* $\mathbf{c}$ *and the search token* τ_s*, outputs a set of file identifiers* $\mathbf{I}_W$*, and a proof* π.

$b \leftarrow \text{Verify}(K, st, \tau_s, \mathbf{I}', \pi)$*: is a deterministic algorithm run by the user that takes as input the secret key* K*, the data state* st*, a search token* τ_s*, a set of file identifiers* $\mathbf{I}'$ *and a proof* π*, outputs 1 as accept or 0 as reject.*

$f \leftarrow \text{Dec}(K, c)$*: is a deterministic algorithm run by the user to decrypt a ciphertext* c*, outputs a plaintext file* f.

$(U : st';\ S : \alpha', \gamma', \mathbf{c}') \leftarrow \text{Add/Update}(U : K, \delta_f, f, st;\ S : \alpha, \gamma, \mathbf{c})$*: is an interactive protocol run between the user* U *and the server* S *to* ***add*** *file to the file set.*

$(U : st';\ S : \alpha', \gamma', \mathbf{c}') \leftarrow \text{Del/Update}(U : K, \delta_f, f, st;\ S : \alpha, \gamma, \mathbf{c})$*: is an interactive protocol run between the user* U *and the server* S *to* ***delete*** *file from the file set.*

2.2 Security Model

We consider the server to be an un-trusted entity, which may deliberately steal or sabotage the user's data, or ignore some special files in the search result. Intuitively, an integrity preserving searchable encryption scheme should meet the following security features: (1) the encrypted files and data structures on the server side should not leak any information about the files to the server; (2) the search requests generated by the user should not leak any information about the keywords he uses; (3) for a fallacious result, the server cannot produce a valid proof and pass the user's verification.

Dynamic CKA2-Secure. This security requirement characterizes the feature that the scheme does not leak any information to the adversary except those defined in the leakage functions. The security definition will be parameterized by the four leakage functions $\mathcal{L}_1 \sim \mathcal{L}_4$. The adversary is allowed to be adaptive, i.e., its queries could base on the previous results. Let $\mathcal{A}$ be a stateful adversary that executes the server-side algorithm, "game" represent the interaction between $\mathcal{A}$ and user or simulator, "view" represent all the information that $\mathcal{A}$ can collect during the game. We assume that, $\mathcal{A}$ can choose the encrypted message, and then generates the queries by interacting with the user adaptively. Therefore, in our security definition, the "view" of $\mathcal{A}$ should only contain the information specified by $\mathcal{L}_1$, $\mathcal{L}_2$, $\mathcal{L}_3$ and $\mathcal{L}_4$ in a simulated way.

Definition 2. *Given the dynamic MSE scheme described in Definition 1, describe $\mathcal{A}$ as a stateful adversary, $\mathcal{S}$ as a stateful simulator, $\mathcal{L}_1$, $\mathcal{L}_2$, $\mathcal{L}_3$, $\mathcal{L}_4$ as stateful leakage functions. Consider the following games:*

$\mathrm{Real}_{\mathcal{A}}(1^k)$:
$K \leftarrow \mathrm{Gen}(1^k)$
$(\delta, \mathbf{f}) \leftarrow \mathcal{A}(1^k)$
$(\gamma, \mathbf{c}, st, \alpha) \leftarrow \mathrm{Setup}(K, \delta, \mathbf{f})$
for $1 \le i \le q$
$\{W_i, f_i, f_i'\} \xleftarrow{\text{one query each time}} \mathcal{A}(\alpha, \gamma, \mathbf{c}, \tau_1, \ldots \tau_{i-1}, c_1, \ldots c_{i-1})$
$\tau_i \xleftarrow{\mathcal{A}} \mathrm{SrchToken}(K, W_i)$, or
$(U : st'; \mathcal{A} : \tau_i, c_i) \xleftarrow{\mathcal{A}} \mathrm{Add/Update}(U : K, \delta_f, f, st; \mathcal{A})$, or
$(U : st'; \mathcal{A} : \tau_i) \xleftarrow{\mathcal{A}} \mathrm{Del/Update}(U : K, \delta_f, f, st; \mathcal{A})$
output $b \leftarrow \mathcal{A}(\alpha, \gamma, \mathbf{c}, \tau_1, \ldots, \tau_q, c_1, \ldots c_q)$

$\mathrm{Ideal}_{\mathcal{A},\mathcal{S}}(1^k)$:
$(\delta, \mathbf{f}) \leftarrow \mathcal{A}(1^k)$
$(\tilde{\alpha}, \tilde{\gamma}, \tilde{\mathbf{c}}) \leftarrow \mathcal{S}^{\mathcal{L}_1(\delta, \mathbf{f})}(1^k)$
for $1 \le i \le q$
$\{W_i, f_i, f_i'\} \xleftarrow{\text{one query each time}} \mathcal{A}(\tilde{\alpha}, \tilde{\gamma}, \tilde{\mathbf{c}}, \tilde{\tau}_1, \ldots \tilde{\tau}_{i-1}, \tilde{c}_1, \ldots \tilde{c}_{i-1})$
$\tilde{\tau}_i \xleftarrow{\mathcal{A}} \mathcal{S}^{\mathcal{L}_2(\delta, \mathbf{f}, W_i)}(1^k)$, or
$(\mathcal{S} : st'; \mathcal{A} : \tilde{\tau}_i, \tilde{c}_i) \xleftarrow{\mathcal{A}} \mathrm{Add/Update}(\mathcal{S}^{\mathcal{L}_3(\delta, \mathbf{f}, f_i)}(1^k); \mathcal{A})$, or
$(\mathcal{S} : st'; \mathcal{A} : \tilde{\tau}_i) \xleftarrow{\mathcal{A}} \mathrm{Del/Update}(\mathcal{S}^{\mathcal{L}_4(\delta, \mathbf{f}, f_i')}(1^k); \mathcal{A})$
output $b \leftarrow \mathcal{A}(\tilde{\alpha}, \tilde{\gamma}, \tilde{\mathbf{c}}, \tilde{\tau}_1, \ldots, \tilde{\tau}_q, \tilde{c}_1, \ldots \tilde{c}_q)$

The dynamic MSE scheme is $(\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3, \mathcal{L}_4)$-secure against adaptive dynamic chosen-keyword attacks if for all PPT adversary $\mathcal{A}$, there exist a probabilistic polynomial time simulator $\mathcal{S}$ such that:

$$\left|\mathbf{Pr}\left[\mathrm{Real}_{\mathcal{A}}(1^k) = 1\right] - \mathbf{Pr}\left[\mathrm{Ideal}_{\mathcal{A},\mathcal{S}}(1^k) = 1\right]\right| \le \mathbf{negl}(1^k),$$

where $\mathbf{negl}(1^k)$ is a negligible function with input 1^k.

Unforgeability. We use game $\mathrm{Forge}_{\mathcal{A}}(1^k)$ to describe our scheme's unforgeability. In the unforgeability game, the adversary interacts with a user that honestly executes the scheme. User initializes his data structures using the data provided by the adversary. After making polynomial times queries, the adversary produces a set of keywords, a wrong search result and a proof to this result. If these outputs pass the user's verification algorithm, the game outputs 1, otherwise it outputs 0. The unforgeability requires that, all PPT adversaries have at most negligible probability to let the game output 1. We give the formal definition as follow.

Definition 3. *Given the dynamic MSE scheme described in Definition 1, for a stateful adversary $\mathcal{A}$, consider the following game:*

$$
\begin{aligned}
&\text{Forge}_{\mathcal{A}}(1^k):\\
&\quad K \leftarrow \text{Gen}(1^k)\\
&\quad (\delta, \mathbf{f}) \leftarrow \mathcal{A}(1^k)\\
&\quad (\gamma, \mathbf{c}, st, \alpha) \leftarrow \text{Setup}(K, \delta, \mathbf{f})\\
&\quad \text{for } 1 \le i \le q\\
&\qquad \{W_i, f_i, f_i'\} \xleftarrow{\text{one query each time}} \mathcal{A}(\alpha, \gamma, \mathbf{c}, \tau_1, \ldots \tau_{i-1}, c_1, \ldots c_{i-1})\\
&\qquad \tau_i \xleftarrow{\mathcal{A}} \text{SrchToken}(K, W_i), \text{ or}\\
&\qquad (\tau_i, c_i) \xleftarrow{\mathcal{A}} \text{Add/Update}(U : K, \delta_{f_i}, f_i, st; \mathcal{A}), \text{ or}\\
&\qquad \tau_i \xleftarrow{\mathcal{A}} \text{Del/Update}(U : K, \delta_{f_i'}, f_i', st; \mathcal{A})\\
&\quad (W, \mathbf{I}', \pi) \leftarrow \mathcal{A}(\alpha, \gamma, \mathbf{c}, \tau_1, \ldots, \tau_q, c_1, \ldots c_q)\\
&\quad \tau_s \leftarrow \text{SrchToken}(K, W)\\
&\quad \text{output } b \leftarrow \text{Verify}(K, st', \tau_s, \mathbf{I}', \pi)
\end{aligned}
$$

where the set $\mathbf{I}' \neq \mathbf{I}_W$. *We say the dynamic MSE scheme is unforgeable if for all PPT adversary* $\mathcal{A}$, *the probability:* $\mathbf{Pr}[\text{Forge}_{\mathcal{A}}(1^k) = 1] \le \mathbf{negl}(1^k)$, *where* $\mathbf{negl}(1^k)$ *is a negligible function with input* 1^k.

3 Integrity Preserving Multi-keyword Searchable Encryption Scheme

In this section, we first construct a multi-keyword searchable encryption scheme, and then add the search authentication mechanism to it to make the search result's integrity verifiable.

In our construction, the set of files $\mathbf{f}$ along with the inverted indexes δ are the initial input. In contrast to the file index, an inverted index is a set of lists that lead by keywords, and each keyword is followed by a set of files that contain that keyword. The keywords of each file are pre-selected, and can be considered as the outputs of some other algorithms, which won't be discussed here.

3.1 Dynamic Searchable Encryption

In the literature, most searchable encryption schemes use symmetric encryption to improve performance. We follow the prior constructions and build our scheme upon the CPA secure private key encryption [18].

The Fig. 2 shows our dynamic searchable encryption structure that is constructed based on the inverted index. Generally speaking, the lookup table contains all the keywords in the system, and each keyword in the table leads a list that stored in the search array. For example, the list of keyword w_2 starts at address 4 in the array, and the node at address 4 has a pointer that points to address 7, and then address 8. By traversing this list, all files that contain the keyword w_2 can be retrieved. All the nodes are stored at random location in the search array. To support efficient file updating,

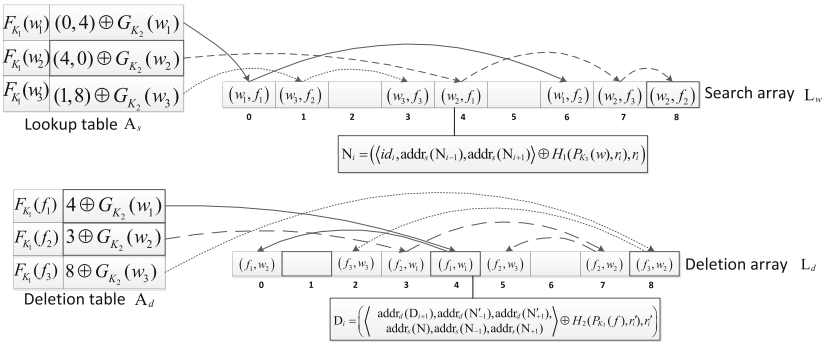

Fig. 2. The schematic search structure

there are also a deletion table and a deletion array. They work the same way, except those lists are led by files.

In order to prevent the server from learning the data, all the tables' entry, all the pointers in the table, and all the nodes in those arrays are encrypted. During a search, given the encrypted keywords, the server first decrypts the pointers in the lookup table and then uses the pointers to find the corresponding file identifiers in the search array. Those keywords remain encrypted throughout the search. Even if the server has searched all those keywords, it can only learn the relationship between the encrypted keywords and the related file identifiers but cannot obtain any useful knowledge about the keywords itself. This could prevent the curious server from learning the files and keywords. A more detailed analysis about this security model will be presented in Sect. 4.

3.2 Making Result Verifiable

In the following content, we discuss the method to make result verifiable. This method can allow a server to prove to a client that it answered a multi-keyword search query correctly.

The method proposed in [14] is a Merkle tree based solution that it computes the accumulated value for each word *w*, and uses these values as leaves to construct the tree. In a search, the server returns a file set *S*, and a Merkle tree proof to this set. The user can compute his own accumulated value using the files in *S*, and use it to perform the Merkle tree verification. If the newly computed root equals to the original one, then the result is correct and can be accepted by the user.

However, while switching to the multiple keywords setting, this solution is obsolete to prove the correctness of the intersection of the results. The server could only generate the proof for each set separately. These sets and proofs must be transferred to the user side to be verified, and subsequently the intersection of these sets could be computed by the user. Obviously, the communication complexity is linear and may have performance problems when the sets are very large.

The reasonable way to address this problem is to let the server compute the intersection, and give the user final result directly. In this case, the correctness of the intersection operation should be proved. We use the bilinear-map accumulator to realize this functionality. The bilinear-map accumulator [19] is an efficient tool to provide proofs of membership for elements that belong to a set. Let $s \in \mathbb{Z}_p^*$ be a randomly chosen trapdoor. The accumulator accumulates elements in $\mathbb{Z}_p$, and outputs an element in $\mathbb{G}$. For a set of elements $\mathcal{X}$ in $\mathbb{Z}_p$, the accumulation value $\text{acc}(\mathcal{X})$ is defined as:

$$\text{acc}(\mathcal{X}) = g^{\prod_{x \in \mathcal{X}} (x+s)} \tag{1}$$

Without knowing the trapdoor s, the value $\text{acc}(\mathcal{X})$ can also be constructed using $\mathcal{X}$ and the pre-computed $(g, g^s, \ldots, g^{s^q})$, where $q \geq \#\mathcal{X}$. The proof of subset containment of a set $\mathcal{S} \subseteq \mathcal{X}$ is the witness $(\mathcal{S}, \mathcal{W}_{\mathcal{S},\mathcal{X}})$ where:

$$\mathcal{W}_{\mathcal{S},\mathcal{X}} = g^{\prod_{x \in \mathcal{X}-\mathcal{S}} (x+s)} \tag{2}$$

Subset containment of $\mathcal{S}$ in $\mathcal{X}$ can be verified by checking:

$$e\left(\mathcal{W}_{\mathcal{S},\mathcal{X}}, g^{\prod_{x \in \mathcal{S}} (x+s)}\right) = e(\text{acc}(\mathcal{X}), g) \tag{3}$$

The security of the bilinear-map accumulator relies on the bilinear q-Strong Diffie-Hellman assumption.

Intuitively, the correctness of the intersection could be defined as follows: given a set $\mathbf{I}$ and a series of sets $S_1, \ldots, S_n$, $\mathbf{I}$ is the correct intersection of $S_1, \ldots, S_n$ if and only if the following conditions hold:

1. The subset condition: $(\mathbf{I} \subseteq S_1) \wedge \cdots \wedge (\mathbf{I} \subseteq S_n)$.
2. The completeness condition: $(S_1 - \mathbf{I}) \cap \cdots \cap (S_n - \mathbf{I}) = \emptyset$.

The subset condition is easy to understand, because as the intersection, the set $\mathbf{I}$ must be included in each set S_i. We use Merkle tree to authenticate the value $\text{acc}(S_i)$. For all $w \in \mathbf{w}$, the values $\text{acc}(\mathbf{f}_w)$ are computed according to (1), then the tree is constructed using these values as leaves.

Since the user does not store those accumulated values, the server should first generates Merkle tree proofs for each $\text{acc}(S_i)$. It's then straight forward to produce the subset witness $(\mathbf{I}, \mathcal{W}_{\mathbf{I},S_i})$ in (2) for each set S_i. Given the $\text{acc}(S_i)$ and the witness $(\mathbf{I}, \mathcal{W}_{\mathbf{I},S_i})$, the validity of the value $\text{acc}(S_i)$ should be first verified using Merkle tree proofs, then the subset containment relationship could be checked by performing the verifications according to the equation in (3).

The completeness condition is also necessary since the set $\mathbf{I}$ must contain all the common elements. To construct the completeness proof, we define the polynomial:

$$P_i(s) = \prod_{f \in S_i - \mathbf{I}} (s + id(f))$$

The following result is based on the extended Euclidean algorithm over polynomials and provides verification for checking the completeness of set intersection.

Lemma 1. *The set* $\mathbf{I}$ *is complete if and only if there exist polynomials* $q_1(s), \ldots, q_n(s)$ *such that* $q_1(s)P_1(s) + \cdots + q_n(s)P_n(s) = 1$ *where* $P_i(s)$ *is defined above. Suppose* $\mathbf{I}$ *is not the complete set, then there exist at least one common factor in* $P_1(s), \ldots, P_n(s)$. *Thus there are no polynomials* $q_1(s), \ldots, q_n(s)$ *to satisfy* $q_1(s)P_1(s) + \cdots + q_n(s)P_n(s) = 1$.

The formal analysis will be given in Sect. 4.

3.3 Explicit Construction

Let $\Pi = (\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ be a private-key encryption system. $F : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^k$, $G : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^*$, $P : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^k$ be pseudo-random functions. Let $H_1 : \{0,1\}^* \to \{0,1\}^*$, $H_2 : \{0,1\}^* \to \{0,1\}^*$ and $H_3 : \{0,1\}^* \to \{0,1\}^k$ be collision-resistant hash functions. Let $z \in \mathbb{N}$ be the initial size of the free list, and $\mathbf{0}$ be a series of 0's. Choose bilinear pairing parameters $(p, \mathbb{G}, \mathcal{G}, e, g)$.

Gen(1^k) : Randomly choose three k - bit strings K_1, K_2, K_3 and generate $K_4 \leftarrow \Pi.\mathrm{Gen}(1^k)$. Choose $s \in \mathbb{Z}_p^*$ at random and output $K = (K_1, K_2, K_3, K_4, s)$ as the private keys. Compute $(g, g^s, g^{s^2}, \ldots, g^{s^q})$ as public parameters where q should be large enough, i.e., should at least satisfy $q \geq \max\{\#\mathbf{f}_w\}_{w \in \mathbf{w}}$.

Setup(K, δ, f) :

1. Let A_s and A_d be arrays of size $|\mathbf{c}|/8 + z$ and let T_s and T_d be dictionaries of size $\#\mathbf{w}$ and $\#\mathbf{f}$, respectively. Use "free" to represent a k - length word not in $\mathbf{w}$.The following step 2 and step 3 should be performed synchronously to set up A_s and A_d at the same time.
2. For every keyword $w \in \mathbf{w}$,
   - Generate a list L_w of $\#\mathbf{f}_w$ nodes $(\mathrm{N}_1, \ldots, \mathrm{N}_{\#\mathbf{f}_w})$ randomly stored in A_s, set $\mathrm{N}_i = (\langle id_i, \mathrm{addr}_s(\mathrm{N}_{i-1}), \mathrm{addr}_s(\mathrm{N}_{i+1})\rangle \oplus H_1(P_{K_3}(w), r_i), r_i)$, where id_i is the identity of the ith file in $\mathbf{f}_w$, r_i is a k - bit random string, and $\mathrm{addr}_s(\mathrm{N}_{\#\mathbf{f}_w+1}) = \mathrm{addr}_s(\mathrm{N}_0) = \mathbf{0}^{\log \#\mathrm{A}_s}$
   - Set $\mathrm{T}_s[F_{K_1}(w)] = \langle \mathrm{addr}_s(\mathrm{N}_1), \mathrm{addr}_d(\mathrm{N}_1')\rangle \oplus G_{K_2}(w)$, where N_1' is the dual of N_1, which has the same (f_1, w) pair as node N_1.
3. For each file f in $\mathbf{f}$,
   - Create a list L_f of $\#f$ dual nodes $(\mathrm{D}_1, \ldots, \mathrm{D}_{\#f})$ $(\mathrm{N}_1, \ldots, \mathrm{N}_{\#\mathbf{f}_w})$ randomly stored in the deletion array A_d. Each node D_i is associated with a word w, and a corresponding node N in L_w. Let N_{+1} be the node after N in L_w, and N_{-1} be the node before N in L_w. Define D_i as:

$$\mathrm{D}_i = \left(\left\langle \begin{array}{c} \mathrm{addr}_d(\mathrm{D}_{i+1}), \mathrm{addr}_d(\mathrm{N}'_{-1}), \mathrm{addr}_d(\mathrm{N}'_{+1}), \\ \mathrm{addr}_s(\mathrm{N}), \mathrm{addr}_s(\mathrm{N}_{-1}), \mathrm{addr}_s(\mathrm{N}_{+1}) \end{array} \right\rangle \oplus H_2(P_{K_3}(f), r'_i), r'_i \right),$$

where r'_i is a random k - bit string, $\mathrm{addr}_d(\mathrm{D}_{\#f+1}) = \mathbf{0}^{\log \#\mathrm{A}_d}$

- Store a pointer to the first node of L_f in the deletion table by setting:

$$\mathrm{T}_d[F_{K_1}(f)] = \mathrm{addr}_d(\mathrm{D}_1) \oplus G_{K_2}(f)$$

4. Generate the free list $\mathrm{L}_{\mathrm{free}}$ by choosing z at random in A_s and in A_d. Let $(\mathrm{F}_1, \ldots, \mathrm{F}_z)$ and $(\mathrm{F}'_1, \ldots, \mathrm{F}'_z)$ be the free nodes in A_s and A_d, respectively. Set: $\mathrm{T}_s[\mathrm{free}] = \left\langle \mathrm{addr}_s(\mathrm{F}_1), \mathbf{0}^{\log \#\mathrm{A}_s} \right\rangle$, and for $1 \le i \le z$, set $\mathrm{A}_s[\mathrm{addr}_s(\mathrm{F}_i)] = \left\langle \mathbf{0}^{\log \#\mathbf{f}}, \mathrm{addr}_s(\mathrm{F}_{i+1}), \mathrm{addr}_d(\mathrm{F}'_i), \mathbf{0}^k \right\rangle$, where $\mathrm{addr}_s(\mathrm{F}_{z+1}) = \mathbf{0}^{\log \#\mathrm{A}_s}$.
5. Fill the remaining entries of A_s and A_d with random strings.
6. For $1 \le i \le \#\mathbf{f}$, let $c_i \leftarrow \Pi.\mathrm{Enc}_{K_4}(f_i)$.
7. For all $w \in \mathbf{w}$, form the leaf node by letting $\theta_w = \left\langle F_{K_1}(w), g^{\prod_{f \in \mathbf{f}_w} (s + id(f))} \right\rangle$. Construct a Merkle tree using H_3 with leaves $\mathcal{L} = \{\theta_w\}_{w \in \mathbf{w}}$ permuted in a random order.
8. Output $(\gamma, \mathbf{c}, st, \alpha)$, where $\gamma = (\mathrm{A}_s, \mathrm{T}_s, \mathrm{A}_d, \mathrm{T}_d)$, $\mathbf{c} = (c_1, \ldots, c_{\#\mathbf{f}})$, st is the root of the tree, and α is the tree itself.

SrchToken(K, W): For $W = (w_1, \ldots, w_n)$, compute each $\tau_i = (F_{K_1}(w_i), G_{K_2}(w_i), P_{K_3}(w_i))$, then output $\tau_s = (\tau_1, \ldots \tau_n)$.

Search$(\alpha, \gamma, \mathbf{c}, \tau_s)$:

1. For each τ_i in τ_s, parse τ_i as $(\tau_{i,1}, \tau_{i,2}, \tau_{i,3})$,
   - Recover a pointer to the first node of the list by computing $(\alpha_1, \alpha'_1) = \mathrm{T}_s[\tau_{i,1}] \oplus \tau_{i,2}$.
   - Lookup node $\mathrm{N}_1 = \mathrm{A}[\alpha_1]$ and decrypt it using $\tau_{i,3}$, i.e., parse N_1 as (v_1, r_1) and compute $(id_1, \mathbf{0}, \mathrm{addr}_s(\mathrm{N}_2)) = v_1 \oplus H_1(\tau_{i,3}, r_1)$. Let $\alpha_2 = \mathrm{addr}_s(\mathrm{N}_2)$.
   - For $j \ge 2$, decrypt node N_j as above until $\alpha_{j+1} = \mathbf{0}$.
   - Let $S_i = \{id_1, \ldots, id_t\}$ be the file identifiers revealed in the previous steps.
2. For the sets $S_1, \cdots, S_n$ generated in step 1, let $\mathbf{I}_W = \{id_1, \ldots, id_m\}$ be the intersection, i.e., $\mathbf{I}_W = S_1 \cap S_2 \cap \ldots \cap S_n$. Compute the proofs in the following steps:
   - For $1 \le i \le n$, find the leaf θ_i in α whose first element is $\tau_{i,1}$ and generate the proof t_i. The t_i includes θ_i and all the sibling nodes in the path from the leaf θ_i to the root. Let $\mathcal{T} = \{t_1, \ldots, t_n\}$.
   - For $1 \le i \le n$, form the polynomial: $P_i = \prod_{f \in S_i - \mathbf{I}_W} (s + id(f))$, then use the public parameters $(g, g^s, g^{s^2}, \ldots, g^{s^q})$ to compute the value g^{P_i}. Let $\mathcal{S} = \{g^{P_1}, \ldots, g^{P_n}\}$ be the subset witness.

- Giving the polynomials $\{P_1, \ldots, P_n\}$ generated in step 2, find the polynomials $\{q_1, \ldots, q_n\}$ that satisfying $q_1P_1 + q_2P_2 + \cdots + q_nP_n = 1$. This can be done using extended Euclidean algorithm over polynomials. Let $\mathcal{C} = \{g^{q_1}, \ldots, g^{q_n}\}$ be the completeness witness.

3. Output the result $\mathbf{I}_W$ and the proof $\pi = \{\mathcal{T}, \mathcal{S}, \mathcal{C}\}$.

Verify$(K, st, \tau_s, \mathbf{I}', \pi)$:

1. Parse π as $\{\mathcal{T}, \mathcal{S}, \mathcal{C}\}$ and verify these proofs in the following steps:

- For each proof t_i in $\mathcal{T}$, let θ_i be the corresponding leaf node in t_i. Parse θ_i as $(\theta_{i,1}, \theta_{i,2})$, i.e. $\theta_{i,1} = F_{K_1}(w_i)$ and $\theta_{i,2} = g^{\prod_{f \in \mathbf{f}_{w_i}} (s + id(f))}$.Verify if the value $\theta_{i,1}$ equals to $\tau_{i,1}$, where $\tau_{i,1}$ is the first element of τ_i in τ_s. Then verify the proof t_i using the root st.
- For $1 \le i \le n$, parse the leaf node θ_i as $(\theta_{i,1}, \theta_{i,2})$, then perform the subset condition verification by checking: $e(g^{\prod_{k=1}^{m} (s + id_k)}, g^{P_i}) \stackrel{?}{=} e(\theta_{i,2}, g)$, where $(id_1, \ldots id_m)$ is from $\mathbf{I}'$ and g^{P_i} is element in $\mathcal{S}$.
- Verify the completeness condition by checking: $\prod_{i=1}^{n} e(g^{P_i}, g^{q_i}) \stackrel{?}{=} e(g, g)$, where g^{P_i} is element in $\mathcal{S}$ and g^{q_i} is the corresponding element in $\mathcal{C}$.

2. If all the verifications succeed, then output 1, otherwise output 0.

Dec(K, c): Output $f = \Pi.\mathrm{Dec}_{K_4}(c)$.
Add/**Update**$(U : K, \delta_f, f, st;\ S : \alpha, \gamma, \mathbf{c})$:
User:
Recover the unique sequence of words $(w_1, \ldots, w_{\#f})$ from δ_f and compute the set $\{F_{K_1}(w_i)\}_{1 \le i \le \#f}$ and send to the server.
Server:

1. For $1 \le i \le \#f$, traverse the Merkel tree α and:

- Find the leaf θ_i in α whose first element is $F_{K_1}(w_i)$.
- Let t_i be the proof in α from θ_i to the root. The proof includes the leaf θ_i, and all the sibling nodes from θ_i to the root.

2. Let $\rho = (t_1, \ldots, t_{\#f})$ and send it to the user.

User:

1. Verify the proofs in $(t_1, \ldots, t_{\#f})$ using st, if fails, output $\perp$ and terminate.
2. For $1 \le i \le \#f$,

- Let θ_i be the leaf in t_i, parse θ_i as $(\theta_{i,1}, \theta_{i,2})$.
- Compute the new leaf node $\theta'_i = (\theta_{i,1}, (\theta_{i,2})^{s + id(f)})$.

3. Update the root hash st using $(\theta'_1, \ldots, \theta'_{\#f})$ and the information in $(t_1, \ldots, t_{\#f})$.
4. Compute $\tau_a = (F_{K_1}(f), G_{K_2}(f), \lambda_1, \ldots \lambda_{\#f})$, where for all $1 \le i \le \#f$ ::

$$\lambda_i = \begin{pmatrix} \theta'_{i,1}, \theta'_{i,2}, G_{K_2}(w_i), \langle id(f), \mathbf{0}, \mathbf{0}, \rangle \oplus H_1(P_{K_3}(W_i), r_i), \\ r_i, \langle \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0} \rangle \oplus H_2\big(P_{K_3}(f), r'_i\big), r'_i \end{pmatrix},$$

where r_i and r'_i are random k - bit strings.

5. Let $c_f \leftarrow \mathrm{SKE.Enc}_{K_4}(f)$ and send (τ_a, c_f) to the server, then output the new root st'.

Server:

1. Parse τ_a as $(\tau_1, \tau_2, \lambda_1, \ldots, \lambda_{\#f})$ and return $\perp$ if τ_1 is already in T_d.
2. For $1 \le i \le \#f$,

   - Find the first free location φ in A_s, second free location φ_{+1} in A_s, first free location φ' in A_d, and second free location φ'_{+1} in A_d, by computing $(\varphi, \mathbf{0}) = \mathrm{T}_s[\mathrm{free}]$, $(\mathbf{0}, \varphi_{+1}, \varphi') = \mathrm{A}_s[\varphi]$ and $(\mathbf{0}, \varphi_{+2}, \varphi'_{+1}) = \mathrm{A}_s[\varphi_{+1}]$.
   - Update the search table by setting $\mathrm{T}_s[\mathrm{free}] = (\varphi_{+1}, \mathbf{0})$.
   - Recover N_1's address α_1 by computing $(\alpha_1, \alpha'_1) = \mathrm{T}_s[\lambda_i[1]] \oplus \lambda_i[3]$.
   - Parse $\mathrm{N}_1 = \mathrm{A}_s[\alpha_1]$ as (v_1, r_1), then update N_1's back pointer point by setting: $\mathrm{A}_s[\alpha_1] = (v_1 \oplus \langle \mathbf{0}, \varphi, \mathbf{0} \rangle, r_1)$.
   - Store the new node at location φ and modify its forward pointer to N_1 by setting:$\mathrm{A}_s[\varphi] = (\lambda_i[4] \oplus \langle \mathbf{0}, \mathbf{0}, \alpha_1 \rangle, \lambda_i[5])$.
   - Update the search table by setting:$\mathrm{T}_s[\lambda_i[1]] = (\varphi, \varphi') \oplus \lambda_i[3]$.
   - Parse $\mathrm{D}_1 = \mathrm{A}_d[\alpha'_1]$ as (v'_1, r'_1), set $\mathrm{A}_d[\alpha'_1] = (v'_1 \oplus \langle \mathbf{0}, \varphi', \mathbf{0}, \mathbf{0}, \varphi, \mathbf{0} \rangle, r'_1)$.
   - If $i < \#f$, set $\mathrm{A}_d[\varphi'] = \big(\lambda_i[6] \oplus \langle \varphi'_{+1}, \mathbf{0}, \alpha'_1, \varphi, \mathbf{0}, \alpha_1 \rangle, \lambda_i[7]\big)$.
   - If $i = \#f$, set $\mathrm{A}_d[\varphi'] = \big(\lambda_i[6] \oplus \langle \mathbf{0}, \mathbf{0}, \alpha'_1, \varphi, \mathbf{0}, \alpha_1 \rangle, \lambda_i[7]\big)$.
   - If $i = 1$, then update the deletion table by setting $\mathrm{T}_d[\tau_1] = \varphi' \oplus \tau_2$.

3. Update the cipher texts by adding c to $\mathbf{c}$.
4. Let $\theta'_i = (\lambda_i[1], \lambda_i[2])$, update the tree α by replacing the leaves $(\theta_1, \ldots, \theta_{\#f})$ with $(\theta'_1, \ldots, \theta'_{\#f})$.
5. Output $(\alpha', \gamma', \mathbf{c}')$, where α' is the updated tree.

Del/Update$(U : K, \delta_f, f, st;\ S : \alpha, \gamma, \mathbf{c})$:

User:

Recover the unique sequence of words $(w_1, \ldots, w_{\#f})$ from δ_f and compute the set $\{F_{K_1}(w_i)\}_{1 \le i \le \#f}$ and send to the server.

Server:

1. For $1 \le i \le \#f$, traverse the Merkel tree α and:

   - Find the leaf θ_i in α whose first element is $F_{K_1}(w_i)$.
   - Let t_i be the proof in α from θ_i to the root. The proof includes the leaf θ_i, and all the sibling nodes from θ_i to the root.

2. Let $\rho = (t_1, \ldots, t_{\#f})$ and send it to the user.

User:

1. Verify the proofs in $(t_1, \ldots, t_{\#f})$ using st, if fails, output $\perp$ and terminate.
2. For $1 \le i \le \#f$,

- Let θ_i be the leaf in t_i, parse θ_i as $(\theta_{i,1}, \theta_{i,2})$.
- Compute the new leaf node $\theta'_i = (\theta_{i,1}, (\theta_{i,2})^{1/(s+id(f))})$.

3. Update the root hash st using $(\theta'_1, \ldots, \theta'_{\#f})$ and the information in $(t_1, \ldots, t_{\#f})$.
4. Compute $\tau_d = (F_{K_1}(f), G_{K_2}(f), P_{K_3}(f), id(f), \theta'_1, \ldots, \theta'_{\#f})$.
5. Send τ_d to the server, then output the new root st'.

Server:

1. Parse τ_d as $(\tau_1, \tau_2, \tau_3, id, \theta'_1, \ldots, \theta'_{\#f})$.
2. Find the first node of L_f by computing $\alpha'_i = \mathrm{T}_d[\tau_1] \oplus \tau_2$.
3. While $\alpha'_i \neq \mathbf{0}$,

- Parse $\mathrm{D}_i = \mathrm{A}_d[\alpha'_i]$ as (v'_i, r'_i), decrypt D_i by computing $(\alpha_1, \ldots, \alpha_6) = v'_i \oplus H_2(\tau_3, r'_i)$.
- Delete D_i by setting $\mathrm{A}_d[\alpha'_i]$ to a random string.
- Find address of the first free node by computing $(\varphi, \mathbf{0}) = \mathrm{T}_s[\text{free}]$.
- Update the first node of the free list in the T_s point to D_i's dual by setting $\mathrm{T}_s[\text{free}] = (\alpha_4, \mathbf{0})$.
- Free D_i's dual by setting $\mathrm{A}_s[\alpha_4] = (\mathbf{0}, \varphi, \alpha'_i)$.
- Let N_{-1} be the node before D_i's dual. Update N_{-1}'s next pointer by setting $\mathrm{A}_s[\alpha_5] = (\beta_1, \beta_2, \beta_3 \oplus \alpha_4 \oplus \alpha_6, r_{-1})$, where $(\beta_1, \beta_2, \beta_3, r_{-1}) = \mathrm{A}_s[\alpha_5]$. Also, update the pointers of N_{-1}'s dual by setting:$\mathrm{A}_d[\alpha_2] = (\beta_1, \beta_2, \beta_3 \oplus \alpha'_i \oplus \alpha_3, \beta_4, \beta_5, \beta_6 \oplus \alpha_4 \oplus \alpha_6, r'_{-1})$,where $(\beta_1, \ldots, \beta_6, r'_{-1}) = \mathrm{A}_d[\alpha_2]$.
- Let N_{+1} be the node after D_i's dual. Update N_{+1}'s previous pointer by setting $\mathrm{A}_s[\alpha_6] = (\beta_1, \beta_2 \oplus \alpha_4 \oplus \alpha_5, \beta_3, r_{+1})$, where $(\beta_1, \beta_2, \beta_3, r_{+1}) = \mathrm{A}_s[\alpha_6]$. Also, update N_{+1}'s dual's pointers by setting: $\mathrm{A}_d[\alpha_3] = (\beta_1, \beta_2 \oplus \alpha'_i \oplus \alpha_2, \beta_3, \beta_4, \beta_5 \oplus \alpha_4 \oplus \alpha_5, \beta_6, r'_{+1})$, where $(\beta_1, \ldots, \beta_6, r'_{+1}) = \mathrm{A}_d[\alpha_3]$.
- Set $\alpha'_i = \alpha_1$.

4. Remove the cipher text corresponding to id from $\mathbf{c}$.
5. Remove τ_1 from T_d.
6. Update the tree α by replacing the leaves $(\theta_1, \ldots, \theta_{\#f})$ with $(\theta'_1, \ldots, \theta'_{\#f})$.
7. Output $(\alpha', \gamma', \mathbf{c}')$, where α' is the updated tree.

4 Security Analysis

4.1 Dynamic CKA2-Secure

In the following, we analyze our dynamic MSE scheme and investigate which information has been leaked during the execution of these algorithms and protocols. The formal definition will be given afterwards.

In our scheme, for each word w_i, the value $F_{K_1}(w_i)$ can be treated as a unique identifier, and we denote it by $id(w_i)$. For each file f_i, there are two identifiers, the $id(f_i)$ in the array A_s and the $F_{K_1}(f_i)$ in the table T_d. Both of them can uniquely represent a file, so for convenience, we do not distinguish between them.

Given the encrypted index $\gamma = (\mathrm{T}_s, \mathrm{A}_s, \mathrm{T}_d, \mathrm{A}_d)$, the Merkle tree α and the ciphertexts $\mathbf{c}$, the server can learn the size of A_s, the set $[id(w)]_{w\in\mathbf{w}}$ from T_s, the set $[id(f)]_{f\in\mathbf{f}}$ and the length of each file $[|f|]_{f\in\mathbf{f}}$. We denote these by $\mathcal{L}_1$, i.e.,

$$\mathcal{L}_1(\delta, \mathbf{f}) = \Big(\#\mathrm{A}_s, [id(w)]_{w\in\mathbf{w}}, [id(f)]_{f\in\mathbf{f}}, [|f|]_{f\in\mathbf{f}}\Big).$$

The search operation reveals to the server $id(w)$ for all $w \in W$, and the relationship between $id(w)$ and the identifiers of all files that contains w. We denote these by $\mathcal{L}_2$, i.e.,

$$\mathcal{L}_2(\delta, \mathbf{f}, W) = \Big([id(f)]_{f\in\mathbf{f}_w}, id(w)\Big)_{\text{for all}w\in W}.$$

In the add protocol, the server can learn the identifier of the file to be added, the length of the file, and the identifiers of the words that belong to the file. In addition, it can tell whether the word w contained in the file is a new word by checking the table T_s. We denote these by $\mathcal{L}_3$, i.e.,

$$\mathcal{L}_3(\delta, \mathbf{f}, f) = \Big(id(f), [id(w), \mathrm{apprs}(w)]_{w\in\mathbf{w}_f}, |f|\Big),$$

where $\mathrm{apprs}(w)$ is a one bit flag set to 1 if the word w exists in the index before the file f is added, otherwise, it is set to 0.

Similarly, in the delete protocol, the server can learn the identifier of the file to be deleted, and know the relationship between $id(f)$ and those word identifiers. In addition, for each $w \in \mathbf{w}_f$, by removing the word pair (f, w) from the list L_w, the server learns the locations of the pair's neighbors in L_w. We denote these by $\mathcal{L}_4$.

$$\mathcal{L}_4(\delta, \mathbf{f}, f) = \Big(id(f), [id(w), \mathrm{prev}(f, w), \mathrm{next}(f, w)]_{w\in\mathbf{w}_f}\Big),$$

where $\mathrm{prev}(f, w)$ and $\mathrm{next}(f, w)$ are the file identifiers of the file before and after f in the word list L_w. For the head and the tail of the list, the corresponding value is set $\perp$ to indicate that there are no more nodes before or after this one.

Now we use the following theorem to claim that the construction in Sect. 4 is dynamic CKA2-secure in the random oracle model with the leakage functions described above.

Theorem 1. *If the private-key encryption system Π is CPA-secure, and the F, G and P are pseudo-random functions, then the dynamic MSE scheme is $(\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3, \mathcal{L}_4)$-secure against adaptive chosen-keyword attacks in the random oracle model.*

Proof: The primary goal of providing this proof is to construct a PPT simulator $\mathcal{S}$ that can generate the simulated values in the ideal game using the information given in these leakage functions. Those simulated values should be indistinguishable from ones in the real game to any PPT adversary.

Given the information received from $\mathcal{L}_1$, the simulator could determine the length and the structure of encrypted index γ, ciphertexts $\mathbf{c}$ and tree α. Then it can use

randomly chosen strings to construct these structures and produce these values as the simulated one $(\tilde{\gamma}, \tilde{\mathbf{c}}, \tilde{\alpha})$. If a PPT adversary can distinguish the tuple $(\tilde{\gamma}, \tilde{\mathbf{c}}, \tilde{\alpha})$ from $(\gamma, \mathbf{c}, \alpha)$ with non-negligible probability then it can break at least one of these properties with non-negligible probability: the CPA security of the encryption scheme; the pseudo-randomness of the PRFs and the elliptic curve discrete logarithm assumption.

Given the information received from $\mathcal{L}_2$, $\mathcal{L}_3$ and $\mathcal{L}_4$, the simulator should respond the simulated search token, the simulated add token and the simulated delete token during the adversary's queries. These steps become more complex due to the fact that simulator needs to track the dependencies between the information revealed by these queries to ensure consistency among these simulated tokens. We define additional assisting structures iA_s, iA_d, iT_s and iT_d in the simulator side to maintain consistency during updation. The simulator uses these assisted structures to record those dependencies that are revealed by $\mathcal{L}_2$, $\mathcal{L}_3$ and $\mathcal{L}_4$ in the queries, and builds internal relationship in iA_s, iA_d, iT_s and iT_d, while the values in $\tilde{\gamma} = (\tilde{\mathrm{A}}_s, \tilde{\mathrm{T}}_s, \tilde{\mathrm{A}}_d, \tilde{\mathrm{T}}_d)$ remain random. This gives the simulator the ability to respond the adversary's queries like a real user, except using those simulated values.

Analyze.

1. If the pseudo-randomness of F, G and P holds, then for all PPT adversaries $\mathcal{A}$, there exist negligible value ε_1 such that:

$$\left|\mathbf{Pr}[1 \leftarrow \mathcal{A}(\delta, \mathbf{f}, \mathrm{A}_s, \mathrm{T}_s, \mathrm{A}_d, \mathrm{T}_d)] - \mathbf{Pr}\left[1 \leftarrow \mathcal{A}\left(\delta, \mathbf{f}, \tilde{\mathrm{A}}_s, \tilde{\mathrm{T}}_s, \tilde{\mathrm{A}}_d, \tilde{\mathrm{T}}_d\right)\right]\right| \leq \varepsilon_1.$$

Because each cell in $\tilde{\mathrm{A}}_s$ can be recognized as the form $\langle \tilde{\mathrm{N}}, \tilde{r} \rangle$ where $\left|\tilde{\mathrm{N}}\right| = 2\log \#\mathrm{A}_s + \log \#\mathbf{f}$ and $|\tilde{r}| = k$. The cell in A_s is $\mathrm{N}_i = (\langle id_i, \mathrm{addr}_s(\mathrm{N}_{i-1}), \mathrm{addr}_s(\mathrm{N}_{i+1}) \rangle \oplus H_1(P_{K_3}(w), r_i), r_i)$, due to the pseudo-randomness of P and the random oracle H_1, all PPT adversaries $\mathcal{A}\ominus$ cannot distinguish $\tilde{\mathrm{A}}_s$ from A_s. Similarly, it cannot distinguish $\mathrm{T}_s, \mathrm{A}_d, \mathrm{T}_d$ with $\tilde{\mathrm{T}}_s, \tilde{\mathrm{A}}_d, \tilde{\mathrm{T}}_d$ if the pseudo-randomness of F, G and P holds. It means the adversary can distinguish the real index $\mathrm{A}_s, \mathrm{T}_s, \mathrm{A}_d, \mathrm{T}_d$ from the simulated index $\tilde{\mathrm{A}}_s, \tilde{\mathrm{T}}_s, \tilde{\mathrm{A}}_d, \tilde{\mathrm{T}}_d$. Therefore, the probability ε_1 is negligible.

2. Based on the elliptic curve discrete logarithm assumption and the pseudo-randomness of F, any PPT adversary $\mathcal{A}$ cannot distinguish the real leaf nodes $\mathcal{L}$ from the simulated one $\tilde{\mathcal{L}}$, therefore cannot distinguish the tree $\tilde{\alpha}$ from α, since they are generated by these leaves. i.e. there exist negligible value ε_2 such that:

$$|\mathbf{Pr}[1 \leftarrow \mathcal{A}(\delta, \mathbf{f}, \alpha)] - \mathbf{Pr}[1 \leftarrow \mathcal{A}(\delta, \mathbf{f}, \tilde{\alpha})]| \leq \varepsilon_2.$$

Because the pseudo-randomness of F holds, any PPT adversary cannot distinguish the random bits from the output of PRF F. So it cannot distinguish the random bits $\gamma_s(id(w_i))$ with $F_{K_1}(w_i)$. In addition, due to the discrete logarithm assumptions, any PPT adversary cannot can distinguish g^{ω_i} with $g^{\prod_{f \in \mathbf{f}_{w_i}} (s + id(f))}$. As we know, the real

leaf nodes $\mathcal{L}$ from the simulated one $\tilde{\mathcal{L}}$ are: $\tilde{\mathcal{L}} = \{\tilde{\theta}_w\}_{w\in\mathbf{w}} = \{(\tilde{\theta}_{w,1}, \tilde{\theta}_{w,2})\}_{w\in\mathbf{w}} = (\gamma_s(id(w_i)), g^{\omega_i})$; $\mathcal{L} = \{\theta_w\}_{w\in\mathbf{w}} = \{(\theta_{w,1}, \theta_{w,2})\}_{w\in\mathbf{w}} = (F_{K_1}(w_i), g^{\prod_{f\in\mathbf{f}_{w_i}}(s+id(f))})$,

So $\mathcal{A}$ cannot distinguish $\mathcal{L}$ from $\tilde{\mathcal{L}}$. The tree $\tilde{\alpha}$ and α are build by the collision-resistant hash function H_3 of the leaf nodes, so the adversary cannot distinguish the tree $\tilde{\alpha}$ and α, then ε_2 is negligible.

3. If the private-key encryption system Π is CPA-secure, then for all PPT adversaries $\mathcal{A}$, there exists negligible value ε_3 such that:

$$|\mathbf{Pr}[1 \leftarrow \mathcal{A}(\mathbf{f}, \mathbf{c})] - \mathbf{Pr}[1 \leftarrow \mathcal{A}(\mathbf{f}, \tilde{\mathbf{c}})]| \leq \varepsilon_3.$$

Because the private-key encryption system Π is proved to be CPA secure, so the ciphertexts it produces do not reveal any partial information about the plaintext. So any PPT adversary $\mathcal{A}$ cannot distinguish the ciphertexts that generated by two different inputs using the SKE encryption. As we know, $\tilde{\mathbf{c}}$ and $\mathbf{c}$ are:$\mathbf{c} = (c_1, \ldots, c_{\#\mathbf{f}})$, in which $c_i = \Pi.\mathrm{Enc}_{K_4}(f_i)$; $\tilde{\mathbf{c}} = (\tilde{c}_1, \ldots, \tilde{c}_{\#\mathbf{f}})$, in which $\tilde{c}_i = \Pi.\mathrm{Enc}_{K_4}(\mathbf{0}^{|f|})$. So $\mathcal{A}$ cannot distinguish $\tilde{\mathbf{c}}$ from $\mathbf{c}$. Therefore, ε_3 is negligible.

In addition, there exists negligible value ε_4 such that:

$$|\mathbf{Pr}[1 \leftarrow \mathcal{A}(\delta, \mathbf{f}, \tau_s)] - \mathbf{Pr}[1 \leftarrow \mathcal{A}(\delta, \mathbf{f}, \tilde{\tau}_s)]| \leq \varepsilon_4$$

Because the pseudo-randomness of F, G, P holds, any PPT adversary cannot distinguish the random bits from the output of PRF F, G, P. So it cannot distinguish $\tilde{\tau}_i = \left(\gamma_s(id(w_i)), \tilde{\mathrm{T}}'_s[\gamma_s(id(w_i))] \oplus \mathrm{iT}_s(id(w_i)), K_{id(w_i)}\right)$ with $\tau_i = (F_{K_1}(w_i), G_{K_2}(w_i), P_{K_3}(w_i))$, so as $\tau_s = (\tau_1, \ldots \tau_n)$ and $\tau_s = (\tilde{\tau}_1, \ldots \tilde{\tau}_n)$. Therefore, ε_4 is negligible.

In the same way, based on the elliptic curve discrete logarithm assumption and the pseudo-randomness of F, G, P, we have:

$$\left|\mathbf{Pr}\left[1 \leftarrow \mathcal{A}\left(\delta, \mathbf{f}, \tau_a, c_f\right)\right] - \mathbf{Pr}\left[1 \leftarrow \mathcal{A}\left(\delta, \mathbf{f}, \tilde{\tau}_a, \tilde{c}_f\right)\right]\right| \leq \varepsilon_5,$$

$$|\mathbf{Pr}[1 \leftarrow \mathcal{A}(\delta, \mathbf{f}, \tau_d)] - \mathbf{Pr}[1 \leftarrow \mathcal{A}(\delta, \mathbf{f}, \tilde{\tau}_d)]| \leq \varepsilon_6,$$

where $\varepsilon_5, \varepsilon_6$ are all negligible values.

To sum up, we have the conclusion that for all PPT adversaries $\mathcal{A}$, the output of $\mathrm{Real}_{\mathcal{A}}(1^k)$ and $\mathrm{Ideal}_{\mathcal{A},\mathcal{S}}(1^k)$ are identical, except with negligible probability $\mathbf{negl}(1^k)$, i.e.:

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

Therefore our dynamic MSE scheme is $(\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3, \mathcal{L}_4)$-secure against adaptive chosen-keyword attacks in random oracle model. □

4.2 Unforgeability

Theorem 2. *If H_3 is collision-resistant hash function and the bilinear q-SDH assumption holds then the dynamic MSE scheme is unforgeable.*

Proof: The main idea to give the proof is that, if there exists a PPT adversary $\mathcal{A}$ such that $\text{Forge}_{\mathcal{A}}(1^k) = 1$, then there exist a PPT simulator $\mathcal{S}$ that breaks at least one of the assumptions : The collision-resistance property of H_3 and the Bilinear q-SDH assumption.

During the game, the simulator $\mathcal{S}$ interacts with $\mathcal{A}$ using real algorithm. Assume after q times queries, $\mathcal{A}$ outputs a set of file identifiers $\mathbf{I}' \neq \mathbf{I}_W$ and a valid proof π. This means the proof $\pi = \{\mathcal{T}, \mathcal{S}, \mathcal{C}\}$ he produces under query W passes all three steps of the verification phase. We categorize the forgery into three types:

Type I forgery: For some word $w_i \in W$, the adversary outputs a different leaf value $\widehat{\theta}_{w_i}$ in Merkle tree proof $\widehat{t}_i$ and passes the verification step 1.

Type II forgery: For some word $w_i \in W$, $\mathbf{I}' \not\subset S_i$. The adversary gives the simulator the real accumulation value in the proof t_i, and outputs a subset witness $\widehat{g}^{P_i}$ that passes the verification step 2.

Type III forgery: The set $\mathbf{I}'$ is a proper subset of $\mathbf{I}_W$. The adversary gives the simulator the real $\mathcal{S} = \{g^{P_1}, \ldots, g^{P_n}\}$, and outputs a completeness witness $\widehat{\mathcal{C}}$ which passes the verification step 3.

It is clear that if $\mathbf{I}' \neq \mathbf{I}_W$ and proof π is valid then one of the above mentioned forgeries must occur. Next we show that the simulator $\mathcal{S}$ can use type I forgery to break the collision-resistance property of H_3, and use type II or III forgeries to break the bilinear q-SDH assumption.

1. The collision-resistance property of H_3

The hash function H_3 is collision-resistance if it is difficult for all PPT adversaries to find two different messages m_1 and m_2, such that $H_3(m_1) = H_3(m_2)$.

First, given the hash function H_3, the simulator $\mathcal{S}$ interacts with the adversary $\mathcal{A}$ according to the game $\text{Forge}_{\mathcal{A}}(1^k)$. If $\mathcal{A}$ wins the game and the Type I forgery occurs, that means for some word $w_i \in W$, $\mathcal{A}$ outputs a different leaf value $\widehat{\theta}_{w_i}$ in Merkle tree proof $\widehat{t}_i$.. Then, the simulator $\mathcal{S}$ verifies the value $\mathcal{A}$ outputs, which passes the verification step 1. Let $\widehat{\theta}_{w_i} = (\widehat{\theta}_{w_i,1}, \widehat{\theta}_{w_i,2})$. Passing the verification step 1 means the following two conditions hold:

- The search key $F_{K_1}(w_i) = \widehat{\theta}_{w_i,1}$.
- The Merkle tree verification using the leaf $\widehat{\theta}_{w_i}$ succeeds.

According to the verification step 1, the adversary $\mathcal{A}$ may only forge the $\widehat{\theta}_{w_i,2}$. Then passing the Merkle tree verification implies that the adversary is able to find the collision of H_3, because it can generate the same root with the modified leaf.

2. Bilinear q-SDH assumption

Given the simulator $\mathcal{S}$ an instance of bilinear q-SDH problem: $(p, \mathbb{G}, \mathcal{G}, e, g)$ and a $(q+1)$-tuple $(g, g^s, \ldots, g^{s^q})$. $\mathcal{S}$ interacts with the adversary $\mathcal{A}$ in the following way.

First, since $\mathcal{S}$ doesn't know the value s in the given bilinear q-SDH instance, it needs to reconstruct the following algorithms which related to s in the game $\text{Forge}_{\mathcal{A}}(1^k)$:

In the algorithm Gen, the simulator $\mathcal{S}$ directly uses $(g, g^s, \ldots, g^{s^q})$ as the public parameters without knowing s, and sends them to the adversary.

And in the algorithm Setup, for the leaf nodes: $\theta_w = \left(F_{K_1}(w), g^{\prod_{f \in \mathbf{f}_w}(s+id(f))}\right)$, the simulator $\mathcal{S}$ computes the value using $(g, g^s, \ldots, g^{s^q})$:$g^{\prod_{f \in \mathbf{f}_w}(s+id(f))}$.

It is worth mentioning that, the simulator $\mathcal{S}$ needs to construct an extra auxiliary data structure $\mathcal{N}$. It stores for each leaf nodes θ_w the polynomial:$n_w = \prod_{f \in \mathbf{f}_w}(s+id(f))$, which is used to form the add/delete tokens later.

Simulator $\mathcal{S}$ cannot directly compute the value of τ_a in Add/Update protocol. In the Add/Update protocol's user's step 2, when computing the value of the new leaf node θ_i' in τ_a, it first finds $\mathcal{N}$ to find the polynomials n_i that equals $\theta_{i,2}$, then computes the value $g^{n_i \cdot (s+id(f))}$ using $(g, g^s, \ldots, g^{s^q})$. The value $\theta_i' = (\theta_{i,1}, g^{n_i \cdot (s+id(f))})$ is the updated leaf node.

Similarly, in the Del/Update protocol, $\mathcal{S}$ finds the n_i in $\mathcal{N}$ and removes the factor $s+id(f)$ from n_i and then computes the value $g^{n_i/(s+id(f))}$ using $(g, g^s, \ldots, g^{s^q})$ and gets a new leaf node $\theta_i' = (\theta_{i,1}, g^{n_i/(s+id(f))})$.

In this modified game, the values that related to s are computed in a new way. However, it produces same output as it was produced by earlier version of algorithm. So in the adversary $\mathcal{A}$'s view, these values are still valid, it cannot distinguish this game with the original one.

If $\mathcal{A}$ wins the game, and the following two types of forgeries occur, then the simulator $\mathcal{S}$ may solve the given bilinear q-SDH instance.

- If the Type II forgery occurs, i.e., the adversary $\mathcal{A}$ outputs a set $\mathbf{I}' = \{x_1, \ldots, x_m\}$ and a witness $\widehat{g}^{P_i}$ for some $w_i \in W$. Let $S_i = \{id_1, \ldots, id_q\}$ be the file identifier set related to the word w_i, then $g^{(s+id_1)(s+id_2)\cdots(s+id_q)}$ is the corresponding accumulation value. Since $\mathbf{I}' \not\subset S_i$, there exists some $1 \le j \le m$, such that $x_j \notin S_i$. This means in the equation: $e(g^{\prod_{x \in \mathbf{I}'}(s+x)}, \widehat{g}^{P_i}) = e(g^{(s+id_1)(s+id_2)\cdots(s+id_q)}, g)$, $(s+x_j)$ does not divide $(s+id_1)(s+id_2)\cdots(s+id_q)$. Therefore there exists polynomial $Q(s)$ of degree $q-1$ and constant $c \neq 0$, such that:$(s+id_1)(s+id_2)\cdots(s+id_q) = Q(s)(s+x_j) + c$.

  Then the simulator $\mathcal{S}$ has $e(g, \widehat{g}^{P_i})^{(s+x_j)\prod_{x \in \mathbf{I}' \wedge x \neq x_j}(s+x)} = e(g,g)^{Q(s)(s+x_j)+c}$.

  After transformation, it can finally have $e(g,g)^{1/s+x_j} = e(g, \widehat{g}^{P_i})^{\prod_{x \in \mathbf{I}' \wedge x \neq x_j}(s+x)} e(g,g)^{-Q(s)})^{1/c}$. This means the simulator $\mathcal{S}$ can solve the instance of bilinear q-SDH problem in polynomial time.

- If the Type III forgery occurs, i.e., the adversary $\mathcal{A}$ outputs a set $\mathbf{I}' = \{x_1, \ldots, x_m\}$ and the completeness witness $\widehat{\mathcal{C}}$. Since the set $\mathbf{I}'$ is the proper subset of $\mathbf{I}_W$, there exists at least one common factor in polynomials $P_1, \ldots, P_n$. We use $(s+x)$ to denote the factor, where $x \notin \mathbf{I}'$. These values can pass the verification step 3 means the following holds: $\prod_{i=1}^{n} e(g^{P_i}, g^{q_i}) = e(g, g)$. And extract $(s+x)$ from each P_i by computing $g^{P'_i} = (g^{P_i})^{1/(s+x)}$, then $\prod_{i=1}^{n} e(g^{P_i}, g^{q_i}) = (\prod_{i=1}^{n} e(g^{P'_i}, g^{q_i}))^{s+x} = e(g, g)$.

Thus the simulator $\mathcal{S}$ can easily form the solution of the instance of bilinear q-SDH problem by computing: $e(g, g)^{1/(s+x)} = \prod_{i=1}^{n} e(g^{P'_i}, g^{q_i})$. This means the simulator can also solve the instance of bilinear q-SDH problem in polynomial time.

The above analyses show that, if the adversary $\mathcal{A}$ could successfully forge a proof, it must have the ability to break at least one of these assumptions above. Therefore we have the conclusion that for all PPT adversaries $\mathcal{A}$, the probability

$$\mathbf{Pr}[\mathrm{Forge}_{\mathcal{A}}(1^k) = 1] \leq \mathbf{negl}(1^k),$$

where $\mathbf{negl}(1^k)$ is a negligible function with input 1^k. Thus our dynamic MSE scheme is unforgeable. □

5 Conclusion

Searchable encryption is an important cryptographic primitive for cloud storage environment. It is well motivated by the popularity of cloud storage services. At the same time, the authentication methods utilized for the search results' verification is a significant supplement that makes the search more reliable and it would greatly promote the development of cloud storage service.

In this paper, we described a searchable encryption scheme which supports multiple keywords search and authentication. The scheme can greatly reduce the communication cost during the search. We demonstrated that, taking into account the security challenges in the cloud storage, our scheme can withstand the chosen-keyword attack carried out by the adaptive adversaries. Proposed scheme also prevents the result from being maliciously altered by those adversaries.

In the future, we will perform a detailed analysis of the security aspects in this paper and investigate the feasibility of designing improved security model to enhance the scheme's security features. Moreover, we will give consideration to the authenticate techniques to achieve more efficiency to meet practical needs.

Acknowledgement. This work was supported in part by the National Science and Technology Major Project under Grant No. 2013ZX03002006, the Liaoning Province Science and Technology Projects under Grant No. 2013217004, the Liaoning Province Doctor Startup Fund under Grant NO. 20141012, the Fundamental Research Funds for the Central Universities under Grant No. N130317002, the Shenyang Province Science and Technology Projects under Grant No. F14-231-1-08, and the National Natural Science Foundation of China under Grant Numbers 61472184, 61321491, 61272546.

References

1. Cachin, C., Keidar, I., Shraer, A.: Trusting the cloud. ACM SIGACT News. **40**(2), 81–86 (2009)
2. Kamara, S., Lauter, K.: Cryptographic cloud storage. In: Sion, R., Curtmola, R., Dietrich, S., Kiayias, A., Miret, J.M., Sako, K., Sebé, F. (eds.) RLCPS, WECSR, and WLC 2010. LNCS, vol. 6054, pp. 136–149. Springer, Heidelberg (2010)
3. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: IEEE Symposium on Security and Privacy (S&P 2010), Oakland, California, USA, pp. 44–55. IEEE Computer Society (2010)
4. Goh, E.J.: Secure indexes. cryptology. ePrint Archive, Report 2003/216
5. Chang, Y.-C., Mitzenmacher, M.: Privacy preserving keyword searches on remote encrypted data. In: Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 442–455. Springer, Heidelberg (2005)
6. Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. J. Comput. Secur. **19**(5), 895–934 (2011)
7. Kamara, S., Papamanthou, C., Roeder, T.: Dynamic searchable symmetric encryption. In: Proceedings of CCS 2012, pp. 965–976. ACM (2012)
8. Cash, D., Jaeger, J., Jarecki, S., Jutla, C.S., Krawczyk, H., Rosu, M.C., Steiner, M.: Dynamic searchable encryption in very-large databases: data structures and implementation. I Cryptology ePrint Archive, Report 2014/853
9. Stefanov, E., Papamanthou, C., Shi, E.: Practical dynamic searchable encryption with small leakage. In: Proceedings of the Network and Distributed System Security Symposium (NDSS 2011), 23–26 February 2011, San Diego, California, USA. The Internet Society (2011)
10. Golle, P., Staddon, J., Waters, B.: Secure conjunctive keyword search over encrypted data. In: Jakobsson, M., Yung, M., Zhou, J. (eds.) ACNS 2004. LNCS, vol. 3089, pp. 31–45. Springer, Heidelberg (2004)
11. Ballard, L., Kamara, S., Monrose, F.: Achieving efficient conjunctive keyword searches over encrypted data. In: Qing, S., Mao, W., López, J., Wang, G. (eds.) ICICS 2005. LNCS, vol. 3783, pp. 414–426. Springer, Heidelberg (2005)
12. Byun, J.W., Lee, D.-H., Lim, J.-I.: Efficient conjunctive keyword search on encrypted data storage system. In: Atzeni, A.S., Lioy, A. (eds.) EuroPKI 2006. LNCS, vol. 4043, pp. 184–196. Springer, Heidelberg (2006)
13. 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. **25**(1), 222–233 (2014)
14. Kamara, S., Papamanthou, C., Roeder, T.: CS2: a searchable cryptographic cloud storage system. TechReport MSR-TR-2011-58, Microsoft Research (2011)
15. Merkle, R.C.: A certified digital signature. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 218–238. Springer, Heidelberg (1990)
16. Kurosawa, K., Ohtaki, Y.: UC-secure searchable symmetric encryption. In: Keromytis, A.D. (ed.) FC 2012. LNCS, vol. 7397, pp. 285–298. Springer, Heidelberg (2012)
17. Papamanthou, C., Tamassia, R., Triandopoulos, N.: Optimal verification of operations on dynamic sets. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 91–110. Springer, Heidelberg (2011)
18. Katz, J., Lindell, Y.: Introduction to Modern Cryptography. CRC Press, Boca Raton (2014)
19. Nguyen, L.: Accumulators from bilinear pairings and applications. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 275–292. Springer, Heidelberg (2005)