(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 103281175 B
(45)授权公告日 2015.11.04

(21)申请号 201310176324.6

(22)申请日 2013.05.14

(73)专利权人 电子科技大学
地址 611731 四川省成都市高新区(西区)西源大道2006号

(72)发明人 徐杰 尹华云 孙健 隆克平

(74)专利代理机构 成都行之专利代理事务所(普通合伙) 51220
代理人 温利平

(51)Int.Cl.
H04L 9/08(2006.01)

(56)对比文件

CN 101257382 A,2008.09.03,全文.

CN 101488849 A,2009.07.22,全文.

Deuk-Whee Kwak等.An efficient LKH Tree balancing algorithm for group key management.《IEEE》.2006,第10卷(第3期),第1089-7798页.

范书平等.一种改进LKH的组播密钥管理方案.《计算机工程与应用》.2010,第46卷(第35期),第104-108页.

李辉.基于密钥树的批量密钥更新方法.《计算机工程》.2009,第35卷(第23期),第133-135页.

刘利芬等.一种改进的R-LKH算法.《计算机工程》.2008,第34卷(第18期),第176-178页.

赵光凯.基于LKH的组播密钥管理技术的研究.《中国优秀硕士学位论文数据库信息科技辑》.2009,第I138-62页.

晏轲.基于LKH的移动组播密钥管理.《中国优秀硕士学位论文数据库信息科技辑》.2006,第I136 -396页.

审查员 郑红萍

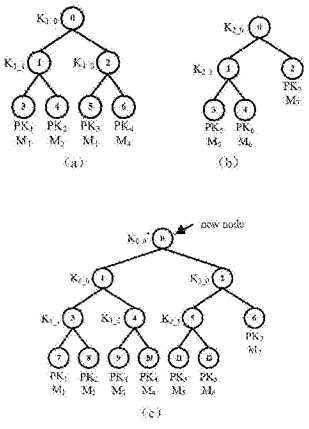
权利要求书3页 说明书6页 附图7页

(54)发明名称

一种LKH密钥管理树动态平衡方法

(57)摘要

本发明公开了一种LKH密钥管理树动态平衡方法通过对加入和离开密钥管理组的用户数目N_J和N_D进行判断,采用不同的方法对原始密钥管理树进行动态平衡,得到新的密钥管理树即平衡密钥管理树,从而保证了LKH在组密钥管理上的高效性。



CN 103281175 B

1.一种逻辑密钥分层LKH密钥管理树动态平衡方法，其特征在于，包括以下步骤：

(1)、定义变量N_J和N_D分别表示加入和离开密钥管理组的用户数目；

(2)、对加入和离开密钥管理组的用户数目N_J和N_D进行判断；

2.1)、当加入用户数量N_J> 离开用户数量N_D，则执行：

a1、选择数量为N_D的加入用户替代原始密钥管理树中的离开用户；

a2、将剩余的加入用户和原始密钥管理树合并为一棵新的密钥管理树：将加入用户看成单个节点的密钥管理树，将单节点密钥管理树和原始密钥管理树组成一个待合并密钥管理树集合，从待合并密钥管理树集合选择两棵高度最小的密钥管理树，将其合并为一棵密钥管理树并置于待合并密钥管理树集合，同时从待合并密钥管理树集合中删除这两个高度最小的密钥管理树；在待合并密钥管理树集合中再选择两棵高度最小的密钥管理树进行合并，这样重复，直至只剩下一棵密钥管理树为止，该密钥管理树为新的密钥管理树；

2.2)、当加入用户数量N_J＝用户数量离开用户数量N_D，只需用加入用户替代原始密钥管理树中离开用户，得到新的密钥管理树即可；

2.3)、当加入用户数量N_J< 离开用户数量N_D，执行：

b1、用所有加入用户替代原始密钥管理树中的数目为N_J的离开用户；

b2、在原始密钥管理树中删除未被替代的离开用户，并进行节点删除：

若原始密钥管理树中节点缺少右节点子树或左节点子树，此节点将会被它剩下的节点子树替代；

b3、从下到上，从右至左依次检查原始密钥管理树中的每个节点是否为平衡节点，若节点不平衡，则合并不平衡节点的左右子树，并用新合并形成的子树替代不平衡节点，得到新的密钥管理树；

(3)、密钥管理中心根据新的密钥管理树即平衡密钥管理树中节点变化情况产生密钥更新消息和位置更新消息，对组密钥进行管理。

2.根据权利要求1所述的逻辑密钥分层LKH密钥管理树动态平衡方法，其特征在于，步骤2.1)、步骤2.3)中所述的合并为：

待合并的两棵密钥管理树为Tree1和Tree2，并且密钥管理树Tree1和Tree2 的最大高度为$H_{Max_Tree1} \geq H_{Max_Tree2}$；

c1、当密钥管理树Tree1的最大高度H_{Max_Tree1}-密钥管理树Tree2的最小高度$H_{Min_Tree2} \leq 1$时，采用合并方法一进行合并，即：创建一个新的节点，并将密钥管理树Tree1和Tree2作为新建节点的左右子树，新建节点即为合并后的平衡密钥管理树的根节点；

c2、当密钥管理树Tree1的最大高度H_{Max_Tree1}-密钥管理树Tree2的最小高度H_{Min_Tree2}＞1，且密钥管理树Tree1的最大高度最小高度相等即$H_{Max_Tree1} = H_{Min_Tree1}$时，采用合并方法二进行合并，即：

第一步、计算出密钥管理树Tree1和Tree2最大高度H_{Max_Tree1}、H_{Max_Tree2}的差值h即$h = H_{Max_Tree1} - H_{Max_Tree2}$；

第二步、在密钥管理树Tree1第h层上，从左至右寻找一个待更新节点，若存在这样的节点，标记它；若不存在这样的节点，标记密钥管理树Tree1第h层上最后一个节点；

第三步、创建一个新的节点替代第二步标记的节点，并将密钥管理树Tree2和以标记节点为根节点的子树作为新建节点的左右子树；

c3、当密钥管理树 Tree1 的最大高度 H_{Max_Tree1}- 密钥管理树 Tree2 的最小高度 H_{Min_Tree2} > 1，且密钥管理树 Tree1 的最大高度最小高度不相等即 $H_{Max\_Tree1} \neq H_{Min\_Tree1}$ 时，采用合并方法三进行合并：

第一步、定义参数：

Set_M 表示一个容器，用来储存密钥管理树 Tree1 的最小高度子树；

Set_T 表示一个容器，用来储存待合并的密钥管理树，最初容器中只包含密钥管理树 Tree2；

Num_M 表示容器 Set_M 中最小高度子树的数目；

H_i 表示容器 Set_M 中第 i 棵最小高度子树的最大高度；

$MinTree\_H_i$ 表示高度为 H_i 的最小高度子树；

H_MergeTree 表示容器 Set_T 中高度最大的树；

第二步：找出密钥管理树 Tree1 所有的最小高度子树，并按最大高度由大到小将这些最小高度子树记录在容器 Set_M 中，即有 $H_1 \geq \cdots \geq H_{Num\_M}$；

第三步：在容器 Set_T 中找出最大高度为最大的密钥管理树 H_MergeTree；

第四步：比较密钥管理树 H_MergeTree 和容器 Set_M 中最小高度子树的最大高度，存在以下三种情形：

情形 1：当密钥管理树 H_MergeTree 的最大高度 $H_{Max\_H\_MergeTree}$ 等于最小高度子树 $MinTree\_H_i$ 的最大高度 H_i ($1 \leqslant i \leqslant Num\_M$) 即 $H_{Max\_H\_MergeTree} = H_i$，则采用合并方法一的方法对密钥管理树 H_MergeTree、最小高度子树 $MinTree\_H_i$ 进行合并；其次用合并得到密钥管理树替代密钥管理树 Tree1 中的 $MinTree\_H_i$，最后将 H_MergeTree 从容器 Set_T 删除；

情形 2：当密钥管理树 H_MergeTree 的最大高度 $H_{Max\_H\_MergeTree}$ 小于最小高度子树 $MinTree\_H_i$ 的最大高度 H_i 且大于最小高度子树 $MinTree\_H_{i+1}$ 的最大高度 H_{i+1} 即 $H_{Max\_H\_MergeTree} < H_i$ 且 $H_{Max\_H\_MergeTree} > H_{i+1}$ ($i \neq Num\_M$) 时，如果 $H_i - H_{Min\_H\_MergeTree} > 1$，用合并方法二的方法合并密钥管理树 H_MergeTree 和最小高度子树 $MinTree\_H_i$；否则利用合并方法一的方法合并 H_MergeTree 和 $MinTree\_H_i$，并用新合并的密钥管理树替代密钥管理树 Tree1 中的 $MinTree\_H_i$，最后将 H_MergeTree 从容器 Set_T 删除；

情形 3：当密钥管理树 H_MergeTree 的最大高度 $H_{Max\_H\_MergeTree}$ 小于最小高度子树 $MinTree\_H_1$ 的最大高度 H_1 时，密钥管理树 H_MergeTree 不可能一次性合并至密钥管理树 Tree1 中，采取将密钥管理树 H_MergeTree 分解的措施，分步将其合并至密钥管理树 Tree1：

c31、在密钥管理树 H_MergeTree 中搜索一棵高度为 H_1 的子树，记为子树 $H\_MergeTree\_H_1$；

c32、标记子树 $H\_MergeTree\_H_1$ 的父节点到密钥管理树 H_MergeTree 根节点路径上的所有节点，移去标记节点后，将除子树 $H\_MergeTree\_H_1$ 外的其他剩余子树添加记录到容器 Set_T；

c33、利用合并方法一的方法合并子树 $H\_MergeTree\_H_1$ 和最小高度子树 $MinTree\_H_1$，并用合并后的子树替代密钥管理树 Tree1 中的最小高度子树 $MinTree\_H_1$；

c34、从容器 Set_T 中删除密钥管理树 H_MergeTree；

第五步：检查容器 Set_T 是否为空，若为空则合并完成，否则清空容器 Set_M，返回第一

步；

其中，所述合并步骤中，所述的密钥管理树的最大高度其在数值上等于密钥管理树根节点的最大高度即根节点到其孩子节点的最大长度路径，密钥管理树的最小高度其在数值上等于密钥管理树根节点的最小高度即根节点到其孩子节点的最小路径长度；

所述的平衡节点为如果节点的最大高度和最小高度之差不超过 1，则称此节点为平衡节点；

所述的平衡密钥管理树为若密钥管理树中所有节点都为平衡节点，则称此密钥管理树为平衡密钥管理树；

所述的树的层为设定密钥管理树的根节点位于第 1 层，层数从根节点层次开始依次加 1；

所述的最小高度子树为在一棵非完全的平衡密钥管理树中，若节点的最小高度和最大高度相等但其父节点的最小高度和最大高度不相等，定义由此节点及其所有孩子节点组成的子树为最小高度子树。

一种 LKH 密钥管理树动态平衡方法

技术领域

[0001] 本发明属于网络信息安全技术领域，更为具体地讲，涉及一种 LKH 密钥管理树动态平衡方法。

背景技术

[0002] 快速发展的多播网络带来了多播业务的广泛应用，比如 IPTV、车载通信、视频会议等。如何保证多播组内通信的安全是开展多播业务的基础，最常用和有效的方法是通过给组内消息加密，然后在组内合法成员之间共享一个加密密钥的方式来保证多播组内通信的安全性，这样可以有效阻止非法用户窃取组内通信内容。考虑到组内随时会有用户离开或者加入，为保证组内通信的前向和后向安全，组密钥管理中心需要及时更新加密密钥。如何高效地管理和更新组内加密密钥是一个极其重要的问题。目前，已提出了多种组密钥管理方法，譬如说 OFT、ELK、SDR、SHKD 和 LKH，对于大型动态多播网络 LKH（逻辑密钥分层）是高效的密钥管理方法之一。

[0003] LKH 采用一棵如图 1 所示的密钥管理树对组密钥进行管理。密钥管理树中存在三种类型的密钥，位于树根节点的为加密密钥 K_0，它用来直接加密组内通信内容；位于叶子节点为用户私钥 PK_1～PK_6，每个用户加入到组时，组密钥管理中心都会给他分配一个用户私钥 PK_1～PK_6；树中其他节点为分发密钥 K_1～K_4，它主要是用来分发加密密钥。组内每个成员保存从它对应的叶子节点到树的根节点这条路径上所有密钥，组密钥管理中心保存密钥管理树中所有的密钥。当组内成员有变更时，密钥管理中心就会更新相应的密钥。如图 1 所示，假若成员 M_1 离开通信组，为保证成员 M_1 不能再解密组内通信内容，密钥管理中心需要更新密钥 K_0、K_1 和 K_3，产生如下密钥更新消息 $\{K_3'\}PK_2$，$\{K'_1\}K'_3$，$\{K'_1\}K_4$，$\{K'_0\}K'_1$ 和 $\{K'_0\}K_2$，其中，K_i' 代表 K_i 的新密钥，$\{K_i'\}K_j$ 表示用 K_j 加密密钥 K_i'。对于成员 M_2 来说，当它接收到这些密钥更新消息后，它首先利用它的私钥 PK_2 解密消息 $\{K_3'\}PK_2$ 得到新的密钥 K_3'，然后利用密钥 K_3' 解密消息 $\{K'_1\}K'_3$ 得到新密钥 K_1'，最后利用谜语 K_1' 解密消息 $\{K'_0\}K'_1$ 得到新的加密密钥 K'_0。这样，组内剩余成员能成功得到了新的加密密钥，但成员 M_1 不可能再获得新的加密密钥，这样保证了通信的安全。假若成员 M_1 添加到组，同样为保证成员 M_1 不能利用密钥解密组内以前的通信内容，所以当成员 M_1 添加到组时，组密钥管理中心需要更新加密密钥。如图 1 所示，密钥 K_0、K_1 和 K_3 需要更新，密钥更新消息为 $\{K_3'\}PK_1$，$\{K_3'\}PK_2$，$\{K_1'\}K_3'$，$\{K'_1\}K_4$，$\{K'_0\}K'_1$ 和 $\{K'_0\}K_2$，这样成员 M_1 根据消息 $\{K_3'\}PK_1$，$\{K_1'\}K_3'$ 和 $\{K'_0\}K'_1$ 能得到新的加密密钥，同样，组内其他成员根据密钥更新消息也可以得到新的加密密钥。LKH 密钥管理开销和组内成员数量的对数相关，因而，它是一种低开销的组密钥管理方案。

[0004] 然而，LKH 密钥管理高效的前提是有一棵平衡的密钥管理树，若组内用户关系变化导致密钥管理树不平衡，对 LKH 密钥管理高效性将是一个严重的挑战。

发明内容

[0005] 本发明的目的在于克服现有技术的不足，提供一种 LKH 密钥管理树动态平衡方

法，使LKH密钥管理树在任何情况下都能保持平衡，从而保证了LKH在组密钥管理上的高效性。

[0006] 为实现上述发明目的，本发明LKH密钥管理树动态平衡方法，其特征在于包括以下步骤：

[0007] (1)、定义变量N_J和N_D分别表示加入和离开密钥管理组的用户数目；

[0008] (2)、对加入和离开密钥管理组的用户数目N_J和N_D进行判断；

[0009] 2.1)、当加入用户数量N_J> 离开用户数量N_D，则执行：

[0010] a1、选择数量为N_D的加入用户替代原始密钥管理树中的离开用户；

[0011] a2、将剩余的加入用户和原始密钥管理树合并为一棵新的密钥管理树：将加入用户看成单个节点的密钥管理树，将单节点密钥管理树和原始密钥管理树组成一个待合并密钥管理树集合，从待合并密钥管理树集合选择两棵高度最小的密钥管理树，将其合并为一棵密钥管理树并置于待合并密钥管理树集合，同时从待合并密钥管理树集合中删除这两个高度最小的密钥管理树；在待合并密钥管理树集合中再选择两棵高度最小的密钥管理树进行合并，这样重复，直至只剩下一棵密钥管理树为止，该密钥管理树为新的密钥管理树；

[0012] 2.2)、当加入用户数量N_J= 用户数量离开用户数量N_D，只需用加入用户替代原始密钥管理树中离开用户，得到新的密钥管理树即可；

[0013] 2.3)、当加入用户数量N_J< 离开用户数量N_D，执行：

[0014] b1、用所有加入用户替代原始密钥管理树中的数目为N_J的离开用户；

[0015] b2、在原始密钥管理树中删除未被替代的离开用户，并进行节点删除：

[0016] 若原始密钥管理树中节点缺少右节点子树或左节点子树，此节点将会被它剩下的节点子树替代；

[0017] b3、从下到上，从右至左依次检查原始密钥管理树中的每个节点是否为平衡节点，若节点不平衡，则合并不平衡节点的左右子树，并用新合并形成的子树替代不平衡节点，得到新的密钥管理树；

[0018] (3)、密钥管理中心根据新的密钥管理树即平衡密钥管理树中节点变化情况产生密钥更新消息和位置更新消息，对组密钥进行管理；

[0019] 其中，步骤2.1)、步骤2.3)中所述的合并为：

[0020] 待合并的两棵密钥管理树为Tree1和Tree2，并且密钥管理树Tree1和Tree2的最大高度为$H_{Max_Tree1} \geq H_{Max_Tree2}$；

[0021] c1、当密钥管理树Tree1的最大高度H_{Max_Tree1}- 密钥管理树Tree2的最小高度$H_{Min_Tree2} \leq 1$时，采用合并方法一进行合并即：创建一个新的节点，并将密钥管理树Tree1和Tree2作为新建节点的左右子树，新建节点即为合并后的平衡密钥管理树的根节点；

[0022] c2、当密钥管理树Tree1的最大高度H_{Max_Tree1}- 密钥管理树Tree2的最小高度$H_{Min_Tree2}>1$，且密钥管理树Tree1的最大高度最小高度相等即$H_{Max_Tree1}=H_{Min_Tree1}$时，采用合并方法二进行合并，即：

[0023] 第一步、计算出密钥管理树Tree1和Tree2最大高度H_{Max_Tree1}、H_{Max_Tree2}的差值h即$h=H_{Max_Tree1}-H_{Max_Tree2}$；

[0024] 第二步、在密钥管理树Tree1第h层上，从左至右寻找一个待更新节点，若存在这样的节点，标记它；若不存在这样的节点，标记密钥管理树Tree1第h层上最后一个节点；

[0025] 第三步、创建一个新的节点替代第二步标记的节点，并将密钥管理树 Tree2 和以标记节点为根节点的子树作为新建节点的左右子树；

[0026] c3、当密钥管理树 Tree1 的最大高度 H_{Max_Tree1}- 密钥管理树 Tree2 的最小高度 H_{Min_Tree2}>1，且密钥管理树 Tree1 的最大高度最小高度不相等即 $H_{Max\_Tree1} \neq H_{Min\_Tree1}$ 时，采样合并方法三进行合并：

[0027] 第一步、定义参数：

[0028] Set_M 表示一个容器，用来储存密钥管理树 Tree1 的最小高度子树；

[0029] Set_T 表示一个容器，用来储存待合并的密钥管理树，最初容器中只包含密钥管理树 Tree2；

[0030] Num_M 表示容器 Set_M 中最小高度子树的数目；

[0031] H_i 表示容器 Set_M 中第 i 棵最小高度子树的最大高度；

[0032] MinTree_H_i 表示高度为 H_i 的最小高度子树；

[0033] H_MergeTree 表示容器 Set_T 中高度最大的树；

[0034] 第二步：找出密钥管理树 Tree1 所有的最小高度子树，并按最大高度由大到小将这些最小高度子树记录在容器 Set_M 中，即有 $H_1 \geq \cdots \geq H_{Num\_M}$；

[0035] 第三步：在容器 Set_T 中找出最大高度为最大的密钥管理树 H_MergeTree；

[0036] 第四步：比较密钥管理树 H_MergeTree 和容器 Set_M 中最小高度子树的最大高度，存在以后三种情形：

[0037] 情形 1：当密钥管理树 H_MergeTree 的最大高度 $H_{Max\_H\_MergeTree}$ 等于最小高度子树 MinTree_H_i 的最大高度 H_i（$1 \leq i \leq Num\_M$）即 $H_{Max\_H\_MergeTree}=H_i$，则采用合并方法一的方法对密钥管理树 H_MergeTree、最小高度子树 MinTree_H_i 进行合并；其次用合并得到密钥管理树替代密钥管理树 Tree1 中的 MinTree_H_i，最后将 H_MergeTree 从容器 Set_T 删除；

[0038] 情形 2：当密钥管理树 H_MergeTree 的最大高度 $H_{Max\_H\_MergeTree}$ 小于最小高度子树 MinTree_H_i 的最大高度 H_i 且大于最小高度子树 MinTree_H_{i+1} 的最大高度 H_{i+1} 即 $H_{Max\_H\_MergeTree} < H_i$ 且 $H_{Max\_H\_MergeTree} > H_{i+1}$（$i \neq Num\_M$）时，如果 $H_i - H_{Min\_H\_MergeTree} > 1$，用合并方法二的方法合并密钥管理树 H_MergeTree 和最小高度子树 MinTree_H_i；否则利用合并方法一的方法合并 H_MergeTree 和 MinTree_H_i，并用新合并的密钥管理树替代密钥管理树 Tree1 中的 MinTree_H_i，最后将 H_MergeTree 从容器 Set_T 删除；

[0039] 情形 3：当密钥管理树 H_MergeTree 的最大高度 $H_{Max\_H\_MergeTree}$ 小于最小高度子树 MinTree_H_1 的最大高度 H_1 时，密钥管理树 H_MergeTree 不可能一次性合并至密钥管理树 Tree1 中，采取将密钥管理树 H_MergeTree 分解的措施，分步将其合并至密钥管理树 Tree1：

[0040] c31、在密钥管理树 H_MergeTree 中搜索一棵高度为 H_1 的子树，记为子树 H_MergeTree_H_1；

[0041] c32、标记子树 H_MergeTree_H_1 的父节点到密钥管理树 H_MergeTree 根节点路径上的所有节点，移去标记节点后，将除子树 H_MergeTree_H_1 外的其他剩余子树添加记录到容器 Set_T；

[0042] c33、利用合并方法一的方法合并子树 H_MergeTree_H_1 和最小高度子树 MinTree_H_1，并用合并后的子树替代密钥管理树 Tree1 中的最小高度子树 MinTree_H_1；

[0043] c34、从容器 Set_T 中删除密钥管理树 H_MergeTree ；

[0044] 第五步 :检查容器 Set_T 是否为空，若为空则合并完成，否则清空容器 Set_M，返回第一步 ；

[0045] 其中，所述合并步骤中，所述的密钥管理树的最大高度其在数值上等于密钥管理树根节点的最大高度即根节点到其孩子节点的最大长度路径，密钥管理树的最小高度其在数值上等于密钥管理树根节点的最小高度即跟节点到其孩子节点的最小路径长度 ；

[0046] 所述的平衡节点为如果节点的最大高度和最小高度之差不超过 1，则称此节点为平衡节点 ；

[0047] 所述的平衡密钥管理树为若密钥管理树中所有节点都为平衡节点，则称此密钥管理树为平衡密钥管理树 ；

[0048] 所述的树的层为设定密钥管理树的根节点位于第 1 层，层数从根节点层次开始依次加 1 ；

[0049] 所述的最小高度子树为在一棵非完全的平衡密钥管理树中，若节点的最小高度和最大高度相等但其父节点的最小高度和最大高度不相等，定义由此节点及其所有孩子节点组成的子树为最小高度子树。

[0050] 本发明的发明目的是这样实现的 ：

[0051] 本发明 LKH 密钥管理树动态平衡方法通过对加入和离开密钥管理组的用户数目 N_J 和 N_D 进行判断，采样不同的方法对原始密钥管理树进行动态平衡，得到新的密钥管理树即平衡密钥管理树，从而保证了 LKH 在组密钥管理上的高效性。

附图说明

[0052] 图 1 是 LKH 采样密钥管理树对组密钥进行管理的示意图 ；

[0053] 图 2 是本发明中密钥管理树合并方法一合并示意图 ；

[0054] 图 3 是本发明中密钥管理树合并方法二合并示意图 ；

[0055] 图 4 是合并方法三中密钥管理树 Tree1 和最小高度子树一具体事例示意图 ；

[0056] 图 5 是合并方法三中情形 1 示意图 ；

[0057] 图 6 是合并方法三中情形 2 示意图 ；

[0058] 图 7 是合并方法三中情形 3 示意图 ；

[0059] 图 8 是合并方法三中执行第一合并后的密钥管理树示意图 ；

[0060] 图 9 是合并方法三中执行三次合并后的密钥管理树示意图 ；

[0061] 图 10 是本发明中原始密钥管理树示意图 ；

[0062] 图 11 是本发明加入用户数量大于离开用户数量的平衡示意图 ；

[0063] 图 12 是本发明加入用户数量等于离开用户数量的平衡示意图 ；

[0064] 图 13 是本发明加入用户数量大于离开用户数量的密钥管理树示意图 ；

[0065] 图 14 是图 13 所示的密钥管理树平衡后的示意图。

具体实施方式

[0066] 下面结合附图对本发明的具体实施方式进行描述，以便本领域的技术人员更好地理解本发明。需要特别提醒注意的是，在以下的描述中，当已知功能和设计的详细描述也许

会淡化本发明的主要内容时，这些描述在这里将被忽略。

[0067] 合并方法一

[0068] 在本实施例中，图 2（a）和（b）分别表示满足执行合并方法一的待合并密钥管理树 tree1 和 tree2。利用合并方法一合并两棵密钥管理树时，只需新建一个节点 new node 并将待合并密钥管理树 tree1 和 tree2 分别作为新建节点 new node 的左孩子子树和右孩子子树，合并后的密钥管理树如图 2（c）所示。

[0069] 合并方法二

[0070] 在本实施例中，图 3（a）表示密钥管理树 Tree1，图 3（b）表示删除成员 M2 后的密钥管理树 Tree1，由于成员 M2 离开，节点 0、节点 1 和节点 3 的密钥需要更新。现要求将删除成员 M2 后的密钥管理树 Tree1 与如图 3（c）所示的密钥管理树 Tree2 合并。根据合并方法二，首先在图 3（b)所示密钥管理树 Tree1 的第 2 层寻找一个待更新的节点，显然节点 1 满足要求，标记节点 1。然后新建一个节点 new node 替代标记的节点 1，并将以节点 1 为根节点的子树和 Tree2 作为新建节点 new node 的左右子树，合并后的密钥管理树如图 3（d）所示。

[0071] 合并方法三

[0072] 图 4（a)给出了满足执行合并方法三条件的待合并密钥管理树 Tree1，初始的密钥管理树 Tree1 共有三棵最小高度子树如图 4（b）所示，它们的高度分别为 3、1、1，将密钥管理树 Tree1 的最小高度子树记录到容器 Set_M 并按高度降序排序，即 H_1=3，H_2=1，H_3=1。同时，在执行合并步骤前，还需将待合并的密钥管理树 Tree2 添加到容器 Set_T 中。根据待合并密钥管理树 Tree2 的高度，分三种情形详细讨论合并方法三。在此需要说明的是第一次从容器 Set_T 中选取高度最大的待合并树 H_MergeTree 即为合并密钥管理树 Tree2。

[0073] 情形 1：待合并密钥管理树 Tree2 的最大高度和密钥管理树 Tree1 某棵最小高度子树的高度相等。

[0074] 如图 5（a)所示的密钥管理树 Tree2 满足情形 1，有 $H_{Max\_Tree2}=H_1$。利用合并方法一的方法合并密钥管理树 Tree2 和最小高度子树 MinTree_H_1，并用合并后的树替代密钥管理树 Tree1 中高度为 H_1 最小高度子树 MinTree_H_1。此情形下，合并密钥管理树 Tree1 和 Tree2 形成的最终合并树如图 5（b)。

[0075] 情形 2：待合并密钥管理树 Tree2 的最大高度介于密钥管理树 Tree1 的两棵最小高度子树的高度之间。

[0076] 图 6（a)所示的密钥管理树 Tree2 满足情形 2，有 $H_1>H_{max\_Tree2}>H_2$。利用合并方法二的方法合并密钥管理树 Tree2 和最小高度子树 MinTree_H_1，并用合并后的树替代密钥管理树 Tree1 中高度为 H_1 的最小高度子树 MinTree_H_1。此情形下，合并密钥管理树 Tree1 和密钥管理树 Tree2 形成的最终合并树如图 6（b）所示。

[0077] 情形 3：待合并的密钥管理树 Tree2 的最大高度大于密钥管理树 Tree1 中高度最大的最小高度子树。

[0078] 图 7（a)所示的密钥管理树 Tree2 满足情形 3，有 $H_{max\_Tree2}>H_1$。此时需要将密钥管理树 Tree2 拆分，首先在密钥管理树 Tree2 中找一棵高度和 H_1 相等子树，记为 H_MergeTree_H_1，如图 7（b)所示，密钥管理树 Tree2 的剩余子树如图 7（c)所示，剩余子树被添加到容器 Set_T 中。利用合并方法一的方法将子树 H_MergeTree_H_1 和最小高度子树 MinTree_H_1 合

并，并利用合并后的树替代密钥管理树 tree1 中的最小高度子树 $MinTree\_H_1$，经过第一次合并后的 Tree1 如图 8 所示。将密钥管理树 Tree2 从容器 Set_T 中删除，第一次合并结束，但此时容器 Set_T 中还有密钥管理树 Tree2 的剩余子树，需继续执行合并操作，清空容器 Set_M 返回。执行完三次合并操作后，最终的密钥管理树如图 9 所示。

[0079] 密钥管理树的动态平衡方法

[0080] 假若密钥管理中心为组内用户建立了一棵如图 10 所示的密钥管理树即原始密钥管理树。现在具体实施组内用户动态变化时，密钥管理树动态变化过程。

[0081] 情形 1 :加入用户数量大于离开用户数量

[0082] 如图 10，假若成员 M_4 和 M_7 离开，成员 M_{12}、M_{13}、M_{14}、M_{15} 和 M_{16} 将加入。首先选取加入成员 M_{12} 和 M_{13} 替代离开成员 M_4 和 M_7，然后考虑将剩余加入成员 M_{14}、M_{15} 和 M_{16} 和原始密钥管理树合并成一棵新的平衡密钥管理树。具体操作 :将 M_{14}、M_{15} 和 M_{16} 看成单节点密钥管理树，加上原始密钥管理树，共有四棵密钥管理树，每次从密钥管理树中选择高度最小的两棵密钥管理并利用合并方法将它们合并为一棵密钥管理树，直到最终只剩一棵密钥管理树。经过前两次合并，成员 M_{14}、M_{15} 和 M_{16} 合并成如图 11（a）所示的密钥管理树。经过第三次合并后，最终的密钥管理树如图 11（b）所示。

[0083] 情形 2 :加入用户与离开用户数量相等

[0084] 假若成员 M_4 和 M_7 离开，成员 M_{12} 和 M_{13} 加入，直接用加入用户替代离开用户，得到的密钥管理树如图 12 所示。

[0085] 情形 3 :加入用户数量小于离开用户数量

[0086] 假若成员 M_4 和 M_7 离开，没有成员加入。删除离开用户 M_4 和 M_7 得到如图 13 所示的密钥管理树。由于用户的删除可能造成密钥管理树的不平衡，所以从左至右、从下至上依次检查密钥管理树中所有节点是否平衡。当检查到节点 1 时，发现节点 1 不平衡，这里采用合并算法 3 将节点 1 的左右两棵子树合并为一棵新的平衡密钥管理树，然后用它替代节点 1。平衡后的密钥管理树如图 14 所示。

[0087] 尽管上面对本发明说明性的具体实施方式进行了描述，以便于本技术领域的技术人员理解本发明，但应该清楚，本发明不限于具体实施方式的范围，对本技术领域的普通技术人员来讲，只要各种变化在所附的权利要求限定和确定的本发明的精神和范围内，这些变化是显而易见的，一切利用本发明构思的发明创造均在保护之列。

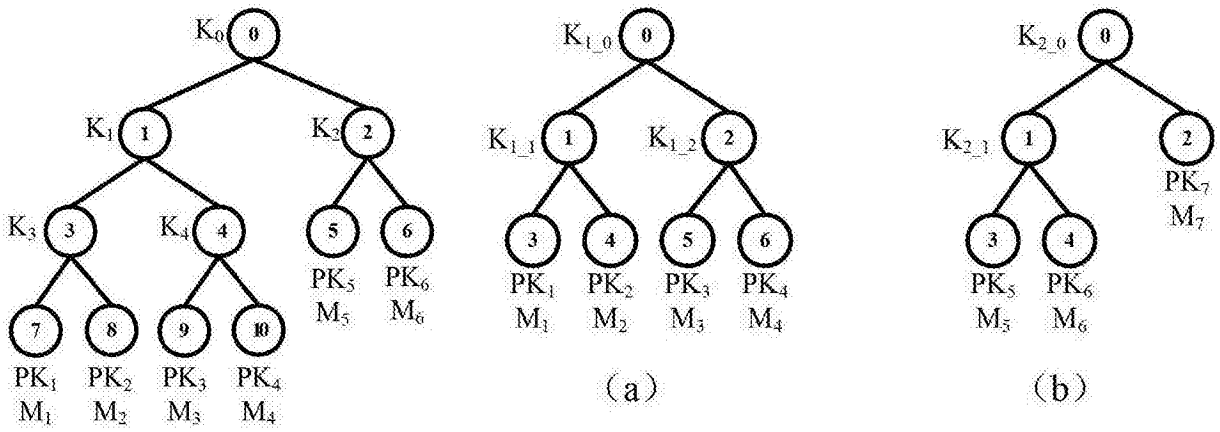

K0 0
K1 1
K2 2
K3 3
K4 4
5
6
7
8
9
10
PK5 M5
PK6 M6
PK1 M1
PK2 M2
PK3 M3
PK4 M4
K1_0 0
K1_1 1
K1_2 2
3
4
5
6
PK1 M1
PK2 M2
PK3 M3
PK4 M4
(a)
K2_0 0
K2_1 1
2
3
4
PK7 M7
PK5 M5
PK6 M6
(b)

图 1

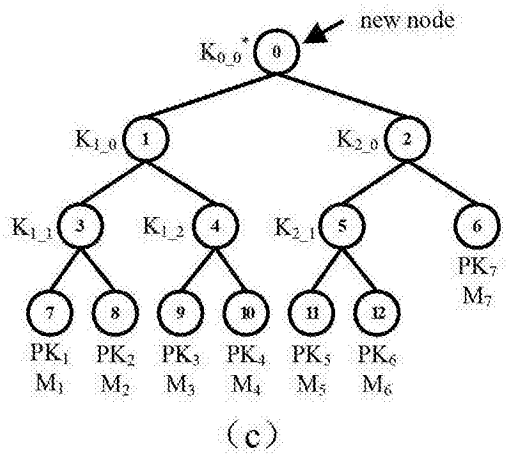
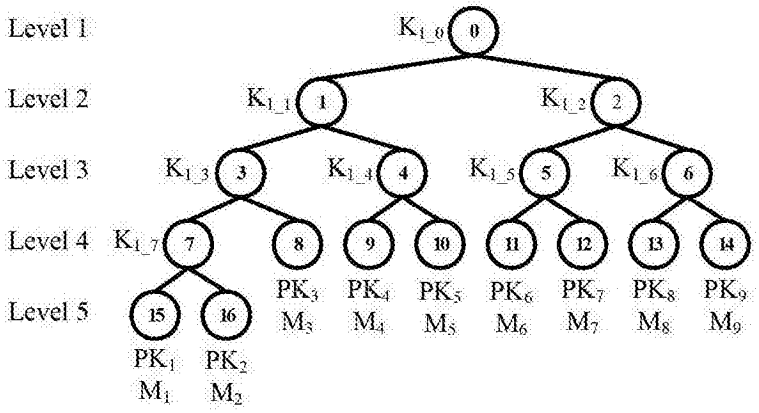
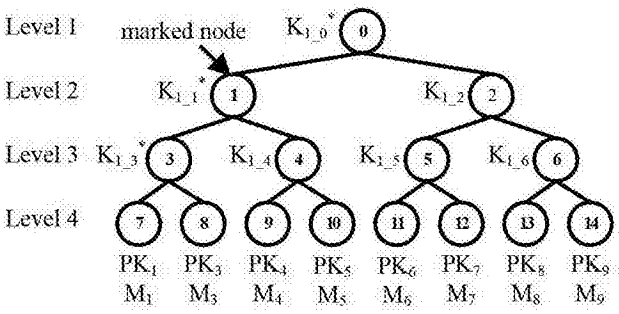

new node
K0_0* 0
K1_0 1
K2_0 2
K1_1 3
K1_2 4
K2_1 5
6
PK7 M7
7
8
9
10
11
12
PK1 M1
PK2 M2
PK3 M3
PK4 M4
PK5 M5
PK6 M6
(c)

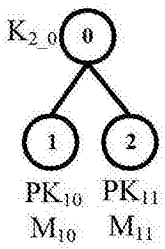
图 2


Level 1
Level 2
Level 3
Level 4
Level 5
K_{1_0} 0
K_{1_1} 1
K_{1_2} 2
K_{1_3} 3
K_{1_4} 4
K_{1_5} 5
K_{1_6} 6
K_{1_7} 7
8 9 10 11 12 13 14
15 16
PK_1 M_1
PK_2 M_2
PK_3 M_3
PK_4 M_4
PK_5 M_5
PK_6 M_6
PK_7 M_7
PK_8 M_8
PK_9 M_9

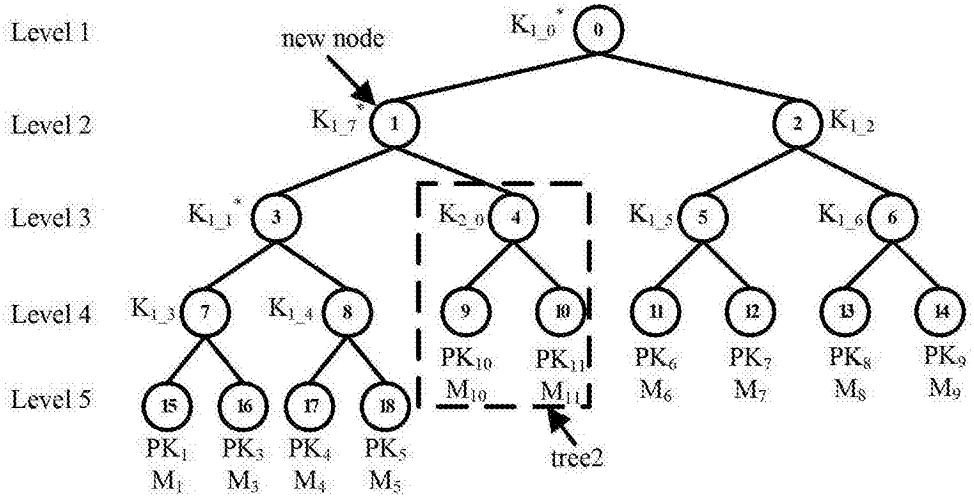
(a)


Level 1
Level 2
Level 3
Level 4
marked node
$K_{1\_0}^*$ 0
$K_{1\_1}^*$ 1
K_{1_2} 2
$K_{1\_3}^*$ 3
K_{1_4} 4
K_{1_5} 5
K_{1_6} 6
7 8 9 10 11 12 13 14
PK_1 M_1
PK_3 M_3
PK_4 M_4
PK_5 M_5
PK_6 M_6
PK_7 M_7
PK_8 M_8
PK_9 M_9

(b)


K_{2_0} 0
1 2
PK_{10} M_{10}
PK_{11} M_{11}

(c)


Level 1
Level 2
Level 3
Level 4
Level 5
new node
$K_{1\_0}^*$ 0
$K_{1\_7}^*$ 1
2 K_{1_2}
$K_{1\_1}^*$ 3
K_{2_0} 4
K_{1_5} 5
K_{1_6} 6
K_{1_3} 7
K_{1_4} 8
9 10 11 12 13 14
15 16 17 18
PK_1 M_1
PK_3 M_3
PK_4 M_4
PK_5 M_5
PK_{10} M_{10}
PK_{11} M_{11}
PK_6 M_6
PK_7 M_7
PK_8 M_8
PK_9 M_9
tree2

(d)

图 3

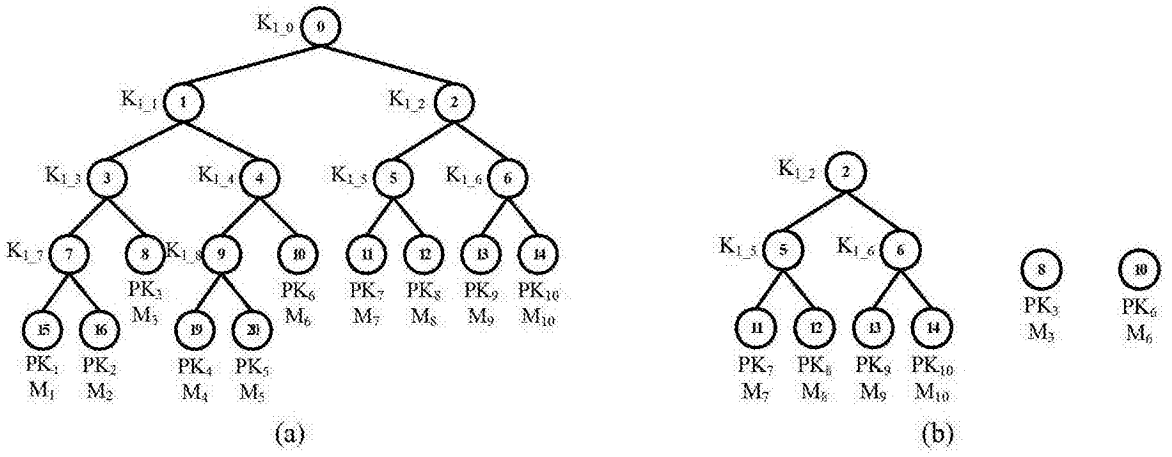
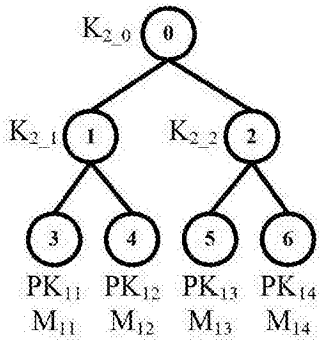
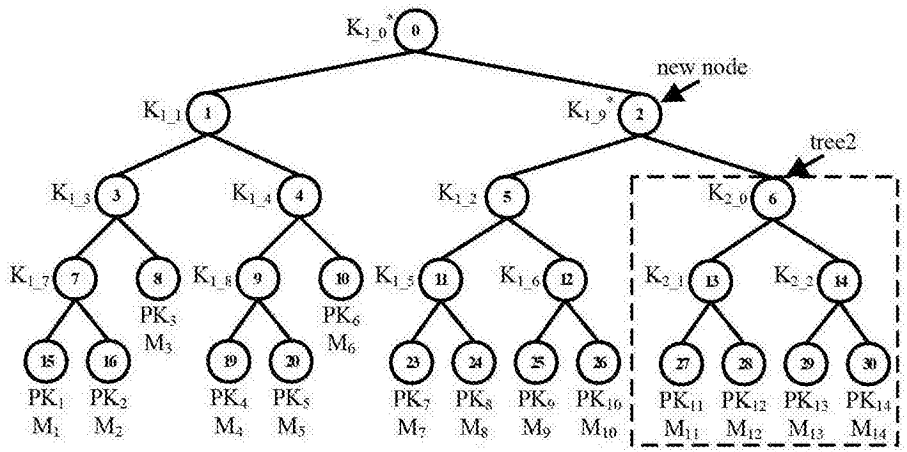

K1_0 0
K1_1 1
K1_2 2
K1_3 3
K1_4 4
K1_5 5
K1_6 6
K1_7 7
8
K1_8 9
10
11
12
13
14
15
16
19
20
PK3 M3
PK6 M6
PK7 M7
PK8 M8
PK9 M9
PK10 M10
PK1 M1
PK2 M2
PK4 M4
PK5 M5
(a)
K1_2 2
K1_5 5
K1_6 6
11
12
13
14
PK7 M7
PK8 M8
PK9 M9
PK10 M10
8
10
PK3 M3
PK6 M6
(b)

图 4

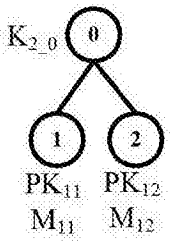

K2_0 0
K2_1 1
K2_2 2
3
4
5
6
PK11 M11
PK12 M12
PK13 M13
PK14 M14

(a)

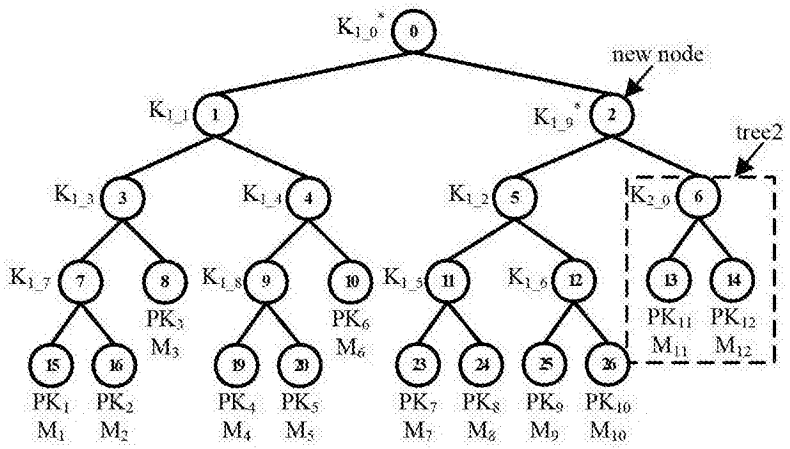

K1_0* 0
new node
K1_1 1
K1_9* 2
tree2
K1_3 3
K1_4 4
K1_2 5
K2_0 6
K1_7 7
8
K1_8 9
10
K1_5 11
K1_6 12
K2_1 13
K2_2 14
15
16
19
20
23
24
25
26
27
28
29
30
PK3 M3
PK6 M6
PK1 M1
PK2 M2
PK4 M4
PK5 M5
PK7 M7
PK8 M8
PK9 M9
PK10 M10
PK11 M11
PK12 M12
PK13 M13
PK14 M14

(b)

图 5


K2_0
0
1
2
PK11
M11
PK12
M12

(a)


K1_0*
0
new node
K1_1
1
K1_9*
2
tree2
K1_3
3
K1_4
4
K1_2
5
K2_0
6
K1_7
7
8
K1_8
9
10
K1_5
11
K1_6
12
13
14
PK3
M3
PK6
M6
PK11
M11
PK12
M12
15
16
19
20
23
24
25
26
PK1
M1
PK2
M2
PK4
M4
PK5
M5
PK7
M7
PK8
M8
PK9
M9
PK10
M10

(b)

图 6

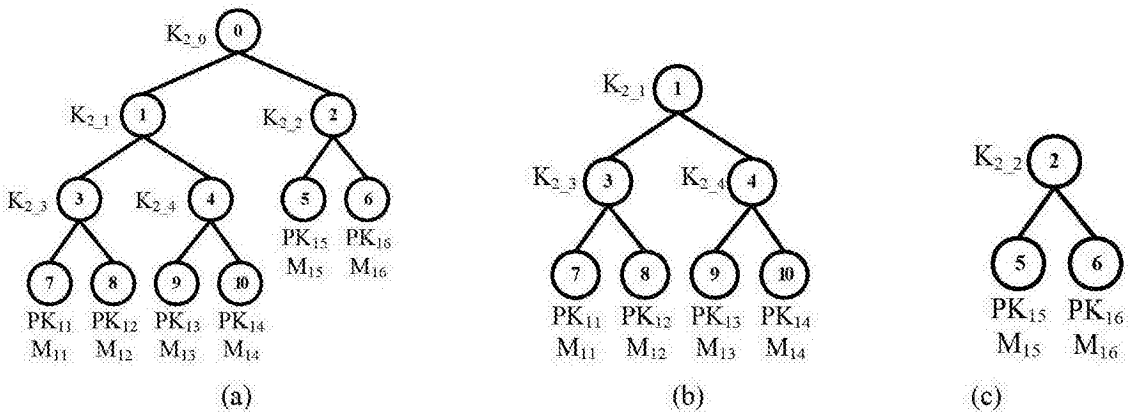

K2_0
0
K2_1
1
K2_2
2
K2_3
3
K2_4
4
5
6
PK15
M15
PK16
M16
7
8
9
10
PK11
M11
PK12
M12
PK13
M13
PK14
M14
(a)
(b)
(c)

图 7

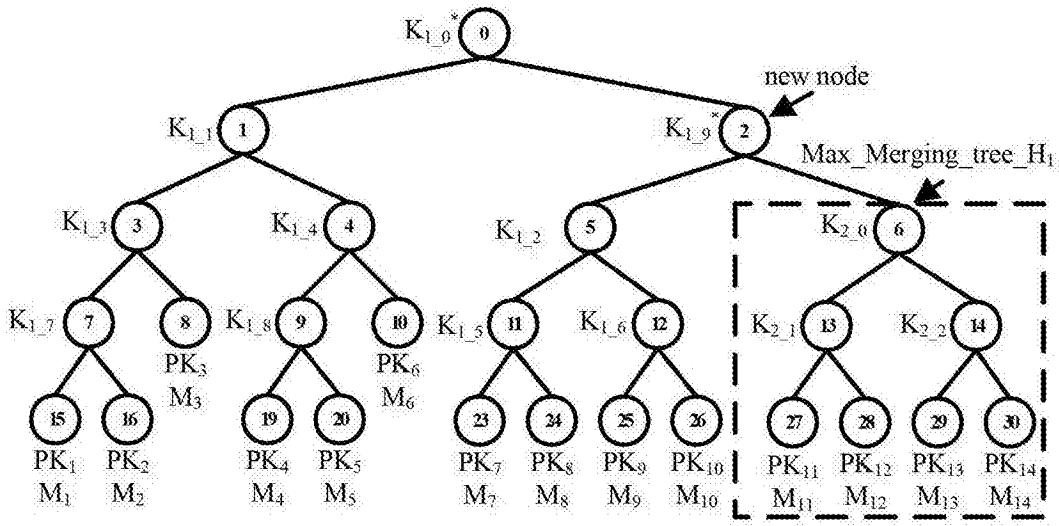

K1_0*
0
new node
K1_1
1
K1_9*
2
Max_Merging_tree_H1
K1_3
3
K1_4
4
K1_2
5
K2_0
6
K1_7
7
8
K1_8
9
10
K1_5
11
K1_6
12
K2_1
13
K2_2
14
PK3
M3
PK6
M6
15
16
19
20
23
24
25
26
27
28
29
30
PK1
M1
PK2
M2
PK4
M4
PK5
M5
PK7
M7
PK8
M8
PK9
M9
PK10
M10
PK11
M11
PK12
M12
PK13
M13
PK14
M14

图 8

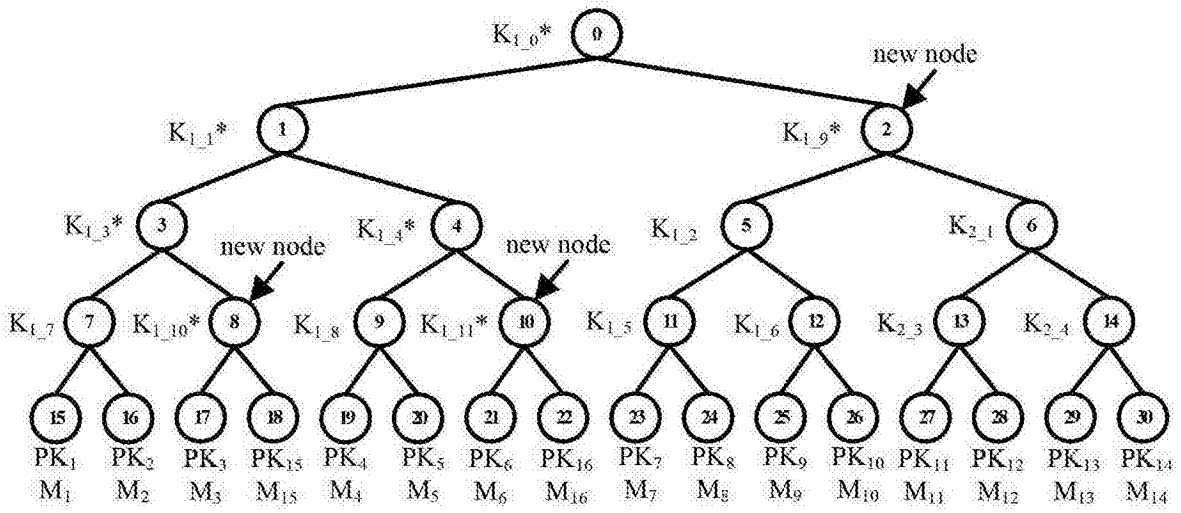

K1_0*
0
new node
K1_1*
1
K1_9*
2
K1_3*
3
new node
K1_4*
4
new node
K1_2
5
K2_1
6
K1_7
7
K1_10*
8
K1_8
9
K1_11*
10
K1_5
11
K1_6
12
K2_3
13
K2_4
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
PK1
M1
PK2
M2
PK3
M3
PK15
M15
PK4
M4
PK5
M5
PK6
M6
PK16
M16
PK7
M7
PK8
M8
PK9
M9
PK10
M10
PK11
M11
PK12
M12
PK13
M13
PK14
M14

图 9

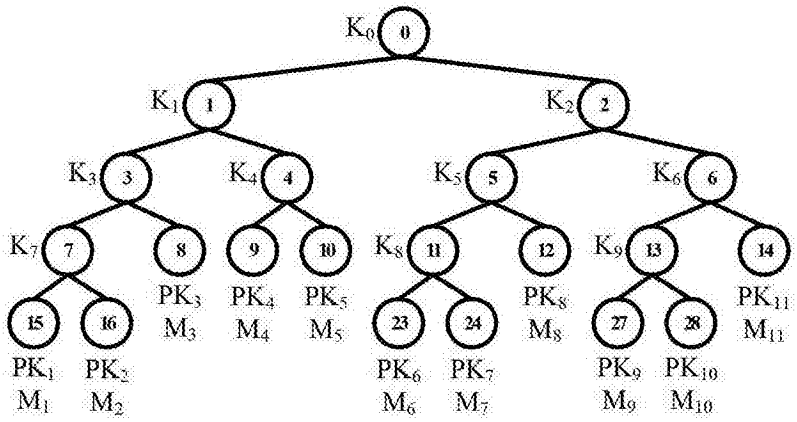
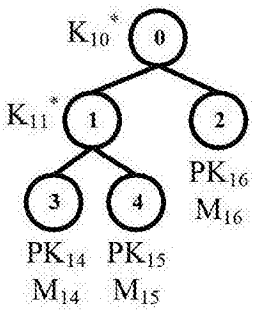
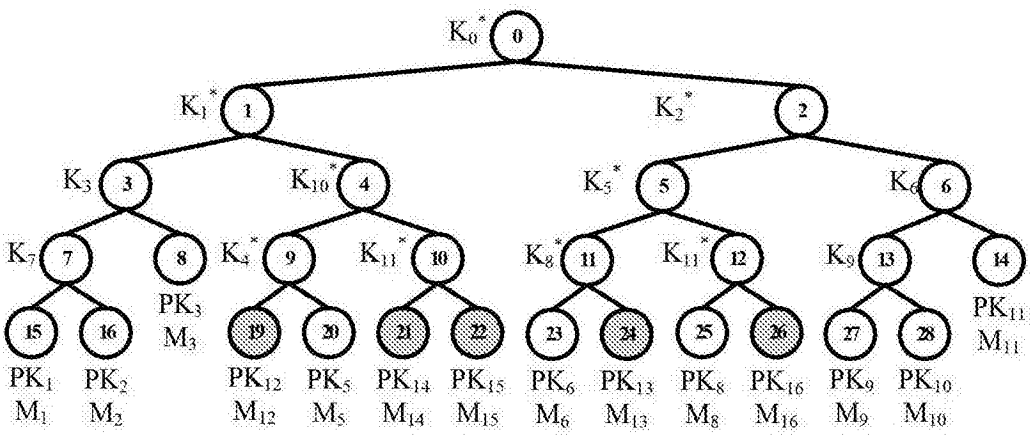

K0 0
K1 1
K2 2
K3 3
K4 4
K5 5
K6 6
K7 7
8
9
10
K8 11
12
K9 13
14
15
16
PK3 M3
PK4 M4
PK5 M5
23
24
PK8 M8
27
28
PK11 M11
PK1 M1
PK2 M2
PK6 M6
PK7 M7
PK9 M9
PK10 M10

图 10


K10* 0
K11* 1
2
3
4
PK16 M16
PK14 M14
PK15 M15

(a)


K0* 0
K1* 1
K2* 2
K3 3
K10* 4
K5* 5
K6 6
K7 7
8
K4* 9
K11* 10
K8* 11
K11* 12
K9 13
14
PK3 M3
15
16
19
20
21
22
23
24
25
26
27
28
PK11 M11
PK1 M1
PK2 M2
PK12 M12
PK5 M5
PK14 M14
PK15 M15
PK6 M6
PK13 M13
PK8 M8
PK16 M16
PK9 M9
PK10 M10

(b)

图 11

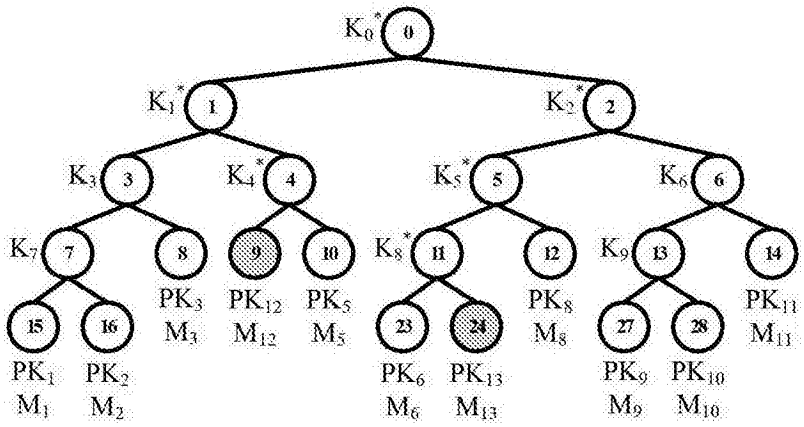

K0* 0
K1* 1
K2* 2
K3 3
K4* 4
K5* 5
K6 6
K7 7
8
9
10
K8* 11
12
K9 13
14
15
16
PK3 M3
PK12 M12
PK5 M5
23
24
PK8 M8
27
28
PK11 M11
PK1 M1
PK2 M2
PK6 M6
PK13 M13
PK9 M9
PK10 M10

图 12

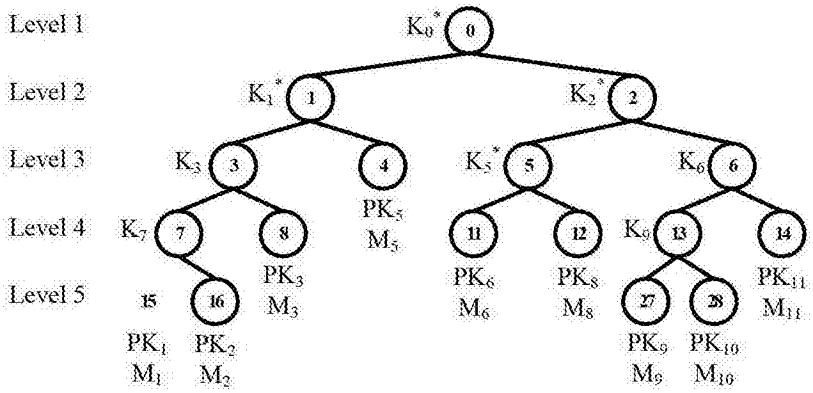

Level 1
Level 2
Level 3
Level 4
Level 5
K0* 0
K1* 1
K2* 2
K3 3
4
K5* 5
K6 6
K7 7
8
PK5 M5
11
12
K9 13
14
PK3 M3
PK6 M6
PK8 M8
15
16
27
28
PK11 M11
PK1 M1
PK2 M2
PK9 M9
PK10 M10

图 13

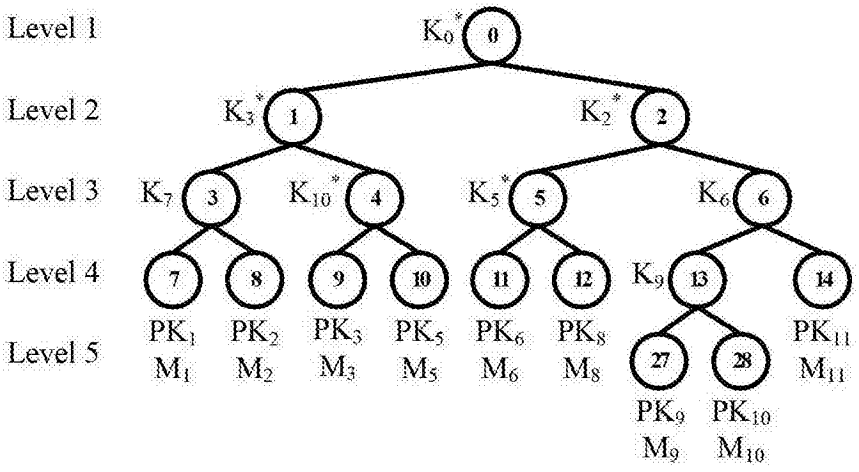

Level 1
Level 2
Level 3
Level 4
Level 5
K0* 0
K3* 1
K2* 2
K7 3
K10* 4
K5* 5
K6 6
7
8
9
10
11
12
K9 13
14
PK1 M1
PK2 M2
PK3 M3
PK5 M5
PK6 M6
PK8 M8
27
28
PK11 M11
PK9 M9
PK10 M10

图 14