(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 102986163 A
(43)申请公布日 2013.03.20

(21)申请号 201180022607.4
(22)申请日 2011.03.04
(30)优先权数据
61/311,106 2010.03.05 US
61/314,395 2010.03.16 US
(85)PCT申请进入国家阶段日
2012.11.05
(86)PCT申请的申请数据
PCT/US2011/027287 2011.03.04
(87)PCT申请的公布数据
WO2011/109772 EN 2011.09.09
(71)申请人 交互数字专利控股公司
地址 美国特拉华州
(72)发明人 A·施米特 A·莱切尔 I·查 S·B·帕塔尔 Y·C·沙阿
(74)专利代理机构 北京润平知识产权代理有限公司 11283
代理人 刘国平 南毅宁
(51)Int.Cl.
H04L 9/32(2006.01)

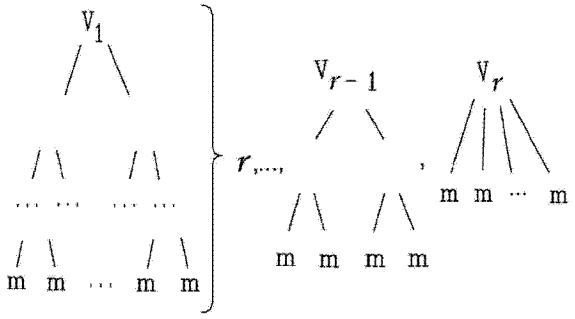
权利要求书 2 页 说明书 49 页 附图 28 页

(54)发明名称

给设备提供安全性的方法和装置

(57)摘要

提供了一种用于生成验证数据的系统、方法和装置，验证数据可以用于确认无线发射接收单元(WTRU)。验证数据可以使用树结构生成，其中该树结构具有表示为根节点的受保护的寄存器和表示为叶节点的组件测量。验证数据可以用于确认 WTRU。确认可以使用分割确认来执行，这是所描述确认的将确认任务分布到两个或更多个实体上的一种形式。还描述了子树证实，其中树结构的子树可以由第三方进行证实。



CN 102986163 A

1. 一种在无线发射接收单元(WTRU)中用于生成能用于该WTRU的确认的验证数据的方法，该WTRU包括一个或多个组件并具有包括多个安全寄存器的安全环境，该方法包括：

针对所述WTRU的多个组件中的每一个组件，获得表示所述WTRU的所述组件的测量的值；

生成测量日志(ML)并将该ML存储在所述WTRU上，其中该ML包含所述组件测量值和其他组件专用数据的记录；

根据针对每个组件的所述组件测量值生成验证数据，并将该验证数据存储在所述安全环境内的所述安全寄存器的一个或多个安全寄存器中；以及

将所述验证数据和所述ML组织到树结构中，其中包含所述验证数据的所述安全寄存器定义所述树结构的根，所述ML定义所述树结构的内部节点，以及包含在所述ML中的所述测量值定义所述树结构的叶，而且

其中，使用所述安全环境的安全扩展操作来形成所述树结构。

2. 根据权利要求1所述的方法，其中所述树结构是二叉树结构。

3. 根据权利要求1所述的方法，其中所述安全环境是可信平台模块。

4. 根据权利要求1所述的方法，其中所述测量日志是存储的测量日志。

5. 根据权利要求1所述的方法，其中使用寄存器移位操作进一步形成所述树结构。

6. 一种在无线发射接收单元(WTRU)中用于使用安全环境内的多个安全寄存器生成验证数据的方法，该WTRU包括一个或多个组件并具有安全环境，该方法包括：

获得表示所述WTRU的组件的测量的值；

根据所述测量值生成验证数据，并将该验证数据存储在所述安全环境内的所述寄存器的一个寄存器中；

将所述测量值存储到树结构中的叶节点；以及

在所述安全环境内执行一个或多个扩展操作以将存储在所述叶节点中的所述值扩展到所述树结构的根节点，其中所述根节点包括在所述安全寄存器中的数据，所生成的验证数据被存储在该安全寄存器中。

7. 根据权利要求6所述的方法，其中所述树结构包括在从所述叶节点到所述根节点的路径上的至少一个内部节点，其中所述至少一个内部节点形成所述树结构的子树的根，且其中所述将存储在所述叶节点中的值扩展到所述树结构的所述根节点进一步包括：

执行一个或多个扩展操作以将存储在所述叶节点的所述值扩展到所述至少一个内部节点，其中所述至少一个内部节点包括在所述安全环境内的另一个安全寄存器；以及

执行一个或多个其他扩展操作以将来自所述至少一个内部节点的所述值扩展到所述树结构的所述根节点。

8. 根据权利要求6所述的方法，其中所述树结构是二叉树结构。

9. 根据权利要求6所述的方法，其中所述安全环境是可信平台模块。

10. 根据权利要求6所述的方法，其中执行一个或多个其他扩展操作以将来自所述至少一个内部节点的所述值扩展到所述树结构的所述根节点的所述步骤基于所述至少一个内部节点到所述根节点的距离。

11. 一种用于确认由无线发射/接收单元(WTRU)生成的树形验证数据的方法，其中所述树形验证数据包括组织到树结构中的验证数据元、测量日志(ML)和组件测量值，其中所

述验证数据元定义所述树结构的根节点，所述 ML 定义所述树结构的内部节点，以及所述组件测量值定义所述树结构的叶节点，该方法包括：

接收在所述树结构中组织的树形验证数据；

从在所接收的树形验证数据的根处的验证数据元开始，遍历所述树结构；

作为遍历所述树结构的部分，将所接收的树结构的分支节点和该分支节点的孩子节点处的值与参考树中相同节点位置处的值进行比较；

基于所述节点值的比较，确定是确认所述 WTRU 还是确认该 WTRU 中的单个组件。

12. 根据权利要求 11 所述的方法，该方法进一步包括：

如果分支节点或该分支节点的孩子节点的节点值与所述参考树的相应节点值不匹配，则通过对所述孩子节点值应用扩展操作来重新计算分支节点值；以及

将重新计算的节点值与所述参考树的相应节点值进行比较。

13. 根据权利要求 11 所述的方法，其中所述参考树被本地存储在确认器上。

14. 根据权利要求 11 所述的方法，其中由与所述 WTRU 相关联的网关设备执行所述步骤。

15. 根据权利要求 11 所述的方法，其中由与所述 WTRU 相关联的可信第三方执行所述步骤。

16. 根据权利要求 15 所述的方法，其中从与所述 WTRU 相关联的网关设备接收所述树形验证数据。

17. 一种用于证实由无线发射 / 接收单元（WTRU）生成的测量日志（ML）的节点值的方法，其中所述 ML 的值被存储作为树结构的节点，该树结构包括根节点、内部节点和叶节点，该方法包括：

接收证明包，该证明包指示将由子树证书授权中心（SCA）证实的节点值；

将所述节点值识别为能由所述 SCA 证实的节点值；

产生与所述节点值相关联的清单，其中该清单包括与所述节点值相关联的确认信息；

产生用于所述节点值的证书，该证书为被配置成将所述确认信息绑定到所述 WTRU 的安全环境；以及

发布具有所述清单的该证书，其中该证书和清单被提供给所述 WTRU 的所述安全环境，该 WTRU 将所述证书存储在该 WTRU 的 ML 中。

18. 根据权利要求 17 所述的方法，其中所述证书用于代替所述 WTRU 的所述 ML 的节点值。

19. 根据权利要求 17 所述的方法，其中所述证明包包括签名、所述节点值、由可信证书授权中心发布的证书以公共证明身份密钥。

20. 根据权利要求 17 所述的方法，其中所述 SCA 是与所述 WTRU 相关联的网关设备。

给设备提供安全性的方法和装置

[0001] 相关申请的交叉引用

[0002] 本申请要求 2010 年 3 月 16 日申请的美国临时申请顺序号为 61/314,395 和 2010 年 3 月 5 日申请的美国临时申请序列号为 61/311,106 的权益，其全文以引用的方式结合于此。

技术领域

[0003] 本申请涉及给设备提供安全性。

背景技术

[0004] 随着机器对机器通信（M2M）的到来，电子健康、地理跟踪、家庭自动化和消费电子设备中的应用成为可能。许多这种应用需要将网络运营商的装备放置在用户端。这些装备和设备易受恶意攻击。为了打击这种恶意攻击，除了包括防火墙和病毒保护的其它形式的设备保护之外，这种基于用户端的装备还需要设备完整性验证。

[0005] 已经讨论了用于设备完整性保护的若干方法。这些方法包括安全启动（boot）——其中可信的执行环境加载并仅执行通过完整性验证（verification）的软件组件（component）。但是这些方法需要一组未经组织的测量（measurement），在这种测量的数量很大时管理起来变得很繁琐。因此，需要的是这样的方法和相关装置，有助于收集、分类并组织这些测量以有利于有效搜索完整性失败的组件。

发明内容

[0006] 本文公开了用于生成验证数据的各种技术，包括一种用于生成可用于无线发射 - 接收单元（WTRU）的确认（validation）的验证数据的方法。WTRU 可以具有一个或多个组件和一个具有多个安全寄存器的安全环境。根据一个实施方式，对于 WTRU 的多个组件中的每一个可以获得一个值，代表 WTRU 组件的测量。可以生成一个测量日志（ML），包含组件测量值的记录，其它的组件专用数据可以存储在 WTRU 上。可以根据针对每个组件的组件测量值生成验证数据，验证数据可以存储在可信平台模块内的一个或多个安全寄存器中。验证数据和 ML 可以被组织成树结构。包含验证数据的安全寄存器可以定义树结构的根。ML 可以定义树结构的内部节点，包含在 ML 中的测量值可以定义树结构的叶。可以使用安全环境的安全扩展操作形成树结构。

[0007] 根据另一实施方式，可以获得代表 WTRU 组件的测量的值。可以根据测量值生成验证数据，验证数据可以存储在 WTRU 上安全环境内的寄存器中。测量值可以被存储在树结构中的一个叶节点处。可以在安全环境中执行一个或多个扩展操作以将存储在叶节点中的值扩展到所述树结构的根节点。根节点可以包括安全寄存器中的数据，生成的验证数据被存储在该安全寄存器中。

[0008] 根据另一实施方式，描述了一种对由无线发射 / 接收单元（WTRU）生成的树形（tree-formed）验证数据进行确认（validate）的方法。树形验证数据可以包括被组织成

树结构的验证数据元(data element)、测量日志(ML)和组件测量值。验证数据元可以定义树结构的根节点。ML 可以定义树结构的内部节点。组件测量值可以定义树结构的叶节点。树形验证数据可以在组织的树结构中接收。从接收的树形验证数据的根处的验证数据元开始,对树结构进行遍历(traverse)。作为遍历树结构的一部分,在接收的树结构的分支(branching)节点和分支节点的孩子(child)节点的值可以与在参考树中相同节点位置的值进行比较。然后可以基于节点值的比较确定是对 WTRU 还是对 WTRU 的单个组件进行确认。

[0009] 根据另一实施方式,描述了一种方法,用于证实(certify)由无线发射 / 接收单元(WTRU)生成的测量日志(ML)的节点值。ML 的值可以存储为包括根节点、内部节点和叶节点的树结构的节点。可以接收证明(attestation)分组,其指示将由子树证书授权(SCA)证实的节点值。节点值可以被识别为能够由 SCA 证实的节点值。可以创建与节点值相关联的清单(manifest),其包括与节点值相关联的确认信息。可以创建用于节点值的证书,其被配置为将确认信息绑定到 WTRU 的安全环境。证书可以随着清单一起发布,并提供给 WTRU 的安全环境,WTRU 将证书存储在其 ML 中。

[0010] 根据下面详细描述和相关附图,本文描述的系统、方法和装置的其它特性和方面将变得清楚。

附图说明

[0011] 当结合附图阅读时,前述的发明内容以及下面详细描述将更好理解。为了说明本文描述的系统、方法和装置,在附图中示出了示意性实施方式,但是,本发明不限于所公开的具体方法和手段。在附图中 :

[0012] 图 1 示出了示例性长期演进无线通信系统 / 接入网 ;

[0013] 图 2 是图 1 的长期演进无线通信系统的示例性框图 ;

[0014] 图 3 示出了树形存储测量日志(SML)和验证数据的一般结构 ;

[0015] 图 4 示出了显示树形成的一个算法(算法 1)的示例 ;

[0016] 图 5 示出了在右边沿的正确配置 ;

[0017] 图 6 示出了显示不完整树清除(cleanup)的一个算法(算法 2) ;

[0018] 图 7 示出了深度 3 的不完整树的顺序形成 / 树分支 ;

[0019] 图 8 示出了树验证数据的最大容量安排,其中在叶处的测量值表示为 m ;

[0020] 图 9 示出了树形 SML 中节点配置的分类 ;

[0021] 图 10 在 $2^d f$ 个坏叶随机分布中坏的内部节点的期望分数,其中 d=16 ;

[0022] 图 11 示出了 SML 树中每一个基本三角形的值的正确配置 ;

[0023] 图 12 示出了算法 1,用于找到线性哈希(hash)链中的第一个失败(failure)点 ;

[0024] 图 13 示出了 Huffman (哈夫曼)编码树的一个示例 ;

[0025] 图 14 示出了树修剪的一个示例 ;

[0026] 图 15 示意了最优树形验证系统图 / 系统图和相关通信 ;

[0027] 图 16 示意了具有替代孩子链路的树,其中由一个模块表示的软件组件或功能使用另一个模块 ;

[0028] 图 17 示出了算法 2——用于确定具有度量的二叉树的总体(population) ;

[0029] 图 18 示出了算法 3——用于确定使用 TPM 命令的树的总体；

[0030] 图 19 示出了算法 4——用于确定具有度量的 n 元(n-ary)树的总体；

[0031] 图 20 示出了算法 5——用于确定具有替代孩子链路的二叉树的总体；

[0032] 图 21 示出了比较和修剪算法 6——来确定完整性检查失败的节点和叶；

[0033] 图 22 示出了显示 TPM_reduced (减少)_Tree (树)_Verify (验证)_Load (加载)的算法 -1；

[0034] 图 23 示出了显示 TPM_reduced_Tree_Verify 的算法 -2；

[0035] 图 24a 示出了显示 TPM_Tree_Node (节点)_Verify 的算法 -3；

[0036] 图 24b 示出了显示 TPM_Reduced Tree_Update (更新)的算法 -4；

[0037] 图 25 示出了用于 PVM 的数据种类；

[0038] 图 26 示出了用于具有根的子树的子树证实(certification)协议；

[0039] 图 27 示出了证书子树绑定；

[0040] 图 28 示出了左边不平衡的多树结构；

[0041] 图 29 示出了如本文描述的树结构的一个示意性实施方式；

[0042] 图 30 是组件子树结构；

[0043] 图 31 示出了分割确认步骤 1 :测量的收集；

[0044] 图 32 示出了分割确认步骤 2 :子树的证实；

[0045] 图 33 示出了分割确认步骤 3 :服务连接；

[0046] 图 34 示出了分割确认的 H(e)NB 用例；

[0047] 图 35 示出了 H(e)NB 阻止接入恶意设备；

[0048] 图 36 示出了基于设备类型、设备类别(class)、设备特性或连接配置文件(profile)的 M2M GW 分组设备，并为设备确认树提供组证书；

[0049] 图 37 示出了 M2M GW P2P 分割确认；

[0050] 图 38 是可以在其中执行一个或多个公开的实施方式的示例性通信系统的系统图；

[0051] 图 39 是可在图 38 中示出的通信系统中使用的示例性无线发射 / 接收单元(WTRU)的系统图 ;以及

[0052] 图 40 是可在图 38 中示出的通信系统中使用的示例性无线电接入网和示例性核心网的系统图。

具体实施方式

[0053] 下文提到时，术语“无线发射 / 接收单元(WTRU)”包括但不限于用户设备(UE)、移动站、固定或移动用户单元、寻呼机、蜂窝电话、个人数字助理(PDA)、计算机或能够在无线环境中运行的任意其它类型的设备。下文提到时，术语“基站”包括但不限于节点 B、站点控制器、接入点(AP)或能够在无线环境中运行的任意其它类型的接口设备。

[0054] 本文公开了用于生成验证数据的各种技术，包括一种用于生成可用于确认无线发射 - 接收单元(WTRU)的验证数据的方法。WTRU 可以具有一个或多个组件和一具有多个安全寄存器的安全环境。安全环境可以包括提供安全执行环境的安全硬件和 / 或软件环境。例如，安全环境可以是可信平台模块(TPM)、智能卡、通用集成电路卡(UICC)或其任意组

合。安全环境可以用于保护诸如例如加密功能的安全功能、诸如例如操作寄存器的安全资源、存储器、随机数生成器、定时器和 / 或时钟。

[0055] 根据一个实施方式，可以通过对于 WTRU 的多个组件中的每一个获得代表 WTRU 组件的测量的值来生成验证数据。可以生成一个测量日志（ML），其包含组件测量值的记录，其它的组件专用数据可以存储在 WTRU 上。对于每个组件可以根据组件测量值生成验证数据，验证数据可以存储在可信平台模块内的一个或多个安全寄存器中。验证数据和 ML 可以被组织成树结构。包含验证数据的安全寄存器可以定义树结构的根。ML 可以定义树结构的内部节点，包含在 ML 中的测量值可以定义树结构的叶。可以使用安全环境的安全扩展操作来形成树结构。

[0056] 根据另一实施方式，可以获得代表 WTRU 组件的测量的值。可以根据测量值生成验证数据，验证数据可以存储在 WTRU 上安全环境内的寄存器中。测量值可以存储在树结构中的一叶节点处。可以在安全环境中执行一个或多个扩展操作将存储在叶节点中的值扩展到所述树结构的根节点。根节点可以包括安全寄存器中的数据，生成的验证数据存储在该安全寄存器中。

[0057] 根据另一实施方式，描述了一种对由无线发射 / 接收单元（WTRU）生成的树形验证数据进行确认的方法。树形验证数据可以包括组织为树结构的验证数据元、测量日志（ML）和组件测量值。验证数据元可以定义树结构的根节点。ML 可以定义树结构的内部节点。组件测量值可以定义树结构的叶节点。树形验证数据可以在组织的树结构中接收。从接收的树形验证数据的根处的验证数据元开始，对树结构进行遍历。作为遍历树结构的一部分，在接收的树结构的分支节点和分支节点的孩子节点处的值可以与参考树中相同节点位置处的值进行比较。然后可以基于节点值的比较确定是对 WTRU 还是对 WTRU 的单个组件进行确认。

[0058] 根据另一实施方式，描述了一种方法，用于证实由无线发射 / 接收单元（WTRU）生成的测量日志（ML）的节点值。ML 的值可以存储为包括根节点、内部节点和叶节点的树结构的节点。可以接收证明分组，其指示将由子树证书授权（SCA）证实的节点值。节点值可以被识别为能够由 SCA 证实的节点值。可以创建与节点值相关联的清单，其包括与节点值相关联的确认信息。可以创建用于节点值的证书，其被配置为将确认信息绑定到 WTRU 的安全环境。证书可以随着清单一起发布，并提供给 WTRU 的安全环境，WTRU 将证书存储在其 ML 中。

[0059] 结构化的确认是一种确认方法，其中确认的数据和操作方面被结构化。本文描述了结构化确认的分离但相关的概念和方法。例如，本文描述了树形确认（TFV），专注于使用子树证实、TFV 的扩展和变体以及分割确认的方法，其中将确认任务分布到两个或更多个网络实体之间，其允许网络实体以分布方式执行对于（所连接的）设备的设备完整性确认，从而每个确认实体可以不必要必须对整个设备进行确认而可以确认其一部分。

[0060] 图 1 示出了长期演进（LTE）无线通信系统 / 接入网 400，包括一个演进的通用陆地无线电接入网（E-UTRAN）605。该 E-UTRAN 605 包括一个 WTRU 610 和多个演进的节点 B（eNB）620。WTRU 610 与 eNB 620 进行通信。eNB 620 使用 X2 接口彼此连接。eNB 620 的每个通过 S1 接口与移动性管理实体（MME）/ 服务网关（S-GW）630 进行连接。尽管在图 1 中示出了单个 WTRU 610 和三个 eNB 620，但是应当清楚无线和有线设备的任意组合都可以包含在无线通信系统接入网 600 中。

[0061] 图 2 是 LTE 无线通信系统 500 的示例性框图，包括 WTRU 610、eNB 620 和 MME/S-GW 630。如图 2 所示，WTRU 610、eNB 620 和 MME/S-GW 630 被配置为执行用于向设备提供安全性的方法。

[0062] 除了在典型的 WTRU 中找到的组件外，该 WTRU 610 还包括具有可选的链接存储器 722 的处理器 716、至少一个收发信机 714、可选的电池 720 以及天线 718。处理器 716 配置为执行用于向设备提供安全性的方法。

[0063] 收发信机 714 与处理器 716 和天线 718 进行通信以便于无线通信的传输和接收。在 WTRU 610 中使用电池 720 的情况下，电池 720 给收发信机 714 和处理器 716 供电。

[0064] 除了在典型的 eNB 中找到的组件外，eNB 620 还包括具有可选的链接存储器 715 的处理器 717、收发信机 719 和天线 721。处理器 717 被配置为执行向设备提供安全性的方法。收发信机 719 与处理器 717 和天线 721 进行通信以便于无线通信的传输和接收。eNB 620 连接到移动性管理实体 / 服务网关（MME/S-GW），所述 MME/S-GW 630 包括具有可选链接存储器 734 的处理器 733。

[0065] 通常，为了保证设备完整性，执行软件 / 固件或硬件的测量（诸如，例如加密哈希值）并与可信的参考值相比较。测量的这种比较或者针对可信参考值的这种测量（称为验证数据）的函数或分组可以在设备内部（自主确认）执行或在外部由确认实体（半自主或远程确认）完成。在半自主或远程确认的情况下，测量可以作为未组织的集合在净荷（payload）中被发送，该净荷可以是经过加密、完整性保护和加密证实过的。

[0066] 为了找到完整性验证失败的组件，可以将测量的集合与参考值的集合进行比较，得到完整性测量失败的索引集合。然而，如果这种测量的数量很大，这种未组织的测量集合可能难于管理。

[0067] 为了优化完整性检查失败的模块的搜索，以测量日志（例如存储的测量日志（SML））的哈希树的形式生成验证数据。由可信计算组（TCG）架构和规范使用的术语 SML，可以用在描述本文描述的测量日志的各种实施方式中，但是 SML 是测量日志的一种示意性实施方式。组织的一个示例是树形确认（TFV）。存储的测量日志可以被组织成平衡的二叉树，且可以提出一种算法来使用 TPM 命令生成树。算法可以是通用的，并可以扩展到平衡的二叉树。由树形验证产生的 SML 可以在内容上与 TCG SML 类似，但是可以在通信中构造、形成、计算、使用、存储、获取、更新、删除、传送和 / 或接收，或者与如何处理和 / 或操作 TCG SML 不同地进行处理或操作。

[0068] TFV 可以在 TrE 中实现，例如下面描述的那样。这将引起许多其它用例类型的创新，且是用于 TrE 针对的复杂类型的网络侧平台确认的参考使能者（enabler）。本文描述 TFV 的技术内容。本文描述 TFV 指向（pointing）的概要。TFV 还被放入 TrE 的上下文中，其应用在本文中描述。

[0069] TFV 的一个元素（element）是验证数据的分层结构，其为远程确认器（validator）的平台确认带来各种好处。本文描述用于创建和管理树形验证数据的算法。此外，对平台的子结构（即验证数据的子树）的证明可以用于确认 TFV 中的平台。该层级中最高层的验证数据，称为根（集），可以诸如，例如通过具有特殊的（例如，硬件）保护的验证数据寄存器在 TFV 中被保护。TFV 可以通过小型有效的算法对树形验证数据进行操作。TFV 的全部实施可以在 TCB，或甚至是硬件保护的安全执行环境中实现。确认数据的复杂结构可以与树形验证数

据的树状层次(hierarchy)相关联和/或由其保护。子树可以通过结构-子结构的关系向验证和确认数据提供固有语义(semantics)。因此子树可以向确认器识别平台的功能部分。

[0070] TFV 的一个实施方式可以使用经修改的 TPM。可信计算作为现有技术针对 TFV 的实现和/或标准化是潜在的入口点。TFV 还容易从树(诸如二叉树上)概括出具有子结构的更一般的结构。

[0071] 下面描述的是可以由 TFV 提供的元素。

[0072] 安全性:在 TFV 中通过保护树形验证数据的根来提供硬件保护。参考保护等级是 TPM 中 PCR 的保护等级,显示为可以由 TFV 维护。平台侧的小型算法和低复杂度使得能够在小型 TCB 或片上实现。

[0073] 管理:TFV 包括安全可靠地挑选出平台的子结构并基于这种子结构的收集对该平台进行管理的方法和装置。用 TFV 子树表示的模块可以灵活地进行改变、更新或甚至是在平台之间移动,具有任意情况所要求的安全特性。

[0074] 分布:TFV 的层次允许实体间确认的分层分割。这使得通信场景和用例的设计更加灵活,并有利于效率。

[0075] 效率:用于搜索的一个有效数据结构——二叉树——可以在 TFV 中实现。它表明,例如,对失败组件(具有不想要的完整性测量值)的搜索在 TFV 中可以比在 TCG 类平台证明中更有效率。由 TFV 引入的天然层次提供了确认过程中负荷分布的选择。

[0076] TFV 的一个示意性特征是 TFV 可以根据其内置的分层顺序进行设计,用于由“更核心的”实体对大量低能力平台的确认。因此,TFV 可以适用于连接到网关和/或经由网关连接到网络的设备。一个示意性用例可以包括 M2M 通信场景。TFV 的这个特性可以提供给它一个与许多现有概念正交的概念,以向可信平台进行的状态证明提供更多语义。平台证明的其它方法显示了完全相反的理念。它们适用于能够产生复杂确认数据的平台,但是没有过多假设其层次化、模块化的结构。TFV 可以与其它正交的方法,诸如 PBA、语义证明、虚拟化和/或 HIM 进行结合。

[0077] 下面给出本说明书中用到的词汇表和缩略词列表。

[0078] RIM 参考完整性度量提供了实际测量数据可以与之比较的参考值。RIM 是,为了确认,由设备提供的测量值的副本(counterpart)。它们作为期望目标值与报告的测量值比较的参考。RIM 在它们唯一与组件相关联的意义上提供完整性的证明,例如作为在安全测试设施中获得的组件代码的加密摘要值。在它们允许与测量值直接(确定性)比较的意义上它们是一种度量。

[0079] RIM 证书 RIM 证书包含用于特定组件的 RIM,由 TTP 签名。

[0080] AuV 自主确认。

[0081] IDS 入侵检测系统。

[0082] DoS 拒绝服务(攻击)。

[0083] PVM 平台确认和管理。PVE 进行的平台确认和 (e)HMS 进行的平台的 OTA 管理的组合。包括能够从该组合中得到的所有潜在组合功能。

[0084] DMS 设备管理系统。对家庭(e)节点 B 管理系统(HMS)的泛化记法(3GPP LTE)(TS 32.583,[4]),适用于一般设备,由 PVM 功能增强。

[0085] RIM 管理器/管理者 RIMman 管理确认数据库 V_DB 的实体。是唯一授权那样做的

实体，是唯一对数字证书(验证，生成)进行加密操作的实体。

[0086] SAV 半自主确认。PVM 基于的确认变量。

[0087] TCG 可信计算组。

[0088] TSS 可信软件栈。

[0089] TPM 可信平台模块。

[0090] TTP 可信第三方。

[0091] TCB 可信计算基础。不能在运行时间评估可信度从而必须无条件信任的系统的部分。

[0092] CN 运营商核心网。

[0093] SHO 选择的本地(home)运营商。

[0094] PKI 公钥基础设施。

[0095] PDP 策略决策点。

[0096] PEP 策略执行点。

[0097] D-H Diffie-Hellman。

[0098] TFV 树形确认。

[0099] 验证数据在安全启动过程中对内部验证的总的结果进行唯一识别的数据。最好的例子是链式哈希值形式的大量测量值的 PCR 值，即摘要。

[0100] 确认数据在区分验证数据时，确认数据是提交给另一方(确认器)的所有数据，并用于评估平台状态的可信度。

[0101] 确认确认数据提交给确认器，例如实现为根据 TCG 的远程证明，以及由确认器对其进行评估的过程恰当地称为确认。确认数据通常可以包括验证数据，诸如引用的验证数据寄存器(PCR)值。确认可以，除验证数据的加密验证外，包括策略评估和确认器进行的动作触发。

[0102] 验证数据寄存器针对未经授权的接入和改变，保护验证数据的硬件存储。

[0103] 对可信平台的信任关系的建立依赖于确认过程。确认允许外部实体基于提供的平台配置的证据对平台的预期行为建立信任。在确认机制，诸如远程证明中，可信平台显示在启动过程中产生的验证数据。这些数据可以是平台配置寄存器的硬件保护的值，包含所有加载或启动组件的嵌套测量值，例如哈希值。这些值可以由安全的扩展操作以线性顺序产生。确认器基于验证数据的线性顺序和相关联测量日志对组件的低粒度诊断可能是低效的。提供了一种产生树形验证数据的方法，其中组件测量值表示叶，受保护的寄存器表示根。使用有限数量的硬件保护寄存器和标准扩展操作示出该方法的功能。如此，可以维持验证数据的安全性，同时存储的测量日志可以始终组织成树。讨论了使用树形测量日志和验证数据确认平台的基本机制。

[0104] 对计算平台建立信任的过程可以按照一个独特的通用模式。在平台启动过程中，可以由平台上的受保护实体测量组件。例如，组件可以在被加载和 / 或执行之前被测量。信任链的生成对于可信计算系统是重要的概念。这个链可以无间隙地从系统根一直扩展到当前的系统状态，包括执行的指令和程序。可以要求每个组件在执行下面的组件之前对其进行测量和报告。直接继任者的测量可以防止测量和实际执行之间的代码的未经监视的执行。测量过程可以由测量的信任的根进行保护，可以例如通过计算代码和配置数据之上的

摘要值来实现。

[0105] 验证数据可以通过受保护的操作从测量值进行编译和 / 或存储在受保护的存储中。验证数据可以，诸如在完成安全启动后，唯一地识别平台状态。这些过程的实施方式可以被认证(authenticate)并且是可信计算组(TCG)指定的安全启动。认证的启动可以用于 PC 客户端。也可以使用移动平台的安全启动。两者的差异在于安全启动增加了本地验证和执行引擎，如果组件的测量值等于可信参考值，该引擎就使组件启动。

[0106] TCG 提出分别经由可信平台模块(TPM)和移动可信模块(MTM)的扩展操作，根据是组件代码和 / 或数据的哈希值的测量值计算验证数据。数据可以存储在平台配置寄存器(PCR)中。作为示例，根据规范的版本 1.1 可以存在最小 16 个 PCR，在 TPM 的版本 1.2 中可以识别至少 24 个 PCR。PCR 可以由授权的命令访问。扩展操作建立了线性顺序的、嵌套的哈希值链，类似于 Merkle-Damgard 变换，如下：

[0107] $$V_i \leftarrow V_i \Diamond m \ \underline{\text{def}} \ H(V_i \| m), \text{(式 1)}$$

[0108] 其中 V_i 代表验证数据寄存器(对于 PCR，i=0；：：：;23)，H 是防冲突的哈希函数(在 TPM 情况下的 SHA-1)，m=H (数据) 是测量值。因此，TCG 可信平台的验证数据可以防止 TPM 的保护功能和屏蔽能力的操作而保证安全。

[0109] 验证数据可以伴有更具表现力的测量值记录和 / 或存储的测量日志(SML)中的其它组件专用数据。在针对外部实体的确认中，验证和 / 或其它数据，诸如 SML，可以由平台签名并传递到确认器。确认器能够将平台的可信度评估到任意期望的颗粒度(granularity)，其由在确认过程中传递的总的信息限制。用于确认的示范实施方式可以由 TCG 在证明协议中定义。TCG 设想确认可以最终用于对可信平台采用纠正步骤(例如，根据第一网络或服务接入时)，如 TCG 的可信网络连接工作组所预想的那样。

[0110] 提供了一种在树(诸如，例如 Merkle 哈希树)中不同于 TCG 规范预见的线性顺序组织验证数据和 SML 的方法。从应用的观点来看，线性链式的验证数据的效率问题突出。将验证数据组织为树的主要安全性问题是使得它们的生成与同样提供的 TCG 规范的测量扩展操作一样安全。还提出了一种在一组有限的如实表示哈希树根节点的硬件保护的寄存器中生成验证数据的方法和算法。还示出了树状验证数据和 SML 如何能够有效率并有效地用于确认。还讨论了树形验证数据的实施选项和进行的实验。

[0111] 验证数据提供关于具有无条件安全性的系统状态的信息。例如，它们可以独立于 SML 而安全，其根据 TCG 标准，在平台上或者确认中(可能不是签名的证明数据的一部分)没有特别保护。签名的 PCR 值，即，验证数据本身，可以为 SML 提供隐含的完整性控制。为此，可以根据 SML 中的测量，通过折回(retrace)所有扩展操作，重新计算验证数据。

[0112] 在认证的启动中使用 PCR 值确保测量日志安全的 TCG 标准化的方法可以基于由 Schneier 和 Kelsey 引入的用于确保不可信机器上审计日志安全的技术。事实上，其可以是一个简化，因为哈希链的最后的元素保持在 PCR 中，同时 SML 可以包含测量值而不是哈希链的中间项。使用 TPM 的完整性测量可以在完整性测量架构(IMA)中实现，该 IMA 作为 Linux 内核模块以使用 TPM 测量完整性并生成线性 SML。

[0113] 由线性链式扩展操作产生的验证数据可以是用于平台的远程诊断和 / 或诸如组件方面纠正的先进管理的限定值。本质上，SML 的操作位置，通过在测量值扩展到 PCR 之前篡改测量值，或者在安全启动之后篡改 SML 本身，都不能肯定地进行定位。此外，具有数以

百计或千计被测组件的真实世界 SML 的空间复杂度可以使得其针对确认失败的组件(测量值不同于“好的”参考值的组件)以高昂代价进行筛选(sift)。为了检查代码和 / 或数据，可以有各种可用的加密校验和(checksum)函数，且它们可以要求保持对“正确”数据的校验和的完整性。对各种机器上有效版本中软件的集中式数据库的要求可能是一个重大的管理问题，需要一种有效的解决方案。进一步，连网设备的大规模部署，诸如，例如机器对机器通信情景所需要的，可能需要固体设备和网络侧平衡且有效的信任基础设施。松散连接到网络且半自主运行的设备的安全性要求高。行业考虑的情况可能带来对连接设备的远程完整性检验或确认的高级别的要求。本文描述的方法和装置可以用于使确认具表现力、有效和 / 或安全。

[0114] TCG 基础框架工作组的规范可以包括解决这个问题的方法，分层区分验证的组件和子组件。描述了表示平台结构的信任树(ToT)的概念和符号。ToT 的节点可以表示平台组件，从 TPM 直到应用，以信任和安全性声明(statement)进行注释。这可以用于对应当投入到平台的信任进行评估，或甚至根据一定的约束重新组织平台。

[0115] 另一个在其中信任仅仅是线性链的缺点变得迫在眉睫的技术领域是虚拟化技术。虚拟机可以在许多潜在的层上动态地生成和 / 或破坏，导致信任依赖性的树状动态结构。尽管该共同体可能已经承认可能需要结构化的确认数据来真正评估平台的可信度，这种树形数据层次与验证数据(PCR 值)的颗粒度关联可能是缺乏的。

[0116] 验证数据和 SML 可以被组织进二叉树结构。在这样的结构中，验证数据寄存器是根，SML 数据结构可以包括内部节点，叶可以是组件测量值。整个结构可以是树形的，诸如，例如代表 Merkle 哈希树的类别。该方法可以被概括到 n 元和任意的树。图 3 的圣诞树用于示出树形验证的一般概念。

[0117] 图 3 示出了树形 SML 和根据验证数据的一般结构。星星代表存储在验证数据寄存器中的树的根。组件(代码和 / 或数据)由叶处的小包(packet)表示。由活结(slipknot)表示组件的测量哈希值。内部节点(球)将验证信息向上游传输到根。线暗示针对确认遍历树，稍后更详细地解释。

[0118] 代表哈希树根节点的验证数据的安全生成可能提出一个问题。在正常扩展操作中，使用由组件的可信测量根(RoTM)采用的测量值和当前的验证数据寄存器值 V_i，且操作本身在硬件保护的 TPM 中执行。

[0119] 因此，特别地，在生成过程中不使用在 SML 中无保护的存储的测量。这对哈希树是不可能的，其中增加一个新的叶可能影响该树的 d-2 个内部节点，其中 d 是树的深度。挑战可能是在受限数量的硬件保护的寄存器(PCR)内生成树形验证数据，诸如，通过使用单个叶测量作为输入，并使用 TPM 扩展操作和其它 TPM 能力。

[0120] 根据系统创建和保护树形验证数据所需的最低要求，可以清楚下述说明中的方法和装置可以不受限于遵守 TCG 标准的平台和安全硬件单元。

[0121] 可以在加载之前和启动的同时执行程序的验证。也可以如本文描述的那样使用证明。代码认证是可信计算的目标之一。执行的代码可以由平台的安全启动进行保护。例如，硬件机制可以用于将信任引导(bootstrap)到标准 PC 硬件上具有安全协处理器(coprocessor)的主机中。可以应用可信平台。安全硬件可以包含在安全引导过程中。例如，如果检测到异常，安全协处理器可以停止启动过程。这认为引导 ROM 是安全的。为了确保

这一点，可以配置系统的地址空间，以使得启动向量（boot vector）和启动码（boot code）由安全协处理器直接提供或者启动 ROM 本身可以是一个安全硬件。

[0122] 无论如何，安全协处理器通过针对已知值检查软件签名对系统软件（OS 内核、系统相关的用户级软件）进行验证。在该过程中可以实现防篡改的代码。解决该问题的一个方法可以在硬件（例如 XOM（只执行存储器）处理器架构和建立在其上的 XOM 操作系统）中固定用于程序的信任。这可能不解决安全加载程序和 / 或证明外部实体的问题。AEGIS 在 PC 上使用安全启动。例如，AEGIS 使用签名的哈希来识别启动过程中的每个层，如 Terra 所做的，其用在虚拟机的证明末尾的完整证书链来证明加载的组件。

[0123] TCG 规范定义了双边远程证明，以通过验证二进制可执行文件来远程地验证平台的完整性。当执行代码被加载时可以测量执行代码。该测量可以存储在 PCR 中作为验证数据，TPM 可以通过用 TPM 保护的密钥对这些数据进行签名来对这些数据进行证明。验证器可以，在接收到这些度量时，决定平台是否可以被认为可信。因为配置可以被传送并验证，验证器可以知道机器的配置。此外，二进制证明公开了配置，因此提出了隐私风险。在不同的解决方案中，讨论了“特性”和“基于特性的证明”（PBA）。PBA 允许向验证器确保经验证的平台的安全性特性而不透露详细的配置数据。可信第三方（TTP）用于颁发证书，其将平台配置映射到可以在该配置中实现的该特性（尤其是期望 / 不期望的功能）。然后 TPM 可以使用零知识证明，向验证器证明这些特性，而不公开完整的配置。

[0124] PBA 将平台确认的基础设施问题转移到 TTP，与 TCG 的隐私证书授权（CA）类似，但是扩展了其作用。另一个替换方式由 Nexus OS 提出，其建立最小可信计算基础（TCB）以在用户空间和特权程序之间建立强大的隔离。Nexus 具有安全存储区域和监视及执行机来保护它们。一个应用可以将设备驱动器移动到用户空间。Nexus 的证明将描述性标签连接到被监视的程序，因此允许类似于 PBA 的表现，但是是系统固有的。PBA 概念以及 Nexus 方法都没有办法对由多个组件构成，进一步将被动态管理的复杂系统进行确认。两种方法都与现有的一种方法正交，并可以与其结合。

[0125] 分层完整性管理（HIM），提出了用于组件方面完整性测量和平台组件的策略启用管理的框架。组件和子组件通过依赖关系图（对此目的有用的最普通的结构）在 HIM 中相关。但是 HIM 的目的不在于远程平台确认，不保护 PCR 中的结构化的平台验证数据。而是，它保持测量一起处于全球组件配置寄存器（软件寄存器）表中。

[0126] 用于大型数据集的完整性保护的哈希树的一个应用，诸如，例如由 Merkle 引入的哈希树可以是 PKI 中的证书管理。这可以使用树结构，诸如 Merkle 树或经认证的搜索树产生 CA 的长期的责任性（accountability）。哈希树的使用可以扩展到用于数字数据的普通长期安全归档。哈希树可以用于运行时间存储保护。

[0127] 系统可以使用哈希树用于存储装置和 / 或存储器保护，并可以分成不可信存储装置和 TCB。运行在 TCB 上的程序可以使用哈希树维护存储在不可信存储装置上的数据的完整性，不可信存储装置可以是例如方便访问的大块存储装置（bulk store），在其中程序定期存储和加载不适用于 TCB 的数据。整棵树的根可以存储在固定大小的片上可信寄存器中，但是其它节点可以存储在主存储器或缓存中。哈希树的另一个用途可以包括示出它们如何支持无线传感器网络（WSN）中的分布式代码的认证。还是在 WSN 中，包括多个节点的数据聚合可以使用哈希树进行完整性保护。

[0128] 使验证数据可搜索的另一个实施方式可以包括经认证的仅附加的跳表，其可以是有序的链表，设计为通过采用“快捷方式（shortcut）”允许快速查找存储的数据元。

[0129] 然而，树可能更适合平台状态的确认，诸如，例如用于有效确定确认失败的叶处的组件的子集。本文描述了使用一组有限的防篡改验证数据寄存器从组件测量值生成树结构（例如二进制 Merkle 树）的系统、方法和装置。树结构可以使用 TPM 的能力生成，诸如，例如标准扩展操作。算法可以足够小以在 TCB 内，尤其是在片上执行。该方法的这部分可以提高哈希树的根的生成的安全性，其反过来可以向树节点提供更多的安全性。本文还描述了系统、方法和装置，用于利用通过通用 PCR 值和 SML 的增强的诊断能力开发用于有效确认的树结构来增加远程平台确认的安全性特性，同时从失败点搜索的树状结构的效率中受益。树结构数据的这种使用可以用于安全诊断、确认和 / 或证明。

[0130] 本文描述的系统、方法和装置可以使用有限数量的验证数据寄存器来安全地生成一个根验证值。TCG 指定的可信计算的具体实施方式的每个参考，诸如，例如 TPM 操作、PCR 和 / 或 SML，可以是本文描述的系统、方法和装置的实施中使用的示意性实施方式。算法和 / 或进程可以适用于具有其所使用的最低限度能力的每种安全技术。

[0131] 硬件保护的寄存器中的一个，$v \stackrel{def}{=} \{V_1, \ldots, V_n\}$，例如 PCR，可以包括最终树的根。该树可以是二进制的，保持算法紧凑并，例如，提供失败组件的细粒度检测。叶可以携带测量值，而内部节点可以保存在修改的 SML 中。SML 可以以支持确认数据的树结构的方式被修改，即，其可以不是测量值的线性列表，但是数据结构可以支持标准树操作和遍历。为了在平台确认过程中有效搜索，SML 可以支持增加新叶并保持边的关系。在深度 d 的树的叶处增加一个新的测量可能需要重新计算存储在 $V \in v$ 中的树根和 / 或该叶的减少的哈希树的 d-1 个内部节点。Merkle 树的“左”“右”边分别有天然着色（coloring），因为二进制扩展操作（1）是不可交换的。叶继承这种顺序，并从左到右地增加。二进制，d 数字表示叶 n，$0 \leqslant n \leqslant 2^d-1$，记为 $\langle n \rangle$，产生从叶到根的唯一路径上内部节点和边的自然（natural）坐标。就是说，第 k 个数字（从 MSB 计数，k=1，....，d），$\langle n \rangle_k=1$，分别通过 $\langle n \rangle_k=0$ 或 $\langle n \rangle_k=1$ 确定该路径上深度为 k-1 的节点是否分别由左、右边连接。

[0132] 在算法执行的过程中产生的每一个子树的根可以安全地存储在 $V \in v$ 中。如果存在两个具有相同深度 d’的子树（测量值是深度为 0 的子树），它们可以合并为单个的深度为 d’+1 的树。在使用合并操作中，两个保护子树根的 V 中的一个可以在合并操作后被释放。可以制定用于新到达的测量值的更新算法，从而寄存器 $V_1, \ldots, V_{d-1}$ 包含深度为 1，....，d-1 的“激活”子树的当前状态，因此 V_d 可以包含当前的全局（global）根值。

[0133] 此处“激活”描述为一个子树，其根等待与相同深度的子树合并完成。谨慎地制定从而使用实际的测量值、保护的寄存器和 / 或正常的扩展操作，且不涉及未经保护的存储空间。以 nil（无）表示深度为 d 的全二叉树的空节点。树的形成可以由图 4 中所示的算法 1 执行。

[0134] 算法 1 中涉及的各种操作包括：

[0135] M 向 V_d 增加测量；$V_d \leftarrow m$。

[0136] S_V 存储验证数据寄存器到 SML；$V_k \rightarrow$ SML。

[0137] S_m 存储测量到 SML；$m \rightarrow$ SML。

[0138] V 复制验证数据寄存器；$V_k \rightarrow V_{k+1}$。

[0139] E1 以测量扩展 V_d ;$V_d \leftarrow V_d \diamond m$。

[0140] E2 扩展内部节点寄存器 ;$V_k \leftarrow V_k \diamond V_{k+1}$。

[0141] 上述符号可互换地表示操作及其执行次数。一个丢失操作 m ← RoTM 可以被归入 Sm。

[0142] 如果 $n<2^d$,树可能在右边处残缺,可以执行图 6 中所示的算法 2 中示出的清除过程。

[0143] 算法 2 (图 6)可以导致根的最终合并,从而 V_1 最终包含所有的子树信息。由于图 4 中所示的算法 1 的第 17-21 行的测试,如果该树还没满,那么可以进行清除过程。完成数所根据的规则是图 5 所示配置在右边处正确。

[0144] 内部节点可以写到 SML,即使它们是沿着左边的结果(带来稍许的冗余)。形式上,上述完成树所根据的规则可以解释为修改“◇”操作的概念,使得 $x \diamond nil=x$,如此处解释的。

[0145] 如果叶和内部节点以图 4 所示算法 1 规定的顺序添加到 SML,可以获得最后得到的树的天然顺序化(serialization)。对于深度为 3 的不完整树该顺序如图 7 所示。

[0146] 在图 7 中,得到的 SML 中标记的项 10 和 11 相同,这是由于 11 是由清除算法 2 的前向操作创建的。二进制搜索可以使用 SML 顺序为 SML 中的树节点定址。给定长度为 $2^{d+1}-1$ 的 SML 中的序列号 K,这样一个搜索从最后一个项——根进行。其余 $2^{d+1}-2$ 个项同等地划分为大小为 2^d-1 的部分,且决定 K 是在左边部分还是右边部分。重复这个过程直到 K 指向当前部分的最右的元素(element)。做出决定的顺序产生了 SML 中从根引到具有索引 K 的节点的左 - 右边的顺序。

[0147] 图 4 的树形成算法 1 可以容易地适应任意、统一的元数(函数或操作的元数是该函数使用的自变量(argument)或操作数的个数),诸如 b 的树。为此,二进制坐标 <n> 必须由 b 元坐标 $\langle n \rangle^{(b)}$ 以及其分别在图 4 所示算法 1 的第 4、12 行评估的第 d 和第 k 位数字代替,其中评估的表达式必须变为 $\langle n \rangle \frac{(b)}{d} = b-1$。

[0148] 算法 2 (图 6)可以相应地调整。进一步推广到任意树可能需要建立关联节点坐标,即,n →节点的映射。在每个元数超过 2 的节点,由于对于连接到每个节点的支线(leg)的哈希扩展是线性的,因此可能会有上述提到的缺点,且可能出现检测粒度的损失。

[0149] 从生成过程可以清楚,可以以有限数量 V_1, ... V_r 的验证数据寄存器覆盖树的叶处的有限数量的组件。最大容量可以计算如下。用于第一寄存器 V_1 的过程可以使用 r-1 个其它寄存器作为长度为 r-1 的管道来建立深度为 r 的树。当 V_1 被占用时,第二寄存器可以支持深度为 r-1 的树,等等,直到最后一个寄存器 V_r,针对该 V_r 管道的深度为 0,树的深度为 1。因此寄存器的树携带的叶的总的数量可以这样给出 :

[0150] $$C_{trees} \sum_{k=1}^{r} 2^k = 2^{r+1} - 2 \text{(式 2)}$$

[0151] 对于附属于 v 1.2 规范的 TPM 的 PCR 数量 r=24,这产生 33,554,430 个位置用于 r 个树的叶处的组件测量。如果限制为最后 16 个 PCR,因为,例如,根据 TCG 的 PC 客户端规范,PCR 0-7 是预留的,规范还要计数 131,070 个测量。尽管这个容量很高,但不是无限的,因为标准是线性可扩展的。因此,由于在启动过程中或者在运行时间进行的测量的数量不

是预先已知的，所以最后一个寄存器可以，作为后备（fallback），在达到容量限制后进行线性扩展。图 8 示出了这种安排——显示树的验证数据的最大容量安排。在图 8 中，叶处的测量值表示为 m。

[0152] 树形成算法的空间复杂度很小。因为内部数据明确需要三个 :d ∈ {1，....r}，n ∈ {0，...2^d-1}，以及 k ∈ {1，....d}，数据的大小最大为$d+2\lceil \log_2 d \rceil \le r+2\lceil \log_2 r \rceil d$个比特。

[0153] 另外，根据实施，一个寄存器可能被需要来接收并保持当前的测量值，和 / 或作为用于对验证数据寄存器操作的中间寄存器。SML 适度增加了尺寸。对于深度为 d 的完全充满的二叉树，2^{d+1}-2 个节点值，包括叶测量，被存储在 SML 中（根节点被包含在 V_i 中）。也就是说，树形 SML 小于仅包含测量值的线性形成的 SML 的大小的双倍。

[0154] 对于时间复杂度的估计，考虑深度为 d 的完整树，即，2^d 个叶测量。根据树的结构，可以对操作的发生进行计数。S_m 在每个叶处发生，即，2^d 次。E1 和 M 在深度为 d-1 的每个内部节点处发生，即 2^{d-1} 次。V 和 E2 在深度 d-2 以上的每个内部节点处发生，即 2^{d-1}-1 次。最后，S_v 在树中除根以外的每个内部节点处发生，根保持在 V_1 中。也就是说，S_v 发生 2^d-2 次。这样一共产生这样的估计 :不管流控制，对于算法执行时间为 $2^{d-1}(E1+M)+(2^{d-1}-1)(V+E_2)+2^dS_m+(2^d-2)S_v$。对类似的操作进行分组 $\{E_1,E_2\}$，$\{M,S_v,S_m\}$ 产生 $2^{d-1}(E_1+E_2)-E_2+2^{d-1}(M+2S_V+2S_m)-2S_V+(2^{d-1}-1)V$。

[0155] 假设存储器操作在耗时上近似相等，并以通用常数$M \approx S_v \approx \frac{1}{2}S_m \approx \frac{1}{2}V \le S$为边界，其中因子 2 包含在 V 中用于简单的读取 / 存储实施，在 S_m 中用于上述提到的丢失操作，同样也用于扩展操作 $E_1 \approx E_2 \leqslant E$，对于 d>1 的树形成的时间复杂度的粗估计由$\le 2^d\left(E+4\frac{1}{2}S\right)-(E+4S)$给出。当扩展操作是主导因素时，树形成可能需要小于经认证的启动的线性链的一个扩展操作。

[0156] 对于由上述描述的过程生成的树形验证数据的确认，描述了在每个树节点处使用可用信息的确认策略。平均计算成本可以分别关于失败的测量的相对分担（share）、数量进行计算。

[0157] 采用在普通经认证的启动中生成并存储并随后扩展到 PCR 的测量的线性链作为参考情况，可以看出树的遍历确认是显著不同的。在前者的情况下，SML 的操作原则上可以不本地化，而遍历树形 SML 可以允许识别在其中发生了操作的子树。同样的考虑适用于诊断确认，即，搜索不符合经确认的平台的期望参考配置的组件（本文描述为失败组件）。对于线性链式 SML，这可能需要比较每个测量和参考值和 / 或重新计算直到 PCR 的扩展操作链以验证 SML 的完整性。由于线性 SML 中的操作可以不本地化，那么复制 PCR 值的失败可能意味着诊断确认是不可能，可能不能将失败的组件与好的组件区分开。

[0158] 对于树形 SML，情况更好一些。如果识别了一个子树，其中 SML 的操作被怀疑，那么 SML 树中它的补集（complement）仍然需要确认。而且，对于诊断确认，可以期望在确定失败组件的集合并同时验证根验证数据寄存器的内容上具有显著的加速。

[0159] 在可能的情况下，树形 SML 的确认可以用于找到无法确认的叶的子集和 / 或检测 SML 的操作。可以假设在确认器处存在用于本地可用比较的参考树。确认可以从树的根，即

验证数据元 V 开始，遍历 SML 数据的树。这可以生成组件的叶集合，针对该组件的测量不同于参考值，该组件称为失败组件。在遍历树的过程中，可以应用具有修剪的深度优先搜索，在每个分支节点做决定。树可以是二进制的。分支节点及其两个孩子处的 SML 树值可以与相同节点位置的参考树值进行比较，结果可以记为 g（good）用于一致和 / 或 b（bad）用于不一致。在这种表示法中，可能出现下述的情况，如图 9 中所示。图 9 示出了树形 SML 中节点配置的分类。

[0160] 在图 9 中，情况（a），该父节点以下的整个子树可能都被肯定地确认，遍历在该节点结束。在图 9 中，情况（b），父节点可能由确认器将扩展操作应用到孩子节点值来重新计算。如果重新计算的值与该父节点处的值不匹配，那么这可以表明子树之一的 SML 操作具有标记为坏的根。这可以作为异常处理。

[0161] 否则，确认可以继续到下一级树，分别遍历发现坏值的子树，即，（b）中的左、右或两个子树。情况（c），可以检测到树操作异常。这种检测可以不对扩展操作进行重新计算就发生。最后一种情况（d），可以在二叉树不完整和 / 或右侧树枝是空的时发生。那么值 x 与值 y 相等，在该情况下遍历可以进行到左侧，否则可能发生树操作异常。

[0162] 确认树形 SML 的一个优点是具有正确根的子树可以从对失败组件的进一步搜索中丢弃。现在提供一种对树确认的性能进行量化评估的简单概率模型。假设，例如，SML 是深度为 d 的完整树。确认器具有表示已知期望平台配置的完整参考树。重新计算哈希操作可以用于估计确认复杂度的主要成本因素，而比较是廉价的。假设一组随机的失败叶。

[0163] 可以使用一种乐观的确认策略，称为诊断确认，其遍历从根到失败组件（即相对于参考树的叶具有坏测量值的组件）的路径。这种策略的一个特性就是它发现具有真实测量值的失败组件。诊断确认可以如下进行。当访问不同于参考树中相应节点的内部父节点，即坏的父节点时，可以遇到图 9 中的一种情况，情况（b），或者情况（c）的最右边的配置。在后一种情况下，可以不执行对父节点的重新计算，因为很明显 SML 完整性是失败的。具有这个根配置的子树可以从进一步遍历中丢弃，因为其不能产生关于失败组件的可信信息。这种情况下，进一步的步骤可以依赖于确认器的策略。情况（b）中的节点配置是这样的，可能需要通过一个扩展操作 0 根据根哈希重新计算父哈希，来肯定从确认器的参考树不能得知的这个配置是真实的。其根是在仔细检查下坏的父节点的好的孩子节点的子树可以从进一步的遍历中丢弃。诊断确认的这个过程隐含地从诊断确认中排除了图 9 的配置 / 情况（a）以及三个左边的配置。只要有意义，可以在 SML 树的进一步取证（forensic）评估中被考虑。

[0164] 诊断确认可能需要一个访问，并在从失败（坏）叶到根的路径联合（union）中的坏内部节点处执行哈希操作。在相反的未经篡改的树中，这可以隐含地排除了具有坏的父节点的右侧配置 / 情况（c）。独立且理想分布（i.i.d.）的坏叶的子集构成叶的分数 $f \in [0, 1]$。坏叶的数量是 $2^d f$。可以如下面解释的那样计算坏的内部节点的期望数量 $E^{inner(内部)}(f)$。

[0165] 本文解决的一个问题是对由叶的随机 i.i.d. 选择生成的二叉树进行双向着色（例如，坏的对（vs.）好的内部节点），以及对将它连接到根的路径进行着色。这种叶的随机选择和路径可以相当于长度为 d 的 i.i.d. 的比特串的随机选择。对从 $N=2^d$ 个叶的集合中进行 k 次选择后被着色的叶的期望数量 E_k^N 进行计算。以递归方式，$E_0^N=0$，且 $E_{k+1}^N = E_k^N \frac{E_k^N}{N} + \left(E_k^N + 1\right)\frac{1-E_k^N}{N} = 1 + E_k^N \frac{E_k^N}{N}$。对其求解得到：$E_k^N = N\left(1-\left(1-N^{-1}\right)^k\right)$。

[0166] 由于选中的比特串的所有子串在统计上是独立的，在等级 d-1，....0 的内部节点应用相同的自变量。因此，着色的内部节点的期望数量由合计 d-1 获得$E_k^{inner}=\sum_{l=0}^{d-1}E_k^{2^l}$。其余要找到的是选择 k 的期望数量，对应于着色的叶的某一期望数量$E_k^N=fN$，其中 0 ≤ f ≤ 1 是叶的目标分数。对于 k 求解该等式，产生$k=\frac{\ln(1-f)}{\ln(1-2^{-d})}$，其中插入了 N = 2^d。由此，可以计算依赖于 f 的坏的内部节点的期望数量，$E_{inner}(f)$。

[0167] 图 10 示出了 2^d-1 个内部节点的分数，其中 d=16，根据上述描述可以在其中进行哈希操作。这代表肯定确定坏的组件所需要的哈希操作的数量。线性 SML 的参考情况可能需要 2^d+1 次哈希操作来重新计算最终的 PCR 值。这种情况大致由图 10 的上纵轴表示。

[0168] 关于与参考值的比较，情况可能稍有不同。用于诊断确认的树遍历可以随着与参考树的相应内部节点的比较失败的坏的内部节点传下来。因为这样，在每种情况下坏的节点的两个孩子节点可以进行比较，从而在比较方面的复杂度可以是数量$E^{inner}(f)$的两倍。线性 SML 可能需要全部 2^d 个测量与参考值进行比较。

[0169] 如果 h 是确认器处哈希操作的代价，c 是两个哈希值进行比较的代价（对于 SHA-1 的 160 比特），那么线性情况下总的确认代价是 $(2^d+1)h+2^dc=2^d(h+c)+h>2^d(h+c)$。这以最小的努力从线性 SML 获得与通过树形 SML 的诊断确认相同的信息。另一方面（计数中包括根）对于树形 SML，代价是（$E^{inner}(f)+1$）（2c+h）。如果$\frac{E^{inner}(f)+1}{2^d}\le\frac{h+c}{h+2c}=\frac{\lambda+1}{2\lambda+1}$，其中 λ =c/h<<1，树形确认更加有效率。即使具有很大的余量，λ <0.01，可以对 r.h.s（右手边）产生 0.99 的边界。那么对于 d=16，树形确认对于坏叶的分数高达 85% 可以更有效率。

[0170] 树形 SML 的诊断确认在哈希操作方面可以比具有线性 SML 的执行得更好，甚至针对大分数的坏组件优于线性 SML。树形 SML 的诊断确认对于小分数的失败组件是大大有利的。当坏叶不均匀分布，例如表现为成簇时，树确认可能更有效率。

[0171] 直接比较的线性和诊断树确认都可用时，如果最终的 PCR 重新计算失败，则线性确认是不可能的，自此以后，单个测量的比较不产生可靠信息——每个测量可以被假装（fake）在 SML 中以隐藏打断哈希链的那个测量。树形的一个优点是甚至在用于确认器的计算复杂性下降时也可以产生确认数据。

[0172] 对于树形成算法本身，为了达到与 TCG 标准兼容的可信启动过程相同等级的安全性，验证数据寄存器上的操作可以在硬件保护的 TPM 环境内运行。尽管在上述列出的树形成算法的大多数操作中的部分操作不是可以在标准相容的 PCR 上执行的标准 TPM 函数；实际上，正常扩展操作 E_1 可以是内部的标准函数，S_v 和 S_m 可以由 PCR 读操作实现。

[0173] 讨论了将 TPM 扩展到把 PCR 变成树形验证数据寄存器所需的最小修改，而树形成算法可以在 TPM 外部运行。接着，描述了用于树形成的 TPM 内部命令。描述的另一个实施是树形验证数据的基于软件的实施，其中根寄存器可以是可信应用管理的软寄存器，这种寄存器的当前状态由“真正的”寄存器，例如 PCR 进行保护。最后，描述了具有整合在 TPM 软件模拟环境中的 TPM 模拟器的树形成的试验实施，“ethemba”。

[0174] 采用最低要求方法来实现树形成并开创对标准 TPM 的改变，使得 PCR 能够与算法 1 和 2 使用。这种方法考虑实施上述由 TPM 命令或其修改列出的基本操作。算法的核心，包

括代表内部节点当前状态的寄存器中的簿记任务，可以实现为信任的软件根，用于在系统完整性测量过程，诸如认证的或安全启动中执行树形成。

[0175] 操作 S_v 和 S_m 不产生问题，并可以分别由 TPM_PCR 读命令实现，或者直接在树形成软件中实现。E_i 可以在树的最低等级的每个右边发生，并可以对包含来自扩展到 V 中测量的左侧兄弟（sibling）的已经测量的值的 V 进行扩展。因此，E_1 可以包括在由式（1）定义的标准 TPM_Extend 操作中。E_2 可以在树内的右边发生，反过来，可以由后跟 TPM_Extend 的 TPM_PCRRead 建模。

[0176] 操作 M 和 V 可以分别在最低等级的左边、树内发生。它们由于两个原因而提出问题。首先，PCR 可以不被直接写入，作为 M 或 V 中第一步骤的通过 TPM_PCR_Reset 复位它们的自然方法可能是有问题的，因为标准 TPM 的仅 16 个以上 PCR 可以被复位，且仅能从正确的位置复位。因此有必要足够的 PCR 可复位，且它们对树形成软件作为可信代码执行的位置做出响应。

[0177] 其次，即使在复位之后，可以修改 PCR、TPM_Extend 的操作也不能直接将值拷贝到寄存器中，而是以复位 PCR 的现有值真正地执行（1），该值是 160 比特二进制 0x00 和输入值，其产生不同于输入值的结果。避免暴露直接写入的新命令或者在 PCR 之间移位（shift）值的一个选项可以是以指明它们在复位之后处于原始状态的复位标记来增大 PCR。那么，可以修改 TPM_Extend，从而在该标记为真时直接将 TPM_Extend 写入到 PCR，然后将该标记设置为假。

[0178] 意识到 M 和 V 始终发生在树的左侧边，如果右边的兄弟为空（nil），那么根据涉及的两个兄弟确定地产生输出，第三选项将稍微偏离 Merkle 哈希树的定义。接着可以在图 11 中示出 SML 树中每个基本三角中的值的正确配置。

[0179] 也就是 V 或 M 可以由后跟 TPM_Extend 的 TPM_PCR_Reset 进行建模，以在第一步骤中获得 $0 \diamond x=H(0||x)$。接着右侧兄弟可以在那个寄存器中正常扩展，结果写到 SML。下面还描述了树的中间阶段和最终确定中对 nil 节点值一致的处理方式。

[0180] 在许多情况下，存储在 SML 中的哈希树可能是不完整的，即，包含空的叶和内部节点。在连续测量过程中，这种节点，具有表示为 nil 的值，可以由操作 M 和 V 程序化地处理，意味着可以忽略右侧的 nil 兄弟。这发生在用于树形成的中间阶段的算法 1 的第 8 和 18 行，以及在最后测量之后的树的完成的算法 2 的第 29 行。

[0181] 通常，即，越过标准 TPM 的约束，nil 可以是用于操作 $\diamond$ 的双侧单元，即，$x \diamond nil=nil \diamond x=x$，且 $nil \diamond nil=nil$ （式 3）

[0182] 这种约定表明了如上所述的规则 / 情况（d）。它是对通常扩展操作的重新解释，也可以用于消除算法制定中的操作 M 和 V。也就是，M 和 V 可以由寄存器 V 复位到 nil 替换，后面分别跟着操作 $V \leftarrow V \diamond m$，$V \leftarrow V \diamond V'$。

[0183] 对于这种约定的实施，nil 可以表示为 PCR 寄存器的额外标记，以及 $\diamond$ 的输入和输出。对于 PCR，nil 标记可以由特定复位命令设置。当遇到作为扩展操作对 PCR 的输入的 nil 时，那么 TSS 或者 TPM 修改的逻辑可以防止哈希操作（1）的执行，并直接写到 PCR。

[0184] 树形成的分割的 TPM/ 软件实施在产生的根验证数据寄存器值的安全性等级方面妥协（compromise）。树形验证数据可以由提出的算法的 TPM 内部实施产生。为此，TPM 修改可以如下运行。修改的 TPM 可以揭示具有与通常 TPM 扩展命令相同输入参数的命令 TPM_

Tree_Extend。TPM 可以维持用于 PCR 的标记，表明它们中的哪些是当前指定的树根、哪些被占用和锁定以及哪些可以由算法用做中间 V。而且，TPM 维护上述提到的附加数据。最简单的情况下，内部逻辑可以防止并行使用多个 PCR 用于树形成。当 TPM_Extend 可以输出目标 PCR 值的更新，TPM_Tree_Extend 可以按顺序返回更新后的验证寄存器数据的变量号 1,.....d，从而它们产生上述的自然顺序。这些返回值可以是算法 1 和 2 的 SML 写操作的输出。当返回了 d 个值时，接收机可以知道这棵树已经用尽，相应的根 V 被锁定。另一个选项可以包括 TPM_Tree_Extend 来在每个呼叫上返回所有的中间 V。

[0185] 描述了一种方法，用于以与具有 PCR 的传统做法相同的方式使用 Merkle 哈希树来保护可信平台的安全启动过程的完整性。已经显示了由于使用平台确认中树形验证数据得到的效率和灵活性增益。这可能是有效的，尤其在经由网络进行平台的远程确认和管理中。给定小规模和低复杂度的树形成算法，如果相应地修改了规范，那么该算法可以是 TPM 内直接的实施操作。这可以是用于未来 TPM 产生的可行方法。

[0186] 泛化方面，树当然不是最通用的结构，因此可以应用使用加密摘要的完整性保护。例如，某些研究人员已经将哈希扩展为提供有向图的识别。其它人应用变种单向函数，例如多重(multi-set)哈希，来唯一识别复杂数据结构，诸如 RDF 图。按照这些原则，可以设想将树形验证数据泛化到，例如有向无环图和依赖图。虽然可能对复杂的平台管理和保护任务感兴趣，每个这样的泛化将导致用于确认的二叉树的复杂度增加，并失去效率。因此在本阶段不进行这种泛化的应用情况。

[0187] 但是，上述提出的 TPM 完整性测量功能，TPM_Tree_Extend 的单命令扩展可以是灵活的基于 TPM 的树验证数据管理架构的起始点。子树根的安全更新可以被启用，诸如，例如用于动态平台管理，最终以与 TPM_Quote 提供给远程确认器用于 PCR 值的安全断言(assertion)相同的安全性断言引用树形 SML 的内部节点。下面描述这种进一步的扩展。

[0188] 结构化测量有助于识别不能通过完整性检查的组件。在结构化的一个这种实施方式中，测量可以以线性链组织，验证数据可以获得为 $:V_i=f(V_{i-1},m_i)$，i=0,1,...,n-1，其中 n 是线性链的长度，m_i 表示第 i 个测量，V_i 表示从线性链的第 i 个迭代获得的验证数据。通过处理组件的 n 个测量哈希值获得的最终值 V_{n-1} 可以用做验证数据。对于每个 V_i :i=0,1,...,n-1，存在一个可信的参考值 R_i。如果在验证过程中，最终值 V_{n-1} 指示为不同于其参考值 R_{n-1}，那么这表明至少一个中间值 V_k 与其参考值 R_k 不匹配。这样的一种不匹配可能是由于某个组件的完整性失败，或者仅仅由于正确计算的(和 / 或完整性完好的)验证数据 V_j 与损坏或错误的参考数据 R_j 之间比较的不匹配。由于测量日志为链式(V_0 到 V_{n-1})，验证数据与其参考值不匹配的首次出现可以使用图 12 中所示的算法 1 发现。存在 n 个元素，索引范围从 0 到 n-1，对索引操作执行整数除法。

[0189] 参考值 {Ri :i=0,1,...,n-1} 是可信且不可妥协的。这种情况下，任何不匹配表明第 i 个组件的完整性的失败，或者第 i 个组件本身可能是完整性完整的，但是其测量 m 可能本身是受损(compromised)的。在两种情况的任意一种情况下，在操作验证系统中，可以声称第 i 个组件是不可信的并可以在加载之前进行补救。算法 1 发现失败的首次出现，但是不能发现后续的失败。

[0190] 为了优化失败的完整性检查模块的搜索，可以以如上述的树形验证过程中描述的存储的测量日志(SML)的哈希树的形式生成验证数据。存储的测量日志被组织为平衡的二

叉树，且提出了使用 TPM 命令生成树的算法。算法是通用的，可以扩展到平衡的 n 元树，其中 n>2。

[0191] 将如上描述的树形验证的概念扩展到考虑树是不平衡的的树形验证。进一步它们可以被扩展到考虑软件组件之间的相互依赖。术语设备用于表明其软件 / 固件 / 硬件的完整性将被验证的任意实体。注意到实体可以是家庭网关、M2M 网关、接入点、H（e）NB、HNB、中继节点、移动设备、平板设备或任意其它网关或设备。

[0192] 不失一般性，与其它组件相比，某些软件组件可能受到更多的攻击。例如，在设备中，通信协议栈可能比实现装饰特性的组件更频繁地受到黑客攻击。这种论点还适用于功能。在远程确认和半自主确认中，完整性测量被发送到网络中的平台确认实体。这样的确认实体可以观察并保持失败的组件或功能的历史。

[0193] 而且，通常假设当设备（和 / 或其软件组件和 / 或功能）受到攻击和 / 或其完整性受损时，对设备的利益相关者的影响或成本可能根据组件 / 功能而变化。

[0194] 上述方面使组件 / 功能验证策略成为必要，其中攻击的频率和 / 或成本或影响的严重性可以在验证数据结构和验证算法的设计中考虑和反映。

[0195] 但是，首先，考虑攻击频率（或者发生的概率）不同的情况。基于软件组件或功能的完整性失败的频率，可以估计软件组件或功能上完整性失败的概率分布函数（PDF）并将其用于验证数据结构的构造中。概率分布是离散分布函数，且对于具有 n 个组件 / 功能的设备，可以表示为向量（P_0 到 P_{n-1}）。可以通过平均所有的观察或者在其中以最新足够丰富的样本执行平均的窗口平均来生成这样一个概率分布函数。另外，平均可以是非加权平均或者使用权重函数 W[n] 的加权平均。权重函数的例子有，但不限于，指数、高斯等。也可以使用重叠或非重叠滑动窗口获得在时间域变化的 PDF。这个概率分布向量（或多个向量）可以通过消息或配置被传递给设备。不失一般性，我们假设一组时间上固定的 PDF $\{P_0,\ldots,P_{n-1}\}$。

[0196] 在设备处，使用软件组件的分布（PDF）和哈希完整性测量值，可以使用 Huffman 算法、算术编码或类似这种最优编码树构造算法来构建最优哈希树。图 13 示出了使用 Huffman 代码的树结构的例子。构造这样一棵树的优点在于被攻击最频繁的组件具有更短的代码长度，从而减小了搜索时间的预期值。因此减小了搜索失败的组件或功能的开销。而且，预期将处于最频繁攻击下的组件将被最早地搜索。

[0197] 为了 Huffman 算法或算术编码算法起作用，概率不能为零。因此在构造树时可以丢弃攻击概率为零的那些组件或功能。这会得到缩小尺寸的树，减少了通信过载和搜索时间。可替换地，如果包含了所有的节点，那么可以将非常小的非零值 δ 分配给具有零攻击概率的组件或功能。通常，由于攻击的概率分布不均匀，结果树将不平衡。

[0198] 在网络中，这样的一棵树可以基于设备的生产时间设置或者在认证、授权且完整性可验证的软件 / 固件更新之后，由经证实的授权中心（authority）预先构造并填入导出的哈希值，并安全地存储以供参考。在该设备中，每一次设备安全启动，如果不是预先构造并存储的，这棵树可以通过应用最优代码树算法（诸如 Huffman 代码、算术代码）使用攻击频率的 PDF 重新构造，且节点和根用评估的哈希值（或扩展的 PCR 值）填充。

[0199] 算法可以发展到在设备处使用软件组件的测量填充树的节点和树根。然后该设备可以将这样一棵验证树发送到网络中的平台验证确认实体，并与参考树进行比较以建立设备的信任状态，并唯一识别失败的节点。本文描述了该算法。

[0200] 为了减少通信过载，可以发送特定深度为 d，包括根和深度为 d，d-1，...，0 的节点的修剪树。可以基于门限值评估深度 d。修剪的另一种方法是丢弃攻击出现的累积概率小于指定门限的任何节点。又一种可能更合理的方法可以是在已包含在树中的节点的累积分布值已经超过指定门限时对树的所有剩余节点进行修剪。图 14 示出了用于深度 d=2 的树修剪。

[0201] 图 15 示出了系统图和通信互换。图 15 示出了，在 1，基于攻击历史，生成组件 / 功能上的攻击的离散概率分布。在 2，Huffman 编码算法用于生成代码树。在 3，攻击的概率分布被传递给设备。在 4，Huffman 编码算法用于在 WTRU 处生成编码树。在 5，安全启动过程中，以信任值填充树。在 6，从 WTRU 发送信任树。在 7，用参考来验证该树并针对失败节点搜索该树。在 8，采取纠正措施。

[0202] 在以上描述中，通过考虑用于组件的非均匀攻击概率模型可以看出导致不平衡树的使用。除了攻击概率，还可以考虑任何攻击的成本或影响。假设设备的任意第 i 个组件（i=0，1，...，n-1）与 $\{P_i, C_i\}$ 相关联，其中 P_i 表示出现的概率，C_i 表示对利益相关者的成本或影响。那么，组件可以由“由于攻击 / 受损导致的预期成本（影响）”进行排序，其中通过下述比较执行排序：

[0203] $E(\text{cost})_i = P_i \times C_i$。接着可以根据 $Ef(\text{cost})_i \frac{E(\text{cost})_i}{\sum_{k=0}^{n-1} E(\text{cost})_k}$ 确定标准化的分数预期成本。

[0204] 一旦组件进行了排序（从最高标准化预期成本到最低），那么可以使用相同的 Huffman 编码或算术编码来形成树。以这种方式形成的验证树可以给出其失败对相关利益者可能具有最大预期成本 / 影响的组件的最大权重（用于搜索）。

[0205] 树结构本身可能不能完全捕获软件结构的相互依赖性。例如，可能存在由不同组件使用的共享库。因此为了指示这种相互依赖性，在构建了最初的最优代码树之后，在该树的节点和节点 / 叶之间增加链路。从而，树可以扩展到捕获设备的软件架构或者组件间的相互依赖性。树叶可以映射到软件组件，节点映射到库和文档。软件模块和共享库的相互依赖性可以由替代孩子链路映射。这样树就变为广义的树。以测量值填充节点的算法可以被修改为额外地遍历替代的孩子链路。图 16 示出具有这种替代的孩子链路的树，其中由模块 A 表示的软件组件或功能依赖于或使用由模块 D 代表的共享库，类似地，由模块 B 表示的软件组件或功能使用叶组件 / 功能 C。

[0206] 如果设备包括可信执行环境，诸如，例如 TrE，来保证算法的可信操作和 / 或节点和 / 或根以获得的哈希值进行可信填充，算法可以使用由 TrE 提供的可信操作和指令。

[0207] 如果使用可信平台模块（TPM）和上层可信软件栈（TSS）来构建这样一个 TrE，可以考虑用于 TPM 和 TSS 的适当命令扩展。

[0208] 如上所述，任何最优代码构造算法可以用于构造树。提出的一个示例是 Huffman 算法用于构造树。

[0209] 在下面的描述中，叶代表软件组件或功能。为简单起见，提到组件，但是该论证也可以适用于功能。避免冲突的哈希函数 H 可以用于计算组件的测量。使函数 F 用于通过使用所考虑的节点的孩子的度量计算节点的度量。可以调用（call）具有设定为最优代码树的根的节点的算法 Populate_metric（填充度量）。树的叶度量可以初始化为相应组件的测

量。

[0210] 图 17 中所示的算法 2 执行树的后排序遍历，并计算内部节点的度量，并以计算出的度量值填充节点。为了修改用于 PTM 的算法，函数 F 是 PCR 扩展操作。在下述的算法中，Reg_Number 是表示 PCR 寄存器号的下标索引。

[0211] 如图 18 所示的算法 3 是对算法 2（图 17）的修改，用于作用于 PCR 寄存器并使用 PCR_Extend 操作。算法 2 和 3，分别如图 17 和 18 所示，可以修改为作用于 n 元树。然而，相应地可以将 Huffman 算法修改为生成树。不是为左孩子和右孩子增加最低的 2 个概率，而是增加 n 个最低概率。相应地树枝（branch）的标签包含 log2（n）个比特。例如，对于 4 元树，增加了 4 个概率并将 4 个树枝标记为 00,01,10,11。如图 19 所示，算法 4 是对针对 n 元树的算法 2 的更新。如图 20 所示，算法 5 可以类似地进行更新。

[0212] 为了如上所述构建泛化为处理替代孩子链路的树，可以更新如图 18 所示的算法 3。算法 5 是考虑了替代孩子链路的算法 2 的更新。最初使用 Huffman 算法构建二叉树。之后增加相互依赖性作为替代孩子链路。附着在左和右孩子上的标签 0 和 1 仍然用于发现任何叶的代码。替代孩子链路可以用于填充节点中的度量。替代孩子链路本身可以不附着有标签。

[0213] 用度量填充节点后，可以将验证树发送到网络中的确认实体。可以执行完整性树和参考树的比较。这可以根据图 21 所示的算法 6 执行。

[0214] 如图 21 所示的算法 6 对于由于存在节点间依赖性而没有替代孩子链路的（通常是不平衡的）树是有效的。

[0215] 如果还处理具有替代孩子链路的树，那么也可以使用同样的算法。这种情况下，运行该算法而不对代理孩子链路关系做任何特殊处理。当以这种方式运行时，算法仍然获得失败的组件的同样的修剪树。但是，替代孩子链路提供对失败的额外的可见性，因为当组件被识别为具有完整性检查失败时，根据该算法，替代孩子链路关系深入了解哪个其它节点（可能不是直接位于失败的组件之上）也可能受到识别出的失败的组件的影响。

[0216] 在上述算法的另一个变体中，可以以“交互的”方式执行这些和类似的故障检测算法。在一个实施方式中，返回参考图 14，例如，发送方可以发送针对仅位于直到某内部深度 $d \leqslant D$ 的节点的测量，其中 D 是最长树枝的深度。验证器可以验证那些内部节点。

[0217] 如果检测到了失败，那么验证器可以请求更多的对应于失败的内部节点以下的子树的数据从发送方发送。发送方发送该数据，然后验证器可以检查更深等级的测量的完整性。该方法可以包含更多的这种相互作用，而不是仅两个相互作用。

[0218] 如果由于例如通信带宽或净荷限制的原因很难马上发送树的所有节点的所有树形 SML，那么使用诸如上述的“交互”协议可能是有意义的。完全形成的二叉树包含最多 SML 的线性链结构的数据的两倍数据。这些需要发送的额外数据量依赖于树形成的深度和丰满度，但是在某些情况下，甚至完全形成的二叉树需要发送的额外数据量也可以是适度的，例如，多一些 Kilobyte（千字节）的测量值。然而，如果以更具表现力的语义确认数据，诸如详细证书，RIM 证书或 OMA DM 信息或类似的，来注释树由此在数据规模上很大，则可以更合理地考虑交互协议。

[0219] 对确认系统优化准则的特别、特定选择可以得到二叉树形成的确认数据结构，这些准则是 1）验证对内部 SML 受损的保证手段，以及 2）对受损组件的快速搜索。但是，二叉

树形成的确认数据结构导致成本，诸如 1)在设备和验证器两者处完全构造树所增加的计算成本，2) 由于需要存储内部 SML 而增加的存储成本(高达两倍于线性 SML 的量)。如果在设备的不同组件中存在大量的相互依赖性，那么树可能不是最佳结构。

[0220] 其它类型的系统优化准则可以导致可能不同于树的确认结构。例如，如果兴趣在于绝对最小化内部存储介质而不是失去检测失败的快速精细粒度的能力的成本，那么线性 SML 结构可能是最佳选择。

[0221] 另一个可能的优化准则可以是快速确定一个或少量已知处于危险中的组件附近(诸如小矩形或环形的)的完整性的轻松 / 速度，其中邻近组件已知具有高度未结构化的相互依赖性。在这种情况下，例如，可以出现看起来像互联环或网格的非树形确认数据结构。对于本主题更普遍的处理，可以使用图形 - 理论研究。

[0222] 下面描述具有树形成的受保护的验证数据寄存器的安全操作。从概念上将功能加入到可信平台模块(TPM)以处理平台配置寄存器(PCR)，这些寄存器表示保护树形存储的测量日志(SML)的完整性的哈希树的根。这使得能够进行 SML 的内部节点的验证和更新，甚至以与用于正常 PCR 相同的安全等级对其值进行证明。作为一个重要的应用，示出了 SML 子树的证实如何能够实现平台特性的证明。

[0223] 更多可信功能可以增加到树形验证数据的操作中。示出了根在验证数据寄存器中受保护的树形 SML 的内部节点可以以维持总体安全性等级的受控方式被验证完整性，并用新的值进行更新。引入了用于内部树节点的 TPM_Quote (引用)命令的变体，其可以如 TPM_Quote 对正常的 PCR 值所做的那样对其完整性进行精确证明。对于定义的命令集合，TPM 的完整性测量功能可以由操作树形 PCR 和 SML 的综合的命令集合进行补充。使用它们，树形验证和 / 或确认数据可以使用得远比正常、线性链式 TPM PCR 和 SML 更加灵活和具有表现力。

[0224] 还描述了基本系统模型和注释。还提供了上述的 TPM 命令扩展并进行定义，以及还提供了用于这些操作的某些相关结构扩展和基本用途分类。作为引入命令的重要用例，还提供了用于由可信第三方对树形 SML 中子树的根节点进行证实的协议和对树形平台确认方法的简要评估的展示，以及还描述了对未来工作的展望。

[0225] 描述了后来需要的平台的最小元件和能力。本文使用并描述了 TCG 术语和某些概念，但是本文描述的实施方式不被限制为遵守 TCG 标准的平台和安全硬件元件，诸如 TPM、SML 和例如根据 PC 客户端规范设计的系统。

[0226] 上述定义的扩展操作的树形成变体在 TPM 内操作，采用单个测量值作为输入，否则对于 TPM 外部的系统是无活动的(inert)。对于本文描述的更新功能情况不是这样。后者在 TPM 内部受保护的一定数量 r 的验证数据寄存器 V，$\underline{vdef}\{V_1,\ldots,V_r\}$上操作，也在保存在受较少保护的存储装置中的 TPM 外部存储的哈希树数据上操作。

[0227] 也就是，包含在存储的测量日志(SML)中的哈希树可以由可信软件栈(TSS)进行管理，其中 TSS 被授权对接入更新操作所必需的 TPM 功能。TSS 经由授权的完整性保护的命令接口调用 TPM。虽然 TCG 被用做实际原因的通用说法，但是此处提出的概念可以不限制为 TCG、TPM 和平台。可以使用一组硬件保护的验证数据寄存器和扩展操作。扩展操作可以由普通的 TPM 扩展操作进行定义：$V \leftarrow V \Diamond m \underline{\text{def}} H(V \| m)$(式 4)，其中 V 表示验证数据寄存器，H 是避免冲突哈希函数(TPM 情况下是 SHA-1)，m=H(数据) 是测量值。下文中◇自由

地与作为自变量的任意寄存器 V 一起使用，因此不出现混淆。

[0228] SML 可以包含由二进制单向操作得到的深度为 d 的二叉树，诸如，例如 Merkle 哈希树，由上述的 TPM_Tree_Extend 命令产生。内部节点和叶的自然坐标是长度为 1,.....d 的二进制串，其中串的长度 l 是节点所在的树中的等级。使 n 作为内部节点或叶，并写入 $n^{\sim}\langle n\rangle=(n_1,.....,n_1)\in\{0,1\}^{x1}$ 用做坐标 n 的二进制表示。使 $\langle n\rangle = n_k$, $k = 1,....1$，作为 <n> 的第 k 个数字。因此不会发生混淆，节点可以以其在 SML 中的值（例如 160 比特哈希值）而被识别，同时从其坐标区分。除此之外，n=(n,<n>) 用于一节点的值坐标对。

[0229] n 的轨迹（trace）T 是从 n 到根的路径上所有内部节点的有序列表，包括 n 个，即，$T(n)=(t_1,.....t_1)$，其中 $t_k^{\sim}$ $(n_{1,.....n}k)$。（式 5）

[0230] 节点的自然偏序被写作 $m\leqslant n$，相当于 $n\in T(m)$。当且仅当 $\forall m\in M:\exists n\in N:m\leq n$ 时，通过设置 $M\leqslant N$ 将偏序扩展为节点集合 M，N。

[0231] n 的减小树 R 是其轨迹的所有兄弟的列表。这很容易在自然坐标中表示：$R(n)=(r_1,.....r_1)$；其中 $r_k\sim(n_1,...\ \neg n_k)$，（式 5）。其中 $\neg$ 表示二进制的非。

[0232] 哈希链操作 $x\ \Diamond y\ \underline{\text{def}}\ H(x||y)$ 在使自变量顺序依赖二进制参数的变体中使用固定长度的输入哈希值 x，y。对于 $c\in\{0,1\}$，可以如下设置：

[0233] $\langle x\ //\ y\rangle=\{x\Diamond y$ 对于 c=1；

[0234] $\langle x\ //\ y\rangle=\{y\Diamond x$ 对于 c=1；

[0235] 手性（chiral）Merkle-Damgard 操作是扩展操作的一个版本，其允许区分树中的左和右兄弟并以正确的顺序计算它们的父节点。不考虑实施问题，（扩展的）TPM 能够在内部执行 <.|.|.>=。

[0236] 在某些情况下，存储在 SML 中的哈希树可能是不完整的，即，包含空的叶和内部节点，由 nil 表示。对于 Merkle 哈希树中 nil 节点的一致处理，假设 nil 对于操作◇是双侧单元是有用的，即，x ◇ nil=nil ◇ x=x，以及 nil ◇ nil=nil（式 6）。

[0237] 这是对通常 TPM 扩展操作的重新诠释，也可以用于通过首先复位 V 为 nil，接着对某些值 x 执行 V ◇ x 来建模直接写入 $V\in v$。为实施这种约定，nil 可以表示为验证数据寄存器以及 <.|.|.> 的输入和输出的标记。对于 V，nil 标记可以由特定复位命令来设置。当遇到 nil 作为对 V 的扩展操作的输入时，TSS 或 TPM 修改的逻辑可以防止扩展的执行和直接写入到 PCR。

[0238] 描述了与树形 SML 安全操作的标准 TPM 的操作扩展。保护的目标是对于这种 SML 的内部节点和叶实现与在 PCR 中保护的传统验证数据寄存器值相同的保证等级。首先描述了由新节点值进行的根的更新，接着进一步显示了与树形验证数据一同使用的结构性和命令扩展。

[0239] 用于 SML 树的内部节点或叶的安全更新的策略如下。首先，可以验证那个节点当前值的真实性。这通过使用包含在与该节点相关联的减小的哈希树中的数据，重新计算在寄存器 V（在本节剩余部分保持固定以简化表示）中保护的树根来执行。该验证必须是 TPM 内受保护的操作，称为 TPM_Reduce_Tree_Verify_Load（TPM 减小的树验证加载）。该验证还将经验证的减小的树数据加载到一组验证数据寄存器中以与后续更新操作 TPM_Reduce_Tree_update（TPM 减小树更新）一起使用。该函数使用新的值用于将要更新的节点，并使用减小的树的数据对直到根 V 的父节点进行更新。两个命令可以独立用于各种目的，例如

独立节点完整性验证。为方便起见，它们还可以合并到单个节点和根更新命令中。

[0240] 假设 n 是在等级 1 ≤ d 处具有在验证数据寄存器 V ∈ v 中保护的根、深度为 d 的 SML 树的节点。第一步是以针对 n 用新值更新 V，验证 SML 中的减小的树 R（n）未经篡改。

[0241] 为了维持 V 的安全等级，这种验证也可以由 TPM 保护的操作执行。为此，TSS 以自变量((<n>,n R(n))调用 TPM_Reduced_Tree_Verify Load。从 V 中选择 1+1 个可用的寄存器，称它们为 B_1，.....B_1 和 V^*。如图 22 所示的算法 1 示出了如何验证 SML 节点以及其减小的树如何加载到一组验证数据寄存器中。

[0242] 在该算法中主要使用的手性扩展保证了在轨迹 n 的其父元素的计算中孩子节点的正确顺序。TSS 获得计算出的轨迹 T(n) 和验证状态作为返回值。在算法 1 中，如图 22 所示，存在额外的验证数据寄存器 1+1。

[0243] 图 22 中算法 1 的简单变体可以使用单个验证数据寄存器，通过相继地处理减小的树，而不将减小的树存储在 TPM 内而运行。辅助命令 TPM_Reduced_Tree_Verify 可能对 TSS 或另一方的 SML 的普通验证有用。这在图 23 所示的算法 2 中示出。该变体所需的 R(n) 的序列化可以使用在 TSS 下的软件层（例如，TPM 设备驱动器）中实现的输入缓冲器来实现，或者由相应的 TPM 内部逻辑实现。

[0244] 像描述的（图 4 和 6）原始树形成算法一样，算法 1 和 2（分别在图 22 和 23）使用非标准的操作，尤其是手性扩展。由于手性扩展的输出目的总是验证数据寄存器，操作可以通过以下而被执行：将另一个自变量加载到另一个验证数据寄存器中（如果不是已经存在，如图 22 的算法 1），且在具有寄存器交换的 TPM 内部操作◇之前，依赖于手性扩展的中间自变量。这可以保证所有自变量的相同保护等级。

[0245] 图 22 和 23 各自的算法 1 和 2 执行的验证意义有限，因为它保证了关于输入减小的树的输入节点值的完整性。在 SML 树的完整性被破坏的情况下，期望更详细的信息。至少可以通过经由上述向下树遍历执行确认策略而获得发生完整性破坏的树的等级。

[0246] 算法 3（图 24a）中所示的命令 TPM_Tree_Node_Verify 返回不正确的减小树和 / 或轨迹元素首次打破从根到 n 的完整性链所在的等级。不允许确定哪个兄弟破坏了链。当如上所述参考树可用时，进一步的诊断是可能的。

[0247] 对于可能以新值 n’被更新的节点 n，可以执行 TPM_Reduce_Tree_Verify_Load。命令 TPM_Reduced Tree_Update，如算法 4 所示（图 24b），用自变量 n’来调用，并可以对在确定节点坐标（n）和将要更新的寄存器 V 的 TPM_Reduce_Tree_Verify_Load 之前所确定的结果起作用。为了实现这个命令序列的绑定，可以运用各种方法。

[0248] 首先，TPM 可以存储并管理如下面描述的用于树操作的状态和额外数据。此外，可以例如，通过如 TPM 保护的 OIAP/OSAP 授权命令会话执行的滚动随机数加密地绑定命令 TPM_Reduce_Tree_Verify_Load 和 TPM_Reduced Tree_Update 的序列。最后，两个命令可以加入到单个更新命令 TPM_Tree_Node Vrified_Update 中，具有自变量（<n>,n,n’,R(n)）。节点更新命令返回新轨迹 n”和新值 V，然后 TSS 用它们来更新 SML。

[0249] 利用验证数据寄存器和哈希树的某些节点或根的关联，以及相关联的命令 TPM_Tree Extend（上述定义的）、TPM_Reduced_Tree_Verify_Load、TPM_Reduced_Tree_Update，这些寄存器 V 获得状态性。特别重要的状态可以是：

[0250] 激活根（AR），表示当前在 TPM_Tree_Extend 操作构造下的 SML 树的根。

[0251] 完整根(CR),表示树的根,其是许多组件(即,TPM_Tree_Extend 操作)的测量的完整结果。当树是完整的,即包含 2d 个叶测量时,或者如果在某一阶段期望关闭树,可以由 TSS 触发时,AR 可以转换为 CR。CR 状态中的 V 可以受到保护防止进一步以 TPM_Tree_Extend 更新,但是可以由 TPM_Reduced_Tree_Update 或者甚至正常的 TPM_Extend 操作根据策略和授权进行访问。

[0252] 树建立(TB),表示用于在另一个 AR 中由 TPM Tree_Extend 操作建立激活树的寄存器。

[0253] 减小的树节点(RT),表示 TPM_Reduce_Tree_Verify_Load 的结果,即寄存器 B_k 中的一个。RT V 可以一直受到保护,直到相应的 TPM_Reduced_Tree_Update 或另一个授权的命令出现。

[0254] 当管理超过一棵树时,V' 状态需要例如,使用唯一标识(UID)与其各自的树相关联。此外,可能需要为每个或某些寄存器存储节点坐标。这些数据可以在 TPM 内部的验证数据分配表(VDAT)中保留,并由树数据管理单元(TDMU)管理。

[0255] 节点值的 TPM 保护验证可以实现针对通过证明的平台确认的新的核心语义。特别地,可以定义证明某一节点值的 TPM_Quote (引用)的变体。可以用与 TPM_Quote 相同的自变量加上 TPM_Reduced_Tree_Verify 的自变量调用这个命令 TPM_Node_Quote。接着执行图 23 所示的算法 2,但是此外在另一个 PCR V' 中保存 n 的拷贝。一旦成功就在 V' 上执行 TPM_Quote。这种引用的接收方应当清楚获得的签名在 SML 树的内部节点之上。一个可能性将是改变在签名的一 TPM_Quote 命令中包含的固定串,通常是“QUOT”,也就是说“TREEQUOT”。

[0256] 用这个命令证明节点值向节点提供了与使用 TPM_Quote 引用验证数据寄存器(PCR)值相同的安全性断言。然而,其可能会负担该值对应于 SML 树的某些内部节点的额外语义,即,它有效地证明 n 是根的某子树的状态。为了明确地向确认器传达该语义,可以在 AIK (证明身份密钥)签名的证明包,例如“树节点”串中包含额外的数据。如果根寄存器是由于 TPM_Tree_Extend 命令导致的受控 SML 根寄存器,即,如上述讨论的其处于 CR 状态,如果这种属性由 TPM_Tree_Node_Quote 指派,则这种属性的意思可以被明显加强。这种控制可以是引用生成的一部分。

[0257] 对于证明消息的确认,确认器可能需要引用的节点 n= (n,<n>) 的值 n。可能不需要发送到确认器的更多的信息,因此 TPM_Tree_Node_Quote 的上述描述遵循最小启示的原则。如果节点在 SML 树中的位置对确认很重要,那么命令的变体也可以签名节点坐标(n)。发送到确认器的扩展的确认数据可以包括 n 的减小的树和根验证数据寄存器。

[0258] 作为可替换实施方式,给确认器分配减小的树的验证的任务是可能的。这个方法在别处用于证明由网页服务器传递的网页的完整性,并使用普通的 TPM_Quote 命令将该证明绑定到服务器状态的证明。这带来如下的 TPM_Tree_Node_Quote 的变体实现。

[0259] 命令接收节点值 n、节点值 R(n) 和用于根 V 的选择器作为自变量。TPM 在控制 V 的 CR 状态之后签名这个(级联)数据,且使用“REDTREEQUOT”固定串属性。值得注意的是对于用于证明网页完整性的方法,变通方法用于通过将这些额外数据的哈希插入到 TPM_Quote 命令的随机数 (nonce) 输入中(其通常用于保证引用的新鲜度(freshness)),将减小的树和来自 TPM 的引用进行绑定。

[0260] 用于引用内部节点的第一和第二实现变体代表相反的可能性,在某种意义上第一

实现变体将验证加载与平台放在一起，而第二实现变体将其与确认器放在一起。因此，两者具有不同的实际效率范围。例如，许多如在用于第一实现变体的 M2M 通信中的分布式确认平台，和许多用于第二实现变体的分布式确认器(诸如上述描述的 Web 客户端)。但是由于向确认器示出了由 V 表示的完整状态，第二变体对于平台的信息披露具有原则性的缺陷。这可能有损于隐私。

[0261] 此处介绍的 TPM 完整性测量功能的各种扩展可以被分组成下述类别。TPM_Tree_Extend 在建立具有 PCR 保护的根的特定 SML 树的连续测量过程中使用。TPM_Reduced_Tree_Verify_Load、TPM_Reduced_Tree_Verify 和最后的 TPM_Tree Node_Verify 是用于平台内部诊断的命令。除了使用 TPM_Reduced_Tree_Verify_Load 作为 TPM_Reduced_Tree_Update 的准备步骤外，它们可以在其它事件发生之前用于验证由 SML 子树表示的平台的某些特性。

[0262] TPM_Reduced_Tree Update 和 TPM_Tree_Node_Verified_Update 可以用于子树的受控更新。内部节点更新操作的特定用途描述如下：

[0263] 单个系统组件的更新。在这种情况下新值更新叶。

[0264] 由子树表示的系统模块的更新。这种情况下新子树的根更新原始树的内部节点。

[0265] 最后，TPM_Tree Node_Quote 是使树形 SML 可用于由远端方进行的平台确认的命令。它展示了使用树形数据的确认的主要新元素，亦即对仅表示系统状态的定义部分的子树进行证明的可能性。下面描述了特定的用例。

[0266] 可以在树形确认中分开使用命令扩展。下面描述了一种方法，其以模块化的方式合并命令扩展以确认子树。

[0267] 使用树形 SML 和验证数据寄存器的确认向远程证明增加语义。对 SML 的子树进行证明的可能性使得表现力远远超出传统的远程证明。树形验证数据是具体化其它提案以向平台证明增加语义，例如与确认数据的关联和确认数据的特性的一种方法，如在 PBA 中那样。可信计算的一个问题是语义到平台证明的关联。TCG 规范定义了双边远程证明，其中在执行的代码被加载时对其进行测量。测量结果被存储在 PCR 中作为验证数据，且 TPM 通过用 TPM 保护的证明身份密钥(AIK)对这些数据进行签名来对这些数据进行证明。由于传送了完整配置的摘要，验证器可以知道机器的配置(如果考虑系统动态，则总是知道)。因此所传送的用于确认的数据缺乏表现力以实现多方面且有效率的远程平台确认。早就意识到了需要语义证明，其中提出将单个证明的范围限制在具有有限复杂性的虚拟化子系统中，允许复杂、动态和高等级程序特性的证明。

[0268] 本文描述了特性和基于特性的证明(PBA)。PBA 允许经由可信第三方(TTP)，称为子树证书授权中心(SCA)向验证器保证经验证的平台的安全性特性。SCA 发布证书，其将平台配置映射到可以在该配置中实现的特性(尤其是期望 / 不期望的功能)。PBA 将平台确认的基础设施问题转移到 SCA，这与该 TCG 隐私 CA 的情况类似，扩展 TCG 隐私 CA 的作用除外。子树的证实是克服所描述问题的一种方法。一个相关的实施方式是由 TPM 命令执行的硬件支持的更新，该 TPM 命令基于更新证书为另一个平台配置重新密封(re-seal)数据。

[0269] 与验证数据不同，确认数据是提交给另一方，确认器的全部数据，并用于对平台状态的可信度进行评估。向确认器提交确认数据的过程，实现为根据例如 TCG 的远程证明，且由确认器进行其评估，被恰当地称为确认。确认数据可以包含验证数据，诸如引用的验证数

据寄存器(例如 PCR)值。确认可以,除验证数据的加密验证外,包括策略评估和由确认器进行的动作触发。

[0270] 与 SML 树相关联的树形验证数据和确认数据提供结构化数据,其除了用于如上所述方法外还用于增强平台证明的语义。提供了另外一种使用树形验证数据的方法来实现与 PBA 相关的概念。

[0271] 示出了 SCA 如何用关于被测组件的子树的有意义的可信声明——子树证书来替换 SML 树中的节点,其中后一个节点是该子树的根。这个过程由 SCA 实现平台的部分确认,并得到被摄入(ingest)到平台的可用确认数据的可信断言。后面这可以用于另一个确认器以更全面地确认平台。接下来,描述了可以实施以用于变体实现和用例的子树证实的通用基础协议。还描述了多种变体。

[0272] 子树证实是一个过程,通过该过程,可信平台(其可以是,但不限于以简化系统模型的 TPM 和 TSS 的组合)从 SCA 获得对于树形 SML 的内部节点值的证书。为此,平台向 SCA 提交签名的声明,表明节点值包含在具有在验证数据寄存器中保护的根值的 SML 树中。TPM 是一种可能的实施,且可以使用任何验证数据寄存器。基于这一现象,SCA 可以向平台发布一个具有关于这个节点值的附加属性的证书,然后该证书可以被摄入到 SML 树中。该过程使用受保护的树操作。

[0273] 平台可以拥有一个激活 AIK a,具有由可信隐私 CA (PCA)发布的证书 C_a。平台和 SCA 之间的通信是加密的,完整性受到保护以减轻人为中间干预的攻击。选择一个内部节点用于证实。在该协议中,没有提到失败情况和否定响应。该子树证实协议以至少 5 个阶段进行,如图 26 所示。

[0274] 阶段 1 生成 s 上的引用。为此,TSS 调用 TPM_Tree_Node_Quote,使用自变量(<s>,s,R(s),a)(为简便起见,图 26 没有示出可以被调用的所有自变量)并接收返回的 $P=Sig_a(s)$。

[0275] 如果树根,即寄存器 V 被选择用于证实,那么反而在 V 上使用 TPM_Quote。在阶段 2,TSS 生成证明包 Q。它包含用于验证 SCA 的信息,至少为 :$Q \subseteq \{P, s, C_a, a_{pub}\}$。当不是从 C_a 中明显看出时,a 的公共部分 a_{pub} 可以包含在 Q 中。而且,s 的值可以为 SCA 可知,接着从 Q 中删掉。

[0276] 可以包括更多信息,例如当作为引用的一部分时的节点坐标<s>。Q 被发送(以 SCA 的公共加密密钥进行加密并且进行完整性保护)到 SCA。这个阶段与在 TCG 规定的远程证明中的类似。

[0277] 阶段 3 包含 SCA 的活动。首先,SCA 可以通过验证 P 的签名和 / 或追踪证书链直到 PCA 的根证书来验证 Q。如果 SCA 认识到 s 是它可以证实的节点值,它就为其生成清单 M_s。这个清单可以包含关于与 SML 中具有根 s 的子树的存在相关联的平台状态的额外信息,诸如时间戳、平台功能、从子树的叶测量表示的加载的组件合并的模块识别和 / 或另一个平台特性。清单是由向确认器提供节点值 s 的语义含义的子树证实增加的确认数据。SCA 可以为 s 生成一个证书。这个证书 C_s 通过将其绑定到 AIK a,将由 M 表示的特性绑定到平台。这可以以两种方式实现,亦即 :

[0278]

$$C_s = \begin{cases} Sig_{SCA}(M_s \| P) & \text{如果 s是显式的;} \\ Sig_{SCA}(M_s \| bind(a)) & \text{如果s是隐藏的。} \end{cases}$$

[0279] 在第一种情况下，SCA 签名清单和 AIK 签名的节点值，从而建立到 a 的间接绑定。如果平台显示了节点值 s，那么可以验证 C_s 到 a 的绑定。在第二种选项中，绑定通过以下动作直接完成：使 SCA 签名唯一识别 a 的某些数据 bind（a），诸如 a 的公用部分，序列号 C_a 或者 C_a 的指纹。在公钥基础设施的语义中，通过绑定 C_s 是与 C_a 相关联的属性证书。最后，SCA 生成包 R，至少包含 M_s 和 C_s，以及在第二种情况下的 bind（a），并将 R 返回到平台。

[0280] 阶段 4 为用从 R 得到的某些数据更新 SML 做准备。SML 更新可以产生子树证书和树中经证实的节点位置 <s> 之间的绑定关联。

[0281] 这可以使得平台向确认器声称由 C_s 和 M_s 证明的特性已经在平台配置中存在。可以设想各种 SML 更新将 C_s 绑定到表示的子树的方式，每一种适用于不同的特定用例。

[0282] 生成具有下列特性的新节点的集合 $U = \{u_1, \ldots, u_k\}$（例如，在 SML 树中的值和位置）。首先，该集合可以保持 $U \leqslant s$，使得 s 下的子树可以被更新触及。这可以执行，因为旧的 SML 树节点 $n \leqslant U$ 严格在 U 之下，即，$n/ \in U$ 被更新无效，而且不再关于更新的根验证数据寄存器而被验证。其次，U 是没有依赖性的，即，$u, u' \in U: u \leqslant u'$。

[0283] 没有依赖性是确保由 U 以在 Merkle 哈希树中体现的单向（上行）信息流执行的树更新的一致性的特性。尤其是，它使得更新结果不依赖于 U 的元素进行处理的顺序。

[0284] 阶段 5 正是 SML 树更新。重复 $u \in U$，以自变量（u,n,u,R(n)）调用 TPM_Tree_Node Verified_Update，其中 n 是位置 <u> 处旧的 SML 节点值。这返回新的轨迹 T(u)，TSS 以其更新 SML。以上述描述的方式执行树的更新保持了对于 SML 和根验证数据寄存器一致的安全等级。亦即，操作◇在 TPM 内被执行。当 U 包含许多元素时，以针对阶段 5 描述的方式执行更新可能是没有效率的，这是因为 TPM_Tree_Node_Verified_Update 在这种情况下将验证多个重叠的减小的树并因此导致（复杂）哈希计算中的冗余。下面描述了更有效的更新算法，称为“有效安全节点集更新”。

[0285] 如上所述，使用 TPM_Tree_Node_Verified_Update 更新一大组 U 的内部节点可能是徒劳的，因为根据 U 元素的减小的树的重叠，可能出现用于验证的许多冗余哈希操作。可以使用上述 TPM_Tree_Extend 命令，应用批量更新策略以改善子树证实协议的阶段 5 的朴素（naive）算法。它依靠这样的观察，其中更新集 U 的子集跨越不依赖于旧 SML 值的子树，即，它们的根依赖 U 中的节点。从而被这些集合跨越的树根可以无需昂贵的验证而被预先计算。

[0286] 提供了一些定义。假设 $U = \{u_1, \ldots, u_k\}$ 是没有依赖性的更新集。如果 a）SML 树中的一个节点是 U 的元素，b）SML 树中的一个节点的兄弟节点在 U 中，或者 c）SML 树中的一个节点的兄弟节点是 U- 内在的（U-intrinsic），则称这个 SML 树中的节点为 U- 内在的。这种递归定义捕获了其更新后的值依赖于 U 而不依赖于 U 的补集中的 SML 节点的节点。子集 $V \subseteq U$ 的跨度根是 V 的所有元素的轨迹的唯一交叉点。子集 $V \subseteq U$ 跨越的子树是 V 的元素轨迹的并集，忽略了严格在跨度根之上的节点。如果子集 V 的跨越的子树的所有元素都是 U- 内在的，那么子集 V 称为 U- 内在的。

[0287] 根据这些设置，用 U 对 SML 的更有效更新可以执行如下：

[0288] 1）识别（互不相交的）U- 内在子集 $\mathbf{V}_1, \ldots, \mathbf{V}_k \subseteq U$。

[0289] 2）重复 V_i，i=1，…，k。

[0290] a）标准化 V_i 的元素的坐标，通过：

[0291] i）截短 V_i 跨度根的坐标给出的前缀，以及

[0292] ii）对由 V_i 跨越的子树的深度进行后修正加零，直到所有坐标具有相同的长度。

[0293] b）根据其标准化的坐标对 V_i 的元素按字母表排序，产生有序的列表$\widetilde{V}_i$。

[0294] c）用 nil 值填满的$\widetilde{V}_i$中的所有间隙（在标准化的坐标中）。

[0295] d）选择空闲的验证数据寄存器 V'。

[0296] e）按顺序在目标为 V' 的$\widetilde{V}_i$的元素上使用 TPM_Tree_Extend。

[0297] f）从 U 中移走 V_i。

[0298] g）插入 (V',<v_i>) 到 U，其中 v_i 是 V_i' 的跨度根。

[0299] 3）对于 U 的剩余元素，应用使用 TPM Tree_Node_Verified_Update 的如上所述的正常更新进程。

[0300] 子树证实协议的变体可以结合分别用于 AIK 和在单个协议运行中的子树证实的 PCA 和 SCA 的作用。优点在于可能无需进行明确的 AIK 认证 C_a 的生成和验证，因为 AIK 的生成、激活和使用绑定到一个会话中。这种协议的结合是直接的。

[0301] 将接收的子树证书绑定到平台状态意味着将它绑定到正确配置中的树形 SML，即经证实的子树根的位置。如上所述，这对于在整体平台配置的上下文中的有意义的基于子树证书的确认是必要的。将 C_s 绑定到 SML 树的一个特定目标是完整性保护，这是因为例如，避免了后面的用不同的证书进行替换。绑定可以通过用唯一可验证地识别子树证书的数据更新树的部分来实现。很宽范围的数据项可以从子树证书产生并输入到 SML 树的各种位置。

[0302] 在一个示例中，SML 更新可能是微不足道的，U 可能是空的。如果 a 由上述描述的第一选项组成，显示 s，则这是可能的。那么 s 可以保留在 SML 树中，其下的子树是否也保留依赖于用例。绑定关联经由 C 签名的实际节点值 s。

[0303] 在另一个示例中，考虑这样的情况，关于由证书证明的平台特性的数据可以由更新后的树进行保护，例如，用于取证用途。也就是说，三个数据条目 s，M_s 和 C_s 可以输入更新集。当节点值 s 已经是正确的数据格式时，另外两个首先被处理成 m(M_s) 和 m(C(s))。操作 m 可以是由平台的用于测量的信任根（RTM）生成哈希值，或者另一种适当的单向操作。如果某些数据条目已经包含了适当节点值格式的合适唯一识别数据，那么它可以直接被提取并用作节点更新值。一个特定的示例可以是包含在 C(s) 内的证书指纹。接着三个更新节点可以在更新集中配置为生成，例如，如图 27 所示的 SML 树中的更新后的节点的配置。

[0304] 更新后的子树的根被插入到 s 的旧位置并具有值 k=(m(C_s) ◇ m(M_s)) ◇ s。这个配置向子树证书和清单提供独立的完整性保护，并独立地保留旧的节点值。在该配置中，对由 C_s 表示的平台特性的证明仍然可以不透露关于 s 的信息，通过仅引用子树左侧内部节点 * 来完成。

[0305] 针对子树绑定的证书的变体大量存在。平台还可能想要包括在其中自身生成的数据（的完整性保护值），例如来自安全时钟的内部时间戳。此处的实施可能是用例专用。

[0306] 对于由针对确认器的子树证书表示的特性的证明，平台可以使用 TPM_Tree_Node_

Quote 引用保护期望确认数据的更新后的子树中或其上的任意节点，所述期望验证数据适当地至少包含清单和证书。平台然后视需要向确认器提交确认数据，至少是声称的特性的验证所需的所有数据，再一次包含至少 M_s 和 C_s。已经由提交的引用保护的确认数据原则上不需要额外的完整性保护。

[0307] 确认器可以验证确认数据的平台绑定。证实这种特定，即，确认平台与对 SCA 执行的子树证实是一样的，可能是重要的。实现它的一种方式是在子树确认中使用与在证实中使用的相同的 AIK，a。平台接着也可以提交 a_{pub}，且如果必要，C_a 也可以作为确认数据的一部分。是否需要 C_a 可依赖于子树证书的语义，即，SCA 可能已经检查了 AIK 证书，且 C_s 可以声明其真实性。相应的信息可以放置在清单中。重新使用相同的 AIK 部分地危害隐私，且也可以考虑解决问题的其它方法。

[0308] 使用子树证实的一个步骤是发现平台可以从特定 SCA 获得证书所针对的子树。没有深入讨论平台、SCA 和确认器两两之间交互的细节，下文描述了两种子树发现方法。第一种将子树发现的工作量放在平台上，而第二种假设了一个“哑的（dumb）”平台，将更多的负荷放在 SCA 上。

[0309] 在一个示例中，SCA 向平台发送一些子树发现数据，最简单的情况下是准备进行证实的节点值的列表。平台可以在其 SML 树中搜索这些值并对每个识别的节点执行子树证实。这个基线过程提出了各种精细化，尤其是丰富发现数据和将发现过程扩展到平台与 SCA 之间的协商协议。例如，发现数据在可证实的根的情况下可以包含根节点位置，在经认证的启动过程中产生 SML 的情况下其对应于这样一种事实：在后一种子树的建立过程中加载的组件在平台启动的定义阶段被加载。

[0310] 对于配置可能动态改变的复杂平台，这种对节点进行绝对定位的情况很难实践。因此更精细的情况也可以表达某些例如可证实的根的对的相对位置。SC 可以声明表达式，诸如，如果例如“r”先于 s（即，r 可以依赖到有序 SML 树中 s 的左侧），表达式宣称 s 是可证实的。这最后可以被解释为某功能在平台上操作，如果使另一个功能在它之前可操作的话。

[0311] 模型的一个更根本不同的变体是发现数据不包含子树根，即内部节点，而是包含叶，即测量值。“自下而上”发现过程可能需要平台根据接收到的 SCA 声称知道是可信值的叶测量值进行关于哪些节点可证实的“经验猜测”。一个方法是找到其叶全部在发现数据中的子树的跨度根的集合。接着平台可以引用子树根并将其与其 SML 子树一起发送到 SCA。一般，SCA 可能必须对 SML 子树进行验证，并决定是否准备好证实那个根，因为这可能仍然依赖于叶的排序。在某些情况下，平台可能想要获得子树的证书，为此 SCA 知道某些叶值，即，相应子树的叶集合可能关于发现数据具有间隙。如果平台有其它可信数据，例如从 SCA 信任的一方获得的 RIM 证书，平台可以向辅助 SCA 提交子树证实的阶段 2 中的这些数据，连同其证实子树根的决定。

[0312] 在设备不能执行子树的本地发现的情况下，可以使用将计算移到 SCA 的实施。平台选择一个内部节点 n，目的是从 SCA 获取证书用于 n 以下任意合适的子树。在平台想要使所有可证实的节点被证实的情况下，节点 n 可以等于完整树（V）的根。接下来的两个步骤可以与在上述阶段 1 和 2 中描述的相同，即，如果选择了 V，平台分别执行 TPM_Tree_Node_Quote 或 TPM_Quote。证明包与引用的根之下的 SML 子树打包在一起。SCA 接收这个信息，

接着，使用树遍历技术，验证树的完整性并同时找到一个或多个（不相交的）具有可证实的根的集合 S 的子树 S_i。

[0313] SCA 接着重复如上所述协议的阶段 3，对所有 $s_i \in S$ 生成证书。由于协议允许多个节点更新，该多个节点合并到更新节点列表 U，所有发现的子树的更新都可以在单个的协议运行中执行。一个变体可以是对于平台不在第一步发送 TPM_Tree_Node_Quote 或者 TPM_Quote，而是向 SCA 提供从选中的节点 n 开始的 SML。然后 SCA 搜索将要证实的潜在候选子树，接着请求平台向识别的子树的根节点提供 TPM_Tree_Node Quote。在某种意义上这是一种权衡，允许平台发送 SML 而不必提前执行加密操作。尽管如此，平台可以在 SCA 的证实之前提供合适的引用以提供对于发送的 SML 的完整性保护。

[0314] 上述描述的是用于实施 TFV 的系统、方法和装置。下文描述的是 TFV 的变体和 / 或扩展。

[0315] 确认器对平台提交的树形 SML 进行遍历的方法可以如上所述是逐级下降的。下文描述的是变体过程，其中在遍历 SML 树寻找失败的叶组件之前，确认器首先在 SML 树中寻找子树根的已知好值，来识别经确认的平台的已知好的部分。

[0316] 保持整个平台的 SML 树的参考树可能是困难的。这样的树可能很大，并敏感地依赖于平台配置，诸如，例如组件的加载顺序。树遍历确认假设一种静态的测量顺序来确定其中描述的诊断确认识别不符合经确认的平台的期望的参考配置的组件。事实上，由于 Merkle-Damgard 变换不是可交换的，也不是关联的（它不是多集哈希函数），所以树根值依赖于精确的测量顺序。既然线性生成的 PCR 值保持了相同，那么上述的性能比较仍然是合理的。下面进一步处理树根顺序敏感度的问题。那里描述的方法可以与此处描述的相结合。

[0317] 在朴素解法（naive solution）中，事件的状态可能有必要针对平台的有效（功能上以及关于安全性）等效配置维护参考树。因此可能期望比完整参考树更细粒度的数据集以用于确认。

[0318] 代替保持 SML 树的根的已知好值和对应参考树的数据库，确认器保持不同深度的子树的已知好的子树根值的扩展数据库。这些值存储为对 (r,d)，其中 r 是子树根的节点值，h 是子树的深度。深度 d 的范围从完全 SML 树的深度 D 到 0，其中 d=0 意味着对应的已知好的参考值实际上是叶测量。当在诊断树遍历中达到一定等级 L 时，且在加密验证之后，确认器通过广度优先搜索，比较在该等级的接收到的 SML 树的节点值和已知的好值 $\{(r,d) | d=D-L\}$。具有匹配根的子树被标记为已知且从进一步的诊断遍历中排除，然后该遍历继续进行等级 L+1。

[0319] 对于确认器，与树从根向下相反，保持已知子树根的数据库还允许不同的确认策略。确认器可以通过任何它认为有效的搜索过程在 SML 树中寻找已知好子树根值（的子集）。然后确认器可以验证这些子树根的完整性。为此，确认器可以使用子树根节点的减小的树和别处描述的该值的验证过程。如果该值进行了完整性验证，那么对应的子树可以被标记为 OK，并从随后的诊断确认中删除。

[0320] 像这样的已知子树的识别，给平台的不同配置流出了空间，尤其是已知组件和失败组件的不同负荷序列。例如可能存在每一个具有 8 个测量叶的两个子树 A 和 B，其在它们之间具有另一个包含失败组件的子树。如果确认器识别 A 和 B 为已知，但是通常将以顺序 B,A 接受它们，且在即时顺序中，或者在中间具有另一个已知子树 C 的顺序 A,C,B 接受，

那么确认器可能必须根据策略决定是否接受平台配置为有效。

[0321] 作为描述的变体的扩展，确认器可以动态地根据以前的平台确认结果通过交互学习提供参考树数据库。一个可能性是执行平台第一次确认的诊断确认，并根据如本文所述的确认器组件(叶)可信参考值和策略，从叶测量的子顺序生成已知的好子树，其对应于好的平台配置。当越来越多的平台进行确认时，确认器可以根据标准，诸如通用性和风险(与配置相关联)，建立已知好子树和平台配置的统计加权数据库。例如，可以首先搜索普遍出现的树。

[0322] 所描述的方法类似于为子树证实方法描述的子树发现方法。它是可以由 SCA 实体与学习策略一起应用或者分开应用的一种方法。

[0323] 确认器遍历由平台提交的树形 SML 的方法是逐级向下的。为此，整个树在一个通信步骤中提交给确认器，并针对参考树进行评估。此处描述了一种可替换实施方式，其逐个地顺序传送在不同等级的树节点。

[0324] 整个 SML 树可以提交给确认器。这通常不是最优的，尤其当失败的叶测量的比例很小时。那么，许多子树将作为整个 SML 树的一部分被传送，但是它们包含的节点数据可能不用于确认，因为它们的子树根相对于参考树中的对应节点是正确的。

[0325] 最小化 TFV 中传送的数据的基本方法是仅传送这些数据到确认器，如在别处描述的那样，这对于树遍历确认中的当前步骤是实际需要的。确认器首先接收 SML 树的根验证数据寄存器，将其评估为等级 L=0 的交互树遍历确认。如果其值与参考树的根一致，则确认结束。否则，确认器从平台请求根的两个孩子节点。接收了它们之后，确认器检测本文描述的图 9 中的完整性失败情况 c) 并停止，或者根据孩子重新计算根。包含孩子节点的消息，原则上不需要对完整性和新鲜度进行加密保护，因为其内容针对父节点通过扩展操作的重新计算而被验证。根，反过来，被假设由 TPM_Quote 或类似的操作，例如数字签名，进行保护而被传送。

[0326] 比较 L=1 处的两个孩子，确认器可以发现两个孩子中的哪个相对于参考树是正确的。它们可以从进一步的搜索中丢弃。接着确认器请求在树深度 1 处的所有“坏”节点的孩子(也就是，在树深为 2 处的 2 或 4 个孩子节点)。对于它们中的每一个，如前进行验证和参考树的比较。通过迭代继续该过程。这样，需要发现失败的叶测量的坏的内部节点实际上被传送给确认器。如前所述，这可以非常有效地完成，消息尺寸短且没有加密开销。相对于树形 SML 的大小的数据量依赖于失败的叶的比例。其显示在本文描述的图 10 中。

[0327] 对于 TFV 适应性存在树形验证数据和其它方法的各种组合：首先，提供确认数据用于频繁改变、动态的平台(部分的)状态，其次，为验证数据提供可扩展空间。

[0328] PCR 或者验证数据寄存器，只是一个(硬件的)保护的存储空间，众所周知平台缺少那种存储空间。如本文描述的，哈希树原则上取消了这个限制。但是具有根保护的验证数据寄存器，例如 PCR 的树形 SML 的安全生成过程可以使得树具有固定深度并因此具有用于叶测量的有限容量。当平台具有的要测量的组件比适合树形 SML 的平台更多时，问题就出现了。

[0329] 如果验证数据保护频繁改变的部分平台状态诸如存储区域的记录，可能出现另一个问题。更新 PCR 值的硬件保护的操作，TPM_Extend 操作是缓慢的。因此 PCR 不适用于频繁改变数据的保护。从而 PCR 可能不适用于直接运行时间存储保护。为此各种系统应用

Merkle 哈希树，但是它们的根无法正常地存储在硬件保护的寄存器中，因为它们的更新将会太慢以至于无法跟上平台的动态。

[0330] 用于保护由依赖于(离散)时间 t 的某些数据 W(t) 表示的动态改变平台状态的基本想法，是对于相关联的验证数据寄存器 R 的缓存再更新策略。某些可信软件 APP_R 保护 R 中的这个状态序列。APP_R 收集长度为 L 的状态 W(kL+1),...,W((k+1)L) 的间隔，在使用摘要函数测量它们之前对其进行缓存，以及例如，基于 Merkle-Damgard 变换，使用 PCR 扩展操作或类似的链接操作将它们扩展到 R。

[0331] 各种提高的安全性要求适用于 APP_R。特别地，对 APP_R 应用足够的保护以防止修改，且必须应用篡改(方法是已知的，例如，虚拟化或者整合到安全内核)。此外，APP_R 的操作可以对外部确认器的评估开放。为此的最小需求可以是对 APP_R 的可信度，例如，在安全启动期间进行验证，通过在加载前对 APP_R 进行测量而注册，并以确认器可证实的方式将其与 R 中保护的实际序列 W 进行绑定。

[0332] 提供期望特性的方法如下。序列 W(t) 可以被分割成长度为 L 的间隔：

[0333]

$$\underbrace{W\left(L\left(k-1\right)+1\right),\ldots,W\left(Lk\right)}_{\text{长度}L},\underbrace{W\left(Lk+1\right),\ldots,W\left(L\left(k+1\right)\right)}_{\text{长度}L},\ldots$$

[0334] 摘要函数应用到部分序列以产生间隔测量值：M_k = m(W(L(k-1)+1)||...||W(Lk))，k>0。作为一种变体，它对于合并间隔测量中来自安全源的时间戳是有用的，例如：

[0335] M_k = m(W(L(k-1)+1)||...||W(Lk)||TPM_Tick(now))

[0336] 将时间戳加入到测量间隔值使得对(子)结构重新排序，而不会丢失关于组件加载顺序的信息。此外，它允许测量的“老化”，其中确认器能够在给定时间段后请求测量值的再生(renewal)。间隔测量值通过将它们扩展到验证数据寄存器来进行保护，例如通过扩展操作：R(k+1)=R(k) ◇ Mk+1，得到类似于正常线性成链 SML 的序列

[0337]

$$\begin{array}{ccc} \rightarrow R\left(k\right) & \rightarrow & R\left(k+1\right) \\ \uparrow & & \uparrow \\ M_k & & M_{k+1} \end{array}$$

[0338] 其中即时状态也可以在 SML 中保护，即，保护 W 的 SML 看起来像：

[0339] $SML_W \subseteq \left[R_{init},R\left(0\right),M_1,W\left(1\right),\ldots,W\left(L\right),R\left(1\right),\ldots\right]$

[0340] 至管理应用 APP_R 的可信度状态的期望绑定可以实现如下。平台的安全或经认证的启动设备在加载前测量 APP_R，并将测量值 m (APP_R)扩展到验证数据寄存器 Q (其可以用在可传递的信任链中之前的 PCR P 的值进行初始化)中。接着，实际记录寄存器 R 的序列用 P 和初始化值 R_init，例如会话随机数或时间戳或两者，进行初始化。这得到这样一个结构

[0341]

$$P \rightarrow Q \rightarrow R = R(0)$$
$$\uparrow \quad \uparrow$$
$$m(APP_R) \quad R_{init}$$

[0342] 如果缺少验证数据寄存器，P、Q 和 R 可以是一个。对于 W 的验证可以在扩展的 SML 上进行，这可以包含：[(W(Lk+1),...,W(L(k+1)))，M_{k+1},R(k+1),R(k),...,/R(0),R_{init},Q,m(APP_R),P] 和 P、Q、和 R 的引用。如果存在，SML 还可以包含时间戳的值。

[0343] 实际硬件保护的验证数据寄存器可以从树形 SML 的根解耦，接着可以存储在保护较少的存储空间，但是由前者进行保护。为此，特定软件 TFV_APP 管理由根 W 保护的哈希树，根 W 可能仍然是普通的存储空间，在深度为 d 的内部树中，具有叶 V_1,…,V_（2^d-1）。W(t) 的状态序列反过来由硬件保护的实际验证数据寄存器 R 进行保护，例如通过使用最后子部分的间隔法。那么虚拟寄存器 V 可以用作 SML 树的根并由 TFV_APP 进行管理。对于它们，TFV_APP 可以将别处描述的命令透露给外部应用，如同它们是根验证数据寄存器那样。R 的结构是：

[0344]

$$\rightarrow R_{N+1}(t-1) \rightarrow R_{N+1}(t)$$
$$\uparrow \quad \uparrow$$
$$W(t-1) \quad W(t)$$

[0345] 而耦合的 W 表示树根，如图 29 所示。

[0346] V_i(t) 的验证可以在下述验证数据上扩展：V_i(t),R(V_i(t)),W(t),R_{N+1}(t),R_{N+1}(t-1),R_N,R_{N-1},m(TFV_APP),Q({R_{N-1},R_N,R_{N+1}})，其中 Q 是 PCR R_{N-1},R_N,R_{N+1} 的引用，并根据下述进行：

[0347] i. 关于其减小的树 R(V_i(t)) 和根 W(t) 对 V_i(t) 进行检查

[0348] ii. 根据（2）检查 R_{N+1}(t)=R_{N+1}(t-1) ◇ W(t)

[0349] iii. 如果另外传送了 R_{N+1}(0)，W(0)，检查 R_{N+1}(0)=R_N ◇ W(0)

[0350] iv. 检查 R_N=R_{N-1} ◇ m(TFV_APP)

[0351] v. 针对参考值检查 m(TFV_APP)

[0352] vi. 检查引用 Q。

[0353] 另外，可以如上所述应用时间戳。假设发生 TFV_APP 到根 R_{N+1} 的绑定，类似于：

[0354]

$$R_{N-1} \rightarrow R_N \rightarrow R_{N+1}(t=0)$$
$$\uparrow \quad \uparrow$$
$$m(TFV\_APP) \quad W(0)$$

[0355] 其中 W(0) 是某个初始化值。

[0356] SML 树的安全形成具有天然的容量限制。解除该限制的一种方法是上述子树证实方法的简化配置文件（profile）应用。某一固定深度的 SML 树被完整填充，平台对树进行（自）证实，并将获得的证书作为第一测量插入到下一个空的 SML 树中。这得到一个左边不平衡的多树结构，如图 28 中所示。

[0357] 寄存器 V 可以保护树形 SML 的根，SML_1。安全地继续下一棵树的独立方式如下。

平台使用 AIK a_1 调用 TPM_Quote(V,a_1) 以获得签名的声明 Q_1。TPM_Quote 表示一个签名命令，其可以在指定验证数据寄存器 V 上的 TP 的 TrE 内(例如，由 TPM)执行。从而，签名和 a1 的 PCA 证书一经验证，Q_1 向任意验证器断言值 V 曾经包含在验证数据寄存器 V 中。平台保存增大的 SML_1 :SML_1*:=SML_1||V||Q_1||Cert_PCA(a_1)，即，原始 SML_1、其根的值、引用和相应的 AIK 证书用于后面的验证。平台复位 V。通过上面提到的特性，SML1* 语义上等同于 SML1 和包含其根的寄存器 V。对于下一个树的建立寄存器 V 现在是空闲(free)的。

[0358] 后面的树可以绑定到前面的树以确保由可信根负责的持续内部验证，即，对于包含的叶测量持续的完整性保证。用于此的一个示意性方法是以绑定到最后的树的第一测量值开始(例如通过修改平台 RTM 的方式)下一棵树，SML_2。下面将描述完成这些的各种示例。

[0359] 例如，最后的根值 V 可以直接用作第一测量。这可以不保留时间的序列，因为接下来下一棵树不包含任何对最后树应用的证实的语义的任何提示(hint)。根据另一个示例，可以使用测量 m(Q_1) 作为 SML_2 的第一个叶。根据另一个示例，可以使用 m(V||(Q_1)||Cert_PCA(a_1)) 作为 SML_2 的第一个叶。根据另一个示例，可以使用 m(SML_1*) 作为 SML_2 的第一个叶。

[0360] 最后三个选项在语义上等价，但是在验证和 / 或生成复杂性方面具有不同的实践意义。树形 SML 的证实的延续可以无限制地反复继续。代替通过证实的内部延续，可以调用可信第三方对 SML_1 树进行证实。

[0361] 树的经证实的延续可以由调用的 TTP 以额外的语义进行扩展。尤其是，如果 SML_1 随根证实请求一起被提交，并被验证，即，通过根的重新计算检查一致性。然后 SML_1 的哈希值可以直接嵌入到从 TTP 获得的证书中。这随后向确认器提供了一个保证 :当针对 TTP 证实被提交时，SML_1 没有被篡改。这使得 SML_1 的根值 V 在原则上已经过时。确认器需要在 SML_1 团点(blob) 上验证全局哈希，以及在其上验证 TTP 证书的签名，来检查 SML_1 的完整性。这是有利的，因为它意味着，对于想要使用 SML 上的参考树用于树遍历确认的确认器，不需要重新计算内部节点哈希操作，可以使用与参考树节点的直接比较。这可以加速对失败组件的搜索。作为进一步的变体，可以将 TPM 滴答(tick)戳或 TTP 时间戳加入到证实中。

[0362] 下面描述了将更具表现力(expressivity) 的确认数据绑定到树形 SML 的验证数据中的一些方法和选项。

[0363] 本身，任意结构的 SML 的表现力限制为组件代码和数据的裸测量。通常，这对于确认器有效处理确认并根据确认结果有效管理平台没有产生足够的信息。由于这个原因，TCG 将 SML 的概念充实到还记录组件标识符、加载时间、参数和相关信息的事件日志中，且 TCG IWG 引入 (实际上是树结构的) IML、复杂、具表现力的数据结构的概念以捕获在经认证的启动中收集的相关联信息。

[0364] 然而，这种丰富和结构化的确认信息可能缺乏与适当相关联结构，诸如树形 SML 的验证数据的绑定。特别地对于树形 SML，出现了问题，关于组件和平台特性的上下文信息或其它有用的确认数据如何可以安全地绑定到包含实际验证数据的哈希树中。

[0365] 在实际的测量过程中，即平台的启动阶段，通过建立用于被测组件的小型子树，元

数据可以加到确认数据中并实质上绑定到 SML 树。上面描述了一个示例的实现(但是,关于大量组件及其测量叶)。在 TFV 的上下文中,测量过程可以如下扩展为在 SML 树中包括元数据。

[0366] 1. 执行测量和 SML 建立(等同于或配合 RTM)的代理加载组件测量清单,包含必须收集哪些数据并将其包括在用于该组件的 SML 树中的规定。

[0367] 2. 如果 SML 树中的下一个叶是右边的兄弟(其二进制坐标的最小有效位是“1”),代理向 SML 树写一个 nil,并使用 TPM_Tree_Extend 对它进行保护。这保证了组件启动一个新的独立子树。

[0368] 3. 代理测量组件本身并将测量值写到 SML,以及利用 TPM_Tree_Extend 将其扩展到根 PCR。

[0369] 4. 对于每个项,代理在清单上重复 :收集需要的元数据 ;将其附加到确认数据序列 ;进行元数据的测量,例如加密的摘要 ;将该元数据测量插入到 SML 树,以及 ;利用 TPM_Tree_Extend 将其扩展到根 PCR。

[0370] 5. 代理通过附加 nil 值填满用于该组件的子树,直到用于该组件的测量序列的长度达到 2 的幂数。

[0371] 这获得图 30 中所示形式的组件子树结构,其中 m(.) 表示合适的测量函数(其对于不同形式的元数据可以不同)。

[0372] 元数据测量可以在其它地方插入,例如内部节点。元数据的插入可以与测量过程在程序上被分离,这可以有助于例如加速平台的启动。接着,元数据可以通过一个不同过程被收集并且安全地存储(至少采用完整性保护的某种形式)直到它插入到 SML 树中。测量代理根据组件测量清单中元素的数量在一个子树中通过对其填写 nil 值来准备足够的空闲空间。在启动过程中结合裸组件测量的树形 SML 完成之后,元数据可以被测量并之后利用 TPM_Tree_Node_Verified_Update 命令被插入到适当位置。

[0373] 可以包括在一个组件子树中并描述其潜在用处的一些元数据有用类型描述如下。

[0374] 组件标识符 :一个组件标识符,包括例如组件生产商,名称,版本,结构,发布日期等,可以是包括的最基础的元数据。尽管原则上组件自身的测量值已经唯一地标识它,一个独立的标识符以一种重要方式增加了语义。它声称平台尝试独立于测量结果加载已命名的组件。因此,特别地,如果组件代码或数据被危害,这向确认器提供关于哪个组件的完整性验证失败的独立的信息。这信息非常有用,因为它允许为有问题的组件找到正确的 CRV。作为组件的顺序准则对于上述方法也是有用的。根据 TCG 标准的事件日志或 IML 可能已经包括了组件标识符,但是它们通常不由验证数据保护。此处它们可能绑定到 SML 树的验证数据,从而获得由所提供的完整性保护的更高保证水平。组件标识符可能已经是一种供包含的格式(例如 160 比特的串),或者可以被测量以获得这种格式。

[0375] RIM 证书 :TRV 的适当标识可能是用于绑定到组件的 SML 树的有用元数据。它可以实现为,例如,相应 RIM 证书的指纹。它可以用于,例如(确认验证器)根据正确的证书,在自己的数据库中或经由 TTP 寻找所包含的 RIM(TRV),或者用于获得证书状态信息。在这种情况下,RIM 证书恰当地不需要合并到平台确认数据中,并可以不被传送到确认器。如果平台的启动过程安全,例如,实现为安全启动过程,那么相应 RIM 证书信息的包含获得额外的语义。即,它宣称由那个特定 RIM 证书证实的 TRV 用于与组件测量值的比较。这种情况下,根

据例如取证评估的需要，树形 SML 中组件的实际测量值可能是过时的。

[0376] 组件上下文：组件上下文可以包括组件加载到其中的运行时间环境的描述。

[0377] 时间戳：作为元数据增加的受保护的时间戳通过向确认器提供组件的加载事件将上下文加到测量中。这允许确认器对组件的加载顺序进行验证，如果必须确认组件的依赖性，则该验证很重要。它还允许评估单个测量的新鲜度，且如果数据结构允许这个，则确认器可以请求组件的新测量。这可以基于测量老化过程来完成，其中测量可以具有定义的生命周期，在该生命周期之后确认器可以请求组件的新测量。此外，将单独的时间戳应用到测量允许对结构化数据重新排序并使得确认器能够得到原始的启动加载器(boot-loader)。

[0378] 组件开始参数，诸如运行时间 / 前置时间设置，可以根据设备的当前情况或运行环境进行不同的设置。

[0379] 组件安全策略：当平台中包含执行实体时，与每个组件相关联并由实体执行的安全策略对于确认器评估平台可信度是重要的数据。通过组织子树中的组件测量和元数据，可以将其确认委派给 TTP，诸如上述描述的 SCA，以用于子树证实方法。

[0380] 组件调用时的地理位置信息为测量的表现力增加了另一个维度(位置)。这个元数据字段对于移动的设备是特别有用的，其中移动设备经常改变位置，而且应用还可以是位置相关的(例如，资产 / 货物跟踪报告软件)。代替在特定时间报告组件被测量，这还可以包括测量的位置。

[0381] 对于 TFV 的实际实现，组件子树可以包含大多属于“静态”性质的数据以避免下述的识别问题。如果期望的元数据包括频繁改变的数据或者对于平台相当独特的数据，为这些数据保留由其自身的根验证数据寄存器保护的完全独立的 SML 树或组件 SML 树的指定子树是有意义的。在这样一棵短暂的元数据 SML 树中的位置可以通过顺序、组件标识符的重复或组件测量叶简单地绑定到原始 SML 树，或者通过任何其它方式将短暂的元数据 SML 树的内容链接到原始 SML 树中的相应组件子树。

[0382] 像线性成链 SML 和合成的 PCR 一样，树形 SML 和根验证数据寄存器对于测量值序列的微小改变很敏感。因此，由于已知的树和子树根的值依赖于精确的测量叶顺序，在有效识别已知的树和子树根方面出现了问题。对这个问题的解决方案的不同可能性进行了描述。它们包含测量以根据对平台和确认器是已知的顺序识别测量叶，因此减小了源于序列敏感性造成的组合复杂性。还描述了根据用于有效 TFV 的技术标准识别叶测量的方法和技术过程。

[0383] 由 Merkle-Damgard 变换得到的 PCR 值，即，非关联和非交换运算的单向函数的合并结果对于输入的顺序非常敏感。这个原则从安全的角度看是非常需要的，这种变换的一个特性反映了检测输入数据最小变化的能力。特性同样保持分别用于树形线性成链 SML 和根验证数据寄存器、PCR。当来解释所包含的信息时，保存 Merkle-Damgard 变换特性的该信息导致了复杂性。

[0384] 当确认器尝试根据提交的树形 SML 对平台进行评估时，排序复杂性问题在 TFV 中格外显现出来。本文讨论了用于搜索失败组件的一般性树遍历过程。与甚至对于高比例的失败(即，确认器未知的)叶测量的线性 SML 的评估进行比较时，它表现得很有效。然而，这种确认模型中的失败也可能由于修改或序列失败导致的，即，在相对于确认器使用的参考树的未知位置中的测量。两个已知测量叶的单个置换(permutation)产生两个未知的测量。

插入或省略在某一位置的测量叶使所有后面的那些，即，该位置右侧的叶位置无效。没有充分考虑这个问题可能损失 TFV 的效率增益。

[0385] 平心而论，当与线性 SML 相比，这真不是一个的问题，因为它们具有相同的排序敏感性，且树形确认不比线性情况更复杂和 / 或计算花费更大。对于最后得到的线性 SML 的 PCR，作为为输入测量的所有置换保持的参考 PCR 值的需求，即，作为空间复杂度，可能出现同样的问题。另一方面，TFV 可能失去部分由排序问题带来的其有利特性。

[0386] Merkle 哈希树可以用于各种研究与示范系统中存储器的运行时间验证。排序问题没有出现，对于存储器保护树的叶的存储器地址的静态指派是很平常的。恢复（如果仅仅部分）平台和确认器都已知的顺序是下面描述的方法背后的一个想法。

[0387] 考虑的这个问题具有信息论的性质，因为它考虑平台（发送方）和确认器（树形确认数据的接收方）之间信息的不对称性。被测组件的序列是可以以有效率方式在双方共享的那条信息，其中效率的意思特别在于确认器既不用保存参考确认数据（SML 树）的过大规模数据库，确认器也不必进行昂贵的匆忙的预先计算。

[0388] 信息的有效预共享是满足所描述需求的一种方法。为此，编码将由平台和确认器使用，通过双方都在 TFV 过程中遵守的约定进行建立。下面描述的方法包括这一总体策略的实现。下面描述的方法在适用范围上不限制为树形验证和确认数据的情况。尽管它们带来的好处可能是有限的，它们也可以直接应用到线性 SML 的传统情况中。由于后者数据的固有结构化，由用于平台确认的验证数据的传统排序带来的效率增益增加了（假设验证数据的合理实现是依赖于顺序的）。因此，排序方法可以应用于其它结构，诸如（直接或非周期的）图形或注释跳跃列表（其还可以是用于快速搜索的优化结构）。

[0389] 除了所描述的排序方法，还有另外一种方法解决由确认器识别验证数据的问题。即，确认器可以随时间学习典型平台配置。这对于频繁改变配置和不太频繁确认的平台不太有效，诸如个人计算机，因为几乎对于每一个启动周期配置都可能不同，或者每周更新是常见的，而用于远程工作场所而重新连接到公司的 VPN 仅发生在一周中的某些天。这样一个 PC 的配置，经由其 PCR 值传送，通过远程证明和附加协议（诸如 TNC）向网络证明，该配置可以每次都不相同。另一方面，移动设备，M2M 网关，和特殊的嵌入式设备像 HeNB 和多媒体 STB 具有更多的静态配置，同时连接到网络—从而更频繁地进行确认。而且，对它们的更新由可以了解已知设备配置的网络实体进行管理。对于这样的设备，如果确认器应用设备配置的学习策略，这可能就足够了。例如，确认器参考 SML 树可以在设备的首次确认时根据已知 TRV（例如，具有相应证书的 RIM）被建立。对于具有相同配置的设备，这样一棵树可能实际上不能用作参考。这些学到的参考树的子树在另一平台的首次确认时，甚至可以通过识别它们的根值而被找到，并适当地用于新学到的参考树的确认和组成。

[0390] 存在两个基本选项来提供对树形 SML 的期望排序。其一是，在由 RTM 影响测量之前应用已知的顺序（预排序），其导致一个被操纵（rigged）的 SML 形成，或者测量过程保持不变，稍后以安全方式对完整 SML 树应用排序（后排序）。

[0391] 预排序变化的一个实现是程序的实际加载顺序，从而符合给定的 SML 排序。如果这在特定平台上可行，它反而可以使安全启动和测量过程不受干扰。另一选项是将测量序列存储在安全缓存中并仅在平台启动后将它们插入到受保护的 SML 树中。然而这种方法可能需要将测量值安全发送到缓存，其中它们可能接着被进行完整性保护，直到执行了所有

的测量。然后，排序操作可能必须作用于存储的数据，不允许对它们进行修改。因此这个选项可能具有额外的需求，如果不能适当满足就会带来安全漏洞。

[0392] 后排序很有趣，因为它允许“懒惰执行”作为系统已经运行时的后台任务。应用的时间限制是当 SML 用于确认时 SML 可以被排序。由于现在已经存在一种直接的方式来安全地混洗（shuffle）树形 SML 的叶，后排序可以对第二个空的 SML 树进行操作，该树由另一个根验证数据寄存器进行保护。请求根据原始 SML 树建立后排序的 SML 树的步骤由平台上的某个可信代理执行，如下。后排序代理针对期望的 SML 序列在组件标识符的序列上重复，并且：1. 识别下一个被识别组件的叶位置（为此，需要实际被测组件的标识符。它们可以记录在测量过程的事件日志中，或者直接附上，诸如，例如以摘要的形式，作为组件的测量叶的兄弟）。2. 使用例如别处描述的 TPM_TreeNode_Verify 命令，针对根验证数据寄存器验证叶测量。3. 一成功，使用 TPM_Tree_Extend 将测量插入到排序的树中。以及 4. 例如由于依赖性，预排序可能对平台的所有组件不是都可行。

[0393] 广泛的后排序可能并不是期望的，因为它引起计算上的成本。因此预排序和后排序的混合形式是建议的。具体地，平台组件的子集可以是预排序的，而组件的另一可能更小的部分在测量之后经历后排序。这可以从测量得到至少两棵 SML 树，其随后例如通过子树证实被加入到一棵完整的测量树。

[0394] 虽然预排序和后排序位于平台且是在传送确认数据之前执行的操作，但第三个不同的选项是将某些计算负担强加到确认器上以获得正确的参考树排序。为此，平台可以传送组件标识符的序列，指示树形 SML 或其子树的测量排序。可替换地，平台可以发送叶测量值的普通列表，通过它确认器还能唯一识别每个叶处的组件（如果是已知的）。接着确认器可以使用该组件标识符序列的接收和实际确认之间的时间间隔以可以期望的顺序建立参考树。这可能比建立平台的完整参考树需要更少的哈希运算，至少是在置换的叶的部分不是很大的情况下。这种情况下，平台 SML 树的许多子树可能已经被包含在确认器的树中，并可能被重新使用。一从平台接收到排序信息，确认器的任务就是识别对应于已知子序列的叶的子序列，在该子序列相应的已知子树可以被插入到顺序正确的树中。

[0395] 如果用于确认的完整 SML 树的大部分按照惯例具有固定序列，且较小的子树对于确认器具有未知的序列，那么这个过程是有效的。在这种情况下，平台可以与用于确认的完整 SML 树一起或在其之前提交子树的序列和其在完整树中的位置，确认器可以根据参考值建立未知序列的子树，并将其插入到其参考树中的正确位置。

[0396] 一个实施是使用组件的字典排序，其中排序的数据可以是组件的参考测量值，例如 RIM 值（不是由 RTM 获得的实际测量，如果组件受损其可能不同）。另一种排序标准可以是组件的唯一标识符或者其摘要值。根据上述方法，该数据也可以被包括在树形 SML 中。字典排序的一个变体是给测量值和 / 或其它数据指派一个编码并使用得到的代码作为字典排序标准。

[0397] 被测组件可以根据其失败概率进行排序。为了实现这个，确认器可以对已知组件定义失败概率等级，例如根据从众多被确认的平台估计的失败率。之后某失败率等级的组件可以被传递给平台并可以被包含到保留的子树中用于后一等级。这可能得到一棵为了树遍历确认而被优化的 SML 树，因为其包含很可能在遍历中从搜索中丢弃的子树（具有低失败概率的组件）。

[0398] 如果平台的组件具有分层组织，那么测量可以相应地被排序。作为一个示例，如果平台是由包含下级模块的功能组成，那么子树可以被保留用于功能并包含其各自模块的测量。在属于功能的子树内部，可以使用任何可以应用的其它排序。反过来，功能子树可以被排序以合并到平台的完整树形 SML 中。

[0399] 在系统组件的分层组织中，例如根据功能模块关系，可能存在影响最后得到的测量序列的依赖性。一个非常普遍的情况是模块属于(类似于共享的库或类似的)不止一个功能。这种模块可以在平台的安全启动期间被加载并因此被测量一次。在由功能分层组织的 SML 树中正确地捕获这种模块的一种方式如下。执行安全启动的代理可以了解所有已加载的模块，从而能够识别重复 - 例如如果模块标识符在规定的测量序列中再次出现。当建立分层有序的 SML 树时，代理在碰到模块的重复时，首先寻找其在 SML 树中首次出现的地方，接着在这个首次出现的位置验证实际测量值，然后在新位置将其插入到树中。

[0400] 当平台加载了一个其完整性被正确验证(针对参考值)但是对于确认器来说是未知的组件时，发生了特殊情况。不按顺序插入到 SML 树可能导致失败的组件和以后组件的放置失败。如果确认器和平台共享关于哪些组件是确认器知道的知识，那么平台可能能够将未知测量值排序到独立的子树中，同时以确认器知道的顺序准备已知测量的子树。然后两棵树可以结合用于树遍历确认，例如，使得已知 SML 树是左子树，而未知组件树是右子树。后者在确认器的参考树中被准备为包含 nil 值的空树。那使得对于已知(左)子树的树遍历确认不受干扰地进行。

[0401] 上述系统、方法和装置生成并使用分层结构，诸如树结构，用于验证数据和确认。下面描述的是在网络确认中使用这种树结构的进一步的实施方式。

[0402] 树形确认(TFV)是用于复杂类型网络侧平台确认的使能者。TFV 是结构化确认的实现，其意味着由确认器使用结构化数据来评估平台的状态。数据的结构可以在一定程度上反映平台的结构，从而帮助确认。结构化确认是在别处讨论的平台确认和管理(PVM)中定义的抽象平台确认和管理过程的实现。考虑了有效并有效率的远程 PVM 的平台展示了一个模块化的分层架构。引入了平台确认操作的数据的类别。

[0403] 在描述 PVM 时，可以定义在平台和确认器之间传输的数据和其它便于平台管理功能的实体的基本概念。根据安全性相关特性和语义内容可以将它们分为三种类别。

[0404] 存在四种类别的由平台产生并在整个 PVM 过程中使用的用于 PVM 的数据。对于确认平台它们是固有的。一个不同的额外类别是可信参考数据，用于比较确认数据和已知的好值。它可以位于确认器，在平台的确认中使用，或者在平台内部，其中其可以用于安全启动或者作为确认和 / 或管理数据被传输到确认器。这在图 25 中示出，其中描述了概念间的相互关系。

[0405] 验证数据是可验证地识别平台状态，或者具有良好定义的保证等级的部分状态的数据。验证数据在验证过程(例如，在平台的安全启动期间和 / 或在运行期间)中在平台内部生成的。验证数据的一个保护目标是完整性，其可以在其生成(即验证)期间，且至少遍及一个平台的操作周期(例如，启动周期)被维持。另一个保护目标是新鲜度。定义(并提高)验证数据的保证等级的一种方式是分离验证数据的一部分作为具有特殊保护等级的受保护的验证数据，例如，通过硬件保护，在这种情况下存储空间称为验证数据寄存器。

[0406] 验证数据可能需要受保护的绑定，例如加密的，来保护验证数据。绑定的强度和保

护的强度定义了验证数据的保证等级。下面描述验证数据和验证的某些实现。

[0407] SML 可以包括在经认证的启动过程中根据加载的组件的测量生成的 160 比特 SHA-1 值的一个序列。这些哈希值唯一地识别加载的组件。SML 值的保证等级可能依赖于：执行测量的 RTM 的安全性和例如由验证数据寄存器(例如 PCR)执行的 SML 的完整性保护。一个变体是 TFV 的树形 SML。

[0408] TPM 的 PCR 典型地表示受保护的验证数据寄存器。用于 TS 的内部验证的一个通用方法是经认证的启动，并使用 TCB 的能力在 TS 初始化时(例如上电时)评估加载的或启动的软件或硬件组件的可信度。经认证的启动通过在 TS 的其它部分之前启动 RoT 和 TCB 的特定功能来实现。这些部分作为用于测量的 RoT (RTM)运行。这意味着后来启动或加载的组件被测量，即，它们及其启动后的状态和配置例如通过在硬件组件嵌入式代码和加载的程序的(二进制)表示上形成加密的摘要值而被唯一识别。根据具体需求，测量值可以存储在安全存储装置，例如 PCR 中，其形成验证数据的受保护部分。

[0409] 安全启动是经认证的启动的扩展。对于可能具有一些独立和离线功能需求的设备，诸如机顶盒或移动手机，这是非常重要的。配备安全启动的设备的共同特点是当它们无法例如，在网络接入前将关于其可信度的断言传达到外部时，它们可以在一组可信的状态中运行。在安全启动中，TS 配备有本地验证器(验证实体)和管理启动过程的本地实施者(enforcer)，其建立策略执行点(PEP)和策略决定点(PDP)的组合来控制安全启动过程。本地验证器比较新加载或启动的组件的测量值和参考完整性测量(RIM)值，其位于 TCB，或者在 TS 内由 TR 保护，例如，它们位于受保护的存储空间，并决定它们是否分别被加载或启动。因此，确保了系统引导到定义的可信状态。

[0410] 在安全启动，经认证的启动的扩展中，平台配备有本地验证器(验证实体)和管理启动过程的本地实施者，其建立 PEP 和 PDP 的组合来控制安全启动过程。本地验证器比较新加载或启动的组件的测量值和 RIM 值，其为于 TCB，或者在 TS 内由 TR 保护，例如，它们位于受保护的存储空间，并决定它们是否分别被加载或启动。因此，确保了系统引导到定义的可信状态。在安全启动中使用的 RIM 的序列包含验证数据。此外被测量的序列可以扩展到验证数据寄存器中作为受保护的验证数据。在这种情况下，RIM 证书(更精确地说是它们的加密指纹)还可以是验证数据的受保护部分，具有由发行它们的 TTP 提供的保护以及基本的证书基础设施。

[0411] 一些研究者提出了用于 WSN 的平台验证的一种奇特变体，称为“软件证明”。它包含在执行探测代码中，探测代码由远程确认器发送到平台，其检查平台状态，例如存储器的内容。在传统的零知识证明中，返回给确认器作为验证数据的信息可以是探测的运行时间。模糊在一定程度上确保了攻击者可能能够在可测量的延迟之后产生返回信号。

[0412] 不同于验证数据，确认数据是能够被收集并提交给另一方(确认器)的数据的验证数据的超集(superset)，并用于评估平台状态的可信度，尤其是通过针对所包含的验证数据对其进行检查。提交确认数据给确认器的过程，例如，实现为根据 TCG 的远程证明和由确认器执行的它的评估，恰当地称为确认。确认数据可以包含验证数据，诸如引用的验证数据寄存器(例如 PCR)值。确认可以除验证数据的加密验证外，包括确认器进行的策略评估和动作触发，使用例如额外的可能与确认数据相关联的管理数据。

[0413] 与验证数据类似，确认数据识别全部或部分系统状态，但是此外还提供更多信息

使得确认对于确认器是有效率的并确定确认结果的粒度等级。用于确认数据的例子有：命名的系统特性诸如由 PBA 生成的，或者组件的名称和版本，它们的状态和参数，如 TCG 定义的 TS 的平台证书，以及系统组件的厂商证书；TTP 的名称，其中可以取回进一步的数据，例如证书链信息；组件－子组件关系，例如在嵌套 XML 数据结构中捕获的，由 TCG IWG 指定的；用于确认的平台身份，称为确认身份，例如，通过 AIK 密钥对实现。

[0414] 确认的一个方面是将确认数据绑定到相应的验证数据。这种绑定在传递信任链的精神上传递验证数据的保证等级给确认数据。因此，这种绑定可以例如通过数字签名来验证。确认数据的概念可以通过绑定的需求进行限制。确认数据是这样一种数据，其可以由确认器使用验证数据进行验证，特别用于数据完整性。因此，确认数据没有下面定义的一般管理数据那么随意。

[0415] 远程证明是由平台签名的确认的初始阶段，即确认数据到确认器的安全传输的一种实现。反过来，这种绑定的一个例子是 TPM_Quote 操作，其用 AIK 签名 PCR 值。通过验证该签名，并根据 SML 重新计算 PCR 并将 SML 测量值与在确认数据中命名的已知组件的 RIM 相关联，确认器可以验证命名的组件是平台已经在例如经认证的启动期间测量的组件。

[0416] 管理数据包含确认数据并作为其补充。它根据确认数据和结果向特别用于平台管理的其它数据增加表现力。管理数据到确认数据的绑定是符合逻辑的，即，管理数据的元素象征性地链接到确认数据中的相关联的元素，反之亦然。管理数据的可信度（例如由其源）可以分开根据平台的确认进行评估，特别是在管理数据来自 TTP 的情况下。

[0417] 管理数据的典型示例有：从确认结果推断管理行为的策略；可以取回代码更新的位置；事件报告数据；用户通知数据；提供给受确认结果制约的确认平台的服务凭证。

[0418] 可信参考数据是用于比较确认数据和已知好值的数据。那些构成可信参考数据的值称为可信参考值（TRV）。它们最为人所知的例子是，例如在 TCG 的 MPWG 规范中指定的 RIM。它们可以真正地，a）在安全启动中由平台本身使用以确保启动了测量值符合 TRV 的组件，或者 b）由确认器用于比较确认数据和已知好值，从而在确认中评估平台状态。

[0419] 这样，可信参考数据通过某些关于它的安全性断言而变得可信，其可以由确认器或代理使用讨论的 TRV 验证。这种可验证的断言可以，例如由 TTP 发布的数字证书实现，在具体的示例中引致所谓的 RIM 证书。可信参考数据的信任断言还可以包括额外的信息，例如关于组件或平台的外部评估（例如，根据通用标准 EAL）。

[0420] 分割确认描述了一种概念，其允许将确认任务分布到两个（或更多）联网的实体中。这个想法专注于确认过程本身，而不是在特定结构。然而，选择了一个架构模型来提出一般概念，主要是 M2M 设备充当用于连接到它的 M2M 设备的 M2M 网关的架构，该 M2M 网关连接到 MNO 的网络。概念本身不限制到那个架构，并可以应用到不同的架构，其展示了某一分层架构（例如，HeNB 和连接的 UE/WTRU，在簇中具有主节点的设备簇，等等）。

[0421] 本文描述的一般方法，诸如子树证实过程和 / 或发现选项是实现本文描述的分割确认的基本技术。

[0422] 假设一个客户端－服务器模型，其中客户端 C 想要访问服务 S，如果 C 处于可信状态，服务器向 C 提供服务。C 可能能够向 S 传递确认数据 v（例如测量值）。

[0423] 在这种情况下，S 可以具有从接收的确认数据 v 中推导出 C 的状态的工具，其可以由参考值 r 实现，S 可以将 v 与 r 进行比较。参考值 r 不限定为给 v 提供已知好值的参考

度量。参考值提供接收的数据 v 和设备可信度之间的映射。因此，r 可以看做 v 的映射来访问用于服务的策略，这意味着 r 可以是任意类型的参考，允许 S 根据接收到的 V 推导出关于 C 的可信度的声明。

[0424] 在某些情况下，C 可以经由网关设备 G（例如 M2M 网关，作为 WTRU 的网关的 HeNB）连接到 S。如果 G 能够执行连接的 C 的确认的初始部分，即，G 配备有用于所连接设备的 r，那么用于确认任务的负荷可以被分布在 G 和 S 之间。

[0425] 在两个联网的实体之间的确认的分布被称为分割确认。为了进行分割确认，对于 G 确认单个连接的 C 必须是可能的，并且 S 可以相信 G 以一种安全的方式执行这一确认。而且，如果确认数据 v 展示为一种内部（分层）结构，执行有效率的分割确认可以更容易。

[0426] 作为分割确认的一个实现，实体可以能够使用树形确认（TFV）数据。G 可以能够生成它自己的结构化的确认数据。G 可以能够执行树更新，包括子树的移除和增加。G 可以被配备一个测量代理（MA），利用 MA 能够整合 C 的确认数据到 G 的确认树中。这些子树可以直接被 G 或者被 TTP 证实的（并且因此被确认和签名）。

[0427] G 内的 MA 从连接的 C 中收集测量并整合它们到 G 的树中作为新子树。测量的收集可以在 C 向 G 的本地表示（signify）验证期间（例如，通过执行 C 到 G 链路上的 SAV、混合确认、远程确认）或者在一个额外的协议步骤中认证之后被执行。这一步骤可以被绑定到 C 中的一个设备认证，以防止重放并且伪造确认数据。然而，可以不需要这一认证来表示由 C 用来访问 S 的认证（例如，用于接入网络的 MNO 凭证），因为当 G 为这一特定的 C 建立到 S 的连接时，这一认证可以在额外的步骤中执行。如图 31 所示。

[0428] 在收集相关数据之后，G 联系向 G 发布子树证书的 TTP。G 之后能够把证书合并到它自己的确认树中，形成一个新的并更新（up-dated）的确认树。作为一个变体，TTP 可以也成为 G 的一部分，例如，实现为一个运行在 G 的 TrE 内部的或者由 TrE 进行完整性保护的应用。这在图 32 中示出。

[0429] 在最后一步中，G 发送更新后的确认树给 S，包括为连接的设备替代一些或全部子树的证书。为了示例的目的 S 可以分解为两个子单元，确认实体 VE 和服务实体 SE。VE 能够确认从 G 接收的数据（自主地或在 TTP 的帮助下），而 SE 负责正提供给 C 的实际服务（以及可选地提供给 G 的服务）。在 G 不能用访问 S 的凭证对 C 进行认证的一个变体中，SE 还可以执行认证，例如，S 可以执行 3G 认证，而 G 对设备中的 TrE 进行了认证以验证接收的确认数据和所连接的 C 的设备 /TrE 身份的真实性。这在图 33 中示出。成功确认之后，该服务可以提供给 C。

[0430] 在分割确认中，G 可以保持连接的 S 对 C 的隐私，因为 S 可以接收代替所连接设备的子树的证书。S 可以信任 TTP 和 G 中的 MA，其中对 MA 的信任可以从 G 接收的确认数据导出，因为确认数据包含 MA 的测量。

[0431] 在另一个变体中，分割确认可以扩展为支持动态更新，包括支持在不同 G 之间漫游 C，涉及 G 之间的 C 子树切换。用于分割确认的另一个变体是在网关 G 处单个 C 的部分确认，用证书只替换子树的部分。

[0432] 本文描述的方法和算法考虑了用于分割确认的操作的有效实现。它们根据允许设备建立、报告、更新和引用树或子树的 TPM 的扩展版本使得分割确认能够使用。而且，算法通过子树证实规定用证书替换子树节点的有效替换。

[0433] 分割确认中的一个问题是确定哪些 C 可以由 TTP 确认(从而替代它们在 G 处的子树)以及 C 的哪个部分可以由 S 确认,即,哪些参考值在 G 可以用来证实所连接的 C 的子树的 TTP 处可用。一般地,这种发现问题的解决方案可以分为三种类型 :基于 TTP 的、基于网关的和共享发现。

[0434] 基于 TTP 的方法适用于缺乏用于执行复杂分析以为证实发现合适子树的计算能力的 G。在这种方法中,TTP 从 G 接收完整的树,并以用于 TTP 可以验证的子树的一组子树证书的形式返回结果。然后 TTP 可以丢弃它不能证实的剩余数据。证书被发送回 G,带有指示它们适合树中哪些位置的指示符。在 G 处的 MA 执行证书的摄入,即,用接收的证书代替目前经证实的子树。G 可以不必再执行发现,因为 TTP 在具有证书的消息中传递了针对子树位置的指示符。

[0435] 在变体过程中,G 不传送完整的树,而是传送其一部分给 TTP,允许 TTP 从连接的设备搜索测量子集中潜在可证实的子树。这种机制可能是有用的,例如,如果树形确认数据显示了一定结构,这允许 G 根据某些设备特征(例如设备类别)将树分解成多个子树。根据设备类别,不同的 TTP 能够确认一特定组的设备。

[0436] 在基于网关的发现中,G 具有度量来决定哪些 C 可以由 TTP 证实,且 TTP 可以接收必要的数据。这种方法允许最小化 G 和 TTP 之间的通信量,因为所有被传输的数据属于可证实的子树。然而,G 可以被预先配备有合适的度量,这可以允许 G 发现右边的子树以及可以对其进行证实的其 TTP。而且,在这种方法中 G 可以搜索其自身的确认树以寻找合适的子树,把更多的计算工作量放在 G 上。

[0437] 在共享发现模型中,TTP 和 G 共同努力来确定哪些设备以及因此哪些子树可以被证实。这种发现步骤协议的结果是可以被证实的子树列表。这种发现协议可以包括额外(元)数据的交换,该数据例如是设备类别、能力、设备配置文件、动态参数(诸如负荷和带宽)。尽管需要最初的通信确定可证实的子树的集合,该方法还是最小化数据量,因为没有不必要的数据从 G 传送到 TTP。通信负荷可能比在基于网关发现中的多,但是可能比在基于 TTP 发现中的少。G 能够传传协商阶段的结果,这允许 G 后来执行基于网关的决定。在共享发现模型中,G 可以被预先准备有一组可证实的子树,其然后可以在共享发现阶段被更新。

[0438] 在一个变体中,G 完全确认某些 C,其中 C 的选择例如在 G 是毫微微小区的情况下可以基于 CSG (封闭用户组)。S 之后完全确认剩余的 C。将在 G 处被确认的 C 的集合可以基于静态或动态数据,而将被 S 确认的设备的集合将会是剩余的 C 或者可以有重叠部分,意味着一些设备可以被确认两次。

[0439] G 和 S 可以确认方面集合中定义的可以预先规定的或者动态调整的并且可能重叠的方面。这些方面的集合可以包括 :设备配置文件的完整性,启动代码 /OS/ 驱动器的完整性,毛细管网功能 / 组件的完整性,WAN 连接功能 / 组件的完整性,高级应用的完整性,以及 G 处使用的可信参考值的完整性。

[0440] 确认可以基于加载被动态地分割。在这个动态方式中,G 和 S 或者 G 和 TTP 比较它们自身上的以及 G 和被连接的 C 之间的和 G 和 S 或者 G 和 TTP 之间的链路上的计算负荷。基于该负荷,TTP 或者 S 确认直接在 G 处被确认的设备集合。可以存在一个预先规定的集合,之后被动态更新。

[0441] 在一个基于知识的分割中,分割可以基于知识集合和确认策略在 S (或者 TTP)和

G之间被确定。知识集合可以包括若干因素，如行为(过去观察的，或者未来预见的)，可信性(过去的历史，当前的值，或者未来预期)，成本与效益分析，确认重要性等，这可以允许S和G得到或更新确认策略。

[0442] 在一个示例中，H(e)NB可以使用UE的分割确认。在HeNB的情况下，架构可以包括一个能够通过如下过程确认某些或者全部已连接的UE的H(e)NB，如图34所示。

[0443] 如图34所示，WTRU在1处发送完整性测量给H(e)NB。在2处，H(e)NB整合收到的数据并确认某些(或者全部)WTRU。在3处，SeGW/PVE执行包括已连接设备的H(e)NB树的确认。PVE在4处确认尚未被确认的设备并发布一个参考证书，它可以发回给H(e)NB以将被包含在未来的通信中，例如，使H(e)NB能够在未来的连接尝试中确认WTRU。

[0444] 另一个变体可以在公司部署情形下，其中H(e)NB能够确认每个WTRU的组件的子集，例如公司软件，它可能不披露给MNO（SeGW和PVE)。PVE可以执行WTRU的基础平台组件的网络侧确认。

[0445] 在一个流氓设备连接到H(e)NB的情况下，H(e)NB为报告目的可以包括UE的完整性树。在公司网络上的PVE或者确认网络实体可以能够验证接收到的数据并能够接着命令H(e)NB阻止设备的接入或者进行补救步骤，如替代组件或软件，执行防病毒更新等。这在图35中示出。

[0446] 由于M2M架构与H(e)NB有些相同，分割确认可以给M2M情形带来一些好处。在一个典型的M2M部署中，更小但更多设备可能被连接到单个网关，提供对网络的接入。而且在M2M情形中，多种不同的设备可以经由单个网关连接，并且因此网关可能不能够确认所有的设备(或者所有的设备类型)。执行分割确认可以允许网络卸载一定工作量到M2M网关。

[0447] 在进一步的一个变体中，M2M GW可以基于所连接的设备的类型、设备类别、设备特性或者连接配置文件(协议，连接网络，带宽)对所连接的设备进行分组，然后为设备确认树提供组证书。这在图36中示出。

[0448] 另一个变体是端对端方法，其中多个M2M网关(GW)呈现一个簇结构，这可以允许它们在更好的链路(例如，更多带宽，更少等待时间等)上通信。每个M2M GW有一个自己的到网络的回程链路。然而，本地通信可以比利用回程链路(例如3G)更便宜(例如，经由WiFi，蓝牙，ZigBee，MBUS等)，这意味着将业务量卸载到本地网络提供了一定好处。M2M GW间的本地网络被称作局域交换网络(LAEN)。LAEN上所有的M2M GW可以相互地认证，这允许M2M GW信任来自LAEN上其它M2M GW的消息。可以针对完整性和真实性保护LEAN上的通信。在隐私需求情形下，一个完全加密的LEAN可以被建立来提供机密性保护。

[0449] 如果一个M2M GW不能够证实一个子树(例如，没有发现TTP来证实子树或者设备对M2M GW来说是未知的)，M2M GW推出LAEN上的子树(图37中步骤1)，具有一个用于获得合适证书的请求消息。如果另一个M2M GW能够证实这个子树(在它自身上或者经由一个TTP，其可能是第一个M2M GW无法到达或者不知的)，这个M2M GW返回证书给进行请求的GW(图37中步骤2)。M2M GW然后可以整合证书到它自己的确认树中。M2M GW也能够存储证书(或者产生证书所需的参考值，或者TTP的地址)用于将来同设备级别的连接设备的确认。

[0450] 子树的交换可以使用树哈希交换(THEX)格式来执行，这允许交换Merkle哈希树。交换格式包括哈希树的串行化(serialized)表示。它使用考虑了多个净荷的直接因特网消息封装(DIME)格式(文本或者二进制)。Merkle哈希树串行化格式包括两种不同净荷。第

一个是 XML 编码的关于哈希树的元数据，第二个是树本身的二进制串行化。

[0451] LAEN 上能够确认给定子树的网络实体（例如其它 M2M GW）的定位，可以使用分布式哈希表（DHT）结构来存储 LAEN 上能够确认特定设备类型或设备类别的节点的网络地址。从 P2P 网络得知的 DHT 概念，考虑了用于确认的相关节点的快速检测。而且在 LAEN 中的每个节点处存储哈希表，并且经由节点间的 LAEN 组播消息更新该哈希表是可能的，由此 LAEN 中的每个节点可以知道特定子树能够在哪里被确认。

[0452] 以上为分割确认和其它结构化确认技术如子树证实开发的概念和方法可以不仅用于系统确认，也可以用于其它安全相关功能。

[0453] 例如，在用于包含网络实体 NE，网关 G，以及多个设备 D 的网络的分割认证模型中，可以设想到网关 G 当它与 NE 交互使得 NE 可以认证该网关 G 时，可以包括关于 D 的认证消息以使 NE 能够同时认证 G 和 D 的集合。在一个变体中，G 能够认证 D’，并且在它之后发送以传递 NE 认证 G 所需的信息的消息中包括关于认证步骤和结果的报告。如上所述，根据几个不同的准则，划分 D’ 也可以是可能的，以使 G 可以相对 NE 可以认证的 D’ 而进行认证。

[0454] 类似地，其它安全性功能如密钥导出和密钥管理（例如包括分配，刷新，和反对），授权，和接入控制，也可以用类似方式完成。

[0455] 为子树验证开发的技术也可以以类似的方式被用于这些其它安全性功能。

[0456] 方法、算法和装置可以被应用在任何技术领域，包括 ：安全 / 隐私 / 权限管理，混合网络，协作通信，并且可以被用在或用于用户设备（UE），无线发射 / 接收单元（WTRU），手持设备，数据卡，膝上型笔记本 / 上网本，游戏设备，基础设施设备，基站 / 节点 B，毫微微基站，接入点，BSC/RNC，网关，应用服务器，系统，会话层，表示层，应用层，DSP，软件，硬件，ASIC。方法和算法也适用于 ：应用，系统信息，平台安全性，整个系统安全性，呼叫准入，家庭 e 节点 B （HeNB），HNB，毫微微小区，硬件或者软件实现，计费管理，用户管理，策略控制，服务质量（QoS），安全性，信任，译码，注册 /AAA 等。

[0457] 尽管前面以特定组合描述了特征和元素，但是每个特征或元素可以独立于其它特征和元素单独使用或者与或不与其它特征和元素进行各种组合使用。本文提供的方法或流程图可以在结合在通用计算机或处理器执行的计算机可读存储介质中的计算机程序、软件或者固件中实施。计算机可读存储介质的例子包括只读存储器（ROM）、随机存取存储器（RAM）、寄存器、缓存器、半导体存储设备、磁介质如内部硬盘和可移动盘、光磁介质，和光介质如 CD-ROM 盘和数字通用盘（DVD）。

[0458] 以示例的方式，适合的处理器包括通用处理器、专用处理器、常规处理器、数字信号处理器（DSP）、多个微处理器、与 DSP 核相关联的一个或多个微处理器、控制器、微控制器、专用集成电路（ASIC）、专用标准产品（ASSP）、现场可编程门阵列（FPGA）电路、任何其它类型的集成电路（IC），和 / 或状态机。

[0459] 与软件相关联的处理器可以被用于实现在无线发射接收单元（WTRU）、用户设备（WTRU）、终端、基站、移动性管理实体（MME）或演进分组核心（EPC）、或者任何主机计算机中使用的射频收发信机。WTRU 可以与以硬件和 / 或软件（包括软件定义的无线电（SDR））实现的模块和其他组件结合使用，该组件例如是相机、视频摄像模块、视频电话、扬声器电话、振动设备、扬声器、麦克风、电视收发信机、免提耳机、键盘、蓝牙模块、调频（FM）无线电单元、近场通信（NFC）模块、液晶显示（LCD）显示单元、有机发光二极管（OLED）显示单元、数字音

乐播放器、媒体播放器，视频游戏机模块、因特网浏览器，和 / 或任何无线局域网（WLAN）或超宽带（UWB）模块。

[0460] 进一步的工作可以继续这一方向并且考虑具有树形验证和确认数据的平台确认——被称作树形确认（TFV）——的具体架构。一个可想到的选择是利用 SCA 和 / 或确认器以一种考虑了使用已知组件子结构（例如，按序加载的依赖的程序，或者具有统一关联的安全策略的组件）的子树的模块化构建的方式有效地组织参考树的数据库。子树发现、被确认的平台组件之间依赖性的表达以及根据 TFV 的结果对平台进行管理（更新、修正）的架构和方法是正在研究的课题。

[0461] 本文描述的系统、方法和装置可以在通信系统中实现，诸如下面描述并如图 38、39 和 40 所示的通信系统。

[0462] 图 38 是一个或多个公开的实施方式可以在其中实现的示例性通信系统 100 的图。通信系统 100 可以是向多个无线用户提供诸如语音、数据、视频、消息发送、广播等内容的多接入系统。通信系统 100 可以使得多个无线用户能够通过共享包括无线带宽的系统资源访问这些内容。例如，通信系统 100 可以使用一个或多个信道接入方法，诸如码分多址（CDMA）、时分多址（TDMA）、频分多址（FDMA）、正交 FDMA（OFDMA）、单载波 FDMA（SC-FDMA）等。

[0463] 如图 38 所示，通信系统 100 可以包括无线发射 / 接收单元（WTRU）102a、102b、102c、102d，无线电接入网（RAN）104，核心网 106，公共交换电话网（PSTN）108，因特网 110，和其它网络 112，不过应该理解的是公开的实施方式考虑到了任何数量的 WTRU、基站、网络和 / 或网络元件。WTRU102a、102b、102c、102d 中的每一个可以是配置为在无线环境中进行操作和 / 或通信的任何类型的设备。作为示例，可以将 WTRU 102a、102b、102c、102d 配置为发送和 / 或接收无线信号，并可以包括用户设备（UE）、移动站、固定或者移动用户单元、寻呼器、蜂窝电话、个人数字助理（PDA）、智能电话、膝上型电脑、上网本、个人计算机、无线传感器、消费电子产品等等。

[0464] 通信系统 100 还可以包括基站 114a 和基站 114b。基站 114a、114b 的每一个都可以是配置为与 WTRU 102a、102b、102c、102d 中的至少一个无线对接以便于接入一个或者多个通信网络，例如核心网 106、因特网 110 和 / 或网络 112 的任何类型的设备。作为示例，基站 114a、114b 可以是基站收发信台（BTS）、节点 B、演进的节点 B（e 节点 B）、家庭节点 B、家庭 e 节点 B、站点控制器、接入点（AP）、无线路由器等等。虽然基站 114a、114b 每个被描述为单独的元件，但是应该理解的是基站 114a、114b 可以包括任何数量互连的基站和 / 或网络元件。

[0465] 基站 114a 可以是 RAN 104 的一部分，RAN 104 还可以包括其它基站和 / 或网络元件（未显示），例如基站控制器（BSC）、无线电网络控制器（RNC）、中继节点等。可以将基站 114a 和 / 或基站 114b 配置为在特定地理区域内发送和 / 或接收无线信号，该区域可以被称为小区（未显示）。小区还可以被划分为小区扇区。例如，与基站 114a 关联的小区可以划分为三个扇区。因此，在一个实施方式中，基站 114a 可以包括三个收发信机，即每一个用于小区的一个扇区。在一个实施方式中，基站 114a 可以使用多输入多输出（MIMO）技术，因此，可以将多个收发信机用于小区的每一个扇区。

[0466] 基站 114a、114b 可以通过空中接口 116 与 WTRU 102a、102b、102c、102d 中的一个或者多个通信，该空中接口 116 可以是任何合适的无线通信链路（例如，射频（RF）、微波、红

外(IR)、紫外线(UV)、可见光等)。可以使用任何合适的无线电接入技术(RAT)来建立空中接口 116。

[0467] 更具体地,如上所述,通信系统 100 可以是多接入系统,并可以使用一种或者多种信道接入方案,例如 CDMA、TDMA、FDMA、OFDMA、SC-FDMA 等等。例如,RAN 104 中的基站 114a 和 WTRU 102a、102b、102c 可以使用例如通用移动电信系统(UMTS)陆地无线电接入(UTRA)的无线电技术,其可以使用宽带 CDMA (WCDMA)来建立空中接口 116。WCDMA 可以包括例如高速分组接入(HSPA)和 / 或演进的 HSPA (HSPA+)的通信协议。HSPA 可以包括高速下行链路分组接入(HSDPA)和 / 或高速上行链路分组接入(HSUPA)。

[0468] 在一个实施方式中,基站 114a 和 WTRU 102a、102b、102c 可以使用例如演进 UMTS 陆地无线电接入(E-UTRA)的无线电技术,其可以使用长期演进(LTE)和 / 或高级 LTE (LTE-A)来建立空中接口 116。

[0469] 在另一个实施方式中,基站 114a 和 WTRU 102a、102b、102c 可以使用例如 IEEE802.16 (即全球微波接入互操作性(WiMAX))、CDMA2000、CDMA20001X、CDMA2000EV-DO、暂行标准 2000 (IS-2000)、暂行标准 95 (IS-95)、暂行标准 856 (IS-856)、全球移动通信系统(GSM)、GSM 演进的增强型数据速率(EDGE)、GSM EDGE (GERAN)等等的无线电技术。

[0470] 图 38 中的基站 114b 可以是例如,无线路由器、家庭节点 B、家庭 e 节点 B 或接入点,并且可以使用任何适当的 RAT 来方便局部区域中的无线连接,例如商业场所、住宅、车辆、校园等等。在一个实施方式中,基站 114b 和 WTRU 102c、102d 可以实现例如 IEEE 802.11 的无线电技术来建立无线局域网(WLAN)。在一个实施方式中,基站 114b 和 WTRU 102c、102d 可以实现例如 IEEE 802.15 的无线电技术来建立无线个域网(WPAN)。在另一个实施方式中,基站 114b 和 WTRU 102c、102d 可以使用基于蜂窝的 RAT (例如,WCDMA,CDMA2000,GSM,LTE,LTE-A 等)来建立微微小区或毫微微小区。如图 38 所示,基站 114b 可以具有到因特网 110 的直接连接。因此,基站 114b 可以不必经由核心网 106 接入到因特网 110。

[0471] RAN 104 可以与核心网 106 通信,所述核心网 106 可以是被配置为向 WTRU 102a、102b、102c、102d 中的一个或多个提供语音、数据、应用和 / 或通过网际协议的语音(VoIP)服务的任何类型的网络。例如,核心网 106 可以提供呼叫控制、计费服务、基于移动位置的服务、预付费呼叫、因特网连接、视频分配等,和 / 或执行高级安全功能,例如用户认证。虽然图 38 中未示出,应该理解的是 RAN 104 和 / 或核心网 106 可以与使用和 RAN 104 相同的 RAT 或不同 RAT 的其它 RAN 进行直接或间接的通信。例如,除了连接到正在使用 E-UTRA 无线电技术的 RAN 104 之外,核心网 106 还可以与使用 GSM 无线电技术的另一个 RAN (未示出)通信。

[0472] 核心网 106 还可以充当 WTRU 102a、102b、102c、102d 接入到 PSTN 108、因特网 110 和 / 或其它网络 112 的网关。PSTN 108 可以包括提供普通老式电话服务(POTS)的电路交换电话网络。因特网 110 可以包括使用公共通信协议的互联计算机网络和设备的全球系统,所述协议例如有 TCP/IP 网际协议组中的传输控制协议(TCP)、用户数据报协议(UDP)和网际协议(IP)。网络 112 可以包括被其它服务提供商拥有和 / 或操作的有线或无线的通信网络。例如,网络 112 可以包括连接到一个或多个 RAN 中的另一个核心网,该 RAN 可以使用和 RAN 104 相同的 RAT 或不同的 RAT。

[0473] 通信系统 100 中的 WTRU 102a、102b、102c、102d 的某些或全部可以包括多模式能

力，即 WTRU 102a、102b、102c、102d 可以包括用于在不同无线链路上与不同无线网络进行通信的多个收发信机。例如，图 38 中示出的 WTRU 102c 可被配置为与基站 114a 通信，所述基站 114a 可以使用基于蜂窝的无线电技术，以及与基站 114b 通信，所述基站 114b 可以使用 IEEE 802 无线电技术。

[0474] 图 39 是示例性的 WTRU 102 的系统图。如图 39 所示，WTRU 102 可以包括处理器 118、收发信机 120、发射 / 接收元件 122、扬声器 / 麦克风 124、键盘 126、显示器 / 触摸板 128、不可移动存储器 130、可移动存储器 132、电源 134、全球定位系统（GPS）芯片组 136 和其它外围设备 138。应该理解的是 WTRU 102 可以在保持与实施方式一致时，包括前述元件的任何子组合。

[0475] 处理器 118 可以是通用处理器、专用处理器、常规处理器、数字信号处理器（DSP）、多个微处理器、与 DSP 核相关联的一个或多个微处理器、控制器、微控制器、专用集成电路（ASIC）、场可编程门阵列（FPGA）电路、任何其它类型的集成电路（IC）、状态机等等。处理器 118 可执行信号编码、数据处理、功率控制、输入 / 输出处理和 / 或使 WTRU 102 能够在无线环境中运行的任何其它功能。处理器 118 可以耦合到收发信机 120，所述收发信机 120 可耦合到发射 / 接收元件 122。虽然图 39 示出了处理器 118 和收发信机 120 是单独的部件，但是应该理解的是处理器 118 和收发信机 120 可以一起集成在电子封装或芯片中。

[0476] 发射 / 接收元件 122 可以被配置为通过空中接口 116 将信号发送到基站（例如，基站 114a），或从基站（例如，基站 114a）接收信号。例如，在一个实施方式中，发射 / 接收元件 122 可以是被配置为发送和 / 或接收 RF 信号的天线。在一个实施方式中，发射 / 接收元件 122 可以是被配置为发送和 / 或接收例如 IR、UV 或可见光信号的发射器 / 检测器。在另一个实施方式中，发射 / 接收元件 122 可以被配置为发送和接收 RF 和光信号两者。应该理解的是发射 / 接收元件 122 可以被配置为发送和 / 或接收无线信号的任何组合。

[0477] 此外，虽然发射 / 接收元件 122 在图 39 中示出为单独的元件，但是 WTRU102 可以包括任意数量的发射 / 接收元件 122。更具体地，WTRU 102 可以使用 MIMO 技术。因此，在一个实施方式中，WTRU 102 可以包括用于通过空中接口 116 发送和接收无线信号的两个或更多个发射 / 接收元件 122（例如，多个天线）。

[0478] 收发信机 120 可以被配置为调制要由发射 / 接收元件 122 发送的信号，和解调由发射 / 接收元件 122 接收的信号。如上所述，WTRU 102 可以具有多模式能力。因此，收发信机 120 可以包括使 WTRU 102 能够经由多个 RAT 通信的多个收发信机，所述多个 RAT 例如有 UTRA 和 IEEE 802.11。

[0479] WTRU 102 的处理器 118 可以耦合到下述设备，并且可以从下述设备中接收用户输入数据 :扬声器 / 麦克风 124、键盘 126 和 / 或显示器 / 触摸板 128（例如，液晶显示器（LCD）显示单元或有机发光二极管（OLED）显示单元）。处理器 118 还可以输出用户数据到扬声器 / 麦克风 124、键盘 126 和 / 或显示器 / 触摸板 128。此外，处理器 118 可以从任何类型的适当的存储器访问信息，并且可以存储数据到所述存储器中，例如不可移动存储器 130 和 / 或可移动存储器 132。不可移动存储器 130 可以包括随机存取存储器（RAM）、只读存储器（ROM）、硬盘或任何其它类型的存储器设备。可移动存储器 132 可以包括用户标识模块（SIM）卡、记忆棒、安全数字（SD）存储卡等等。在其它的实施方式中，处理器 118 可以从在物理位置上没有位于 WTRU 102 上（例如服务器或家用计算机（未示出）上）的存储器访问信息，并且

可以将数据存储在该存储器。

[0480] 处理器 118 可以从电源 134 接收电能，并且可以被配置为分配和 / 或控制到 WTRU 102 中的其它部件的电能。电源 134 可以是给 WTRU 102 供电的任何适当的设备。例如，电源 134 可以包括一个或多个干电池（例如，镍镉（NiCd）、镍锌（NiZn）、镍氢（NiMH）、锂离子（Li-ion），等等），太阳能电池，燃料电池等等。

[0481] 处理器 118 还可以耦合到 GPS 芯片组 136，所述 GPS 芯片组 136 可以被配置为提供关于 WTRU 102 当前位置的位置信息（例如，经度和纬度）。除来自 GPS 芯片组 136 的信息或作为其替代，WTRU 102 可以通过空中接口 116 从基站（例如，基站 114a、114b）接收位置信息，和 / 或基于从两个或更多个邻近基站接收的信号的定时来确定其位置。应该理解的是 WTRU 102 在保持与实施方式一致时，可以通过任何适当的位置确定方法获得位置信息。

[0482] 处理器 118 可以进一步耦合到其它外围设备 138，所述外围设备 138 可以包括一个或多个提供附加特性、功能和 / 或有线或无线连接的软件和 / 或硬件模块。例如，外围设备 138 可以包括加速计、电子罗盘、卫星收发信机、数字相机（用于照片或视频）、通用串行总线（USB）端口、振动设备、电视收发信机、免提耳机、蓝牙®模块、调频（FM）无线电单元、数字音乐播放器、媒体播放器、视频游戏机模块、因特网浏览器等等。

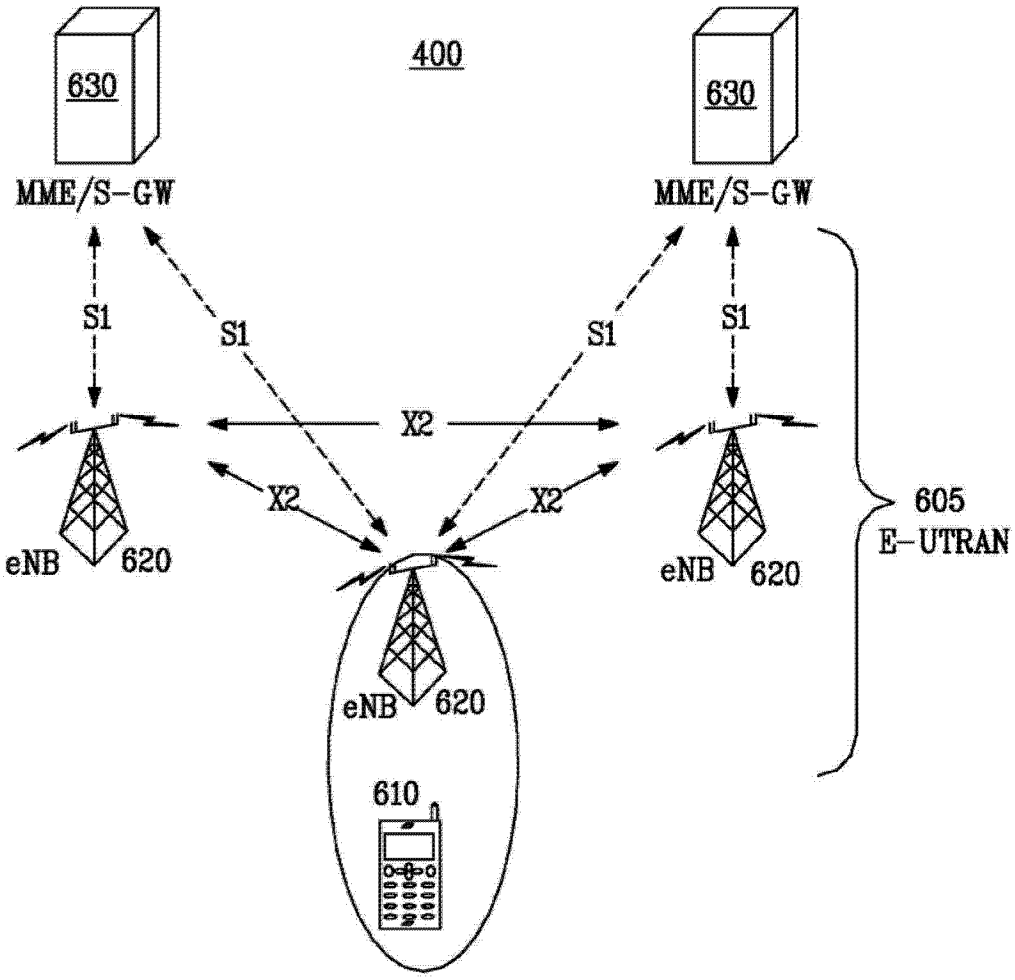

400
630
MME/S-GW
630
MME/S-GW
S1
S1
S1
S1
X2
X2
X2
eNB
620
eNB
620
eNB
620
605
E-UTRAN
610

图 1A

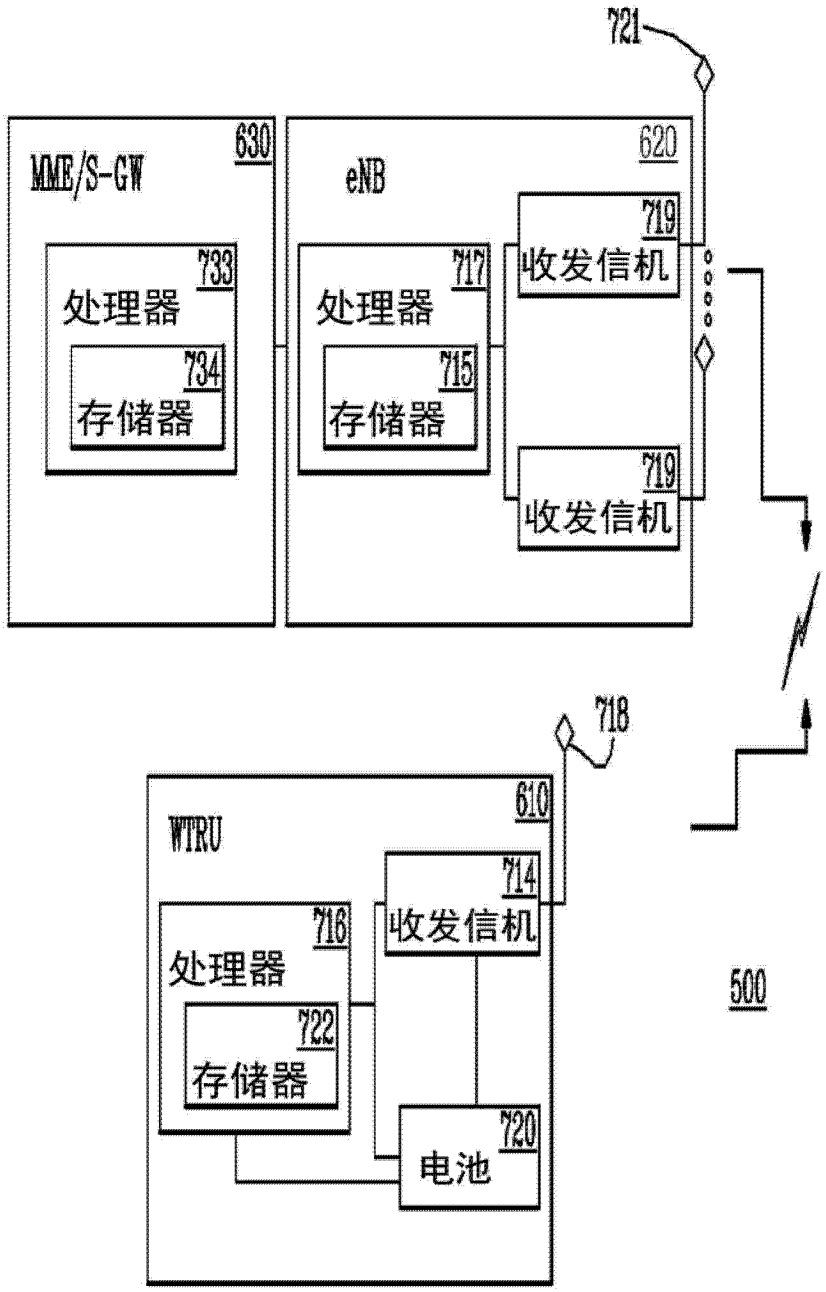

721
630
MME/S-GW
733
处理器
734
存储器
620
eNB
717
处理器
715
存储器
719
收发信机
719
收发信机
718
610
WTRU
714
收发信机
716
处理器
722
存储器
720
电池
500

图 2

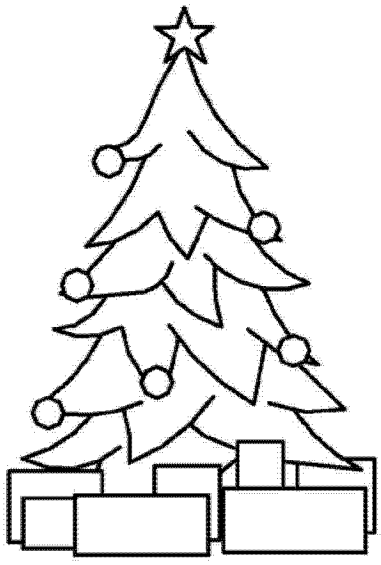


图 3

算法1 树形成算法

```
Require: V_1 ,..., V_d ∈ V, m ∈ {0,1}^x 160
Ensure: V_1 ,..., V_d = nil                    ►将子树的根初始化为空。
 1: n ← 0
 2: while (m RoTM) ≠ nil do                    ►得到新的测量。
 3:     m → SML                                ►如果非空，作为新叶增加。
 4:     if (n)_d = 1 then                      ►从右侧到达的值
 5:         V_d ← V_d ◇ m                      ►对深度为d-1的根进行扩展。
 6:         V_d → SML                          ►净化成SML。
 7:     else                                   ►如果是从左侧到达，
 8:         V_d ← m                            ►放入深度为d-1的根，
 9:         goto line 22                       ►但是不进一步。
10:     end if
11:     k ← d - 1                              ►更新深度为2，...，的子树
12:     while (n)_k = 1 do                     ►只要来自右侧。
13:         V_k ← V_k ◇ V_k+1
14:         V_k → SML
15:         k ← k - 1
16:     end while
17:     if k > 0 then
18:         V_k ← V_k+1
19:     else
20:         break while and return "tree full"
21:     end if
22:     n ← n + 1
23: end while
```

图 4

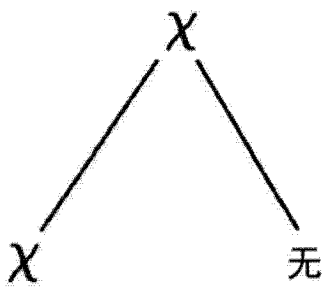


图 5

算法2　　不完整树的清理

```
24: for k ← k - 1 to 1 do
25:     if (n)_k = 1 then
26:             V_k ← V_k ◊ V_{k+1}
27:             V_k → SML
28:     else
29:             V_k ← V_{k+1}
30:             V_k → SML
31:     end if
32: end for
```

图 6

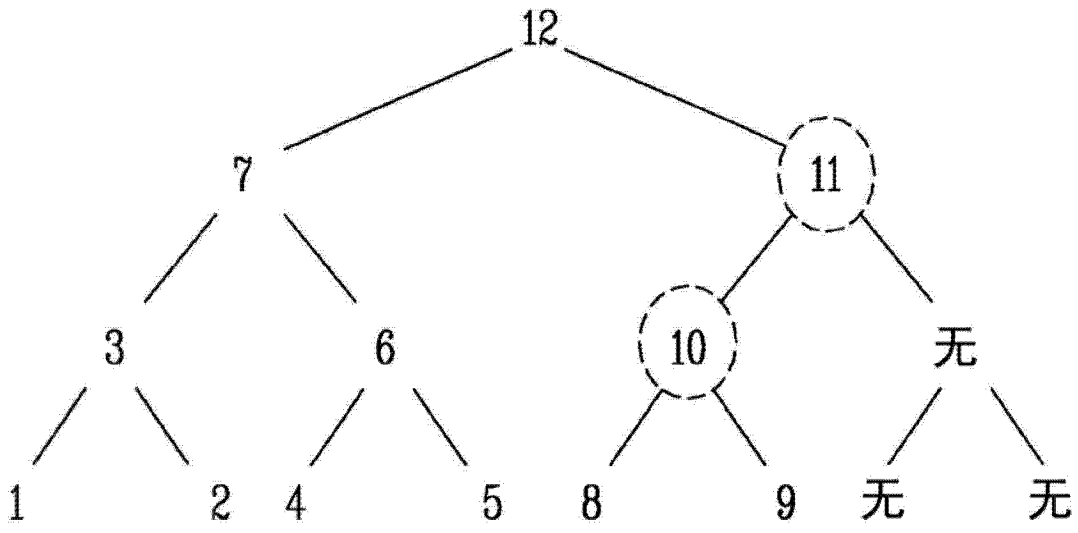


图 7

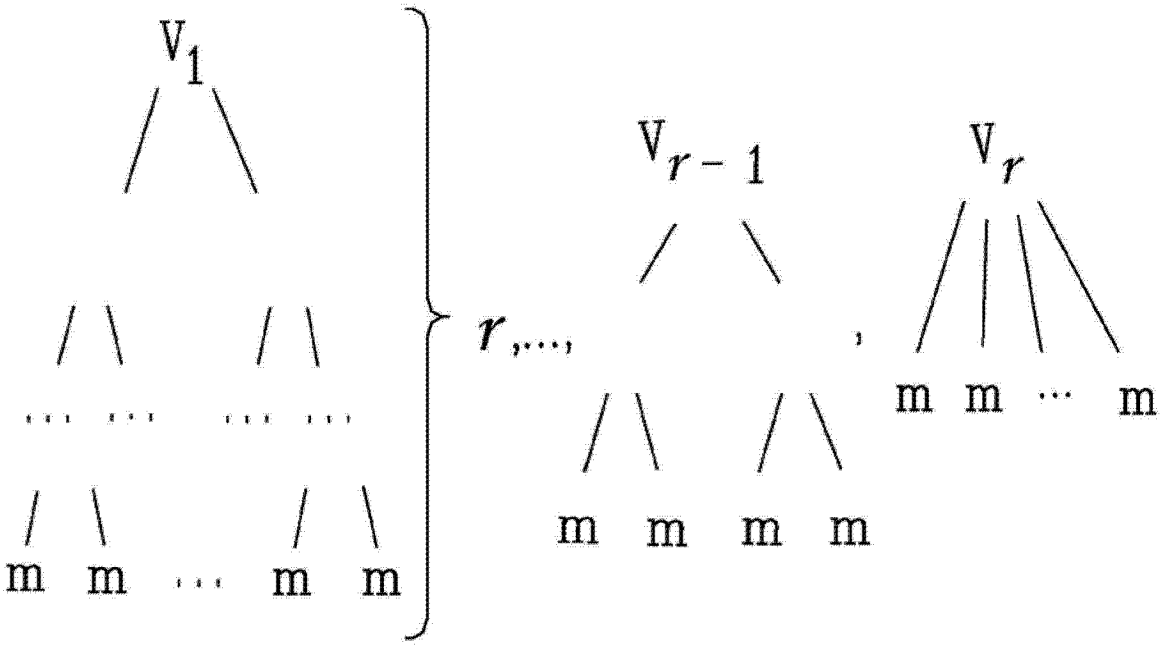

V1
m m … m m
r,…,
Vr - 1
m m m m
,
Vr
m m … m

图 8

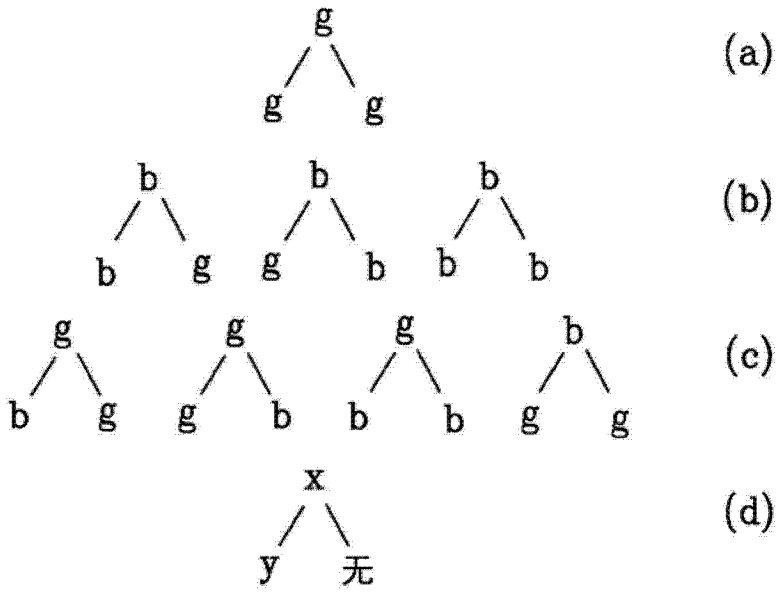

g
g g
(a)
b b b
b g g b b b
(b)
g g g b
b g g b b b g g
(c)
x
y 无
(d)

图 9

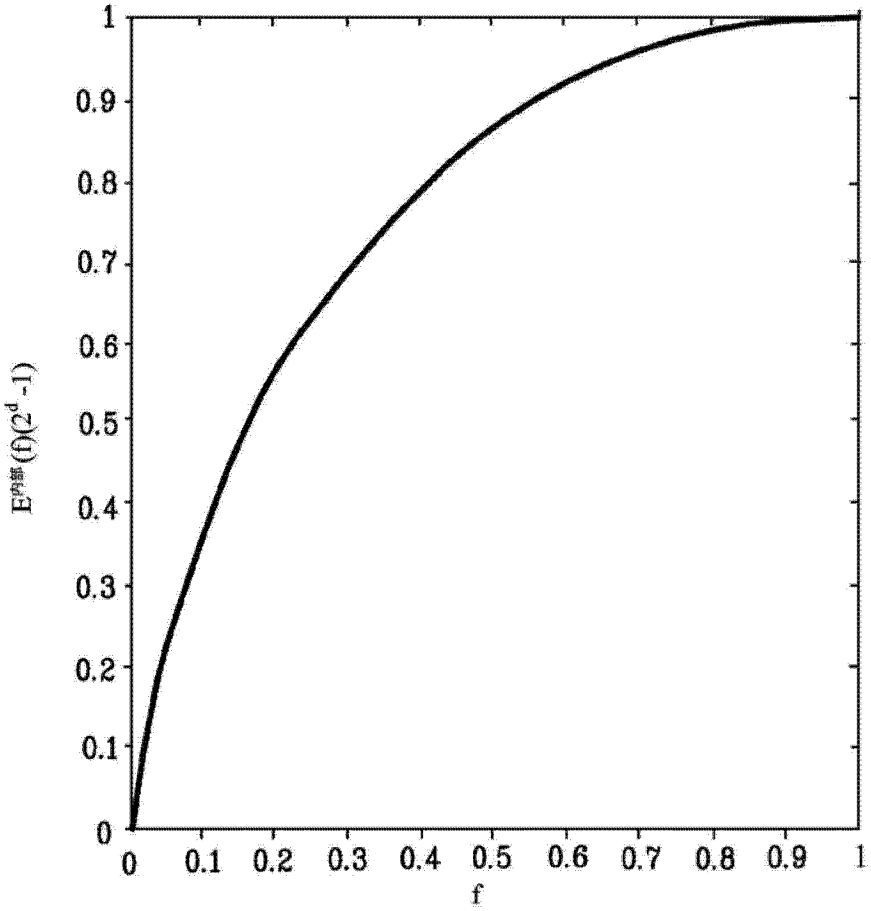

1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
f
E内部(f)(2d -1)

图 10


(0◊x)◊y
x
y

图 11

算法1：在线性链接的哈希链中找到首个失败点

1. 使i代表检测中的测量链的开始索引，j代表结束索引。因此一开始，i=0，j=n-1
2. 如果i=j，去步骤5，否则使用V_0和测量{mk:k=0, 1…, (j+i)/2}计算$V_{(j+i)/2}$，并将它与相应的参考测量$R_{(j+i)/2}$进行比较。
3. 如果它们相等，那么我们知道值V_i到$V_{(j+i)/2}$是好的。设置i=1+（（j+i）/2）并返回到步骤2。
4. 如果它们不相等，那么我们知道故障在于值V_i到$V_{(j+i)/2}$。设置j=（j+i）/2并返回到步骤2。
5. 索引i是线性链中V_i与其参考值R_i之间的首个不匹配点。

图 12

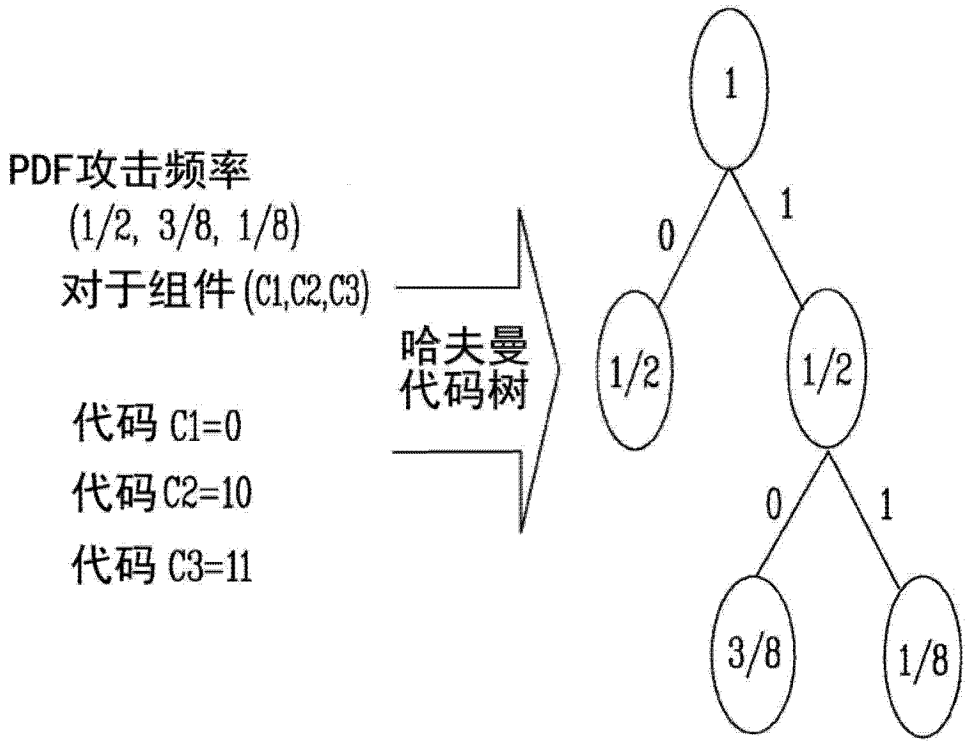


图 13

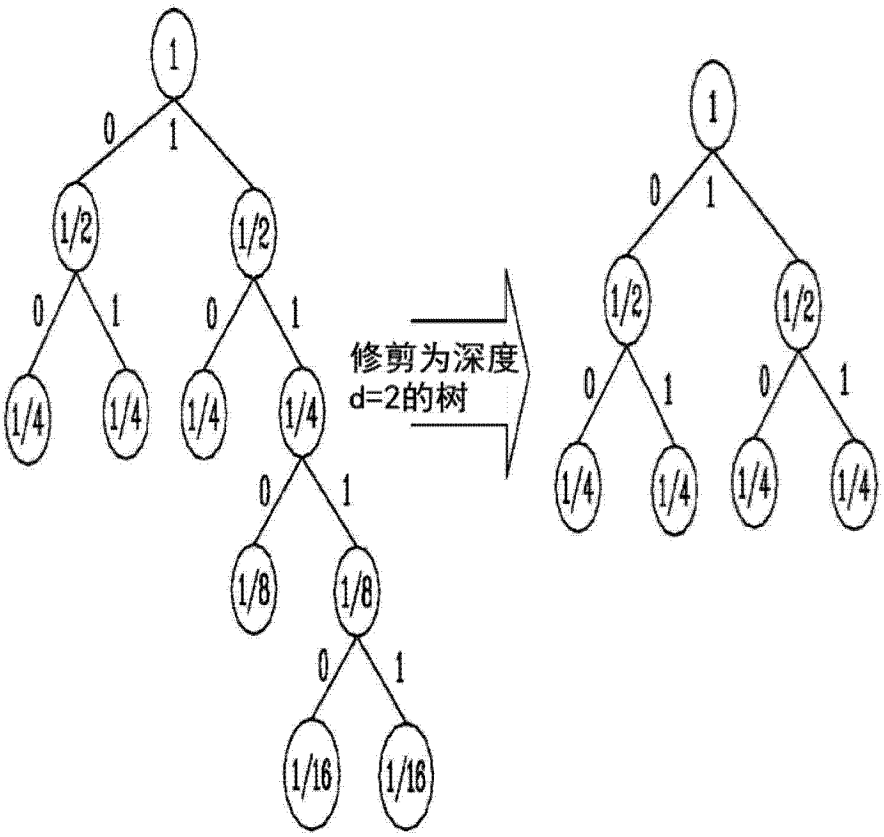

1
0
1
1/2
1/2
0
1
0
1
1/4
1/4
1/4
1/4
0
1
1/8
1/8
0
1
1/16
1/16
修剪为深度
d=2的树
1
0
1
1/2
1/2
0
1
0
1
1/4
1/4
1/4
1/4

图 14

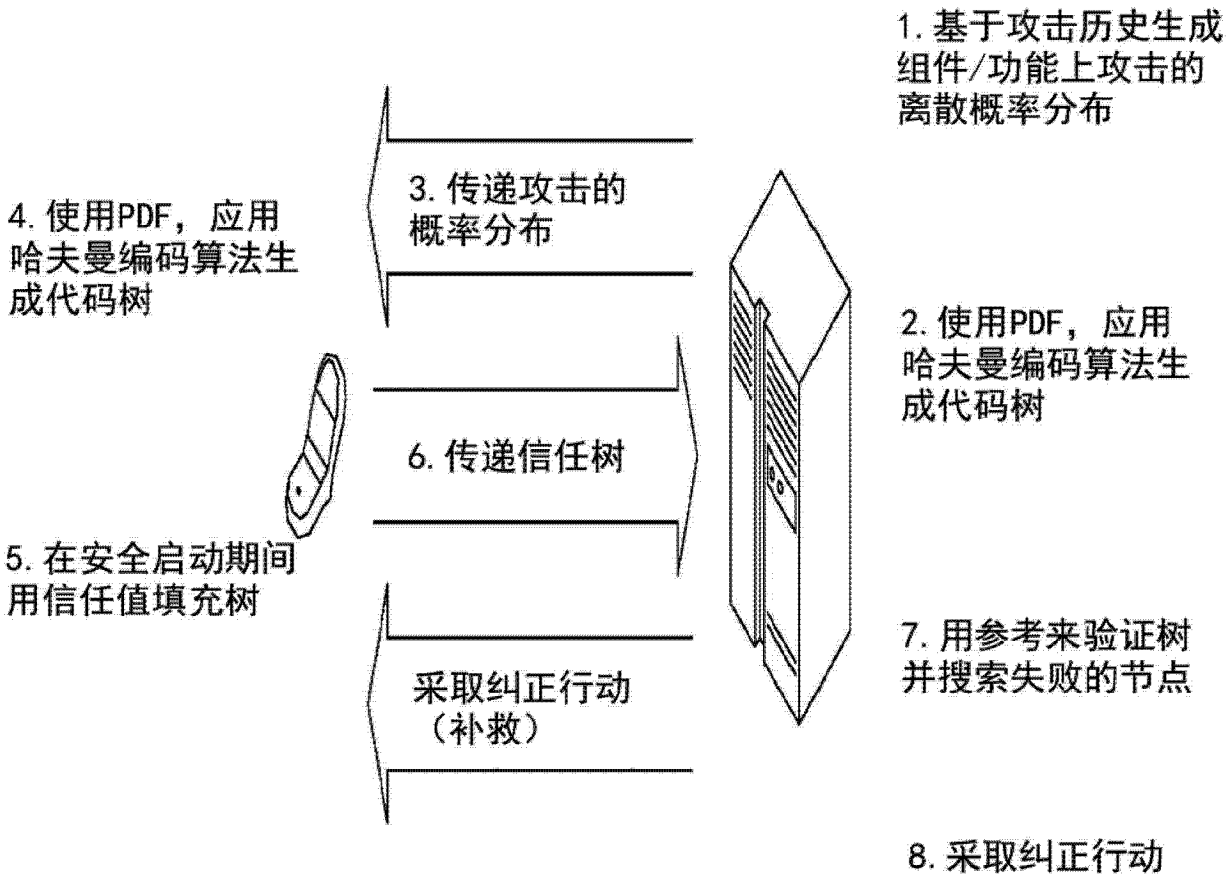

1. 基于攻击历史生成组件/功能上攻击的离散概率分布
3. 传递攻击的概率分布
4. 使用PDF，应用哈夫曼编码算法生成代码树
2. 使用PDF，应用哈夫曼编码算法生成代码树
6. 传递信任树
5. 在安全启动期间用信任值填充树
7. 用参考来验证树并搜索失败的节点
采取纠正行动（补救）
8. 采取纠正行动

图 15

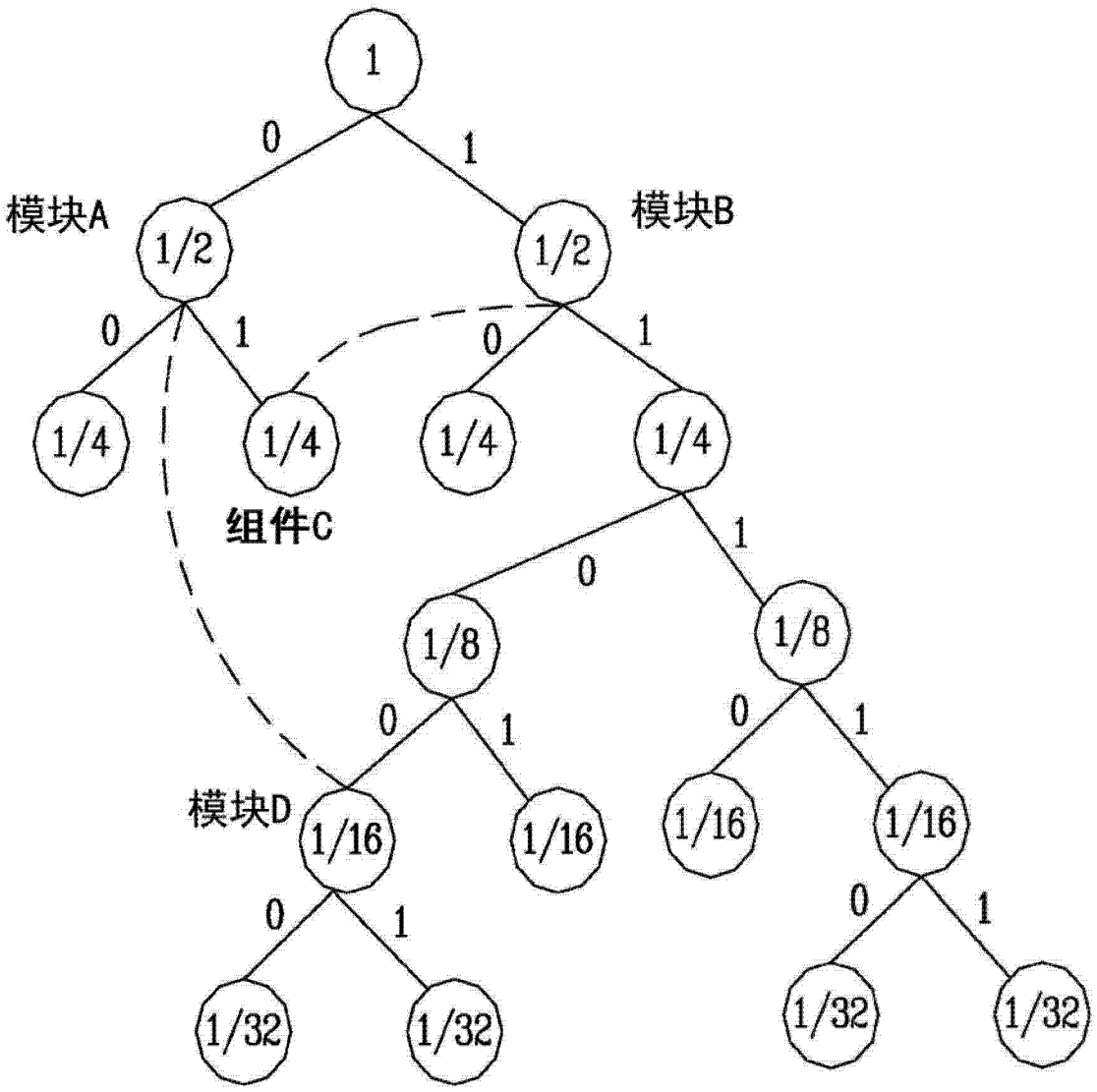

1
0
1
模块A
模块B
1/2
1/2
0
1
0
1
1/4
1/4
1/4
1/4
组件C
0
1
1/8
1/8
0
1
0
1
模块D
1/16
1/16
1/16
1/16
0
1
0
1
1/32
1/32
1/32
1/32

图 16

算法2：用度量填充二叉树

```
1. Populate_metric (Node)
2. {
3.       If (Node is leaf)
4.            Return H(component);
5.
6.       left_metric = Populate_metric(Node->Left Child);
7.       right_metric = Populate_metric(Node->Right Child);
8.
9.       Node.metric = F(left_metric, right_metric);
10.
11.      Return Node_metric;
12. }
```

图 17

算法3：使用TPM命令用度量填充树

```
1. Reg_Number = 0
2. Populate_metric (Node)
3. {
4.     If (Node is leaf)
5.     {
6.       Node.metric = H(component);
7.       PCR_Register Reg_Number = Node.metric;
8.       Reg_Number++;
9.       Return;
10.    }
11.
12.    Populate_metric(Node->Left Child);
13.    Populate_metric(Node->Right Child);
14.    Reg_Number--;
15.
16.    PCR_Register Reg_Number-1  =PCR_Extend(PCR_Register Reg_Number, PCR_Register Reg_Number-1 );
17.    Node.metric = PCR_Register Reg_Number-1 ;
18.
19.    Return;
20.}
```

图 18

算法4：用度量填充n元树

```
1. Populate_metric (Node)
2. {
3.   If (Node is leaf)
4.       Return H(component);
5.   Metric_0 = Populate_metric(Node->Child_0);
6.   for ( i=1; i<n; i++ )
7.       {
8.          Metric_i = Populate_metric(Node->Child_i);
9.          Node.metric = F(Metric_i, Metric_(i-1));
10.      {
11.
12.  Return Node_metric;
13.}
```

图 19

算法5：用替代孩子链路填充二叉树

```
1. Populate_metric (Node)
2. {
3.    If (Node is leaf)
4.        Return H(component);
5.
6.   left_metric = Populate_metric(Node->Left Child);
7.   right_metric = Populate_metric(Node->Right Child);
8.
9.   Node.metric = F(left_metric, right_metric);
10.
11.  If (surrogate_link_present)
12.     Surrogate_metric = Populate_metric(Node->Surrogate Child);
13.
14.  Node.metric = F(Node.metric, Surrogate_metric);
15.
16.  Return Node_metric;
17. }
```

图 20

算法6：找到完整性检查失败的节点和叶的比较和修剪算法

1. 在根开始比较。如果根度量匹配，那么所有节点和叶是完整的，没有被修改。退出算法。
2. 如果度量不匹配，那么继续下面的步骤。
3. 使j=1；
4. 比较深度为j的所有节点。
5. 如果深度为j的特定节点n_{ij}处的度量匹配，那么对n_{ij}节点下的节点和子树进行修剪。如果深度为j的特定节点n_{ij}处的度量不匹配，那么通过设置j=j+1比较深度为j的节点的孩子，向前递归地执行步骤3。
6. 执行这些步骤直到达到树的完整深度。
7. 剩余部分的树仅包含具有不匹配节点或叶测量（其被认为指示完整性失败的节点或叶）的节点和叶。

图 21

算法1 TPM_Reduced_Tree_Verify_Load

Require: $B_i,...,B_\ell, V^* \in \mathcal{V}, ((n),n, R(n))$

Ensure: $B_i \leftarrow r_i,...,B_\ell \leftarrow r_\ell, V^* \leftarrow n$

▷ Initialize buffer with reduced tree and node to verify.

1. for $k=\ell,...,1$ do
2. $V^* \rightarrow$ TSS
3. $V^* \leftarrow (B_k|(n)_k|V^*)$
4. end for
5. if $V \equiv V^*$ then
6. return V
7. else
8. return "verification error"
9. end if

图 22

算法2 TPM_Reduced_Tree_Verify

Require: $B \in \mathcal{V}$, ((n),n, R(n))

Ensure: $B \leftarrow n$ ▷ Initialize buffer with node to verify.

1. for $k = \ell,..., 1$ do
2. $B \rightarrow$ TSS
3. $B \leftarrow (r_k|(n)_k|B)$
4. end for
5. if $V \equiv B$ then
6. return V
7. else
8. return "verification error"
9. end if

图 23

算法3 TPM_Tree_Node_Verify

Require: $B, C, D \in \mathcal{V}$, ((n), R(n), T(n))

Ensure: $C \leftarrow V$ ▷ Initialise comparison register with root.

1. for $k = 1 ,..., \ell$ do
2. $B, D \leftarrow t_k$ ▷ Load trace child into buffers
3. $B \leftarrow (r_k|(n)_k|B)$
4. if $C \equiv B$ then

▷ Compare result with current parent, and if OK,

5. $C \leftarrow D$

▷ make the trace element just verified the new parent.

6. else
7. return "verification error at level" || k, B
8. end if
9. end for
10. return "OK"

图 24A

算法4 TPM_Reduced_Tree_Update

Require: (n), n'

Ensure: $B_1 = r_1, \ldots, B_\ell = r_\ell$

1. $V \leftarrow n'$
2. for $k = \ell, \ldots, 1$ do
3. $V \rightarrow TSS$
4. $V \leftarrow (B_k | \langle n \rangle_k | V)$
5. end for
6. return V

图 24B

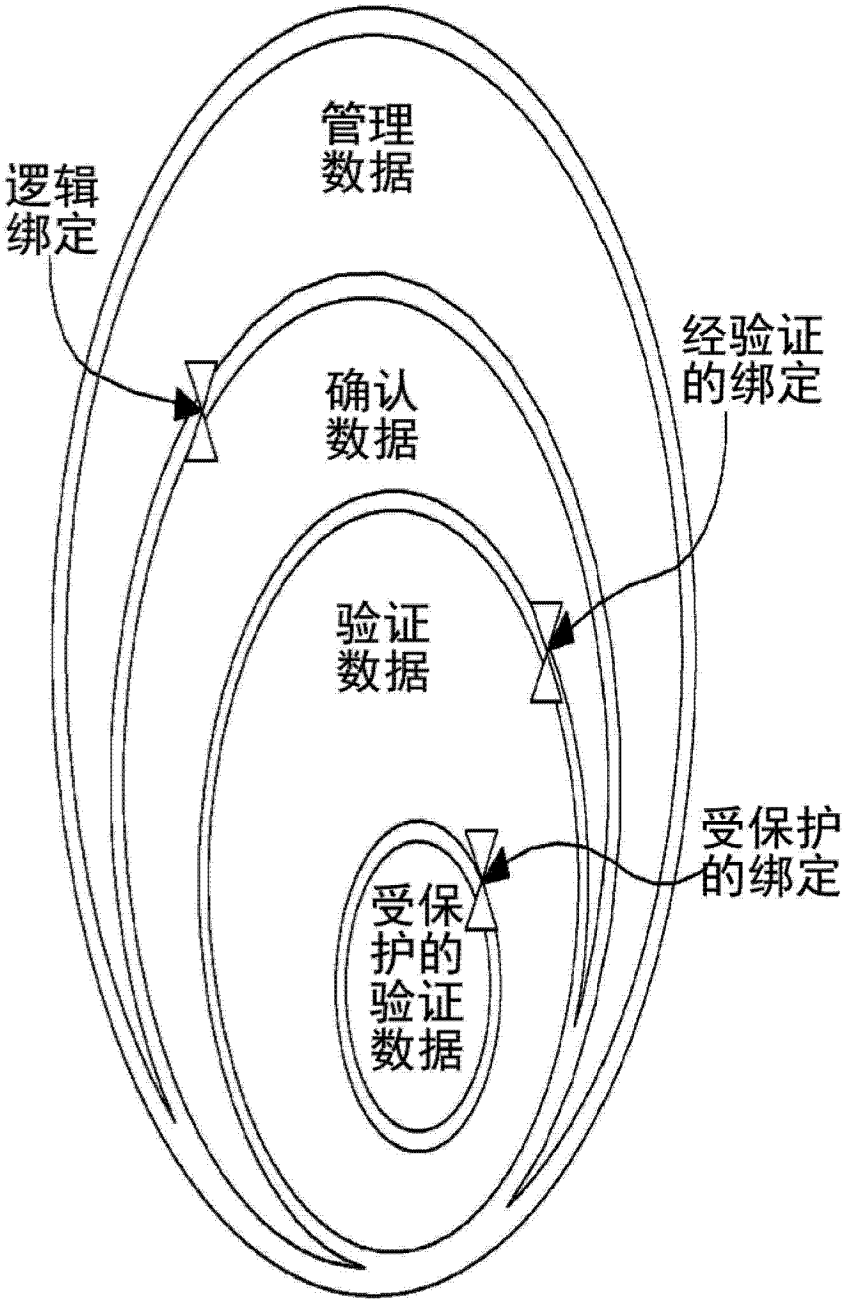


图 25

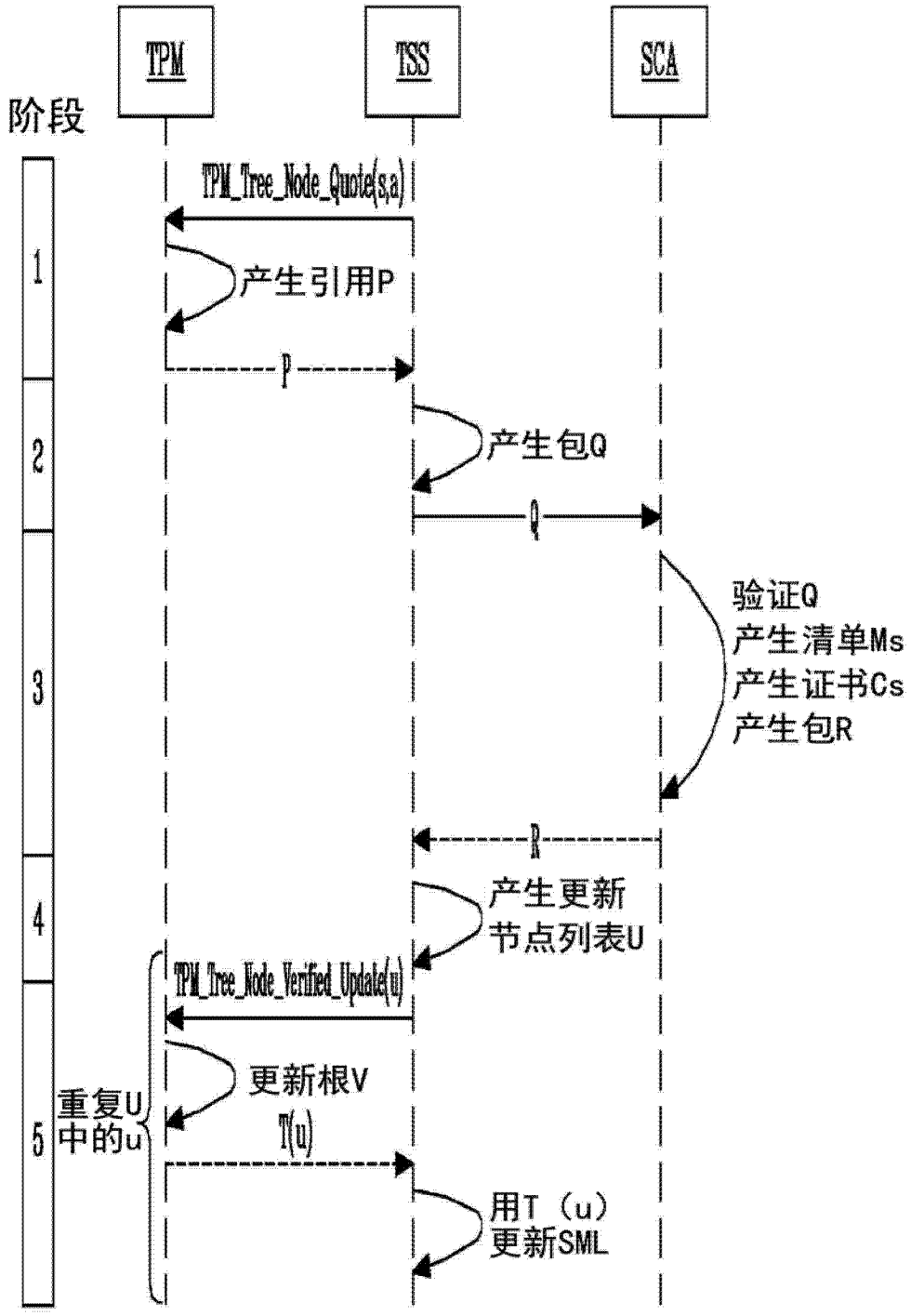

TPM
TSS
SCA
阶段
TPM_Tree_Node_Quote(s,a)
1
产生引用P
P
2
产生包Q
Q
3
验证Q
产生清单Ms
产生证书Cs
产生包R
R
4
产生更新
节点列表U
TPM_Tree_Node_Verified_Update(u)
5
重复U
中的u
更新根V
T(u)
用T（u）
更新SML

图 26

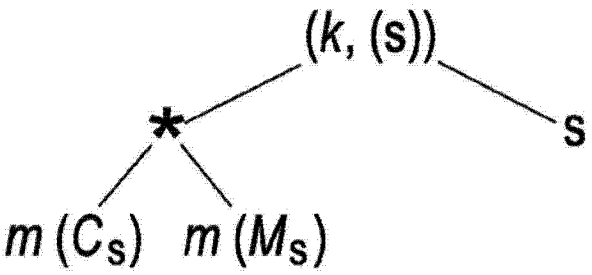


图 27

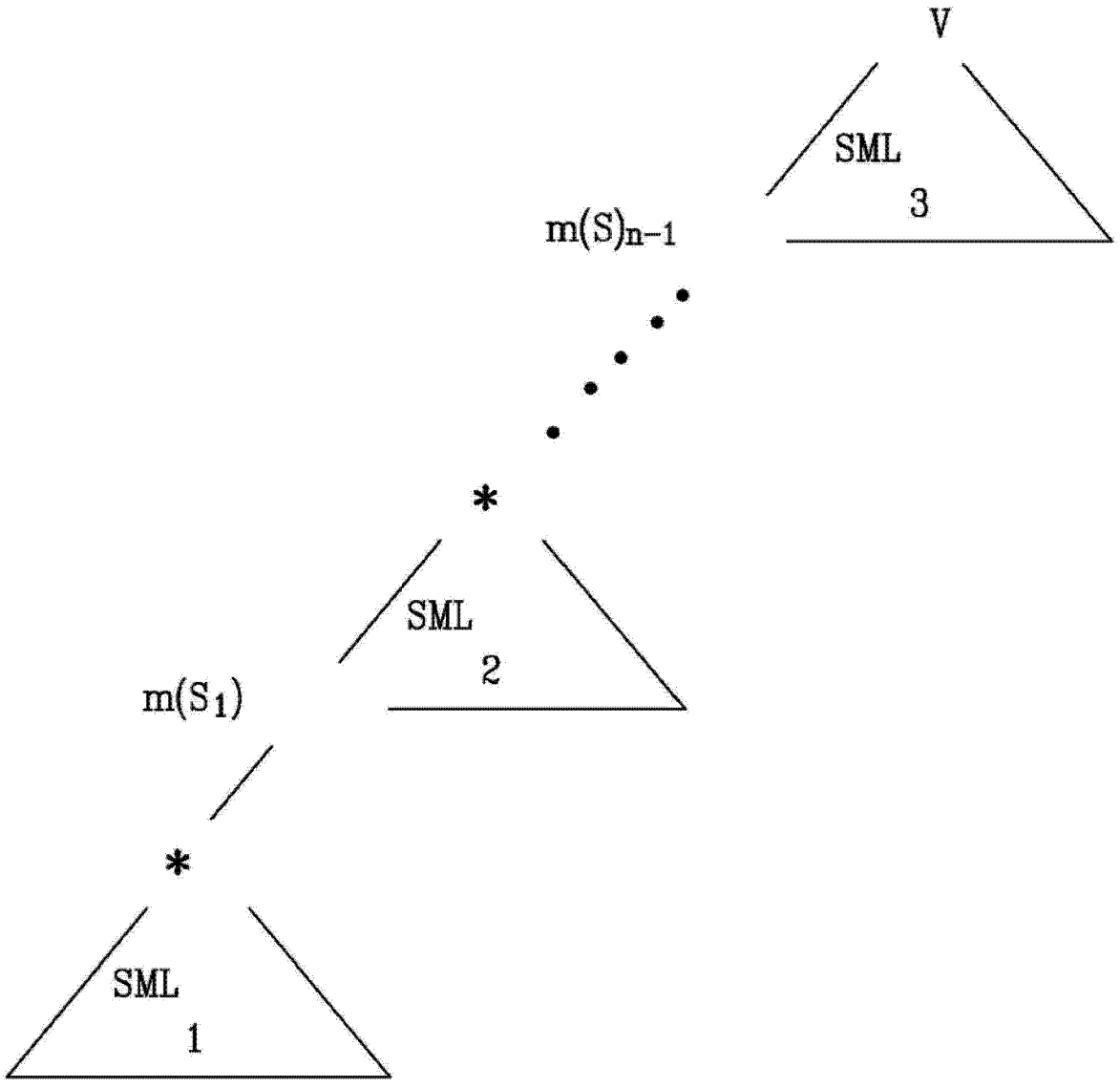


图 28

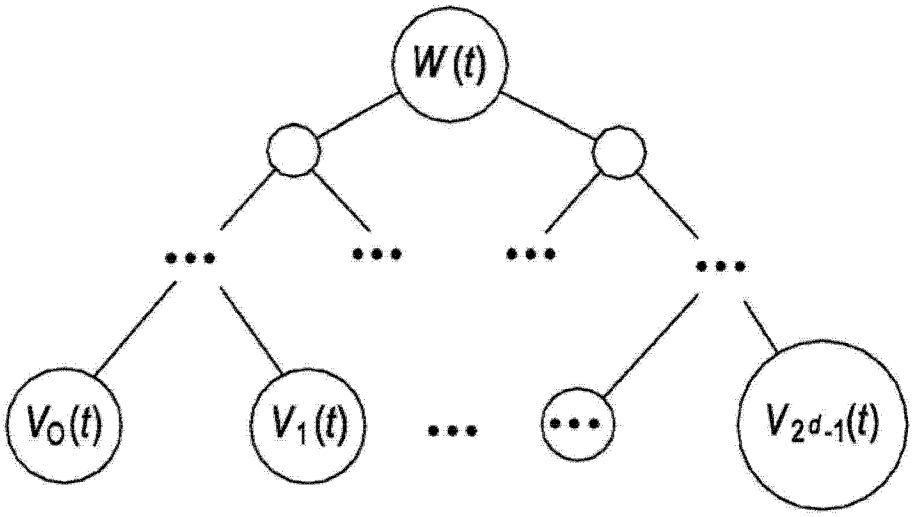


图 29

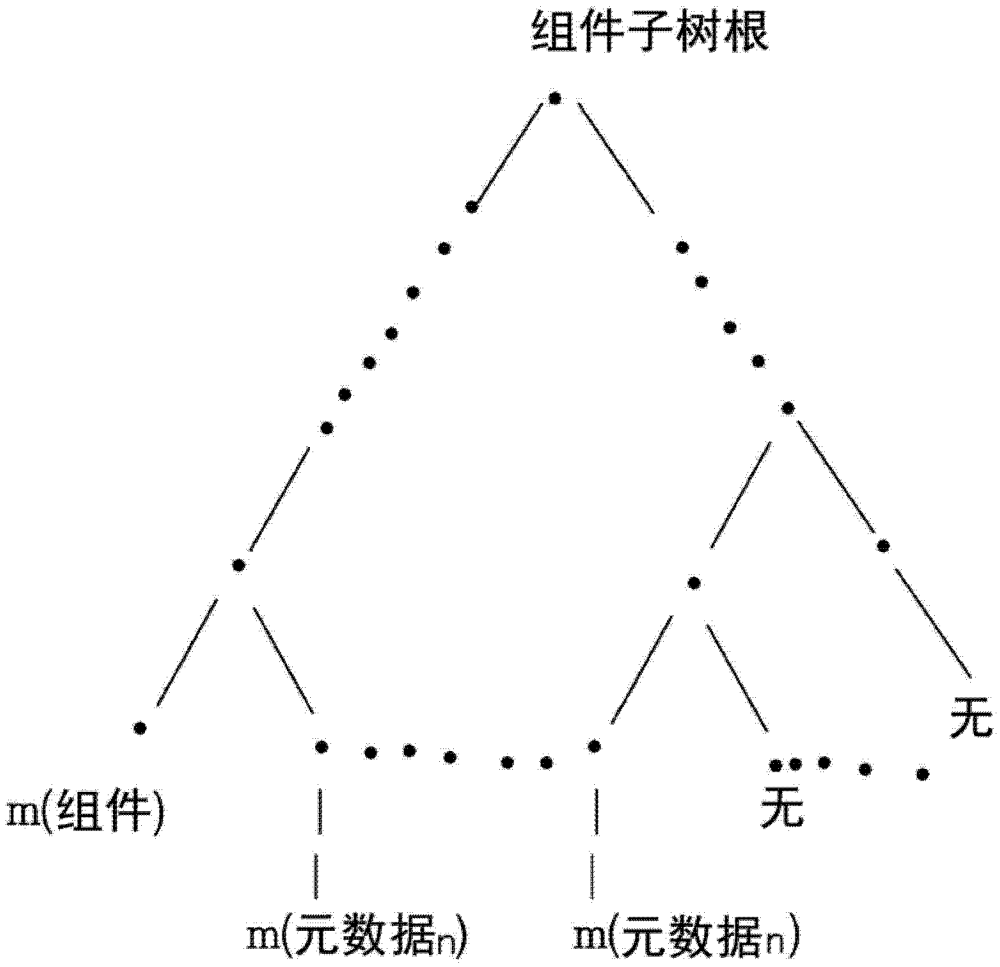


图 30

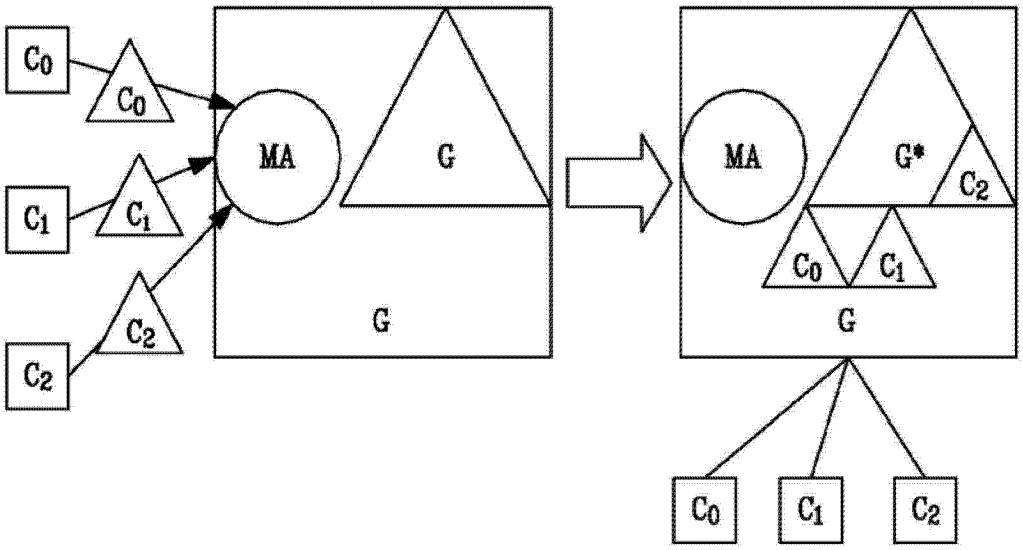

C0
C0
C1
C1
C2
C2
MA
G
G
MA
G*
C2
C0
C1
G
C0
C1
C2

图 31

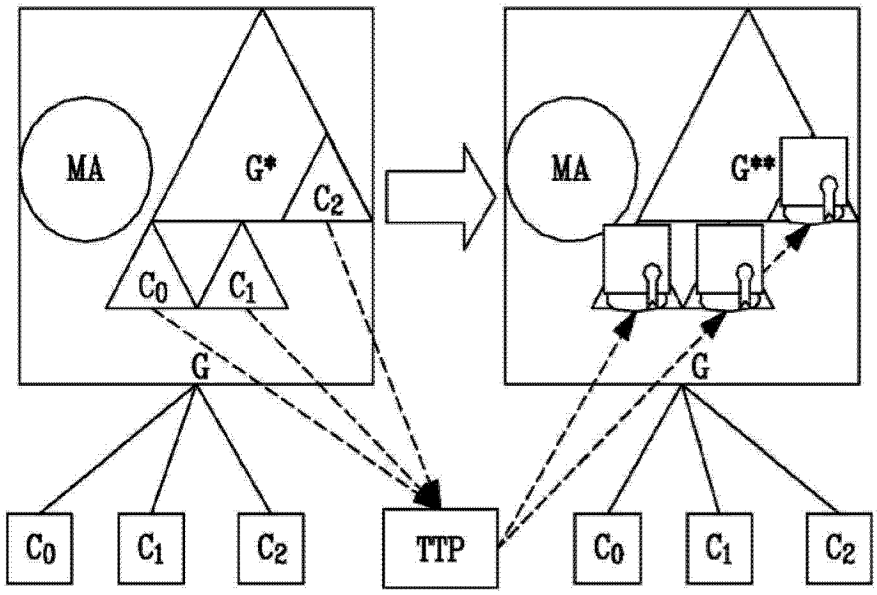

MA
G*
C2
C0
C1
G
MA
G**
G
C0
C1
C2
TTP
C0
C1
C2

图 32

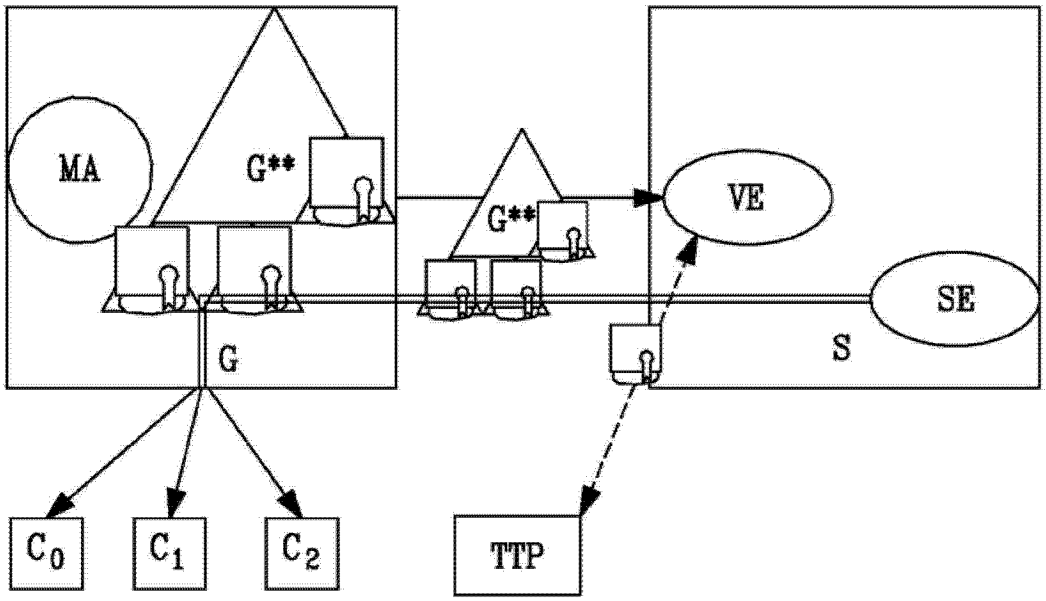


图 33

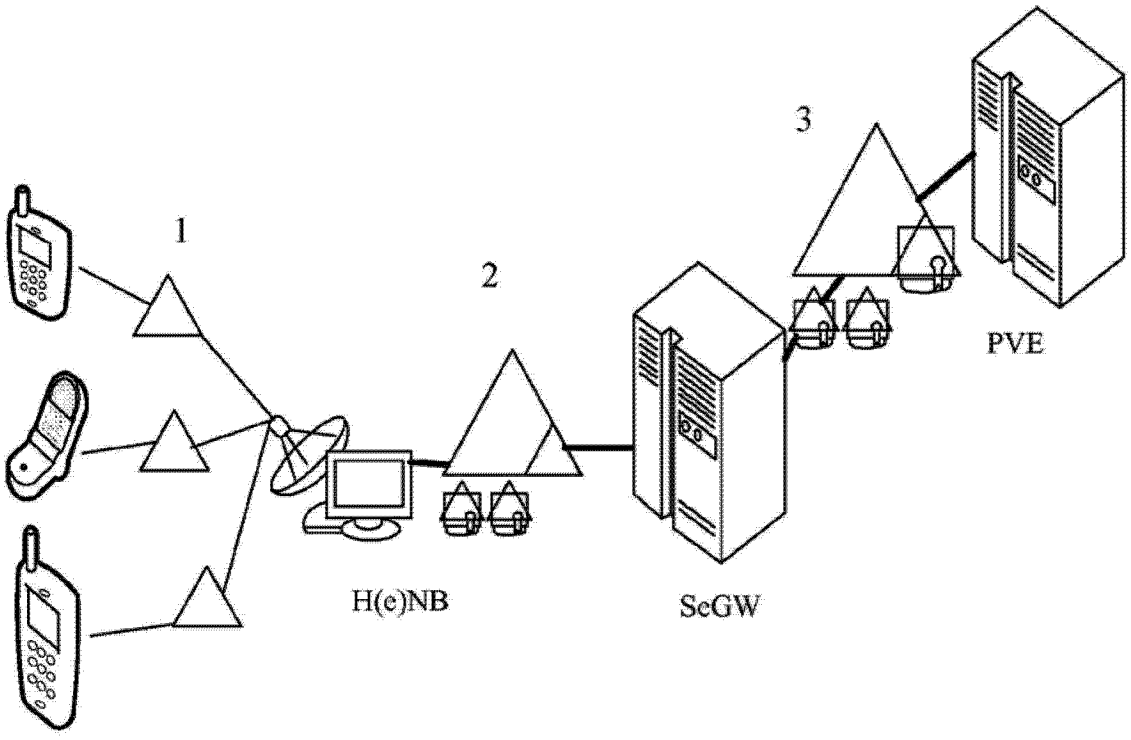


图 34

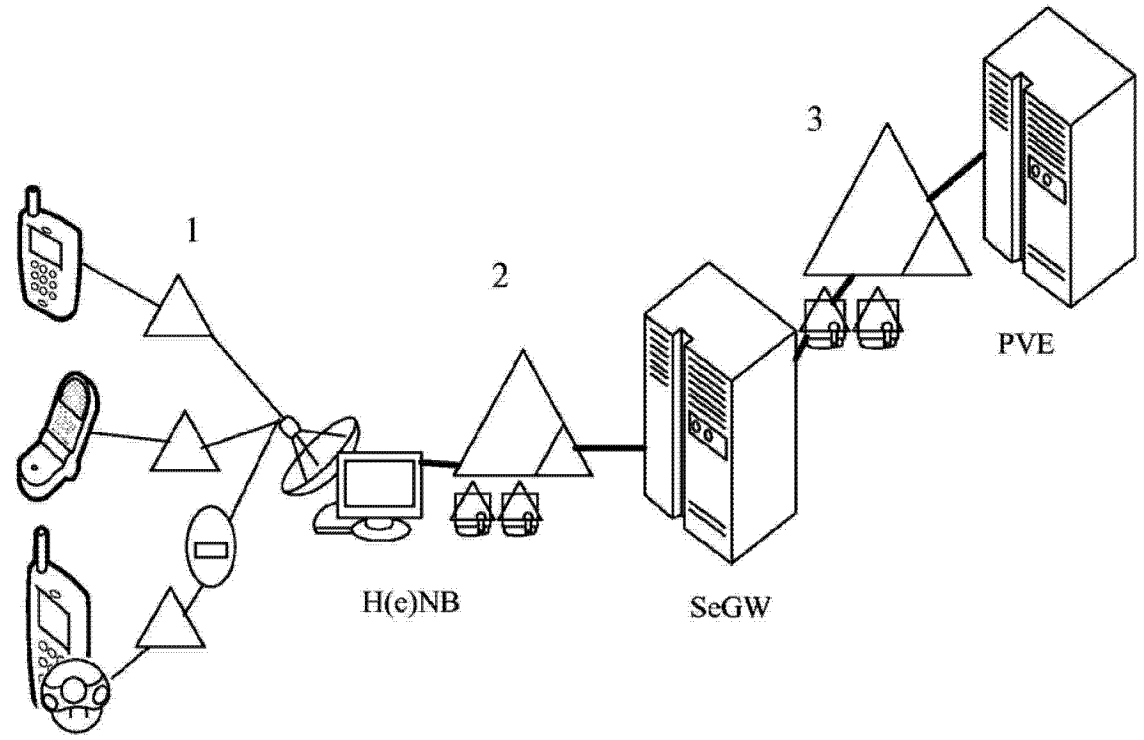


图 35

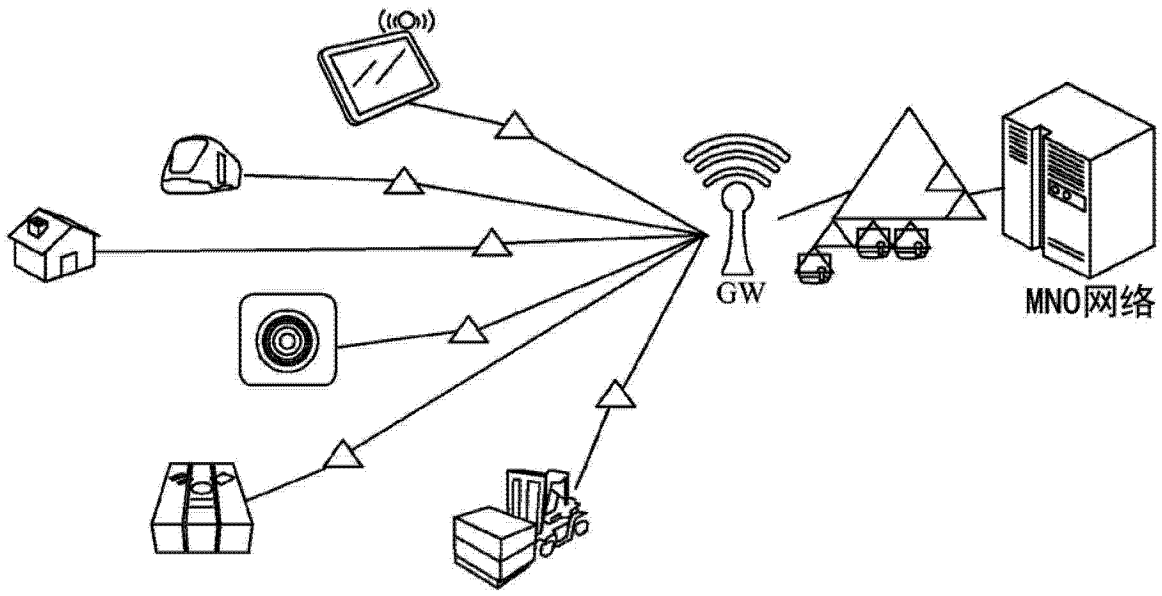


图 36

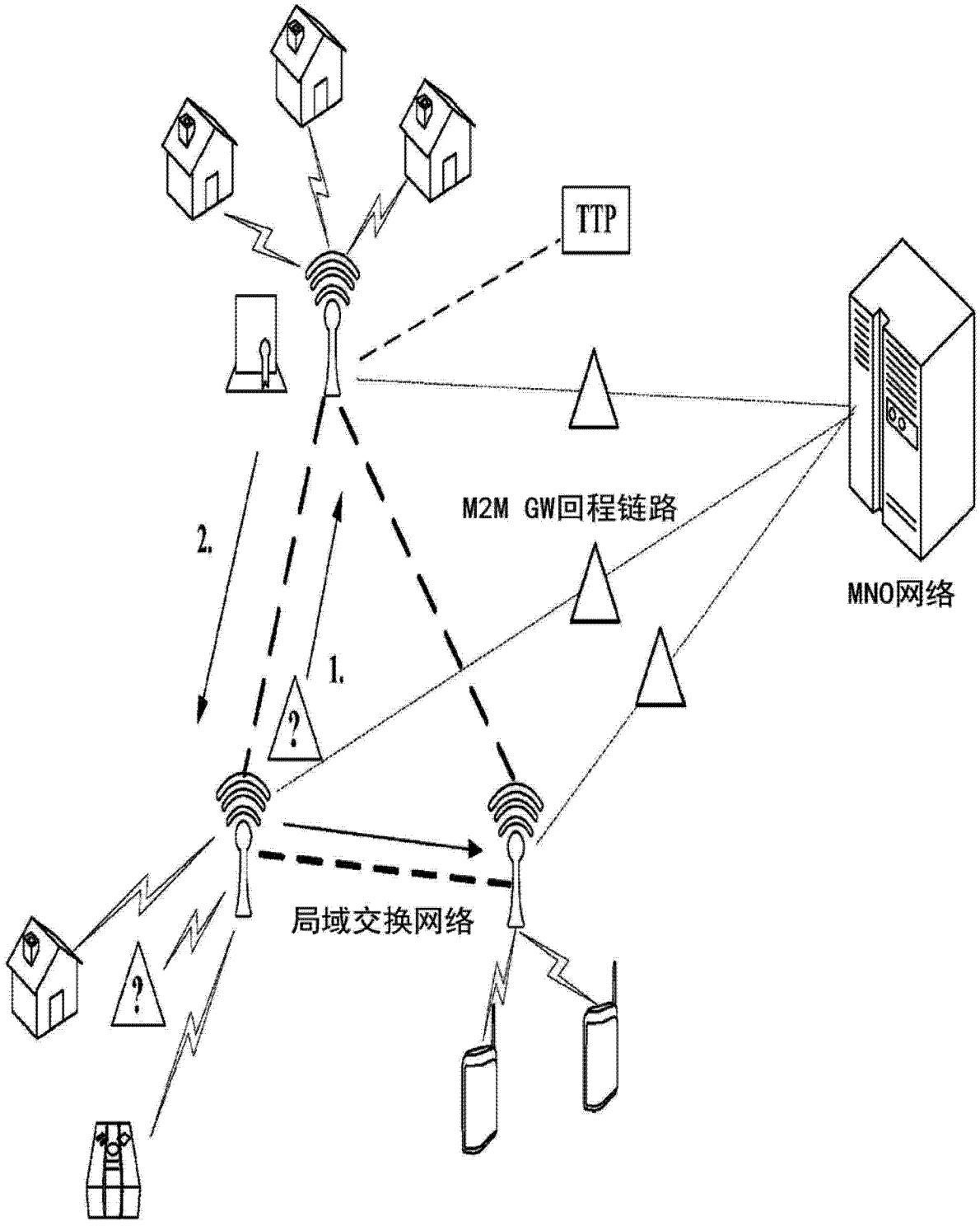


图 37

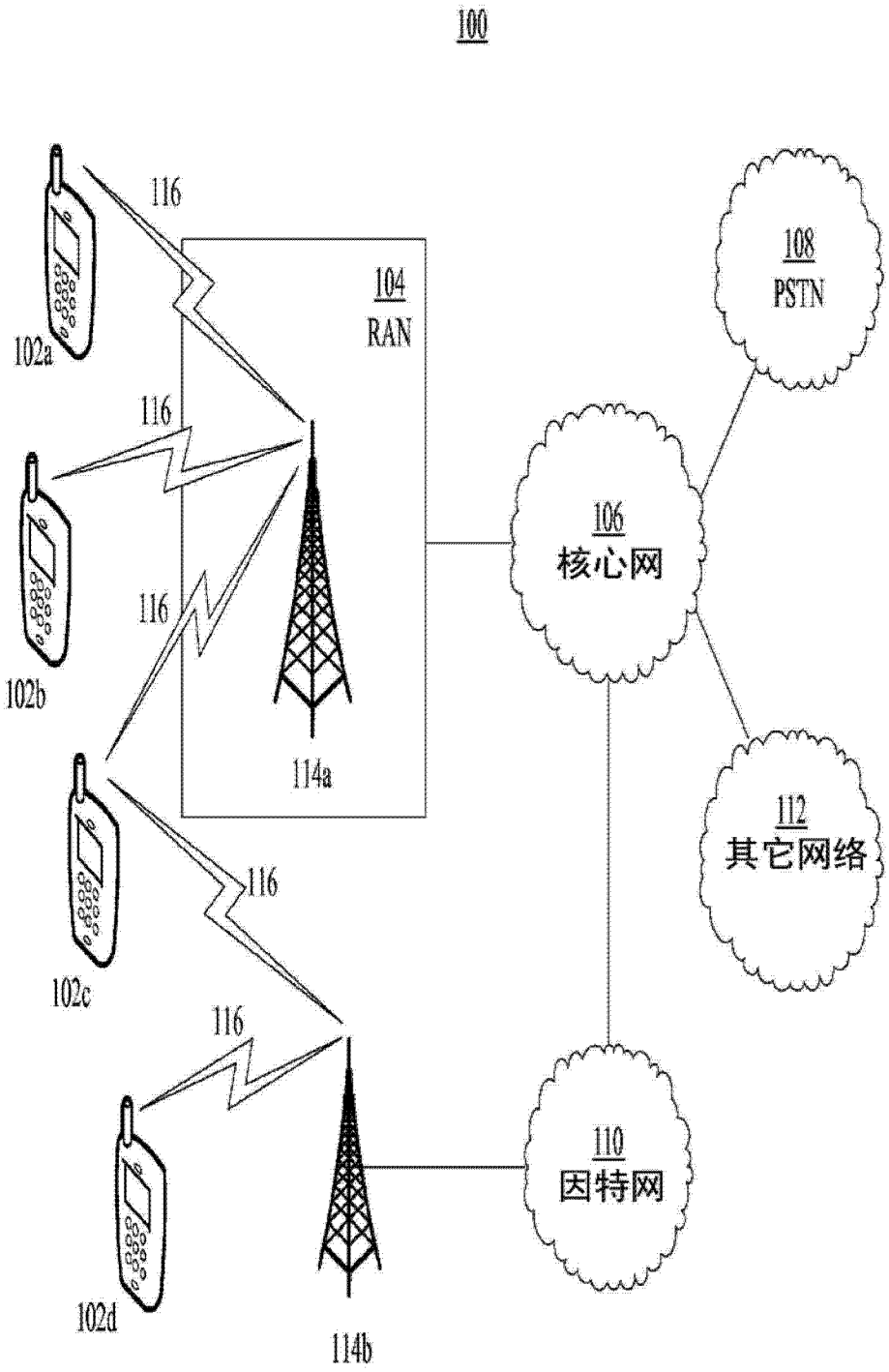

100
116
102a
104
RAN
108
PSTN
116
102b
106
核心网
116
114a
112
其它网络
116
102c
116
110
因特网
102d
114b

图 38

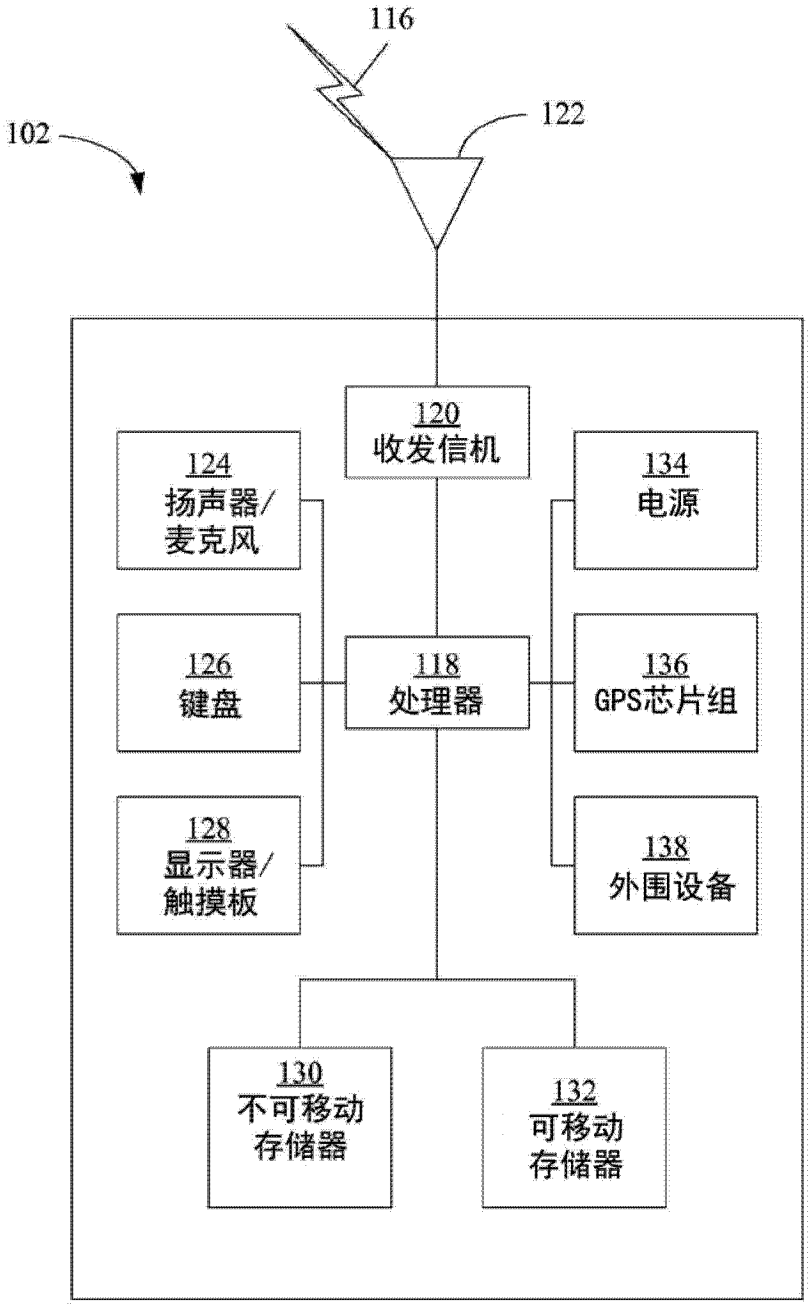

116
122
102
120
收发信机
124
扬声器/
麦克风
134
电源
126
键盘
118
处理器
136
GPS芯片组
128
显示器/
触摸板
138
外围设备
130
不可移动
存储器
132
可移动
存储器

图 39

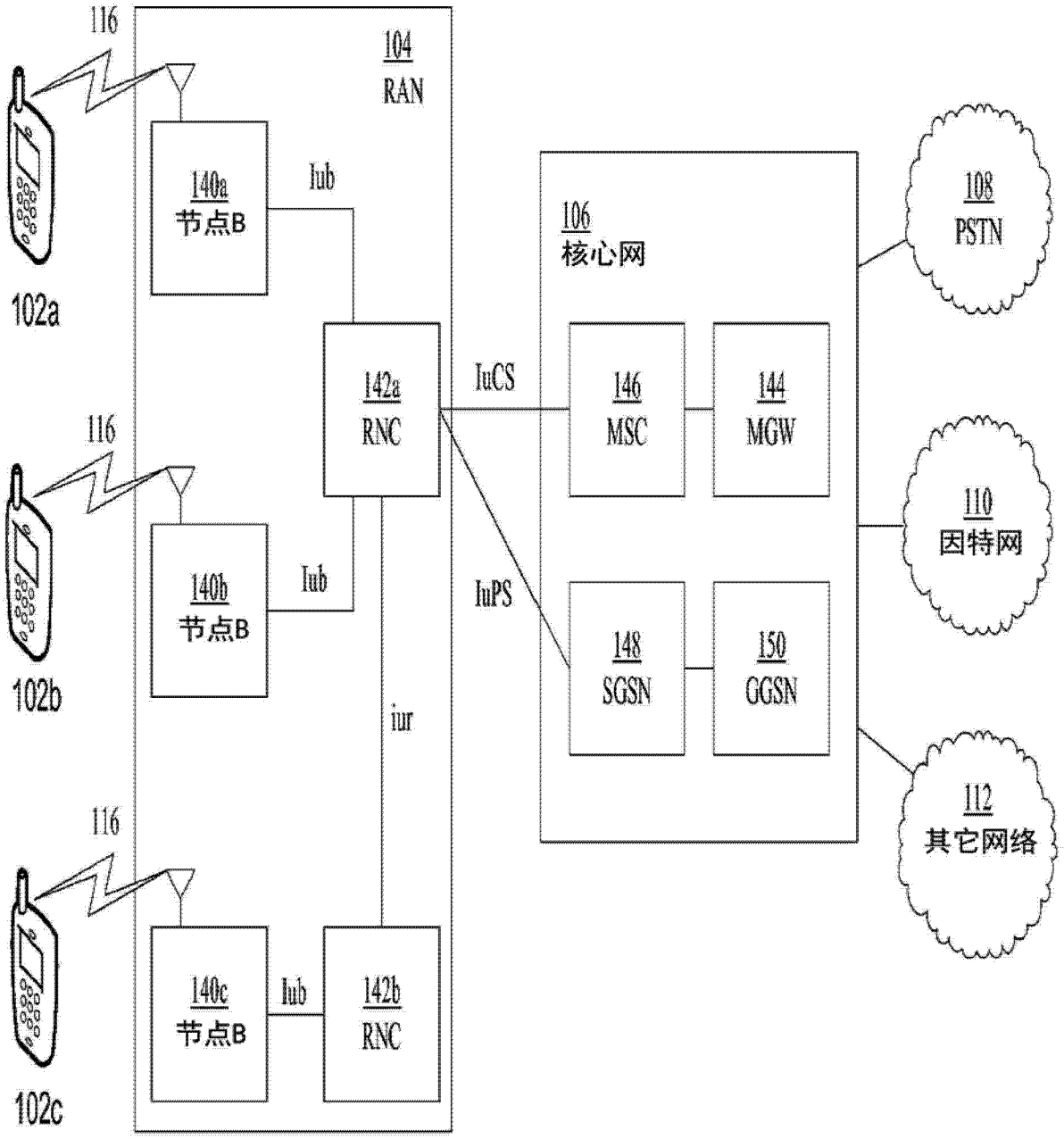

116
104
RAN
140a
节点B
Iub
102a
106
核心网
108
PSTN
142a
RNC
IuCS
146
MSC
144
MGW
116
140b
节点B
Iub
IuPS
110
因特网
102b
iur
148
SGSN
150
GGSN
116
112
其它网络
140c
节点B
Iub
142b
RNC
102c

图 40