Information Sciences 424 (2018) 1–26

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Blocked linear secret sharing scheme for scalable attribute based encryption in manageable cloud storage system

Jing Wang[a,b,c], Chuanhe Huang[b,c,*], Neal N. Xiong[d], Jinhai Wang[b,c]

[a] School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China
[b] State Key Lab of Software Engineering, Computer School, Wuhan University, China
[c] Collaborative Innovation Center of Geospatial Technology, China
[d] Colorado Technical University, USA

ARTICLE INFO

Article history:
Received 25 November 2016
Revised 4 September 2017
Accepted 10 September 2017
Available online 23 September 2017

Keywords:
Cloud
Data security
Access control
Attribute-based encryption
Access policy management

ABSTRACT

Cloud provides outsourced storage services in a cost-effective manner. A key challenge of cloud storage is the security and privacy of outsourced data. A security mechanism known as attribute-based encryption (ABE) represents the state-of-the-art in providing fine-grained access control for cloud storage. The managing of access policy is a critical issue of ABE. Policy managing may incur substantial computation and communication overhead in the ABE scheme with unscalable access policy. In this work, we firstly propose a form of scalable access policy named blocked linear secret sharing scheme (BLSSS). The scalability of BLSSS provides efficient policy managing interface for ABE scheme. Then, we propose a scalable ciphertext-policy attribute-based encryption (SCP-ABE) scheme which uses BLSSS as access policy. Significantly, the proposed SCP-ABE is low-cost in computation and communication during policy managing. Furthermore, sufficient simulation experiments demonstrate that SCP-ABE outperforms most existing ABE schemes in terms of policy managing.

1. Introduction

Cloud storage offers a number of advantages over traditional storage in terms of availability, scalability, portability and so on. It develops a unified strategy to collect, preserve and manage data for current and future. Although the advantages of cloud storage are clear, a critical issue of the data outsourcing scenario is the enforcement of strong data security mechanisms [16]. Thus, adequate access control techniques are required to ensure such outsourcing data security. It is important for users to trust the storage service providers. Attribute based encryption (ABE) is a novel cryptographic tool which can provide fine-grained ciphertext access control. The core advantages of such ABE-based access control mechanisms are given as follows: (1) access control of the data is maintained by the customer instead of the service provider; (2) the security properties strictly depend on cryptography technology, as opposed to traditional access control. In brief, it provides great superiority over other access control mechanism based on public cloud infrastructures.

A core concept of ABE is named access policy or access structure, which is provided to describe fine-grained access privilege. Specifically, in a ciphertext-policy attribute-based encryption (CP-ABE) scheme, each ciphertext is assigned with

* Corresponding author.
E-mail addresses: wangj478@mail.sysu.edu.cn (J. Wang), huangch@whu.edu.cn (C. Huang), xiongnaixue@gmail.com (N.N. Xiong), wangjinhai@whu.edu.cn (J. Wang).

https://doi.org/10.1016/j.ins.2017.09.032
0020-0255/© 2017 Elsevier Inc. All rights reserved.

a unique access policy. The data owner manages access privilege of data by maintaining its access policy. In fact, most performances of CP-ABE scheme significantly depend on the access policy properties. For example, the scale of access policy decides the size of ciphertext of ABE scheme, the expression of access policy decides the flexibility of ABE based access control mechanism. Many researchers focus on optimizing CP-ABE by improving the access policy. However, scalability, as an important property of access policy, was usually ignored. The scalability of access policy indicates the ability of dynamic updating. The access policy with strong scalability supports real-time and fine-grained updating while data having been encrypted. It is important that access policy frequently requires to be updated due to various reasons. In most scenarios, the access policy just needs to be partially updated instead of entirely updated. However, the access policy with weak scalability can only be updated by completely re-encrypting data with new policy. It brings heavy computation and communication overhead. Significantly, the strong scalability of access policy is required by the manageable CP-ABE scheme to provide efficient policy updating interface to data owner. It provides three advantages of manageable CP-ABE scheme: (1)supporting real-time policy managing; (2)supporting fine-grained policy managing; (3)low-overhead of policy managing.

The grand challenge of improving policy scalability in a CP-ABE scheme is to jointly guarantee correctness, completeness and security [40]. At the same time, efficiency is further considered as an important requirement in this paper. Recently, policy scalablity only has been discussed in a few ABE schemes [11,27,40]. However, all of these schemes are still hard to meet all requirements in terms of correctness, completeness, security and efficiency. In Goyal and Sahai proposed schemes [11,27], the updated access policy should be more restrictive than the previous one, because the scalability of the access policy is limited. Although Yang's scheme has improved the completeness of policy updating method, the updating process of this scheme is still not efficiency enough [40]. In some cases, the access policy is required to re-construct. Especially, the processing of threshold updating is cumbersome. In general, the policy scalability of these existing schemes is not strong enough.

Focusing on scalability, we propose a new form of access policy called block linear secret sharing scheme (BLSSS), a special linear secret sharing scheme(LSSS). BLSSS provides efficient updating interface which greatly improves the scalability of policy. More specifically, there are four advantages of describing access policy as BLSSS. Firstly, the storage cost of BLSSS is much less than general LSSS. Because BLSSS is generated by a random seed and a tree, data owner only needs to store the seed and tree circuit instead of the whole BLSSS matrix. Secondly, the computation and communication overhead of policy updating are both low. Each block of BLSSS is independent, updating of a block is non-interfering with others. Thus, there is not additional overhead produced for other blocks, the overall updating overhead is minimized. Thirdly, BLSSS is universal for CP-ABE scheme. Not only LSSS, but also other common forms of access policy can be equivalently transformed into BLSSS[1]. Thus, BLSSS can be widely used in most existing CP-ABE scheme. Fourthly, the computational complexity of decryption is reduced for CP-ABE scheme. In the CP-ABE scheme with BLSSS, the decrypting operation can be decomposed and processed in block. Thus, its decrypting computation scale is mitigated[2]. Furthermore, the scalable ciphertext-policy attribute based encryption(SCP-ABE) scheme with strong policy scalability is proposed in this paper. SCP-ABE is provided with efficiency policy managing interface by taking full advantage of BLSSS matrix. A lot of simulation results have demonstrated that the proposed SCP-ABE outperforms the existing main schemes [11,27,40] in terms of policy scalability. The contributions of this paper are summarized as follows.

(1) We proposed a scalable access policy of ABE called BLSSS which is provided with block structure.
(2) We provide efficient scaling functions of BLSSS. These functions are convenient to process arbitrary policy updating.
(3) We provide a SCP-ABE scheme using BLSSS. SCP-ABE outperforms the existing main ABE schemes in terms of policy managing.

The remaining of this paper is organized as follows. Section 2 gives the related work. Then, we propose the scalable access policy(i.e. BLSSS) in Section 3. Thirdly, we propose the SCP-ABE scheme in Section 4. In Section 5, we analyse the performance of our BLSSS and SCP-ABE scheme. Finally, the conclusions and future works are given in Section 6.

2. Related work

Cloud computing is a new architecture can be viewed as the next generation computing paradigm [7,25]. It introduces a major change in data storage and application execution [35]. For instance, the United States Library of Congress moved its digitized content to the cloud [1]. It develops a national strategy to collect, preserve and make available digital content for current and future generations based on cloud storage.

Many researchers focus on the security of cloud storage systems [9,15,29,34,38,39]. ABE provides a smart way to construct a fined-grained access control for cloud [11–13,24,32,36,37]. An important issue of ABE is managing access privilege. Recently, there are two kinds of solution of this issue: key revocation [14,19,20,41] and policy updating [10,11,27,40]. Significantly, policy updating provides more fine-grained management pattern for ABE, it allows data owner to manage the privilege in a more flexible and scalable way.

In fact, in a CP-ABE scheme, access privilege of data is described by expressive access policy which is derived from a secret sharing scheme [30]. A wide range of researches have been proposed to design secret sharing scheme, such as the

[1] The detailed transformation of monotonous Boolean formula and access tree are all given in Section 3.3.1
[2] The detailed explanation, analysis and instance are all given in Section 3.4.1

Table 1
Comparison of ABE Schemes provided with policy managing interface.

Scheme	Goyal's [11]	Sahai's [27]	Yang's [40]	Ours SCP-ABE
Constraint condition	Y	Y	N	N
Policy form	Access tree	LSSS	Boolean formula, Access tree, LSSS	Boolean formula, Access tree, LSSS
Policy re-constructing	N	Y	Y	N
Updating unit	Node	Row	Node/Row	Sub-tree/Block

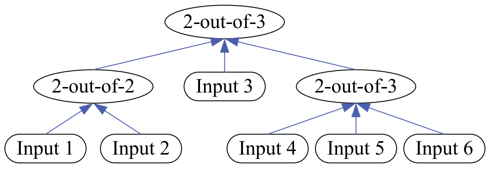


Fig. 1. An access policy described as tree-circuit.

schemes proposed by Shamir [28], Benaloh and Leichter [2], Bertilsson and Ingemarsson [5], Brickell [6], Massey [22], Blakley and Kabatianskii [4], Simonis and Ashikhmin [26], Ventzislav and Nikova [23], Mauricio and Wigderson [17], Dehkordi and Ghasemi [8], Sun et al. [31] and so on. Thus, access policy is often expressed as various forms, such as monotonic Boolean formula, access tree and LSSS. However, most forms of access policy are weak in scalability, which makes the policy updating operation limited and inefficient.

As a frequently-used access policy, LSSS is first proposed by Blakley and Kabatianskii [4], and its formal definition is proposed by Beimel [3]. Furthermore, LSSS is firstly introduced into ABE as access policy by Waters [33]. Then, Liu et al. [21] proposed a specific LSSS which is more expressive and efficient. However, the scalability of LSSS is still ignored in [21], it provided a possibility of improving policy scalability. The BLSSS proposed in this paper is a improved version of [21] with strong scalability and efficient scaling functions. It provide efficient interface for data owner to manage access policy.

Recently, only a few ABE schemes have considered the policy managing issue [10,11,27,40]. However, the scheme proposed by Fugkeaw and Sato [10] provided with very lightweight proxy re-encryption (VL-PRE) operation, it only gives outline of each phase during policy updating. It is hard to work as a baseline during comparison. Thus, Table 1 only shows the comparison of other existing schemes [11,27,40] and our SCP-ABE in terms of policy managing. Significantly, the policy managing operations of these schemes are compared in four aspects: constraint condition of policy updating, applicative policy form, requiring of policy re-construct and policy updating unit. Firstly, different from Goyal's scheme [11] and Sahai's scheme [27], there is no restriction of policy updating in Yang's scheme [40] and our SCP-ABE scheme. That implies arbitrary policy updating can be achieved in the later two schemes. Secondly, Goyal's scheme [11] and Sahai's scheme [27] only provide updating operations for the policy expressed as single form while Yang's scheme [40] and our SCP-ABE supporting updating of multiform policy. Thirdly, only the updating operations proposed in Goyal's scheme [11] and SCP-ABE do not need to re-construct policy at all. Fourthly, the updating unit of our SCP-ABE is block/sub-tree while Goyal's scheme [11], Sahai's scheme [27] and Yang's scheme [40] processing a group of updated rows/nodes in order. It implies that our SCP-ABE is more efficient than others in term of policy managing. In summary, BLSSS updating is more flexible and efficient, it provides optimized policy managing interface to SCP-ABE.

3. Scalable access policy

3.1. Prime definitions

The access policy of ABE is usually defined as monotonic access structures.

Definition 1 (monotonic Access Structures). Let $P = \{P_1, P_2, \dots, P_N\}$ be a set of parties. A collection $\mathbb{A} \subset 2^{\{P_1, P_2, \dots, P_N\}}$ is monotonic if

$$\forall B, C \subset P, B \in \mathbb{A} \wedge B \subset C \to C \in \mathbb{A}$$

Furthermore, $\forall A \in \mathbb{A}$ is called authorized set and $\forall A \notin \mathbb{A}$ is called unauthorized set. In ABE, the role of the parties is taken by the attribute. Thus, an access policy $\mathbb{A}$ is a collection of attribute sets. In fact, an access policy is realized by a Secret Sharing Scheme which decomposes a secret s into a share set $S = \{s_i, \dots, s_n\}$. Let ρ be a map form S to P, S' be a subset of S and $P' = \{\rho(i) | s_i \in S'\}$. Not that secret s can be recovered by S' iff P' is authorized.

Furthermore, an access policy can be viewed as a tree-circuit which consists of a set of gates (e.g. AND, OR, threshold and so on) and inputs, as shown in Fig. 1. The circuit takes a set of shares S' as inputs and outputs the secret s iff P' is authorized. For example, $\{input1, input2, input3\}$ is an authorized set of the access policy given in Fig. 1.

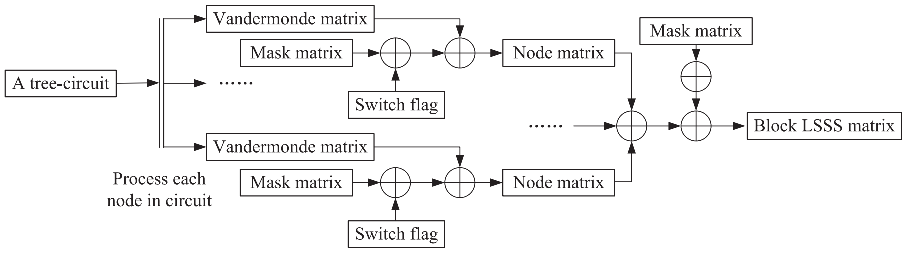


Fig. 2. Tree-circuit to BLSSS.

Table 2
Key notations.

Notation	Description
I_k	The unit matrix with order k.
$O_{m\times n}$	The zero matrix with m rows and n columns.
$R(*)$	The rank of the matrix $*$.
ε	A binary vector in $\{0, 1\}^n$, ε_i denotes the i^{th} entry of ε.
$\mathbf{e_i}$	A unit vector whose i^{th} entry is 1 and other entries are all 0.
M^{ε}	A sub-matrix of M, M_i is a row of M^{ε} iff $\varepsilon_i = 1$.
$* \approx **$	Matrix $*$ and matrix $**$ can describe the same access policy.
$\|\|*\|\|_0$	The standard zero-norm of the vector $*$.

As a frequently-used form of access policy, LSSS is defined as follows [33]:

Definition 2 (Linear Secret Sharing Scheme)**.** A secret sharing scheme Π over a set of parties P with secret $s \in \mathbb{Z}_p$ is linear (over $\mathbb{Z}_p$) if:

1. The shares of each party from a vector over $\mathbb{Z}_p$.
2. There exists a matrix $M \in \mathbb{Z}_p^{l\times n}$. For all $i \in \mathbb{Z}_l$, the i^{th} row of M is associated with an attribute $\rho(i)$. Then, a column vector $v = (s, r_2, \dots, r_n)$ is chosen, where $s \in \mathbb{Z}_p$ is the secret and $r_2, \dots, r_n \in \mathbb{Z}_p$ are chosen at randomly. Finally, $S = Mv$ is the vector includes l shares of s.
3. The secret of Π is reconstructed as $s = \sum_{i\in A} \omega_i s_i$, where A denotes arbitrary authorized set, s_i denotes the s^{th} entity of S and $\sum_{i\in A} \omega_i M_i = (1, 0, 0, \dots, 0)$[3].

The BLSSS proposed in this paper is a special LSSS with block structure and the formal definition is given as follows:

Definition 3 (Block Linear Secret Sharing Scheme)**.** Let $\mathbb{T}$ be a tree-circuit of an access policy and matrix M be the LSSS corresponding to $\mathbb{T}$. M is a BLSSS if $M = (\mathbf{m}_{i,j})_{l'\times n'}$ where i denotes the index of leaf l_i in $\mathbb{T}$, j denotes the index of non-leaf n_j in T, l' denotes the total of leaves of $\mathbb{T}$, n' denotes the total of non-leaves of $\mathbb{T}$, $\mathbf{m}_{i,j}$ denotes a sub-matrix of M indicated by l_i and n_j. Furthermore, $\mathbf{m}_{i,j}$ depends on the node matrix of n_j.

Furthermore, BLSSS is an improved version of the LSSS proposed by Liu et al. [21] and this paper also adapts BLSSS generating method from the method provided in [21]. Fig. 2 shows the process of BLSSS generating following the definition. Firstly, each non-leaf node is assigned a node matrix[4]. The node matrix is generated by a Vandermonde matrix and a mask matrix. Secondly, a BLSSS is generated by compounding all of the node matrices. Note that the BLSSS generated in this step is a sparse block matrix. Additionally, the BLSSS can be masked with a full rank matrix. However, the mask matrix is chosen as unit matrix while the BLSSS does not needs to mask.

Significantly, each node matrix of BLSSS is independent. It implies that the updating of a node matrix bring little influence to other parts of the BLSSS. Thus, the updating operation of BLSSS is more efficient than general LSSS.

3.2. Prime notations

In the following paper, unless otherwise indicated, a boldface lowercase letter denotes a vector, an uppercase letter denotes a matrix and frequently referred notations are summarized in Table 2.

Additionally, we define some matrices with specific structure:

$$I'_k = \begin{pmatrix} O \\ I_{k-1} \end{pmatrix} \tag{1}$$

[3] The detailed proof of correctness is given in [3].
[4] The detailed description of node matrix is given in Section 3.3.

where O is the simplified symbol of $O_{1\times(k-1)}$. For simplicity, the subscript is omitted while it is unambiguous.

$$F(V,k)=\begin{pmatrix} V \\ O_{(k-1)\times l}\end{pmatrix} \tag{2}$$

where $V\in Z^{l}$.

$$E(A,k_1,\ldots,k_n)=\begin{pmatrix} F(A_1,k_1) & I'_{k_1} & \ldots & O \\ \vdots & \vdots & \ddots & \vdots \\ F(A_n,k_n) & O & \ldots & I'_{k_n}\end{pmatrix} \tag{3}$$

where $A\in Z^{n\times l}$ and A_i denotes the ith row of A.

$$U(M_1,\ldots,M_n)=\begin{pmatrix} M_1 & \ldots & O \\ \vdots & \ddots & \vdots \\ O & \ldots & M_n\end{pmatrix} \tag{4}$$

Let $M_i=(\bar{M}_i,\hat{M}_i)\in Z^{l_i\times k_i}$ where $\bar{M}_i\in Z^{l_i\times 1}$, $\hat{M}_i\in Z^{l_i\times(k_i-1)}$, we can get

$$\begin{aligned} U^*(A,M_1,\ldots,M_n) &= U(M_1,\ldots,M_n)E(A,k_1,\ldots,k_n) \\ &= \begin{pmatrix} \bar{M}_1A_1 & \hat{M}_1 & \ldots & O \\ \vdots & \vdots & \ddots & \vdots \\ \bar{M}_nA_n & O & \ldots & \hat{M}_n\end{pmatrix}\end{aligned} \tag{5}$$

Specifically,

$$U^{\dagger}(A,k,M)=\begin{pmatrix} A_1 & O \\ \vdots & \vdots \\ A_k\bar{M} & \hat{M} \\ \vdots & \vdots \\ A_n & O\end{pmatrix} \tag{6}$$

3.3. Node matrix

Node matrix is a special LSSS which can describe a node in the tree-circuit of access policy. Generally, a t-out-of-n node (as a general gate) can be described by a matrix $M\in Z^{n\times t}$ iff $\forall\varepsilon\in\{0,1\}^n, R(M^{\varepsilon})=t$ while $||\varepsilon||_0=t$.

3.3.1. Transformation

In order to make the BLSSS more universally be used in ABE schemes, we present a method to transform various gate into node matrices. Specifically, the CP-ABE schemes adopting other forms of access policy can equivalent process their ciphertext policy as BLSSS by using the transformation method. In the transforming, the node matrix must keep the same inputs and output as the gate does. It implies that the share set generated by the node matrix is equal to which generated by the gate. Thus, the access policy expressed as other forms can be equivalently transformed into BLSSS[5].

In a monotonic Boolean expression, there are two gate forms: AND($\wedge$) and OR($\vee$) [18]. Specially, threshold(t-out-of-n, $Thr_{t,n}(a_1,\ldots,a_n)$) can be represented by a combination of ANDs and ORs here, e.g. $Thr_{2,3}(a_1, a_2, a_3)\Longleftrightarrow(a_1\wedge a_2)\vee(a_2\wedge a_3)\vee(a_1\wedge a_3)$. Thus, we only give the transforming of AND and OR for monotonic Boolean expression. Let an original AND gate share set be $S=\{s_1,\ldots,s_n\}$ and secret $s=\sum s_i$. The node matrix and vector are defined as follows:

$$M_A=\begin{pmatrix} 1 & -1 & \ldots & 0 \\ 0 & 1 & \ddots & \vdots \\ \vdots & \vdots & \ddots & -1 \\ 0 & 0 & \ldots & 1\end{pmatrix}, V_A=\begin{pmatrix} s & v_2 & \ldots & v_n\end{pmatrix}$$

[5] The transformation of policy is processed as follows. Firstly, each gate of other form policy is transformed into equivalent node matrix. Then, Algorithm 2 proposed in Section 3 is run to compound node matrices and generate equivalent BLSSS.

where $v_i = \sum_{j=i}^{n} s_j$. As a result, $\forall i, M_i V_A^T = s_i$, where M_i denote the ith row vector of M_A. OR gate can be processed similarly. Let the secret $s \in Z$, the OR-node matrix and vector are defined as follows

$$M_O = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}, V_O = (s)$$

It is clearly, $\forall i, M_i V_O^T = s$ where M_i denote the ith row vector of M_O.

In an access tree, each t-out-of-n node is described by a node polynomial $f(x)$ and an interpolation set $X = \{x_1, \ldots, x_n\}$ where the order of $f(x)$ is $t-1$, the constant of $f(x)$ is secret s and share $s_i = f(x_i)$. Thus, the node can be transformed into a Vandermonde matrix $V_{n,t}$ and a node vector V_t:

$$V_{n,t} = \begin{pmatrix} 1 & x_1 & \ldots & x_1^{t-1} \\ \vdots & \vdots & & \vdots \\ 1 & x_n & \ldots & x_n^{t-1} \end{pmatrix}, V_t = \begin{pmatrix} s & a_1 & \ldots & a_{t-1} \end{pmatrix}$$

where a_j denotes the coefficient of x^j of $f(x)$. Furthermore, $\forall i, M_i V_t^T = f(x_i) = s_i$.

3.3.2. Scalable node matrix

A Vandermonde matrix $V_{n,t} \in \mathbb{Z}_p^{n\times t}$ can be expressed as a t-out-of-n gate cause of it natural property. A universal scalable node matrix is given as $M_{n,t} = V_{n,t} T_t$ where $T_t \in Z^{t\times t}$ called node mask matrix is a full rank upper triangular matrix. significantly $M_{n,t} \approx V_{n,t}$ because $M_{n,t}$ is provided with the same linear characteristic of $V_{n,t}$. Meanwhile, the node vector is generated as $V_t = (s, v_2, \ldots, v_t)$ where s is the node secret and $\forall v_i \in Z$ is chosen randomly. Finally, a t-out-of-n gate is expressed by a node matrix $M_{n,t}$ and a node vector V_t.

In detail, Algorithm 1 is proposed to generate scalable node matrix. *NodeMatrix* takes six parameters as inputs: n denotes

Algorithm 1 *NodeMatrix*(n, t, s, r_1, r_2, r_3).

```
Input: Children number: $n$;Threshold: $t$;Secret: $s$;Random numbers: $r_1, r_2, r_3$;
Output: Node matrix: $M_{n,t}$;node Vector: $V_t$;
1: Run public random number generator $R_1$ with seed $r_1$ and output $x_1, \ldots, x_n \in Z_p$; //$\forall i, j \in Z_n^*$ and $i \neq j$, we get $x_i \neq x_j$.
2: Generate Vandermonde matrix: $V_{n,t} \leftarrow \begin{pmatrix} 1 & x_1 & \ldots & x_1^{t-1} \\ \vdots & \vdots & & \vdots \\ 1 & x_n & \ldots & x_n^{t-1} \end{pmatrix}$
3: Run public random number generator $R_2$ with seed $r_2$ and output $m_1, \ldots, m_{(1+t)t/2} \in Z_p$;
4: Generate mask matrix: $T_t \leftarrow \begin{pmatrix} m_1 & m_2 & \ldots & m_{t(t-1)/2+1} \\ 0 & m_3 & \ldots & m_{t(t-1)/2+2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & m_{(1+t)t/2} \end{pmatrix}$
5: Generate node matrix: $M_{n,t} \leftarrow V_{n,t} T_t$;
6: Run private random number generator $R_3$ with seed $r_3$ and output $v_2, \ldots, v_t \in Z_p$;
7: Generate node Vector: $V_t \leftarrow (s, v_2, \ldots, v_t)$;
8: return $(M_{n,t}, V_t)$;
```

the number of children of the node; t denotes the node threshold; s denotes the node secret; r_1, r_2, r_3 denote three random seeds. Firstly, *NodeMatrix* runs three random number generator R_1, R_2, R_3 with seeds r_1, r_2, r_3 to get matrices $V_{n,t}$, T_t and vector V_t, respectively. Finally, it outputs the node vector V_t and the node matrix $M_{n,t} = V_{n,t} T_t$.

The node matrix generated by this method is strong at scalability. There are four scaling functions of node matrix defined as follows.

$Node_n^+(node, \Delta n)$: The function modifies the node matrix $M_{n,t}$ into $M_{n+\Delta n,t}$. Let $M_{n,t} = V_{n,t} T_t$ and $\Delta M_{\Delta n,t} = V_{\Delta n,t} T_t$ where $V_{n,t}$, $V_{\Delta n,t}$ are two Vandermonde matrices and T_t is a full rank upper triangular matrix. Note that, arbitrary row vector $\mathfrak{a} \in V_{n,t}$ and $\mathfrak{b} \in V_{\Delta n,t}$ must guarantee that $\mathfrak{a} \neq \mathfrak{b}$. Thus,

$$M_{n+\Delta n,t} = \begin{pmatrix} V_{n,t} \\ V_{\Delta n,t} \end{pmatrix} \tag{7}$$

$$T_t = \begin{pmatrix} M_{n,t} \\ \Delta M_{\Delta n,t} \end{pmatrix} \tag{8}$$

As a result, Δn more rows are inserted into $M_{n,t}$ and the node vector V_t is kept intact.

$Node_n^-(node, \varepsilon)$: The function removes the rows assigned by binary vector $\boldsymbol{\varepsilon}$. Let $M_{n,t} = V_{n,t}T_t$ be the original node matrix. Significantly, $M_{n,t}^{\varepsilon}$ is the modified node matrix where $M_{n,t}^{\varepsilon} = V_{n,t}^{\varepsilon}T_t$. Additionally, the node vector V_t remains intact with this function.

$Node_t^+(node, \Delta t)$: The node threshold t is changed into $t + \Delta t$, i.e., node matrix $M_{n,\,t}$ is changed into $M_{n,t+\Delta t}$. Let $M_{n,t} = V_{n,t}T_t$ and $\Delta V_{n,\,\Delta t}$ be the increment matrix of $V_{n,\,t}$ defined as follows:

$$\Delta V_{n,\Delta t} = \begin{pmatrix} x_1^t & \dots & x_1^{t+\Delta t-1} \\ \vdots & & \vdots \\ x_n^t & \dots & x_n^{t+\Delta t-1} \end{pmatrix} \tag{9}$$

Let $V_{n,t+\Delta t} = (V_{n,t}, \Delta V_{n,\Delta t})$, $T_{\Delta t} \in Z_{\Delta t \times \Delta t}$ be an upper triangular matrix and $P \in Z^{t \times \Delta t}$ be a non-zero matrix. The mask matrix is modified as follows:

$$T_{t+\Delta t} = \begin{pmatrix} T_t & P \\ O & T_{\Delta t} \end{pmatrix} \tag{10}$$

Thus, $M_{n,t+\Delta t} = V_{n,t+\Delta t}T_{t+\Delta t} = (M_{n,t}, \Delta M_{n,\Delta t})$ where $\Delta M_{n,\Delta t} = V_{n,t}P + \Delta V_{n,\Delta t}T_{\Delta t}$. Finally, the node vector V_t is changed into $V_{t+\Delta t} = (V_t, v_{t+1}, \dots, v_{t+\Delta t})$ where $v_{t+1}, \dots, v_{t+\Delta t}$ are chosen at random.

$Node_t^-(node, \Delta t)$: The function changes the node threshold t into $t - \Delta t$. Let $M_{n,t} = V_{n,t}T_t$ be the original node matrix. Similar to $Node_t^+(node, \Delta t)$, we define

$$M_{n,t} = (M_{n,t-\Delta t}, \Delta M_{n,\Delta t}) \tag{11}$$

$$V_{n,t} = (V_{n,t-\Delta t}, \Delta V_{n,\Delta t}) \tag{12}$$

$$T_t = \begin{pmatrix} T_{t-\Delta t} & P \\ O & T_{\Delta t} \end{pmatrix} \tag{13}$$

It implies that $M_{n,t-\Delta t} = V_{n,t-\Delta t}T_{t-\Delta t}$ and $\Delta M_{n,t-\Delta t} = V_{n,t-\Delta t}P + \Delta V_{n,\Delta t}T_{\Delta t}$. Thus, $M_{n,t-\Delta t}$ can be obtain by removing the last Δt columns (i.e. $\Delta M_{n,\,\Delta t}$) from $M_{n,\,t}$ directly. Finally, node vector $V_{t-\Delta t}$ is obtain by removing the last Δt entries from original node vector V_t.

3.4. Block linear secret sharing scheme

The matrix can describe not only a node but also a complex tree-circuit. A theorem in [23] provides an efficient way to construct BLSSS – compounding a *compound matrix* by a set of node matrices. Let $M_1 \in Z^{m_1 \times n_1}, \dots, M_l \in Z^{m_l \times n_l}$ be a set of LSSSs describe access policy $\mathbb{A}_1, \dots, \mathbb{A}_l$, $A \in Z^{l \times k}$ be a LSSS describes access policy $\mathbb{A}'$ and $P_1, \dots, P_l$ be the parties of $\mathbb{A}$. Access policy $(\mathbb{A}_1 \mapsto P_1) \dots (\mathbb{A}_l \mapsto P_l)\mathbb{A}'$ can be described by matrix $M = U^*(A, M_1, \dots, M_l)$.

Following the matrix compounding method, Algorithm 2, an iterative algorithm, is proposed to generate *compound BLSSS*.

Algorithm 2 $TreeToBLSSS(\mathbb{T}, node)$.

```
Input: A tree-circuit: $\mathbb{T}$;A node tree-circuit: $node$;
Output: BLSSS: $M$;Vector: $V$;
1: if $node$ is a leaf then
2:     $M \leftarrow (1)$;
3:     $V \leftarrow (1)$;
4: else//$node$ is a $t$-out-of-$n$ node
5:     Get three random numbers $r_{node,1}, r_{node,2}, r_{node,3}$ and node secret $s_{node}$;
6:     $(M_{node}, V_{node}) \leftarrow NodeMatrix(n, t, s_{node}, r_{node,1}, r_{node,2}, r_{node,3})$;
7:     for each child $c_i$ of $node$ do
8:         $(M_i, V_i) \leftarrow TreeToBLSSS(\mathbb{T}, c_i)$;
9:     end for
10:    $M \leftarrow U^*(M_{node}, M_1, \dots, M_n)$;
11:    $V \leftarrow (V_{node}, \hat{V}_1, \dots, \hat{V}_n)$;
12: end if
13: return $(M, V)$;
```

Let $\mathbb{T}$ be a tree-circuit of access policy and *root* be the root of $\mathbb{T}$. $TreeToBLSSS(\mathbb{T}, root)$ performs a depth-first traversal of $\mathbb{T}$ and generates a BLSSS M by compounding node matrices M_k, $1 \le k \le n$. For instance, Fig. 3 shows the complete generation processing of BLSSS M with tree-circuit $\mathbb{T}$.

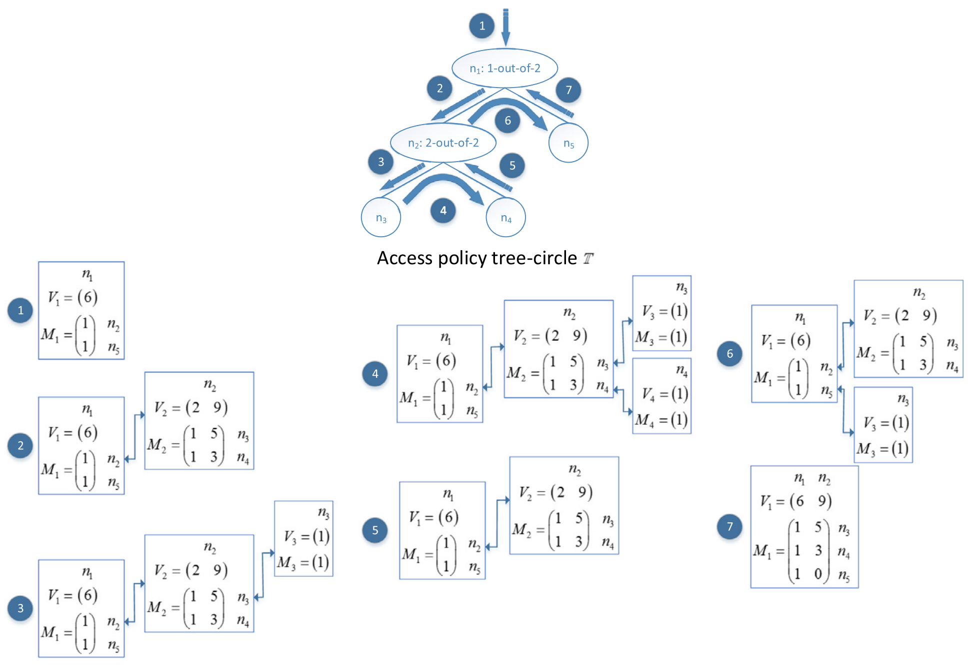


Fig. 3. Generating of BLSSS.

Furthermore, We can define a public random number generator R_0 and a private random number generator $\bar{R}_0$. At beginning of Algorithm 2, we run R_0 with a seed r_0 to generate parameter set $\{r_{node,1}, r_{node,2} | node$ is a non-leaf of $\mathbb{T}\}$ and run $\bar{R}_0$ with the secret s generate parameter set $\{r_{node,3} | node$ is a non-leaf of $\mathbb{T}\}$. Thus, the data owner can only store seed r_0, secret s and tree circuit $\mathbb{T}$ locally to keep the whole information of BLSSS. Fig. 4 shows the storage contents of different LSSS. Fig. 4 shows an access policy which can be described as a BLSSS(as shown in Fig. 4) or a general LSSS(as shown in 4). In Fig. 4, the BLSSS costs 16 units to store the tree circuit(each node costs 2 units), 1 unit to store secret s and 1 unit to store the seed r_0. In Fig. 4, the general LSSS costs 30 units to store a 6×5 matrix, 5 units to store a random vector and 6 units to store attribute indexes. It is clear that the BLSSS cost less storage units than the general LSSS.

Additionally, to facilitate discussions, we introduce the following functional notations.

$\varphi : \{1, 2, \dots, n\} \to N(\mathbb{T})$, where $N(\mathbb{T})$ denotes the set of non-leaves of $\mathbb{T}$.

$\delta : \{1, 2, \dots, l\} \to L(\mathbb{T})$, where $L(\mathbb{T})$ denotes the set of leaves of $\mathbb{T}$.

$Anc(\delta(i))$ or $Anc(\varphi(i))$: the function returns the set of ancestor of $\delta(i)$ or $\varphi(i)$.

$I(j, i)$: the function returning the index of the subtree of $\varphi(j)$ which include node $\delta(i)$ or $\varphi(i)$.

3.4.1. Properties of BLSSS

$\forall \mathbb{T}$ with l leaves and n non-leaf nodes can be transformed into following BLSSS M by Algorithm 2:

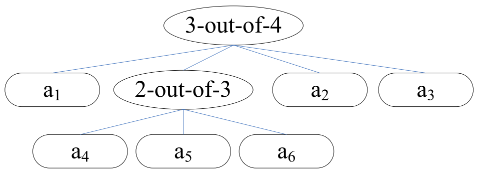
$$M = \begin{pmatrix} c_{1,1} & \mathbf{m_{1,1}} & \dots & \mathbf{m_{1,n}} \\ \vdots & \vdots & & \vdots \\ c_{1,l} & \mathbf{m_{l,1}} & \dots & \mathbf{m_{l,n}} \end{pmatrix} \tag{14}$$

where

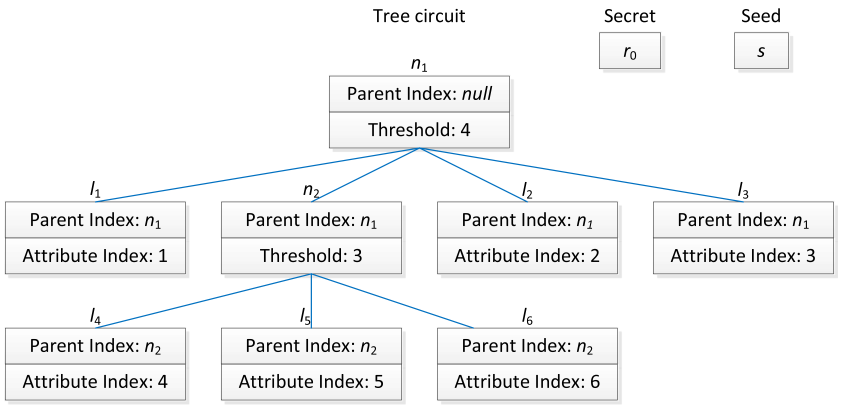
$$\mathbf{m_{i,j}} = \begin{cases} \mathbf{o} & \varphi(j) \notin Anc(\delta(i)) \\ c_{i,j}\hat{M}(\varphi(j))_k & \varphi(j) \in Anc(\delta(i)), k = I(j, i) \end{cases} \tag{15}$$

$c_{i,j} = \sum_{node \in Anc(\delta(i)) - Anc(\varphi(j))} M(node)_{1,k}$. For simplicity, $\forall node \in \mathbb{T}$, we set $\bar{M}(node) = (1, 1, \dots, 1)^T$. That implies the mask matrix of each node is modified as follows

$$T_{node} = \begin{pmatrix} 1 & m_2 & \dots & m_{t(t-1)/2+1} \\ 0 & m_3 & \dots & m_{t(t-1)/2+2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & m_{(1+t)t/2} \end{pmatrix} \tag{16}$$



(a) Access policy



(b) Storage Content of BLSSS

$$V = (432 \quad 911 \quad 182 \quad 264 \quad 164)$$

$$M = \begin{array}{cl} \begin{pmatrix} 13936 & 8549 & 9110 & 10154 & 5022 \\ 117739 & 74216 & 87286 & 83360 & 39042 \\ 22519 & 13958 & 15358 & 16304 & 7938 \\ 10342 & 5316 & 6146 & 8626 & 5670 \\ 18418 & 7348 & 10858 & 17466 & 14094 \\ 13780 & 6204 & 8142 & 12390 & 9234 \end{pmatrix} & \begin{array}{l} \boxed{\text{Attribute Index: } 1} \\ \boxed{\text{Attribute Index: } 2} \\ \boxed{\text{Attribute Index: } 3} \\ \boxed{\text{Attribute Index: } 4} \\ \boxed{\text{Attribute Index: } 5} \\ \boxed{\text{Attribute Index: } 6} \end{array} \end{array}$$

(c) Storage Content of General LSSS

Fig. 4. Storage Content of LSSS.

where $\{m_i, 2 \le i \le (1+t)t/2\}$ is a random integer set generated by the generator R_3 and the seed $r_{node,\,3}$. As a result, $\forall i, j, c_{i,j} = 1$ and equation (14) is simplified as follows

$$\mathbf{m_{i,j}} = \begin{cases} \mathbf{o} & \varphi(j) \notin Anc(\delta(i)) \\ \hat{M}(\varphi(j))_k & \varphi(j) \in Anc(\delta(i)), k = I(j,i) \end{cases} \tag{17}$$

Thus, in following paper, mask matrix is given as equation (16) shown and the first column of node matrix is set to $(1, 1, \ldots, 1)^T$.

It is important that the sorting of the column blocks does not affect the expression of access policy.

Theorem 1. *Let $s(n)$ be a sorting of $\{1, 2, \ldots, n\}$ and described as $\{j_1, j_2, \ldots, j_n\}$. Assume that, M is a BLSSS as shown in equation(14) and $M_{s(n)}$ is defined as follows:*

$$M_{s(n)} = \begin{pmatrix} 1 & \mathbf{m_{1,j_1}} & \ldots & \mathbf{m_{1,j_n}} \\ \vdots & \vdots & & \vdots \\ 1 & \mathbf{m_{l,j_1}} & \ldots & \mathbf{m_{l,j_n}} \end{pmatrix} \tag{18}$$

Then, $M_{s(n)} \approx M$.

Proof. $M_{s(n)}$ can be obtained by executing a series of elementary column transformations of M. Thus, there must be a full rank column transformation matrix T which make $M_{s(n)} = MT$. In another words, $M_{s(n)} \approx M$. □

At the same time, the solution of the BLSSS can be computed by *compounding* the solutions of its node matrices.

Theorem 2. *Let* $M = U^*(A, M_1, \ldots, M_n)$, *where* $A \in Z^{n\times k}, M_1 \in Z^{l_1\times k_n}, \ldots, M_n \in Z^{l_n\times k_n}$ *is a set of LSSSs,* $\mathbf{e_{1,\,i}} \in \{0, 1\}^i$ *denote a unit vector whose first entry is 1 and* $sum = \sum_{i=1}^{n} l_i$. *Assume that* $\mathbf{x}, \mathbf{x_1}, \ldots, \mathbf{x_n}$ *are the solutions of* $A^T X = \mathbf{e_{1,n}}^T, M_1^T X = \mathbf{e_{1,l_1}}^T, \ldots, M_n^T X = \mathbf{e_{1,l_n}}^T$, *respectively. Let* $\mathbf{x}^* = U^*(\mathbf{x}, \mathbf{x_1}, \ldots, \mathbf{x_n})$, *we get* $(M)^T\mathbf{x}^* = \mathbf{e_{1,sum}}$.

Proof.

$$\begin{aligned} M^T\mathbf{x}^* &= (U^*(A, M_1, \cdots, M_n))^T U^*(\mathbf{x}, \mathbf{x_1}, \cdots, \mathbf{x_n}) \\ &= (E(A, l_1, \cdots, l_n))^T (U(M_1, \cdots, M_n))^T U(\mathbf{x_1}, \cdots, \mathbf{x_n}) E(\mathbf{x}, 1, \cdots, 1) \\ &= (E(A, l_1, \cdots, l_n))^T U\left(\mathbf{e_{1,l_1}}, \cdots, \mathbf{e_{1,l_n}}\right)\mathbf{x} \\ &= \begin{pmatrix} A^T \\ O \end{pmatrix}\mathbf{x} \\ &= \mathbf{e_{1,sum}} \end{aligned}$$

□

Theorem 2 proposes an efficient way to find out valid solution of BLSSS. For instance, let $M_1 \in \mathbb{Z}^{m\times n}$ and $M_2 \in \mathbb{Z}^{m\times n}$ be a general LSSS and a BLSSS respectively where $M_1 \approx M_2$. Furthermore, $M_1' \in \mathbb{Z}^{k\times n}$ and $M_2' \in \mathbb{Z}^{k\times n}$ are the minimum authorized sub-matrix[6] of M_1 and M_2 respectively where $M_1' \approx M_2'$. It implies that M_2' is compounded by a set of non-zero node sub-matrix $N_i \in \mathbb{Z}^{k_i\times k_i}, 1 \le i \le l$ where $\sum(k_i - 1) + 1 = k$. It is clear that the computation complexity of solving $(M_1')^T\mathbf{x} = \mathbf{e}_1$ is $O(k^2)$. However, following Theorem 2, the computation complexity of solving $(M_2')^T\mathbf{x} = \mathbf{e}_1$ is $O(\sum(k_i-1)^2)$. Significantly, $\sum(k_i - 1)^2) \ll k^2$. Because the solution $\mathbf{x}$ is necessary during decrypting CP-ABE ciphertext, BLSSS provides mitigating decrypting overhead by reducing the computation complexity of solving $\mathbf{x}$. Furthermore, all of the decrypting operation can be processed by block algorithm following Theorem 2 to mitigate computation overhead.

On the other hand, the BLSSS is sparse. That makes BLSSS weak in security, it is possible to reveal the structure of access policy. For eliminating zero blocks, the BLSSS can be masked by a full rank matrix when strong security is desired. Meanwhile, the mask matrix can also expand the space of LSSS and make the LSSS more flexible. Let $M \in Z^{l\times m}$ be a BLSSS, $T \in Z^{m\times m}$ be a full rank mask matrix and $M' = MT$ be a masked BLSSS. Significantly, $M' \approx M$ and M' is eliminated zero blocks. In fact, it is advantageous to define the mask matrix as an upper triangular matrix. Firstly, the upper triangular mask matrix is provided with strong scalability. Thus, it can entirely keep the scalability of BLSSS. Secondly, the upper triangular mask matrix cost only a little storage resource. Upper triangular mask matrix can be generated by a random number generator R' and a seed $r' \in Z_p$. Thus, data owner only needs to store seed r' and the sorting of column block indexes. In brief, upper triangular mask matrix is an efficient tool to improve BLSSS property with only a little additional cost.

3.4.2. Dynamic updating of BLSSS

In a BLSSS, each pair of blocks is mutual independent, modifying one block has no effect on others. We will give the scalable functions of BLSSS in this section, arbitrary updating of BLSSS can be achieved by these functions.

Firstly, we define four scalable functions for unmasked BLSSS. The functions output non-zero adding blocks, modifying blocks and removing blocks of BLSSS. Note that, we omit the repeated rows of each block to save computation and communication cost.

$Thr(\mathbb{T}, \alpha, \Delta t)$: Let $\varphi(\alpha)$ be a t-out-of-n node of $\mathbb{T}$. The threshold t of $\varphi(\alpha)$ is changed into $t + \Delta t$ by this function. The function is discussed in two cases:

(1) Δt is positive.

$Node_t^+(node, \Delta t)$ is run to add columns of node matrix $M_{\varphi(\alpha)}$ and node vector $V_{\varphi(\alpha)}$. That implies each block $\mathbf{m_{i,\,\alpha}}$ and $V_{\varphi(\alpha)}$ are modified to a $(t + \Delta t)$ dimension vector. In this case, there is only one non-zero adding block $\Delta M_{n,\,\Delta t}$ generated by $Node_t^+(node, \Delta t)$.

(2) Δt is negative.

$Node_t^-(node, |\Delta t|)$ is run to remove the last $|\Delta t|$ columns of node matrix $M_{\varphi(\alpha)}$ and node vector $V_{\varphi(\alpha)}$. That implies each block $\mathbf{m_{i,\,\alpha}}$ in the BLSSS and random vector $V_{\varphi(\alpha)}$ are removed the last $|\Delta t|$ entries. Similarly, there is only one non-zero removed blocks $\Delta M_{n,\,\Delta t}$ generated by function $Node_t^-(node, \Delta t)$.

$Path(\mathbb{T}, \alpha, \gamma)$: A sub-tree of $\mathbb{T}$ with root $\varphi(\alpha)$ is moved to be a sub-tree of node $\varphi(\gamma)$. Let $M(\gamma) \in Z^{n'\times t'}$ be the node matrix of $\varphi(\gamma)$. Firstly, $Node_n^+\varphi(\gamma, 1)$ is run to insert a row $M(\gamma)_{n'+1}$ into $M(\gamma)$. Then, we set the node vectors $V_\alpha = (1, \mathbf{v}_{\alpha,\mathbf{1}}, \ldots, \mathbf{v}_{\alpha,\mathbf{n}})$ and $V_\gamma = (1, \mathbf{v}_{\gamma,\mathbf{1}}, \ldots, \mathbf{v}_{\gamma,\mathbf{n}})$ where

$$\mathbf{v}_{\alpha,j} = \begin{cases} \hat{M}(\varphi(j))_k & \text{if } \varphi(j) \in Anc(\varphi(\alpha)) \text{ and } I(j,\alpha) = k \\ \mathbf{o} & \text{if } \varphi(j) \notin Anc(\varphi(\alpha)) \end{cases} \tag{19}$$

[6] Minimum authorized sub-matrix denotes a full rank sub-matrix $M' \in \mathbb{Z}^{k\times n}$ of LSSS $M \in \mathbb{Z}^{m\times n} (k \le m)$ such that arbitrary sub-matrix $M'' \in \mathbb{Z}^{q\times n} (q < k)$ of M' is non-full rank.

$$\mathbf{v}_{\gamma,j} = \begin{cases} \hat{M}(\gamma)_{n'+1} & \text{if } j = \gamma \\ \hat{M}(\varphi(j))_k & \text{if } \varphi(j) \in Anc(\varphi(\gamma)) \text{ and } I(j,\gamma) = k \\ \mathbf{o} & \text{Otherwise} \end{cases} \tag{20}$$

Let $\Delta V = V_\gamma - V_\alpha$ be the modifying block. Finally,

$$\forall \delta(i) \in L(\alpha), M_i \leftarrow M_i + \Delta V \tag{21}$$

where $L(\alpha)$ denotes the set of leaves in the subtree with root $\varphi(\alpha)$.

$Add(\mathbb{T}, \alpha, sub\mathbb{T})$: A subtree $sub\mathbb{T}$ is inserted into the tree $\mathbb{T}$ with the inserted node $\varphi(\alpha)$ or $\delta(\alpha)$. Let $\mathbb{T}$ be described by matrix M, $sub\mathbb{T}$ be described by matrix ΔM and the random vector of $\mathbb{T}$ and $sub\mathbb{T}$ be expressed as V and ΔV respectively. The function is processed in two cases:

(1) The inserted node is a leaf $\delta(\alpha)$.
Assume that $\delta(\alpha)$ is associated with the row M_α of M. We calculate the updated BLSSS and random vector as follows:

$$M \leftarrow U^\dagger(M, \alpha, \Delta M), V \leftarrow (V, \Delta\hat{V})$$

In this case, there is only one non-zero adding blocks $\Delta\hat{M}$.

(2) The inserted node is a non-leaf $\varphi(\alpha)$.
Firstly, the function $Node_n^+(\varphi(\alpha), 1)$ is run to insert a row $M(\varphi(\alpha))_{n'+1}$ into $M(\varphi(\alpha))$. Then, the inserted vector is defined as $V_{add} = (1, \mathbf{v}_{add,1}, \ldots, \mathbf{v}_{add,n})$ where

$$\mathbf{v}_{add,j} = \begin{cases} \hat{M}(\varphi(\alpha))_{n'+1} & \text{if } j = \alpha \\ \hat{M}(\varphi(j))_k & \text{if } \varphi(j) \in Anc(\varphi(\alpha)) \text{ and } I(j,\alpha) = k \\ \mathbf{o} & \text{Otherwise} \end{cases} \tag{22}$$

Then, V_{add} is inserted into M:

$$M \leftarrow \begin{pmatrix} M \\ V_{add} \end{pmatrix}$$

Finally, the updating LSSS and random vector are calculated as follows:

$$M \leftarrow U^\dagger(M, n+1, \Delta M), V \leftarrow (V, \Delta\hat{V})$$

where n denotes the number of row of the original M. There are two adding blocks V_{add} and $\Delta\bar{M}$ generated in this case.

$Remove(\mathbb{T}, \gamma)$: A subtree $\mathbb{T}_\gamma$ with root $\varphi(\gamma)$ is removed from $\mathbb{T}$. Firstly, $\forall M_\alpha$ is removed from the BLSSS M where $\delta(\alpha) \in L(\varphi(\gamma))$. Then, $\forall \mathbf{m_{i,j}}$, $\varphi(j) \in N(\mathbb{T}_\gamma)$ are removed, which are all zero-block in the rest rows. Thus, there is no non-zero removing block generated by this function. Finally, the random vector V is removed all the node vector V_j, $\varphi(j) \in N(\varphi(\gamma)$ at the end.

Similarly, we define four scalable functions Thr_M, $Path_M$, Add_M and $Remove_M$ for the masked BLSSS.

$Thr_M(\mathbb{T}, \alpha, \Delta t)$: Let access policy $\mathbb{T}$ be described by matrix $M' = MT$ where M is a unmasked BLSSS generated by Algorithm 2 and T is an upper triangle mask matrix. This function is also discussed in two cases.

(1) $\Delta t > 0$.
Let M, T and M' be viewed as following block matrices

$$M = \begin{pmatrix} M_{u_1} & M_{u_2} \\ M_{\alpha_1} & M_{\alpha_2} \\ M_{d_1} & M_{d_2} \end{pmatrix} \tag{23}$$

$$T = \begin{pmatrix} T_1 & P \\ O & T_2 \end{pmatrix} \tag{24}$$

$$M' = \begin{pmatrix} M'_{u_1} & M'_{u_2} \\ M'_{\alpha_1} & M'_{\alpha_2} \\ M'_{d_1} & M'_{d_2} \end{pmatrix} \tag{25}$$

where $(M_{\alpha_1}, M_{\alpha_2})$ is the sub-matrix with non-zero blocks $\mathbf{m_{i,\alpha}}$, and $M_{u_1}, M_{\alpha_1}, M_{d_1}$ are the submatrices with $\mathbf{m_{i,\alpha}}$ as the last column blocks. Firstly, $Thr(\mathbb{T}, \alpha, \Delta t)$ is run and M is modified as M_{thr}:

$$M_{thr} = \begin{pmatrix} M_{u_1} & O & M_{u_2} \\ M_{\alpha_1} & \Delta M_\alpha & M_{\alpha_2} \\ M_{d_1} & O & M_{d_2} \end{pmatrix} \tag{26}$$

where ΔM_α is the non-zero adding block of M. Then T is modified as T_{Thr}:

$$T_{thr} = \begin{pmatrix} T_1 & \Delta P_1 & P \\ O & \Delta T & \Delta P_2 \\ O & O & T_2 \end{pmatrix} \tag{27}$$

where $\Delta T \in Z^{\Delta t \times \Delta t}$ is a full rank upper triangular matrix and ΔP_1, ΔP_2 are random matrix without zero entities. Finally, we get the updated BLSSS M'_{thr} as follows:

$$\begin{aligned} M'_{thr} &= M_{thr} T_{thr} \\ &= \begin{pmatrix} M'_{u_1} & M_{u_1}\Delta P_1 & M'_{u_2} \\ M'_{\alpha_1} & M_{\alpha_1}\Delta P_1 + \Delta M \Delta T & M'_{\alpha_2} + \Delta M \Delta P_2 \\ M'_{d_1} & M_{d_1}\Delta P_1 & M'_{d_2} \end{pmatrix} \end{aligned} \tag{28}$$

There are one adding block A and one modifying block B shown as follows:

$$A = \begin{pmatrix} M_{u_1}\Delta P_1 \\ M_{\alpha_1}\Delta P_1 + \Delta M \Delta T \\ M_{d_1}\Delta P_1 \end{pmatrix} \tag{29}$$

$$B = \Delta M \Delta P_2 \tag{30}$$

2) $\Delta t < 0$.

Similarly, original matrix M and T are expressed as follows

$$M = \begin{pmatrix} M_{u_1} & O & M_{u_2} \\ M_{\alpha_1} & \Delta M_\alpha & M_{\alpha_2} \\ M_{d_1} & O & M_{d_2} \end{pmatrix} \tag{31}$$

$$T = \begin{pmatrix} T_1 & \Delta P_1 & P \\ O & \Delta T & \Delta P_2 \\ O & O & T_2 \end{pmatrix} \tag{32}$$

Furthermore, the BLSSS of $\mathbb{T}$ is described as follows:

$$\begin{aligned} M' &= MT \\ &= \begin{pmatrix} M'_{u_1} & \Delta M'_u & M'_{u_2} \\ M'_{\alpha_1} & \Delta M'_\alpha & M'_{\alpha_2} \\ M'_{d_1} & \Delta M'_d & M'_{d_2} \end{pmatrix} \\ &= \begin{pmatrix} M_{u_1}T_1 & M_{u_1}\Delta P_1 & M_{u_1}P + M_{u_2}T_2 \\ M_{\alpha_1}T_1 & M_{\alpha_1}\Delta P_1 + \Delta M_\alpha \Delta T & M_{\alpha_1}P + \Delta M_\alpha \Delta P_2 + M_{\alpha_2}T_2 \\ M_{d_1}T_1 & M_{d_1}\Delta P_1 & M_{d_1}P + M_{d_2}T_2 \end{pmatrix} \end{aligned}$$

Firstly, $Thr(\mathbb{T}, \alpha, \Delta t)$ is run and M, T are modified as following matrices M_{thr}, T_{thr}:

$$M_{thr} = \begin{pmatrix} M_{u_1} & M_{u_2} \\ M_{\alpha_1} & M_{\alpha_2} \\ M_{d_1} & M_{d_2} \end{pmatrix} \tag{33}$$

$$T_{thr} = \begin{pmatrix} T_1 & P \\ O & T_2 \end{pmatrix} \tag{34}$$

Then, the updated M'_{thr} is calculated as follows:

$$M'_{thr} = \begin{pmatrix} M'_{u_1} & M'_{u_2} \\ M'_{\alpha_1} & M'_{\alpha_2} - \Delta M_\alpha \Delta P_2 \\ M'_{d_1} & M'_{d_2} \end{pmatrix} \tag{35}$$

There are one removing block A and one modifying block B given as follows:

$$A = \begin{pmatrix} \Delta M_u \\ \Delta M_\alpha \\ \Delta M_d \end{pmatrix} \tag{36}$$

$$B = \Delta M_\alpha \Delta P_2 \tag{37}$$

$Path_M(\mathbb{T}, \alpha, \gamma)$: Let access policy $\mathbb{T}$ be described by matrix $M' = MT$. $Path(\mathbb{T}, \alpha, \gamma)$ is run to modify matrix M and outputs increment vector ΔV. Then, the masked increment vector is given as $\Delta V' = \Delta VT$. Finally,

$$\forall \rho(i) \in L(\alpha), M'_i \leftarrow M'_i + \Delta V' \tag{38}$$

$\Delta V'$ is the only modifying block generated by this function.

$Add_M(\mathbb{T}, \alpha, sub\mathbb{T})$: Let access policy $\mathbb{T}$ be described by BLSSS $M' = MT \in Z^{l\times m}$ and the adding sub-tree $sub\mathbb{T}$ is described by BLSSS $\Delta M' = \Delta M \Delta T \in Z^{\Delta l\times \Delta m}$. The function is discussed in following two cases.

(1) The inserted node is a leaf $\rho(\alpha)$.

Let M, M' be described as following block matrices:

$$M = \begin{pmatrix} M_u \\ M_\alpha \\ M_d \end{pmatrix} \tag{39}$$

$$M' = \begin{pmatrix} M_u T \\ M_\alpha T \\ M_d T \end{pmatrix} = \begin{pmatrix} M'_u \\ M'_\alpha \\ M'_d \end{pmatrix} \tag{40}$$

where M_α, M'_α denote the vector representative node $\varphi(\alpha)$ in M and M', respectively. Firstly, $Add(\mathbb{T}, \alpha, sub\mathbb{T})$ is run to modify M and corresponding random vector V. Then, we obtain following updated BLSSS, random vector and mask matrix:

$$M_{add} = \begin{pmatrix} M_u & O \\ \varepsilon M_\alpha & \Delta\hat{M} \\ M_d & O \end{pmatrix} \tag{41}$$

$$V_{Add} = (V, sub\hat{V}) \tag{42}$$

$$T_{add} = \begin{pmatrix} T & P \\ O & \Delta\tilde{T} \end{pmatrix} \tag{43}$$

where $\varepsilon = (1, \ldots, 1)^T$ and $\Delta\tilde{T}$ denotes the mask matirx ΔT of ΔM removed the first column and first row. As a result, the updated matrix M'_{add} is calculated as follows:

$$\begin{aligned} M'_{add} &= M_{add} T_{add} \\ &= \begin{pmatrix} M'_u & M_u P \\ \varepsilon M'_\alpha & \varepsilon M_\alpha P + \Delta\hat{M}\Delta\tilde{T} \\ M'_d & M_d P \end{pmatrix} \end{aligned} \tag{44}$$

Note that, the adding block generated in this case is shown as follows:

$$\begin{pmatrix} M_u P \\ \varepsilon M_\alpha P + \Delta\hat{M}\Delta\tilde{T} \\ M_d P \end{pmatrix} \tag{45}$$

It is important that, $\Delta\hat{M}\Delta\tilde{T} = \Delta\hat{M}' - \Delta\bar{M}\Delta\dot{T}$ where vector $\Delta\dot{T}$ denotes the first row of T removed first entity. Following the definition of our BLSSS, $\Delta\bar{M} = (1, 1, \ldots, 1)^T$. Thus, $\Delta\hat{M}\Delta\tilde{T} = \Delta\hat{M}' - \varepsilon\Delta\dot{T}$. That is an efficient way to calculate adding block.

(2) The inserted node is non-leaf $\varphi(\alpha)$.

We run $Add(\mathbb{T}, \alpha, sub\mathbb{T})$ to output modified BLSSS M_{add} and random vector V_{add}:

$$M_{add} \leftarrow U^\dagger(\begin{pmatrix} M \\ V \end{pmatrix}, n' + 1, \Delta M), V_{add} \leftarrow (V, sub\bar{V})$$

where V is the inserted vector generated in $Add(\mathbb{T}, \alpha, sub\mathbb{T})$. Similarly, we get

$$M_{add} = \begin{pmatrix} M & O \\ \varepsilon V_{add} & \Delta\hat{M} \end{pmatrix} \tag{46}$$

$$T_{add} = \begin{pmatrix} T & P \\ O & \Delta\tilde{T} \end{pmatrix} \tag{47}$$

where $P \in Z^{l\times\Delta l}$ is a random non-zero matrix. Thus, we calculate the updated matrix M'_{add} as follows:

$$M'_{add} = \begin{pmatrix} MT & MP \\ \varepsilon V_{add} T & \varepsilon V_{add} P + \Delta\hat{M}\Delta\tilde{T} \end{pmatrix} \tag{48}$$

There are two adding blocks generated as follows:

$$(V_{add}T, V_{add}P) \tag{49}$$

$$\begin{pmatrix} MP \\ \Delta\hat{M}\Delta\tilde{T} \end{pmatrix}. \tag{50}$$

$Remove_M(\mathbb{T}, \gamma)$: Let access policy $\mathbb{T}$ be described by BLSSS $M' = MT$ and $\varphi(\gamma)$ be the root of the removed subtree. M, T can be expressed as following block matrices:

$$M = \begin{pmatrix} M_u & O & subM_u \\ M_\gamma & subM_\gamma & O \\ M_d & O & subM_d \end{pmatrix} \tag{51}$$

$$T = \begin{pmatrix} T_u & P_{u_1} & P_{u_2} \\ O & T_\gamma & P_\gamma \\ O & O & T_d \end{pmatrix} \tag{52}$$

where $subM_\gamma$ is the block of M describes the removing subtree. We get

$$\begin{aligned} M' &= \begin{pmatrix} M_uT_u & M_uP_{u_1} & M_uP_{u_2} + subM_uT_d \\ M_\gamma T_u & M_\gamma P_{u_1} + subM_\gamma T_\gamma & M_\gamma P_{u_2} + subM_\gamma P_\gamma \\ M_dT_u & M_dP_{u_1} & M_dP_{u_2} + subM_dT_d \end{pmatrix} \\ &= \begin{pmatrix} M'_{u_1} & M'_{u_2} & M'_{u_3} \\ M'_{\gamma_1} & M'_{\gamma_2} & M'_{\gamma_3} \\ M'_{d_1} & M'_{d_2} & M'_{d_3} \end{pmatrix} \end{aligned} \tag{53}$$

Firstly, $Remove(\mathbb{T}, \gamma)$ is run, the modified BLSSS M_{rmv} and mask matrix T_{rmv} are given as follows:

$$M_{rmv} = \begin{pmatrix} M_u & subM_u \\ M_d & subM_d \end{pmatrix} \tag{54}$$

$$T_{rmv} = \begin{pmatrix} T_u & P_{u_2} \\ O & T_d \end{pmatrix} \tag{55}$$

Finally, we calculate M'_{rmv} as follows:

$$\begin{aligned} M'_{rmv} &= M_{rmv}T_{rmv} \\ &= \begin{pmatrix} M'_{u_1} & M'_{u_3} \\ M'_{d_1} & M'_{d_3} \end{pmatrix} \end{aligned} \tag{56}$$

At the same time, the random vector V is removed all the blocks V_j, $\varphi(j) \in N(\gamma)$. The removing block generated by this function is given as follows:

$$\begin{pmatrix} M'_{u_2} \\ M'_{d_2} \end{pmatrix} \tag{57}$$

4. Scalable ciphertext-policy attribute-based encryption scheme

Comparing with existing ABE schemes, SCP-ABE scheme provides optimal policy updating interface. The policy scalability of SCP-ABE is greatly improved by using BLSSS as access policy. During policy updating, data owner calls scaling function of BLSSS to get minimum ciphertext increment set for updating. Thus, the policy updating operation of SCP-ABE is more efficient than other schemes.

Furthermore, as shown in Fig. 5, the system model of SCP-ABE scheme consists of four entities:

Authority. Authority is globally trusted in the system and is responsible for managing public key(PK), secret key(SK) and master key(MK). Additionally, Authority is responsible for dealing intermediate set $\{\Delta C'_t\}$ and generating ciphertext increments $\{\Delta C_t, \Delta\bar{C}_t\}$ during policy updating.

Cloud. Cloud provides data storage and access server to clients. The other important role of the cloud is the updating agent, it is responsible for updating stored ciphertext during policy updating.

Owner. Owners upload ciphertext CT to the cloud and generate the parameter set $\{\Delta_i, \Theta_i\}$ to the authority during policy updating.

User. Users access ciphertext CT stored in cloud and recover CT by their SK.

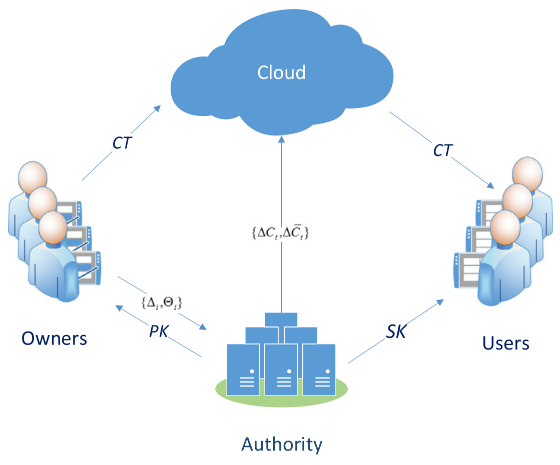


Fig. 5. System Model.

Meanwhile, we define the formalized system model of SCP-ABE as follows five functional modules:

Setup. Authority takes a security parameter as input and outputs PK, MK. MK is only held by Authority and PK is published to all entities.

KeyGen. A user submits an attribute set to Authority and the Authority replies corresponding SK to the user.

Encrypt. A data owner takes a plaintext, an access policy, PK as inputs and outputs corresponding ciphertext.

Decrypt. A user takes a ciphertext and its SK as inputs to recover plaintext. The corresponding plaintext can be recovered iff its attribute set is an authority set of the ciphertext access policy.

Update. Data owner takes an original access policy, an updated access policy as inputs and outputs corresponding increment set of updated ciphertext. In this modules, data owner invokes the scaling functions of BLSSS to generate non-zero added blocks, modified blocks and moved blocks. Such non-zero blocks is necessary for calculating ciphertext increment set.

4.1. Bilinear map

Bilinear maps, as defined following, play a crucial role in ABE.

Definition 4 (Bilinear Map)**.** Assume G_0, G_T are two multiplicative cyclic groups with prime order p, and g is a generator of G_0. Function e: $G_0 \times G_0 \to G_T$ is a bilinear map if e satisfies three criteria:

(1) bilinearity: $\forall u,\ v \in G_0$ and $a,\ b \in Z_p$, $e(u^a, v^b) = e(u, v)^{ab}$;
(2) non-degeneracy: $\forall u,\ v \neq g^0$, $e(u,\ v) \neq 1$;
(3) computability: e can be computed efficiently.

Additionally, we defined some matrix and vector operations in bilinear group G_0. Let g be a generator of G_0 and $M = (m_{i,j}) \in Z^{l\times n}$, g^M denotes a matrix in G_0 :

$$g^M = \begin{pmatrix} g^{m_{1,1}} & \dots & g^{m_{1,n}} \\ \vdots & & \vdots \\ g^{m_{l,1}} & \dots & g^{m_{l,n}} \end{pmatrix} \tag{58}$$

Similarly, Let $\mathbf{v} = (v_1, \dots, v_n) \in Z^n$ and $\mathbf{g} = (g_1, \dots, g_n) \in G_0^n$, we define that $\mathbf{g}^{\mathbf{v}} = \prod_{i=1}^{n} g_i^{v_i}$.

Furthermore, there are some *hard problems* (such as q-Parallel Bilinear Diffie–Hellman Exponent Assumption, q-parallel BDHE) support the security of ABE mechanism.

Assumption 1 (q-parallel BDHE)**.** Let G_0, G_T be two groups with prime order p and e: $G_0 \times G_0 \to G_T$ be a bilinear map. Let $a, b_1, \dots, b_q, s \in Z_p$ be chosen at random and g be a generator of G_0. If a probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ is given following tuple:

$$\begin{aligned} \vec{y} = \{&g, g^s, g^a, \dots, g^{a^q}, g^{a^{q+2}}, \dots, g^{a^{2q}}; \\ \forall 1 \leq j \leq q,\ &g^{sb_j}, g^{a/b_j}, \dots, g^{a^q/b_j}, g^{a^{q+2}/b_j}, \dots, g^{a^{2q}/b_j}; \\ \forall 1 \leq j, l \leq q, l \neq j,\ &g^{asb_l/b_j}, \dots, g^{a^q sb_l/b_j}\} \end{aligned} \tag{59}$$

It must be hard to distinguish a valid element $T_0 = e(g, g)^{a^{q+1}s}$ from a random element $T_1 \in G_T$.

Assume that, $\mathcal{B}$ is a PPT algorithm with output $z\in\{0,1\}$, it has advantage ϵ in solving q-parallel BDHE if

$$|Pr[\mathcal{B}(\vec{y},T=T_0)=0]-Pr[\mathcal{B}(\vec{y},T=T_1)=0]|\geq\epsilon \tag{60}$$

The decisional q-parallel BDHE assumption holds if no PPT algorithm $\mathcal{B}$ has a non-negligible advantage in solving the q-parallel BDHE problem.

4.2. Functional modules

The SCP-ABE scheme includes five functional modules: *Setup, KeyGen, Encrypt, Decrypt* and *Update*.

Setup(1^l): Authority takes security parameter l as input and generates a tuple $\{p, G_0, G_T, e\}$ as public parameters where p denotes a larger prime number, G_0, G_T are two p-order groups and e: $G_0\times G_0\to G_T$ is a bilinear map. Then, it picks g_1, $g_2\in G_0$, w, γ_1, $\gamma_2\in Z_p$ at random and sets $\gamma=\gamma_1\gamma_2$. Let $S=\{A_1,\dots,A_n\}$ be an attribute set defined by authority and $s=\{a_1,\dots,a_n\}\subset Z_p$ be the image set of S. Finally, PK and MK are generated as follows:

$$PK=\{Y=e(g_1,g_2)^w,g_2,\quad P'=g_1,P''=g_1^{\gamma},\quad P_t=g_1^{\gamma_1},P_i=g_1^{\gamma a_i},1\leq i\leq n\} \tag{61}$$

$$MK=\{w,\gamma_1,\gamma_2,\gamma,a_i,1\leq i\leq n\} \tag{62}$$

KeyGen(U_s, MK): Let $U_s\subset S$ be the attribute set of user. Authority picks $p_u\in G_0$ randomly and calculates SK:

$$SK_u=\{D'=g_2^{\omega}p_u^{-\gamma},D''=p_u,D_i=p_u^{\gamma a_i},A_i\in U_s\} \tag{63}$$

$Encrypt(\mu,M(\mathbb{T}),\rho,PK)$: Let μ be the plaintext and $M(\mathbb{T})\in Z_p^{l\times m}$ be the BLSSS describing access policy $\mathbb{T}$. Owner calculates $C_0=\mu Y^s$ and $C'=P'^s$ where $s\in Z_p$ is chosen at random. Then, he picks a vector $V=(s,v_2,\dots,v_m)$ and calculates $C_k=P_{\rho(k)}^{r_k}P''^{s_k},\bar{C}_k=P'^{r_k}$ where $s_k=V(M(\mathbb{T})_k)^T$, $r_k\in Z_p$ and $M(\mathbb{T})_k$ denotes the ith row of $M(\mathbb{T})$. Finally, the ciphertext is given as follows:

$$CT=\{C_0,C',C_k,\bar{C}_k,1\leq i\leq l\} \tag{64}$$

$Decrypt(C,M(\mathbb{T}),SK)$: Let $\sum_{\rho(k)\in I_u}u_kM(\mathbb{T})_k=(1,0,\dots,0)$ where $I_u\subset U_s$ and $u_k\in Z$. The plaintext is recovered as follows:

$$\mu=\frac{C_0}{e(C',D')}\prod_{\rho(k)\in I_u}\left(\frac{e(\bar{C}_k,D_{\rho(k)})}{e(C_k,D'')}\right)^{u_k} \tag{65}$$

In fact, $e(\bar{C}_k,D_{\rho(k)})/e(C_k,D'')=e(g_1,p_u)^{-\gamma s_k}$. Furthermore,

$$\begin{aligned}\prod_{\rho(k)\in I_u}\left(\frac{e(\bar{C}_k,D_{\rho(k)})}{e(C_k,D'')}\right)^{u_k}&=\prod_{\rho(k)\in I_u}e(g_1,p_u)^{-\gamma s_k r_k}\\&=e(g_1,p_u)^{-\gamma V\sum_{\rho(k)\in I_u}u_kM(\mathbb{T})_k}\\&=e(g_1,p_u)^{-\gamma s}\\\frac{C_0}{e(C',D')}&=\frac{\mu e(g_1,g_2)^{sw}}{e(g_1^s,g_2^wp_u^{-\gamma})}\\&=\mu e(g_1,p_u)^{s\gamma}\end{aligned}$$

Thus, equation (65) holds.

$Update(\mathbb{B}_1,\mathbb{B}_2)$: Let $\mathbb{B}_1=\{M_1,\dots,M_{n_1}\},\mathbb{B}_2=\{R_1,\dots,R_{n_2}\}$ be the non-zero adding/modifying block set and removing block set generated by scaling functions, $\mathbb{V}_1=\{\mathbf{v}_1,\dots,\mathbf{v}_{n_1}\},\mathbb{V}_2=\{\mathbf{w}_1,\dots,\mathbf{w}_{n_2}\}$ denote the corresponding blocks in random vector V. Assume that $M_i\in Z^{l_i\times m_i}$ and $R_i\in Z^{l'_i\times m'_i}$. Then, owner calculates

$$\Delta_i=\begin{cases}P_t^{\mathbf{v}_i} & l_i\geq m_i\\P_t^{M_i\mathbf{v}_i^T} & \text{Otherwise}\end{cases} \tag{66}$$

$$\Theta_i=\begin{cases}P_t^{\mathbf{w}_i} & l'_i\geq m'_i\\P_t^{R_i\mathbf{w}_i^T} & \text{Otherwise}\end{cases} \tag{67}$$

where Δ_i, Θ_i are vectors in G_0. $\mathbb{B}_1$, $\mathbb{B}_2$, $\Delta=\{\Delta_i,1\leq i\leq n_1\}$, $\Theta=\{\Theta_i,1\leq i\leq n_2\}$ are sent to authority. Let ϕ_i, τ_i denote the map from row label of M_i and R_i to the row label of the final BLSSS, respectively. For the adding/modifying blocks, the authority calculates

$$\Delta C'_{t,i} = \begin{cases} \Delta_i^{M_{i,j}} & l_i \geq m_i \\ \Delta_{i,j} & \text{Otherwise} \end{cases} \tag{68}$$

$$\Theta C'_{t,i} = \begin{cases} \Theta_i^{R_{i,j}} & l'_i \geq m'_i \\ \Theta_{i,j} & \text{Otherwise} \end{cases} \tag{69}$$

where $t = \phi_i(j)$ or $t = \tau_i(j)$, $M_{i,\,j}$ and $R_{i,\,j}$ denote the jth rows of M_i and R_i respectively, $\Delta_{i,\,j}$ and $\Theta_{i,\,j}$ denote the jth entity of Δ_i and Θ_i respectively,. Then, the increments of attribute ciphertext $C_t, \bar{C}_t$ are calculated as follows:

$$\Delta C_t = \left(\prod_{\phi_i(j)=t} \Delta C'_{t,i}\right)^{\gamma_2} \left(\prod_{\tau_i(j)=t} \Theta C'_{t,i}\right)^{-\gamma_2} (P_{\rho(k)})^{\Delta r_t} \tag{70}$$

$$\Delta \bar{C}_t = (P')^{\Delta r_t} \tag{71}$$

where $\Delta r_t \in Z$ is chosen randomly. Then, increment set $\Omega = \{\Delta C_t, \Delta \bar{C}_t | \exists \phi_i(j) = t \vee \tau_i(j) = t\}$ is sent to cloud and attribute ciphertext is updated as follows:

$$C_t \leftarrow C_t \Delta C_{t,i}, \bar{C}_t \leftarrow \bar{C}_t \Delta \bar{C}_{t,i}$$

If C_t and $\bar{C}_t$ are null, they were initialled as $C_t = \bar{C}_t = 1$.

Let $\Delta s_{t,i} = M_{i,j}\mathbf{v}_i^T$ and $\Theta s_{t,i} = R_{i,j}\mathbf{w}_i^T$ denote the increment of shares s_t caused by block M_i and R_i respectively, where $t = \phi_i(j)$, $t = \tau_i(j)$. Thus, s_t is set to be $s_t + \sum \Delta s_{t,i} - \sum \Theta s_{t,i}$ after updating. It is clear that

$$\Delta C_t = \prod_{\phi_i(j)=t} g_1^{\gamma \Delta s_{t,i}} \prod_{\tau_i(j)=t} g_1^{-\gamma \Theta s_{t,i}} (P_{\rho(t)})^{\Delta r_t}$$
$$\Delta \bar{C}_t = (P')^{\Delta r_t}$$

Finally, the updated ciphertext can be shown as follows

$$C_t = g_1^{\gamma(s_t + \sum_{\phi_i(j)=t} \Delta s_{t,i} - \sum_{\tau_i(j)=t} \Theta s_{t,i})} P_{\rho(t)}^{r_t + \Delta r_t}$$
$$\bar{C}_t = (P')^{r_t + \Delta r_t}$$

Thus, the updating is correct.

4.3. Security proof

The formalized security model of SCP-ABE is given as following interactions between adversary $\mathcal{A}$ and simulator $\mathcal{B}$.

Init: $\mathcal{A}$ submits an original challenge policy (M, ρ) and an updated challenge policy (M', ρ') to $\mathcal{B}$.

Setup: $\mathcal{B}$ generates PK and sends to $\mathcal{A}$.

Phase 1: $\mathcal{A}$ submits an attribute set U_s to $\mathcal{B}$. $\mathcal{B}$ replies corresponding SK to $\mathcal{A}$ iff U_s is unauthorized set of (M, ρ) and (M', ρ').

Challenge 1: $\mathcal{A}$ submits two messages μ_0 and μ_1 to $\mathcal{B}$. $\mathcal{B}$ picks $v \in \{0, 1\}$ at randomly and generates ciphertext $CT = Encrypt(\mu_v, M, \rho, PK)$ for $\mathcal{A}$

Challenge 2: $\mathcal{B}$ generates increment set Ω between $CT = Encrypt(\mu_v, M', \rho', PK)$ and CT . Then, Ω is sent to $\mathcal{A}$

Phase 2: Phase 1 is repeated.

Guess: $\mathcal{A}$ give the guess v' of v.

The security of our scheme is proved in following theorem.

Theorem 3. *Our SCP-ABE scheme is secure in the generic bilinear group and random oracle model while there is no PPT adversary getting non-negligible advantage in Assumption 1.*

Proof. Assume we have an adversary $\mathcal{A}$ with non-negligible advantage $Adv_{\mathcal{A}} = \epsilon$ in the security model against our SCP-ABE scheme. Then, we can build a simulator $\mathcal{B}$ which plays the decisional q-BDHE problem with non-negligible advantage ϵ as follows.

Init: $\mathcal{B}$ gets the q-BDHE challenge $\{\vec{y}, T_b\}$ as given in Assumption 1. $\mathcal{A}$ submits the original policy (M, ρ) (corresponding to tree-circuit $\mathbb{T}$) and updated policy (M', ρ') (corresponding to tree-circuit $\mathbb{T}'$) to $\mathcal{B}$. For describing simple, $\bar{M} = [\mathbf{v}_{i,j}]$ and $\bar{M}' = [\mathbf{v}'_{i,j}]$ are constructed where

$$\mathbf{v}_{i,j} = \begin{cases} 1, & i=0, j=0 \\ O_{1\times t_j}, & \varphi(j) \in \mathbb{T}' - \mathbb{T} \vee \delta(i) \in \mathbb{T}' - \mathbb{T} \\ \mathbf{m}_{i,j}, & \varphi(j), \delta(i) \in \mathbb{T} \wedge t_j \geq t'_j \\ (\mathbf{m}_{i,j}, O_{1\times(t'_j - t_j)}), & \varphi(j), \delta(i) \in \mathbb{T} \wedge t_j < t'_j \end{cases}$$

$$\mathbf{v}'_{i,j} = \begin{cases} 1, & i=0, j=0 \\ O_{1\times t_j}, & \varphi(j) \in \mathbb{T} - \mathbb{T}' \vee \delta(i) \in \mathbb{T} - \mathbb{T}' \\ \mathbf{m}_{i,j}, & \varphi(j), \delta(i) \in \mathbb{T}' \wedge t'_j \geq t_j \\ (\mathbf{m}_{i,j}, O_{1\times(t_j - t'_j)}), & \varphi(j), \delta(i) \in \mathbb{T}' \wedge t'_j < t_j \end{cases}$$

t_j and t'_j describe the threshold of $\varphi(j)$ in $\mathbb{T}$ and $\mathbb{T}'$, respectively. Because of that $\bar{M}$ and $\bar{M}'$ are respectively equivalent to M and M', (M, ρ) and (M', ρ') are replied by $(\bar{M}, \bar{\rho})$ and $(\bar{M}', \bar{\rho})$ in the following proof, where $\bar{\rho}$ is the map form row-index mapping function of $\bar{M}$ and $\bar{M}'$ to attribute set. Furthermore, the size of $\bar{M}$ and $\bar{M}'$ are equal, a increment matrix can be defined as $\Delta M = \bar{M}' - \bar{M}$ which includes all non-zero adding blocks, modifying blocks and removing blocks during updating. Additionally, it is limited that d, m are less than $q/2$ while $\bar{M}, \bar{M}', \Delta M \in Z^{d\times m}$.

Setup: $\mathcal{B}$ chooses $w', \gamma'_1, \gamma'_2 \in Z_p$ and set $\gamma_1 = a\gamma'_1, \gamma_2 = \gamma'_2, \gamma' = \gamma'_1\gamma'_2, \gamma = a\gamma', w = w' + a^{q+1},\ g_1 = g_2 = g$. Thus, $P_t = (g^a)^{\gamma'_1}$, $P'' = (g^a)^{\gamma'}$ and $Y = e(g,g)^w = e(g, g^{w'})e(g^a, g^{a^q})$. Then, it sets

$$a_i = \left(a'_i + \sum_{k\in S_i}\sum_{j=1}^{m} (a^j \bar{M}_{k,j}/b_k + a^j \Delta_{k,j}/b_{d+k}) \right)/\gamma$$

$$P_i = g^{\gamma a_i} = (g^a)^{\gamma' a'_i} \prod_{k\in S_i}\prod_{n=1}^{m} g^{a^j(\bar{M}_{k,n}/b_k + \Delta_{k,n}/b_{d+k})}$$

where $S_i = \{k|\bar{\rho}(k) = A_i\}$, $M_{k,\,n}$ and $\Delta_{k,\,n}$ denote the n^{th} elements of k^{th} row of $\bar{M}$ and ΔM respectively. Note, that, $P_i = (g^a)^{\gamma' a'_i}$ iff $S_i = \varnothing$.

Phase 1: $\mathcal{B}$ replies the SK queries. Assume that, $\mathcal{B}$ is given a SK query with a unauthorized set U_s of $\bar{M}$ and $\bar{M}'$. Then, $\mathcal{B}$ picks $u' \in Z_p$ at random and finds a vector $\mathbf{w} = (\omega_1, \dots, \omega_m) \in Z_p^m$ where $\omega_1 = 1$ and $\forall k, \bar{\rho}(k) \in U_s$, $\omega\bar{M}_k = \omega\bar{M}'_k = 0$. Following the definition of LSSS, such vector ω must exist. $\mathcal{B}$ sets $u = u' + (\sum_{j=1}^m a^{q+1-j}\omega_j)/\gamma'$. Thus,

$$p_u = g^u = g^{u'} \prod_{j=1}^{m} g^{a^{q+1-j}\omega_j/\gamma'}$$

$$D' = g^{w' + a^{q+1}} p_u^{-\gamma} = g^{w'} (g^a)^{-\gamma' u'} \prod_{j=2}^{m} g^{-a^{q+2-j}\omega_j}$$

Then, $\mathcal{B}$ calculates $D_i = p_u^{\gamma a_i}$. we consider the attribute SK D_i in two cases:

(1) $S_i = \varnothing$. In this case, $a_i = a'_i/\gamma$ and $D_i = p_u^{\gamma a_i} = g^{a'_i u' + a'_i/\gamma' \sum \omega_j a^{q+1-j}}$.

(2) $S_i \neq \varnothing$. $a_i = (a'_i + \sum_{k\in S_i}\sum_{l=1}^m a^l(m_{k,l}/b_k + \Delta_{k,l}/b_{d+k}))/\gamma$, that implies

$$\begin{aligned} D_i &= g^{u\gamma a_i} \\ &= g^{\left(u' + 1/\gamma' \sum_{j=1}^m \omega_j a^{q+1-j}\right)\left(a'_i + \sum_{k\in S_i}\sum_{l=1}^m a^l\left(m_{k,l}/b_k + \Delta_{k,l}/b_{d+k}\right)\right)} \\ &= g^{u' a'_i} \prod_{j=1}^{m} \left(g^{a^{q+1-j}}\right)^{a'_i\omega_j/\gamma'} \prod_{k\in S_i}\prod_{l=1}^{m} \left(g^{a^l/b_k}\right)^{u' m_{k,l}} \left(g^{a^l/b_{d+k}}\right)^{u'\Delta_{k,l}} \cdot \\ &\quad \prod_{k\in S_i}\prod_{j=1}^{m}\prod_{l=1}^{m} g^{a^{q+1-j+l}\omega_j m_{k,l}/b_k\gamma'} g^{a^{q+1-j+l}\omega_j \Delta_{k,l}/b_{d+k}\gamma'} \end{aligned}$$

where

$$\sum_{k\in S_i}\sum_{j=1,l=j}^{m} a^{q+1-j+l}\omega_j m_{k,l}/b_k\gamma' = a^{q+1}\sum_{k\in S_i}\sum_{j=1}^{m}\omega_j m_{k,l}/b_k\gamma' = 0$$

$$\sum_{k\in S_i}\sum_{j=1,l=j}^{m} a^{q+1-j+l}\omega_j \Delta_{k,l}/b_{d+k}\gamma' = a^{q+1}\sum_{k\in S_i}\sum_{j=1}^{m}\omega_j \Delta_{k,l}/b_{d+k}\gamma' = 0$$

Thus,

$$D_i = g^{u'a'_i} \prod_{j=1}^{m} (g^{a^{q+1-j}})^{a'_i \omega_j/\gamma'} \prod_{k \in S_i} \prod_{l=1}^{m} (g^{a^l/b_k})^{u' m_{k,l}} (g^{a^l/b_{d+k}})^{u' \Delta_{k,l}} \cdot \\ \prod_{k \in S_i} \prod_{j=1}^{m} \prod_{l \neq j} g^{a^{q+1-j+l} \omega_j m_{k,l}/b_k \gamma'} g^{a^{q+1-j+l} \omega_j \Delta_{k,l}/b_{d+k} \gamma'}$$

Because the unknown term $g^{a^{q+1}}$ is canceled, we can calculate D' and D_j easily.

Challenge 1: $\mathcal{B}$ builds the challenge ciphertext. $\mathcal{A}$ give two messages μ_0, μ_1 to $\mathcal{B}$. It creates $C = \mu_v Te(g^s, g^{\omega'})$ and $C' = g^s$, where $v \in \{0, 1\}$ and $Pr[v = 0] = Pr[v = 1] = 1/2$. Then, $\mathcal{B}$ chooses random $v_2, \ldots, v_m \in Z_p$ and generates vector

$$V = (s, sa + v_2, \ldots, sa^{m-1} + v_m)$$

Thus,

$$s_k = M_k V = s \sum_{j=1}^{m} a^{j-1} m_{k,j} + \sum_{j=2}^{m} v_j m_{k,j}$$

Then, it chooses $r'_k \in Z_p$, sets $r_k = r'_k - sb_k \gamma'$ and calculates attribute ciphertext

$$\bar{C}_k = g^{r_k} = g^{r'_k} (g^{sb_k})^{\gamma'}$$

$$\begin{aligned} C_k &= g^{\gamma s_k} g^{r_k \gamma a_{\rho(k)}} \\ &= g^{r'_k a'_{\rho(k)}} \left(g^{-sb_k}\right)^{\gamma' a'_{\rho(k)}} \prod_{j=1}^{m} g^{sa^j \gamma' m_{k,j}} \prod_{j=2}^{m} (g^a)^{\gamma' v_j m_{k,j}} \prod_{l \in S_{\rho(k)}} \prod_{j=1}^{m} \left(g^{a^j/b_l}\right)^{r'_k m_{l,j}} \left(g^{a^j/b_{d+l}}\right)^{r'_k \Delta_{l,j}} \cdot \\ &\quad \prod_{l \in S_{\rho(k)}} \prod_{j=1}^{m} \left(g^{sa^j b_k/b_l}\right)^{-\gamma' m_{l,j}} \left(g^{sa^j b_k/b_{d+l}}\right)^{-\gamma' \Delta_{l,j}} \\ &= g^{r'_k a'_{\rho(k)}} \left(g^{-sb_k}\right)^{\gamma' a'_{\rho(k)}} \prod_{j=2}^{m} (g^a)^{\gamma' v_j m_{k,j}} \prod_{l \in S_{\rho(k)}} \prod_{j=1}^{m} \left(g^{a^j/b_l}\right)^{r'_i m_{k,j}} \left(g^{a^j/b_{d+l}}\right)^{r'_k \Delta_{l,j}} \cdot \\ &\quad \prod_{l \in S_{\rho(k)} - \{k\}} \prod_{j=1}^{m} \left(g^{sa^j b_k/b_l}\right)^{-\gamma' m_{k,j}} \prod_{l \in S_{\rho(k)}} \prod_{j=1}^{m} \left(g^{sa^j b_k/b_{d+l}}\right)^{-\gamma' \Delta_{l,j}} \end{aligned}$$

Challenge 2: $\mathcal{B}$ builds the updating challenge increments. For $\forall k$, $\Delta s_k = \Delta_k V = s \sum_{j=1}^{m} a^{j-1} \Delta_{k,j} + \sum_{j=2}^{m} v_j \Delta_{k,j}$ and $\Delta r_k = \Delta r'_k - sb_{d+k} \gamma'$ where $r'_k \in Z_p$ is chosen randomly. Thus, the increment of C_k and $\bar{C}_i k$ are calculated as follows:

$$\begin{aligned} \Delta\bar{C}_k &= g^{\Delta r_k} = g^{\Delta r'_k} \left(g^{sb_{d+k}}\right)^{\gamma'} \\ C_k &= g^{\gamma \Delta s_k} g^{\Delta r_k \gamma a_{\rho(k)}} \\ &= g^{\Delta r'_k a'_{\rho(k)}} \left(g^{-sb_{d+k}}\right)^{\gamma' a'_{\rho(k)}} \prod_{j=1}^{m} g^{sa^j \gamma' \Delta_{k,j}} \prod_{j=2}^{m} (g^a)^{\gamma' v_j \Delta_{k,j}} \prod_{l \in S_{\rho(k)}} \prod_{j=1}^{m} \left(g^{a^j/b_l}\right)^{\Delta r'_k m_{k,j}} \left(g^{a^j/b_{d+l}}\right)^{\Delta r'_k \Delta_{k,j}} \cdot \\ &\quad \prod_{l \in S_{\rho(k)}} \prod_{j=1}^{m} \left(g^{sa^j b_{d+k}/b_l}\right)^{-\gamma' m_{l,j}} \left(g^{sa^j b_{d+k}/b_{d+l}}\right)^{-\gamma' \Delta_{l,j}} \\ &= g^{\Delta r'_k a'_{\rho(k)}} \left(g^{-sb_{d+k}}\right)^{\gamma' a'_{\rho(k)}} \prod_{j=2}^{m} (g^a)^{\gamma' v_j \Delta_{k,j}} \prod_{l \in S_{\rho(k)}} \prod_{j=1}^{m} \left(g^{a^j/b_l}\right)^{\Delta r'_k \Delta_{k,j}} \left(g^{a^j/b_{d+l}}\right)^{\Delta r'_k \Delta_{k,j}} \cdot \\ &\quad \prod_{l \in S_{\rho(k)}} \prod_{j=1}^{m} \left(g^{sa^j b_{d+k}/b_l}\right)^{-\gamma' m_{l,j}} \prod_{l \in S_{\rho(k)}/\{k\}} \prod_{j=1}^{m} \left(g^{sa^j b_{d+k}/b_{d+l}}\right)^{-\gamma' \Delta_{l,j}} \end{aligned}$$

Finally, the updated attribute ciphertext can be directly calculated as follows:

$$\{\bar{C}_k = \bar{C}_k \Delta\bar{C}_k, C_k = C_k \Delta C_k, 1 \leq k \leq d\}$$

Phase 2: Phase 1 is repeated.

Guess: $\mathcal{A}$ outputs a guess v' of v. $\mathcal{B}$ outputs 0 to guess that $T = e(g, g)^{sa^{q+1}}$ if $v' = v$;otherwise, it outputs 0 to indicate that T is a random element in G_T. Assume that, $Adv_{\mathcal{A}} = Pr[\mathcal{A}(v) = v'] - 1/2$. If $T = e(g, g)^{sa^{q+1}}$, $\mathcal{B}$ gives a perfect simulation.

Thus,

$$\begin{aligned} Pr[\mathcal{B}(\vec{y}, T = e(g,g)^{sa^{q+1}}) = 0] &= Pr[\mathcal{A}(v) = v'] \\ &= Adv_{\mathcal{A}} + 1/2 \end{aligned}$$

If T is a random element $R \in G_T$, M_v is completely hidden from the adversary and we have

$$Pr[\mathcal{B}(\vec{y}, T = R) = 0] = 1/2.$$

Therefore, $\mathcal{B}$ can play the decisional q-BDHE game with non-negligible advantage ϵ:

$$\begin{aligned} Adv_{\mathcal{B}} &= |Pr[\mathcal{B}(\vec{y}, T = e(g,g)^{sa^{q+1}}) = 0] - Pr[\mathcal{B}(\vec{y}, T = R) = 0]| \\ &= Adv_{\mathcal{A}} \\ &= \epsilon \end{aligned}$$

As a result, our scheme is secure while Assumption 1 is holding. □

Additionally, collusion is one of the major security issue that needs to be avoid in CP-ABE scheme. The security of SCP-ABE scheme against collusion attack is proofed as follows.

Theorem 4. *Proposed SCP-ABE scheme is secure against the collusion attack.*

Proof. Each user u in the system is assigned with a global unique parameter p_u, and all of his attribute secret keys are associated with p_u. Thus, the attribute SKs from different users cannot collude to decrypt ciphertext. As a result, the collusive users acquire the access privilege no more than the sum of their privileges.

The detailed proof is implied in proof of Theorem 3. In the interactions between $\mathcal{A}$ and $\mathcal{B}$, $\mathcal{A}$ can repeated query different SKs for diverse users in Phase 2. As a result, the users can only collude to get only negligible advantage in the security model against SCP-ABE while Assumption 1 is holding. □

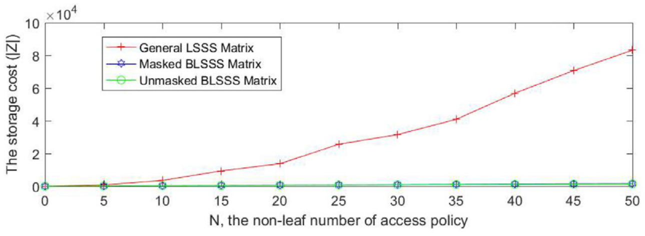
5. Performance evaluation

5.1. Performance analysis of BLSSS

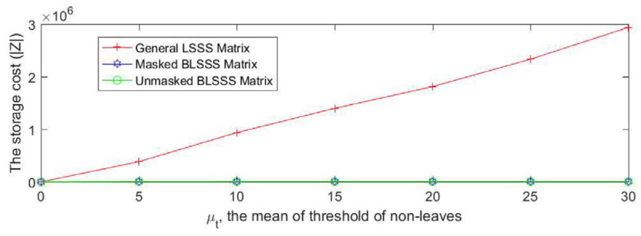
BLSSS is low-cost in storage and computation. In certain scenarios, it needs to sacrifice a little of efficiency to increase security and extend policy space by using mask matrix. However, choosing an upper triangular matrix as mask matrix is an effective way to preserve the scalability of BLSSS, the upper triangular matrix requires limited storage and computation resources during generating and updating. In order to evaluate the performances of BLSSS, we design a series of simulations and give corresponding detailed analysis in this section. Firstly, we compare the storage cost of general LSSS, unmasked BLSSS and masked BLSSS. Then, we measure the runtime of each BLSSS operating functions proposed in this paper. All of these simulations are run on a PC with an Intel Core 2 Duo CPU at 3.14GHz and 8.00 GB RAM and the simulation results are shown in Figs. 6–9.

Assume that $\mathbb{T}$ denote an access policy and non-leaf $\varphi(i) \in \mathbb{T}$ be a t_i-out-of n_i node. Furthermore, let N be the total of non-leaves of $\mathbb{T}$, t_i and n_i obey normal distribution $N(\mu_t, \delta_t)$ and $N(\mu_n, \delta_n)$ respectively. Fig. 6 describes the storage cost of various LSSS affected by N, μ_t and μ_n. In fact, all of the three parameters influence the storage size of LSSS. Firstly, N increasing incurs both the total of column and row of LSSS. Fig. 6 presents the impact of N on the storage cost of LSSS while $\mu_t = 5$ and $\mu_n = 10$. Because all matrix entities need to be stored, the space complexity of general LSSS is $O(N^2)$ in this case. Meanwhile, unmasked BLSSS only needs to store a tree-circuit, a secret and a seed, its space complexity is $O(N)$. similarly, masked BLSSS needs to store a tree-circuit, a secret, a seed and an additional column index sorting, the space complexity is also $O(N)$. Secondly, μ_t increasing only incurs the increasing of LSSS column. Fig. 6 gives the rising tendency of LSSS storage cost along with variable μ_t while $N = 50$ and $\mu_n = 40$. However, the storage cost of general LSSS approximates linear growth with μ_t, the storage cost increasing of BLSSS is negligible in this case. Thirdly, μ_n increasing only incurs the increasing of LSSS row. Fig. 6 shows the impact of variable μ_n on LSSS storage cost while $N = 50$ and $\mu_t = 5$. Significantly, storage cost increasing of general LSSS is fast than BLSSS in this case. In summary, as n, μ_t and μ_n increasing, the storage cost of general LSSS increases much faster than BLSSS, particularly faster than unmasked BLSSS. Thus, general LSSS cost much more storage resource than BLSSS. On the other hand, the masked BLSSS cost only a little of additional storage resource to storing mask matrix information. Thus, it is acceptable for thin client to use masked BLSSS as access policy.

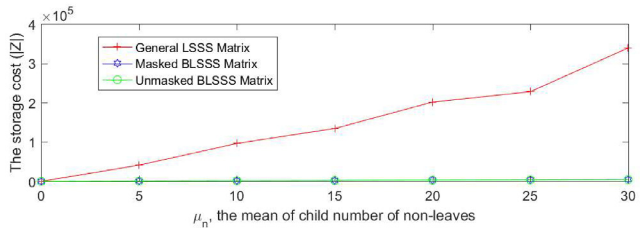
Similarly, Fig. 7 describes the time complexity of generating BLSSS influenced by different variate. Let $\mathbb{T}$ be an access tree will be transformed into LSSS, N be the number of non-leaf $\varphi(i)$ of $\mathbb{T}$ and $\varphi(i)$ be a t_i-out-of-n_i node. For comparison, we set all t_i equal and all n_i equal in simulations. Figure.7 shows the generating time increases with N while $t_i = 10$ and $n_i = 10$. Essentially, the generating process of BLSSS can be viewed as a depth-first traversal of $\mathbb{T}$. Thus, N is a critical factor impacts the time complexity of BLSSS generating. Fig. 7 shows the generating time increases with t_i while $N = 10$ and $n_i = 10$. Meanwhile, Fig. 7 shows the generating time of BLSSS increases with n_i while $N = 10$ and $t_i = 1$. Following the definition, n_i and t_i respectively decide the row number and column number of node matrix $M_{\varphi(i)}$. It implies that the time complexity of node matrix generating is $O(t_i \times n_i)$. Thus, the runtime of BLSSS generating grows linearly as t_i and n_i increases, respectively.



(a) Influence of N

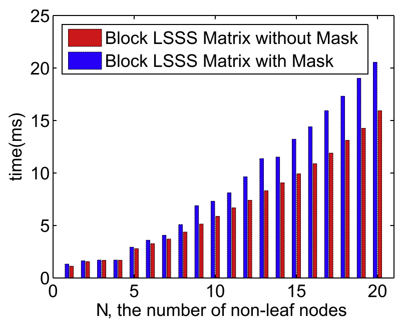


(b) Influence of t_i

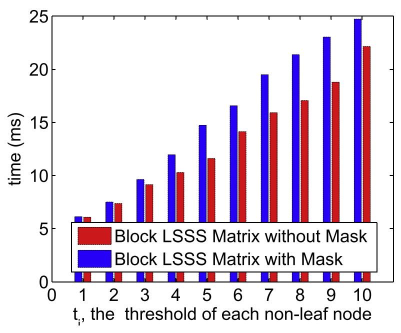


(c) Influence of n_i

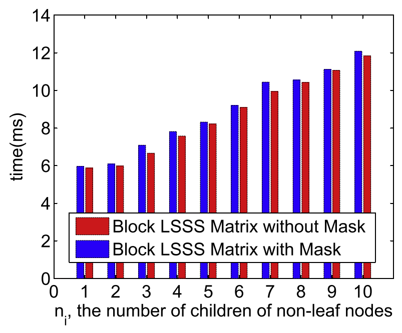
Fig. 6. Storage content size of LSSS.



(a) Influence of N

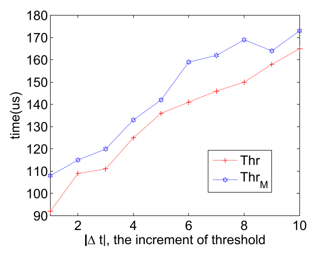


(b) Influence of t_i

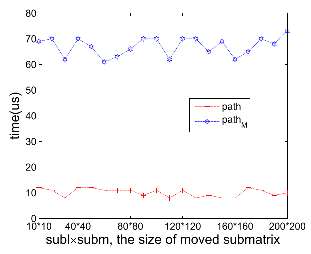


(c) Influence of n_i

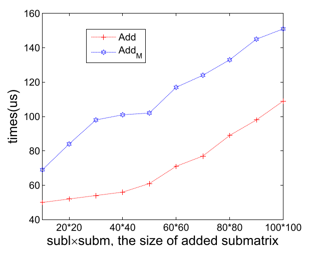
Fig. 7. Generating time of BLSSS.



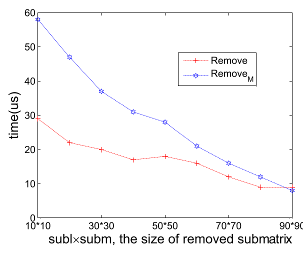
(a)



(b)



(c)



(d)

Fig. 8. Computation time of scaling functions incurred by sub-matirx size.

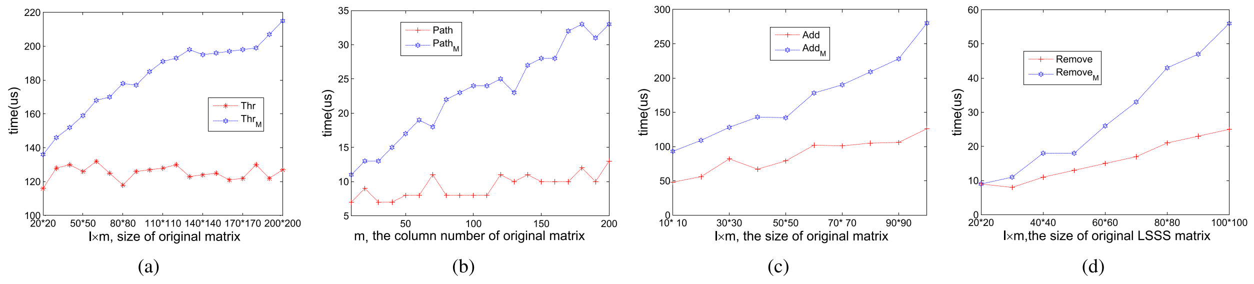


Fig. 9. Computation time of scaling functions incurred by original matrix size.

Furthermore, the three figures also show that, the mask of BLSSS only costs a little processing time, it is much less than the total generating time. Thus, masking BLSSS is an acceptable and efficient way to improve its performance. In order to discuss the efficiency of BLSSS updating, Figs. 8 and 9 are given to present the runtime of its scaling functions.

Let $M \in Z^{l \times m}$ and $subM \in Z^{subl \times subm}$ denote the original BLSSS and modifying/adding/removing sub-matrix, respectively. Fig. 8 describes the efficiency of scaling functions influenced by *subM*. Firstly, Fig. 8(a) describes the runtime of *Thr* and Thr_M increases with threshold increment $|\Delta t|$ while $l = 200$ and $m = 200$. Significantly, the runtime of *Thr* and Thr_M almost linearly increase with $|\Delta t|$. Secondly, Fig. 8(b) shows the runtime of *Path* and $Path_M$ change with the size of moved sub-matrix while $l = 200$ and $m = 200$. It is clear that the runtime of *Path* and $Path_M$ approximate constants without influence of the sub-matrix size. Thirdly, Fig. 8(c) shows the runtime increasing of *Add* and Add_M incurred by added sub-matrix size while $l = 200$ and $m = 200$. It is important that the difference between *Add* runtime and Add_M runtime increases slow with added sub-matrix size. Fourthly, Fig. 8(c) shows the runtime changing of *Remove* and $Remove_M$ incurred by size of *subM* while $l = 200$ and $m = 200$. Note that, the runtime of *Remove* and $Remove_M$ decrease while the size of *subM* increasing.

Similarly, Fig. 9 describes the runtime of scaling functions influenced by the size of original matrix *M*. Firstly, Fig. 9(a) shows the runtime of *Thr* and Thr_M increases with original matrix size where $|\Delta t| = 10$. However, the runtime of *Thr* approximates to a constant, the runtime of Thr_M increases linearly with size of *M*. Secondly, Fig. 9(b) shows the runtime of *Path* and $Path_M$ increases with *m* while $subl = 5$ and $subm = 5$. Significantly, variate *m* presents only a little influence for runtime of *Path* while the runtime of $Path_M$ is distinctly increasing with *m*. Thirdly, Fig. 9(c) shows the runtime of *Add* and Add_M increases with original matrix size while $subl = 200$ and $subm = 200$. However, the runtime of *Add* and Add_M all increase with size of *M*, the runtime of Add_M increases faster than the runtime of *Add*. Fourthly, Fig. 9(d) shows the runtime of *Remove* and $Remove_M$ increases with original matrix size while $subl = 10$ and $subm = 10$. It is clear that the runtime of *Remove* increases more mitigate than the runtime of $Remove_M$.

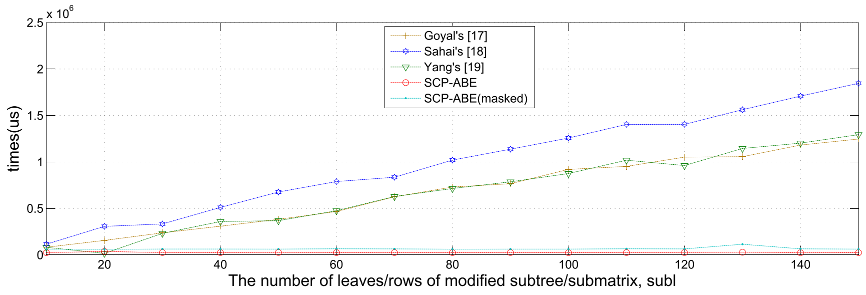
In summary, all the scaling functions are more efficient than the BLSSS generating function. More specifically, scaling functions cost two or three order of magnitude less runtime than BLSSS generating function. Furthermore, Figs. 8 and 9 are also show that the additional runtime of scaling functions caused by mask matrix is limited and acceptable.

5.2. Performance analysis of scalable ABE scheme

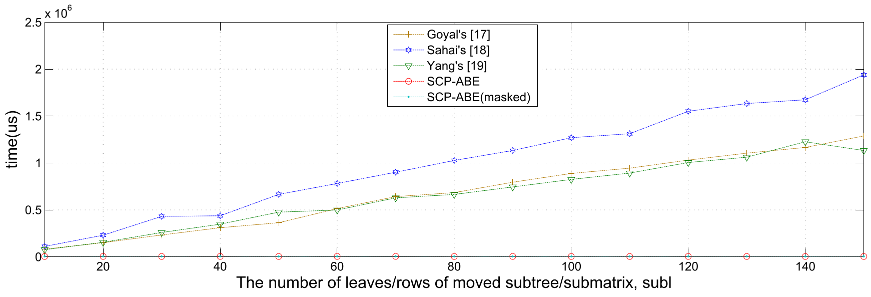
The most advantage of our SCP-ABE scheme is its manageability of access policy. Comparing with schemes [11,27,40], our scheme is more efficient on various types of policy updating operations. We also give the simulation and comparison of these schemes. The simulation is run on a Linux virtual machine with a 3.16GHz CPU and 1.00 GB RAM, PBC lab is used to simulate the bilinear group operations. In fact, the policy updating operations of scheme [27] is too complex to evaluate. In order to compare, we only consider the simplest case of this scheme in the simulation.

Fig. 10 describes the runtime of policy updating operations of various schemes cost by the data owner. Fig. 10(a) shows the runtime of modifying the threshold of a node n_j where the *x* axis represent *subl*, the leaf number of subtree with root n_j. In this situation, the runtime of our scheme is incurred by the threshold increment $|\Delta D|$ while the runtime of others is incurred by *subl*. Because $|\Delta D|$ is much less than *subl*, our scheme is more efficient than others. As shown in Fig. 10(b), our scheme takes only a little time to move a subtree of access policy. However, the runtime of this operation of SCP-ABE approximates to a small constant, the runtime of others linearly increases with *subl*. That implies our scheme cost the lowest computation in this case. Fig. 10(c) shows that the runtime of adding subtree of SCP-ABE is minimum. Because the access policy of our SCP-ABE is scalable, it can only process the added block of and ignores other blocks. That make our scheme more efficient than others. As show in Fig. 10(d), runtime of removing a subtree can be ignored in all of these schemes except of SCP-ABE scheme with masked BLSSS. Actually, only the masked BLSSS generates increment of attribute ciphertexts during removing subtree. In other schemes, there is no non-zero increment produced for valid attribute ciphertext, the access policy can be immediately updated by removing the invalid attribute ciphertexts. In summary, our SCP-ABE scheme is more efficient than others during access policy updating. However, masked BLSSS cost a little additional runtime during policy updating, SCP-ABE with masked BLSSS is more efficient than other schemes in most situation.

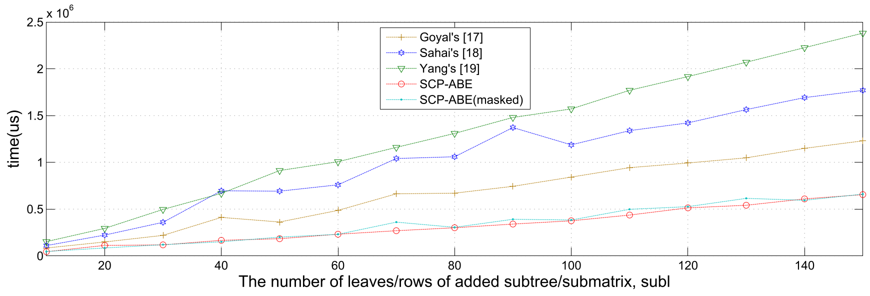
Additionally, other performances of SCP-ABE are briefly discussed. SCP-ABE can be viewed as a common ABE scheme taking BLSSS as access policy and providing efficient policy updating interface. The essential operations of SCP-ABE are only



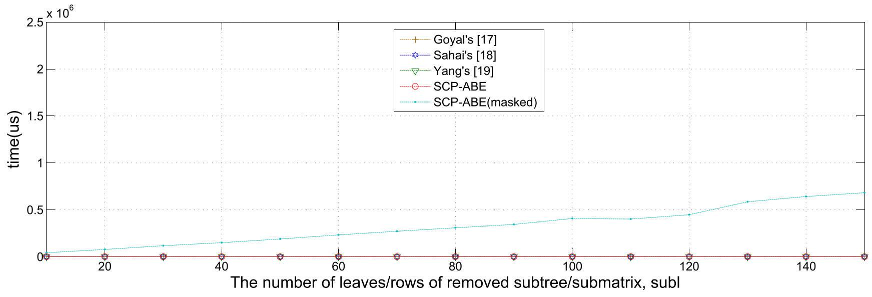
(a) Modifying of Threshold



(b) Moving of Subtree



(c) Adding of Subtree



(d) Removing of Subtree

Fig. 10. Simulation of policy updating operations.

a little different from existing ABE schemes except for policy updating. Thus, SCP-ABE only presents limited difference from other schemes in terms of ciphertext size, encryption efficiency and decryption efficiency. Firstly, because BLSSS cost less storage resource than other forms, it slightly reduces the ciphertext size of SCP-ABE. However, comparing with the storage resource cost by attribute ciphertext, the reduce is limited. Secondly, the computation cost for access policy is much less than which cost for attribute ciphertext during encrypting and decrypting. However, Theorem 2 provides block algorithm to mitigate runtime of decryption, it only brings limited mitigation of decrypting time to SCP-ABE. In summary, SCP-ABE provides an efficient policy updating interface without sacrificing other performances.

6. Conclusion and future work

In this paper, we provide a scalable access policy named BLSSS and a manageable ABE scheme named SCP-ABE . There are two advantages of BLSSS: 1) low cost, BLSSS is low cost in terms of storage and computation; 2) strong scalability, BLSSS is provided with efficient scaling functions. At the same time, we propose the SCP-ABE by taking full advantage of BLSSS. In SCP-ABE, access policy can be dynamic managed by proposed function *Update*. The function produces minimum increment sets Δ, Θ of policy-updated ciphertext. As shown in simulation experiments, the SCP-ABE achieves better performance than existing ABE schemes [11,27,40] in terms of policy managing.

In the future work, we will propose agent policy managing interface for SCP-ABE by utilizing BLSSS. Data owner can authorize a trusted third party (TTP) to manage partial blocks of BLSSS via the interface. The interface will provide more flexible policy manage pattern and reduce the policy managing overhead of thin clients. It will great improves the performance of SCP-ABE.

Acknowledgment

This work is supported by the National Science Foundation of China (No. 61373040, No. 61572370, No. 61602125).

References

[1] A.-L. Hussam, L. Princehouse, H. Weatherspoon, RACS: a case for cloud storage diversity, in: Proceedings of the 1st ACM symposium on Cloud computing., ACM, 2010, pp. 229–240.
[2] J. Benaloh, J. Leichter, Generalized secret sharing and monotone functions, in: Proceedings of the conference on Advances in cryptology, Springer, 1990, pp. 27–35.
[3] A. Beimel, Secure schemes for secret sharing and key distribution[m] technion-israel institute of technology, Faculty Comput. Sci. (1996).
[4] G.R. Blakley, G.A. Kabatianskii, Linear algebra approach to secret sharing schemes, in: Error Control, Cryptology, and Speech Compression., Springer, 1994, pp. 33–40.
[5] M. Bertilsson, A construction of practical secret sharing schemes using linear block codes, in: Advances in CryptologyAUSCRYPT'92., Springer, 1993, pp. 27–35. Ingemar Ingemarsson.
[6] F.E. Brickell, Some ideal secret sharing schemes, in: Proceedings of the conference on Advances in Cryptology-EUROCRYPT'89., Springer, 1990, pp. 468–475.
[7] R. Chellappa, Intermediaries in cloud-computing: A new computing paradigm, in: INFORMS Annual Meeting, Dallas, 1997.
[8] M.H. Dehkordi, R. Ghasemi, A lightweight public verifiable multi secret sharing scheme using short integer solution, Wireless Pers. Commun. (2016) 1–11.
[9] Z. Fu, K. Ren, J. Shu, et al., Enabling personalized search over encrypted outsourced data with efficiency improvement, IEEE Trans. Parallel Distrib. Syst. 27 (9) (2016) 2546–2559.
[10] S. Fugkeaw, H. Sato, Updating policies in CP-ABE-based access control: an optimized and secure service, in: European Conference on Service-Oriented and Cloud Computing., Springer International Publishing, 2016, pp. 3–17.
[11] V. Goyal, et al., Attribute-based encryption for fine-grained access control of encrypted data, in: Proceedings of the 13th ACM conference on Computer and communications security, ACM, 2006, pp. 89–98.
[12] S. Hohenberger, B. Waters, Online/offline attribute-based encryption, in: Proceedings of the Conference on Public-Key Cryptography PKC 2014., Springer, 2014, pp. 293–310.
[13] M. HorvAth, Attribute-based encryption optimized for cloud computing, in: SOFSEM 201 Theory and Practice of Computer Science., Springer, 2015, pp. 566–577.
[14] J. Hur, D.K. Noh, Attribute-based access control with efficient revocation in data outsourcing systems, IEEE Trans. Parallel Distrib. Syst. 22 (7) (2011) 1214–1221.
[15] N.H. Hussein, A. Khalid, A survey of cloud computing security challenges and solutions, Int. J. Comput.Sci. Inf. Secur. 14 (1) (2016) 52.
[16] S. Kamara, L. Kristin, in: International conference on Financial Cryptography and Data Security., Springer, 2010, pp. 136–149.
[17] M. Karchmer, W. Avi, On span programs, in: Structure in Complexity Theory Conference., 1993, pp. 102–111.
[18] A. Lewko, B. Waters., Decentralizing attribute-based encryption, in: Proceedings of the Advances in Cryptology-EUROCRYPT 2011., Springer, 2011, pp. 568–588.
[19] M. Li, S. Yu, Y. Zheng, et al., Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption, IEEE Trans. Parallel Distrib. Syst. 24 (1) (2013) 131–143.
[20] J. Li, Y. Shi, Y. Zhang, Searchable ciphertext-policy attribute-based encryption with revocation in cloud storage, Int. J. Commun. Syst. 30 (1) (2017).
[21] L. Zhen, Z. Cao, S.W. Duncan, Efficient generation of linear secret sharing scheme matrices from threshold access trees, Cryptology (2010). ePrint Archive Report 2010/374 http://eprint.iacr.org/2010/374.
[22] J.L. Massey, Minimal codewords and secret sharing, in: Proceedings of the 6th Joint Swedish-Russian International Workshop on Information Theory., 1993, pp. 276–279.
[23] V. Nikov, N. Svetla, New monotone span programs from old, IACR Cryptology (2004) 282. ePrint Archive 2004
[24] V.R. Pancholi, B.P. Patel, Enhancement of cloud computing security with secure data storage using AES, Int. J. Innovative Res. Sci. Technol. 2 (9) (2016) 18–21.
[25] J.W. Rittinghouse, J.F. Ransome, Cloud Computing: Implementation, Management, and Security, CRC press, 2016.
[26] J. Simonis, A. Ashikhmin, Almost affine codes, Des. Codes Cryptogr. 14 (2) (1998) 179–197.
[27] A. Sahai, H. Seyalioglu, W. Brent, Dynamic credentials and ciphertext delegation for attribute-based encryption, in: Proceedings of the Advances in CryptologyCRYPTO 2012., Springer, 2012, pp. 199–217.
[28] A. Shamir, How to share a secret, Commun. ACM 22 (11) (1979) 612–613.
[29] J. Shetty, M.R. Anala, G. Shobha, An approach to secure access to cloud storage service, Int. J. Res. 2 (1) (2015) 364–368.
[30] M. Stadler, Publicly verifiable secret sharing, in: Proceeding of the Advances in Cryptology-EUROCRYPT'96., Springer, 1996, pp. 190–199.
[31] Y. Sun, G. Li, Z. Lin, et al., A completely fair secret sharing scheme without dealer, in: Consumer Electronics-Taiwan (ICCE-TW), 2016 IEEE International Conference on. IEEE, 2016, pp. 35–36.
[32] V. Vaikuntanathan, P. Voulgaris, Attribute based encryption using lattices: U.s. patent 9,281,944, 2016, 3–8.

[33] B. Waters, Ciphertext-policy attribute-based encryption: an expressive, efficient, and provably secure realization, Public Key Cryptogr. (2011) 53–70. 6571
[34] L. Wei, et al., Security and privacy for storage and computation in cloud computing[j], Inf Sci (Ny) 258 (2014) 371–386.
[35] J. Wu, et al., Cloud storage as the infrastructure of cloud computing, in: Proceeding of the International Conference on Intelligent Computing and Cognitive Informatics. IEEE, 2010, pp. 380–383.
[36] N. Xavier, V. Chandrasekar, Cloud computing data security for personal health record by using attribute based encryption, Int. J. Inf. Business Manag. 7 (1) (2015) 209–214.
[37] F. Xhafa, et al., Designing cloud-based electronic health record system with attribute-based encryption, Multimed. Tools Appl. 74 (10) (2015) 3441–3458.
[38] Z. Xia, X. Wang, L. Zhang, et al., A privacy-preserving and copy-deterrence content-based image retrieval scheme in cloud computing, IEEE Trans. Inf. Forensics Secur. 11 (11) (2016) 2594–2608.
[39] Z. Xia, X. Wang, X. Sun, et al., A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data, IEEE Trans. Parallel Distrib. Syst. 27 (2) (2016) 340–352.
[40] K. Yang, et al., Enabling efficient access control with dynamic policy updating for big data in the cloud, in: Proceedings of the IEEE conference on INFOCOM 2014. IEEE, 2014, pp. 2013–2021.
[41] S. Yu, C. Wang, K. Ren, et al., Attribute based data sharing with attribute revocation, in: Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, ACM, 2010, pp. 261–270.

Wang Jing is a PhD of Wuhan University and a post-doctor of Sun Yat-sen University. Her research interests include security and privacy of cloud.

Huang Chuanhe is a Professor and a PhD supervisor of Wuhan University. His research interests include cryptography, wireless security and trust management and so on.

Neal N. Xiong is current a Professor at School of Computer Science, Colorado Technical University, Colorado Spring, CO, USA. He received his both PhD degrees in Wuhan University (about software engineering), and Japan Advanced Institute of Science and Technology (about dependable networks), respectively. Before he attends Colorado Technical University, he worked in Wentworth Technology Institution, Georgia State University for many years. His research interests include Cloud Computing, Security and Dependability, Parallel and Distributed Computing, Networks, and Optimization Theory.

Wang Jinhai is a PhD of Wuhan University. His research interests include cloud computing and high performance computing.