2012

Privacy enhanced data outsourcing in the cloud

Miao Zhou
University of Wollongong, mz775@uow.edu.au

Yi Mu
University of Wollongong, ymu@uow.edu.au

Willy Susilo
University of Wollongong, willy_susilo@uow.edu.au

Jun Yan
University of Wollongong, jyan@uow.edu.au

Liju Dong
University of Wollongong, liju@uow.edu.au

Publication Details

Zhou, M., Mu, Y., Susilo, W., Yan, J. & Dong, L. (2012). Privacy enhanced data outsourcing in the cloud. Journal of Network and Computer Applications, 35 (4), 1367-1373.

Privacy enhanced data outsourcing in the cloud

Abstract

How to secure outsourcing data in cloud computing is a challenging problem, since a cloud environment cannot been considered to be trusted. The situation becomes even more challenging when outsourced data sources in a cloud environment are managed by multiple outsourcers who hold different access rights. In this paper, we introduce an efficient and novel tree-based key management scheme that allows a data source to be accessed by multiple parties who hold different rights. We ensure that the database remains secure, while some selected data sources can be securely shared with other authorized parties.

Keywords

outsourcing, cloud, privacy, enhanced, data

Disciplines

Physical Sciences and Mathematics

Publication Details

Zhou, M., Mu, Y., Susilo, W., Yan, J. & Dong, L. (2012). Privacy enhanced data outsourcing in the cloud. Journal of Network and Computer Applications, 35 (4), 1367-1373.

Privacy Enhanced Data Outsourcing in the Cloud

Miao Zhou, Yi Mu, Willy Susilo

Centre for Computer and Information Security Research, School of Computer Science and Software Engineering,University of Wollongong, Wollongong, NSW 2522, Australia

Jun Yan

School of Information Systems and Technology, University of Wollongong, Wollongong, NSW 2522, Australia

Liju Dong

School of Computer Science and Software Engineering,University of Wollongong, Wollongong, NSW 2522, Australia
School of Information Science and Engineering, Shenyang University, Shenyang 110044, P. R. China

Abstract

How to secure outsourcing data in cloud computing is a challenging problem, since a cloud environment cannot been considered to be trusted. The situation becomes even more challenging when outsourced data sources in a cloud environment are managed by multiple outsourcers who hold different access rights. In this paper, we introduce an efficient and novel tree-based key management scheme that allows a data source to be accessed by multiple parties who hold different rights. We ensure that the database remains secure, while some selected data sources can be securely shared with other authorized parties.

Keywords:
Access Control, Data Outsourcing, Cloud Computing

1. Introduction

Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (software, data storage, network, etc.) that can be rapidly provisioned and released with minimal management effort or service provider interaction. The datacenter hardware and software is what we call a cloud and the services over the Internet provided by the cloud are referred as cloud services. With these cloud-based services, it

Email addresses: {mz775,ymu,wsusilo}@uow.edu.au (Miao Zhou, Yi Mu, Willy Susilo), jyan@uow.edu.au (Jun Yan), liju@uow.edu.au (Liju Dong)

is possible for enterprises to relocate their IT service and maintenance outside of their organization, regardless of geography and available hardware devices. Figure 1 shows the variety of cloud services, and the roles of the people that as data owner, cloud provider and users in different deployment models of cloud computing.

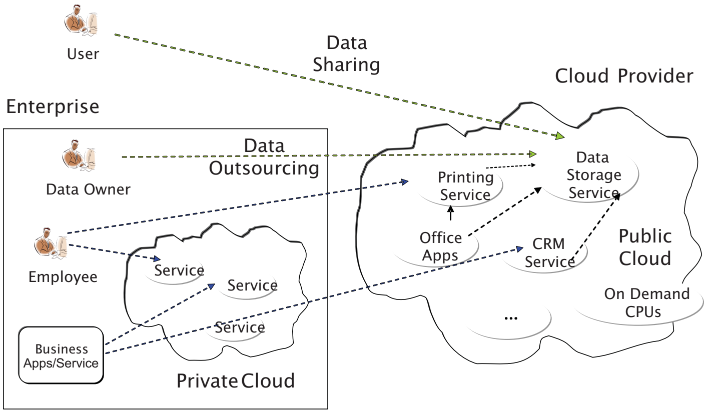

Figure 1: Cloud Computing and Data Outsourcing.

The data stored in a cloud database is considered as data outsourcing, since they are managed by an external party. Among kinds of cloud computing services, we have especially seen the dramatic growth of cloud storage services, with which enterprises outsource their data into cloud environment for location independent resource pooling, rapid resource elasticity and usage-based pricing. For example, cloud Storage Services such as Microsoft's Azure storage and Amazon's S3 have gained a lot of popularity recently.

However, while more and more enterprises store their private data on the cloud storages, which are generally managed by untrusted parties, secure and privacy have become major concerns. As a countermeasure, Microsoft has recently deployed a virtual private storage service [1]. Although the recent efforts in secure cloud computing, there are a number of unsolved security issues. One of such issues is how to protect confidentiality and privacy of user data, while those data have to be shared/managed by multiple parties. This is also the issue to be addressed in this paper.

For a consideration of security, the outsourced data are generally encrypted so that only authorized users can access them. Generally, these outsourced data consist of many data blocks, hence the management of encryption keys is a major challenge.

The classical tree-based hierarchy schemes such as RFC2627 [2] and the scheme proposed by Wong *et al.* [3] have been widely used in group key management. With the tree-based ideas, some key management methods of access hierarchies for data outsourcing have been proposed [4, 5, 6, 7, 8]. These methods provide some useful solutions to minimize the number of cryptographic keys, which have to be managed and stored. Aiming to provide secure and efficient access to outsourced data, Wang, Li *et al.*[4] proposed a tree-based cryptographic key management scheme for data storages in the cloud. They referred the scenario to as "owner-write-users-read". Their tree-based key management structure is similar to a traditional one, where a single root node holds the master key that can be used to derive other node keys. Each node key can be used to derive the keys of its children in the hierarchy. With their scheme, a data block stored in the cloud can be updated by a party who holds either the specific decryption key or a node key corresponding to one of its parents. Assume there is an outsourcing server authorized to manage a node (not the root node) that has several child nodes, then in this case, the outsourced party is granted the node key, which can be used to derive all sub-keys for its child nodes. In another word, once a parent node in the tree is given, all the children would be known. This is a common problem which exists in many tree-based key management schemes. The existing ones such as[4, 9, 7, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19] could work perfectly, only when all legitimate node users are authorized to access all the child nodes under the specific one.

Considering this problem, we propose a practical application for private data management, which we name it as OWUR/W (owner-write-users-read/write) applications, where a data source protected with a node key in a key management tree can be shared with or managed by another party without compromising the security of the data encrypted with its child nodes' keys. Additionally, data can be updated not only by the data owner, but also by other legitimate parties. We found that this scenario is very useful in outsourcing management. We notice that other existing schemes do not offer this feature.

Intuitively, we want to realize that the encrypted data block associated with a node can be decrypted by multiple decryption keys where one of them is associated with the tree and can be utilized to generate its keys children's keys, while other decryption keys are only used to decrypt the data block stored in the node. Let us assume two decryption keys (d_1, d_2), assigned to a node, where one of them is associated with the tree (let us assume that d_1 is the key associated with the tree and is known to the manager only). Both decryption keys are associated with the unique encryption key, e. For a user, who is authorized to access only the data block stored in the node and should not have access to its children, the manager only grants d_2 to the user. With d_2, the user can decrypt the data block but cannot generate the decryption keys of this node's children. We believe that this method offers an additional privacy protection to the outsourced data.

In this paper, we propose a novel tree-based key management scheme and show that it can indeed capture the idea given above. We also show how to apply our scheme to protect outsourced data in a cloud.

The remainder of this paper is organized as follows. In section 2, we present our data outsourcing model for cryptographic cloud storage. In section 3, we describe our key derivation hierarchy for key management in cloud storage. In section 4, we give a concrete example and present the main construction, including the encryption method and the detailed algorithms. In section 5, we analyze the data access procedure. In section 6, we discuss an extension of the proposed scheme. In section 7, we conclude this paper.

2. The Model

In this section, we present our data outsourcing model, following the application scenario presented in section 1. An illustration of our model is presented in Fig. 7. The system consists of four major parties:

- The cloud Provider (P), who provides third-party data storage services.
- The original data owner (O), who holds the master (root) key and is responsible to set up the key management system.
- The sub-tree data manager (M), who holds an authorized node key, which can be used to derive all decryption keys for its child nodes. Notice that we assume each node has two decryption keys: one can be used to derive all decryption keys of its children and the other can only be used to decrypt the encrypted data in the specific node.
- A user or another sub-tree data manager (U), who will probably use or share a data block at a node managed by M but does not hold the full administrative right of deriving the decryption keys of the children of this node.

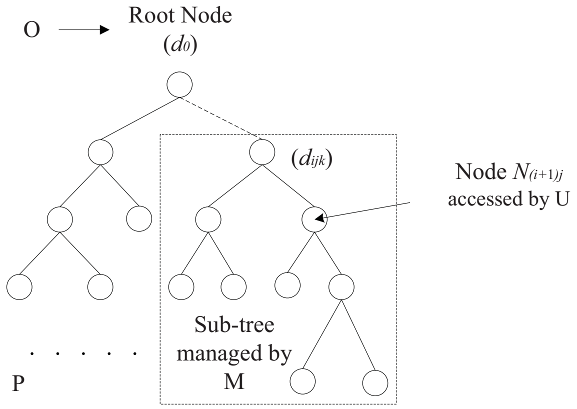

Figure 2: The data outsourcing model.

Our encryption method is asymmetric, in the sense that the encryption key and decryption key(s) are different. For simplicity, we assume that the data stored in each node is encrypted by one encryption key associated with two decryption keys. One of these two decryption keys is used for the decryption of the database located at the corresponding node and the generation of sub-keys for the child nodes, while another one can only be used for the decryption of the database at the same node.

Let e_{ij} denote encryption keys and d_{ijk} the corresponding decryption keys, respectively, where i denotes the level of a tree, j the index of nodes and k the index of decryption keys. The root master key is denoted by d_0. Let N_{ij} denote a node in the tree. Using a binary tree as an example, our model can be described as follows:

- The original owner O generates a master (root) key, d_0, which can be used to derive all other decryption keys and encryption keys in a tree.
- Each node in the tree obtains a master decryption key d_{ij1} and the secondary decryption key d_{ij2}, generated from the root key. We allow the secondary decryption key d_{ij2} to be derived from d_{ij1}.
- The sub-tree data manager M obtains a key d_{ij1} as its master key, which can be used to generate all node keys of the sub-tree, including the secondary keys.
- User U can request a secondary decryption key d_{ij2} from M for accessing the encrypted data stored in node N_{ij}

This key management scheme holds all features from a normal binary tree hierarchy and introduces the new secondary key, which enables the flexibility in key management and additional privacy protection.

3. Key Derivation Hierarchy

Without losing generality, we assume that the outsourced data contain n blocks and $2^{(i-1)} \leqslant n \leqslant 2^i$ where i denotes the level of the tree. Therefore, we can construct a complete binary tree from Node N_0 to N_{ij} where i and j denote the level of a tree and j the node index, respectively.

Before defining the key derivation tree, we first define Key Value.

Definition 1. *(Key Value) Except the root node N_0, any node N_{ij} in the key derivation tree T, has a key value K_{ij} of two decryption keys d_{ij1} and d_{ij2}. These two decryption keys are also denoted as key pair (d_{ij1},d_{ij2}). Such a key pair can generate the encryption key e_{ij} for this node.*

The definition of Key Derivation Tree is given as follows. Notice that the encryption key associated with a key value is less important in the key derivation.

Definition 2. *(Key Derivation Tree) A key derivation tree, denoted T, is a tree $T = \langle N, K \rangle$, rooted at vertex N_0. Any node N_{ij} except the leaves, can derive its child nodes of indices $i(2j-1)$ (for the left) and $i(2j)$ (for the right), for $i = 1, 2, ...$ and $j = 1, ..., 2i$, while its parent (if any) is found at index $(i-1)\lceil \frac{j}{2} \rceil$. $K_{ij} \subseteq K$, denotes the key value of each node N_{ij}, where the key value consists of a set of decryption keys corresponding to this node.*

To construct the key derivation tree, we choose a cryptographic one-way hash function as our key generation function: $H : \{0,1\}^* \rightarrow \mathcal{Z}_q$, which can be used to compute the decryption key of child nodes of any node N_{ij}, while hard to invert the key of N_{ij}. The key value K_{ij} of node N_{ij} is represented by $K_{ij} \leftarrow (d_{ij1}, d_{ij2})$ where d_{ij1} denotes the master decryption key and d_{ij2} denotes the secondary decryption key. The one-way hash function H is also being used to compute d_{ij2} from the input d_{ij1}: $d_{ij2} \leftarrow H(d_{ij1})$. We illustrate the key derivation hierarchy in Figure 2, where $i \geqslant 0, j \geqslant 1$.

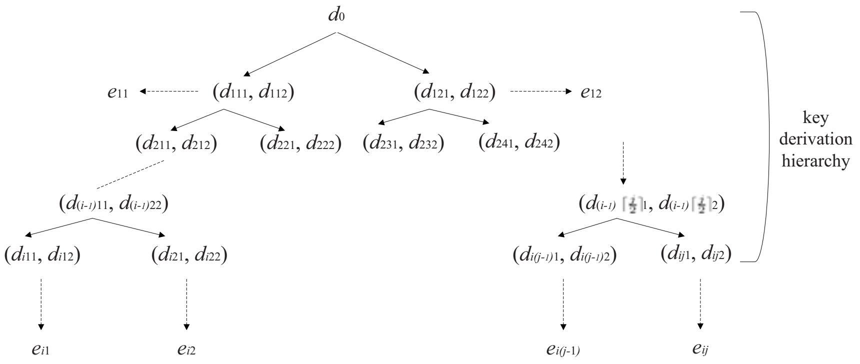

Figure 3: Key derivation hierarchy.

With the root key d_0 for node N_0, we can derive all the key pairs:

$$d_0 \rightarrow d_{111} \rightarrow d_{112}, \quad d_0 \rightarrow d_{121} \rightarrow d_{122}.$$

$$d_{111} \rightarrow d_{211} \rightarrow d_{212}, \quad d_{111} \rightarrow d_{221} \rightarrow d_{222}.$$

$$d_{121} \rightarrow d_{231} \rightarrow d_{232}, \quad d_{121} \rightarrow d_{241} \rightarrow d_{242}.$$

$$\ldots \quad \ldots$$

$$d_{(i-1)\lceil\frac{j}{2}\rceil 1} \rightarrow d_{i(j-1)1} \rightarrow d_{i(j-1)2}, d_{(i-1)\lceil\frac{j}{2}\rceil 1} \rightarrow d_{ij1} \rightarrow d_{ij2}$$

$$\ldots \quad \ldots$$

The encryption keys are generated from key pairs which contain the master decryption key and the secondary decryption key, for example,

$$(d_{111}, d_{112}) \rightarrow e_{11}, \quad (d_{121}, d_{122}) \rightarrow e_{12},$$

$$(d_{211}, d_{212}) \rightarrow e_{21}, \quad (d_{221}, d_{222}) \rightarrow e_{22},$$

$$\ldots \quad \ldots$$

$$(d_{i(j-1)}, d_{i(j-1)2}) \rightarrow e_{i(j-1)}, \quad (d_{ij1}, d_{ij2}) \rightarrow e_{ij}$$

$$\ldots \quad \ldots$$

To derive child keys, the original data owner O conducts the following computations. For a key pair (d_{ij1},d_{ij2}) of node (i, j), its child on the left can be calculated as

$$(d_{(i+1)(2j-1)1}, d_{(i+1)(2j-1)2}) = (H(d_{ij1}\|(2j-1)), H(H(d_{ij1}\|(2j-1))))$$

and its child on the right can be calculated as

$$(d_{(i+1)(2j)1}, d_{(i+1)(2j)2}) = (H(d_{ij1}\|2j), H(H(d_{ij1}\|2j))).$$

Other sub-keys can generated accordingly. In this way, the whole key derivation tree can be constructed. We will give a concrete scheme in the following section.

4. The Concrete Scheme

Having demonstrated how our scheme works, we now provide a concrete construction. We borrow the polynomial introduced in [20] and demonstrate how to apply it to our key derivation tree.

4.1. The Polynomial Function

The security of this system relies on difficulty of computing discrete logarithm. The protocols are based on a polynomial function and a set of exponentials. Let p, q be two large prime numbers such that $q|p-1$, and $g \in \mathcal{Z}_p^*$ be a generator of order q. Let $x_i \in_U \mathcal{Z}_q$ for $i = 0, 1, 2..., n$ be a set of integers. The polynomial function of order n is constructed as follows.

$$f(x) = \prod_{i=1}^{n}(x - x_i) \equiv \sum_{i=0}^{n} a_i x^i \mod q,$$

where $\{a_i\}$ are coefficients:

$$\begin{aligned}
a_0 &= \prod_{j=1}^{n}(-x_j), \\
a_1 &= \sum_{i=1}^{n}\prod_{i\neq j}^{n}(-x_j), \\
&\cdots, \\
a_{n-2} &= \sum_{i\neq j}^{n}(-x_i)(-x_j), \\
a_{n-1} &= \sum_{i=1}^{n}(-x_j), \\
a_n &= 1.
\end{aligned}$$

It is noted that $\sum_{i=0}^{n} a_i x_j^i = 0$. This property is important for our scheme. Having the set $\{a_i\}$, we can then construct the corresponding exponential functions,

$$\{g^{a_0}, g^{a_1}, \cdots, g^{a_n}\} \equiv \{g_0, g_1, \cdots, g_n\} \mod p.$$

All elements here are computed under modulo p. For convenience, we will omit modulo p in the rest of this paper.

Now we are ready to construct an asymmetric-key system where the encryption key is the tuple $\{g_0, g_1, \cdots, g_n\}$ mapping to n decryption keys $\{x_i\}$.

4.2. Key Derivation Tree and Data Encryption

Let us use a binary tree as an example and (i, j) as an arbitrary node. Then the main construction contains four algorithms: *Key Generation*, *Encryption*, *Decryption* and *Key Derivation*.

4.2.1. Key Generation

The decryption keys are denoted by (d_{ij1}, d_{ij2}), which correspond to (x_1, x_2) in the 2-degree polynomial defined above, where $d_{ij2} = H(d_{ij1})$. For simplicity, we denote $(d_{ij1}, d_{ij2}) = (d_1, d_2)$. The encryption key corresponding to (d_1, d_2) is $e = (g_0, g_1, g_2)$, where $g_0 = g^{a_0} = g^{d_1 d_2}$, $g_1 = g^{a_1} = g^{-(d_1+d_2)}$, $g_2 = g^{a_2} = g$. For simplicity, we have omitted the subscripts of e_{ij}.

4.2.2. Encryption

The encryption algorithm takes as input a message $M \in \{0, 1\}^*$, the encryption key e, a random $k \in \mathcal{Z}_q$, and a generator $h \in \mathcal{Z}_p^*$, and outputs a ciphertext (c_1, c_2), where

$$c_1 \leftarrow (h^k \cdot g_0^k, g_1^k, g_2^k), \qquad c_2 = M \cdot h^k.$$

Let us rewrite c_1 as (b_1, b_2, b_3) for convenience. The encryption scheme is a variant of ElGamal encryption, which is proven to be secure under the assumption of Chosen Plaintext Attack when the group $\mathcal{Z}_p^*$ is properly selected. It can be easily converted into Chosen Ciphertext Security by Fujisaki-Okamoto transformation [21].

4.2.3. Decryption

This algorithm takes as input the ciphertext (c_1, c_2) and one of decryption keys d_1 and d_2, and outputs M. h^k can be computed from $b_1 \cdot b_2^{d_i} \cdot b_3^{{d_i}^2}$, for $i \in \{1, 2\}$. Thus, M can be computed as $M = c_2/h^k$.

4.2.4. Key Derivation

This algorithm takes as input the master decryption key d_{ij1} and a one way hash function $H : \{0, 1\}^* \rightarrow \mathcal{Z}_q$. It outputs the two child nodes of key d_{ij1}. During the key derivation procedure, the left child node can be computed as

$$(d_{(i+1)(2j-1)1},\ d_{(i+1)(2j-1)2}) = (H(d_{ij1} \| (2j-1)),\ H(H(d_{ij1} \| (2j-1))))$$

and the right child node can be computed as

$$(d_{(i+1)(2j)1},\ d_{(i+1)(2j)2}) = (H(d_{ij1}\|2j),\ H(H(d_{ij1}\|2j))).$$

By repeating this algorithm, the whole key derivation tree can be generated.

Our scheme also considers "Write" applications and allows the user to re-encrypt the data. This means that the encryption key e should be given to the user. The reader might think that the user could alter the encryption key by changing a new encryption key, as he can easily replace his existing decryption key with a different one. However, this action will fail if the data manager checks the correctness of the encryption regularly. An alternative solution is to use an RSA modulus and assume that two corresponding primes, which form the modulus, is only known to the manager. This change makes the encryption key unmalleable.

5. Data Access Procedure

In this section, we describe the data access procedure for four associated parties given in Section 2.

5.1. Notations

- M, O, P, U: abbreviated names appear as the four major parties as given in our model.
- M → O: m. M sends message m to O.
- k_{XY}: symmetric key shared between parties X and Y.
- T_X: a timestamp generated by X;

5.2. Data access procedure

Using N_{ij} as example, the data block is encrypted with e_{ij}, which is corresponding to two decryption keys (d_{ij1}, d_{ij2}). The data access procedure is described in two phases, shown in Figure 4 and Figure 5.

5.2.1. The First Phase

In the first phase, Data Owner O, Sub-tree Manager M and Cloud Provider P execute the following five steps.

1. M sends O a key request message: M_REQ, where M_REQ = {*sub_id*, T_M, MAC(k_{MO}, *sub_id*, T_M)}. MAC denotes the message authentication code with the key k_{MO} shared by M and O. After the original data owner O sets up the system, the first step in the data access procedure are run by O and its sub-tree data manager M. M sends O an access request message M_REQ in order to obtain the sub-tree root key. The *sub_id* field in this message provides the index of the sub-tree root node. Upon receiving the M_REQ message, O performs data-integrity validation by checking the MAC of *sub_id* and timestamp.

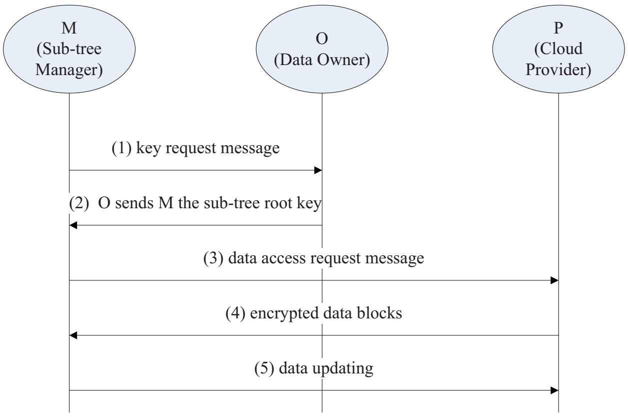
M
(Sub-tree
Manager)
O
(Data Owner)
P
(Cloud
Provider)
(1) key request message
(2) O sends M the sub-tree root key
(3) data access request message
(4) encrypted data blocks
(5) data updating

Figure 4: The first phase.

2. O sends M the sub-tree root key d_{ij1} encrypted with $\mathsf{E}_{k_{MO}}$. Upon receiving d_{ij1}, M can use it to derive the second key and other keys of its child nodes and access the the encrypted data block as given in the next step.

3. M sends P an access request message: M_REQ, where M_REQ $= \{$*sub_tree_id*, T_M, MAC(k_{MP}, *sub_tree_id*, T_M)$\}$

4. M accesses the stored data block.

5. With the sub-tree root key d_{ij1} for N_{ij}, M can update all the sub-tree nodes. We call this sub-procedure as a write step because M can modify and more importantly, re-encrypt the sub-tree data. After that, M sends the re-encrypted data block back to P.

5.2.2. The Second Phase

In the second phase, User U, Data Owner O and Cloud Provider P execute the following five steps:

1. U sends M a key request message: U_REQ, where U_REQ $= \{$*req_id*, T_U, MAC(k_{MU}, *req_id*, T_U)$\}$. This step runs between User U and the sub-tree data manager M. Without any interaction with the original data owner O, the user U starts the connection by sending a key request message U_REQ to the sub-tree manager M. MAC is the message authentication code using the key k_{MU}. The *req_id* field contains the node information for U. M performs data-integrity validation with the MAC, upon receiving the U_REQ message.

2. M sends U the secondary decryption key d_{ij2} with respect to the same node. U can use this decryption key to decrypt the same data block. If U is a legitimate user that can write the data, M also provides the encryptions key e_{ij} corresponding to this node.

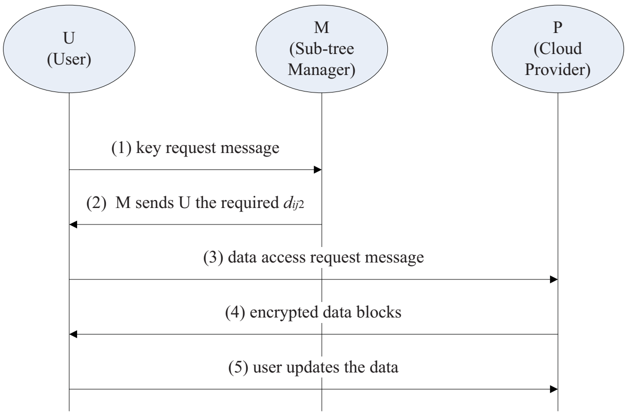
U
(User)
M
(Sub-tree
Manager)
P
(Cloud
Provider)
(1) key request message
(2) M sends U the required d_{ij2}
(3) data access request message
(4) encrypted data blocks
(5) user updates the data

Figure 5: The second process.

3. U sends P an access request: U_REQ = $\{req\_node\_id$, T_U, MAC$(k_{UP}$, req_node_id, $T_U)\}$.
4. U accesses the encrypted data block.
5. Given the encryption key e_{ij}, U can update this specific data node. We call this sub-procedure as a user write step where a legimate user can modify and re-encrypt the specific data block.

Our protocol can be directly applied to a practical application. We can apply it to the OWUR/W application, for example. There are many other practical applications such as securely outsourcing data management of a telecom, where the outsourced manager can only manage the registration of users, while the payment history can only be accessed by staff from the original telecom.

6. Generalization of Key Derivation Hierarchy

A flexible and efficient key management scheme should be adaptable and expandable for different application scenarios. For this consideration, we first expand our scheme into multiple branches and n sub-keys respectively. We then have a generalized key derivation hierarchy with a consideration of both cases.

6.1. Multiple Branches

We can easily expand our scheme to multiple branches ($m > 2$) as Figure 6.

Every parent node has m child nodes which can be derived in a similar way. Taking parent node (d_{1m1}, d_{1m2}) as an example, its first child node $(d_{2[m(m-1)+1]1}, d_{2[m(m-1)+1]2})$ can be computed as

$$(H(d_{1m1}||[m(m_1)+1]), H(H(d_{1m1}||[m(m_1)+1]))),$$

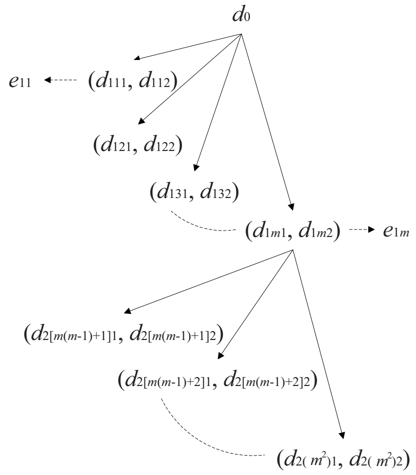
d_0
e_{11}
(d_{111}, d_{112})
(d_{121}, d_{122})
(d_{131}, d_{132})
(d_{1m1}, d_{1m2})
e_{1m}
$(d_{2[m(m-1)+1]1}, d_{2[m(m-1)+1]2})$
$(d_{2[m(m-1)+2]1}, d_{2[m(m-1)+2]2})$
$(d_{2(m^2)1}, d_{2(m^2)2})$

Figure 6: A key derivation tree with multiple branches.

and its last child node $(d_{2(m^2)1}, d_{2(m^2)2})$ can be computed as

$$(H(d_{1m1}||m^2, H(H(d_{1m1}||m^2))).$$

This expansion lowers the level of derivation hierarchy, by increasing the number of nodes in each level.

6.2. Multiple Sub-keys

We can also expand it to n sub-keys ($n > 2$), where these sub-keys map to a single encryption key used to encryption the corresponding data block. In regard to node N_{ij}, this n sub-keys are denoted by (d_{ij1},d_{ij2},...,d_{ijn}) and the encryption key by e_{ij}. Every decryption key can be used to decrypt the data block, while only the master decryption key d_{ij1} can derive n sub-keys in the tree.

Taking $n = 3$ as an example (shown in Figure 7), the decryption keys are denoted by (d_{ij1},d_{ij2},d_{ij3}), which correspond to (x_1, x_2, x_3) in the 3-degree polynomial defined in Section 4, where $d_{ij2} = H(d_{ij1}||1)$, $d_{ij3} = H(d_{ij1}||2)$. Denote $(d_{ij1}, d_{ij2}, d_{ij3})$ by (d_1, d_2, d_3), the encryption key corresponding to (d_1, d_2, d_3) is $e = (g_0, g_1, g_2, g_3)$, where $g_0 = g^{a_0} = g^{-d_1 d_2 d_3}$, $g_1 = g^{a_1} = g^{d_1 d_2 + d_1 d_3 + d_2 d_3}$, $g_2 = g^{a_2} = g^{-(d_1+d_2+d_3)}$, $g_3 = g^{a_3} = g$.

Combining both cases above, we can have a generalized key derivation tree. An m-branch n sub-key derivation tree is presented in Figure 8.

7. Conclusion

Key management and access control are important for secure cloud computing. As a traditional approach, tree-based key management has attracted

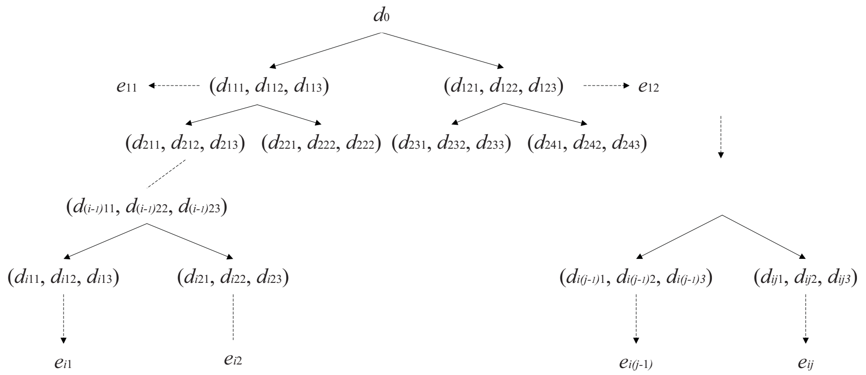
d_0
e_{11}
$(d_{111}, d_{112}, d_{113})$
$(d_{121}, d_{122}, d_{123})$
e_{12}
$(d_{211}, d_{212}, d_{213})$
$(d_{221}, d_{222}, d_{222})$
$(d_{231}, d_{232}, d_{233})$
$(d_{241}, d_{242}, d_{243})$
$(d_{(i-1)11}, d_{(i-1)22}, d_{(i-1)23})$
$(d_{i11}, d_{i12}, d_{i13})$
$(d_{i21}, d_{i22}, d_{i23})$
$(d_{i(j-1)1}, d_{i(j-1)2}, d_{i(j-1)3})$
$(d_{ij1}, d_{ij2}, d_{ij3})$
e_{i1}
e_{i2}
$e_{i(j-1)}$
e_{ij}

Figure 7: Take $n = 3$ as an example.

a lot of attention. We found that a traditional tree-based approach has some drawbacks in data oursourcing in that a node key holder to derive all child keys, which is also an important feature for key management. In order to solve the outsourcing problem and maintain the key management feature, we proposed OWUR/W applications for data sourcing and presented a secure and flexible tree-based key derivation hierarchy, which allows only the outsourcing party to access the data block located at a specified node while he cannot access the data blocked encrypted with child keys. We believe that our tree-based outsourcing key management opens up an entirely new approach for secure and flexible key management.

References

[1] S. Kamara, K. Lauter, Cryptographic cloud storage, in: R. Sion, R. Curtmola, S. Dietrich, A. Kiayias, J. Miret, K. Sako, F. Seb (Eds.), Financial Cryptography and Data Security, Vol. 6054 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2010, pp. 136–149.

[2] D. Wallner, E. Harder, R. Agee, Rfc2627: Key management for multicast: Issues and architectures.

[3] C. K. Wong, M. Gouda, S. S. Lam, Secure group communications using key graphs, in: Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication, SIGCOMM '98, ACM, New York, NY, USA, 1998, pp. 68–79.

[4] W. Wang, Z. Li, R. Owens, B. Bhargava, Secure and efficient access to outsourced data, in: Proceedings of the 2009 ACM workshop on Cloud computing security, CCSW '09, ACM, New York, NY, USA, 2009, pp. 55–66.

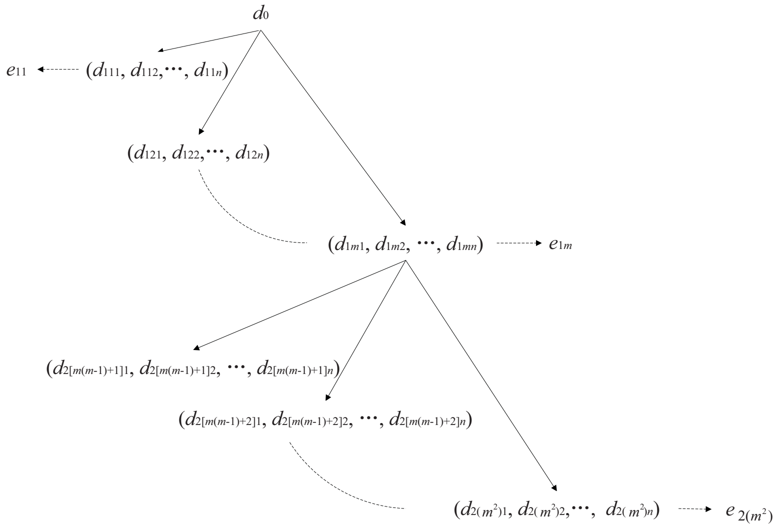
d_0
e_{11}
$(d_{111}, d_{112}, \cdots, d_{11n})$
$(d_{121}, d_{122}, \cdots, d_{12n})$
$(d_{1m1}, d_{1m2}, \cdots, d_{1mn})$
e_{1m}
$(d_{2[m(m-1)+1]1}, d_{2[m(m-1)+1]2}, \cdots, d_{2[m(m-1)+1]n})$
$(d_{2[m(m-1)+2]1}, d_{2[m(m-1)+2]2}, \cdots, d_{2[m(m-1)+2]n})$
$(d_{2(m^2)1}, d_{2(m^2)2}, \cdots, d_{2(m^2)n})$
$e_{2(m^2)}$

Figure 8: Expanding to M-Branch N-Tuple Hierarchy.

[5] E. Damiani, S. D. C. Vimercati, S. Jajodia, S. Paraboschi, P. Samarati, Balancing confidentiality and efficiency in untrusted relational dbmss, in: Proceedings of the 10th ACM conference on Computer and communications security, CCS '03, ACM, New York, NY, USA, 2003, pp. 93–102.

[6] C. Blundo, S. Cimato, S. De Capitani di Vimercati, A. De Santis, S. Foresti, S. Paraboschi, P. Samarati, Efficient key management for enforcing access control in outsourced scenarios, in: D. Gritzalis, J. Lopez (Eds.), Emerging Challenges for Security, Privacy and Trust, Vol. 297 of IFIP Advances in Information and Communication Technology, Springer Boston, 2009, pp. 364–375.

[7] S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, P. Samarati, Over-encryption: Management of access control evolution on outsourced data, in: Proceedings of the 33rd international conference on Very large data bases, VLDB '07, VLDB Endowment, 2007, pp. 123–134.

[8] M. J. Atallah, M. Blanton, N. Fazio, K. B. Frikken, Dynamic and efficient key management for access hierarchies, ACM Trans. Inf. Syst. Secur. 12 (2009) 18:1–18:43.

[9] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, G. Pelosi, P. Samarati, Preserving confidentiality of security policies in data outsourcing, in: Proceedings of the 7th ACM workshop on Privacy in the electronic society, WPES '08, ACM, New York, NY, USA, 2008, pp. 75–84.

[10] M. J. Atallah, K. B. Frikken, M. Blanton, Dynamic and efficient key management for access hierarchies, in: Proceedings of the 12th ACM conference on Computer and communications security, CCS '05, ACM, New York, NY, USA, 2005, pp. 190–202.

[11] Y. R. Yang, X. S. Li, X. B. Zhang, S. S. Lam, Reliable group rekeying: A performance analysis, in: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, SIGCOMM '01, ACM, New York, NY, USA, 2001, pp. 27–38.

[12] E. Damiani, S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, P. Samarati, Key management for multi-user encrypted databases, in: Proceedings of the 2005 ACM workshop on Storage security and survivability, StorageSS '05, ACM, New York, NY, USA, 2005, pp. 74–83.

[13] E. Damiani, S. De Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, P. Samarati, An experimental evaluation of multi-key strategies for data outsourcing, in: H. Venter, M. Eloff, L. Labuschagne, J. Eloff, R. von Solms (Eds.), New Approaches for Security, Privacy and Trust in Complex Environments, Vol. 232 of IFIP International Federation for Information Processing, Springer Boston, 2007, pp. 385–396.

[14] Y. Kim, A. Perrig, G. Tsudik, Simple and fault-tolerant key agreement for dynamic collaborative groups, in: Proceedings of the 7th ACM conference on Computer and communications security, CCS '00, ACM, New York, NY, USA, 2000, pp. 235–244.

[15] D. Naor, M. Naor, J. B. Lotspiech, Revocation and tracing schemes for stateless receivers, in: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology, CRYPTO '01, Springer-Verlag, London, UK, 2001, pp. 41–62.

[16] D. Liu, P. Ning, K. Sun, Efficient self-healing group key distribution with revocation capability, in: Proceedings of the 10th ACM conference on Computer and communications security, CCS '03, ACM, New York, NY, USA, 2003, pp. 231–240.

[17] D.-W. Kwak, S. J. Lee, J. W. Kim, E. Jung, An efficient lkh tree balancing algorithm for group key management, Communications Letters, IEEE 10 (3) (2006) 222 – 224.

[18] Y. Sun, K. Liu, Scalable hierarchical access control in secure group communications, in: INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies, Vol. 2, 2004, pp. 1296 – 1306 vol.2.

[19] S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, P. Samarati, A data outsourcing architecture combining cryptography and access control, in: Proceedings of the 2007 ACM workshop on Computer security architecture, CSAW '07, ACM, New York, NY, USA, 2007, pp. 63–69.

[20] Y. Mu, V. Varadharajan, K. Quac Nguyen, Delegated decryption, in: M. Walker (Ed.), Cryptography and Coding, Vol. 1746 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 1999, pp. 797–797.

[21] E. Fujisaki, T. Okamoto, Secure integration of asymmetric and symmetric encryption schemes, in: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology, CRYPTO '99, Springer-Verlag, London, UK, 1999, pp. 537–554.