2012 41st International Conference on Parallel Processing Workshops

Invitation-oriented TGDH: Key management for dynamic groups in an asynchronous communication model

István Lám, Szilveszter Szebeni, Levente Buttyán
Tresorium Kft, and Laboratory of Cryptography and Systems Security (CrySyS)
Budapest University of Technology and Economics
Budapest, Hungary
{*lam,szebeni*}*@tresorium.hu*, *buttyan@crysys.hu*

***Abstract*—In this paper, we propose a suite of group key management protocols that allows a group of users to agree on a shared group key, which can be used to protect a shared file system stored remotely in the cloud. Our protocols support the refreshment of the group key at each group membership modification. Compared to other similar solutions, the key novelty of our protocols is that they require asynchronous communication channels, and only constant number of exponentiations on average, worst case logarithmic number of exponentiations even if membership modifications are controlled by arbitrary user. Moreover, our scheme is fully decentralized, and its design is led by a practical, invitation-oriented approach.**

***Index Terms*—group-key agreement, dynamic groups, self-balancing key-tree**

I. Introduction

An increasing number of enterprises and private individuals choose cloud based data storage solutions instead of building their own storage systems. Outsourcing storage has several doubtless advantages, including flexibility and cost efficiency. In addition, cloud based data storage systems provide increased dependability and easy access to the data from anywhere and at any time.

Yet, cloud based data storage systems also have disadvantages. Most importantly, data owners lose the control over their data: the cloud operator can delete the data, or on the contrary, deny the execution of a requested delete operation. Furthermore, there are also serious security issues, stemming from the fact that the cloud operators have access to the data of millions of people and thousands of companies: attackers may break into the cloud system to obtain access to the data stored there, or the cloud operator itself may be tempted to misuse its privileged position. Exploitation of a single vulnerability may lead to the compromise of a large amount of information (e.g. Dropbox no password login [1]).

In today's cloud based data storage systems, access control to the stored data is based on the traditional access control list approach, where a trusted reference monitor enforces the access control policy represented by the access control lists. However, in cloud based data storage systems, the reference monitor is under the control of the cloud operator, and hence, it cannot be fully trusted. A better approach would be to use cryptographic protection mechanisms, and store only encrypted data in the cloud ([2] [3]). This, in turn, requires appropriate key management schemes in order to support abstractions such as groups and shared resources.

In this paper, we follow the second approach. More specifically, we propose a suite of group key management protocols that allows a group of users to agree on a shared group key, which can be used to protect a shared file system stored remotely in the cloud. Our protocols support the refreshment of the group key at each group membership modifications. Compared to other similar solutions, the key novelty of our protocols is that they require asynchronous communication channels, and only constant number of exponentiations on average, worst case logarithmic number of exponentiations even if membership modifications are controlled by arbitrary user. For this efficiency we are using a novel self-balancing key-tree. Moreover, our scheme is fully decentralized, and its design is led by a practical, invitation-oriented approach, similar to Dropbox, where data is shared with new users with just one click. Our group key management protocols are being implemented as part of a full-fledged encrypted storage system called Tresorium, which is described in a companion paper [4].

The paper is organized as follows: in Section II, we present our system and attacker models and introduce some terminology. In Section III, we give a quick overview of the Tree-based Group Diffie-Hellman (TGDH) [5] protocol, which inspired our design. In Section IV, we present our invitation-oriented group key agreement protocol, which is based on TGDH, but achieves a significantly greater flexibility. Finally, in Section V, we report on some related works, and in Section VI we conclude the paper.

II. System model

A. Entities and terminology

Entities of our system model are defined as follows: *users* are humans using some client system (hardware and software together), shortly *client* to connect to the *cloud*. There are an asynchronous multicast and unicast channels between clients. More details on the communication model are discussed in

1530-2016/12 $26.00 © 2012 IEEE
DOI 10.1109/ICPPW.2012.40

Section II-C. The cloud is maintained by *cloud operator*, or shortly operator.

Users can form multiple *groups*, where each group has a shared *group key*. The group key is used to protect shared resources of the group. In this paper, we model the shared resources of a group as an online file-system, which we call the *tresor* of the group. The content of tresor should be only accessible by the *members* of the group.

Let us consider the following example: Alice, Bob and Carol are colleagues, and they form a group. This group has a tresor, which can be accessed only by Alice, Bob and Carol. In this tresor they share company related documents, spreadsheets, presentations, etc. Physically this tresor is stored on a cloud storage run by Amazon, as the cloud operator.

B. Procedures

In our model, group membership changes from time-to-time: user(s) *leave* the group and other user(s) can *join* the group. These operations require that the permissions to the shared tresor must be changed, such that only the members of changed group can have access to it. A join or leave operation is coordinated by a *sponsor*, who can be an arbitrary member of the group.

Any group member can *calculate* the actual shared group key using the group's public information and the member's own private key. Any group member can *refresh* its public key in the group, replacing its old key with a new one.

Continuing our example, one day Carol is fired from the company, therefore she has to be removed from the group. The remove operation (leave) is done by the sponsor, say Alice. Since Carol's seat cannot be left empty, the management decides to employ Daniel. As Daniel has to access all the business files that were accessible by Carol, Daniel must join the group. For this reason an arbitrary member of the group, say Bob, invites Daniel. This means Bob will act as the *sponsor* in that join operation.

u_i	i^{th} user, member of a group
$\mathcal{U}$	set of users
g_i	group of users, $g_i \subset 2^{\mathcal{U}}$
gk_i	group key of g_i
$sp(g_i)$	sponsor of group g_i
$join(u_i, u_j, g_k)$	u_i joins to g_k group, u_j is a sponsor
$leave(u_i, u_j, g_k)$	u_i leaves g_k group, u_j is a sponsor
$calc(u_i, g_k)$	calculation of gk_k using private information of u_i and public information of g_k
$refresh(u_i, g_k)$	user u_i refresh its public key in the group g_k

Table I
NOTATION OF ENTITIES AND PROCEDURES

C. Communication model

We assume asynchronous multicast and unicast communication channels between the clients.

The rationale behind assuming asynchronous channels is that in practice clients may not be on-line at the same time, therefore synchronous connection may not be possible. An example for asynchronous unicast channel is e-mail, whereas an asynchronous multicast channel can be implemented as a shared file stored in the cloud.

D. Attacker model

The attacker model is defined by the attacker's *goal* and attacker's *capabilities*. The goal of the attacker is to gain access to content of a tresor.

According to its capabilities attackers can be categorized in the following types:

1) *Operator:* has access to encrypted tresors and channels between clients. We assume that operator is honest, but curious, which means that it does not mount denial-of-service type attacks (e.g. deleting a tresor, preventing communications,etc.), but it would like to access the content of the tresors.
2) *Previous group member:* a group member, who was removed from the group, and it had access to a series of previous group keys and to the previous states of the tresor.
3) *Outsider:* does not have direct access to the tresor, but has access to communication channels, in particular has access to the channels used for upload or download files to/from the cloud.

III. ORIGINAL TGDH

A. Static definitions of TGDH

TGDH is a distributed – so it does not require any trusted third party –, group key-agreement protocol based on the Decisional Diffie-Hellman problem. This protocol assumes an authenticated channel. The scheme is based on a binary key-tree (see Figure 1. as an example), where the root node represents the established group key.

We will use the notation summarized in Table II., in accordance with [5]. Every member is represented as a leaf of the key-tree – in the example on Figure 1., members are (m_1, m_2, m_3).

n	number of group members
$< l, v >$	the v^{th} tree node on l^{th} level
l_s, l_d	level of shallowest and level of deepest node
m_i	i^{th} group member
$k_{<i,j>}$	private key of node $< i, j >$
$bk_{<i,j>}$	public (blinded) key of node $< i, j >$
p, q	(big) prime integers
g	group generator of $\mathbb{Z}_p$, where $g \in \mathbb{Z}_p$

Table II
NOTATION OF TGDH

As in the original Diffie-Hellman protocol, the public keys are computed as follows:

$$\forall i, j \quad bk_{<i,j>} = g^{k_{<i,j>}}$$

The private keys of the internal nodes are defined as the combination of the private keys of the two child nodes:

$$k_{<l,v>} = g^{k_{<l+1,2v>} k_{<l+1,2v+1>}}$$

$$= bk_{<l+1,2v>}{}^{k_{<l+1,2v+1>}}$$

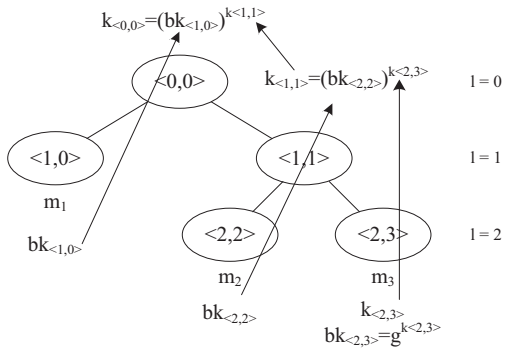


Figure 1. TGDH key-tree and definitions of internal nodes' private keys

$$= bk_{<l+1,2v+1>}{}^{k_{<l+1,2v>}}$$

This means that starting from a leaf, and using that leaf's private key, a group member can always *calculate* the private key of the root node – which is used as the group key (Figure 1). The complexity of calculating the group is $\mathcal{O}(h)$, where h is the height of the tree.

B. Protocols of TGDH

There are 5 protocols in TGDH: *join*, *leave*, *merge*, *partition* and *key-refresh*. After each group change, the public keys in the tree are changed so the group key is also changed. Here, we show briefly the join and the leave protocols.

TGDH Join: Let us assume the currect members are $m_1, m_2, \ldots, m_n$, and the joining member is m_{n+1}. The *join* protocol is run as follows (Figure 2):

1) joining member m_{n+1} broadcasts its public key bk_{n+1} to all the current members.
2) Each member determines the *insertion node*, where the new node – representing the joining member – will be connected. If the tree is balanced, then the insertion node is the root, else the shallowest, rightmost node.
3) The sponsor will be the rightmost leaf of the subtree rooted at insertion node, and each member determines by itself if it the sponsor or not.
4) Sponsor broadcasts the refreshed public keys of the tree.
5) Each member calculates the new group key with a recursive algorithm similar to the one illustrated in Figure 1.

TGDH Leave: The *leave* protocol is defined as follows:

1) The sponsor is determined as the rightmost leaf of the subtree rooted at the sibling node of leaving node.
2) The sponsor broadcasts the refreshed tree to each member.
3) Each member calculates the new group key with a recursive algorithm similar to the one illustrated in Figure 1.

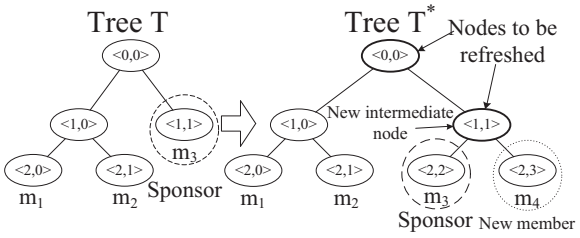


Figure 2. An example for TGDH join [5]

Notice that the cooperation of the leaving node is not needed, hence forced leave (remove) is possible.

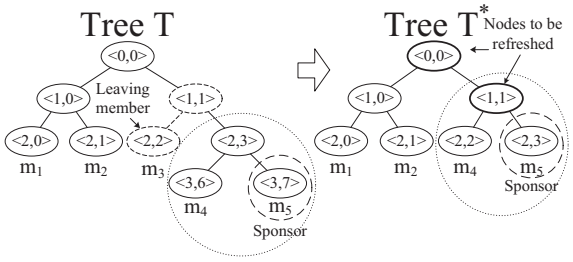


Figure 3. An example for TGDH leave [5]

C. Properties of TGDH

Among distributed group key agreement, TGDH was found efficient [6]. Choosing the actual insertion node influences only the computational performance of the protocol, and not its security [7].

IV. Modified TGDH

A. Problems with TGDH

In the original TGDH article on-line members and synchronous channels were assumed. Joining member has to send a 'request-for-join' message to the sponsor, and the actual sponsor is determined by rigid rules for efficiency. However, in practice it is desirable that arbitrary user can *invite* and act as sponsor for new members. For example, in Dropbox anyone can re-share a shared folder with its friends. If we used TGDH to generate a key for Dropbox, we could not support the original flexibility of Dropbox. The invitee would need to wait for specific user – sponsor – to come online, even if the inviter is online.

In TGDH, the authorization is also a problem: despite the fact that the channel is authenticated, the sponsor cannot decide if a new user is authorized to join the group or not. This is illustrated in the following example: Alice and Bob form a group, and share a group key using TGDH. Bob would like to share the protected resources with Cecil. Cecil sends a 'request-for-join' to Alice, who is the actual sponsor. Alice

can authenticate her, but cannot decide if Cecil is authorized or not, because Alice was not the one who would like to share resources with Cecil. If the inviter (Bob) was the sponsor, this problem would not exist: Bob invited Cecil, so Bob already decided that Cecil is authorized to access shared resources.

Although the original TGDH exactly specifies one sponsor every time, it could be extended such that any group member can be the sponsor, however for that flexibility we pay with $\mathcal{O}(n)$ complexity, instead of the original $\mathcal{O}(\log n)$ worst case [7].

In this article, we propose an *efficient*, invitation-oriented modification of TGDH, with $\mathcal{O}(\log n)$ calculation complexity as a solution for the above problem. Our protocol ensures that a single interaction between the inviter and the new user is enough for conducting a join, without a need for another member as a sponsor.

B. Design requirements

1) Usability: An arbitrary member of the group can act as a sponsor irrespectively of its position in the tree, and a *join* or *leave* operation just need an interaction between the sponsor and the joining or leaving user. After a group change, every member of the group should be able to calculate the actual group key.

2) Computational performance: The scheme should need $\mathcal{O}(\log n)$ exponentiations and other operations.

3) Communication performance: The scheme should need $\mathcal{O}(1)$ *asynchronous* messages, and multiple *join* or *leave* operations can be started simultaneously.

4) Security: The scheme should preserve the cryptographic and security properties of TGDH. These design requirements are defined as follows.

Assume that group key changed n times, and the series of keys are $\{k_1, k_2, \ldots, k_n\}$. Requirements is similar to [5]:

1) *Group key secrecy*: for all i, the discovery of k_i is computationaly infeasible for an adversary
2) *Backward secrecy*: an adversary cannot calculate k_j, even if it knows $\{k_1, k_2, \ldots, k_i\}$ for $i < j$.
3) *Forward secrecy*: an adversary cannot calculate k_j, even if it knows $\{k_i, k_{i+1}, \ldots, k_l\}$ for $j < i < l$.
4) *Key independence*: even if an adversary knows a subset of keys $\mathcal{K}^* \subset \{k_1, k_2, \ldots, k_n\}$, it cannot discover any k_j key, where $k_j \in \{k_1, k_2, \ldots, k_n\} \backslash \mathcal{K}^*$.
5) *Perfect forward secrecy:* compromising any k_i in the future does not effect the secrecy of any $k_j, j \neq i$.

C. Proposed scheme

We propose the usage of *shadow nodes*, as special leaves helping group change operations: the private key of a shadow node is a temporal key generated by a sponsor, and after the join or leave operation the private key is discarded. Shadow nodes are only leaves, and the keys of them are distinguished from a normal member's key, because they do not represent an actual user.

We propose *red-white-black key-trees*, which guarantee that after any group change, coordinated by arbitrary member in the group as a sponsor, the key-tree remains „quasi-balanced". The goal of such trees is to keep the computation complexity of group key $\mathcal{O}(\log n)$.

1) Using shadow nodes: The sponsor must update the public keys of tree after a group change on the way up to the root from the changed node. Because of that sponsor has to know the private key of sibling node of the joining or leaving node, otherwise it cannot calculate the public keys. That is why in TGDH join protocol the sponsor was chosen to become the sibling of the joining user after creating a new intermediate node (see Figure 2.), and in TGDH leave protocol the sponsor was chosen as member of the subtree rooted in the sibling of the leaving node (see Figure 3.).

Using a shadow node, arbitrary member can calculate $k_{\langle 2,3\rangle}$, so arbitrary member can continue the steps of the key refresh (Figure 4.). Continuing the example, the node associated with the joining member m_4 is $\langle 3,6\rangle$, and its sibling $\langle 3,7\rangle$ is a shadow node. $k_{\langle 3,7\rangle}$ is freshly generated by the sponsor, so the sponsor can calculate $k_{\langle 2,3\rangle} = (bk_{\langle 3,6\rangle})^{k_{\langle 3,7\rangle}}$. Using this key, the actual sponsor can update the public keys of $\langle 2,3\rangle, \langle 1,1\rangle$ (for calculation steps, see Figure 1.). This means, that by using shadow nodes, any user of the group can be the sponsor.

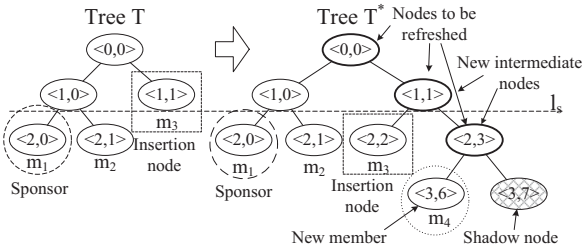


Figure 4. An example for TGDH join using a shadow nodes when joining point is a normal node

D. Invitation oriented join

Inviter u_I sends an invitation message to new user u_N through an authenticated, asynchronous channel containing the basic parameters of the group. Inviter u_I is an arbitrary user of the group, and u_I can send multiple invitations at the same time simultaneously, in any order. Even more, inviters can send invitations independently from each other. Invited new users can reply in arbitrary order.

New user u_N replies with a request message for join to u_I through an authenticated, asynchronous channel, containing its public key, bk_N. As soon as u_I receives the request, it will act as the sponsor of the group. Briefly, u_I performs the following steps to insert the public key of u_N:

1) It determines the optimal insertion point. Intuitively, we could see that replacing the shallowest shadow node helps to keep the optimality (Figure 5).
2) If node associated with is not one of the shallowest nodes, then u_I uses shadow nodes to insert u_N (Figure 4).

3) u_I updates the public keys of the internal nodes from the insertion point on the way up to the root.
4) u_I sends the updated tree to every member.

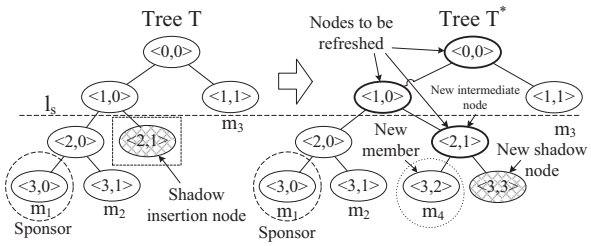


Figure 5. An example for TGDH join using a shadow nodes when joining point is a shadow node

Algoritm 1 describes the join operation in more detail.

E. Forced leave

Let us assume that an authorized user u_S decides to remove u_L from the group, where u_S and u_L can be arbitrary leaves of the tree.

Our first idea could be replacing u_L with a shadow node. The problem with this approach is that after several execution of leave protocol the tree could lose its „quasi-balanced" property, which means that the calculation of the group key would need linear, or even super liner exponentiations, instead of the logarithmic complexity (Figure 6).

For keeping the „quasi-balanced" property of the tree, we propose the usage of red-white-black key-trees which has the nice property that the key-tree remains „quasi-balanced" all the time.

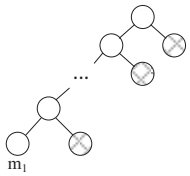


Figure 6. If simple, greedy algorithm is used this situation can easily happen

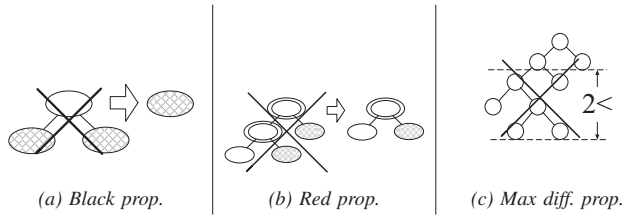
A **red-white-black key-tree** is a colored key-tree, where shadow node is colored black, the parent of the shadow node is red, and remaining nodes are colored white. Red-white-black key-tree has the following properties (Figure 7), which we call $\mathcal{P}$:

- *Black property:* black node does not have any child, and does not have black sibling.
- *Red property:* red node does not have a red child.
- *Max differential property:* the difference between shallowest and deepest nodes is $l_d - l_s \leq 2$.

It can be proven that the previously presented join protocol keep the $\mathcal{P}$ properties.

Algorithm 1 Detailed description of join using shadow nodes: $join(u_L, u_S, g_i)$

function $optimal\_join(T, u_S)$
  look for shallowest nodes in T.
  if sponsor's node on level l_s **then**
    return sponsor's node
  else if there is a shadow node on l_s **then**
    return rightmost shadow node on l_s
  else if sponsor's node is on $l_s + 1$ **then**
    return sponsor's node
  else if there is a shadow node on $l_s + 1$ **then**
    return rightmost shadow node on $l_s + 1$
  else
    return rightmost normal node on l_s
  end if
end function
function $refresh\_nodes(\langle l, v\rangle)$
  $k_{\langle l-1, \lfloor \frac{v}{2} \rfloor\rangle} \leftarrow (bk_{sibling(\langle l,v\rangle)})^{k_{\langle l,v\rangle}}$
  if $\langle l, v\rangle = \langle 0, 0\rangle$ **then**
    return
  else
    call $refresh\_nodes(\langle l-1, \lfloor \frac{v}{2} \rfloor\rangle)$
  end if
end function
Lock T tree
$\langle l, v\rangle = optimal\_join(T, u_S)$
if $\langle l, v\rangle$ = sponsor's node **then**
  if sibling of $\langle l, v\rangle$ is not a shadow node **then**
    continue with normal TGDH join (Fig 2.)
  else
    replace the shadow node with u_N
    continue with normal TGDH join (Fig 2.)
  end if
else
  generate a new random shadow key, k_{sh}
  if $\langle l, v\rangle$ is shadow node **then**
    $\langle l, v\rangle$ is replaced with $\langle l, v\rangle^*$ (Fig 5.)
    $bk_{\langle l+1,2v\rangle} = b_N$, representing u_N
    $k_{\langle l+1,2v+1\rangle} = k_{sh}$, which is a shadow node.
    $k_{\langle l,v\rangle^*} \leftarrow (bk_{\langle l+1,2v\rangle})^{k_{sh}}$.
  else
    $\langle l, v\rangle$ is replaced with $\langle l, v\rangle^*$ (Fig 4.)
    $bk_{T^*\langle l+1,2v\rangle} \leftarrow bk_{T\langle l,v\rangle}$
    $bk_{\langle l+2,2(2v+1)\rangle} \leftarrow bk_{u_N}$
    $k_{\langle l+1,2v+1\rangle} \leftarrow (bk_{u_N})^{k_{sh}}$
    $k_{\langle l,v\rangle^*} \leftarrow (bk_{T^*\langle l+1,2v\rangle})^{k_{\langle l+1,2v+1\rangle}}$
  end if
  $refresh\_nodes(\langle l, v\rangle^*)$
end if
Multicast public keys of T^* to members.
Unlock the tree.
Discard k_{sh}.



(a) Black prop. *(b) Red prop.* *(c) Max diff. prop.*

Figure 7. Red-white-black key-tree properties and recovery of them

Sponsor u_S tries to run the normal TGDH leave protocol, if possible. Otherwise u_S does the steps described in Algorithm 2, and recovers red and black properties. It can be proven, that in this case the *Maximum differential property* will also be recovered. Briefly, u_S performs the following steps to remove u_L from the group (detailed description Algorithm 2.):

1) If u_S is member of the subtree rooted at the sibling of u_L, then u_S continues with the normal TGDH leave protocol(Fig 3.).
2) If u_L is on l_s then it is replaced with the deepest normal node if ($l_d > l_s$) (Figure 8), or with a shadow node if $l_s = l_d$.

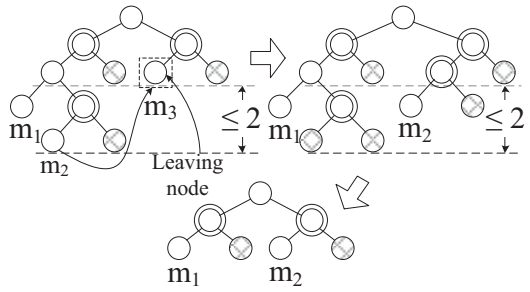


Figure 8. Forced leave if u_L is on l_s, and the result of red and black property recovery

3) If u_L is not on l_s, u_L is replaced with a shadow node (Fig 9.).

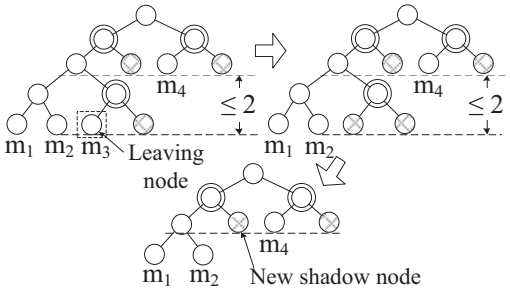


Figure 9. Forced leave if u_L is not on l_s, and the result of red and black property recovery

4) u_S recovers $\mathcal{P}$ properties, following the steps demonstrated in Figure 7./(a),(b).
5) u_S refreshes the keys of the internal nodes of the tree from the changed nodes on way up to the root.
6) u_S sends the updated tree to every member.

F. Refresh

Arbitrary user can call $refresh(\cdot,\cdot)$ in order to replace its old key with a new one. The simple refresh consists of the following steps:

1) u_R generates a new private key, k_R.
2) u_R replaces the old public key of its node with $bk_R = g^{k_R}$
3) u_R calls the $refresh\_nodes(u_R)$ function described in Algorithm 1,
4) u_R sends new T^* to every member.

G. Calculate

Arbitrary member – by using its private key – is able to calculate the actual group key by following the recursive algorithm described in Algorithm 3. This is similar to the calculation steps described in TGDH.

H. Analysis

1) Usability: As we showed, the *join* protocol needs only interaction between the sponsor and the new user, and the sponsor can be arbitrary user in the tree.

The *leave* protocol does not need any interaction between members, because the sponsor can prepare the modified tree alone.

Since the tree is the special case of TGDH trees, any member is able to calculate the actual group key using the original algorithm illustrated in Figure 1.

2) Computational performance: It can be proven that the sponsor u_S will recover the red-white-black key-tree properties in the leave protocol after a constant number (< 4) of steps. The recovery steps do not require any exponentations, which is the most expensive operation in terms of computations. It is easier to see, that after the $\mathcal{P}$ properties are recovered, the sponsor can always refresh the public keys of the internal nodes, using its own shadow nodes, with its own temporal private keys. Note that only the nodes on the way up to the root need to be refreshed.

The red-white-black tree properties guarantee that the key-tree always remains quasi-balanced. The maximum differential property guarantees that actual tree has at most two more levels compared to the optimal tree. Shadow nodes give one more extra level, therefore the calculation of the group-key needs maximum $\log N + 3 = \mathcal{O}(\log N)$ exponentiations. During the join protocol, the sponsor has to refresh the tree once, with maximum $\log N + 3$ exponentiations, and during the leave protocol, the sponsor has to refresh the tree maximum two times with maximum $2(\log N + 3)$ steps.

Furthermore, the average complexity of the protocols is much better than the worst-case complexity. In order to see this, let us consider Figure 4 and let us denote by S the random

Algorithm 2 Detailed description of forced leave using shadow nodes: $leave(u_L, u_S, g_i)$

function $refresh\_nodes(\langle l, v\rangle)$
  $k_{\langle l-1,\lfloor \frac{v}{2} \rfloor\rangle} \leftarrow (bk_{sibling(\langle l,v\rangle)})^{k_{\langle l,v\rangle}}$
  if $\langle l, v\rangle = \langle 0, 0\rangle$ **then**
    return
  else
    call $refresh\_nodes(\langle l-1, \lfloor \frac{v}{2} \rfloor\rangle)$
  end if
end function
function $recover\_\mathcal{P}(\langle l, v\rangle)$
  if $is\_shadow(\langle l, v\rangle)$ **then**
    if $is\_shadow(sibling(\langle l, v\rangle))$ **then**
      replace $\langle l-1, \lfloor \frac{v}{2} \rfloor\rangle$ with $\langle l, v\rangle$ (Fig 7/(a))
      call $recover\_\mathcal{P}(\langle l-1, \lfloor \frac{v}{2} \rfloor\rangle)$
    else if $is\_shadow(sibling(\langle l-1, \lfloor \frac{v}{2} \rfloor\rangle))$ **then**
      replace $sibling(\langle l-1, \lfloor \frac{v}{2} \rfloor\rangle)$ with $\langle l, v\rangle$
      replace $\langle l-1, \lfloor \frac{v}{2} \rfloor\rangle$ with $sibling(\langle l, v\rangle)$ (Fig 7/(b))
      call $recover\_\mathcal{P}(sibling(\langle l-1, \lfloor \frac{v}{2} \rfloor\rangle))$
    end if
  end if
end function
Lock T tree
if u_S member of subtree rooted in $sibling(u_L)$ **then**
  continue with TGDH leave (Fig 3.).
end if
if u_L on l_S **then**
  if $l_d > l_s$ && $is\_shadow(sibling(u_L))$ **then**
    replace u_L with a new node $\langle l, v\rangle$
    move rightmost normal node on l_d to $\langle l+1, 2v\rangle$ (Fig 8.).
    complement removed node with new shadow node s_1.
    $\langle l+1, 2v+1\rangle$ is a new shadow node s_2.
    call $recover\_\mathcal{P}(s_1)$
    call $recover\_\mathcal{P}(s_2)$
    call $refresh\_nodes(s_1)$
    call $refresh\_nodes(s_2)$
  else
    replace u_L with a new shadow node s_1 .
    call $refresh\_nodes(s_1)$
  end if
else if u_L is not on l_s **then**
  replace u_L with a new shadow node s_1 (Fig 9.).
  call $recover\_\mathcal{P}(s_1)$
  call $refresh\_nodes(s_1)$
end if
Send T^* to every remaining member, and release lock.

Algorithm 3 Detailed description of calculation of group key, illustrated in Fig 1

function $calc(u_C, T)$
  $k_{parent(u_C)} = (bk_{sibling(u_C)})^{k_{u_C}}$
  if $parent(u_C) = \langle 0, 0\rangle$ **then**
    return $k_{\langle l-1, \lfloor \frac{v}{2} \rfloor\rangle}$
  else
    return call $calc(parent(u_C), T)$
  end if
end function

variable representing the required number of exponentiations. A group change might happen in the subtree of $\langle 1, 0\rangle$ with probability $\frac{1}{2}$, because half of the members are in that subtree and the tree is quasi-balanced. This means that a member of subtree rooted in $\langle 1, 1\rangle$ might need to do only one exponentiation for calculating the group key (note $gk = k_{\langle 0,0\rangle}$). This is symmetrical, so we can state that $\Pr\{S = 1\} = \frac{1}{2}$ with a relatively small error. This is also true for the subtrees at deeper leaves, and hence we get that

$$\Pr\{S = 2\} \cong \frac{1}{4}, \Pr\{S = 3\} \cong \frac{1}{8}, \dots \Pr\{S = n\} \cong \frac{1}{2^n}$$

$$\mathbb{E}(S) = 1{*}\Pr\{S = 1\}{+}2{*}\Pr\{S = 2\}{+}\cdots{+}n{*}\Pr\{S = n\} \cong$$

$$\cong \sum_{i=1}^{n} i * \frac{1}{2^i} \leq 2$$

3) Communication performance: The join protocol needs 2 multicast, asynchronous messages (lock notification, update and lock release). These steps do not give extra latency, because the sponsor does not need to wait for reply. The join protocol also needs 2 unicast messages, the sponsor's *invite* message, and as a reply for that, a *request* message from the new user. Note that the tree must be locked only if the request messaged is received, and can be unlocked as soon as the sponsor refreshed the tree.

The *leave* protocol does not need any interaction, just 2 multicast asynchronous messages (the lock message, and the update and lock release in a single message).

Apart from the short critical sections – while the tree is locked, which is in practice is relatively a short time –, multiple *join* and *leave* protocols can run simultaneously.

4) Security: The invitation oriented *join* is a special case of the normal TGDH merge operation: we could define invitation oriented join as merging a two-level tree, constructed from the public key of u_N and the private key of a shadow node (example: Fig 4., node $\langle 2, 3\rangle$ is the root of the merged subtree). This reduction to the original TGDH merge guarantees the stated security design requirements.

The forced leave can be reduced to several TGDH join and leave operations:

1) The replacement of the leaving node illustrated in Figure 8 can be reduced to the invitation oriented join,

2) The recovery steps of the red and black properties (illustrated in Figure 7) are the special cases of the TGDH leave protocol.

V. Related works

In order to exchange a key between two parties several protocols were proposed, like Diffie-Hellman [8] which is a key-agreement, and like Kerberos [9], or Centralized Key Distribution (CKD) [10], which are key-transport protocols.

Among group-key exchange protocols the problem of dynamism arises, however there are several protocols which do not solve this problem at all (e.g. [11], [12], [13]). Group-key exchange protocols for dynamic groups, like Groud Diffie Hellman (GDH) [14], Burmester-Desmedt (BD) [15], Skinny Tree (STR) [16] usually need $\mathcal{O}(n)$ messages, and $\mathcal{O}(n)$ exponentiations. TGDH [5] with its $\mathcal{O}(log(n))$ computational, and $\mathcal{O}(1)$ communicational complexity was found quite efficient among group-key exchange protocols [6], only if the sponsor cannot be arbitrary user. Several group-key agreement protocols were proposed based on key-trees, like [17] or [18], to improve computational efficiency. Cryptree was proposed for file-system specific problem of sharing a group key [19].

The group-key exchange protocols mentioned above assume authenticated channels. Several authenticated protocols were proposed, like S-TGDH [20], which is an extension of TGDH, nPAKE+ [21], an interesting password-based Diffie-Hellman exchange, or [22] and [23].

The properties of our proposed red-black-white key-tree was motivated by the red-black search tree [24] [25], because red-black tree has a self-balancing property.

VI. Conclusion

In this paper, we proposed a suite of group key management protocols that allows a group of users to agree on a shared group key, which can be used to protect a shared file system stored remotely in the cloud. Our protocols support the refreshment of the group key at each group membership modifications, and they work in an asynchornous communication model, which makes them suitable for practical cloud based storage systems, where users may not be on-line simultaneously all the time. In addition, compared to other similar solutions, where the required number of exponentiations and messages is a function of the group size, our protocols are more efficient, as they require only constant number of exponentiations on average, worst case logarithmic number of exponentiations and only a constant number of messages even if membership modifications are controlled by arbitrary user.

References

[1] C. Interactive, "Dropbox confirms security glitch–no password required," http://news.cnet.com/8301-31921_3-20072755-281/dropbox-confirms-security-glitch-no-password-required/, 2011.

[2] E.-J. Goh, H. Shacham, N. Modadugu, and D. Boneh, "SiRiUS : Securing Remote Untrusted Storage," in *Proceedings of NDSS*, no. 0121481, ISOC. Geneva: The Internet Society, 2003, pp. 131–145.

[3] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu, "Plutus: Scalable secure file sharing on untrusted storage," in *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, vol. 42. USENIX Association, 2003, pp. 29–42.

[4] I. Lam, S. Szebeni, and L. Buttyan, "Tresorium: cryptographic file system for dynamic groups over untrusted cloud storage," in *Submitted to 4th International Workshop on Security in Cloud Computing*, 2012.

[5] Y. Kim, A. Perrig, and G. Tsudik, "Tree-based group key agreement," *ACM Transactions on Information and System Security*, pp. 60–96, 2004.

[6] Y. Amir, Y. Kim, C. Nita-Rotaru, and G. Tsudik, "On the performance of group key agreement protocols," *ACM Transactions on Information and System Security (TISSEC)*, pp. 457–488, 2004.

[7] Y. Kim, F. Maino, M. Narasimha, and G. Tsudik, "Secure group services for storage area networks," in *First International IEEE Security in Storage Workshop, 2002. Proceedings.*, 2002, pp. 80–93.

[8] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[9] B. C. Neuman and T. Ts'o, "Kerberos: An Authentication Service for Computer Networks," *IEEE Communications Magazine*, vol. 32, no. 9, pp. 33–38, 1994.

[10] Y. Amir, Y. Kim, C. Nita-Rotaru, J. Schultz, J. Stanton, and G. Tsudik, "Secure group communication using robust contributory key agreement," *IEEE Transactions on Parallel and Distributed Systems*, pp. 468–480, 2004.

[11] M. Steiner, G. Tsudik, and M. Waidner, "Diffie-hellman key distribution extended to group communication," in *Proceedings of the 3rd ACM conference on Computer and communications security*, 1996.

[12] G. Ateniese, S. M., and T. G., "New multiparty authentication services and key agreement protocols," *IEEE Journal on Selected Areas in Communications*, p. 628, 2000.

[13] I. Ingemarsson, D. T. Tang, and C. K. Wong, "A conference key distribution system," *IEEE Transactions on Information Theory*, p. 714, 1982.

[14] M. Steiner, G. Tsudik, and M. Widner, "Key agreement in dynamic peer groups," *IEEE Transactions on Parallel and Distributed Systems*, pp. 769–780, 2000.

[15] M. Burmester and Y. Desmedt, "A secure and efficient conference key distribution system," in *Advances in Cryptology - EUROCRYPT '94*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1995, pp. 275–286.

[16] D. G. Steer, L. Strawczynski, W. Diffie, and M. Wiener, "A secure audio teleconference system," in *Advances in Cryptology - CRYPTO' 88*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1990, pp. 520–528.

[17] S. Lee, Y. Kim, K. Kim, and D.-H. Ryu, "An Efficient Tree-Based Group Key Agreement Using Bilinear Map," *Lecture Notes in Computer Science including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics*, vol. 2846, pp. 357–371, 2003.

[18] S. Tripathi and G. P. Biswas, "Design of efficient ternary-tree based group key agreement protocol for dynamic groups," *2009 First International Communication Systems and Networks and Workshops*, pp. 1–6, 2009.

[19] D. Grolimund, L. Meisser, S. Schmid, and R. Wattenhofer, "Cryptree: A folder tree structure for cryptographic file systems," in *Proceedings of the 25th IEEE Symposium on Reliable Distributed Systems*, 2006, pp. 189–198.

[20] F. Cuppens, N. Cuppens-Boulahia, and J. Thomas, "S-TGDH, secure enhanced group management protocol in ad hoc networks," *CRiSIS'2007 : International Conference on Risks and Security of Internet and Systems*, 2007.

[21] Z. Wan, R. H. Deng, F. Bao, B. Preneel, and M. Gu, "NPAKE+: A tree-based group password-authenticated key exchange protocol using different passwords," *Journal of Computer Science and Technology*, vol. 24, no. 1, pp. 138–151, 2009.

[22] E. Bresson, O. Chevassut, and D. Pointcheval, "Provably authenticated group diffie-hellman key exchange - the dynamic case," in *Advances in Cryptology - ASIACRYPT 2001*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2001, pp. 290–309.

[23] E. Bresson, O. Chevassut, D. Pointcheval, and J. J. Quisquater, "Provably authenticated group diffie-hellman key exchange," in *Proceedings of the 8th ACM conference on Computer and Communications Security*, 2001, pp. 255–264.

[24] R. Bayer, "Symmetric binary B-trees: Data structure and maintenance algorithms," *Acta Informatica*, vol. 1, no. 4, pp. 290–306, 1972.

[25] L. J. Guibas and R. Sedgewick, "A dichromatic framework for balanced trees," in *Foundations of Computer Science 1978 19th Annual Symposium on*, ser. Lecture Notes Comput. Sci. IEEE, 1978, pp. 8–21.