[19] 中华人民共和国国家知识产权局

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

[12] 发明专利说明书

专利号 ZL 00810761.0

[45] 授权公告日 2006 年 1 月 25 日

[11] 授权公告号 CN 1238989C

[22] 申请日 2000.7.20 [21] 申请号 00810761.0
[30] 优先权
[32] 1999. 7.23 [33] EP [31] 99305870.0
[86] 国际申请 PCT/GB2000/002813 2000.7.20
[87] 国际公布 WO2001/008348 英 2001.2.1
[85] 进入国家阶段日期 2002.1.23
[71] 专利权人 英国电讯有限公司
地址 英国伦敦
[72] 发明人 罗伯特·约翰·布里斯科
审查员 杨红丽

[74] 专利代理机构 北京三友知识产权代理有限公司
代理人 李 辉

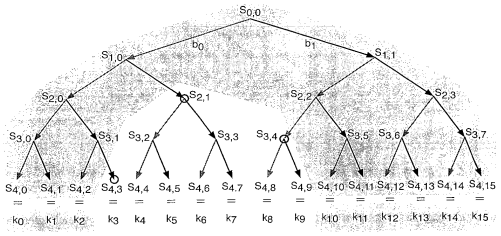
权利要求书 5 页 说明书 41 页 附图 19 页

[54] 发明名称

数据发布

[57] 摘要

在数据发布系统中，数据被划分为许多应用数据单元。 系统地生成一个密钥序列，在数据源用不同的密钥对每个数据单元进行加密。 在接收方，生成对应的密钥用于解密数据单元以使用该数据。 用于生成密钥的结构是：通过传送选择的一个或多个种子值的组合，用户可以使用整个密钥序列的一个固有限定的子集。



1. 一种发布数据的方法，包括:

(a) 对多个数据单元用密钥序列进行加密;

(b) 将加密过的数据单元传送给多个用户终端;

(c) 将至少一个种子值传送给一个用户终端;

(d) 从该种子值生成一个密钥序列，其密钥个数大于传送给用户终端的种子值的个数，所述序列构成了在步骤（a）中产生的密钥序列的一个具有上限和下限的任意部分，其中所述部分的上限和下限在序列中的位置由步骤（c）中传送的至少一个种子值决定；以及

(e) 在用户终端使用步骤（d）中产生的密钥序列解密数据单元。

2. 根据权利要求 1 所述的方法，其中步骤(a)中使用的密钥序列如下生成:

（A）对一个或多个初始种子值进行运算，生成更多个中间种子值，该中间种子值隐蔽该初始种子值;

（B）进一步对前一步骤生成的值进行运算，生成更多个深一层的值，该深一层的值隐蔽前一步骤生成的值;

（C）重复步骤(B)直到生成的值的个数等于或大于步骤（a）所需的密钥的个数。

3. 根据权利要求 1 或 2 所述的方法，其中步骤(d)包括组合从多个不同的种子值中导出的值。

4. 根据权利要求 1 所述的方法，其中步骤(d)包括利用多个不同的隐蔽函数中的每一个对多个种子值进行运算。

5. 根据权利要求 4 所述的方法，包括:

(I) 用一组不同的隐蔽函数中的每一个函数对至少一个根种子值进行运算，生成多个深一层的值;

(II) 用该组不同的隐蔽函数中的每一个函数对前一步骤生成的深一层的值或从其导出的值进行运算;

(III) 重复步骤（II），通过该重复或每个重复，生成值树中的下一层；

(IV) 在步骤（a）中，使用从步骤（III）中生成的一层或多层中的种子序列中导出的值作为密钥序列；

(V) 在步骤（c）中，将来自树体内的至少一个值传送给一个用户终端，传送给该用户终端的该值或每个值在树中的位置决定了用户用于解密数据单元的可用密钥序列部分的位置和范围。

6. 根据权利要求 5 所述的方法，包括，在步骤（I）中：

(i) 用所述组不同的隐蔽函数对多个不同的种子值进行运算

(ii) 对所述不同的隐蔽函数中的每一个，组合用一个隐蔽函数对种子值中的一个进行运算的结果和用同一个或另一个隐蔽函数对另一个种子值进行运算的结果，从而生成多个深一层的值。

7. 根据权利要求 3 所述的方法，其中步骤（d）包括:

(I) 组合分别从第 1 和第 2 隐蔽函数链中导出的第 1 和第 2 值，从而生成一个第 1 下一个种子或密钥，该第 1 和第 2 隐蔽函数链有各自不同的种子

(II) 组合从第 1 链中第 1 值的位置后面的一个位置导出的一个值和从第 2 链中的第 2 值的位置前面的一个位置导出的一个值，从而生成一个深一层的下一个种子或密钥值。

8. 根据权利要求 7 所述的方法，包括重复步骤（II）从而生成深一层的密钥值，在每一个重复中，来自第 1 链中的前一个位置后面的位置的值和第 2 链中的前一个位置前面的位置的值被组合。

9. 根据权利要求 1 所述的方法，其中所述种子值通过一个通信网络传送给所述用户终端。

10. 根据权利要求 9 所述的方法，其中所述种子值被从多个密钥管理节点传送给用户终端。

11. 一种加密发布的数据的方法，包括:

(a) 用一个或多个隐蔽函数对至少一个根种子值进行运算，从而生成

更多数量的深一层的值;

(b) 用一个或多个隐蔽函数对前一步骤生成的深一层的值或从其导出的值进行运算;

(c) 重复步骤（b），通过该重复或每个重复生成值树中的下一层;

(d) 用步骤（c）中生成的一层或多层中导出的一个密钥值序列加密多个数据单元。

12. 一种将数据传送给一组用户的方法，包括:

(a) 加密发布的多个数据单元;

(b) 系统地并独立于组成员变化地改变用于加密发布的多个数据单元的密钥;

(c) 将加密数据单元传送给所述用户;

(d) 在所述用户的终端解密数据单元，特征在于，从许多初始种子值生成更多个中间种子值并从该中间种子值导出多个用于加密发布的多个数据单元的密钥。

13. 根据权利要求 12 所述的方法，其中密钥序列的每个可能的子集都可以从相应的种子值组合中导出。

14. 根据权利要求 1，11 或 12 所述的方法，其中每个加密的数据单元承载一个未加密的索引号以便向各用户标识应该使用序列中的哪一个密钥来解密该单元数据。

15. 根据权利要求 1，11 或 12 所述的方法，其中以隐含地标识每个种子的顺序传送任何接收者构建用于整个密钥序列的一个特定子范围所需的种子。

16. 根据权利要求 1，11 或 12 所述的方法，其中多个数据发送者使用彼此相同的密钥序列来加密相同的或不同的数据单元。

17. 根据权利要求 1，11 或 12 所述的方法，其中从种子生成的序列中的每个密钥被作为一个中间密钥以与另一个中间密钥或密钥序列组合以生成一个用以加密或解密数据单元的密钥序列。

18. 一种发布数据的方法，包括用密钥序列加密多个数据单元；将加密过的数据单元传送给多个用户终端，其特征在于，根据一个密钥构造算法并利用一个或多个种子值生成该密钥序列并将其分配到数据单元，并且，该密钥构造算法的副本被发布给多个密钥管理器，以便在使用中，接收者可以从一个密钥管理器获得所述密钥序列中的用于访问数据的一个任意部分的一个任意密钥序列部分，而不必向任何数据发送者查询。

19. 一种操作用户终端的方法，包括:

a)接收多个用一个密钥序列加密过的数据单元;

b)接收一个或多个种子值;

c)从一个或多个种子值生成一个密钥序列，其密钥个数大于在步骤(b)中接收的种子的个数，所述序列构成了在步骤（a）中使用的密钥序列的一个具有上限和下限的任意部分，其中所述部分的上限和下限在序列中的位置由步骤（b）中接收的一个或多个种子值决定；以及

d)使用在步骤(c)中生成的密钥序列解密所述数据单元。

20. 一种密钥管理器，包括:

用于从数据发送者接收一个或多个初始种子值的装置;

用于从所述数据发送者接收多个应用数据单元的装置;

用于从所述数据发送者接收密钥构造算法的副本的装置;

用于使用所述算法以根据所述接收到的初始种子值产生密钥序列并利用所述密钥序列解密所述应用数据单元的装置;

用于接收一个来自用户终端的访问所述数据的一个任意部分的请求的装置；和

用于把所述种子值发送到所述用户终端以允许所述用户终端产生与所述密钥序列的一部分对应的密钥的装置。

21. 一种用户终端，包括:

用于接收用密钥序列加密的多个数据单元的装置;

用于接收一个或多个种子值的装置;

用于根据所述一个或多个种子值产生密钥序列的装置，所述密钥序列的密钥个数大于所述接收的种子值的个数，所述序列构成了所述用于加密所述多个数据单元的密钥序列的一个具有上限和下限的部分，其中所述部分的上限和下限在序列中的位置由所述接收的一个或多个种子值决定；和

用于使用所述产生的密钥序列解密所述应用数据单元的装置。

22.一种通信网络，包括:

数据发送者;

一个或多个根据权利要求 20 所述的密钥管理器，用于与所述数据发送者通信；和

一个或多个根据权利要求 21 所述的用户终端。

23.根据权利要求 22 所述的网络，其中使用一种多点传送或广播传输模式来发布数据。

24.根据权利要求 22 或 23 所述的网络，其中所述网络包括一个虚拟个人网络（VPN），并且其中用于构造用于解密数据的不同密钥子范围的不同种子组合给予虚拟个人网络的成员访问该 VPN 的不同期间的权利。

数据发布

技术领域

本发明涉及一种提供对数据的受控访问的数据发布系统。例如，根据用户的预付款，该系统可用于在一定的时期内提供对多点传送的声频或视频节目内容的访问。但是本发明并不限于用于多点传送数据分组网络，也可能用于，比如包括诸如 DVD（数字万用磁盘）的存储介质在内的其它的成批数据发布渠道。

背景技术

多点传送技术被开发用于因特网，它能够将数据高效地从数据源发布给大量的接收者。但是，现有的多点传送协议的高效性和可升级性部分地依赖于这一事实，即数据源不必有任何有关数据接收者的知识。这在希望在数据源和接收者之间建立保密关系，例如以便将诸如电视节目的流式视频数据仅传送给那些已为接收节目而预先付费的订购者时将存在问题。

一般地，可以通过在数据源对数据进行加密，然后控制用户对解密数据所需密钥的访问来建立这种保密关系。一个简单的方法是用一个单个话路密钥加密数据。该话路密钥一直保持不变直到一个新的用户希望访问该数据，或直到现有的用户之一被排除。此时，需要一个新的话路密钥并将其发布给所有的用户。虽然通过使用密钥的分级可以一定程度地改善该密钥发布方案的效率，在排除或加入一个特定的用户时在这种密钥的分级中只有一些需要改变，但是在新用户加入或离开时，该方案仍然有较大的传输开销。

在本发明共同待审的国际专利申请 PCT/GB98/03753(BT case: A25728/WO)中描述的另外一个方法中，数据在数据源被划分为一系列的应用数据单元（ADU）并且每个 ADU 使用一个不同的密钥。这些密钥由一个初始种子值系统地生成为

一个序列。该种子值也被传送给每个用户终端处的保密模块，该保密模块控制最终用户可否使用该密钥。

发明内容

根据本发明的第一方面，提供一个发布数据的方法，包括:

（a）对多个数据单元用密钥序列进行加密;

（b）将加密过的数据单元传送给多个用户终端;

（c）将至少一个种子值传送给一个用户终端;

（d）从该种子值生成一个密钥序列，其密钥个数大于传送给用户终端的种子值的个数，所述序列构成了在步骤（a）中产生的密钥序列的一个具有上限和下限的任意部分，其中所述部分的上限和下限在序列中的位置由步骤（c）中传送的至少一个种子值决定；以及

（e）在用户终端使用所述步骤（d）中产生的密钥序列解密数据单元。

本发明提供一个发布数据的方法，其中如在上面提到的共同待审的申请中揭示的系统中那样，用一个不同密钥的序列加密连续的数据单元。但是，本发明的方法提供进一步的显著的优点，其中每个用户可用的密钥序列的范围不是如在前一个系统中那样可能是无限的，而是被双重界定，也就是说，用户可用的密钥序列的开始和结束位置是事先确定的。如下面进一步说明的那样，数据发送者，或代表数据发送者而动作的密钥发布方，可以通过选择发送给用户的种子值来任意地确定用户可用的密钥序列的开始和结束点及长度。对密钥序列的任何所希望的部分都存在一个种子的集合，该集合提供对该部分并且仅对该部分的访问。通过给用户提供对双重界定的密钥集合的访问，而不是对一个无限制的密钥集合的访问，本发明使不必在每个用户终端安装由数据发送者控制的用于控制用户访问密钥的保密模块。

优选地在步骤（a）中使用的密钥序列如下生成:

（A）对一个或多个初始种子值进行运算，生成更多个中间种子值，该中间

种子值隐蔽该初始种子值;

（B）进一步对前一步骤生成的值进行运算，生成更多个深一层的值，该深一层的值隐蔽前一步骤生成的值;

（C）重复步骤（B）直到生成的值的个数等于或大于步骤（a）所需的密钥的个数。

优选地，该方法包括:

（i）用一组不同的隐蔽函数中的每一个函数对一个根种子值进行运算，生成多个深一层的值;

（ii） 用该组不同的隐蔽函数中的每一个函数对前一步骤生成的深一层的值进行运算;

（iii） 重复步骤（ii），通过该重复或每个重复，生成值树中的下一层;

（iv） 在步骤（a）中，使用步骤（ii）的最后一次重复中导出的值作为密钥序列。

（v）在步骤（c）中，将至少一个来自根种子值下面的树的值传送给一个用户，传送给该用户的值在树中的位置决定了用户用于解密数据单元的可用密钥序列部分的位置和范围。

隐蔽函数对输入值进行运算以生成输出值，其中即使已知该函数，也不易从输出值推导出输入值。隐蔽函数可以包括，比如，诸如MD5（消息摘要数5）的加密哈希函数。

发明人发现，一种系统地生成一系列的密钥同时可以对用户可用的序列部分的位置和范围进行简单控制的有效方法是用一组不同的隐蔽函数重复运算以生成一个值的树。在下面更详细的说明的例子中，使用了一个由一对对称的隐蔽函数形成的二叉树。一个函数是一个右转移位后跟一个哈希函数，另一个函数是左旋移位后跟一个哈希函数。本发明的本特征并不限于使用二叉树，也可以用三叉树或更高级树来实现。

优选地，在步骤（c）中，种子值被通过通信网络传送给用户终端，在本情

形下，种子值优选地从多个密钥管理节点被传送给用户终端，其中节点在不同地点连接到网络上。

本发明的优选的实施例的另一个优点是：提供对编码数据的访问的发布种子值的处理可以被移交给多个位于不同地点的密钥管理节点，这些节点的一些或全部远离数据源。这样建立的数据控制系统可灵活用于大量的接收者。

一般，数据单元和种子值通过相同的通信网络来发布。但是，这个并不必要。例如，可以将数据单元发布在诸如 DVD 的成批数据存储介质上，而随后通过因特网将种子值在线地发送给用户。很明显，这些例子仅用于说明，可以采用很多不同的实现方法。

优选地以隐含地标识每个种子的顺序传送任何接收者构建用于整个密钥序列的一个特定子范围所需的密钥的种子。在这种情况下，根据要求的最小和最大值和传送种子的预先确定的顺序的知识，推导出该种子的索引，而不用显式地列出每个种子的索引号。优选地每个加密的数据单元承载一个未加密的索引号以向各用户标识解密该数据单元应该使用序列中的哪一个密钥。

根据本发明的另一个方面，提供一种加密发布数据的方法，包括：

（a）用一个或多个隐蔽函数对至少一个根种子值进行运算，从而生成更多数量的深一层的值；

（b）用一个或多个隐蔽函数对前一步骤生成的深一层的值或从其导出的值进行运算；

（c）重复步骤（b），通过该重复或每个重复生成值树中的下一层；

（d）用由步骤（c）中生成的一个或多个推导出的密钥值序列加密多个数据单元。

根据本发明的另一个方面，提供一种将数据传送给一组用户的方法，包括：

（a）加密发布的多个数据单元；

（b）系统地并独立于组成员变化地改变用于加密发布的多个数据单元的密钥；

（c）将加密数据单元传送给所述用户；

（d）在所述用户的终端解密数据单元，特征在于，从许多初始种子值生成更多个中间种子值并从该中间种子值导出多个用于加密发布的多个数据单元的密钥。

根据本发明的另一个方面，提供一种发布数据的方法，包括：用密钥序列加密多个数据单元，将加密的数据单元传送给多个用户终端，其特征在于，根据一个密钥构造算法并利用一个或多个种子值生成该密钥序列并将其分配到数据单元中，并且，该密钥构造算法的副本被发布给多个密钥管理器，以便在使用中，接收者可以从一个密钥管理器获得所述密钥序列中的用于访问数据的一个任意部分的一个任意密钥序列部分，而不必向任何数据发送者查询。

本发明的这个方面提供一种数据发布的方法，其中密钥管理可以被移交给多个密钥管理节点，使系统可以被升级以包含大量的用户。

根据本发明的另一个方面，提供一种操作用户终端的方法，包括：

a)接收多个用一个密钥序列加密过的数据单元；

b)接收一个或多个种子值；

c)从一个或多个种子值生成一个密钥序列，其密钥个数大于在步骤(b)中接收的种子的个数，所述序列构成了在步骤（a）中使用的密钥序列的一个具有上限和下限的任意部分，其中所述部分的上限和下限在序列中的位置由步骤（b）中接收的一个或多个种子值决定；以及

使用在步骤(c)中生成的密钥序列解密所述数据单元。

根据本发明的另一个方面，提供一种密钥管理器，包括：

用于从数据发送者接收一个或多个初始种子值的装置；

用于从所述数据发送者接收多个应用数据单元的装置；

用于从所述数据发送者接收密钥构造算法的副本的装置；

用于使用所述算法以根据所述接收到的初始种子值产生密钥序列并利用所述密钥序列解密所述应用数据单元的装置；

用于接收一个来自用户终端的访问所述数据的一个任意部分的请求的装置；和

用于把所述种子值发送到所述用户终端以允许所述用户终端产生与所述密钥序列的一部分对应的密钥的装置。

根据本发明的另一个方面，提供一种用户终端，包括：

用于接收用密钥序列加密的多个数据单元的装置；

用于接收一个或多个种子值的装置；

用于根据所述一个或多个种子值产生密钥序列的装置，所述密钥序列的密钥个数大于所述接收的种子值的个数，所述序列构成了所述用于加密所述多个数据单元的密钥序列的一个具有上限和下限的部分，其中所述部分的上限和下限在序列中的位置由所述接收的一个或多个种子值决定；和

用于使用所述产生的密钥序列解密所述应用数据单元的装置。

根据本发明的另一个方面，提供一种通信网络，包括：

数据发送者；

一个或多个根据本发明的密钥管理器，用于与所述数据发送者通信；和

一个或多个根据本发明的用户终端。

附图说明

下面参照附图来详细说明实施本发明的系统，其中：

图 1 是一个实施本发明的网络的示意图；

图 2 表示图 1 所示网络中的一个用户终端的结构；

图 3 表示图 1 所示网络中的一个密钥管理节点的结构；

图 4 表示图 1 所示网络中的一个数据发送者的结构；

图 5 表示图 1 所示网络中传输的数据分组的格式；

图 6 表示通过密钥管理节点进行的密钥发布；

图 7 表示一个双向哈希链；

图 8 表示两个隐蔽函数的生成;

图 9 表示一个二叉哈希链树;

图 10 表示一个连续二叉哈希树;

图 11 表示一个混合二叉哈希链树的元素;

图 12 表示一个混合二叉哈希链树;

图 13 说明混合二叉哈希链树的构造;

图 14 表示一个二叉哈希链树的生长序列;

图 15 表示一个连续二叉哈希链树混合;

图 16 表示一个第二二叉哈希树的元素;

图 17 说明揭示和隐蔽 BHC-T 中的种子对;

图 18 说明揭示和隐蔽第二二叉哈希树中的种子;

图 19 表示一个多维密钥序列;

图 20a 到 20e 表示一个通用模型。

具体实施方式

一个数据通信系统包括一个通过数据通信网络 2 连接到多个用户终端 3 的数据服务器 1。虽然为了便于说明只示出几个用户终端，但在实际中数据服务器 1 可能与很多终端同时通信。在本例中，数据通信网络 2 是公共因特网并由通过节点 4 相互连接的多个子网络 2a-2c 组成。该子网络和相关联的路由器支持 IP（因特网协议）多点传送。

在本例中，数据服务器 1 是视频服务器。该数据服务器从一个海量存储设备 1a 读取一个视频数据流并用诸如 MPEG2 的合适的压缩算法压缩数据。然后数据服务器 1 中的保密模块将压缩的视频数据流分成应用数据单元（ADU）。例如，每个 ADU 可以包含对应于一分钟的视频信号的数据。然后利用一个加密算法用系统地生成的密钥序列对 ADU 加密，其中每个 ADU 用不同的密钥。合适的加密算法包括 DES(数据加密标准)（US 联合标准 FIPSPUB46）。这是一个传统的私有

的密钥算法。用于生成密钥序列的种子值也从数据服务器 1 传送给多个密钥管理节点。该密钥管理节点散布在数据通信网络中的不同位置。为了响应来自一个用户终端的请求，密钥管理节点将多个种子值传送给该终端。在传送种子值前相应的密钥管理节点可能实施检查，例如，确认相关的用户终端是否有权访问所请求的数据。例如，该用户可能有所请求的访问从视频数据服务器 1 多点传送的某一电影的权利。如果该电影是每次付费观看，该密钥管理节点负责检查该用户是否有该视频数据服务器的运营者的帐户和是否已为该电影适当地预付费。如果这些条件都满足，则该密钥管理节点将选择的允许用户生成密钥。该密钥对应于在数据服务器中用来加密组成该电影的 ADU 的密钥序列部分。如下的详细说明，用于生成密钥序列的算法使适当选择的种子值可用来提供对原始密钥序列的一个任意界定部分的访问。

图 2 表示一个用户终端 3 的主要的功能组成部分。网络接口 22 从或向数据通信网络 2 接收或发送 ADU。该 ADU 从接口 22 传递给访问模块 23。与访问模块 23 可能位于一个独立的保密模块（例如智能卡）中的以前的系统不同，在实施本发明的系统中，访问模块可以是一个在用户终端的主处理器中运行的软件模块。访问模块 23 包括一个解密模块 D，一个密钥生成模块 K 和一个种子存储器 SS。种子存储器存储从密钥管理节点接收的种子值并用一个密钥构造算法，如下面详细说明的那些算法，来处理那些种子值以生成一系列的密钥。该一系列密钥有由种子存储器 SS 保存的种子值决定的开始点和结束点。把本序列中的密钥顺序地传递给解密模块 D。解密模块 D 解密从接口 22 接收的一系列的 ADU，并将其传递给应用程序层模块 24。这将进行进一步的处理，例如用 MPEG2 解密算法，并将结果数据传递给一个输出设备，在本例中是一个视频显示单元 VDU 25。在优选的实现中，接口 22 可以通过 ISDN 调制解调器用硬件实施或通过 TCP-IP（传输控制协议-因特网协议）栈用软件实施。

用户终端可以用多种形式实施。例如，终端可以包括带有适当网络接口的个人计算机、智能移动电话、以及与电视机协力提供因特网访问的机顶盒。

图 3 表示图 1 的网络中的一个密钥管理节点的一个例子的结构。该节点通过 TCP-IP 栈将数据分组传送给数据发送者和用户终端或“接收者”。以传统的方式使用一个公共密钥加密算法通过加密套接字协议层(SSL)32 传送数据分组，密钥管理应用程序 33 以在后面进一步详细说明的方式从数据发送者接收种子值并将发布的种子值发送给用户终端。与密钥管理应用程序 33 关联的数据存储器 330 保存从该数据发送者或每个数据发送者接收的种子值。用户通过一个用户接口 34 与密钥管理应用程序相互作用，例如，用户接口 34 可以用 HTML（超文本链接标示语言）和 CGI 为用户终端提供网页。

图 4 表示图 1 的网络中的一个数据发送者的一个例子的结构。数据应用程序 41 输出数据，在本例中，数据包括一个被分为多个应用数据单元的 MPEG2 视频数据流。该视频节目内容从存储器 410 中导出。ADU 被传递给访问模块 42。它包括一个加密子模块、一个密钥生成子模块 K 和一个种子存储器 SS。子模块 K 协同诸如后面见一步详细说明的密钥构造算法使用随机生成的种子值来生成一个密钥序列。种子值也可以通过加密套接字层 43 和 TCP-IP 栈 44 从访问模块 42 输出。加密的 ADU 也通过 TCP-IP 栈 44 输出。

数据服务器和密钥管理节点都可以用商用的平台，如 COMPAQ Proliant™服务器或 Sun Microsystems Enterprise 5000™服务器来实现。

图 5 表示数据发送者输出的一个数据帧的格式。它包括一个承载加密 ADU 的数据有效负载、一个密钥索引 k_i 和一个话路标识符 SE。密钥索引和话路标识符按原样发送。

一般，密钥值可以通过与用于发布 ADU 相同的通信网络发布给用户终端。

在本说明书中用术语“应用数据单元”（ADU）来描述从保密或商业的观点来看有用的最小数据单元。ADU 的大小可能随着所需的应用和保密性变化。它可能是一个初始化帧和一个视频序列中的相关的“P-帧”集合，或者可能是对一个网络游戏的 10 分钟的访问。用来加密的 ADU 可能与一个应用的不同层使用的 ADU 不同。例如，在本例中 ADU 与 MPEG 压缩算法处理的视频帧的持续期间不同，

也与用户购买的个人节目项的持续期间不同。为了提高系统的性能，可能只部分地加密 ADU，其余的按原样发送。ADU 的大小可能在随内容而定的流的持续期间变化。如果在 15 分钟内有一百万个接收者加入一个多点传送数据流，应用数据单元的大小是系统可升级性的一个主要的决定因素，但 ADU 的大小也是 15 分钟，那么这将只需要一个重新分配密钥（re-key）事件。下面详细说明不同的密钥序列构造算法。

发送者去耦结构

本发明不限于使用图 1 的方法中的简单的时序。例如，本发明可应用于大规模的网络游戏。

在这种游戏中，ADU 的经济价值与时间或数据量无关，仅与一个完全的应用特定因素有关。在本例中，参加者按每“游戏分钟”付费，持续期间不是严格地与实际时间的分钟有关，而是由游戏时间保持器定义和发信号表示。该游戏包括很多虚拟地域，每个由不同的地域控制器调节。该地域控制器提供带给地域生命的后台事件和数据。它们将这个加密的数据传送给每个地域的一个多点发送地址，但是在所有的地域的任何时间使用相同的 ADU 索引和密钥。因此尽管被散布到多个多点传送地址，但是整个游戏是一个单一保密多点传送话路。游戏者只要有当前密钥就可以调入任何地域的后台数据。地域中的游戏者创建的前台事件不加密，但如果不参看后台数据它们将毫无意义。

图 6 表示这种游戏的数据流。只示出了与游戏安全性相关的数据流以及在游戏进程中，而不是在设立时，发送的数据流。所有的游戏者都在发送数据，但图中只示出加密的发送者，S-地域控制器。同样，只示出解密的接受者，R-游戏者。游戏控制器设立游戏保密性，其在图中未示出，在下面说明。密钥管理操作委派给多个复制的密钥管理器，KM，其使用保密 Web 服务器技术。

保密多点传送话路的密钥在一个序列中每游戏分钟（每 ADU）变化一次。所有的加密数据都以一个未加密的 ADU 索引开头，该索引指向用于解密的密钥。在设立阶段后，游戏控制器、地域控制器和密钥管理器保存有可以计算整个游

戏持续期间使用的密钥序列的初始种子。也可以使用分阶段的设立。

游戏设立

1. 游戏控制器（未示出）在验证身份的有效性后，单点传送一个共享的“控制话路密钥”给所有的 KM 和 S。所有 S 和所有 KM 运行保密 Web 服务器以便通过客户验证的加密套接字层（SSL）通信传送给每个 KM 和 S 用各公共密钥加密。游戏控制器还把它将要用于控制消息的多点传送地址通知给所有的 KM 和 S，这些 KM 和 S 要立即加入该地址。

2. 游戏控制器然后生成用于构造整个密钥序列的初始种子值并将其多点传送给所有的 KM 和所有 S，用控制话路密钥加密该消息并使用一个适于所涉及的大概是包含少量目标的可靠的多点传送协议。

3. 游戏以一个验证的话路目录声明来宣告，如在 Mark Handley（UCL）的“在升级的因特网多媒体会议系统”，博士论文（1997 年 11 月 14 日）中描述的那样定期地重复多点传送（未示出）。经验证的声明可以避免攻击者建立虚假付费服务器来收取游戏的收益。该声明协议被增强以包括密钥管理器地址和每游戏分钟的价格的详细信息。为了获得每游戏分钟的价格，密钥管理器监听该声明以及接收者。该声明还必须指明正在使用的密钥序列构造。

接收者话路设立，持续期间和终止

1. 从话路目录听到该游戏广告后，一个希望付费加入游戏的接收者用合适的形式联系一个 KM 网络服务器，请求一定数量的游戏分钟。这在图 6 中表示为“单点传送设立”。R 为所请求的游戏分钟付费给 KM，可能是给出她的信用卡的信息或以一些电子现金的形式或在先前游戏中赢得的代用币支付。作为回报，KM 将一组中间种子发送给 R 使其可以计算她购买的密钥序列的子范围。在下一节描述的该密钥序列的构造成为可能。所有这些都发生在加密套接字协议层（SSL）通信中，只有 KM 需要验证，而不是 R。

2. R 用她购买的中间种子生成相关的密钥。

3. R 加入由游戏应用程序确定的相关多点传送，其中的一个多点传送总

是来自一个S的加密的后台地域数据。R用前一步计算的密钥序列解密这些消息，这样使其余的游戏数据有意义。

4. 每当时间保持器发出表示一个新的游戏分钟的信号时（通过控制多点传送），所有的地域控制器递增其 ADU 索引并使用序列中的下一个密钥。 它们都使用相同的 ADU 索引。每个 R 注意到来自 S 的消息中的 ADU 索引被递增并使用序列中的下一个适当的密钥。

5. 当游戏分钟索引到达 R 所购买的序列的末尾时，应用程序在游戏者失去访问权之前给她发出一个“插入硬币”的警告。游戏分钟继续递增直到到达所需密钥超出 R 可计算的范围的位置。如果 R 没有购买更多的游戏分钟，她必须退出该游戏。

这种情形说明了只要密钥管理器通过一些预先安排知道每个 ADU 索引的经济价值或每个 ADU 的访问策略，发送者如何与所有接收者的加入和退出活动完全去耦。不必在密钥管理器和发送者之间进行任何通信。发送者不必监听任何接收者的活动。如果密钥管理器需要避免出售已经传输的 ADU，它们只需与来自发送者的变化的 ADU 序列号流保持同步。在本例中，密钥管理器通过监听多点传送数据本身来同步。在其他情形下，同步可能只简单地基于时间，可以是通过显式的同步信号也可以通过一天中的时间的同步来隐含地进行。在另外的情形下（例如，商用软件的多点传送发布），传输的时间可能无关。例如，可能是定期地重复传输，购买序列的一部分的密钥的接收者可以在以后的任何时间收听。

在本例中用预付来购买种子。这可以保证密钥管理器不保持关于他们的用户的状态。这意味着由于不需要中央状态库所以他们可以被无限地复制，否则将会是下列情形：如果种子是用帐户购买则需要检查用户的帐户状态。

下面说明密钥构造的不同方法。

密钥序列构造

在下面的所有密钥序列构造中，使用下列的符号：

● b（v）是表示用于隐蔽 v 值的函数的符号。即，对手不能通过有限的计算从 b(v)得到 v。一个隐蔽或单向函数的例子是诸如 MD5 哈希（IETF RFC1321）或标准加密哈希 1（NIST Sha-1）的哈希函数。好的哈希函数一般只需要较少的计算资源。哈希函数被设计为将任何大小的输入减小为固定大小的输出。在所有情况下，我们都将使用已经具有与输出相同大小的输入，只使用哈希的隐蔽属性，而不使用大小减小属性。

● b^h（v）表示重复应用于前一个结果的函数 b（），总共 h 次。

● r（v）是任何可快速计算的 1 对 1 函数，该函数将一组输入值映射到它自己。循环的（旋转的）位移位是这种函数的一个例子。

● c（$v_1, v_2, \ldots$）是一个函数，其组合数值 v_1, v_2 等，使得在给出结果和除一个运算数以外的所有运算数的情况下，剩余的运算数可以被简单地推导出来。也可以选择 c（）以便使如果运算数的位是独立的和无偏差的，则结果的位也将是独立的和无偏差的。XOR 函数是这种组合函数的一个简单例子。c（）理想上也应该是用来简单地推导出剩余运算数的函数，如 XOR 的情况下，为：v_1=c（c（$v_1, v_2, \ldots$），$v_2, \ldots$）。

在后面说明所有的构造的一个通用模型，但先用它自己的术语介绍每个方案更为清楚。

双向哈希链（BHC）

双向哈希链结构只被证明在一种有限的形式下是安全的，但是我们坚持对其进行描述。因为该有限版本形成了一个后面方案的基础。也可能有使用无限形式的情形：

1. 发送者随机地生成 2 个初始种子值，v(0, 0)&v(0, 1)。作为一个具体的例子，我们将假设这些值为 128 位宽。

2. 发送者决定所需的最大密钥序列的长度，H

3. 发送者对每个种子重复地应用相同的隐蔽函数以生成 2 个长度（H）相等的种子链。因而值为 v(0, 0)到 v(H-1, 0)和 v(0, 1)到 v(H-1, 1)。由于项 H-1

经常出现，简明起见，我们引入另一个常数 G=H-1。

这样形式上，$v(h,0)=b^h(v(0,0))$； $v(h,1)=b^h(v(0,1))$ (4.1.1)

4. 为了生成密钥 k_0，发送者组合链 0 的第一个种子 $v(0,0)$ 和链 1 的最后一个种子 $v(G,1)$ 的。

为了生成密钥 k_1，发送者组合链 0 的第二个种子 $v(1,0)$ 和链 1 的倒数第二个种子 $v(G-1,1)$，以此类推。

形式上，$k_h = c(v(h,0), v(G-h,1))$ (4.1.2)

严格地说，使用的流密码不需要 128b 密钥，因此可以通过截断该组合结果的最高（或最低）有效位导出一个较短的密钥，一般截至 64b。流密码的选择是不相关的只要它是快速和安全的。

5. 发送者开始多点传送该流，用 k_0 加密 ADU_0（应用数据单元 0），用 k_1 加密 ADU_1，以此类推。但至少保持 ADU 序列号为不加密。

6. 如果发送者委托密钥管理，它必须秘密地将 2 个初始种子值传送给密钥管理器。生成新的初始种子对并将其与用先前计算的密钥加密过的流式数据并行传送给密钥管理器。

接收者如下重新构造该序列的一部分:

1. 当接收者被授权访问 ADU_m 到 ADU_n 时，发送者（或密钥管理器）单点传送种子 $v(m,0)$ 和 $v(G-n,1)$ 给该接收者。

2. 该接收者通过对用（4.1.1）发送的种子重复地应用隐蔽函数来生成种子链 $v(m,0)$ 到 $v(n,0)$ 和 $v(G-n,1)$ 到 $v(G-m,1)$。

3. 接收者用与发送者相同的方式用（4.1.2）生成密钥 k_m 到 k_n。

但是，该接收者无法知道种子 $v(h,0)$（其中，$h<m$）或 $v(h,1)$（其中，$h>n$），除非对那些发送者已揭示的隐蔽种子之前的隐蔽种子进行穷尽搜索。因此，接收者无法计算超出 k_m 到 k_n 范围的密钥。

4. 通过发送该范围的界限处的相关种子，任何其它的接收者可以访问一个完全不同的 ADU 范围；“开始”种子来自第 1 链，“结束”种子来自第 2 链。

图 7 中深灰色背景的种子的范围表示对第 1 个提到的接收者隐蔽的种子。连向深灰色背景的密钥的引线也是对该接收者隐蔽的。

因此，通过对于每个话路每个接收者仅发送 2 个种子，每个接收者可以被授权访问任何邻接密钥范围。不幸的是，这种构造只能有限的使用，除非每个接收者可以被限制为在一个发送者序列中曾经只有一个密钥范围被揭示过。如果一个接收者被授权访问一个早先范围，然后另一个后来范围（比如 k_0 到 k_1 然后为 k_{G-1} 到 k_G），她可以计算在（k_0 到 k_G）之间的所有的值。这是因为种子 v(0,0)，v(G-1,1)，v(G-1,0)和 v(G,1)将必须被揭示，但仅 v(0,0)和 v(G,1)可以揭示整个序列。

一个可以绕过这个限制的方法是定期地用一个新的种子值重新开始第 2 个链（即保持 H 为低）并禁止一个接收者在 H 个 ADU 内进行两次访问。但是，这需要在密钥管理器上保存每个用户的状态。可能有一些适合本方案的合适的（niche）应用，例如用户只能扩充订购而不能退出然后恢复的商业模型。这种情况下，这将是一个非常有效的方案。

第二个可以绕过这个限制的方法是注意只有在两个最低限度的短链之间有间隙空间的情况下才可能有两个不相交的链。换言之，H<4 的链永远是安全的。这种短链似乎不大使用，但后面我们将用该特征从短 BHC 片段建立一个混合构造。

二叉哈希树（BHT）

二叉哈希树需要两个公知的隐蔽函数 $b_0()$ 和 $b_1()$。我们将它们称为“左”和“右”隐蔽函数。一般它们可以通过在单个隐蔽函数 b()前应用两个简单的 1 对 1 函数 $r_0()$ 和 $r_1()$ 中的一个，由单个隐蔽函数 b()来构造。如图 8 所示。

这样：

$b_0(s) = b(r_0(s))$；$b_1(s) = b(r_1(s))$

例如，第 1 个公知的隐蔽函数可以是一个一位左旋循环移位后跟一个 MD5 哈希函数，而第 2 个隐蔽函数可以是一个一位右旋循环移位后跟一个 MD5 哈希

函数。其它的可选方法可以是在一个隐蔽函数之前先与 1 进行 XOR 或连接一个公知的字。看来选择消耗处理器资源最小但消耗量一样的两个函数是有利的，因为这将在所有情形下平衡负载并限制对隐藏信道易感性，如果处理器负载水平将揭示出所选择的被执行函数就可能出现这种隐藏信道。另外，为了提高有效性，可以使用哈希函数的两个类型，例如具有两个不同的初始化向量的 MD5。但是，窜改试验算法似乎是不明智的。

密钥序列如下构造:

1. 发送者随机地生成一个初始种子值 $s(0,0)$，作为一个具体的例子，我们假定其值为 128 位宽。

2. 在需要一个新的初始值前，发送者决定需要的最大的树深度 D，该深度将导出一个最大密钥序列长度，$N_0 = 2^D$。

3. 发送者分别对该初始种子应用“左”和“右”隐蔽函数，生成两个 “左”和“右”第一层中间种子值:

$s(1,0) = b_0(s(0,0))$; $s(1,1) = b_1(s(0,0))$。

发送者生成四个第 2 层中间种子值:

$s(2,0) = b_0(s(1,0))$; $s(2,1) = b_1(s(1,0))$;

$s(2,2) = b_0(s(1,1))$; $s(2,3) = b_1(s(1,1))$,

以此类推，创建一个深度为 D 层的中间种子值的二叉树。

形式上，如果 $s_{d,i}$ 是一个比初始种子 $s_{0,0}$ 低 d 层的中间种子:

$$s_{d,i} = b_p(s_{(d-1),i/2}) \qquad (4.2.1)$$

其中 i 为偶数时 p = 0，i 为奇数时 p = 1。

4. 如前所述，通过从该种子值横穿树的叶子或通过它们的截断的推导值来构造该密钥序列。

即，如果 $D = 5, k_0 = s(5,0)$; $k_1 = s(5,1)$; ... $k_{31} = s(5,31)$。

形式上，$k_i = s_{D,i}$ (4.2.2)

5. 发送者开始多点传送该流，用 k_0 加密 ADU_0，用 k_1 加密 ADU_1，以此类推。

但至少保持 ADU 序列号为不加密。

6. 如果发送者委托密钥管理，它必须秘密地将初始种子值传送给密钥管理器。生成新的初始种子并将其与用先前计算的密钥加密过的流式数据并行地传送给密钥管理器。

接收者如下重新构造该序列的一部分:

1. 当一个接收者被授权访问从 ADU_m 到 ADU_n 时，发送者（或一个密钥管理器）将一个种子集合单点传送给该接收者（例如使用 SSL）。该集合由离树根最近的中间种子组成，这些中间种子使得能够计算需要的密钥范围而不能计算该范围之外的任何密钥。

这些通过检验最大和最小种子的索引 i 来识别，该检验基于下面的事实: 一个偶数索引总是一个“左”孩子，而一个奇数索引总是一个“右”孩子。从叶子开始往上，在树的每一层上进行检验。在移到上一层之前总是需要揭示一个“右”最小值或一个“左”最大值。如果一个种子被揭示，索引就向内移动一个种子，以便在移到上一层前，最小值和最大值总分别为偶数和奇数。为了向上移一层，如果需要，最小和最大索引被二等分并下舍入。这保证它们之间的差可预测地减 2。如前所述，对新索引重复该奇数/偶数检验，揭示“右”最小值或“左”最大值。继续该处理直到最小值和最大值交错或相遇。它们可能在其中一个或两个都向内移动后交错。它们可能在两个都向上移动后相遇，在该情形下，需要在结束该过程前揭示它们相遇处的种子。在附录 A 中以类似 C 的编码更形式上说明该过程。

2. 显然，每个接收者需要知道给予他的每个种子在树中的位置。这些种子和它们的索引在被揭示时可以被显式地成对。另外，为了减少所需的带宽，协议可以指明发送种子的顺序以便可以从最小和最大索引及种子的顺序隐含地计算出每个索引。这是可能的，因为只有一个允许重新创建任何一个密钥范围的最小种子集合。

每个接收者可以与发送者一样对这些中间种子重复相同的隐蔽函数对以重

新创建密钥序列，k_m到k_n（等式 4.2.1&4.2.2）。

3. 通过向其它的任何接收者，发送一个不同的中间种子集合，可以授权他访问一个完全不同的 ADU 的范围。

图 9 中示出了一个 D = 4 的密钥序列的创建过程。

作为一个例子，我们循环相关的中间种子，这些中间种子允许一个接收者重新创建从k_3到k_9的密钥序列。对接收者隐蔽的种子和密钥以灰色背景表示。当然，在实践中一般 D 的值大于 4。

注意到每一层可被分配一个任意的 d 值，只要它唯一地标识该层即可。什么也不依赖于 d 或 D 的实际值。因此发送者不必揭示树向上延伸多远，从而改善了保密性。

往往一个话路在开始时会有一个未知的持续期间。显然，D 的选择限制了从任何一个开始点开始的密钥序列的最大长度。最简单的作法(work-round)是：如果需要，生成一个新的初始种子并在旧树旁边开始一个新的二叉哈希树。如果所有的发送者和接收者都知道 D，各方都能立即发现超出最大密钥索引2^D的密钥范围。在这种情形下给每个新树分配一个“树-id”并与每个树的种子一起指明它是明智的。

另一个避免该上限的方法是使 D 是可变的，例如，$D = D_0 + f(i)$，而不是常数。图 10 中示出了这样一个连续的 BHT，其中$D_0 = 4$并且每 M 个密钥 D 加 1。在本例中，M 取固定值 7。但是增加该复杂性并没有太大意义，因为不同的树分支的公共种子只是沿着树的最右手边分支的种子$s_{d,2}d$。如果这些种子中的任何种子曾被揭示过则整个未来的树都将被揭示。因此，这个“改进”将永远不会在向用户揭示任意密钥范围时被用来提高效率，所有其能节省的只是发送者非常偶然地将普通的消息中的一个新初始种子传递给密钥管理器。相反，它引入一个安全缺陷，因为它创建一个“无限”值种子的集合，对此，任何数量的穷尽搜索都是值得做的。另一方面，如第一种作法那样必需定期生成一个新初始种子，对 BHT 的攻击脆弱性设置了一个上限。

二叉哈希链-树混合（BHC-T）

这种构造方法被称为混合是因为二叉哈希树（BHT）用只有两个种子长的双向哈希链（BHC）的片段构建。为了便于理解，我们将以从根到叶子的方向构建树以构造一个 BHC 片段来开始说明，如图 11 所示。这仅仅是为了便于理解，后面我们将推荐构建树的最佳方式是从边而不是从根开始。

1. 我们假设有两个随机生成的初始种子值 s(0, 0)和 s(0, 1)。作为一个具体例子，同样，我们假定它们的值为 128 位宽。

2. 现在我们对每个种子应用相同的隐蔽函数以生成两个隐蔽的种子 v(1, 0)和 v(1, 1)。

3. 为了生成子种子 s(1, 1)，我们将第 1 个种子 s(0, 0)与隐蔽的第 2 个种子 v(1, 1)组合。

为了生成子种子 s(1, 2)，我们将第 2 个种子 s(0, 1)与隐蔽的第 1 个种子 v(1, 0)组合。

4. 如果我们现在随机地生成一个第 3 初始种子 s(0, 2)并隐蔽其以生成 v(1, 2)，我们可以用相同的方式组合第 2 和第 3 初始种子和它们的相反隐蔽值以生成另外两个子种子 s(1, 3)和 s(1, 4)。这意味着每对父母种子生成 4 个孩子，当与其一边的兄弟（同父母）组合时生成两个，当与另一边的异父（母）兄弟组合时生成另外两个。结果，如果新的子种子象他们的父母那样被隐蔽组合，这种构造生成一个二叉树，因为种子的个数在每一次生成时翻倍。但是，该树只在种子的顶行（假定在该行创建了 2 个以上的初始种子）的中间之下分支。如果从顶部构建（参见后叙），树的边向内“萎缩”。

形式上，

$$s(d,i) = c(s(d-1,i/2),\ v(2d-1,\ i/2+1)) \quad i \text{ 为奇数}$$

$$= c(s(d-1,i/2),\ v(2d-1,\ i/2-1)) \quad i \text{ 为偶数} \quad (4.3.1)$$

其中，

$$v(h,j) = b(s((h-1)/2,j))。$$

图 11 a) 图示 BHC-T 混合的两对父母种子 <s(0,0),s(0,1) 和 <s(0,1),s(0,2)。其中的环表示每对公共的父母种子，外边的环中的外侧值落在图的边沿之外，因为我们只关注中央的父母种子 s(0,1) 的后代。图 11b) 示出相同的 3 个父母生成相同的 4 个孩子，但是为了更好地说明二叉树是如何形成的，从图中隐藏了被隐蔽的种子，就好象它们从来就没有通信过。在下面的图中带环的父母种子表示与上面的图中所示的相同的 3 个带环的种子。将序列延续到右边的两个点划箭头表示如果在右边还有另外一个父母，父母种子 s(0,2) 是如何生成另外两个孩子的。与每对箭头相交的点划线表示该线之上的父母组合以生成线之下的孩子。我们将在后面的图中用简化的形式表示这种构造。

图 12 表示一个示例混合树的一部分。正如二叉哈希树，用于加密连续的 ADU 的密钥是树的叶子上的种子的序列或它们的截断的推导值。图中示出如何通过把带环的种子揭示给接收者来揭示一个示例密钥范围，k3 到 k9。

为了说明如何从边而不是从根开始构建树，我们现在来说明本构造的进一步的扭曲（twist）。先前注意到选择 XOR 函数是因为如果两个运算数的 XOR 产生一个第 3 值，这 3 个值中的任何 2 个可以被 XOR 以产生该第 3 个值。这在图 13 中说明，其中所有种子的值都与图 11 中的相同。如果 s(0,1) 初始未知，而 s(0,0) 和 s(1,2) 已知，则由于下列“扭曲”属性可以推导出 s(0,1) 和 s(1,1)：

$$s(0,1) = c(s(1,2), b(s(0,0)))$$

$$s(1,1) = c(s(0,0), b(s(0,1)))$$

图 14 表示发送者如何从“边”开始构建 BHC-T 混合构造。种子创建的顺序用编号的圆圈表示。可以按任何顺序创建的种子都被分配有相同的编号后跟一个区分字母。与带环的节点相邻的较暗的圆圈表示必须随机地生成的种子。我们把这些种子称为初级种子。这些种子决定下一个带环的节点之前的所有的后续中间种子的值。

1. 发送者随机地生成种子 0 的 128 位的值。

2. 然后生成种子 1&2。它们形成 4 个种子构成的框的对角，因此通过“扭

曲”算法设定相反的对角值 3 和 4:

形式上,

$$s(d-1,\ i/2\ ) = c(s(d,i), v(2d-1,\ i/2\ +1)) \text{ 如果 i 为奇数}$$

$$= c(s(d,i), v(2d-1,\ i/2\ -1)) \text{ 如果 i 为偶数} \qquad (4.3.2)$$

其中,

$$v(h,j) = b(s((h-1)/2, j))$$

注意如果根种子的 d = 0，则 d 沿从叶子到根的方向成为递增的负数。

3. 然后必须生成种子 5，与 2 形成另一对对角。

4. 这通过等式（4.3.2）揭示相反的对角种子 6 和 7。

5. 种子 7 和 2 形成另外一个 4 种子框的上面的角，通过等式（4.3.1）设定种子 8a&8b。

6. 在种子 9 被随机地生成之后该图形以类似地方式继续。这种构造方法的一个优点是树可以无限制地生长—不必事先决定任何限制。

7. 发送者开始多点传送该流，用 k_0 加密 ADU_0，用 k_1 加密 ADU_1，以此类推。但至少保持 ADU 序列号为不加密。

即，$k_i = s(D,i)$ 其中 $D = 0$ (4.3.3)

8. 如果发送者委托密钥管理，它必须秘密地将初级种子传送给密钥管理器。生成新的初级种子对并将其与用先前计算的密钥加密过的流式数据并行传送给密钥管理器。

接收者如下重新构造序列的一部分:

1. 当一个接收者被授权访问从 ADU_m 到 ADU_n，发送者（或一个密钥管理器）将一个种子集合单点传送给该接收者。该集合由树中能计算所需密钥范围的中间种子的最小集合组成。

这些通过以与 BHT 类似但镜相的方式检验最大和最小种子的索引 i 来识别。在移到上一层之前总是需要揭示一个“左”最小值或一个“右”最大值。如果一个种子被揭示，索引就向内移一个种子，以便在移到上一层前，最小值和最

大值总分别为奇数和偶数。为了向上移一层，如果需要，最小和最大索引被二等分和并下舍入。这保证它们之间的差可预测地减小 1。对新索引重复该奇数/偶数检验。继续该处理直到最小值和最大值相隔二或三。如果它们相隔二，则它们与它们之间的种子一起被揭示。如果它们被三等分，如果最小值为偶数，则它们仅与它们之间的两个种子一起被揭示。如果最小值为奇数，则需要再向上移一层以便什么也不揭示并允许开始下一轮。在检验开始之前，对于被请求的范围的宽度已经小于 2 的情况检验异常初始条件。

在附录 B 中以类似 C 的编码更形式上说明该过程。

2. 显然，每个接收者需要知道其所给定的每个种子在树中的位置。这些种子和它们的索引在被揭示时可以被显式地成对。另外，为了减少所需的带宽，协议可以指明发送种子的顺序以便从最小和最大索引及种子的顺序隐含地计算出每个索引。例如，附录 B 中的算法对相同的密钥范围总是以相同的顺序揭示相同的种子。

3. 每个接收者可以与发送者一样对这些中间种子重复相同的隐蔽和组合函数对，以重新创建密钥序列，k_m到 k_n（等式 4.3.1，4.3.2&4.3.3）。

4. 通过向其它的任何接收者发送一个不同的中间种子集合，可以授权他访问一个完全不同的 ADU 范围。

由于 BHC-T 可以从边开始构建，它适合用于持续期间未知的话路。连续地随机生成新中间根种子限制了它对攻击的脆弱性，但允许序列的连续计算。为了进一步限制脆弱性，发送者可以延迟未来的种子的生成以拒绝任何接收者计算超出序列中的一个特定未来点的密钥。这将限制种子空间的强力搜索可用的时间。但是，从边开始构建树导致取决于每个新根种子（从而对该种子的攻击值）的密钥的个数成指数地增长。

根种子的值可以通过定期地递增被定义为叶子层的层，在每 M 个密钥（除了第一个外）的序列之后将其向离该根更近的方向移动一层，可以界定根种子的值。

形式上，这需要将等式(4.3.3)替换为：

$k_i = s(-i/M, i)$ $i<M$ 时

$k_i = s(1-i/M, i)$ $i = M$ 时 (4.3.4)

这在图 15 中用 M = 8 来说明。当然，在实践中 M 应该要大得多，以便保证不用触及树的左手边最上面的分支就能有效地描述所有合理长度的接收者话路。

前面我们注意到只有当 H<4 时，BHC 是固有保密的。在本链树混合中使用的 BHC 片段的 H=2，这保证了该混合方案的保密性。我们还建议一个二叉链树混合可以从长度为 3（H=3）的链片段构造而不牺牲保密性。在这种情形下，每个父母种子在与其兄弟或异父（母）兄弟配对时将产生 6 个孩子，使在每一层的树宽度 3 倍增长（一个三叉树 – BHC3-T）。这种构造方法如图 20e 所示，但更详细的分析留给将来的工作。它可能比 BHC-T 更为有效，但稍微复杂一点。

二叉哈希树 II（BHT2）

我们现在提出另一种基于二叉树的构造，其是以加强对强力攻击的安全性的方式组合 BHT 和 BHC-T 方法。我们使用相同的种子符号 $s_{d,i}$，但是在 d 的起源位于 BHT 的根的情况下，其值随着接近叶子而增大。图 16 中示出树的一个元素。与 BHT 的情形相同，我们在这种构造方法中使用两个隐蔽函数 $b_0()$ 和 $b_1()$，我们分别称之为“左”和“右”。

1. 我们假设有两个随机生成的初始种子值 s(0,0) 和 s(0,1)。作为一个具体例子，同样，我们假定它们的值为 128 位宽。

2. 发送者决定需要的最大树深度 D。

我们从每个初始种子产生 2 个隐蔽值，每个种子分别使用每一个隐蔽函数。

$v(1,0) = b_0(s(0,0))$； $v(1,1) = b_1(s(0,0))$；

$v(1,2) = b_0(s(0,1))$； $v(1,3) = b_1(s(0,1))$。

3. 为了生成子种子 s(1,1)，我们将两个左隐蔽种子 v(1,0) 和 v(1,2) 组合。为了生成子种子 s(1,2)，我们将两个右隐蔽种子 v(1,1) 和 v(1,3) 组合。

4. 如果我们现在随机地生成一个第 3 初始种子 s(0,2)，我们可以用相同的方式组合第 2 和第 3 初始种子以产生另外两个子种子 s(1,3)和 s(1,4)。与 BHC-T 混合同样，这意味着每个父母种子生成 2 个孩子以构建一个二叉树，但边沿向内“萎缩”。事实上，如果层 d 包含 n_d 个种子，则 $n_{(d+1)} = 2n_d-2$. 只要使用 2 个以上的初始种子，则该树将趋向一个二叉树。

形式上:

$$s(d,i) = c(v(2d-1,\ i/2),\ v(2d-1,\ i/2+1)) \quad i \text{ 为奇数}$$

$$= c(v(2d-1,\ i/2),\ v(2d-1,\ i/2-1)) \quad i \text{ 为偶数} \qquad (4.4.1)$$

其中，

$v(h,j) = b(s((h-1)/2,j))$。

5. 然后就用横穿树的叶子的种子值构造密钥序列。

形式上，$k_i = s_{D,i}$ (4.4.2)

6. 发送者开始多点传送该流，用 k_0 加密 ADU_0（应用数据单元 0），用 k_1 加密 ADU_1，以此类推。但至少保持 ADU 序列号为不加密。

图 16a)图示 BHT2 的两个父母种子对<s(0,0),s(0,1)和<s(0,1),s(0,2)。与用于说明 BHC-T 混合的方式完全相同，在图的 a)部分和 b)部分中，环表示每对公共的父母种子。与前述相同，图 16b)示出如何描述用 BHT2 构建的种子树。为简明起见，从外观上隐藏了被隐蔽的中间种子值。一旦这些内部值被隐藏，则所得的 BHT2 看上去与图 11 中的 BHC-T 混合相同。

用于计算要揭示一个密钥范围需揭示哪些种子的算法也与附录 B 中的 BHC-T 混合的算法相同，因此图 12 中带环的种子也是将 k_3 到 k_9 揭示给一个特定的接收者。

横穿一个从 3 个初始种子（在层 0）构建的深度为 D 的 BHT2 的叶子的密钥的最大个数为 2^D+2。如果需要一个连续的树，密钥可以被定义为从中间种子层下行而不是停留在横穿它们的层，与图 10 中所示的连续 BHT 类似。

我们已经在(4.4.1)中示出如何只用 4 个隐蔽值的两个组合值来构建一个二

叉树。

每次取 4 个值中的 2 个，得到 6 个可能的组合：

c1 = c(v(1, 0), v(1, 1))

c2 = c(v(1, 2), v(1, 3))

c3 = c(v(1, 0), v(1, 2))

c4 = c(v(1, 1), v(1, 3))

c5 = c(v(1, 0), v(1, 3))

c6 = c(v(1, 1), v(1, 2))

c1 和 c2 每个只依赖于一个父母种子。因此，仅揭示父母就会揭示一个孩子，防止二者都被使用。另外，c6 = c(c3, c4, c5)和 c5 = c(c3, c4, c6)，以此类推。因此揭示这些组合中的任意 3 个就隐含地揭示第 4 个。但是，可以使用任意 3 个组合而不是象 BHT2 中那样仅使用 2 个。对所得到的三叉树（BHT3）的分析留给将来的工作。

通用模型

我们已经提出 4 种密钥序列构造，现在提出一个通用模型，其允许所有这些方案和其它类似的方案可以用相同的术语来描述。

我们定义两个坐标平面

● 一个具有位于坐标（h, j）的离散值 v 的“隐蔽”平面，这样，通常在一个 h 坐标的值被隐蔽以产生 h+1 处的值，具体的映射关系依赖于该方案。

● 一个具有位于坐标（d, i）的离散值 s 的“组合”平面，这些值是以同样依赖于该方案的方式组合来自隐蔽平面的值而得到的结果。

每种构造都由在隐蔽平面上的基本数学“分子”构建。图 20a 到 20e 将这些分子表示为一组粗黑箭头，表示从 h = 0 的轴开始，从一个值 v 映射到下一个值的隐蔽函数，为了示出该构造如何在 j 轴方向生长，粗的浅灰色的箭头表示对完成下一个分子的相邻值的隐蔽。一个分子由三个常数定义：

● H，一个分子沿隐蔽平面的 h 轴的高度

● P，在一个分子中使用的隐蔽函数的个数

● Q，为了产生组合平面中的每个值，组合来自隐蔽平面上的每个分子所得到的值的个数。

隐蔽平面中的一个分子的初始值 v 直接从组合平面上的先前值 s 映射（在图 20a 到 20e 中以点划线表示）：

如果 h mod H = 0; $v(h, j) = s(h/H, j)$ (4.5.1).

隐蔽平面分子中的后续值从先前值隐蔽（用粗箭头表示）：

如果 h mod H 0; $v(h, j) = b_p(v((h-1), j/P))$ (4.5.2).

其中，p = j mod P

然后，隐蔽平面分子中的最终结果值被组合以产生组合平面中的下一些值（用细线表示）：

$s(d, i) = c(v(h_0, j_0), \ldots v(h_q, j_q), \ldots v(h_{(Q-1)}, j_{(Q-1)})$ (4.5.3).

其中 h_q 和 j_q 被定义为每个构造的参数 q 的函数。

因此，沿隐蔽平面的 h 轴的每 H 大小，组合平面中的 d 递增 1。

表 1 给出 H，P 和 Q 的值和定义每个构造的 h_q 和 j_q 的公式。它还参照说明使用本通用模型的各构造的附图。

	BHT2	BHT	BHC	BHC-T	BHC3-T
图	20a	20b	20c	20d	20e
H	2	2	H	2	3
P	2	2	1	1	1
Q	2	1	2	2	2
h_q	Hd - 1		H(d-1) + q(H-1) +(1-2q)(i mod H)		
j_q	i - 1 + 2q	I	i/H + q		

表 1- 定义每个密序列构造的通用模型的系数

在所有情形下，除非需要一个连续的构造，密钥由下式定义的序列构建：

$k_i = s(D, i)$ (4.5.4)

其中 $D = \log(N_0)$

其中 N_0 是需要的密钥的最大个数

存储和处理之间的折衷

在所有的 MARKS 构造中，在发送者加密前和在接收者解密前，都用少量的种子生成较多的密钥。在任一情形下，可能用于存储密钥序列的存储容量有限，密钥序列所需的存储量相对于种子而言成指数增长。在这种情形下，可以计算前面几个密钥而存储允许计算其余密钥的种子。通常，不是节省存储容量就是避免相同计算的重复，这取决于存储或处理时间中哪个最紧缺。

利用 BHC，在计算第 1 密钥前，必须遍历整个反向链。但不必存储每个值。可能存储中间点，4 分之 3 点等处的值而其余的被丢弃。由于该序列回退进（eat back into）该反向链，所以下一个值总可以通过从先前存储的值再次运行哈希链，根据需求在途中存储更多的值来重新计算。

利用所有的树构造，需要在话路开始前计算任何在通向第 1 密钥的树的分支下的中间种子，但同样它们不必被全部存储。离叶子最近的中间种子应被存储（高速缓存），因为它们将最先用来计算下几个密钥。当需要离根较近的中间种子时，只要从未丢弃密钥管理器原始发送的种子就可以重新计算它们。

效率

正如上面提到的，H3 的 BHC 绝对有效但是不安全。因此我们将把讨论局限于已经详细地分析过的基于二叉树的构造。表 2 表示 BHT， BHC-T 和 BHT2 的每个保密多点传送话路的各种参数，其中：

R，S 和 KM 分别是接收者，发送者和密钥管理器，如前面的定义，N（=n-m+1）是接收者需要的随机位于密钥空间中的密钥范围的长度，W_s是种子的大小（一般为 128b），W_h是协议头开销的大小； t_s是隐蔽一个种子（加上一个相对可以忽略的循环移位和／或组合运算）所用的处理器时间。

			BHT	BHC-T	BHT2
每个R	(单点传送消息大小)/W_s-W_h或(最小存储量)/W_s	最小	1	3	3
		最大	2(log(N+2)-1)	2logN	2logN
		平均	0(log(N)-1)	0(log(N))	0(log(N))
每个R	(处理等待时间)/t_s	最小	0	0	0
		最大	log(N)	2(log(N)-1)	4(log(N)-1)
		平均	0(log(N)/2)	0(log(N)-1)	0(2(log(N)-1))
每个R或S	(每个密钥的处理)/t_s	最小	1	2	4
		最大	log(N)	2(log(N)-1)	4(log(N)-1)
		平均	2	4	8
每个S或KM	(最小存储量)/W_s		1	3	3
每个S	(最小随机位数)/W_s				

表 2- BHT，BHC-T 和 BHT2 的每个保密多点传送话路的各种参数

如表中所示，用于每个接收者的话路设立的单点传送消息的大小与每个接收者需要的最小存储量相等。只有当接收者如上所述为了处理而选择折衷存储时才是这样。为最小发送者存储行使用相同的折衷。处理等待时间是指接收者在接收到其话路的单点传送设立消息后准备对输入数据的解密所需的时间。应注意在别的成员加入或退出时没有等待时间开销，就如同在适用于非计划逐出的方案中一样。应注意每个密钥的处理数字假定密钥的连续访问。在这种情形下，在任何话路中要存储的最有效值（除了构造树所需揭示的最小集合外）是从根到当前使用密钥的分支上的那些值。每个密钥的平均处理就是整个树中的哈希运算的次数除以叶子上的密钥的个数。只有发送者（或者有多个发送者时为一个组控制器）被要求为初始种子生成随机位。需要的位数显然等于这些初始种子的最小发送者存储量。

可以看到：只有依赖于组成员数的大小的参数是关于每个接收者的参数。

两个这样的参数（存储量和处理等待时间）的开销被发布给组成员，因此对每个接收者是常数。只有单点传送消息的大小会在密钥管理器处造成随组成员的大小线性增长的开销，但是该开销在每个接收者话路中只负担一次。当然，没有一个每个接受者的开销自身是依赖于组的大小的，就如同在所有允许非计划逐出的方案中一样。因此，提出的所有的构造都是高度可升级性的。

相互比较这些方案，可能令人惊讶，混合 BHC-T 和 BHT2 在发送消息方面的效率与 BHT 非常相近。它们都是只要求每个接收者话路设立消息中多一个种子的这样一个平均水平。如果 N 较大，与每个接收者话路需要的密钥的个数相比较这是微不足道的。平均来讲，BHC-T 需要的处理时间是 BHT 的两倍，BHT2 需要的处理时间是 BHT 的四倍。我们将看到保密性的提高使这些开销是值当的。

BHT

使用 BHT，树中的每个种子在价值上是其孩子的两倍。因此，倾向于穷尽搜索种子空间以找到对树中当前已知的最高种子值隐蔽的正确值。对 MD5 哈希来说，这平均包括 2^{127} 次 MD5 运算。可能会找到一个值，其不正确但是对一个与已知值冲突的值隐蔽（一般用 MD5 每 2^{64} 次运算找到一个）。这只有通过使用种子以产生一个密钥范围并对据推测用其加密的一些数据进行检验时比较明显。在成功破解一层之后，下一层又将在价值上为两倍，但将需要相同的强力来分裂。应注意在 Sun SPARCserver-1000 上一个 128b 输入的 MD5 哈希（可移植源）大约用 4us。因此，2^{128} 次 MD5 运算将需要 4e25 年。

BHC-T

使用 BHC-T 混合，抗攻击的强度取决于攻击发生在哪个方向。如果我们取 BHC-T 的单个元素，它有 4 个种子值-有 2 个父母和 2 个孩子，如表 3 中所示并在图 13 中说明。给出 4 个值中的任何 1 个，都不能计算出其它的值，因为没有足够的信息来检验正确性。给出 4 个值中的 3 个，只用一个隐蔽运算就可以计算出第 4 个。只给出 4 个值中的 2 个，表中列出了计算其它 2 个值的难度，这取决于给出的是哪 2 个值。字母“i”表示一个输入值，单元格中的值表示在给

定输入的情况下确保找到输出值对所必需的隐蔽函数运算的次数。w 是数空间中的位数（MD5 为 128）。图 13 中图示出相同的信息，输入值上带圈，隐蔽值的背景为灰色。

父母	s(0, 0)	i	$1+2^{(w+1)}$	i	$1+2^w$	i	2
	s(0, 1)	i		$1+2^w$	i	2	i
孩子	s(1, 1)	2	i	i	$1+2^w$		i
	s(1, 2)		i	$1+2^w$	i	i	2

表 3-揭示和隐蔽 BHC-T 中的种子对

如果给出在“方块”的一个边上的一个父母和一个孩子，则可以穷尽地搜索对面的父母通过把每个位隐蔽并与 2 个给定值的 XOR 比较来检验每个值。因此对“边路”（sideways）攻击可以保证在 2^w 次隐蔽运算之后能够成功。

如果只给出 2 个孩子值，对父母中的一个进行穷尽搜索稍稍有点麻烦。即，推测一个父母值 s(0, 1)，仅当下式成立时该值是正确的:

$$c(s(0,1), b(c(s(1,1), b(s(0,1))))) = s(1,2)$$

因此，对“向上”攻击可以保证在 $2^{(w+1)}$ 次隐蔽运算之后能够成功。

在这种构造中，找到与 2 个给定值相适合但也不是正确的值对（一个双重冲突）的 2 个未知值的可能性较小。如果这种对确实出现，只能通过用它们产生密钥并用加密数据检验该密钥来检验它们。因此，双重冲突的较小概率可以稍微地减少攻击任务的复杂性。

边路攻击只能在与已知的最高种子相同的层最多取得一个种子。由于在右边的下一个“框”中只知道一个值，对右侧的攻击会在一个偶数索引的孩子处结束。同样地，对左边的攻击被一个奇数索引的孩子阻断。向上的攻击是所剩的唯一的选择。一个成功的向上攻击不给出任何多余的密钥，但是当后跟一个边路攻击时会揭示上一个边路攻击的密钥的两倍之多。

BHT2

BHT2 抗攻击的强度采用与 BHC-T 混合相似的形式，只是设计的抗向上攻击的强度要大得多。与 BHC-T 同样，组成方块的 4 个值中只知道一个时不能揭示

任何其它的值。但是，与 BHC-T 不同，3 个值不一定能立即给出第 4 个。如果只有一个父母未知，需要 2^w 次隐蔽运算来保证找到它。只给出其中的 2 个值时，表 4 列出了计算另外 2 个值的难度，这取决于给出的是哪 2 个值。如前所述，表格中的值表示在给定的输入时保证找到输出值对所必需的隐蔽函数运算的次数。图 18 中图示出相同的信息，输入值上带圈，隐蔽值的背景为灰色。

父母	s (0, 0)	i	2^{2w}	i	$3+2^w$	i	$3+2^w$
	s (0, 1)	i		$3+2^w$	i	$3+2^w$	I
孩子	s (1, 1)	4	i	i	$3+2^w$		I
	s (1, 2)		i	$3+2^w$	i	i	$3+2^w$

表 4-揭示和隐蔽 BHT2 中的种子对

如果给出一个父母和在“方块”的任一个边上的一个孩子，则可以穷尽地搜索对面的父母通过把每个值隐蔽并与 2 个已知值的 XOR 比较来检验每个值。因此对“边路”攻击可以保证在 2^w 次隐蔽运算之后能够成功。给出一个父母和对面的一个孩子的情况下也一样。

如果只给出 2 个孩子的值，在 BHT2 中对父母值的穷尽搜索要麻烦得多。对每个右边父母值 s (0, 1) 的推测值，它必须是左隐蔽的，然后左父母值必须被穷尽地搜索以找到一个左隐蔽的值，将其与第 1 左隐蔽的推测值组合以给出该左孩子的给定值。但是，当这 2 个父母推测值是右隐蔽时，他们不可能被组合而给出正确的右孩子。因此，右父母的下一个推测值必须与一个左父母的隐蔽值的穷尽搜索组合，如此类推。这相当只给出 s (1, 1) 和 s (1, 2)，要解下列联立方程：

$$c(b_0(s(0,0)), b_0(s(0,1))) = s(1,1)$$

$$c(b_1(s(0,0)), b_1(s(0,1))) = s(1,2)$$

为了保证成功为此需要穷尽搜索 2 个父母的组合的方阵，即 2^{2w} 次隐蔽运算。在从孩子到父母的方向上的较大的抗强力攻击的强度在图中以较深的灰色背景显示。另外一种方法是存储一个父母的所有的左隐蔽值和右隐蔽值以避免重新计算它们。但是只有一个父母的每个可能值的未被索引的左隐蔽值会消耗多于

5e27TB 的存储量，其开销使其他的攻击方法在经济上要合算得多。

有关双重冲突的评述与 BHC-T 一样适用于 BHT2，只是错误的值对只可能在 4 个哈希冲突同时地偶然发现时出现，该情况出现的概率非常小。

同 BHC-T 中同样，BHT2 中的边路攻击被限制在最多一个“框”中。因此，为了获得任何有效个数的密钥，马上就必须面临向上攻击。一个边路攻击的 2^w 次隐蔽运算可能比合法地获得被攻击的密钥更昂贵。一旦必须面临向上攻击，2^w 的隐蔽运算次数确实会导致寻找别的方法。

一般安全性

通常，构建一个树需要的随机值越多，它能包含的在由每个新随机种子生成的子树的界限内的持续攻击就更多。但是，正如上面关于连续树的讨论中所述的那样，对于较长的话路，在安全性和连续的密钥空间的方便性之间有一个折衷。随机生成的种子的随机性是另一个须被正确设计的潜在缺陷领域。

所有 MARKS 构造在合法组成员勾结时都是脆弱的。如果一个成员的子组在他们内部同意每个人购买一个不同范围的密钥空间，他们可以共享接收到的种子从而他们都能够访问他们的本来是单独的密钥空间的并集。套利是已讨论过的成员勾结的一个变种。这种情况是一个组成员购买整个密钥序列并将各部分以比出售价低的价格出卖，如果大部分密钥由超过一个的用户购买仍能从中牟利。用无组成员防止勾结的保护方法在后面关于“水印”的部分中讨论。

最后，对任何特定的应用，整个系统的安全性显然依赖于设立话路时使用的安全的强度。上面的例子情形描述需解决的问题并推荐标准的加密技术以满足它们。使用任何 MARKS 构造的应用的总体的安全性总是与最弱的部分的强度一样。

当前工作中讨论的密钥管理方案适用于与其它的机构进行模块组合以满足下述的其他的商业需求。

多发送者的多点传送

只要所有的发送者使用相同的密钥序列，多发送者的多点传送话路可以通

过使用 MARKS 构造来保密。它们不必同时使用相同的密钥，只要它们使用的密钥都是同一个序列的一部分即可。只要将 ADU 索引作为加密的 ADU 的头来不加密地传输，即使每个发送者都在别人的序列之外，接收者也知道使用哪一个密钥。如果对应用的商业模型很重要的话，前面描述的例子情形描述了多个发送者如何同步它们使用的 ADU 索引。

如果多发送者的多点传送中的每个发送者使用不同的密钥或密钥序列，每个发送者创建一个不同的保密多点传送话路即使他们都使用相同的多点传送地址。这由多点传送话路和前面定义的保密多点传送话路之间的区别可以推出。在这种情形下，每个保密多点传送话路必须被独立于其它的话路创建和维护。但是，还有一些被称为分期偿还初始化的范围。即，不同的保密多点传送话路都可以使用相同的设立数据以节省消息发送。例如，商业模型可能是客户总是必须从一组有关的发送者中的每一个购买相同的 ADU，如果他们从每个发送者购买的话。在这种情形下，每个发送者可以将所有发送者共同的密钥的 MARKS 序列和一个该发送者特有的长期密钥组合起来。客户可以购买公共范围密钥的相关的种子，然后购买她希望解密的每个发送者的一个附加的长期密钥。

非连续和多连续密钥的访问

MARKS 构造被设计为在授权每个接收者访问一个较宽的序列的一个任意子范围的密钥序列时有效，但不适用于数据不连续或需要一个序列的几个任意不连接的部分的情况。因此 MARKS 的目标是在一维上自然连续的数据流，例如实时的多媒体流等。

但是，一旦接收者访问过一个范围的密钥，显然不能强迫以连续的顺序访问它们。例如，接收者可能存储音乐流的一个子范围，该音乐流用 MARKS 密钥序列中的一个加密并通过因特网多点传送。使用下载的音轨的索引，接收者以后可以使用从 MARKS 序列中无顺序地取出的相关密钥，以随机的顺序挑选想听的音轨。

MARKS 还可以用来限制对连续但是多维的数据的访问。在 M. Fuchs, C. Diot,

T. Turletti, M. Hoffman 的“一种 ALF 设计的命名方法”（“a Naming Approach for ALF Design”）中描述了这种应用的一些例子。该文章发表在伦敦的 HIPPARCH 学术讨论会的会议录上（1998 年 6 月）。图 19 中示出一个 2 维密钥序列空间。

例如，对多点传送的股票报价的访问可以按订购的期间和按订购的期货市场的范围出售。每个报价需要用 2 个中间密钥的 XOR 结果值来加密。因此实际上用于加密的“最终密钥”是:

$k_{1,j} = c(k'_{0,i}, k'_{1,j})$.

一个中间密钥将来自一个序列 $k'_{0,i}$，其中 i 每分钟递增一次。另一个中间密钥可能来自一个序列 $k'_{1,j}$，其中 j 表示期货报价的月数。一个研究 1 到 2 年期货的商人将不仅根据其想订购的期间购买相关的 $k'_{0,i}$ 的子范围，而且会购买中间密钥 $k'_{1,12}$ 到 $k'_{1,24}$ 的范围。

诸如海法加密（1997 年 1 月）中的 Ross Anderson& Charalampos Manifavas(剑桥大学)的“变色龙--种新的流密码”（“Chameleon – A New Kind of stream Cipher ”）中的方法（上面已经描述过）可以用来给用于解密数据流的密钥加水印。因此，由任何 MARKS 构造生成的密钥都被视为中间密钥。发送者通过组合每个中间密钥和前面描述的一个长期密钥块（具体例子中是 512kB）创建一个最终密钥的序列。每个接收者被给定一个相同块的长期的加水印的版本以从其中间密钥序列产生一个加水印的最终密钥的序列，从而强制水印解密。

但是，这种方法具有变色龙方法的一个共同缺陷。它为由不忠的授权接收者传递到完全无授权的接收者（该接收者没有长期密钥块）的任何密钥经过的密钥或数据创建一个审计跟踪。在这种情形下，如果跟踪密钥或数据，可以跟踪到揭示密钥或数据的叛徒。但是，可以把中间密钥，而不是最终密钥传送给任何有时具有仍然有效的长期密钥块的接收者。因此一个未被授权访问特定中间密钥（其未加水印）的接收者可以用其自己的密钥块创建加水印的最终密钥，从而解密加密流。虽然产生的密钥和数据都加盖有她自己的水印，这只是将审

计跟踪交给泄露的目标，而不是源。因此，对这种类型的“内部”叛徒只有很小的威慑力。

返回到 MARKS 构造的特定情形，变色龙的这个一般缺陷意味着可以内部传递中间种子或中间密钥而不必害怕审计跟踪。例如，在上面的网络游戏的例子中，一组玩游戏者可以勾结起来，每人买一个不同的游戏小时并在他们之间共享他们每人购买的中间种子。为了产生真正的密钥，每个玩游戏者可以使用她自己的有水印的长期密钥块，她需要该密钥块来玩游戏。没有创建用于跟踪谁传递了无水印的中间种子的审计跟踪。但是，如果任何玩游戏者试图将带水印的密钥或数据传送给最近没有玩游戏因此没有其自己的有效的长期密钥块的人，有审计跟踪。类似地，因为长期密钥块也包括一个可跟踪叛徒接收者的水印，如果一个玩游戏者传送他们的长期密钥块时，也有审计跟踪。

非计划驱逐

正如已经指出的那样，MARKS 构造允许随时从组中驱逐成员，但是只能在每个接收者话路设立时已计划过。如果预先计划的驱逐是一般的情形，但偶而也需要非计划的驱逐，任何 MARKS 方案可以与其它的方案，如 LKH++,组合以允许偶然的非计划驱逐。为了实现这个目标，与上述水印一样，由任何 MARKS 构造生成的密钥序列都被视为中间密钥。这些与一个例如用 LKH++发布的组密钥组合（如进行 XOR）以生成一个用于解密数据流的最终密钥。因此要在任何时间生成该最终密钥需要 MARKS 中间密钥和 LKH++中间密钥二者。

当然，可以组合（如进行 XOR）任何个数的中间密钥以同时满足多种需求。例如，MARKS，LKH++和变色龙中间密钥可以被组合以便同时实现低成本的计划驱逐，偶然的非计划驱逐和防止泄露到长期组外的带水印的审计跟踪。

形式上，最终密钥，$k_{i,j\ldots} = c(k'_{0,i}, k'_{1,j})$

其中中间密钥 k'可以用 MARKS 构造或在关于加水印和多维密钥序列的前两节中描述的其它的方法来生成。

通常，以这种方式组合产生一个综合方案，其存储成本是各单独成员方案

的总和。但是，组合 LKH++和 MARKS，其中大部分驱逐是计划的，去掉了 LKH++的所有重新分配密钥的消息，除非确实需要非计划的驱逐。

本发明不限于用于多点传送数据网络。下面通过举例来说明另外两个使用领域。

虚拟个人网络（VPN）

通过建立一个 VPN，一个大公司允许其员工或定约人从因特网上的任何地方与公司的其他部分通信。实现其的一个方法是给每个工人该整个公司使用的一个组密钥。结果，每当一个工人加入或离开公司时，都必须改变该组密钥。作为替换方法，无论工人加入或离开与否，该密钥应该在一个 MARKS 构造决定的序列中定期地变化。当建立每个新雇佣合同时，分配给每个员工允许其计算序列中的下一个值的种子直到其合同需要续约。提前离开的工人被视为一个非计划驱逐。

数字万用盘（DVD）

DVD 原来表示数字化视频光盘，因为其容量适合于该介质。但是，它可用来存储诸如软件或声频等要求较少存储量的内容。代替在每次选择声频音轨或软件主题时，压制一个不同的稀疏地填充的 DVD，通过使用本发明，用数百个相关的音轨或主题填充其容量来生产每个 DVD。每个音轨或主题构成一个 ADU。每个 ADU 可以用使用一种 MARKS 构造生成的序列中的一个不同的密钥来加密。这些 DVD 可以被大量生产并免费分发（如作为杂志上的封面光盘）。任何拥有这些 DVD 中的一个的人然后可以通过因特网购买种子，这将给出一个范围的访问密钥以解密 DVD 上的一些 ADU。MARKS 适用于这种情形，因为一旦 DVD 被压制加密密钥就不能再改变，所以使用物理介质的商业模型不倾向依赖于非计划驱逐。本方案可用于与变色龙组合以对密钥和数据加水印。

上面我们描述了管理很大的组的密钥的方案，通过将发送者与所有接收者的加入和离开活动完全去耦，保持接收者启动的因特网多点传送的可升级性。发送者也完全与承担该接收者的活动费用的密钥管理器去耦。我们已经示出许

多商业应用具有只需要无中央控制的密钥管理器的模型，在这种情形下，可能有无限的密钥管理器的副本。当复制的无中央控制的密钥管理器中的一个失败时，不会影响其姐妹服务器中的进程的处理，从而使整个系统与问题隔离，提高恢复性。为了说明这些点我们提出了一个按每分钟收费的大规模网络游戏的工作例子。

通过使用系统地改变组密钥而不是由用户的加入或离开活动驱动重新分配密钥来实现上述的优点。去耦的实现是通过发送者和密钥管理器事先安排多点传送数据流中的经济值的单元（关于付费的“应用数据单元”）。通过递增数据中声明的 ADU 索引通知系统的密钥改变。使用这种模型，当一个接收者加入或离开时对其他发送者和接收者都没有副作用。我们还确保多点传送不用于密钥管理，只用于数据传送。因此，重新分配密钥不易受随机传输失败的影响，当使用多点传送时随机传输失败的修复比较复杂。

传统的密钥管理方案成功地提高了允许非计划驱逐组成员的技术的可升级性，但是最好的技术仍然在发送消息方面开销很大。相反我们关注于计划驱逐的问题。即，在一些任意将来的 ADU 之后，但在接收者请求一个话路时计划的驱逐。我们断言很多基于预先支付或订购的商业情形不需要非计划的驱逐但确实需要任意的计划驱逐。这样的例子包括付费电视，按次计费电视或网络游戏。

为了实现计划的但任意的驱逐，我们设计了由发送者用来系统地改变组密钥的密钥序列构造的选择，以便任何序列的子范围都可以通过揭示少量的种子（每个 16B）来重新构造。这样接收者可以被授权访问数据序列的任意子范围。所有可行的方案使用 O(log(N)) 个种子可以为每个接收者揭示 N 个密钥。各方案的不同之处在于计算每个密钥的处理负载，其为了安全性而作的折衷。处理负载最重的方案在开始时平均只需要 O(2(log(N)-1)) 次快速哈希运算，然后平均只需要另外 16 次哈希运算以计算序列中的每个新密钥，该操作可事先进行。处理负载最轻的方案比它的处理量少四倍。

为了将本工作放入环境中，对于按秒收费的付费电视，在 15 分钟内有一千

万观看者的 10%接入或离开，最好的替换方案可能会每秒钟生成几十 RB 千个字节数量级的重新分配消息以多点传送给每个组成员。本工作只需要在或许四小时的观看开始时，给每个接收者单点传送一个几百字节的消息。

附录 A-识别 BHT 的中间种子的最小集合的算法

在下列的类 C 语言码段中

- 函数 odd(x)检验 x 是否为奇数
- 函数 reveal(d, i)将种子 $s_{d,i}$ 揭示给接收者

```
min = m; max = n;
if  (min max)  error();      //拒绝 min max
for (d=D;  d=0;  d--)  {  //从树的底部开始操作
                          //每个循环沿树往上移一层
    if (min == max)  {     //min 和 max 会聚
      reveal(d,min);       //...揭示子树的根...
      break;               //...并退出
    }
    if  odd(min)  {        //奇数最小值永远不会是左孩子...
      reveal(d,min);       //...揭示奇数最小种子
      min++;               //将 min 向右移动一个种子
    }
    if  !odd(min)  {       //偶数最大值永远不会是右孩子...
      reveal(d,max);       //...揭示偶数最大种子
      max--;               //将 max 向右移动一个种子
    }
    if  (min max) break;   //如果 min&max 是表兄弟，则退出
    min/ = 2;              //...二等分 min...
    max/ = 2;              //...二等分 max...
  }                        //...向上一层，循环
```

附录 B-识别 BHC-T 的中间种子的最小集合的算法

在下列的类 C 语言码段中

- 函数 odd(x)检验 x 是否为奇数
- 函数 reveal(d, i)将种子 $s_{d,i}$ 揭示给接收者

```
min = m; max = n;
if  (min max)  error();      //拒绝 min max
d=0;                         //从树的底部开始操作
   if  (max <= min+1)  { //请求的 min&max 是相邻的或相同的...
      reveal (d, min);       //...所以揭示左边...
      if  (max < min)      //...请求的 min&max 是不相同的...
          reveal (d, max);       //...所以也揭示右边...
      break;                 //...并退出...
   }
for (d=0; ; d++)  {          //每个循环沿树往上移一层
    if  (max <= min+3) {     //min&max 相隔 2 或 3 个...
       if  (max < min+3) {   // min&max 相隔 2 个...
          reveal (d, min);       //...所以揭示左边...
          reveal (d, max);       //...和右边...
          reveal (d, min+1);      //...和中间...
          break;                //...并退出
       } else {                 // min&max 相隔 3 个,所以...
          if (!odd(min)) {      //...仅当 min 为偶数时...
            reveal (d, min+1);      //...揭示左中间...
            reveal (d, max-1);      //...和右中间...
            break;                //...并退出
          }
```

```
        }
    }
    if  ! odd(min)  {          //偶数最小值永远不会是右孩子...
        reveal(d,min);           //...揭示偶数最小种子
        min++;                   //将 min 向右移动一个种子
    }
    if  odd(max)  {            //奇数最大值永远不会是左孩子...
        reveal(d,max);           //...揭示奇数最大种子
        max--;                   //将 max 向左移动一个种子
    }
    min/ = 2;                  //...二等分 min...
    max/ = 2;                  //...二等分 max...
}                              //...向上一层，循环
```

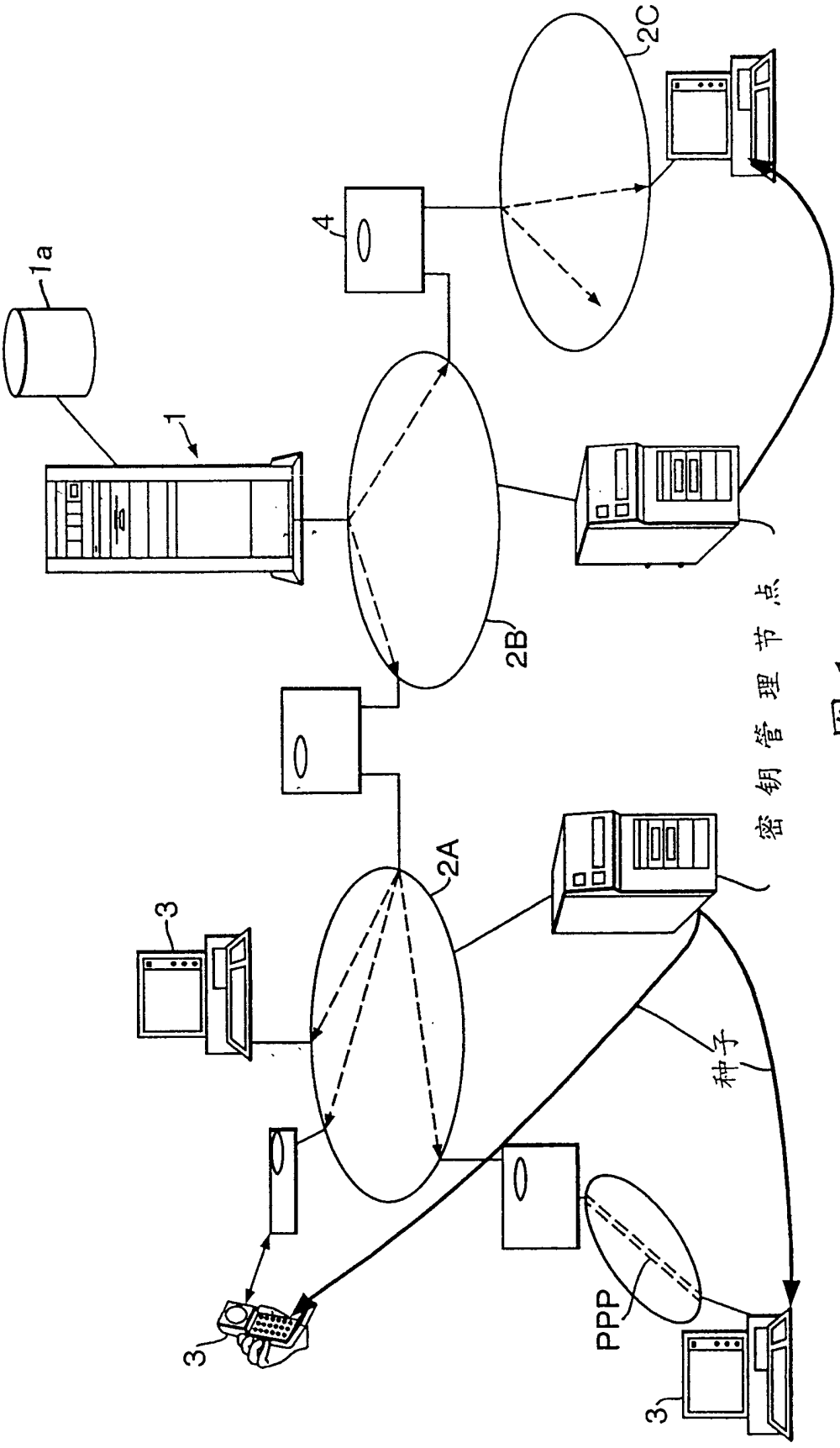


图 1

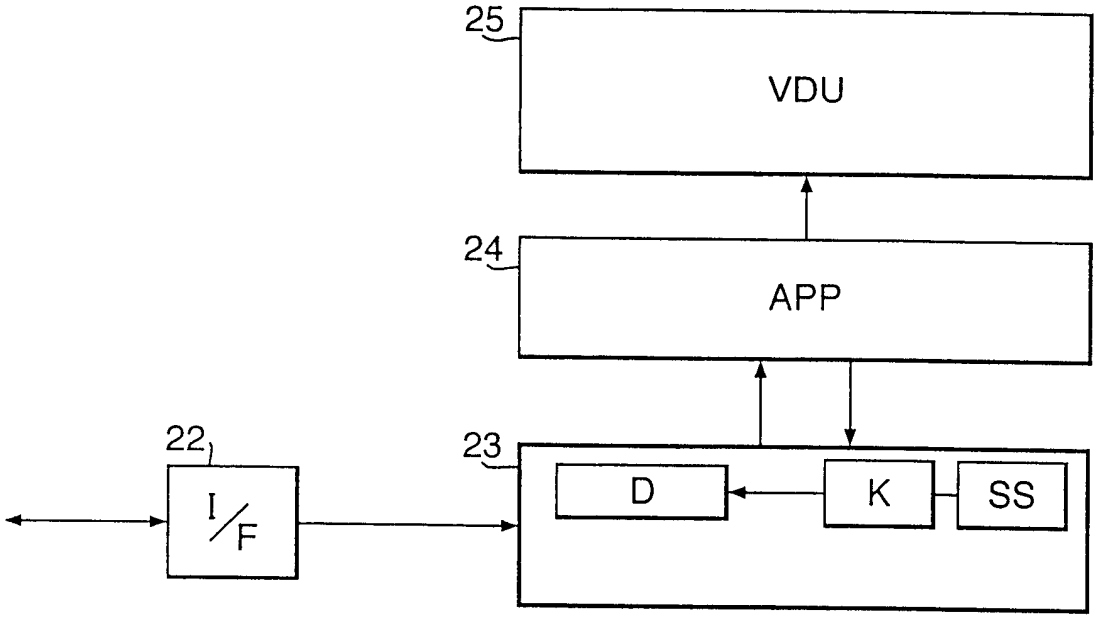

25
VDU
24
APP
22
I/F
23
D
K
SS

图 2

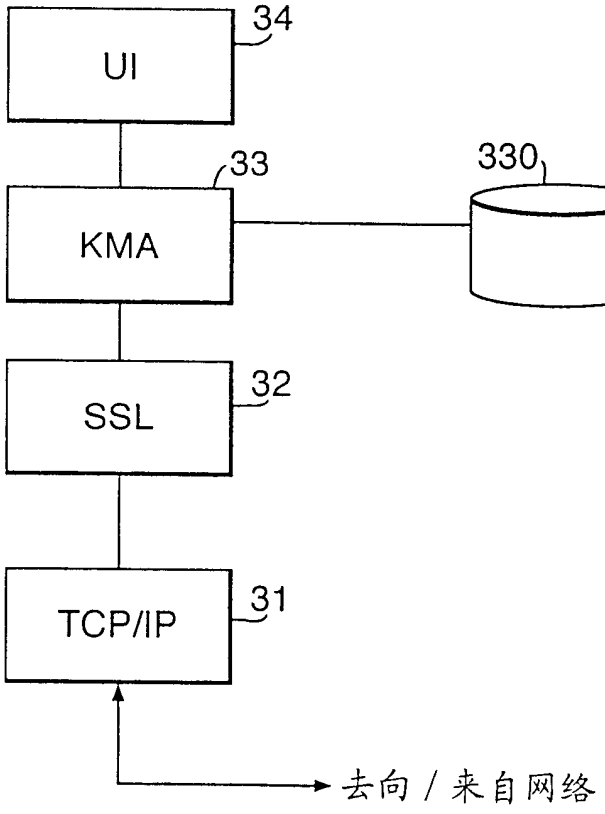

34
UI
33
330
KMA
SSL
32
TCP/IP
31
去向 / 来自网络

图 3

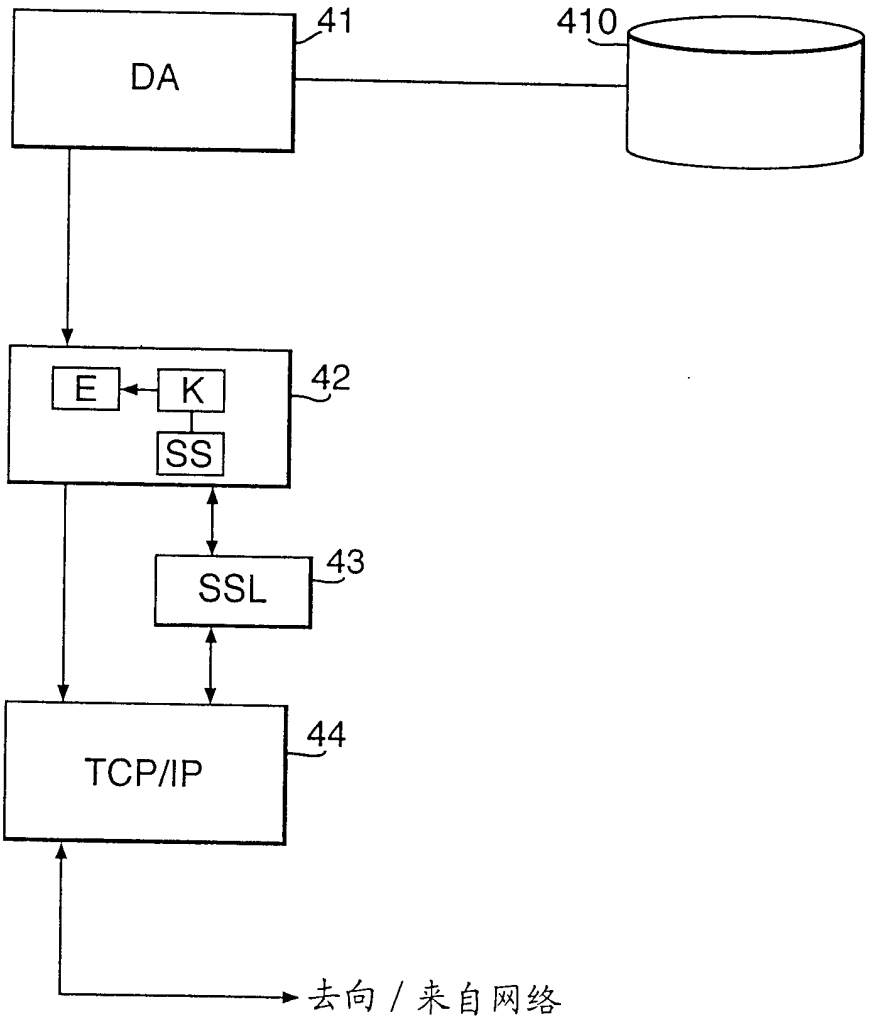


图 4

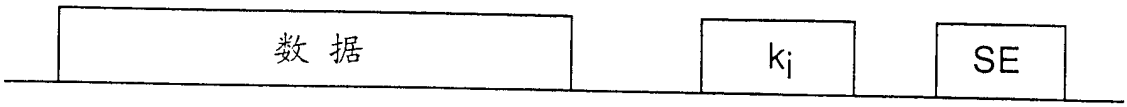


图 5

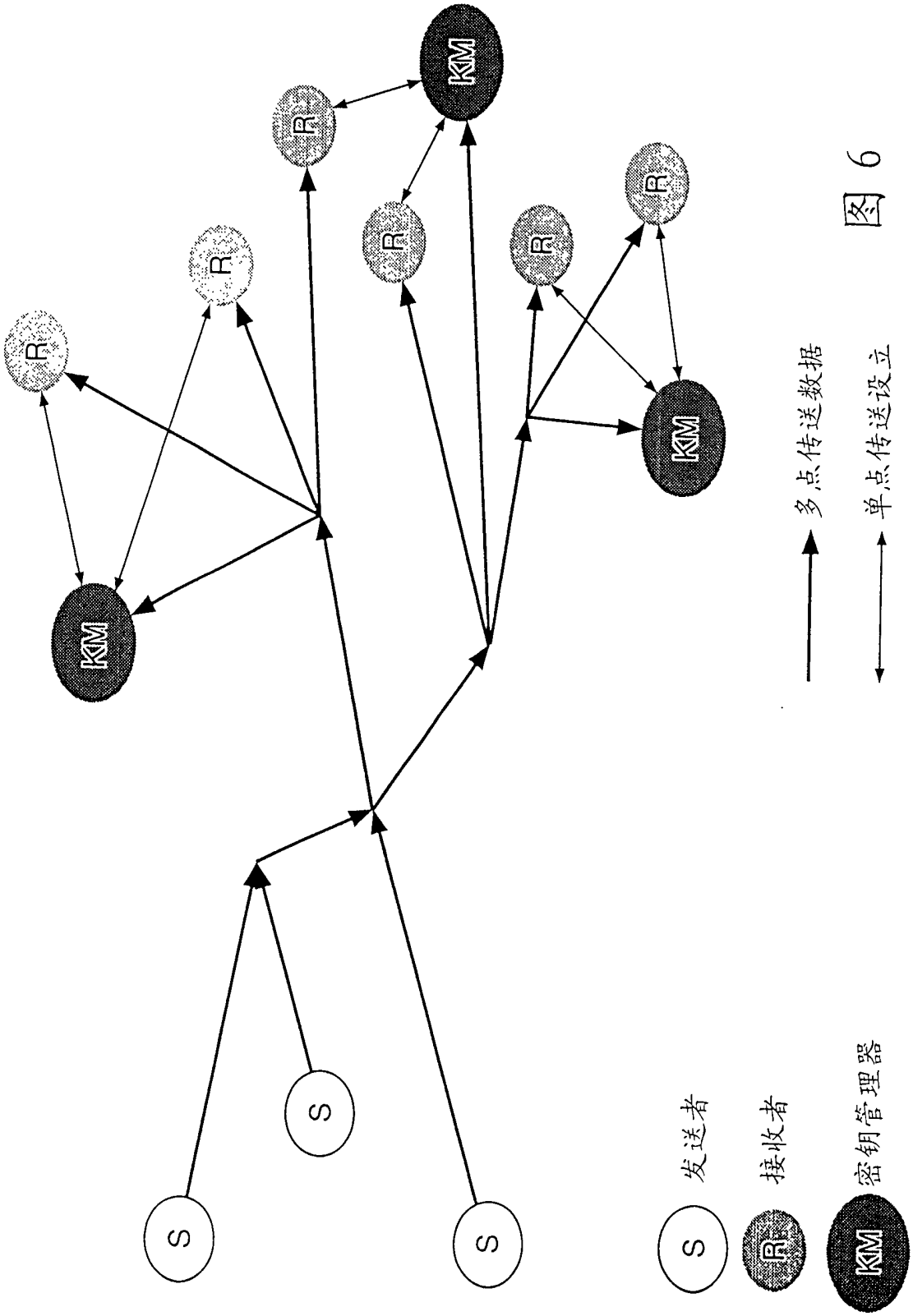


图 6

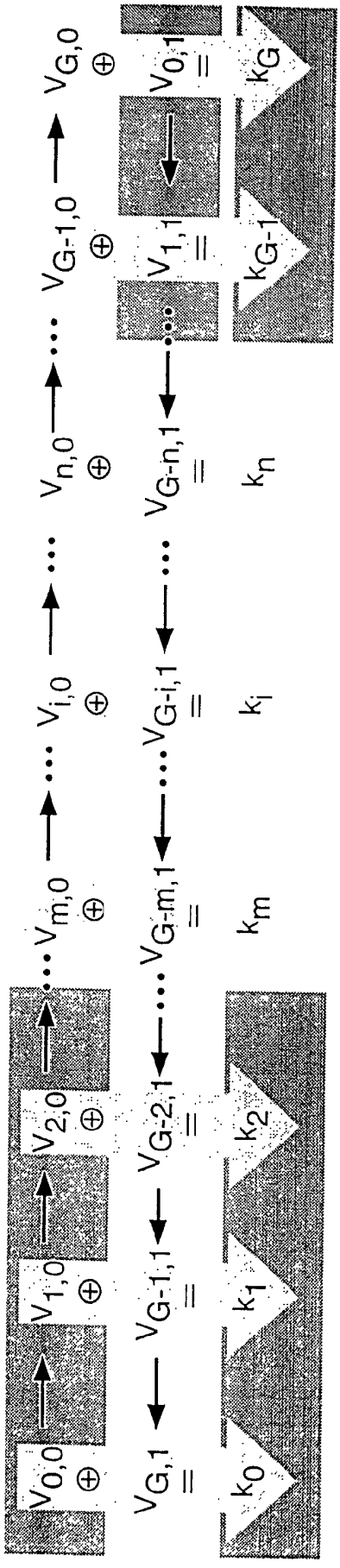

$V_{0,0}$
$V_{1,0}$
$V_{2,0}$
$V_{m,0}$
$V_{i,0}$
$V_{n,0}$
$V_{G-1,0}$
$V_{G,0}$
$V_{G,1}$
$V_{G-1,1}$
$V_{G-2,1}$
$V_{G-m,1}$
$V_{G-i,1}$
$V_{G-n,1}$
$V_{1,1}$
$V_{0,1}$
k_0
k_1
k_2
k_m
k_i
k_n
k_{G-1}
k_G
$V_{0,0} \longrightarrow V_{1,0}$ 表示 v(1,0)=b(v(0,0))
$V_{0,0} \oplus V_{G,1}=k_0$ 表示 k_0=c(v(0,0), v(G,1))

图 7

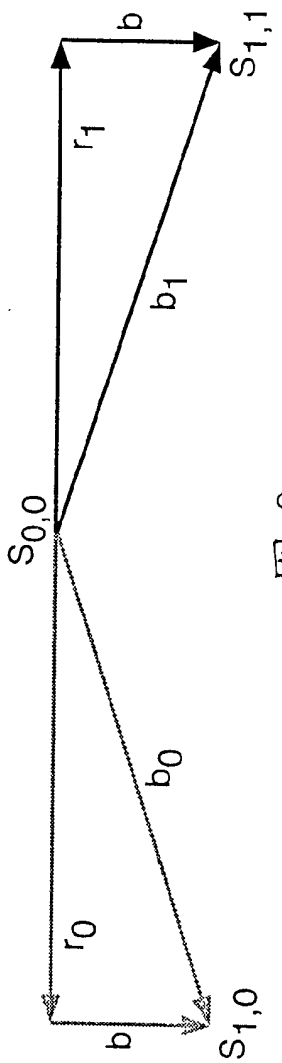

$S_{0,0}$
$S_{1,1}$
$S_{1,0}$
r_0
r_1
b_0
b_1
b
b

图 8

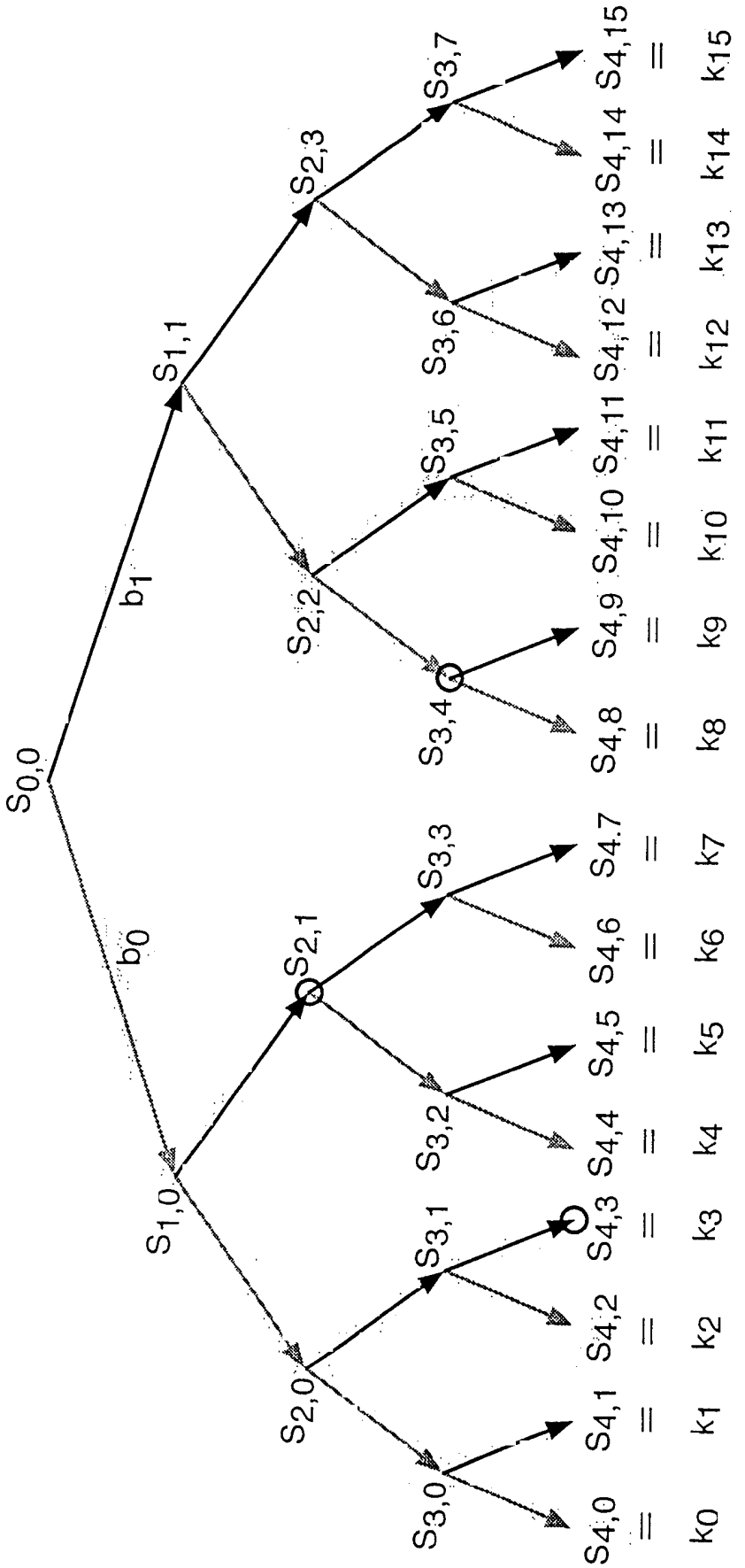

S0,0
b0
b1
S1,0
S1,1
S2,0
S2,1
S2,2
S2,3
S3,0
S3,1
S3,2
S3,3
S3,4
S3,5
S3,6
S3,7
S4,0
S4,1
S4,2
S4,3
S4,4
S4,5
S4,6
S4.7
S4,8
S4,9
S4,10
S4,11
S4,12
S4,13
S4,14
S4,15
k0
k1
k2
k3
k4
k5
k6
k7
k8
k9
k10
k11
k12
k13
k14
k15

图 9

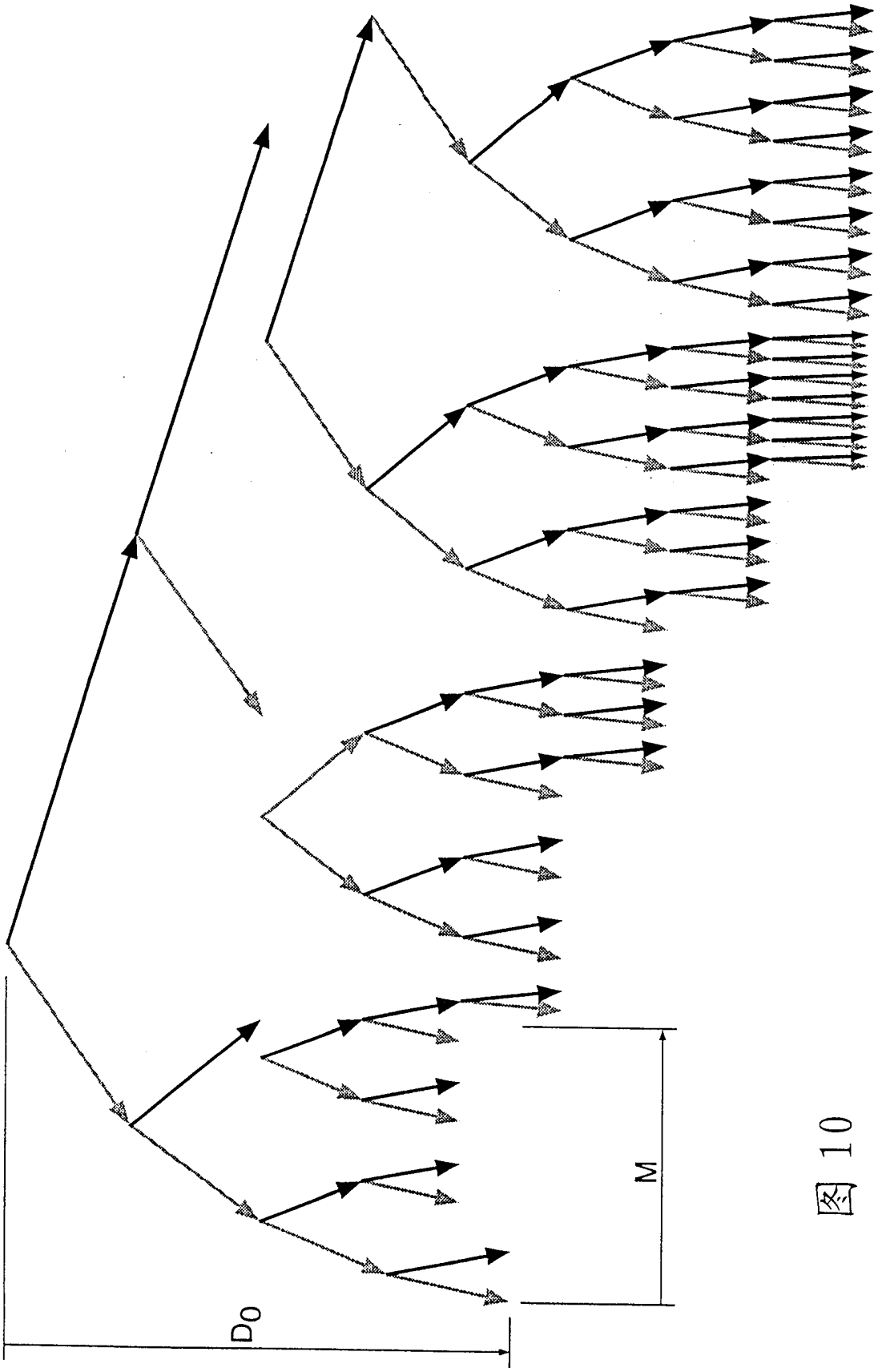


图 10

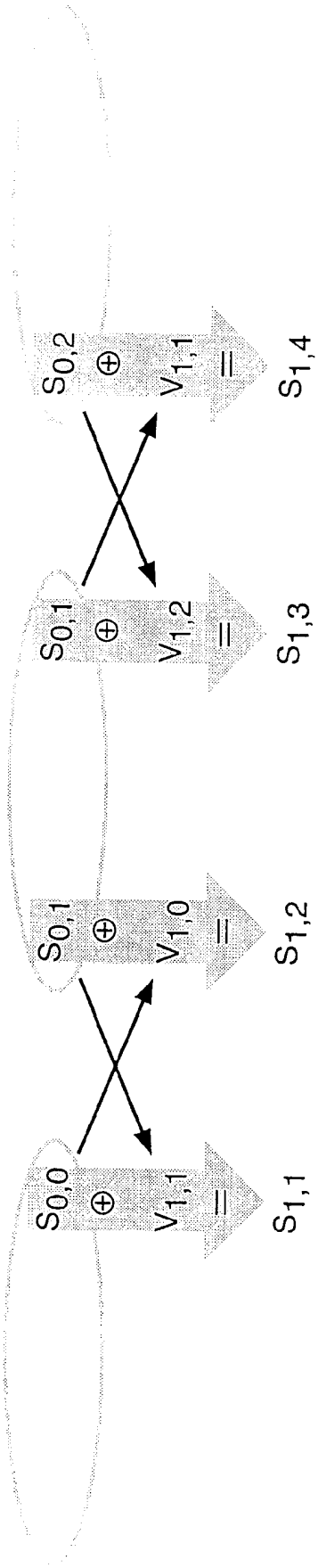

$S_{0,0}$ ⊕ $V_{1,1}$ = $S_{1,1}$
$S_{0,1}$ ⊕ $V_{1,0}$ = $S_{1,2}$
$S_{0,1}$ ⊕ $V_{1,2}$ = $S_{1,3}$
$S_{0,2}$ ⊕ $V_{1,1}$ = $S_{1,4}$
$S_{0,0} \longrightarrow V_{1,0}$ 表示 v(1,0)=b(s(0,0))
$S_{0,0} \oplus V_{1,1} = S_{1,1}$ 表示 s(1,1)=c(s(0,0),b(s(0,1)))

图 11a

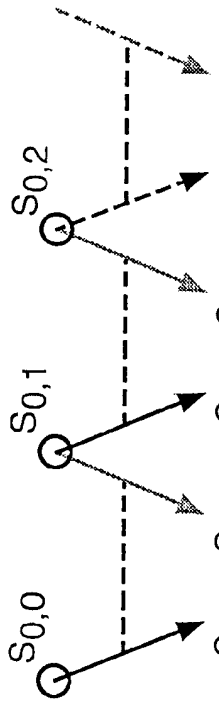

$S_{0,0}$
$S_{0,1}$
$S_{0,2}$
$S_{1,1}$ $S_{1,2}$ $S_{1,3}$ $S_{1,4}$

图 11b

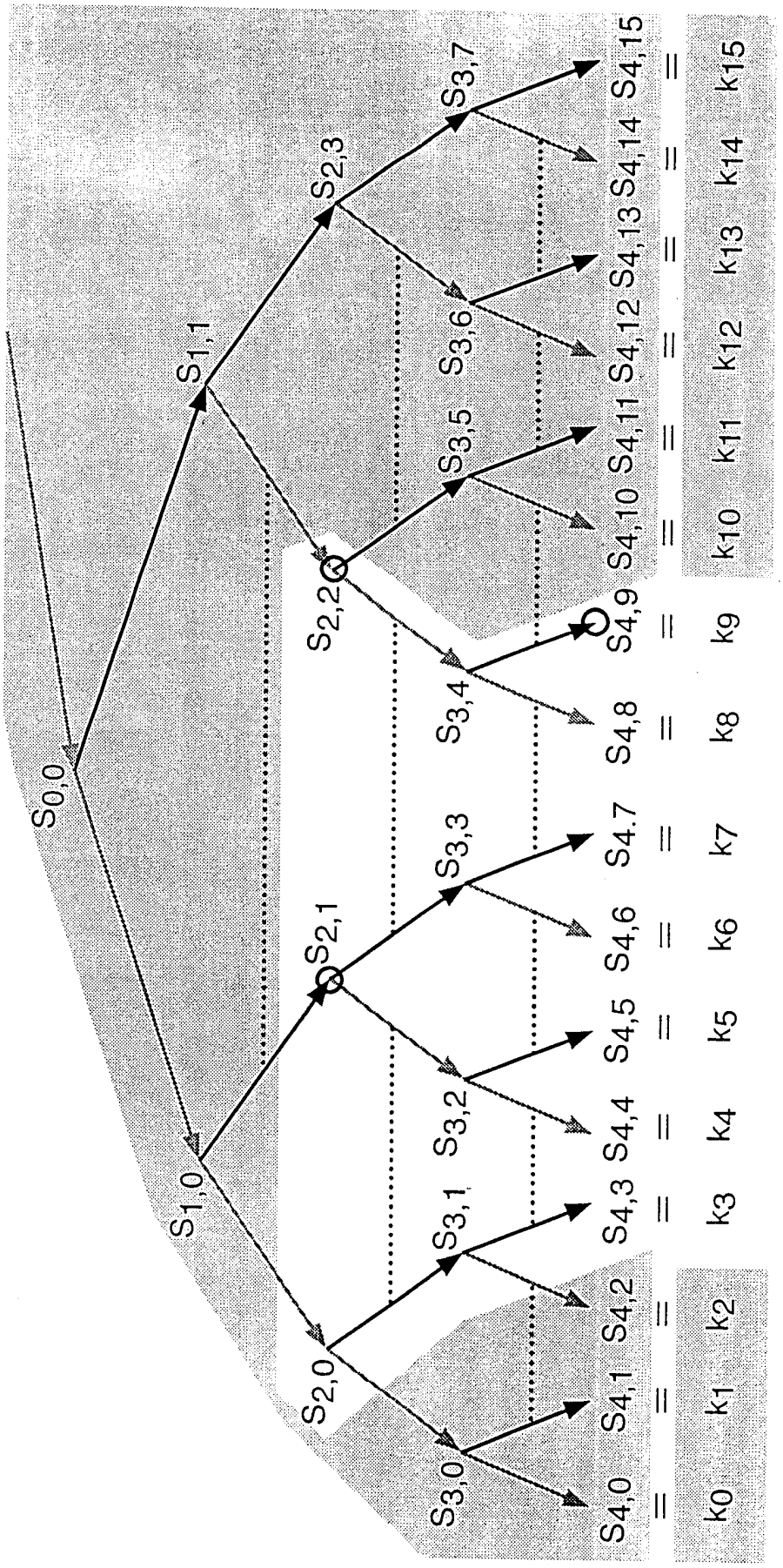

S0,0
S1,0
S1,1
S2,0
S2,1
S2,2
S2,3
S3,0
S3,1
S3,2
S3,3
S3,4
S3,5
S3,6
S3,7
S4,0
S4,1
S4,2
S4,3
S4,4
S4,5
S4,6
S4.7
S4,8
S4,9
S4,10
S4,11
S4,12
S4,13
S4,14
S4,15
k0
k1
k2
k3
k4
k5
k6
k7
k8
k9
k10
k11
k12
k13
k14
k15

图 12

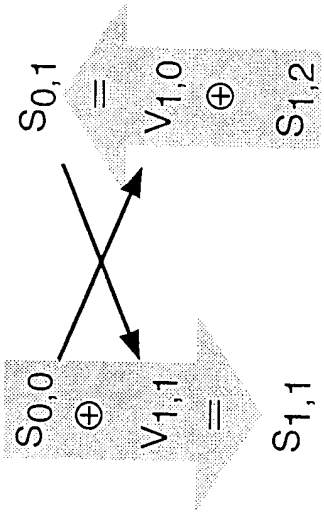

S0,0
⊕
V1,1
=
S1,1
S0,1
=
V1,0
⊕
S1,2

图 13

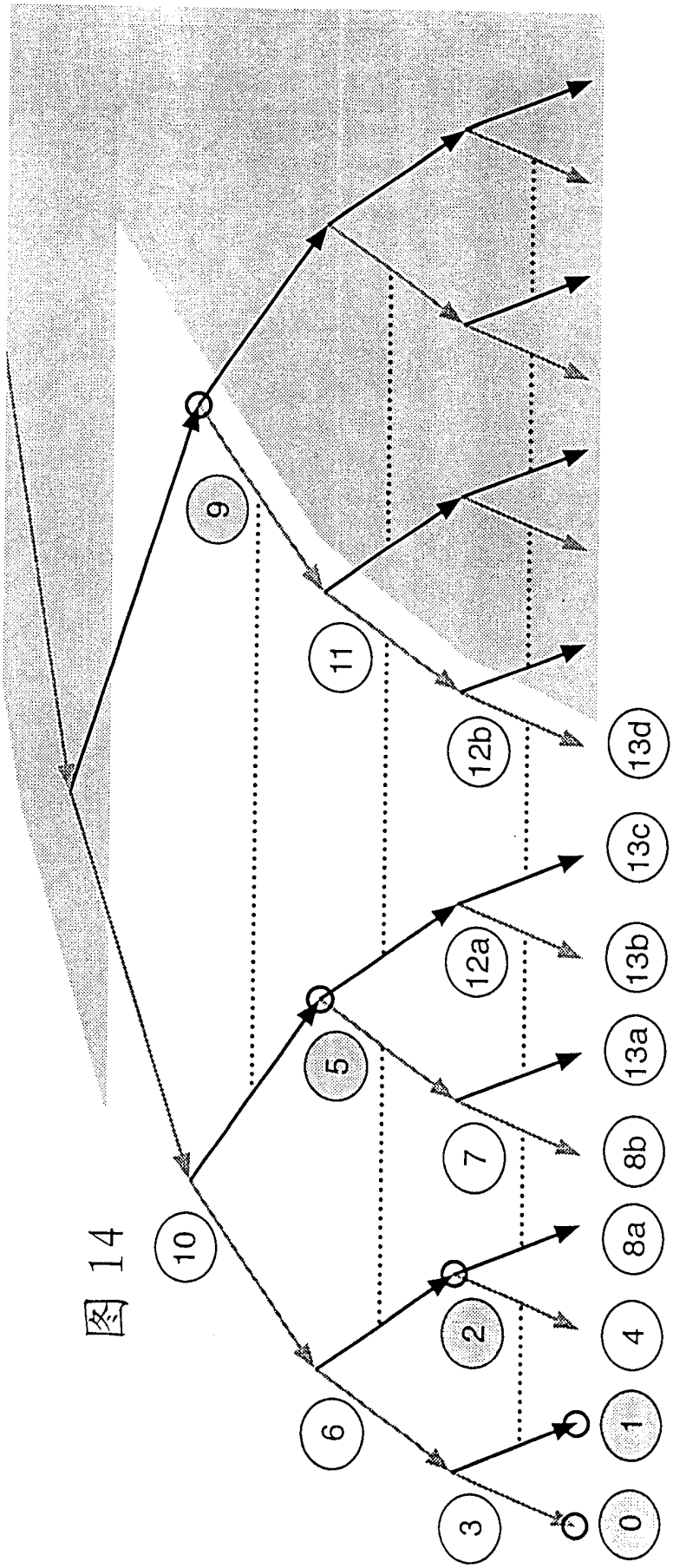

0
1
2
3
4
5
6
7
8a
8b
9
10
11
12a
12b
13a
13b
13c
13d

图 14

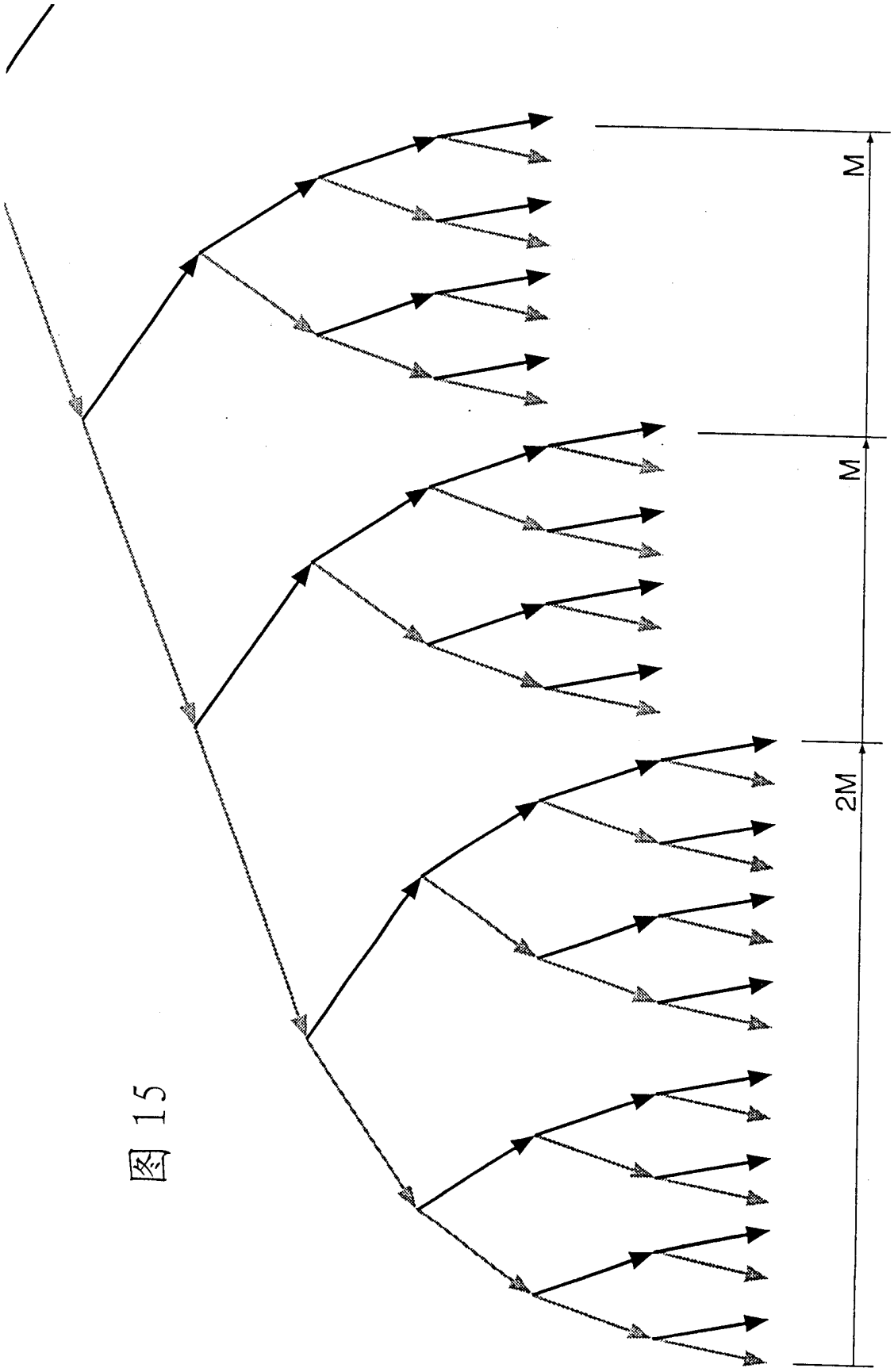


图 15

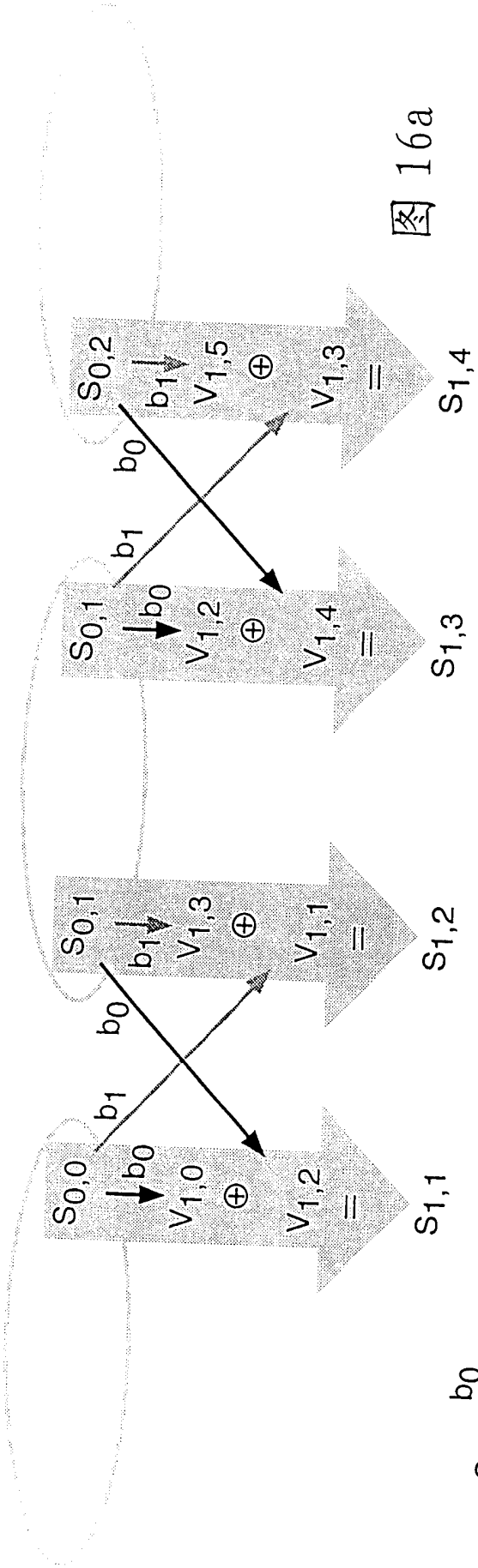


图 16a

$V_{1,0}=b_0(S_{0,0})$ 表示 $b_0 \xrightarrow{S_{0,0}} V_{1,0}$

$V_{1,1}=b_1(S_{0,0})$ 表示 $b_1 \xrightarrow{S_{0,0}} V_{1,1}$

$S_{1,1}=c(b_0(S_{0,0}),b_0(S_{0,1}))$ 表示 $V_{1,0} \oplus V_{1,2}=S_{1,1}$

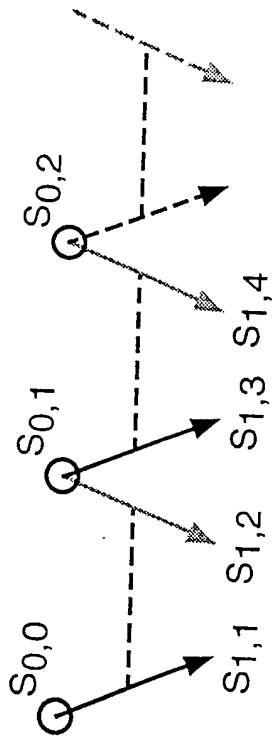


图 16b

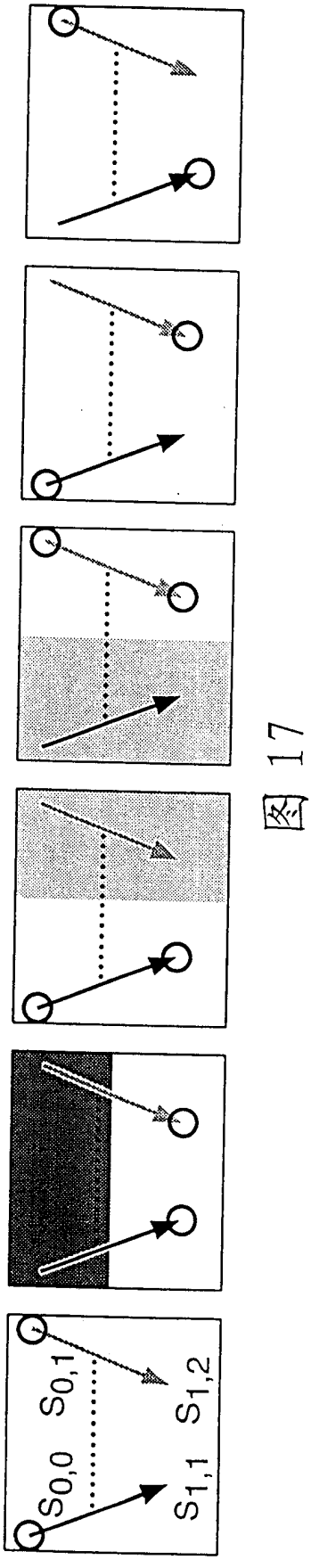


图 17

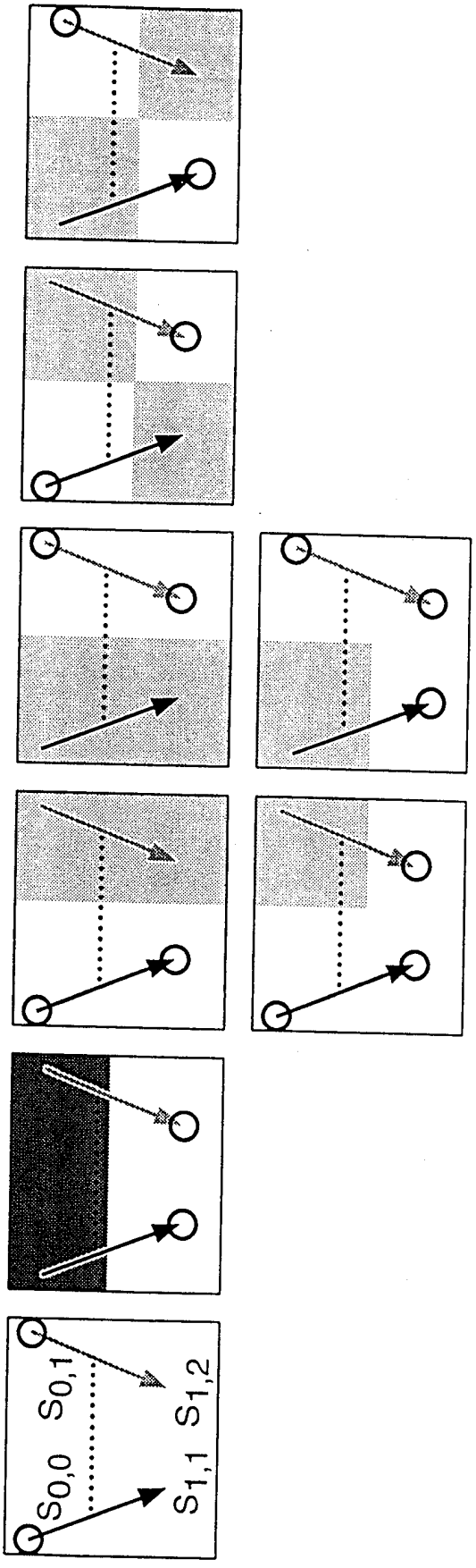


图 18

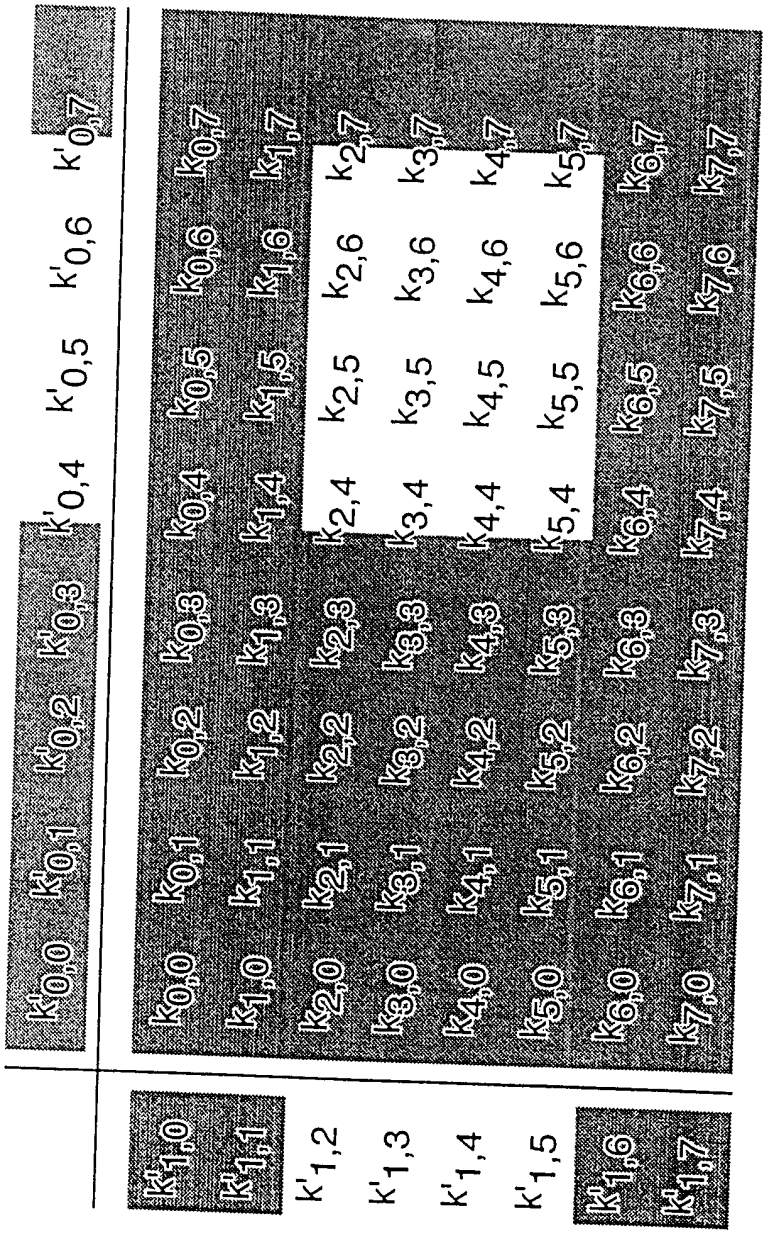

k'0,0 k'0,1 k'0,2 k'0,3 k'0,4 k'0,5 k'0,6 k'0,7
k'1,0 k'1,1 k'1,2 k'1,3 k'1,4 k'1,5 k'1,6 k'1,7
k0,0 k0,1 k0,2 k0,3 k0,4 k0,5 k0,6 k0,7
k1,0 k1,1 k1,2 k1,3 k1,4 k1,5 k1,6 k1,7
k2,0 k2,1 k2,2 k2,3 k2,4 k2,5 k2,6 k2,7
k3,0 k3,1 k3,2 k3,3 k3,4 k3,5 k3,6 k3,7
k4,0 k4,1 k4,2 k4,3 k4,4 k4,5 k4,6 k4,7
k5,0 k5,1 k5,2 k5,3 k5,4 k5,5 k5,6 k5,7
k6,0 k6,1 k6,2 k6,3 k6,4 k6,5 k6,6 k6,7
k7,0 k7,1 k7,2 k7,3 k7,4 k7,5 k7,6 k7,7

图 19

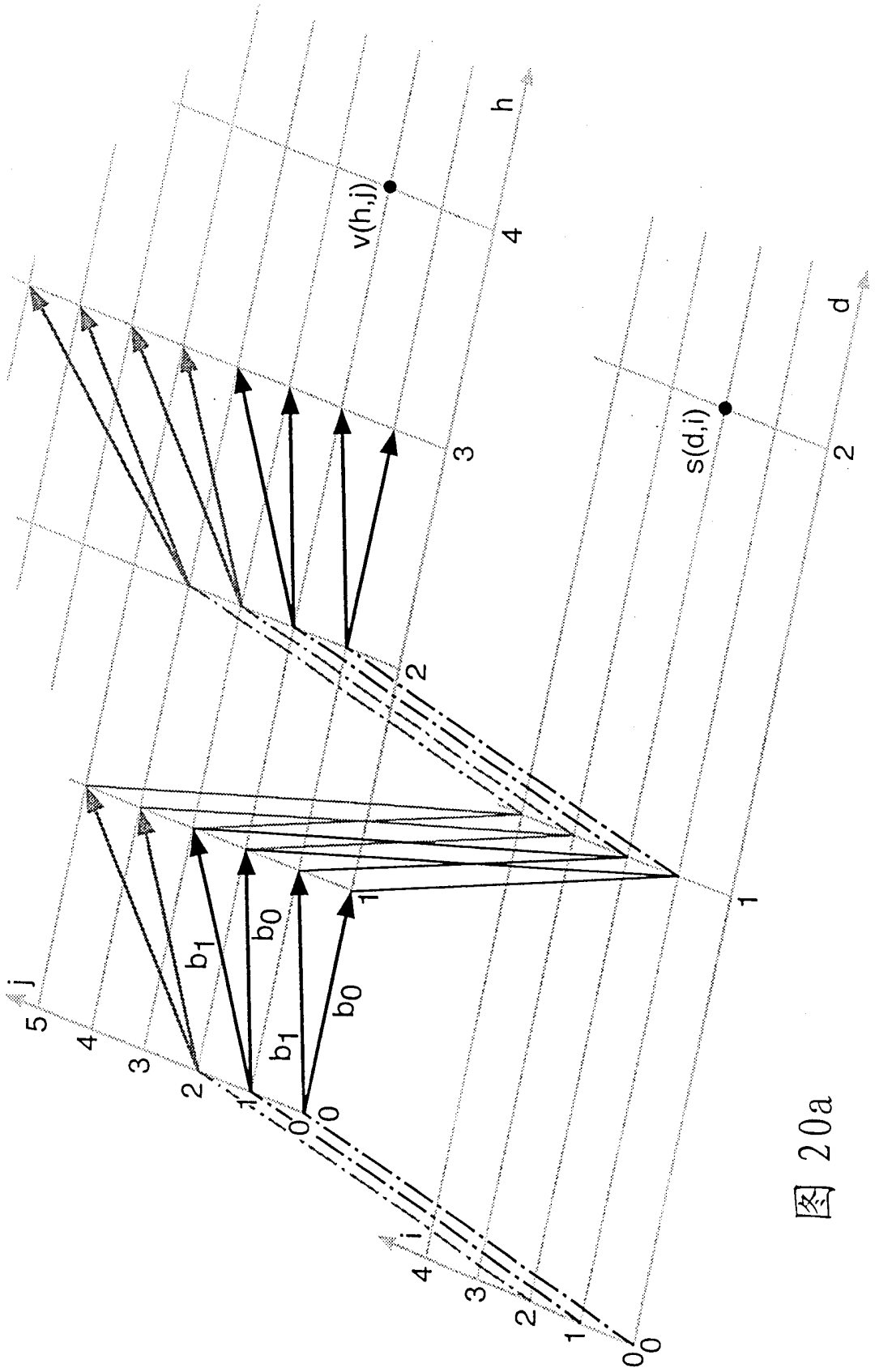


图 20a

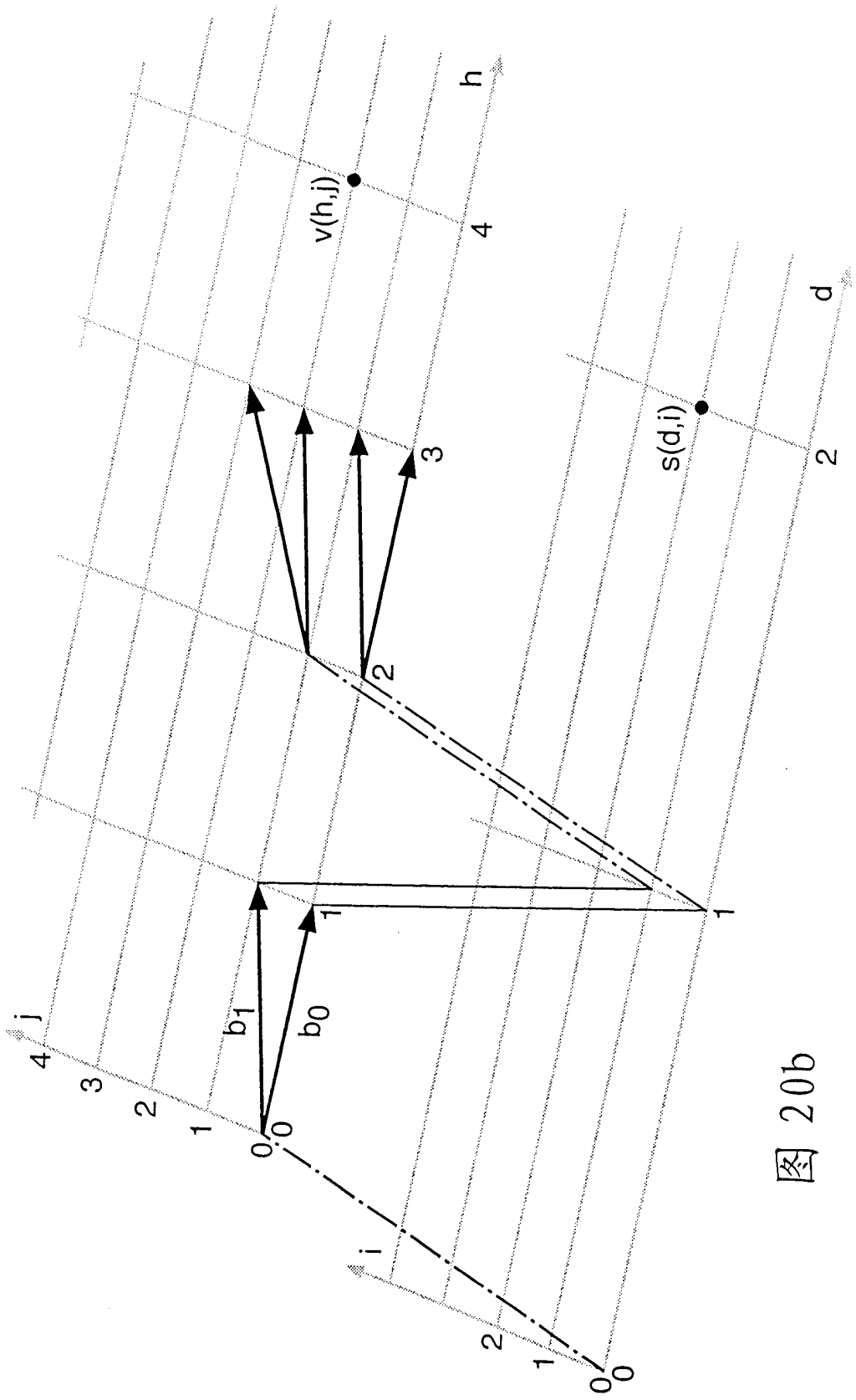


图 20b

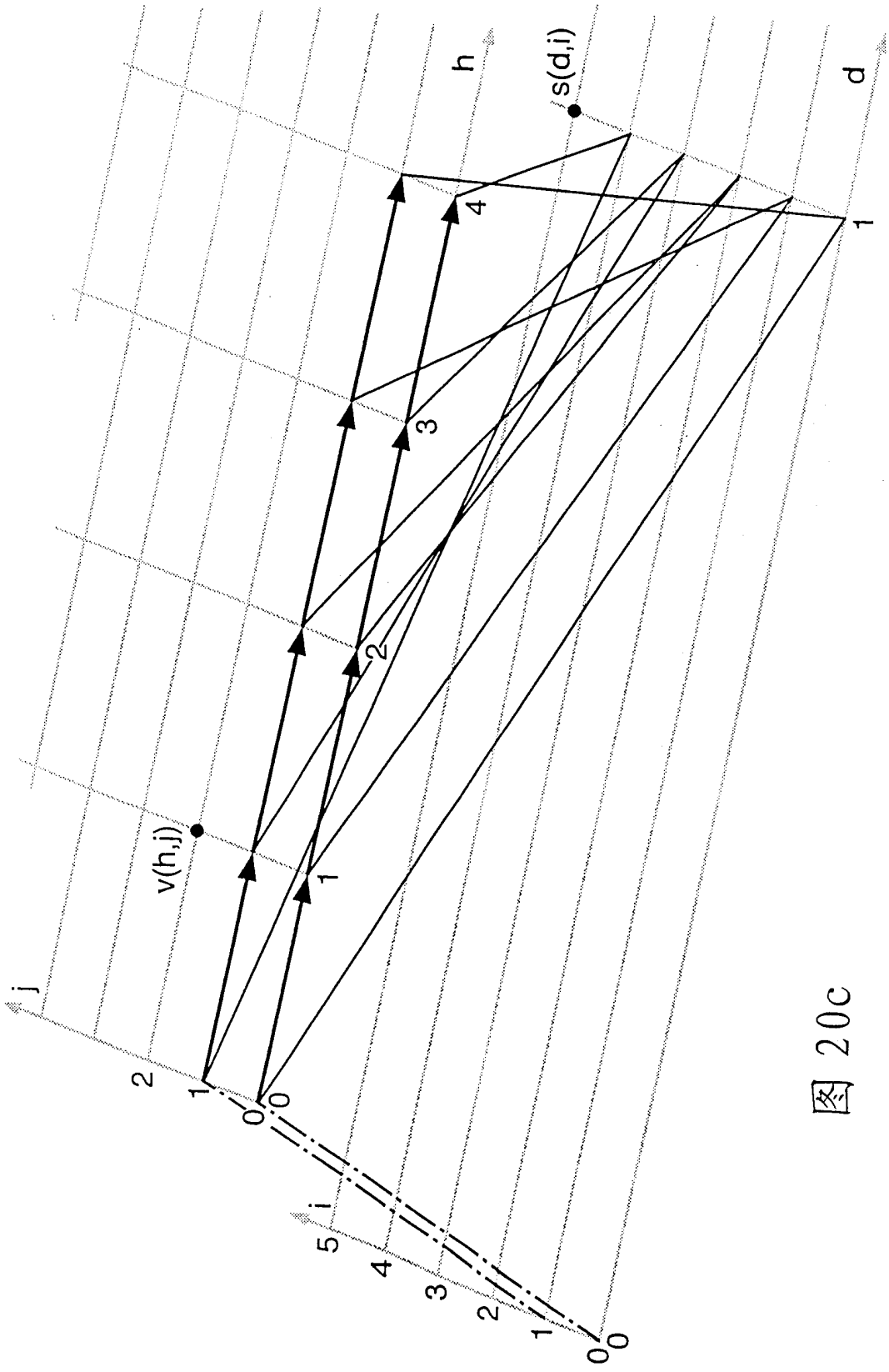


图 20c

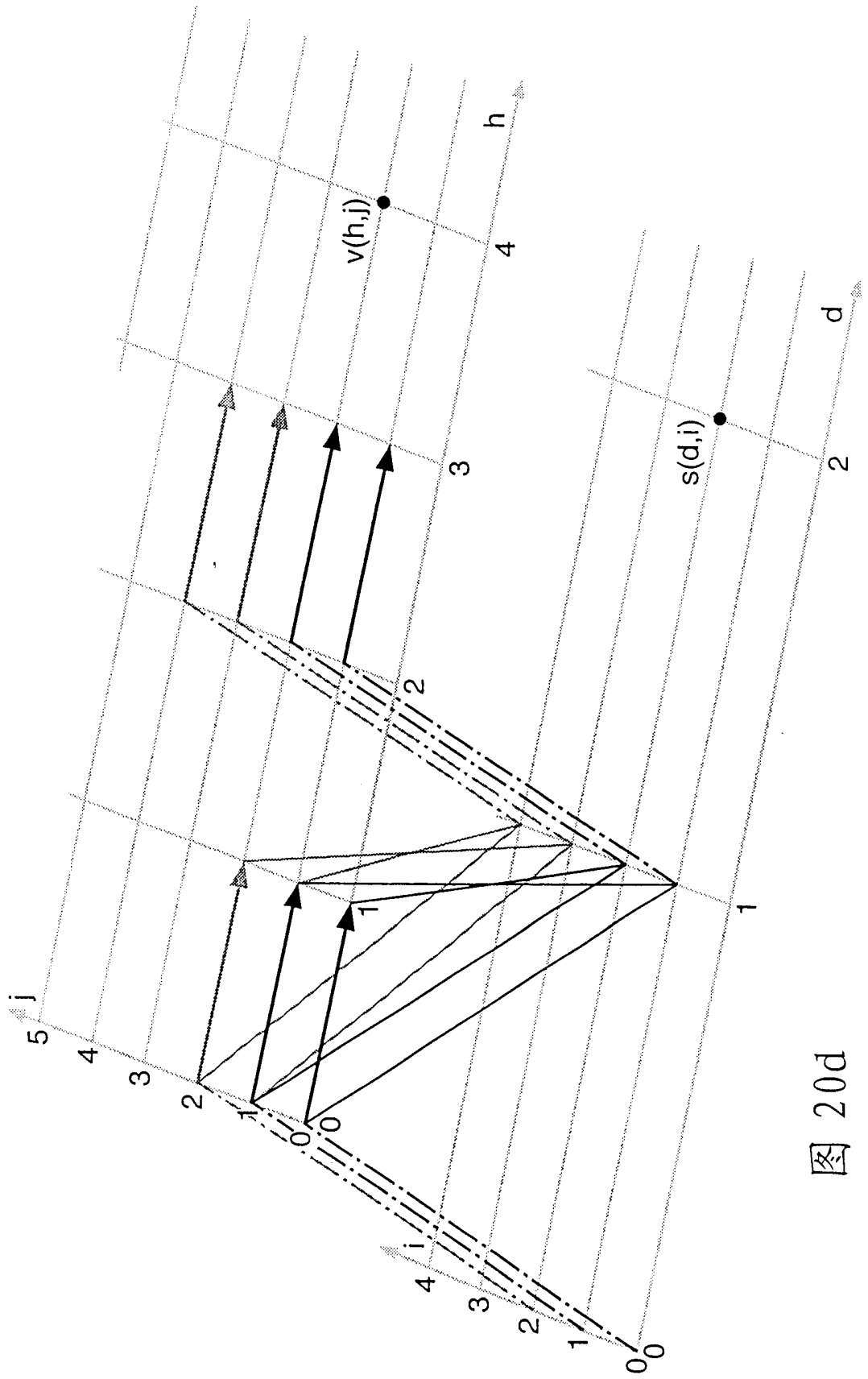


图 20d

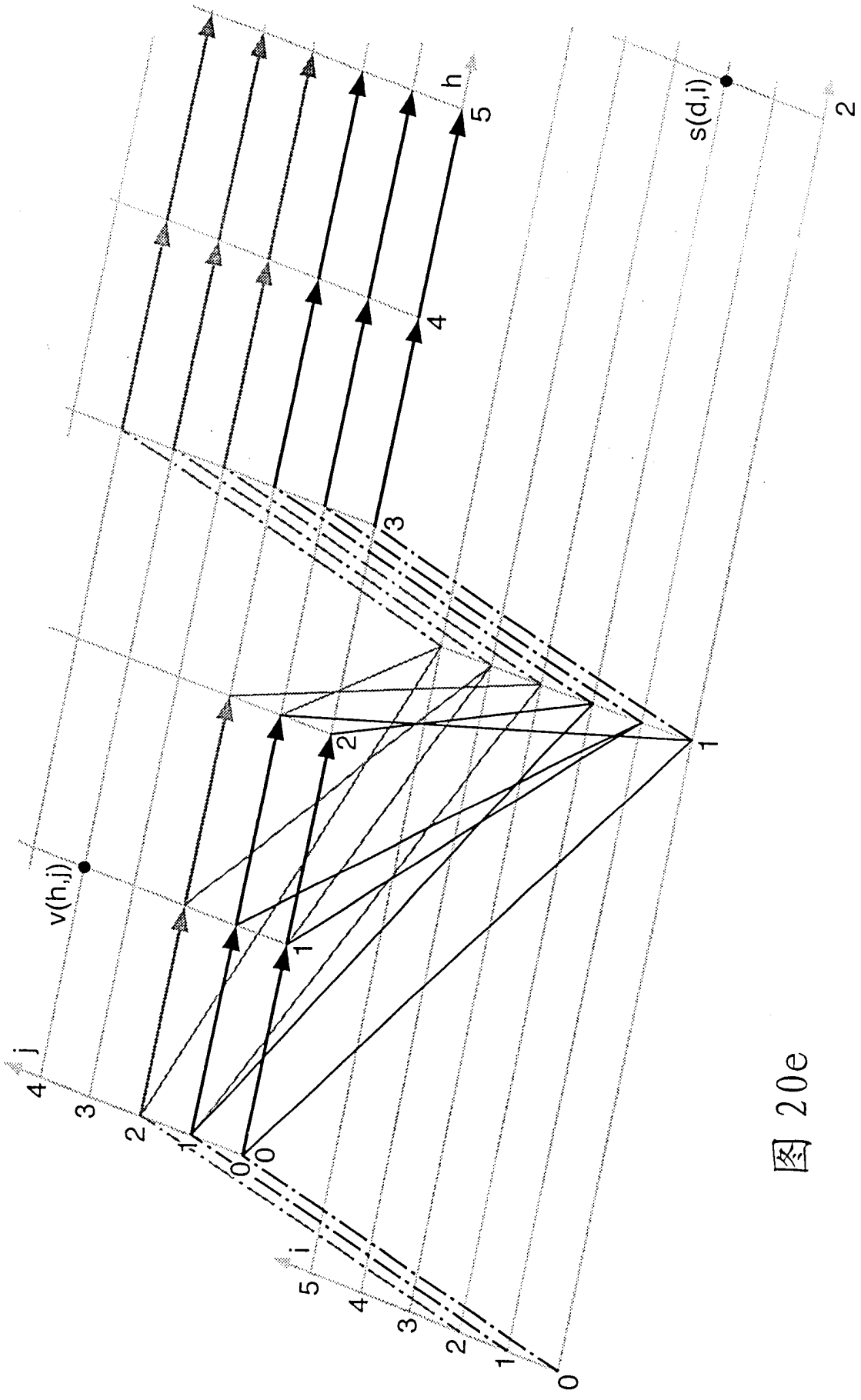


图 20e