一种新的组密钥管理算法*)

唐 扬 兰巨龙

(国家数字交换系统工程技术研究中心 郑州 450002)

摘 要 在对组密钥管理算法进行全面研究的基础上,提出了一种新的用于安全组播的组密钥管理算法。本算法将集中式密钥分配算法与分布式密钥协商算法相结合,吸取了集中式密钥分配算法的可扩展性的优点,同时克服了其单点失效的问题,将密钥分配思想用于密钥协商算法中,产生了一种底层基于密钥链而上层组织为三叉密钥树的密钥分发算法。本算法可以代替密钥协商算法应用于所有成员关系对等且不存在可信第三方的组播组中,并克服了分布式密钥协商算法中的大计算延时问题,适用于成员关系变化频繁的大型组播组,具有较高的可扩展性。

关键词 安全组播,组密钥管理,密钥协商,密钥链,密钥树结构

New Group Key Management Algorithm

TANG Yang LAN Ju-long

(National Digital Switching System Engineering & Technological Research Center, Zhengzhou 450002, China)

Abstract Based on the research of group key management algorithm, a new group key management algorithm applied to secure multicast was proposed. The algorithm combins the conentrated key assignment algorithm with the distributed key agreement algorithm, has good extensiblity and overcomes the single point failure and large computation delay. So it can substitute the key assignment algorithm for being applied to a loarge multicast group in which all the members are equal and there is no trusted third part and the relationships among them change frequently.

Keywords Secure multicast, Group key management, Key agreement, Key china, Key tree structure

1 引言

组密钥管理算法是安全组播中的一个热点,组密钥管理算法根据密钥材料的维护者和建立者的不同可分为集中式密钥分配算法(key distributed)[1-3]和分布式密钥协商算法(key agreement)[4-6]两大类。集中式密钥分配算法中组密钥的建立依赖可信第三方 TTP(Trusted Third Part)完成,当组密钥需要更新时,由该实体来产生和分发密钥,这种管理方法组密钥建立时间短,效率高,但计算开销、通讯开销和存储开销主要集中在该实体,并存在单点失效问题。分布式密钥协商算法应用于对等实体所建立的组播组中,并不存在一个可以生成并分发密钥的第三方可信实体(如 Ad-HOC 网),组密钥由全组成员共同协商生成,因此不存在单点失效问题。但这种密钥管理算法的主要问题在于 DH[7]算法依赖于比较耗时的模指运算,以及全体成员共同协商导致组密钥建立时间长,随着成员数量的增加,可扩展性逐渐降低,以 TGDH[8]算法为例,该算法可扩展性不高的主要原因在于:

(1)在组建立阶段每个成员平均要做 $O(\log_2 n)$次模指运算;

(2)在成员加入或离开时,每个成员要做 $O(\log_2 D)$(D 表示加入或离开节点的高度)次模指运算。

TGDH 算法建立组密钥时需要 $2D$ 次串行的模指运算,模指运算对计算机来说是一种运算量较大且耗时的工作,这将导致组密钥建立时延较大,从 QoS 的角度看,组密钥建立的时间长短直接与用户的满意程度相关。表 1 为密钥长度是 1024 位素数模求幂与密钥长度是 128 位加密 128 位分组的 AES[13]运算计算开销的试验结果,试验平台为一台 P4 2.4G CPU,512MB 内存的 Linux AS 4 主机,采用大数运算库为著名的 Cryto++[12]。实验结果表明,模指运算比普通的加解密运算耗时高一个数量级以上,当组播组规模较大时,密钥协商算法产生组密钥时间较长,无法满足大规模动态组播组的需求。本文介绍一种将密钥分配算法与密钥协商算法相结合的密钥管理方法,使模指运算次数达到最低,从而降低密钥建立时间。

表 1

运算种类	运算速率 kbps
AES	551.8
模指运算	48.6

2 基于密钥链的密钥分配算法

符号系统定义:

n 是组成员个数;d 是密钥树的度;$key(x)$是节点 x 的密钥;$par(x)$是节点 x 的父节点;$l\text{-}child(x)$是节点 x 的左孩子节点;$m\text{-}child(x)$是节点 x 的中间孩子节点;$r\text{-}child(x)$是节点 x 的右孩子节点;$first(x)$是以节点 x 为根节点的树的最左叶节点;$last(x)$是以节点 x 为根节点的树的最右叶节点;M_i 是第 i 个成员;α_i 是成员 i 的私钥;$g^{\alpha_i\alpha_j} \bmod p$ 是成员 M_i

*)国家 863 重大专项"可扩展到 T 比特的高性能 IPv4/v6 路由器基础平台及实验系统"(总体组成员),国家 863 重大课题"高性能 IPv6 路由器协议栈软件",国家 863 重大课题(总体组成员)。**唐 扬** 研究生,助教,主要研究方向为网络安全;**兰巨龙** 教授,博士生导师,主要研究方向为网络交换。

和成员 M_j 之间的共享密钥；$\{m\}_k$ 是用密钥 k 加密消息 m。

2.1 密钥链的生成

如前文所述，组内所有成员关系对等的组播组因为没有可信第三方可以为组内成员生成和分发密钥，所以需要采用协商的方法生成密钥。本算法在组建立时将所有成员逻辑上排成一列，并编号，相邻成员采用 two-part DH 密钥协商，每两个成员之间共享一个密钥，从而可以通过成员之间的共享密钥作为传递信息的通道，该通道称为密钥链。密钥链建立的具体方法如下：每个成员 M_i 与相邻的成员 M_j（$|i-j|=1$）进行 two-part DH 密钥协商。由于所有节点是同时与左右相邻的节点进行 two-part DH 密钥协商的，因此只需一轮 DH 密钥交换时间时延，就可以为所有成员生成一条密钥链，每个成员 M_i 与 $M_{i+1}(1<i<n)$ 之间协商出一个共享密钥。M_i 与左邻居 M_{i-1} 之间的共享密钥称为 M_i 的 leftkey，其左邻居用 $left(M_i)$ 表示；M_i 与右邻居 M_{i+1} 之间的共享密钥称为 M_i 的 rightkey，其右邻居用 $right(M_i)$ 表示。图 1 为 8 个加入该组的成员所形成的密钥链。处于密钥链首尾位置的节点只有唯一的邻居节点，所以只有一个共享密钥，如 M_1 只拥有 $g^{\alpha_1\alpha_2} \bmod p$；而处于密钥链中部的节点因为有左右两个邻居，所以拥有两个共享密钥，如 M_3 同时拥有 $g^{\alpha_2\alpha_3} \bmod p$ 和 $g^{\alpha_3\alpha_4} \bmod p$。密钥链可以为其中的成员提供传输消息的安全通道，如成员 M_3 向成员 M_5 发送消息 m，则首先用自己的 rightkey 加密 m，即将 $\{m\}_{g^{\alpha_3\alpha_4} \bmod p}$ 单播给 M_4，M_4 用其 leftkey 解密 m，然后再用自己的 rightkey 加密 m，将 $\{m\}_{g^{\alpha_3\alpha_4} \bmod p}$ 单播给 M_5，M_5 用 leftkey 将其解密即可得到 m。

图 1 形成的密钥链

2.2 密钥树的最优结构

在密钥链形成之后，为了提高密钥算法的效率，引入辅助密钥将所有成员组织成密钥树。图 2 是将图 1 中的密钥树组织成一棵满二叉树。根据研究[9]，算法的效率取决于密钥树的结构，具有最优的密钥树结构才能具有最优的算法效率。密钥树的度是密钥树最关键的结构参数。下面将以评价密钥算法效率的三个开销：存储开销、通信开销和计算开销作为指标对密钥树的度进行研究。

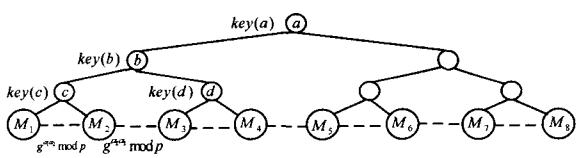


图 2 密钥树的生成

①算法的存储开销

在密钥树中，每个叶节点代表组成员，非叶节点代表辅助密钥且根节点为组密钥。每个组成员拥有密钥链中与相邻节点协商出的共享密钥以及从该节点到根节点所在路径上的所有密钥，这条路径称为密钥路径（Key Path）。如图 2 所示，M_2 需要保存的密钥有共享密钥：$g^{\alpha_1\alpha_2} \bmod p$，$g^{\alpha_2\alpha_3} \bmod p$ 和密钥路径上的密钥：$key(a)$，$key(b)$ 以及 $key(c)$。对一棵平衡 d 叉树来说，每个叶节点（非密钥链中的首尾节点）密钥存储数量 $S(d)$ 为：

$$S(d)=\log_d n+2=\frac{\ln n}{\ln d}+2 \tag{1}$$

从式(1)中容易看出，$S(d)$ 随着 d 的增大而减小，即密钥树的度数越大，每个成员所需存储的密钥量越少，即存储开销越小。

②算法的通信开销

在密钥链建立后，每个辅助密钥的生成是建立在其孩子节点密钥的基础上的，即只有当它孩子节点的密钥产生之后，才能产生该节点的密钥。以图 2 中 b 节点的密钥产生过程为例，该节点的密钥由它的左孩子为根节点的密钥数的最右叶节点负责生成，即由 M_2 节点产生一随机密钥 $key(b)$ 作为 b 节点的密钥，然后用辅助密钥 $key(c)$ 加密并组播，这样拥有该辅助密钥的成员就可以拥有 $key(b)$，M_2 再将 $key(b)$ 用共享密钥 $g^{\alpha_2\alpha_3} \bmod p$ 加密，发送给邻节点 M_3，M_3 利用辅助密钥 $key(d)$ 将 $key(b)$ 加密后发送给拥有该辅助密钥的组成员，从而完成 $key(b)$ 的生成与分发。密钥树度数的不同，负责生成和分发密钥的节点有可能不同，如对于 3 叉树，可以利用其中子树的最右叶节点同时向左右两边分发密钥。密钥路径中的每个密钥是自底向上生成的，如辅助密钥 $key(c)$ 由 M_2 负责生成并单播给 M_1，这需要 $d-1$ 次通信。M_2 分发辅助密钥 $key(b)$ 时，要利用辅助密钥 $key(c)$ 组播给 b 节点的左子树内所有成员，然后单播给 M_3，M_3 利用辅助密钥 $key(d)$ 将其组播 b 节点右子树内的所有成员，这样需要 $2d-1$ 次通信，对一棵平衡 d 叉树来说，生成密钥路径上所有密钥所需要的通信量为：

$$C(d)=d-1+\sum_{i=1}^{\log_d n-1}(2d-1)=(2d-1)\log_d n-d \quad (d\geqslant n) \tag{2}$$

可以证明，当 $d\in[3,\infty)$ 时，有 $C(d)>0$，故当 $d=3$ 时，密钥更新所需通讯量最小。当 $d\in[2,3]$ 时，两者的通信量交替变化，如图 3(a)所示。根据仿真结果可以看出当度数为 2 或 3 时通信量最少。

③算法的计算开销

密钥分发依赖于加解密操作所有计算开销，这种开销主要是对密钥的加解密操作。如图 2 中辅助密钥 $key(c)$ 的产生需要成员 M_2 的一次加密操作和成员 M_1 的一次解密操作，即需要 $2d-1$ 次加解密操作。辅助密钥 $key(b)$ 的生成，需要以节点 b 为根的子树中的每个成员做一次解密操作，共需 d^D 次。以及成员 M_2 和 M_3 的 $2d-1$ 次加解密操作。所以对一棵 d 叉平衡树来说，生成密钥路径中的所有密钥所需的加解密次数为：

$$E(d)=2(d-1)+\sum_{i=2}^{\log_d n}[d^i+2(d-1)]=2(d-1)\log_d n+\frac{nd-d^2}{d-1}+1 \tag{3}$$

经过分析容易得知，$E(d)$ 随着 d 的增大而减小，即密钥树的度数越大，算法的计算开销越小。计算开销与密钥树的度数如图 3(b)所示。

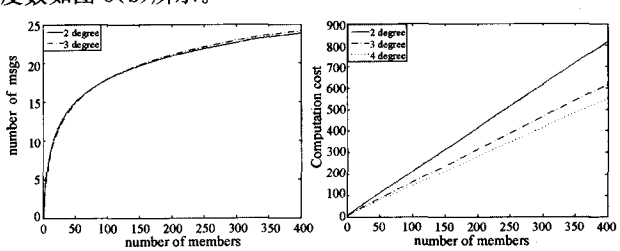


图 3(a)度数为 2 和 3 的比较　　(b)不同度数通信量的比较

对于密钥树度的选取，应该在大多数情况下（尤其是当 n 很大时）降低密钥管理算法中最核心的开销。在密钥协商算法中，主要的瓶颈是计算开销，而本算法采用分发的方式生成和分发辅助密钥和组密钥，从而降低了模指运算的次数并在很大程度上降低了计算开销，但分发的结果是增大了通信开销，所以本算法的核心开销是通信开销，综合对(1)－(3)的分析，可以得出，当密钥树的度数是 2 或 3 时，通信量最低，而在密钥树的度数为 3 时，算法的存储开销和计算开销均要低于度数为 2 的情况，所以度数为 3 的密钥树是本算法的最优密钥树结构。

2.3 组建立算法

组建立算法分为两步：首先根据 2.1 节中所述方法建立密钥链，然后将每 3 个成员分为一组（最后一组可能不是 3 个成员）自底向上生成一棵平衡的密钥树。每个子组父节点密钥的生成方式根据组内成员数量而不同：当组内成员个数为 3 时，由中间结点负责生成父节点密钥并分发给左右两边成员。当组内成员数为 2 时，父节点密钥就是两成员的共享密钥。如图 4 左半部分 c 节点的密钥由 M_5 负责生成，M_5 生成一随机密钥作为 c 节点的密钥并利用密钥链分发给 M_4 和 M_6。子组密钥以外的辅助密钥的生成方法为：由该节点的中子树的最右孩子负责生成和分发。如 a 节点的密钥由以 c 为子树的最右叶节点 M_6 负责生成，M_6 为 a 生成一随机密钥 $key(a)$，利用 $key(c)$加密并组播分发给 M_4 和 M_5，并用与 M_7 共享的密钥加密单播给 M_7，M_7 将 $key(a)$利用 $key(d)$加密分发给 M_8，M_4 将 $key(a)$用与 M_3 的共享密钥加密分发给 M_3，M_3 将 $key(a)$利用 $key(b)$加密组播给 M_1 和 M_2。组建立的最后每个成员都拥有了所在密钥路径上的所有密钥。在成员关系对等的组播组中，由于没有可信第三方实体负责维护密钥树结构，因此每个成员都要维护密钥树结构，在组建立时每个成员执行组密钥树生成算法 construct_kt()生成密钥树，然后执行密钥生成算法 KeySetup()根据成员所在密钥树中的不同位置生成每个节点的密钥。组建立算法 construct_kt(a,b,T)采用递归的方法将编号为 a 到 b 的成员组成以节点 T 为根的三叉树。如要将密钥链中的成员 M_1 到 M_n 组成以 T 为根的密钥树，对应组建立算法为 construct_kt(1,n,T)。下面是组建立算法中所要用到的函数的伪代码，left(a,b)，middle(a,b)和 right(a,b)将连续的序列号 a 到 b 分成 3 部分，new_node()的功能是建立一个新节点。

$$(x,y)=left(a,b)\{\ x=a;y=\left\lceil\frac{b}{3}\right\rceil\}$$

$$(x,y)=middle(a,b)\{x=\left\lceil\frac{b}{3}\right\rceil+1;y=\left\lceil\frac{b-\left\lceil\frac{b}{3}\right\rceil-1}{2}\right\rceil+\left\lceil\frac{b}{3}\right\rceil\}$$

$$(x,y)=right(a,b)\{if(\left\lceil\frac{b-\left\lceil\frac{b}{3}\right\rceil-1}{2}\right\rceil+\left\lfloor\frac{b}{3}\right\rfloor+1=b)\ x=y=b;else\ x=\left\lceil\frac{b-\left\lceil\frac{b}{3}\right\rceil-1}{2}\right\rceil+\left\lfloor\frac{b}{3}\right\rfloor+1,y=b\}$$

组建立算法的伪代码描述如下：

```
construct_kt(a,b,kt)
1. n=b-a;
2. if(n==0)
3.      return;
4. endif
5. if(n<3){ /* create new node for member */
6.     l-child(kt)=new_node();
7.     n=n-1;
8.     par(l-child(kt))=kt;
9.     if(n>0)
10.         m-child(kt)=new_node();
11.         n=n-1;
12.         par(m-child(kt))=kt;
13.    if(n>0)
14.         r-child(kt)=new_node();
15.         par(r-child(kt))=kt;
16.    }
17.    else {/* recursion create the tree */
18.    (x1,y1)=left(a,b);
19.    (x2,y2)=middle(a,b);
20.    (x3,y3)=right(a,b);
21.    construct_kt(x1,y1);
22.    construct_kt(x2,y2);
23.    construct_kt(x3,y3);
24.    }
```

KeySetup 算法用到下面的几个函数：

① $RECV_i(m)$：从成员 i 接收消息 m；

② $SEND_i(m)$：发送消息 m 给成员 i；

③ $MCAST_{i,j}(m)$：组播消息 m 给成员 i 到 j；

④ RandKey()：生成随机密钥；

⑤ NodeNumber(x)：返回节点 x 的孩子节点个数。

```
KeySetup(Mi)
1. x=par(Mi);
2. lnode=l-child(x);
3. mnode=m-child(x);
4. rnode=r-child(x);
5. if(NodeNumber==3){
6.   if(Mi==lnode)
7.        RECV_right(Mi){key(x)}_right(Mi);
8. else if(Mi==mnode){
9.        key(x)=RandKey();
10.       SEND_left(Mi){key(x)}_leftkey(Mi);
11.       SEND_right(Mi){key(x)}_rightkey(Mi);
12. }
13.   else RECV_left(Mi){key(x)}_leftkey(Mi)
14. }
15. else if(NodeNumber==2&&Mi=lnode)
16.       key(x)=rightkey(Mi)
17. else key(x)=leftkey(Mi)
18. x=par(x);
19. while(x! =NULL){ /* x is empty */
20.   lnode=l-child(x);
21.   mnode=m-child(x);
22.   rnode=r-child(x);
23.   if(Mi==last(lnode)){
24.       RECV_left(Mi){key(x)}_leftkey(Mi);
25.       MCAST_first(l),last(l){key(x)}_key(l);
26.   }
27.   if(first(lnode)<=Mi<last(lnode))
28.       RECV_last(lnode){key(x)}_key(lnode);
29.   if(Mi==last(mnode)){
30.       key(x)=RandKey();
31.       SEND_right(Mi){key(x)}_right(Mi);
32. MCAST_first(mnode),last(mnode){key(x)}_key(mnode);
```

```
33.  }
34.  if(Mi==first(mnode)){
35.       RECV_{last(mnode)}{key(x)}_{key(mnode)};
36.       SEND_{left(Mi)}{key(x)}_{leftkey(Mi)};
37.  }
38.  if(first(mnode)<Mi<last(mnode))
39.       RECV_{last(mnode)}{key(x)}_{key(mnode)};
40.  if(Mi=first(rnode)){
41.       RECV_{last(mnode)}{key(x)}_{leftkey(Mi)};
42.  MCAST_{first(rnode),last(rnode)}{key(x)}_{key(rnode)};
43.  }
44. if(first(rnode)<Mi<=last(rnode))
45.       RECV_{first(rnode)}{key(x)}_{key(rnode)};
46.  x=par(x);
47. }
```

2.4 成员加入算法

成员的加入和离开可能会破坏密钥树结构，导致密钥树不平衡，而算法的性能只有在最优结构时才能达到最优[11]，所以当密钥出现不平衡时，需要维护密钥树结构。密钥树的不平衡度定义为：以每个节点的孩子节点为根的子树的高度之差。当不平衡度大于等于 2 时，密钥树处于不平衡状态。合理选择成员加入位置有利于维护密钥树的结构，使算法性能达到最优。为了保持密钥树平衡，成员的加入位置应为密钥树中高度较低的位置。成员加入位置的搜索算法伪代码如下：

```
FindJoinPosition
1. x=root;
2. while(x! =NULL){
3.   if (x is not full){
4.        intsert position is x;
5.        return;
6.   }
7.   else {
8.        find the number of leaf nodes form
          the sub-trees rooted at l-child(x),
          m-child(x) and r-child(x),and store the
          results in l,m and r;
9.        if (l,m,r have the same value)
10.            set x =r-child(x);
11.       else
12.            set x with the node who has the
               smallest value from l,m,r;
13.  }
14. }
```

节点的加入会将原密钥链破坏，根据加入位置的不同，新节点要跟邻居节点进行密钥交换生成共享密钥，从而重建密钥链。密钥链重新建立后，为了满足后向安全性，加入节点所在密钥路径上的所有密钥都需要更新。密钥更新方法为：每个成员对所需要更新的密钥利用单向哈希函数（如 MD5）生成新的密钥。如图 4 中左半部分的密钥树在成员 M_9 加入后转化为图 4 的右半部分，M_9 加入后需要改变的密钥为 $key(a)$和 $key(d)$，拥有 $key(a)$的成员{M_1，M_2，M_3，M_4，M_5，M_6，M_7，M_8}对 $key(a)$进行哈希操作，产生新的密钥。拥有 $key(d)$的成员{ M_7，M_8}对 $key(d)$进行哈希操作产生新的密钥并由节点 M_8 将新产生的密钥分发给新加入节点 M_9，从而完成密钥更新。因为新加入节点在密钥链中与左右邻居节点之间拥有共享密钥，所以可以由左右邻居节点作为 sponsor 为新成员分发所需密钥，当成员的加入位置为密钥链的首部或尾部时，有唯一的邻居节点为其分发所需密钥，而当加入位置有两个邻居节点时，并不是两个节点都有为新成员分发密钥的能力（不具有新成员所在密钥路径上的所有密钥），邻居节点可以通过下面判定算法决定由哪个节点可以为加入节点分发密钥：

```
SelectSponer(M)
1. /* search M belong to which subtree from bottom to up */
2. set x=par(left(M));
3. while (x is not empty){
4.   if(subtree(x) has node M){
5.        set leftcount=distance form x to M;
6.        break;
7.   }
8.   else;
9.        set x=par(x);
10. }
11. set x=par(right(x));
12. while (x is not empty){
13.   if(M belong to subtree(x)){
14.        set rightcount=distance form x to M;
15.        break;
16.   }
17.   else
18.        set x=par(x);
19. }
20. /* compare leftcount with rightcount and the smaller is sponsor
    */
21. if(rightcount>=rightcount)
22.           left(M) is sponsor;
23. else if(rightcount<rightcount)
24.           rightt(M) is sponsor;
25. }
```

2.5 成员离开算法

成员的离开位置具有随机性，因此可能会导致密钥树不平衡，当密钥树不平衡时，需要做重平衡操作：方法是将密钥树中最深的节点移动到离开成员的位置。算法采用紧缩机制，即当一个辅助节点只剩下一个孩子节点时，将用该孩子节点代替辅助节点。为满足前向安全性，变动节点密钥路径上的所有密钥需要更新。当进行重平衡操作时，包括两条密钥路径：离开节点所在的密钥路径以及重平衡操作被移动节点所在的密钥路径。当变动节点位置在密钥链的首尾位置时，不需要 DH 密钥交换；当变动节点位置为密钥链的中部时，需要其两个邻居节点之间进行 DH 交换产生新的共享密钥。在更新密钥链后，自底向上更新密钥路径上的所有密钥。如图 4 中右半部分，当 M_9 离开后密钥树形如图 4 的左半部分。路径中需要变化的密钥为 $key(d)$和 $key(a)$，由 M_8 负责生成新的密钥并利用与 M_7 的共享密钥单播给 M_7，由 a 的中子树的最右叶节点 M_6 负责生成 a 节点的新密钥，利用 $key(c)$组播给 M_4 和 M_5，并用与 M_7 的共享密钥单播给 M_7，M_7 利用 $key(d)$发送给 M_8，M_4 将 a 节点的新密钥用与 M_3 的共享密钥单播给 M_3，M_3 利用 $key(b)$组播给 M_1 和 M_2，从而完成密钥更新。

当成员离开时，要用算法 BalanceTree()判断密钥树是否平衡，该算法中，high(node)函数返回以 node 节点为根的子树的高度，如果 node 为空，则高度为 0：

```
BalanceTree(root,M)
/*judge tree is balance or not*/
1. set l=high(l-child(root));
2. set m=high(m-child(root));
3. set r=high(r-child(root));
4. if(abs(l-m)>=2|abs(l-r)>=2|abs(r-m)>=2)
5.    tree is unbalance,find the max from<l,m,r>
      which stands for the highest tree; move the
      deepest leaf node from the highest tree to the
      leave position;
6. else
7.    tree is balance;
```

每个成员执行上述算法后将得到平衡的密钥树，然后对离开成员所在的密钥路径上的密钥进行密钥更新。当有重平衡操作时，被移动节点所在的密钥路径上的密钥也需要更新。更新算法首先将变动节点所在的密钥路径上的密钥标记出来。标记算法为 MarkKeyPath()，其中 mark(key)表示对密钥 key 做标记：

```
MarkKeyPath(M)
1. set x=par(M);
2. while(x! =NULL){
3.     mark(key(x));
4.     set x=par(x);
5. }
```

对所需更新密钥进行标记后，每个成员执行下述算法自底向上进行密钥更新，更新算法如下：

```
KeyRefresh(Mi)
1. set x=par(Mi);
2. if (key (x) is marked){
3. set lnode=l-child(x);
4. set mnode=m-child(x);
5. set rnode=r-child(x);
6. if(NodeNumber(x)==3){
7.   if(Mi==lnode)
8.       RECV_{right(Mi)}{key(x)}_{right(Mi)};
9.   else if(Mi==mnode){
10.      set key(x)=RandKey();
11.      SEND_{left(Mi)}{key(x)}_{leftkey(Mi)};
12.      SEND_{right(Mi)}{key(x)}_{rightkey(Mi)};
13.  }
14.  else
15.      RECV_{left(Mi)}{key(x)}_{leftkey(Mi)};
16. }
17. elseif(NodeNumber(x)==2&&Mi==lnode)
18.      set key(x)=rightkey(Mi);
19. else
20.      set key(x)=leftkey(Mi);
21. }
22. set x=par(x);
23. while(x! =NULL){ /*x is empty*/
24.   if(key(x) is marked){
25.      set lnode=l-child(x);
26.      set mnode=m-child(x);
27.      set rnode=r-child(x);
28.      if(Mi==last(lnode)){
29.          RECV_{left(Mi)}{key(x)}_{leftkey(Mi)};
30.          MCAST_{first(l),last(l)}{key(x)}_{key(l)};
31.      }
32.      if(first(lnode)<=Mi<last(lnode))
33.  RECV_{last(lnode)}{key(x)}_{key(lnode)};
34.      if(Mi==last(mnode)){
35.          set key(x)=RandKey();
36.          SEND_{right(Mi)}{key(x)}_{right(Mi)};
37. MCAST_{first(mnode),last(mnode)}{key(x)}_{key(mnode)};
38.          }
39.          if(Mi==first(mnode)){
40.          RECV_{last(mnode)}{key(x)}_{key(mnode)};
41.              SEND_{left(Mi)}{key(x)}_{leftkey(Mi)};
42.          }
43.          if(first(mnode)<Mi<last(mnode))
44.          RECV_{last(mnode)}{key(x)}_{key(mnode)};
45.          if(Mi==first(rnode)){
46.          RECV_{last(mnode)}{key(x)}_{leftkey(Mi)};
47.  MCAST_{first(rnode),last(rnode)}{key(x)}_{key(rnode)};
48.          }
49.          if(first(rnode)<Mi<=last(rnode))
50.              RECV_{first(rnode)}{key(x)}_{key(rnode)};
51.          set x=par(x);
52.          }
53. else
54.          set x=par(x);
54. }
```

3 安全性分析

组密钥管理的安全需求是为了满足前向安全性和后向安全性。前向安全性是指离开的成员不能解密后续的组播通信内容；后向安全性是指新加入成员不能解密加入前的通信内容。本算法的安全性依赖于 DH 算法以及单向哈希算法的强度。

3.1 单个成员的加入或离开的安全性分析

当单个成员加入组播组时，会对加入成员所在密钥路径上的所有密钥进行更新，采用的是单向哈希函数，所以算法的后向安全性取决于单向哈希函数的安全性。对于成员的退出，该成员所拥有的密钥均已更新，且该成员无法获得更新后的密钥，所以该成员不能通过后续的密钥更新报文获得组密钥，因此可以满足前向安全性。

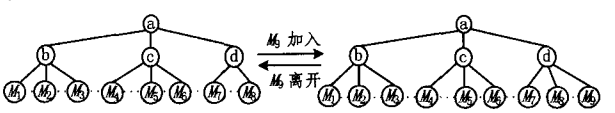


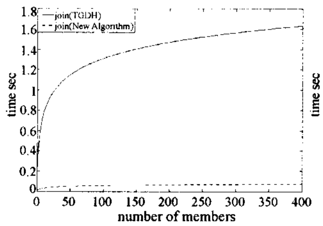
图 4 成员的加入和离开

3.2 多个成员同时加入或离开的安全性分析

组播组可能存在同谋破解的情况。同谋的目的就是要获得当所有同谋者都不是组播组成员时的组密钥。同谋的主要形式表现为以下两种：1)获得所有同谋者都退出组播组之后的组密钥，显然，如果同谋者中还有成员处于组播组时，所有这些同谋者均可获得组密钥，这种情况下组密钥对于非组成员的保密性毫无意义。2)在这些同谋者相继加入组播组之后进行勾结以获得在首个加入组播组的同谋者加入组播组之前的组密钥。根据密钥管理算法，每个成员的加入或离开均要对密钥链与密钥路径中的密钥进行更新，所以：任意多个组成员利用他们所知的密钥以及密钥更新报文都不能获得他们所不应该知道的密钥。

4 性能分析

本算法的存储开销与基于密钥树的组密钥分发具有相同的特点：组成员的存储开销为 O(log$n$)。成员加入时最多只需 4 次 DH 计算和 5 条单播消息，而 TGDH 算法共需要 $2D+3$ 次 DH计算，以及 3 条单播消息。当成员离开时，最多需要 4 次 DH 计算和 D 次加解密计算及通信量，而 TGDH 算法需要 $2(D-1)$次 DH 计算和 2 次通信量，可见本算法在成员离开时将计算开销转化为通信开销。由于本算法采用 3 叉树，当成员数量相同时，本算法的效率要高于 TGDH 算法。算法的试验平台采用的是网络仿真工具 NS2，它可以完整地模拟整个网络环境。图 5 是 NS2 环境下得到的 TGDH 算法和新算法在成员加入时新的组密钥建立时间比较，从图中可以看出，本算法在组建立时具有明显的优势；图 6 是两种算法在成员离开时新的组密钥建立时间比较，当组规模越大，新算法优势越明显。



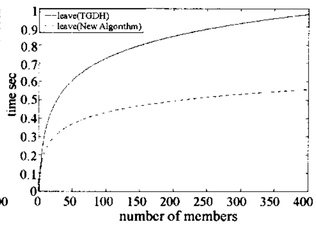


图 5 成员加入组密钥建立时间比较 图 6 成员离开时组密钥建立时间

结束语 本文探讨了一种可用于成员关系对等的组播组的密钥建立及分发的新算法。算法基于组播组的特性采用密钥协商的方法为成员通信建立基本的密钥链，并利用了集中式密钥管理中可信节点为组成员生成和分发密钥具有高效率高可扩展性的优点，根据节点位置的不同选择不同的 sponsor，负责生成和分发密钥，从而很好地解决了集中式密钥分发算法中的单点失效问题，实验证明本算法具有一定的实用性。

参 考 文 献

[1] Hardjono T , Cain B , Doraswamy N . A Framework for Group Key Management for Multicast Security[Z]. IETF,1999

[2] Wong C K,Gouda M,Lam S S. Secure group communications using key graphs. IEEE/ACM Transactions on Networking, 2000,8(1):16-30

[3] Setia S , Koussih S , Jajodia S , et al . Kronos : a scalable group rekeying approach for secure multicast[A]// IEEE Symposium on Security and Privacy[C]. Oakland(USA)CA: IEEE computer Society Press,2000:215-228

[4] Steiner M,Tsudik G,Waidner M. Diffie-Hellman Key Distribution Extended to Group Communication // Proceedings of 3rd ACM Conference on Computer and Communications Security [C]. 1996

[5] Burmester M , Desmedt Y . A Secure and Efficient Conference Key Distribution System

[6] Steiner M,Tsudik G,Waidner M. CLIQUES: A New Approach to Group Key Agreement// Proceedings of ICDCS'98[C]. 1998

[7] Diffe W,Hellman M E. New directions in cryptography. IEEE Trans. Inform. Theory,1976,IT-22:644-654

[8] Kim Y,Perrig A,Tsudik G. Simple and Fault-Tolerant Key Agreement for Dynamic Collaborative Groups// Jajodia S,ed. 7th ACM Conference on Computer and Communications Security [C]. Athens,Greece,2000:235-244

[9] Goshi J,Ladner R E. Algorithms for Dynamic Multicast Key Distribution Trees // 10th ACM Conference on Computer and Communications Security[C]. Boston,USA,2003:240-246

[10] Wallner D,Harder E,Age R. Key management for multicast:Issues and architectures. IETF RFC2627,1999

[11] Snoeyink J,Suhi S,Varghese G. A Lower Bound for Multicast Key Distribution// IEEE INFOCOM. 2001

[12] http//www. crypto. com

（上接第 109 页）

表 7 单个明文误差的扩散

步骤	元胞自动机A（rule30）	元胞自动机B（rule90）
0	————x————	————————
1	-xxxxxx-xx-xx--x———————	xxxxx-xx-xx--x————
2	-xxxxxx-xx—xx--x———————	————————————
3	—-xxxx-x-x-xx—x————————	-xxxx-x-x-xx—x————————
4	—-xxxx-x-x-xx--x————————	————————————
5	-x-xxx-x-xx-x—-x————————	-xxx-x-xx-x—-x————————
6	-x-xxx-x-xx-x—-x————————	————————————
7	-x——xx-xx——x——	——xx xx——x————————
8	-x——xx-xx——x————————	————————————
9	-x——x—x-x—x————————	——x-x-x—x————————

与前面的表 1 比较，由于多耦合元胞自动机系统对多个元胞进行了置换，扩大的两个元胞自动机相互影响的范围，使得原来单向作用的两个元胞自动机转变为相互作用，误差扩散的速度也有了明显的提高。

结束语 本文对耦合元胞自动机系统进行了深入的研究，给出了耦合元胞自动机系统的一般定义，并用耦合系数分析了不同耦合元胞自动机系统的动力学行为。针对单耦合元胞自动机密码系统中的不足，提出了一种新的多耦合元胞自动机密码系统。仿真结果表明，多耦合元胞自动机系统比单耦合元胞自动机系统具有更强的相互作用能力，更为快速的误差扩散，因此具有很高的安全性。

参 考 文 献

[1] 张传武，沈野樵，彭启琮. 细胞自动机反向迭代加密技术研究[J]. 计算机学报，2004，27(1)：125-129

[2] Wolfram S. Cryptography with cellular automata[C]. Advances in cryptology // Crypto' 85 Proceedings, LWCS, 218. Berlin: Springer -Verlag,1986:429-432

[3] Sipper M , Tomassini M . Generating parallel random number generators by cellular programming[J]. International Journal of Modern Physics C,1996,7(2):181-190

[4] Zhao X L,Li Q M,Xu M W,et al. A Synmmetric cryptography based on extended cellular automata[C]// IEEE-SMC2005. Hawaii USA,2005,10

[5] Guan P. Cellular automata public-key cryptosystems[J]. Complex System,1987,1:51-57

[6] Mihaljevic M,Zheng Y,Imai H. A family of fast dedicated one-way hash functions based on linear cellular automata over GF (q) [J]. IEICE Transactions on Fundamentals, 1999, E82-A (1):40-47

[7] Gutowitz H. Method and apparantus for encryption,decryption and authentication using dynamical systems [P]. USA: 5,365, 589,1994

[8] 赵学龙，游静，李千目，等. 耦合触发元胞自动机在数据加密中的应用[J]. 信息与控制，2005，34(6)：746-752