Scalable Access Policy for Attribute Based Encryption in Cloud Storage

Jing Wang[1,2], Chuanhe Huang[1,2](✉), and Jinhai Wang[1,2]

[1] Computer School, Wuhan University, Wuhan 430072, China
{wjing,huangch,wangjinhai}@whu.edu.cn
[2] Collaborative Innovation Center of Geospatial Technology, Wuhan University, Wuhan 430072, China

Abstract. Cloud storage provides outsourced storage services in a cost-effective manner. A key challenge in cloud storage is the security and integrity 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. A critical issue in ABE is the managing of access policy. Policy managing may incur substantial computation and communication overhead in the ABE scheme with unscalable access policy. In this work, we propose a form of access policy named block Linear Secret Sharing Scheme (LSSS) matrix. The scalability of block LSSS matrix provides an efficient policy managing interface for ABE schemes. Thus, the ABE schemes use block LSSS matrix as access policy are light weight in computation and communication, as compared with other schemes during access policy managing. Furthermore, the block LSSS matrix enjoys advantages of efficiency, flexibility and security, bringing a number of improvements in various aspects of ABE.

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

1 Introduction

Cloud data storage offers a number of advantages over traditional storage, in terms of availability, scalability, performance, portability and functional requirements. It develops a national strategy to collect, preserve and make available digital content for current and future generations based on cloud storage. Though the advantages of cloud storage are clear, a critical concern in the present data outsourcing scenario is the enforcement of strong data security mechanisms [11]. Ensuring privacy and security of such data is important for users to trust the service providers. Along that direction, adequate access control techniques are to be deployed, ensuring services are only provided to legitimate users. Attribute Based Encryption (ABE) is a novel cryptographic tool that can provide fine-grained ciphertext access control for cloud storage. The core properties of such a cryptographic storage service include: (1) access control of the data is maintained by the customer instead of the service provider; (2) the security properties are

DOI: 10.1007/978-3-319-27137-8_29

derived from cryptography, as opposed to traditional access control. Therefore, such a service provides several compelling advantages over other storage services based on public cloud infrastructures.

A crucial concept in ABE is known as the access policy/access structure. The data owner can manage the access privilege of his data through manipulating its access policy. In fact, the performance of ABE schemes depends significantly on their access policy. For example, the scale of access policy decides the size of ciphertext and the computation of encrypting, the expression of access policy decides the flexibility of the access control mechanism. Many researchers focus on optimizing ABE via optimizing the access policy. However, scalability, as an important property of access policy, was often ignored. In many scenarios, access policy are dynamically and frequently updated for various reasons. Thus, the data owner needs to re-encrypt the data with the new policy and upload new ciphertext to the cloud during policy updating in such ABE schemes without scalable access policy. Heavy computation and communication overhead are incurred by such processing.

The grand challenge of policy updating in ABE based access control mechanism is to jointly guarantee correctness, completeness and security [17]. At the same time, efficiency is further considered as an important requirement in this paper. The policy updating issue has been discussed in a few ABE schemes [15–17]; yet there are still weaknesses in terms of fulling meeting the above requirements. In [15,16], the updated access policy should be more restrictive than the previous one, because the scalability of the access policy is limited. Although [17] has improved the completeness of policy updating method, the updating process of this scheme is still not efficiency enough. In some cases, the access policy is required to re-construct (especially, the processing of threshold updating is cumbersome).

Focusing on scalability, we propose a new access policy called block LSSS matrix. Block LSSS matrix comes with an efficient managing interface that greatly improves the manageability of ABE schemes. The advantages of describing access policy as a block LSSS matrix are two fold. First, the computation and communication of policy updating are both light weight via processing in block. Because each block of the matrix is independent, block updates are co-relation free. Second, the computational complexity of decryption is reduced for the ABE scheme. Because the decryption operation can be decomposed into blocks, lowering the computation scale. Besides, a feature of block matrix lies in the following intuition: the logical construction of access policy is intuitively presented in a matrix. In order to hide the structure of access policy and extend the space of LSSS matrix, the mask matrix is provided. Mask matrix is an available tool to improve the security and flexibility of block LSSS matrix; it eliminates the intuition of block LSSS matrix yet retainss its logic structure.

The contributions of this paper are summarized as follows.

(1) For optimizing the performance of ABE, we provide a generalized method to describe the access policy in ABE as a block LSSS matrix, which enables high scalability;

(2) In order to improve the flexibility and security of block LSSS matrix, we introduce the concept of mask matrix. The mask matrix can expand the policy and hide the policy structure;
(3) We provide four kinds of managing function for bock LSSS matrix. The functions are convenient and efficient, they can process any type of policy updates.

2 Related Work

Cloud computing is a new architecture that is envisioned as the next generation computing paradigm [8]. It introduces a major change in data storage and application execution, with everything now hosted in the cloud — a nebulous assemblage of computers and servers accessible via the Internet [9]. As a successful case, the United States Library of Congress moved its digitized content to the cloud in 2009 [10]. 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 in cloud storage systems [23–26]. ABE provides a smart way to construct a fined-grained access control for cloud storage [15, 20–22, 27]. Access privileges described by access policy in ABE are more flexible and expressive than in traditional coarse-grained access control mechanisms. The access policy is derived from a Secret Sharing Scheme (SSS) which shares a secret among a group of participants [12]. In such a way, the secret can be reconstructed only by a specified group of shares. A wide range of general approaches for designing secret sharing schemes are known, e.g., Shamir [6], Benaloh [2], Bertilsson [3], Brickell [4], Massey [5], Blakley [1], Simonis [7], Ventzislav [13] and Svetla [14]. Thus, access policy is often described in various forms, such as monotonic Boolean formula, access tree or Linear Secret Sharing Scheme (LSSS) matrix. All these forms of access policy are poor in scalability and the policy updates of ABE is limited or inefficient.

Table 1. Comparison of ABE schemes

Scheme	Goyal [15]	Sahai [16]	Yang [17]	Proposed
Limit of updating	Y	Y	N	N
Form of policy	Access tree	LSSS matrix	Boolean formula Access tree LSSS matrix	Boolean formula Access tree LSSS matrix
Re-construct SSS	N	Y	Y	N
Update unit	Node	Row	Node/Row	Sub-tree/matrix

Recently, only a few ABE schemes have considered policy updating and provided solutions. Table 1 shows the comparison of existing ABE schemes that support access policy updating. Compared with Goyal, Sahai and Yang's scheme, our scalable ABE scheme is optimized for SSS updating. Different from [15,16], the updating of policy in our scheme is more flexible without any restrictions. Compared with the other three schemes, our updating method is processed in sub-tree/matrix instead of in node/row. Consequently, the proposed scheme is more efficient. Specifically, our scheme is more streamlined and intuitive than [17] in dealing with threshold. Thus, the block LSSS matrix as a scalable access policy can improve the ABE scheme in policy managing.

3 Scalable Access Policy

3.1 Definitions and Symbols

Access Policy. The access policy in ABE is usually defined as monotonous access structures.

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

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

Specially, $A \in \mathbb{A}$ is called authorized set, and $A \notin \mathbb{A}$ is called unauthorized set. In the ABE context, the role of the parties are taken by the attributes. Thus, an access policy $\mathbb{A}$ is a collection of attribute sets. In fact, an access policy can be realized by a Secret Sharing Scheme which decomposes a secret s into a share set $S = \{s_i, \ldots, s_n\}$. Let ρ be a map form S to P, S' be a subset of S and $P' = \{\rho(s_i) | s_i \in S'\}$. Secret s can be recovered by S' iff P' is authorized.

Furthermore, an access policy can be viewed as a circuit which consists of a set of gates (AND, OR, threshold) and inputs, as shown in Fig. 1. The circuit takes a set of shares S' as inputs and outputs the secret s iff P', the image set of S', is authorized.

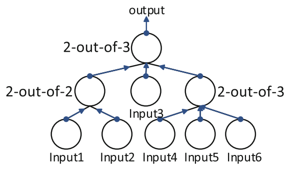


Fig. 1. An access policy described as tree-circuit

The block LSSS matrix proposed in this paper is generated by the tree-circuit and its node matrices[1]. We formalize the processing of block LSSS matrix generation in Fig. 2. Firstly, each non-leaf node is defined a node matrix which is generated by a Vandermonde matrix and a mask matrix. Secondly, a block LSSS matrix is generated by all of these node matrices. Finally, the block LSSS matrix is masked with a full rank matrix.

Matrix Notations and Operations. Unless otherwise indicated, a boldface lowercase letter denotes a vector, an uppercase letter denotes a matrix in this paper. Key notations are summarized in Table 2 for ease of reference.

[1] The detailed describing of node matrix is given in Sect. 3.2.

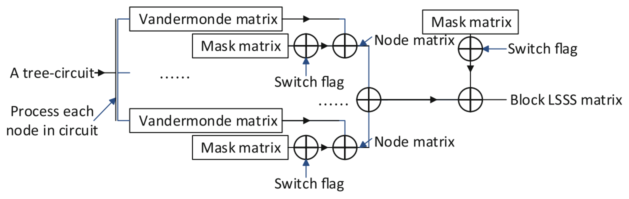


Fig. 2. Tree-circuit to Block LSSS Matrix

Table 2. Key notations

Notation	Description
I_k	The unit matrix with order k
$O_{m\times n}$	The null matrix with m row and n column
$R(*)$	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, each row of M^ε is the i^{th} row of M where $\varepsilon_i = 1$
$* \approx **$	Matrix $*$ and matrix $**$ can describe the same access policy
$\|\| * \|\|_0$	The standard zero-norm of the vector $*$

We further define a few matrices with specific structure:

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

where $O_{1\times(k-1)}$ is simplified to O. For simplicity, we shall omit the subscript in some cases.

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

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 i^{th} 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 $M_i = (\bar{M}_i, \hat{M}_i) \in Z^{k_i \times l_i}$ where $\bar{M}_i \in Z^{k_i \times 1}$, $\hat{M}_i \in Z^{k_i \times (l_i - 1)}$,

$$U^*(A, M_1, \ldots, M_n) = U(M_1, \ldots, M_n)E(A, l_1, \ldots, l_n) = \begin{pmatrix} \bar{M}_1 A_1 & \hat{M}_1 & \ldots & O \\ \vdots & \vdots & \ddots & \vdots \\ \bar{M}_n A_n & O & \ldots & \hat{M}_n \end{pmatrix} \tag{5}$$

Specifically,

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

3.2 Node Matrix

Node matrix is a special LSSS matrix[2] which can describe a node in the tree-circuit. Generally, a t-out-of-n node (as a general gate node) can be described by a matrix $M \in Z^{n \times t}$, where M^ε is full rank, $\forall \varepsilon \in \{0,1\}^n$ and $||\varepsilon||_0 = t$.

Transformation. In order to make the scalable access policy more general, we present a method to transform various gate forms into node matrices. The node matrix must keep the same inputs and output as the gate does. This implies that, the share set generated by the node matrix and by the gate are equal.

In a monotonic Boolean expression, there are two gate forms: AND($\wedge$) and OR($\vee$) [18]. Threshold(t-out-of-n, $Thr_{t,n}(a_1, \ldots, a_n)$) can be represented by a combination of AND and OR. For example, $Thr_{2,3}(a_1, a_2, a_3) \iff (a_1 \wedge a_2) \vee (a_2 \wedge a_3) \vee (a_1 \wedge a_3)$. AND gate is a specific threshold, n-out-of-n. Let the original share set be $S = \{s_1, \ldots, s_n\}$ and the secret $s = \sum s_i$. The node matrix and vector are defined as follows:

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

where $v_i = \sum_{j=i}^{n} s_j$. Let M_i denote the i^{th} row vector of M_A. As a result, $\forall i, s_i = M_i V_A^T$, OR gate is also a specific threshold, 1-out-of-n. Let the secret set be $s \in Z$. The OR-node matrix and vector are defined as $M_O = (1, \ldots, 1)^T$ and $V_O = (s)$.

In an access tree, each threshold node can be described as a Vandermonde matrix [19]. A t-out-of-n node can be described as a $n \times t$ Vandermonde matrix $V_{n \times t}$ and a node vector V_t:

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

[2] The detailed definition of LSSS matrix is given in [16].

where x_i denotes the interpolation of its i^{th} child, a_j denotes the coefficient of x_i^j in the node polynomial $f(x)$ and s denotes the constant term of $f(x)$. Furthermore, $\forall i, M_i V_t^T = f(x_i) = s_i$.

Scalable Node Matrix. A generalized method to define scalable node matrix is given in this section. Let $V_{t,n} \in Z^{n\times t}$ be a Vandermonde matrix, $T_t \in Z^{t\times t}$ be an upper triangular matrix and $R(T_t) = t$. Thus, the matrix $M_{n,t}$ is calculated as $M_{n,t} = V_{n,t}T_t$. Significantly, $M_{n,t} \approx V_{n,t}$ and $M^{\varepsilon}_{n't} \approx V^{\varepsilon}_{n,t}, \forall \varepsilon \in \{0,1\}^n$. T_t is called mask matrix and the space of the node matrix can be extended via the mask matrix. Finally, a secret s is chosen at random and node vector is generated as $V_t = (s, v_2, \dots, v_t)$, where $v_i \in Z$ is chosen randomly. The node matrix generated by this method is expressiveness and scalable. There are four scaling functions of node matrix defined as follows.

$Node^+_n(node, \Delta n)$: The function modifies the node matrix M_n, 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. Thus,

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

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 ε. Let $M_{n,t} = V_{n,t}T_t$ be the original node matrix. $M^{\varepsilon}_{n,t}$ can describe the modified node because $M^{\varepsilon}_{n,t} \approx V^{\varepsilon}_{n,t}$. Besides, the node vector V_t remains intact with this function.

$Node^+_t(node, \Delta t)$: The node threshold t is changed into $t + \Delta t$ and 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:

$$\Delta V_{n,\Delta t} = \begin{pmatrix} x_1^t & \dots & x_1^{t+\Delta t} \\ \vdots & & \vdots \\ x_n^t & \dots & x_n^{t+\Delta t} \end{pmatrix} \tag{8}$$

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

$$T_{t+\Delta t} = \begin{pmatrix} T_t & P \\ O & T_{\Delta t} \end{pmatrix} \tag{9}$$

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 node matrix. Similar to $Node^+_t(node, \Delta t)$, we define

$$M_{n,t} = (M_{n,t-\Delta t}, \Delta M_{n,\Delta t}), V_{n,t} = (V_{n,t-\Delta t}, \Delta V_{n,\Delta t}), T_t = \begin{pmatrix} T_{t-\Delta t} & P \\ O & T_{\Delta t} \end{pmatrix}$$

Thus, $M_{n,t-\Delta t} = V_{n,t-\Delta t}T_{t-\Delta t}$ can be obtain by removing the last Δt columns 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.

3.3 Block LSSS Matrix

The matrix can describe not only a node but also a complete tree-circuit. A theorem in [13] provides an efficient way to construct block LSSS matrix — a *compound matrix* of a set of node matrices.

Theorem 1. *Let $M_1 \in Z^{m_1 \times n_1}, \dots, M_l \in Z^{m_l \times n_l}$ be a set of LSSS matrices described access policy $\mathbb{A}_1, \dots, \mathbb{A}_l$, $A \in Z^{l \times k}$ is a LSSS matrix described access policy $\mathbb{A}'$ and $P_1, \dots, P_l$ be the parties of $\mathbb{T}$. Access policy $(\mathbb{A}_1 \mapsto P_1) \dots (\mathbb{A}_l \mapsto P_l)\mathbb{A}'$ can be described by matrix $M = U^*(A, M_1, \dots, M_l)$.*

Proof. The detailed proof is shown in [13].

Following Theorem 1, Algorithm 1, an iterative algorithm, is proposed to generate *compound LSSS matrix*. Let $\mathbb{T}$ be an access policy/tree-circuit and *root* be the root of $\mathbb{T}$. $TreeToLSSS(\mathbb{T}, root)$ performs a depth-first traversal of $\mathbb{T}$ and generates a matrix M by its node matrices $M_k, 1 \le k \le n$. The matrix M is called block LSSS matrix, which is a sparse block matrix and an effective LSSS matrix.

Algorithm 1. $TreeToLSSS(\mathbb{T}, node)$

```
Input: A matrix-tree: $\mathbb{T}$; A node of matrix: node;
Output: LSSS matrix: $M$; Vector: $V$;
 1: if node is a leaf then
 2:     $M \leftarrow (1)$;
 3:     $V \leftarrow (1)$;
 4: else
 5:     for each child $c_i$ of node do
 6:         $(M_i, V_i) \leftarrow TreeToLSSS(\mathbb{T}, c_i)$;
 7:     end for
 8:     $M \leftarrow U^*(M(node), M_1, \dots, M_n)$;//$M(node)$ is the node matrix of node, $n$ is
        the total number of children of node
 9:     $V \leftarrow (V(node), \hat{V}_1, \dots, \hat{V}_n)$;//$V(node)$ is the node vector of node
10: end if
11: return $(M, V)$;
```

To facilitate discussions, we introduce the following notations.
$\varphi : \{1, 2, \dots, n\} \to N(\mathbb{T})$, where $N(\mathbb{T})$ denotes the set of non-leaf nodes in $\mathbb{T}$ (ψ denotes the inverse of φ).
$\delta : \{1, 2, \dots, l\} \to L(\mathbb{T})$, where $L(\mathbb{T})$ denotes the set of leaves in $\mathbb{T}$ (σ denotes the inverse of δ).
$Anc(\delta(i))$ or $Anc(\varphi(i))$: the set of ancestors of $\delta(i)$ or $\varphi(i)$.
$I(j, i)$: a function returning the index of the subtree of $\varphi(j)$ which include node $\delta(i)$ or $\varphi(i)$.

Performances of Block LSSS Matrix. $\forall \mathbb{T}$ with l leaves and n non-leaf nodes can be viewed as a block matrix M generated by Algorithm 1:

$$M = \begin{pmatrix} c_{1,1} & \mathbf{m_{1,1}} & \dots & \mathbf{m_{1,n}} \\ \vdots & \vdots & & \vdots \\ c_{1,l} & \mathbf{m_{l,1}} & \dots & \mathbf{m_{l,n}} \end{pmatrix} \tag{10}$$

where

$$\mathbf{m_{i,j}} = \begin{cases} \mathbf{o} & \varphi(j) \notin Anc(\delta(i)) \\ c_{i,j}\hat{M}(\varphi(j))_k & \varphi(j) \in Anc(\delta(i)), k = I(j,i) \end{cases} \tag{11}$$

$c_{i,j} = \sum_{node \in Anc(\delta(i)) - Anc(\varphi(j))} M(node)_{1,k}$. $\forall node \in T$, we set $\bar{M}(node) = (1,1,\ldots,1)^T$. That implies every node-mask matrix is an upper triangular matrix with the first column $\mathbf{e_1}$. As a result, $\forall i,j, c_{i,j} = 1$ and Eq. (10) can be simplified:

$$\mathbf{m_{i,j}} = \begin{cases} \mathbf{o} & \varphi(j) \notin Anc(\delta(i)) \\ \hat{M}(\varphi(j))_k & \varphi(j) \in Anc(\delta(i)), k = I(j,i) \end{cases} \tag{12}$$

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

Theorem 2. *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 block LSSS matrix as shown in Eq.* (10) *and $M_{s(n)}$ is defined as follows:*

$$M_{s(n)} = \begin{pmatrix} 1 & \mathbf{m_{1,j_1}} & \ldots & \mathbf{m_{l,j_n}} \\ \vdots & \vdots & & \vdots \\ 1 & \mathbf{m_{l,j_1}} & \ldots & \mathbf{m_{l,j_n}} \end{pmatrix} \tag{13}$$

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 block LSSS matrix can be computed by *compounding* the solutions of its node matrices.

Theorem 3. *Let $M = U^*(A, M_1, \ldots, M_n)$, where $A \in Z^{l \times k}, M_1 \in Z^{l_1 \times k_n}, \ldots, M_n \in Z^{l_n \times k_n}$ is a set of LSSS matrixes, $\mathbf{e_{1,i}} \in \{0,1\}^i$ 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 $A^T X = \mathbf{e_{1,l}}^T, M_1^T X = \mathbf{e_{1,l_1}}^T, \ldots, M_n^T X = \mathbf{e_{1,l_n}}^T$, respectively. Let $\mathbf{x}^* = U^*(\mathbf{x}, \mathbf{x_1}, \ldots, \mathbf{x_n})$. As a result, $(M)^T \mathbf{x}^* = \mathbf{e_{1,sum}}$.*

Proof.

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

On the other hand, the block LSSS matrix is sparse. That makes block LSSS matrix weak in security, revealing the structure of access policy. To eliminate zero blocks, the block LSSS matrix can be masked by a full rank matrix when strong security is desired. At the same time, the mask matrix can expand the space of LSSS matrix and make the LSSS matrix more flexible. Let $M \in Z^{l \times m}$ be a block LSSS matrix, $T \in Z^{m \times m}$ be a full rank matrix chosen as mask matrix and$M' = MT$. As a result, $M' \approx M$ and M' is eliminated zeros. In fact, an upper triangular matrix is easy to generate and scale. Thus, upper triangular matrices are chosen as mask matrices to keep the scalability of block LSSS matrices.

Dynamic Updates of Block LSSS Matrices. The block LSSS matrix is scalable. Each pair of blocks are mutual independent. Modifying one block has no effect on other blocks. We will give the scalable functions of block LSSS matrix in this section, arbitrary updates of block matrix can be achieved by these functions.

Firstly, we define four scalable functions that apply to the LSSS matrix without mask. The functions output non-zero adding blocks, modifying blocks and removing blocks of LSSS matrix. 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. The threshold of $\varphi(\alpha)$ is changed to $t + \Delta t$ via 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 $V_{\varphi(\alpha)}$, each block $\mathbf{m_{i,\alpha}}$ and $V_{\varphi(\alpha)}$ are modified to a $t + \Delta t$ dimension vector. The non-zero adding block $\Delta M_{n,\Delta t}$ is generated by $Node_t^+(node, \Delta t)$.
(2) Δt is negative. $Node_t^-(node, |\Delta t|)$ is run to remove the last $|\Delta t|$ column vectors of node matrix $M_{\varphi(\alpha)}$ and node vector $V_{\varphi(\alpha)}$, each block $\mathbf{m_{i,\alpha}}$ and $V_{\varphi(\alpha)}$ are removed with the last $|\Delta t|$ entries. The removed blocks are denoted as $\Delta M_{n,\Delta t}$.

$Path(\mathbb{T}, \alpha, \gamma)$: 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 $\varphi(\gamma)$, $Node_n^+\varphi(\gamma, 1)$ is run to insert a row vector $M(\gamma)_{n'+1}$ into $M(\gamma)$. Then, we set the block node vectors $V_\alpha = (1, \mathbf{v_{\alpha,1}}, \dots, \mathbf{v_{\alpha,n}})$ and $V_\gamma = (1, \mathbf{v_{\gamma,1}}, \dots, \mathbf{v_{\gamma,n}})$ where

$$\mathbf{v}_{\alpha,j} = \begin{cases} \hat{M}(\varphi(j))_k & \text{if } \varphi(j) \in Anc(\varphi(\alpha)) \text{ and } k = I(j, \alpha) \\ \mathbf{o} & \text{if } \varphi(j) \notin Anc(\varphi(\alpha)) \end{cases} \tag{14}$$

$$\mathbf{v}_{\gamma,j} = \begin{cases} \hat{M}(\gamma)_{n'+1} & \text{if } j = \gamma \\ \hat{M}(\varphi(j))_k & \text{if } \varphi(j) \in Anc(\varphi(\gamma)) \text{ and } k = I(j, \gamma) \\ \mathbf{o} & \text{Otherwise} \end{cases} \tag{15}$$

Let $\Delta V = V_\gamma - V_\alpha$ be the modifying block. Finally,$\forall \delta(i) \in L(\alpha), M_i \leftarrow M_i + \Delta V$, 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 case:

(1) The inserted node is a leaf $\delta(\alpha)$ associated with the insertion vector M_α. Then we calculate the updating LSSS matrix and random vector as follows:

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

The adding blocks are $\Delta\hat{M}$ and M_α in this case.

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

$$\mathbf{v}_{add,j} = \begin{cases} \hat{M}(\varphi(\alpha))_{n'+1} & \text{if } j = \alpha \\ \hat{M}(\varphi(j))_k & \text{if } \varphi(j) \in Anc(\varphi(\alpha)) \text{ and } k = I(j, \alpha) \\ \mathbf{o} & \text{Otherwise} \end{cases} \tag{16}$$

Then,

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

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

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

where n denotes the number of row of the original matrix M. The adding blocks are V_{add} and $\varDelta \bar{M}$.

Remove$(\mathbb{T}, \gamma)$: A subtree $\mathbb{T}_\gamma$ with root $\varphi(\gamma)$ is removed from $\mathbb{T}$. $\forall M_\alpha$ is removed from the block LSSS matrix M where $\delta(\alpha) \in L(\varphi(\gamma))$, $\forall \mathbf{m_{i,j}}$, $\varphi(j) \in N(\mathbb{T}_\gamma)$ 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 of all the node vector V_j, $\varphi(j) \in N(\varphi(\gamma)$.

Similarly, we define four scalable functions for the block matrix with mask.

$Thr_M(\mathbb{T}, \alpha, \varDelta t)$: Let access policy $\mathbb{T}$ be described by matrix $M' = MT$ where M is a LSSS matrix generated by Algorithm 1 and T is a mask matrix. This function is also discussed in two cases.

(1) $\varDelta t > 0$. Let M, T and M' be viewed as a block matrixes

$$M = \begin{pmatrix} M_{u_1} & M_{u_2} \\ M_{\alpha_1} & M_{\alpha_2} \\ M_{d_1} & M_{d_2} \end{pmatrix}, T = \begin{pmatrix} T_1 & P \\ O & T_2 \end{pmatrix}, M' = \begin{pmatrix} M'_{u_1} & M'_{u_2} \\ M'_{\alpha_1} & M'_{\alpha_2} \\ M'_{d_1} & M'_{d_2} \end{pmatrix}$$

where $(M_{\alpha_1}, M_{\alpha_2})$ is the block with non-zero blocks $\mathbf{m}_{\mathbf{i},\alpha}$ and $M_{u_1}, M_{\alpha_1}, M_{d_1}$ are the blocks with $\mathbf{m}_{\mathbf{i},\alpha}$ as the last columns. $Thr(\mathbb{T}, \alpha, \varDelta t)$ is run, M and T are modified into M_{Thr} and T_{Thr}:

$$M_{thr} = \begin{pmatrix} M_{u_1} & O & M_{u_2} \\ M_{\alpha_1} & \varDelta M_\alpha & M_{\alpha_2} \\ M_{d_1} & O & M_{d_2} \end{pmatrix}, T_{thr} = \begin{pmatrix} T_1 & \varDelta P_1 & P \\ O & \varDelta T & \varDelta P_2 \\ O & O & T_2 \end{pmatrix}$$

where $\varDelta T \in Z^{\varDelta t \times \varDelta t}$ is a full rank upper triangular matrix and $\varDelta P_1$, $\varDelta P_2$ are random matrix without zero elements. As a result, we get the matrix:

$$M'_{thr} = M_{thr} T_{thr} = \begin{pmatrix} M'_{u_1} & M_{u_1} \varDelta P_1 & M'_{u_2} \\ M'_{\alpha_1} & M_{\alpha_1} \varDelta P_1 + \varDelta M \varDelta T & M'_{\alpha_2} + \varDelta M \varDelta P_2 \\ M'_{d_1} & M_{d_1} \varDelta P_1 & M'_{d_2} \end{pmatrix} \tag{17}$$

There are one adding block A and one modifying block B:

$$A = \begin{pmatrix} M'_{u_1} \varDelta P_1 \\ M_{\alpha_1} \varDelta P_1 + \varDelta M \varDelta T \\ M_{d_1} \varDelta P_1 \end{pmatrix}, B = \varDelta M \varDelta P_2$$

(2) $\Delta t < 0$. Similarly, original matrix M and T are expressed as follows

$$M = \begin{pmatrix} M_{u_1} & O & M_{u_2} \\ M_{\alpha_1} & \Delta M_\alpha & M_{\alpha_2} \\ M_{d_1} & O & M_{d_2} \end{pmatrix}, T = \begin{pmatrix} T_1 & \Delta P_1 & P \\ O & \Delta T & \Delta P_2 \\ O & O & T_2 \end{pmatrix}$$

Then, we get

$$M' = MT = \begin{pmatrix} M'_{u_1} & \Delta M'_u & M'_{u_2} \\ M'_{\alpha_1} & \Delta M'_\alpha & M'_{\alpha_2} \\ M'_{d_1} & \Delta M'_d & M'_{d_2} \end{pmatrix} \tag{18}$$

Firstly, $Thr(\mathbb{T}, \alpha, \Delta t)$ is run to modify matrix M and vector V. Thus,

$$M_{thr} = \begin{pmatrix} M_{u_1} & M_{u_2} \\ M_{\alpha_1} & M_{\alpha_2} \\ M_{d_1} & M_{d_2} \end{pmatrix}, T_{thr} = \begin{pmatrix} T_1 & P \\ O & T_2 \end{pmatrix}$$

The modified matrix is:

$$M'_{thr} = \begin{pmatrix} M'_{u_1} & M'_{u_2} \\ M'_{\alpha_1} & M'_{\alpha_2} - \Delta M \Delta P_2 \\ M'_{d_1} & M'_{d_2} \end{pmatrix} \tag{19}$$

There are one modifying block $\Delta M \Delta P_2$ and three removing blocks ΔM_u, ΔM_α, ΔM_d.

$Path_M(\mathbb{T}, \alpha, \gamma)$: Let access policy $\mathbb{T}$ be described by matrix $M' = MT$. $Path(\mathbb{T}, \alpha, \gamma)$ is run to modify matrix M. Vector ΔV is generated as shown in $Path(\mathbb{T}, \alpha, \gamma)$ and $\Delta V'$ is set to be ΔVT. Finally, $\forall \rho(i) \in L(\alpha), M'_i \leftarrow M'_i + \Delta V'$. $\Delta V'$ 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 matrix $M' = MT \in Z^{l \times m}$ and $\Delta M' = \Delta M \Delta T \in Z^{\Delta l \times \Delta m}$ denote the submatrix describes $sub\mathbb{T}$. The function is discussed in two cases.

(1) $\rho(\alpha) \in L(\mathbb{T})$. Let M, M' be described as block matrices

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

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

$$M_{add} = \begin{pmatrix} M_u & O \\ \varepsilon M_\alpha & \Delta \hat{M} \\ M_d & O \end{pmatrix}, V_{Add} = (V, sub\hat{V}), T_{add} = \begin{pmatrix} T & P \\ O & \Delta \tilde{T} \end{pmatrix}$$

where $\varepsilon = (1, \ldots, 1)^T$ and $\tilde{T}$ denotes the matirx T removed the first column and first row. As a result,

$$M'_{add} = M_{add} T_{add} = \begin{pmatrix} M'_u & M_u P \\ \varepsilon M'_\alpha & \varepsilon M_\alpha P + \Delta \hat{M} \Delta \tilde{T} \\ M'_d & M_d P \end{pmatrix} \tag{20}$$

$\varepsilon M_\alpha P + \Delta\hat{M}\Delta\tilde{T}$, M_α are the adding blocks. It is important that, $\Delta\hat{M}\Delta\tilde{T} = \Delta\hat{M}' - sub\bar{M}\Delta\dot{T}$ where $\Delta\dot{T}$ denotes the first row of T removed of the first entity. Following the definition of our scalable LSSS matrix, $\Delta\bar{M} = (1, 1, \ldots, 1)^T$. Thus, $\Delta\hat{M}\Delta\tilde{T} = \Delta\hat{M}' - \varepsilon\Delta\dot{T}$. That is an efficient way to calculate adding blocks.

(2) $\varphi(\alpha) \in N(\mathbb{T})$. Then,

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

where V_{add} is generated by $Add(\mathbb{T}, \alpha, sub\mathbb{T})$. Similarly, we get

$$M_{add} = \begin{pmatrix} M & O \\ \varepsilon V_{add} & \Delta\hat{M} \end{pmatrix}, T_{add} = \begin{pmatrix} T & P \\ O & \Delta\tilde{T} \end{pmatrix}$$

where $P \in Z^{l \times \Delta l}$ is generated at random. Thus,

$$M'_{add} = \begin{pmatrix} MT & MP \\ \varepsilon V_{add}T & \varepsilon V_{add}P + \Delta\hat{M}\Delta\tilde{T} \end{pmatrix} \tag{21}$$

There are two adding blocks generated in this case:

$$(V_{add}T, V_{add}P), \begin{pmatrix} MP \\ \Delta\hat{M}\Delta\tilde{T} \end{pmatrix}.$$

Remove$_M(\mathbb{T}, \gamma)$: Let access policy $\mathbb{T}$ be described by LSSS matrix $M' = MT$ and $\phi(\gamma)$ be the root of the removed subtree. M, T can be expressed as follows

$$M = \begin{pmatrix} M_u & O & subM_u \\ M_\gamma & subM_\gamma & O \\ M_d & O & subM_d \end{pmatrix}, T = \begin{pmatrix} T_u & P_{u_1} & P_{u_2} \\ O & T_\gamma & P_\gamma \\ O & O & T_d \end{pmatrix}$$

where $subM_\gamma$ is the block of M which can describe the removed subtree. We get

$$M' = \begin{pmatrix} M_uT_u & M_uP_{u_1} & M_uP_{u_2} + subM_uT_d \\ M_\gamma T_u & M_\gamma P_{u_1} + subM_\gamma T_\gamma & M_\gamma P_{u_2} + subM_\gamma P_\gamma \\ M_dT_u & M_dP_{u_1} & M_dP_{u_2} + subM_dT_d \end{pmatrix} = \begin{pmatrix} M'_{u_1} & M'_{u_2} & M'_{u_3} \\ M'_{\gamma_1} & M'_{\gamma_2} & M'_{\gamma_3} \\ M'_{d_1} & M'_{d_2} & M'_{d_3} \end{pmatrix} \tag{22}$$

Remove$(\mathbb{T}, \gamma)$ is run to get

$$M_{rmv} = \begin{pmatrix} M_u & subM_u \\ M_d & subM_d \end{pmatrix}, T_{rmv} = \begin{pmatrix} T_u & P_{u_2} \\ O & T_d \end{pmatrix}$$

As a result

$$M'_{rmv} = M_{rmv}T_{rmv} = \begin{pmatrix} M'_{u_1} & M'_{u_3} \\ M'_{d_1} & M'_{d_3} \end{pmatrix} \tag{23}$$

At the same time, the random vector V is removed from the blocks $V_j, \varphi(j) \in N(\gamma)$. $((M'^T_{u_2})^T, (M'^T_{d_2})^T)^T$ is the removing block generated by this function.

4 Scalable Attribute-Based Encryption Scheme with Block LSSS Matrix

As shown in Fig. 3, the system model of scalable ABE scheme consists of the following entities:

Authority: Authority is global 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 with 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 most computation during policy updating.

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

User: users access ciphertext CT stored in cloud.

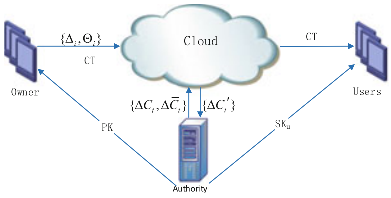


Fig. 3. System model.

4.1 Bilinear Map

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

Definition 2 (Bilinear Map). *Assume G_0, G_T are two multiplicative cyclic groups with prime order p, and g is a generator of G_0. A function $e : G_0 \times G_0 \rightarrow G_T$ is a bilinear map if it 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 map group G_0. Let $g \in G_0$ and $M = (m_{i,j})_{m\times n}$, g^M denotes a matrix in G_0 is shown as follows:

$$g^M = \begin{pmatrix} g^{m_{1,1}} & \dots & g^{m_{1,n}} \\ \vdots & & \vdots \\ g^{m_{m,1}} & \dots & g^{m_{m,n}} \end{pmatrix} \tag{24}$$

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

4.2 Functional Modules

The scheme includes five functional modules: *Setup*, *KeyGen*, *Encrypt*, *Decrypt* and *Updata*.

Setup(1^l): Authority generates a tuple $\{p, G_0, G_T, e\}$ as public parameters where $p \in Z$, G_0, G_T are groups with order p and $e : G_0 \times G_0 \to G_T$ is a bilinear map. Then, it picks $g_1, g_2 \in G_0$, and $\gamma_1, \gamma_2 \in Z_p$ at random and sets $\gamma = \gamma_1\gamma_2$. Let $S = \{A_1, \ldots, A_n\}$ be a set of attributes defined by authority and $s = \{a_1, \ldots, a_n\}$ be the image set of S in Z_p. PK and MK are calculated as follows:

$$PK = \{Y = e(g_1, g_2)^w, g_2, P' = g_1, P'' = g_1^{\gamma}, P_r = g_1^{\gamma_1}, P_c = g_1^{\gamma_2}, P_i = g^{\gamma a_i}, 1 \le i \le n\}$$

$$MK = \{w, \gamma_1, \gamma_2, \gamma, a_i, 1 \le i \le n\}$$

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:

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

Encrypt$(M, M(\mathbb{T}), \rho, PK)$: Let M be the plaintext and $M(\mathbb{T}) \in Z_p^{l \times m}$ be the block LSSS matrix describing access policy $\mathbb{T}$. Owner calculates $C_0 = MY^s$ and $C' = P'^s$ where $s \in Z_p$ is chosen at random. Then, he picks a vector $V = (s, v_2, \ldots, v_m)$ and calculates $C_k = P_{\rho(k)}^{r_k} P''^{s_k}, \bar{C}_k = P'^{r_k}$ where $s_k = VM(\mathbb{T})_k$ and $r_k \in Z_p$. Finally, the ciphertext is:

$$CT = \{C_0, C', C_k, \bar{C}_k, 1 \le i \le l\} \tag{25}$$

Decrypt$(C, M(\mathbb{T}), SK)$: Let $\sum_{\rho(k) \in I_u} u_k M(\mathbb{T})_k = \mathbf{e_1}$ where $I_u \subset U_s$ and $u_k \in Z$. The plaintext is recovered as follows:

$$M = \frac{C_0}{e(C', D')} \prod_{\rho(k) \in I_u} \left(\frac{e(\bar{C}_k, D_{\rho(k)})}{e(C_k, D'')}\right)^{u_k} \tag{26}$$

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

$$\prod_{\rho(k) \in I_u} \left(\frac{e(\bar{C}_k, D_{\rho(k)})}{e(C_k, D'')}\right)^{u_k} = e(g_1, g_u)^{-\gamma V \sum_{\rho(k) \in I_u} u_k M(\mathbb{T})_k} = e(g_1, g_u)^{-\gamma s}$$

Thus, Eq. (26) holds.

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

$$\Delta_i = \begin{cases} P_c^{\mathbf{v}_i} & l_i \ge m_i \\ P_r^{\mathbf{v}_i M_i} & \text{Otherwise} \end{cases}, \Theta_i = \begin{cases} (P'')^{\mathbf{w}_i} & l_i' \ge m_i' \\ (P'')^{\mathbf{w}_i R_i} & \text{Otherwise} \end{cases}$$

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

$$\Delta C_t' = \begin{cases} \Delta_i^{M_{i,j}} (P_k)^{r_{i,j}} & l_i \ge m_i \\ \Delta_{i,j} (P_k)^{r_{i,j}} & \text{Otherwise} \end{cases}, \Delta\bar{C}_t' = (P')^{r_{i,j}}$$

where $t = \kappa_i(j)$, $A_k = \rho(t)$, $M_{i,j}$ denotes the j^{th} row of M_i, $\Delta_{i,j}$ denotes the j^{th} entity of Δ_i and $r_{i,j} \in Z$ is chosen randomly. Then, all C'_t is sent to the authority, who computes:

$$\Delta C_t = \begin{cases} (\Delta C'_t)^{\gamma_2} & l_i \geq m_i \\ (\Delta C'_t)^{\gamma_1} & \text{Otherwise} \end{cases}, \Delta \bar{C}_t = \begin{cases} (\Delta \bar{C}'_t)^{\gamma_2} & l_i \geq m_i \\ (\Delta \bar{C}'_t)^{\gamma_1} & \text{Otherwise} \end{cases}$$

Then, the ciphertext is updated as follows

$$C_t \leftarrow C_t \Delta C_t, \bar{C}_t \leftarrow \bar{C}_t \Delta \bar{C}_t$$

If C_t and $\bar{C}_t$ are non-existence, they were initialed as $C_t = \bar{C}_t = 1$. For the removing blocks, the cloud calculates

$$\Theta C'_t = \begin{cases} \Theta_i^{R_{i,j}} (P_k)^{r_{i,j}} & l'_i \geq m'_i \\ \Theta_{i,j} (P_k)^{r_{i,j}} & \text{Otherwise} \end{cases}, \Theta \bar{C}'_t = (P')^{r_{i,j}}$$

where $t = \mu_i(j)$, $A_k = \rho(j)$, $R_{i,j}$ denotes the j^{th} row of R_i, $\Theta_{i,j}$ denote the j^{th} entity of Θ_i and $r_{i,j} \in Z$ is chosen randomly. Then,

$$C_t \leftarrow C_t (\Theta C_t)^{-1}, \bar{C}_t \leftarrow \bar{C}_t \Theta (\bar{C}_t)^{-1}.$$

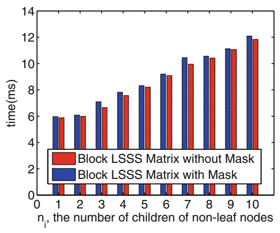
5 Performance Evaluation

5.1 Performance Analysis of Block LSSS Matrix

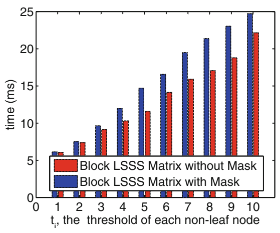
Block LSSS matrix is intuitive and easy to compute. In certain scenarios, the block LSSS matrix needs to sacrifice its readability to increase the security or extend the matrix space via multiplying a full rank mask matrix. Choosing an upper triangular matrix as the mask matrix is an effective way to maintain the scalability of block LSSS matrix. These mask matrices require moderate computation during matrix generation and scaling. We simulate the operation time for each type of matrix functions proposed in this paper. The simulation is run on a PC with an Intel Core 2 Duo CPU at 3.14 GHz and 8.00 GB RAM. The code is written in MATLAB and simulate the generating and managing of block LSSS matrix. The simulation results are shown in Figs. 4 and 5.

Figure 4 describes the efficiency of generation of block LSSS matrix. Let $\mathbb{T}$ be an access tree must be transformed into Matrix, n be the number of non-leaf $node_i$ of $\mathbb{T}$ and $node_i$ be a t_i-out-of-n_i node. We set all t_i equal and all n_i equal in the simulations. As shown in Fig. 4, the generation time increases with n_i, t_i and n. But, the mask of block LSSS matrix only costs a little time that is much less than the total generation time.

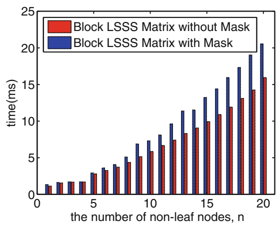
Figure 5 shows the operation time of the scaling functions. Let $M \in Z^{l \times m}$ and $subM \in Z^{subl \times subm}$ denote the LSSS matrix and modifying/adding/removing submatrix, respectively. Figure 5(a) describes the runtime of Thr and Thr_M increases with threshold increment $|\Delta t|$. Figure 5(b), (c) and (d) show the increase of runtime of scaling functions $Path$ and $Path_M$, Add and Add_M, $Remove$ and $Remove_M$ incurred by the size increasing of $subM$ respectively. Similarly, Fig. 5(e), (f), (g) and (h) describe the runtime of the above scaling functions increase with the size increase of M. All the scaling functions are efficient and cost less time than the generation function. Figure 5 is also shows that, the additional computation time of scaling functions caused by mask matrix is limited and acceptable.



(a) $n = 10, t_i = 1$

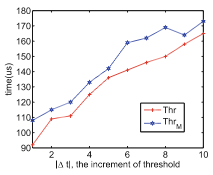


(b) $n = 10, n_i = 10$

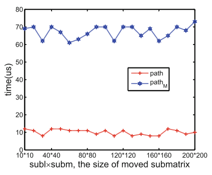


(c) $t_i = 10, n_i = 10$

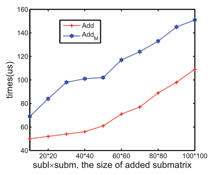
Fig. 4. Generation time of Block LSSS Matrix



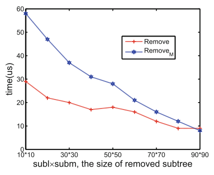
(a)



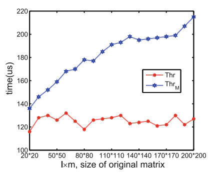
(b)



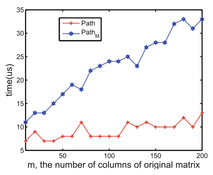
(c)



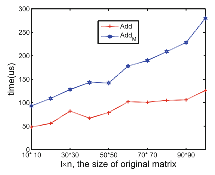
(d)



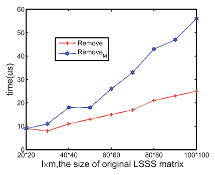
(e)



(f)



(g)



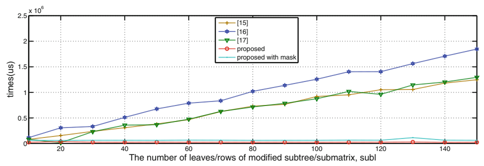
(h)

Fig. 5. Computation time of Scaling Functions

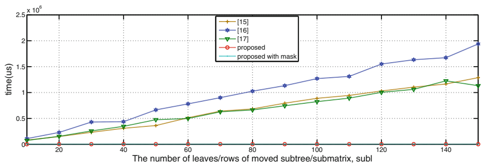
5.2 Performance Analysis of Scalable ABE Scheme

The most advantage of our scalable ABE scheme is efficiency and low-overhead. Comparing with [15–17], our scheme is more efficient on various types of updating operations for access policy. We also give the simulation of these schemes. The simulation is run on a Linux virtual machine with a CPU at 3.16 GHz and 1.00 GB RAM. It uses PBC lab to simulate the bilinear group operation. In fact, the policy managing operations of scheme [16] is too complex to evaluate. In order to compare, we only consider the optimum case of scheme [16] in the simulation.

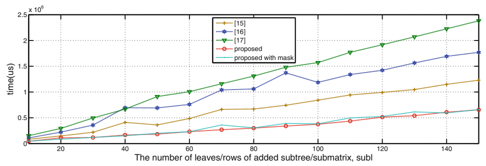
Figure 6 describes the operation time of policy updating of various schemes for the data owner. Figure 6(a) shows the computation time of modifying the threshold of a node n_j where the x axis represent *subl*, the number of leaves in the subtree with root n_j. In this situation, the operation time of our scheme is incurred by the increment/decrement of threshold $|\Delta D|$ while the time of others are all incurred by *subl*. Clearly, our scheme is more efficient than others because $|\Delta D|$ is much less than *subl*. As shown in Fig. 6(b), our scheme takes only a little time to move a subtree. Different from other schemes, the runtime of this our scheme is not really effected by the size of modified subtree but close a small constant. That implies, our scheme cost the lowest computation in this case. From Fig. 6(c), we can see that the computation time of adding a subtree of our scheme is minimum. Because the access policy of our



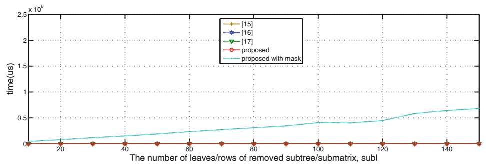
(a) Modifying of Threshold



(b) Moving of Subtree



(c) Adding of Subtree



(d) Removing of Subtree

Fig. 6. Simulation of policy updating operations

scheme is scalable, it can process the adding part of access policy only and do not need to reconstruct other parts. That make our scheme more efficient than others. As show in Fig. 6(d), removing a subtree costs little time can be ignored in all of these schemes except of masked case of our scheme. Actually, only the masked block LSSS matrix generates increment of attribute ciphertexts in this case. In other schemes, the access policies are updated via removing the invalid attribute ciphertexts immediately. In summary, the ABE scheme with block LSSS matrix, the scaling policy, is more efficient in access policy managing than others.

6 Conclusion

We provided a scalable access policy, block LSSS matrix, which is constructed by a set of node matrixes. The block matrix is efficient to generate and update. Eight scaling functions are given to update the block LSSS matrix, each being efficient and intuitive. We construct a scalable ABE scheme which describes the access policy by block LSSS matrix. In the scheme, data own updates access policy by scaling functions and generates minimum increment set Δ and Θ. Authority and cloud update ciphertext stored in the cloud via Δ and Θ. In the updating process, most computation is done by the authority and the cloud. The data owner faces rather low computational complexity to manage access privilege of data stored in cloud. In brief, the ABE scheme with scaling access policy is great improved of policy manageability.

Acknowledgments. This work is supported by the National Science Foundation of China (No.61373040, No.61173137), The Ph.D. Programs Foundation of Ministry of Education of China (20120141110073), Key Project of Natural Science Foundation of Hubei Province (No. 2010CDA004).

References

1. Blakley, G.R., Kabatianskii, G.A.: Linear algebra approach to secret sharing schemes. In: Chmora, A., Wicker, S.B. (eds.) Information Protection 1993. LNCS, vol. 829. Springer, Heidelberg (1994)
2. Benaloh, J., Leichter, J.: Generalized secret sharing and monotone functions. In: Goldwasser, (ed.) Advances in Cryptology – CRYPTO 1988. LNCS, vol. 403, pp. 27–35. Springer, New York (1990)
3. Bertilsson, M., Ingemarsson, I.: A construction of practical secret sharing schemes using linear block codes. In: Zheng, Y., Seberry, J. (eds.) AUSCRYPT 1992. LNCS, vol. 718, pp. 27–35. Springer, Heidelberg (1993)
4. Brickell, E.F.: Some ideal secret sharing schemes. In: Quisquater, J.-J., Vandewalle, J. (eds.) EUROCRYPT 1989. LNCS, vol. 434, pp. 468–475. Springer, Heidelberg (1990)
5. Massey, J.L.: Minimal codewords and secret sharing. In: Proceedings of the 6th Joint Swedish-Russian International Workshop on Information Theory, pp. 276–279 (1993)
6. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)
7. Simonis, J., Ashikhmin, A.: Almost affine codes. Des. Codes Crypt. **14**(2), 179–197 (1998)
8. Chellappa, R.: Intermediaries in Cloud-Computing: A New Computing Paradigm. INFORMS Annual Meeting, Dallas (1997)
9. Wu, J., et al.: Cloud storage as the infrastructure of cloud computing. In: International Conference on Intelligent Computing and Cognitive Informatics, pp. 380–383. IEEE (2010)
10. Abu-Libdeh, H., Princehouse, L., Weatherspoon, H.: RACS: a case for cloud storage diversity. In: Proceedings of the 1st ACM Symposium on Cloud Computing, pp. 229–240. ACM (2010)
11. Kamara, S., Lauter, K.: Cryptographic cloud storage. In: Sion, R., Curtmola, R., Dietrich, S., Kiayias, A., Miret, J.M., Sako, K., Sebé, F. (eds.) RLCPS, WECSR, and WLC 2010. LNCS, vol. 6054, pp. 136–149. Springer, Heidelberg (2010)
12. Stadler, M.A.: Publicly verifiable secret sharing. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 190–199. Springer, Heidelberg (1996)
13. Nikov, V., Nikova, S.: New monotone span programs from old. IACR Cryptology ePrint Archive 2004, p. 282 (2004)
14. Karchmer, M., Wigderson, A.: On span programs. In: Structure in Complexity Theory Conference, pp. 102–111 (1993)

15. Goyal, V., 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, pp. 89–98. ACM (2006)
16. Sahai, A., Seyalioglu, H., Waters, B.: Dynamic credentials and ciphertext delegation for attribute-based encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 199–217. Springer, Heidelberg (2012)
17. Yang, K., 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, pp. 2013–2021. IEEE (2014)
18. Lewko, A., Waters, B.: Decentralizing attribute-based encryption. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 568–588. Springer, Heidelberg (2011)
19. Zhen, L., Cao, Z., Wong, D.S.: Efficient generation of linear secret sharing scheme matrices from threshold access trees. Cryptology ePrint Archive, Report 2010/374. http://eprint.iacr.org/2010/374
20. Xavier, N., Chandrasekar, V.: Cloud computing data security for personal health record by using attribute based encryption. Bus. Manag. **7**(1), 209–214 (2015)
21. Xhafa, F., et al.: Designing cloud-based electronic health record system with attribute-based encryption. Multimedia Tools Appl. **74**(10), 3441–3458 (2015)
22. Horváth, M.: Attribute-based encryption optimized for cloud computing. In: Italiano, G.F., Margaria-Steffen, T., Pokorný, J., Quisquater, J.-J., Wattenhofer, R. (eds.) SOFSEM 2015-Testing. LNCS, vol. 8939, pp. 566–577. Springer, Heidelberg (2015)
23. Khedkar, S.V., Gawande, A.D.: Data partitioning technique to improve cloud data storage security. Int. J. Comput. Sci. Inf. Technol. **5**(3), 3347–3350 (2014)
24. Wei, L., et al.: Security and privacy for storage and computation in cloud computing. Inf. Sci. **258**, 371–386 (2014)
25. Meenakshi, I.K., George, S.: Cloud server storage security using TPA. Int. J. Adv. Res. Comput. Sci. Technol. **2**(1), 295–299 (2014)
26. Shetty, J., Anala, M.R., Shobha, G.: An approach to secure access to cloud storage service. Int. J. Res. **2**(1), 364–368 (2015)
27. Hohenberger, S., Waters, B.: Online/Offline attribute-based encryption. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, pp. 293–310. Springer, Heidelberg (2014)