(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 102073819 B
(45)授权公告日 2013.05.29

(21)申请号 201110009994.X

(22)申请日 2006.10.18

(30)优先权数据
60/728089 2005.10.18 US
60/772024 2006.02.09 US
60/744574 2006.04.10 US
60/791179 2006.04.10 US
60/746712 2006.05.08 US
60/798925 2006.05.08 US
60/835061 2006.08.01 US

(62)分案原申请数据
200680047769.2 2006.10.18

(73)专利权人 英特托拉斯技术公司
地址 美国加利福尼亚州

(72)发明人 G·博康-吉博 J·G·博夫
M·G·梅嫩特 W·B·布拉德利

(74)专利代理机构 中国专利代理(香港)有限公司 72001
代理人 王忠忠

(51)Int.Cl.
G06F 21/10(2013.01)
H04L 9/08(2006.01)

(56)对比文件
US 2005027871 A1,2005.02.03, 说明书【0192】段、【0203】段、【0299】-【0303】段、【0344】-【0346】段,【0351】-【0352】段,附图10-15、21.
CN 1531253 A,2004.09.22, 全文.
CN 1592876 A,2005.03.09, 全文.

审查员 刘申

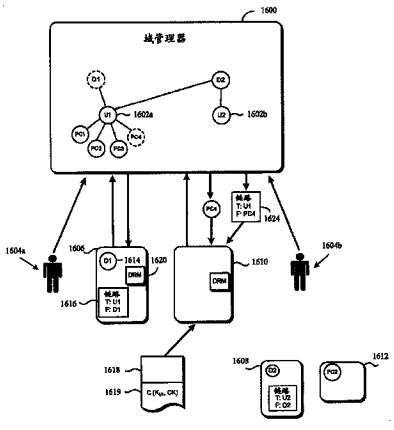
权利要求书1页 说明书231页 附图37页

(54)发明名称

数字权利管理的方法

(57)摘要

描述了用于执行数字权利管理的系统和方法在一个实施例中,提供了一种数字权利管理引擎,用于评估与受保护的内容相关联的许可以确定对所述内容所请求的访问或其它使用是否被授权。在一些实施例中,许可包含可由数字权利管理引擎执行的控制程序。



CN 102073819 B

1. 一种用于授权对一篇电子内容执行给定动作的方法，所述方法包括：

使用在第一数字权利管理引擎上运行的虚拟机来执行第一控制程序，所述第一控制程序可操作来确定是否能够对该篇电子内容执行给定动作，其中所述第一控制程序可操作来评估要使对给定动作的执行被授权而必须满足的第一组一个或多个条件，并且其中所述第一组一个或多个条件中的至少一个包括一个或多个链路对象可以用于数字权利管理引擎的要求，所述链路对象在逻辑上把表示第一实体的第一节点链接到表示第二实体的第二节点；

获取所述一个或多个链路对象，每个所述链路对象表示两个实体之间的关系，并且所述链路对象中的至少一个包括第二控制程序，其中所述第二控制程序可操作来评估要使至少一个链路对象被认为有效而必须满足的第二组一个或多个条件；并且

使用所述数字权利管理引擎来执行所述第二控制程序，

其中所述第一组条件或第二组条件中的至少一组包括在存储器中所存储的计数器不应超过预定义值的要求。

2. 如权利要求 1 所述的方法，其中所述第一组一个或多个条件包括基于时间的条件。

3. 如权利要求 1 所述的方法，其中所述第二组一个或多个条件包括基于时间的条件。

数字权利管理的方法

[0001] 本申请是申请人英特托拉斯技术公司于 2006 年 10 月 18 日提交的同名中国专利申请 200680047769.2 的分案申请。

[0002] 本申请要求了于 2005 年 10 月 18 日提交的美国临时申请号 60/728,089、于 2006 年 2 月 9 日提交的美国临时申请号 60/772,024、于 2006 年 4 月 10 日提交的美国临时申请号 60/744,574、于 2006 年 4 月 10 日提交的美国临时申请号 60/791,179、于 2006 年 5 月 8 日提交的美国临时申请号 60/746,712、于 2006 年 5 月 8 日提交的美国临时申请号 60/798,925 和于 2006 年 8 月 1 日提交的美国临时申请号 60/835,061 的优先权益。这里将美国临时申请号 60/728,089、60/772,024、60/744,574、60/791,179、60/746,712、60/798,925 和 60/835,061 全部引用以用于任何目的。

[0003] 版权许可

[0004] 本专利文档的公开内容的部分包含受版权保护的材料。版权所有者不反对任何人对本专利文献或专利公开内容的影印复制，由于它已经出现在美国专利商标局的专利文档或记录中，但是在其它方面却无论如何都保留所有版权。

[0005] 背景技术或相关信息

[0006] 在现代计算系统中，人们常常希望限制对电子内容、服务和 / 或处理资源的访问和 / 或只允许某些实体执行某些动作。已经开发或提出各种技术来实现这种控制。这些技术常常被称为数字权利管理 (digital rightsmanagement DRM) 技术，这是因为一般地说，它们的目标在于在数字或其它电子内容、服务或资源方面管理各实体的权利。许多现有技术的问题在于它们过于复杂、受限制、相对不那么灵活，不能实现某些自然类型的关系和处理，和 / 或不能与其它 DRM 系统共同使用。

[0007] 这里描述了与改进的 DRM 技术相关的系统和方法，所述系统和方法可以用来改进这些问题中的一些或全部。应当理解，目前描述的所发明工作体的实施例可以依照多种方式来实现，包括作为程序、设备、系统、装置、方法、计算机可读介质和 / 或其组合来实现。

[0008] 用于控制对内容访问的现有系统包括用于结合对电子内容访问的授权来访问许可的组件。然而，这种组件通常对与许可相关联的权利管理信息的链或图、链路或节点执行不太灵活的评估。它们常常不能适于完全不同的授权方案和 / 或常常不能与某些 DRM 系统一起操作来授权对内容的访问。本发明的实施例通过与许可相关联地存储、利用和 / 或执行附加例程或控制程序来克服这种缺点，其可以提供动态授权特征，启用分布式授权资源和 / 或另外使访问功能流水线化。

[0009] 此外，许多现有系统只针对其中维持简单的与授权 / 状态相关的数据的情况。这些系统没能解决其中访问授权可能要求关于多个数据层的相关性的情况，诸如根据先前导出的与其它节点相关联的数据来确定条件。本发明的实施例通过结合 DRM 控制程序实现状态数据库来提供状态存储特征以克服这些缺点，所述状态存储特征是安全的，提供从调用到调用的持久状态信息，或另外启用状态读取和写入功能，所述状态读取和写入功能改进了控制程序的执行和 / 或执行更高效的访问授权。

[0010] 另外的现有系统可以实现 DRM 许可或结构，其包括涉及使用公钥来保护许可组件

的组件。然而，与这些系统相关的缺点包括这样的可能性：黑客可以伪造访问或实施许可或另外利用存在于 DRM 许可结构中相关联的相互关系所要求的数字签名。本发明的一个或多个实施例通过实施许可对象的数字和 / 或联锁签名来克服这种缺点，包括使用特定的受保护的密钥。这些实施例的优点包括通过公钥以及从许可元项的相互关系导出的关联特征来防止未授权的访问。

[0011] 其它现有系统可以包括用于在第一实体和第二实体之间诸如在两个权利管理实体之间进行邻近度 (proximity) 确定的组件。这种系统可以强制执行规则，该规则表明：例如通过执行麻烦的邻近度检查过程，而使受保护的内容无法被拷贝到某一环境之外。然而，这些系统的缺点在于它们还不能在不过于妨碍执行邻近度检查本身的情况下向受保护的内容提供安全性保护。本发明的实施例通过提供经由与传输随机数和 / 或秘密种子相关的特征来实现可靠的简洁的邻近度检测协议来克服这些及其它缺点。一个或多个实施例的相关联优点包括攻击者在密码上不可能确定正确的响应，即便已经拦截了请求也是如此。

[0012] 总之，需要一种可以不依靠过于复杂的、受限制的和 / 或相对不灵活的技术来充分授权对电子内容的访问的系统，能够启用某些自然类型的关系和过程的系统，和 / 或可与其它 DRM 系统一起操作的系统。

发明内容

[0013] 按照本发明的系统、方法和制造产品针对电子内容的授权访问以及与所述电子内容相关联的数据处理。

[0014] 在一个示例性实施例中，提供了一种用于授权访问在存储器中所存储的电子内容的方法。此示例性实施例的方法可以包括接收用于访问电子内容的请求，获取 (retrieve) 与所述电子内容相关联的许可，并且使用数字权利管理引擎来执行第一控制程序以便确定所述请求是否可以被准许。其它示例性实施例可以包括评估在存储器中所存储的一个或多个链路对象并且执行第二控制程序以便确定所述链路是否有效和 / 或一个或多个条件是否被满足。

[0015] 应当理解，以上一般描述以及以下详细描述只是示例性的和解释性的，而并非是限制所描述的本发明。除这里所阐明的那些特征和 / 或变化之外，还可以提供进一步的特征和 / 或变化。例如，本发明可以针对所公开特征的各种组合和子组合和 / 或下面在详细描述中所公开的几个进一步特征的组合和子组合。

附图说明

[0016] 结合附图参照以下详细描述可以容易地理解本发明的发明主体，其中：

[0017] 图 1 示出了用于管理电子内容的使用的说明性系统。

[0018] 图 2 示出了可以用来实施所发明工作主体的实施例的系统的更详细例子。

[0019] 图 3 示出了说明性的数字权利管理 (DRM) 引擎在使用 DRM 的网络中可以怎样起作用。

[0020] 图 4 示出了用于对 DRM 系统中的关系进行建模的节点和链路的集合。

[0021] 图 5 是用于图示 DRM 引擎的实施例可以怎样确定所请求的动作是否被授权的流程图。

[0022] 图 6 依照所发明的工作主体的一个实施例示出了 DRM 许可的例子。

[0023] 图 7A 和 7B 图示了在一个实施例中代理的使用。

[0024] 图 8 示出了 DRM 许可的例子。

[0025] 图 9 是 DRM 引擎可以怎样确定所请求的动作是否被授权的更详细的例子。

[0026] 图 10 是 DRM 引擎怎样在一个实施例对象中执行控制程序的更详细的例子。

[0027] 图 11 示出了在设备上运行的 DRM 引擎的说明性实施例。

[0028] 图 12 是用于图示在一个实施例中执行控制程序中所涉及的步骤的流程图。

[0029] 图 13 示出了在一个实施例中构成内容消费客户端应用的元项。

[0030] 图 14 示出了在一个实施例中构成内容封装 (packaging) 应用的元项。

[0031] 图 15 示出了依照一个实施例的密钥推导机制。

[0032] 图 16 示出了 DRM 系统的例子。

[0033] 图 17 示出了用于规定临时登录的 DRM 系统的例子。

[0034] 图 18 示出了用于管理企业文档的说明性系统的高级体系结构。

[0035] 图 19 示出了诸如在图 18 中所示出的系统可以怎样用来管理对文档的访问或其它使用的例子。

[0036] 图 20 示出了诸如在图 18 中所示出的系统可以怎样用来管理对文档的访问或其它使用的附加例子。

[0037] 图 21 示出了在图 20 中所示出的例子的附加特征。

[0038] 图 22 示出了用于管理企业内的电子内容的另一说明性系统。

[0039] 图 23 图示了可以怎样应用这里所描述的系统和方法来管理保健记录。

[0040] 图 24 是在电子预订服务的情境中可以怎样使用这里所给出的系统和方法的图解。

[0041] 图 25 是在家庭网络域的情境中可以怎样使用这里所描述的系统和方法的图解。

[0042] 图 26 图示了在一个示例性实施例中在主机应用和 DRM 客户端引擎之间发生的交互。

[0043] 图 27 图示了在一个说明性实施例中在主机应用和封装引擎之间发生的交互。

[0044] 图 28A 是依照一个实施例的许可的更详细的图解。

[0045] 图 28B 图示了在一个示例性实施例中在链路和节点之间的关系。

[0046] 图 29 图示了虚拟机的说明性实现的运行环境。

[0047] 图 30 图示了依照一个实施例的扩展状态块数据结构。

[0048] 图 31A 示出了在一个实施例中的数据段的存储器映像。

[0049] 图 31B 示出了在一个实施例中的代码段的存储器映像的例子。

[0050] 图 31C 示出了在一个实施例中的输出入口存储器映像的例子。

[0051] 图 31D 示出了在一个实施例中的输出表入口的通用例子。

[0052] 图 31E 示出了用于示例性入口点 (entry point) 的输出表入口的例子。

[0053] 图 32 示出了许可转送协议的例子。

[0054] 图 33 示出了依照一个实施例的许可转送协议的另一例子。

[0055] 图 34 示出了用于在一个实施例中保护许可对象的完整性的机制。

[0056] 图 35 示出了用于在另一实施例中保护许可对象的完整性的机制。

[0057] 图 36 图示了依照一个实施例的邻近度检查协议。

[0058] 图 37 图示了依照一个实施例的邻近度检查协议的使用。

[0059] 图 38 图示了在一个实施例中在客户端和许可服务器之间的交互。

[0060] 图 39 是在一个实施例中在客户端和许可服务器之间交互的更详细图解。

[0061] 图 40 示出了具有多个角色的实体的例子。

[0062] 图 41 图示了依照一个实施例的引导 (bootstrap) 协议。

[0063] 图 42 示出了在一个实施例中在 c14n-ex 和说明性 XML 规范化之间的关系。

具体实施方式

[0064] 下面提供了所发明工作主体的详细描述。虽然描述了几个实施例，然而应当理解，所发明的工作主体不限于任何一个实施例，而是作为替代包含很多替换、修改和等效方式。另外，虽然在下面描述中阐明了很多具体细节以便提供对所发明工作主体的彻底理解，不过可以在没有这些细节中的一些或全部的情况下实现一些实施例。此外，为了清楚目的，并未详细描述在相关技术中已知的某些技术材料，以免不必要地模糊所发明的工作主体。

[0065] 共同受让的美国专利申请号 10/863,551、公开号 2005/0027871A1（“‘551 申请”）描述了数字权利管理 (DRM) 体系结构和新颖 DRM 引擎的实施例，其克服了许多先前 DRM 实现方式所体现的一些弱点，在此通过引用加以结合以供参考。本申请描述了对在‘551 申请中所描述的体系结构和 DRM 引擎的增强、扩充和修改以及可替换实施例，并且还描述了新的组件、体系结构和实施例。从而应当理解，可以在诸如‘551 申请中所描述的体系结构和 / 或 DRM 引擎的情境内以及在其它情境内使用这里所描述的材料。

[0066] 1. 示例性 DRM 系统

[0067] 图 1 示出了用于管理电子内容的说明性系统 100。如图 1 所示，用于保持电子内容 103 中权利的实体 102 封装内容以供分发和最终用户 108a-e 消费（被一起称作“最终用户 108”，其中附图标记 108 可交换地指代最终用户或最终用户的计算系统，这在本文情境下将是清楚的）。例如，实体 102 可以包括内容所有者、创建者或供应者，诸如音乐家、电影工作室、出版社、软件公司、作者、移动服务供应者、因特网内容下载或预订服务、电缆或卫星电视供应者、公司雇员等或者代表其活动的实体，并且内容 103 可以包括任何电子内容，诸如数字视频、音频或文本内容，电影、歌曲、视频游戏、软件、电子邮件消息、文本消息、字处理文档、报道或任何其它娱乐、企业或其它内容。

[0068] 在图 1 所示出的例子中，实体 102 使用封装引擎 109 来使许可 106 与所封装的内容 104 相关联。许可 106 是基于实体 102 的策略 105 或其它意愿的，并且指定允许和 / 或禁止使用内容和 / 或一个或多个条件，为了利用内容，所述一个或多个条件必须被满足，或者所述一个或多个条件作为使用条件或结果必须被满足。所述内容还可以由诸如加密或数字签名技术之类的一个或多个密码机制来保证安全，为此可以使用信任权威机构 110 来获得适当的密码密钥、证书等。

[0069] 如图 1 所示，可以借助任何适当的手段来向最终用户 108 提供封装的内容 104 和许可 106，诸如经由像因特网的网络 112、局域网 103、无线网络、虚拟专用网络 107、广域网等，经由电缆、卫星、广播或蜂窝式通信 114，和 / 或经由诸如压缩光盘 (CD)、数字多功能盘 (DVD)、快闪存储卡（例如，安全数字 (SD) 卡）之类的可记录介质 116 等。封装的内容 104

可以连同许可 106 一起以单个封包或传输 113 中或以从相同或不同源所接收的独立封包或传输中被递送给用户。

[0070] 最终用户的系统（例如，个人计算机 108e，移动电话 108a，电视和 / 或电视机顶盒 108c，便携式音频和 / 或视频播放器，电子书阅读器等）包含应用软件 116、硬件和 / 或专用逻辑单元，专用逻辑单元可操作来获取并再现所述内容。用户的系统还包括这里被称作为数字权利管理引擎 118 的软件和 / 或硬件，用于评估与所封装的内容 104 相关联的许可 106 并且强制执行它的条款（和 / 或使应用 116 能够强制执行这种条款），诸如通过有选择地准许用户只有当被许可 106 允许时才能访问所述内容来实现。数字权利管理引擎 118 可以在结构上或功能上与应用 116 集成，或者可以包括独立的软件和 / 或硬件。作为选择或另外，诸如系统 108c 之类的用户的系统可以与诸如系统 108b 之类的远程系统（例如，服务器，用户设备网络中的另一设备，诸如个人计算机或电视机顶盒等）通信，所述远程系统使用数字权利管理引擎来确定 120 是否准许用户访问该用户先前所获得或请求的内容。

[0071] 数字权利管理引擎和 / 或用户系统上或与其远程通信的其它软件还可以记录关于用户对受保护内容的访问或其它使用的信息。在一些实施例中，此信息中的一些或全部可以被传送到远程方（例如，交换所 (clearinghouse)122，内容创建者，所有者或供应者 102，用户管理者，代表其活动的实体等），例如以用于分配收益（诸如使用费，基于广告的收益等）、确定用户偏好、强制系统策略（例如，监视怎样以及何时使用机密信息）等。应当理解，虽然图 1 示出了说明性的 DRM 体系结构和一组说明性关系，但是可以在任何适当的情境中实施这里所描述的系统和方法，并且从而应当理解，图 1 是用于图示和解释目的而不是限制目的。

[0072] 图 2 示出了可以用来实施本发明工作主体的实施例的系统 200 的更详细的例子。例如，系统 200 可以包括最终用户的设备 108、内容供应者的设备 109 等的实施例。例如，系统 200 可以包括诸如个人计算机 108e 或网络服务器 105 之类的通用计算设备或诸如蜂窝式电话 108a、个人数字助理、便携式音频或视频播放器、电视机顶盒、信息亭、游戏系统等专门计算设备。系统 200 一般包括处理器 202、存储器 204、用户接口 206、用于接受可拆卸存储器 208 的端口 207、网络接口 210 和用于连接上述元件的一个或多个总线 212。系统 200 的操作一般由在存储器 204 中所存储的程序指导下操作的处理器 202 来控制。存储器 204 通常包括高速随机存取存储器 (RAM) 和非易失性存储器，诸如磁盘和 / 或闪速 EEPROM。存储器 204 的某些部分可以被限制，以致它们无法被系统 200 的其它组件读取或写入。端口 207 可以包括用于接受计算机可读介质 208 的盘驱动器或存储器插槽，所述计算机可读介质 208 诸如软盘、CD-ROM、DVD、存储卡、SD 卡、其它磁或光介质等。网络接口 210 一般可操作来经由诸如因特网或内部网（例如，LAN，WAN，VPN 等）之类的网络 220 来在系统 200 及其它计算设备（和 / 或计算设备网络）之间提供连接，并且可以使用一种或多种通信技术来在物理上进行这种连接（例如，无线，以太网等）。在一些实施例中，系统 200 还可以包括处理单元 203，所述处理单元 203 受到保护以避免被系统 200 的用户或其它实体篡改。这种安全的处理单元可以帮助增强诸如密钥管理、签名验证之类的敏感操作以及数字权利管理过程的其它方面的安全性。

[0073] 如图 2 所示，计算设备 200 的存储器 204 可以包括用于控制计算设备 200 操作的各种程序或模块。例如，存储器 204 一般包括用于管理应用、外围装置等执行的操作系统 220；

用于再现受保护的电子内容的主机应用 230；和用于执行这里所描述权利管理功能中的一些或全部的 DRM 引擎 232。如在此的其它地方所描述，DRM 引擎 232 可以包括、与之相互操作和 / 或控制各种其它模块，诸如用于执行控制程序的虚拟机 222，和用于存储状态信息以供虚拟机 222 使用的状态数据库 224，和 / 或用于执行密码运算（诸如加密和 / 或解密内容、计算散列函数和消息认证代码、评估数字签名等）的一个或多个密码模块 226。存储器 204 一般还包括受保护的内容 228 和相关联的许可 229 以及密码密钥、证书等（未示出）。

[0074] 一个本领域普通技术人员应当理解，这里所描述的系统和方法可以利用与图 2 中所图示的类似或完全相同的计算设备或实际上任何其它适当的计算设备来实施，包括不拥有在图 2 中所示出的一些组件的计算设备和 / 或拥有未示出的其它组件的计算设备。从而应当理解，图 2 只用于图示而并非是限制目的。

[0075] 这里描述了数字权利管理引擎和相关的系统和方法，其可以用来提供在图 1 和 2 中所示出的那些系统或其它类型系统中的权利管理功能中的一些或全部。另外，下面将描述各种其它系统和方法，其可以用在图 1 和 2 中所示出的那些系统情境中以及其它情境中，包括与数字权利管理无关的环境。

[0076] 2. DRM 引擎体系结构

[0077] 在一个实施例中，使用相对简单、开放和灵活的数字权利管理 (DRM) 引擎来执行核心 DRM 功能。在优选实施例中，此 DRM 引擎被设计成用于相对容易地集成到诸如在‘551 申请中所描述的 web 网络服务环境中，以及集成到可能的任何主机环境或软件体系结构中。在优选实施例中，DRM 引擎与特定的媒体格式和密码协议无关，允许设计者按照特定情况的要求来灵活地使用标准化或专有技术。由 DRM 引擎的优选实施例所使用的管理支配 (governance) 模型是简单的，但是该模型可以用来表达良好的关系和业务模型。

[0078] 下述 DRM 引擎的一些说明性实施例涉及被称为“章鱼 (Octopus)”的示例性实现方式；然而应当理解，本发明并不限于章鱼例子的具体细节，并且仅用于说明而并非限制性目的。

[0079] 1.1. 概述

[0080] 图 3 示出了说明性的 DRM 引擎 303a 在使用 DRM 的系统 302 中可以怎样来起作用。如图 3 所示，在一个实施例中，DRM 引擎 303a 被嵌入或集成到主机应用 304a（例如，诸如音频和 / 或视频播放器之类的内容再现应用，诸如电子邮件程序、文字处理器、电子书读取器或文件阅读器等文本再现应用）内或与之通信。在一个实施例中，DRM 引擎 303a 执行 DRM 功能并且对于诸如加密、解密、文件管理之类的服务依赖主机应用 304a，和 / 或可以由主机来更有效地提供其它功能。例如，在优选实施例中，DRM 引擎 303a 可操作来操纵 DRM 对象 305，所述 DRM 对象 305 包括受保护内容 308 的许可 306。在一些实施例中，DRM 引擎 303a 还可以向主机应用 304a 递送密钥。如图 3 所示，DRM 引擎 303a 和主机应用 304a 中的一个或两个可以利用要处理的 web 网络服务 305a 和 / 或主机服务 306a 和 / 或为完成它们各自的任务所需要的信息。‘551 申请提供了这种服务的例子以及其中 DRM 引擎 303a 和主机应用 304a 可以相互操作的方式。

[0081] 在图 3 所示出的例子中，DRM 引擎 303a、主机应用 304a、主机服务 306a 和 web 网络服务接口 305a 被加载到诸如最终用户的个人计算机 (PC) 之类的设备 300a 上。设备 300a 可通信地耦合到服务器 300b 以及便携式设备 300d，其中从所述服务器 300b 获得内容 308

和许可 306，并且设备 300a 可以向便携式设备 300d 转发内容 308 和 / 或许可 306。这些其它设备中的每个可以包括与 DRM 引擎 300a 类似或完全相同的 DRM 引擎 303，其可以与所述设备的特定主机应用和主机环境相集成。例如，服务器 300b 可以包括主机应用 304b，用于执行成批封装内容和 / 或许可，并且利用 DRM 引擎 300a 来评估与正被封装的内容相关联的控制，以便与任何重新分发的限制相一致。类似地，设备 300c 可以包括能够再现并封装内容的主机应用 304c，而设备 300a 可以包括只能再现内容的主机应用。作为潜在的主机环境多样性的又一例子，设备 300d 可以不包括 web 网络服务接口，但是作为替代可以依赖于与设备 300a 的通信和 web 网络服务接口 305a，以达到这样的程度 ：主机应用 304d 和 / 或 DRM 引擎 303d 需要使用任何 web 网络服务。图 3 只是其中可以使用 DRM 引擎的系统的一个例子 ；应当理解，这里所描述的 DRM 引擎的实施例可以依照许多不同的方式执行并且与应用和系统集成，并且不限于在图 3 中所示出的说明性例子。

[0082] 1.2. 对象

[0083] 在优选实施例中，内容保护和管理支配对象用来表示系统中的实体、保护内容、使使用规则与内容相关联并且确定当请求时访问是否被准许。

[0084] 如下面所更详细地描述，在一个实施例中使用以下对象 ：

[0085]

对象类型	功能
节点	表示实体
链路	表示在实体之间的直接关系
内容	表示内容（例如，媒体内容）
内容密钥	表示加密内容所使用的加密密钥
控制	表示管理支配与内容交互的使用规则
控制器	表示在控制和内容密钥对象之间的关联
保护器	表示在内容和内容密钥对象之间的关联

[0086] 1.2.1. 节点对象

[0087] 节点对象用来表示系统中的实体。在实践中，节点通常表示用户、设备或组。节点对象一般还具有相关联的属性，所述相关联的属性表示与节点相关联的实体的某些特性。

[0088] 例如，图 4 示出了两个用户 (Xan 400 和 Knox 402)、两个设备 (PC404 和便携式设备 406) 和用于表示组（例如，凯里家庭的成员 408、公共图书馆的成员 410、特定音乐服务的订户 412、RIAA 批准的设备 414 以及特定公司所制造的设备 416) 的几个实体，每一个都具有相关联的节点对象。

[0089] 在一个实施例中，节点对象包括用于定义该节点用来表示什么的属性。属性的一个例子是节点类型。除表示用户、组或设备之外，节点类型属性可以用来表示其它实体。在一些实施例中，节点对象还可以包括密码密钥信息，诸如当使用别处描述的密钥推导和分发技术的实施例时的密码密钥信息。

[0090] 在一些实施例中，节点对象还包括保密性的非对称密钥对，用于把机密信息目标定为有权访问所述节点对象的机密部分的子系统。这可以是节点所表示的实体（例如，音

乐服务 412）或负责管理所述节点的某个实体（例如，最终用户（例如，Knox 402）可以负责管理他或她的便携式设备 406）。

[0091] 1.2.2. 链路对象

[0092] 在优选实施例中，链路对象是用于示出在两个节点之间关系的签名的（signed）的对象。例如，在图 4 中，从 PC 节点 404 到 Knox 402 的链路 418 示出了所有权。从 Knox 402 到凯里家庭节点 408 的链路示出了成员资格，从所述 Carey 家庭节点 408 到音乐服务订户节点 412 的链路也是这样。在一个实施例中，链路对象表达在两个节点之间的关系，从而在图 4 中所示出的关系可以使用十个链路来表示。

[0093] 如图 4 所示，图 420 可以用来表达在节点之间的关系，其中链路对象是在节点之间的有向边。例如在图 4 中，在凯里家庭节点 408 和音乐服务节点 412 之间的关系断言在该图中存在其顶点为凯里家庭节点 408 和音乐服务节点 412 的有向边 422。Knox 402 和 Xan 400 是凯里家庭 408 的成员。因为 Knox 402 被链接到凯里家庭 408 并且所述凯里家庭 408 被链接到音乐服务 412，所以认为在 Knox 402 和音乐服务 412 之间存在路径。当存在从一个节点到其它节点的路径时，DRM 引擎认为该节点可从另一节点到达。这允许写入控制，用于允许根据节点可从其中正执行用于请求访问受保护内容的应用的设备到达的条件来许可访问受保护的内容。

[0094] 如下面所更详细地描述，链路对象选择性地还可以包含用于允许推导内容密钥的一些密码数据。链路对象还可以包含用于定义所述链路可以被认为有效的条件的控制程序。这种控制程序可以被 DRM 引擎的虚拟机执行或解释（这里这些术语可交换地使用）以便评估链路的有效性（例如，以便确定所述链路是否可以用来到达授权图中的给定节点）。

[0095] 在一个实施例中，链路被签名。可以使用任何适当的数字签名机制，并且在一个实施例中 DRM 引擎没有定义链路对象怎样被签名并且没有评估任何相关联的证书，作为替代它依赖主机系统来验证任何这种签名和 / 或证书。这允许系统设计者或管理员定义链路对象的使用期、撤消它等（例如，通过使用期满密钥或证书、撤销等），从而在特定的受保护内容和 / 链路的情境下在通过 DRM 引擎评估控制程序和 DRM 对象所提供的策略管理和安全性之上提供了附加的策略管理和安全性层（例如，链路期满作为选择或另外可以通过在链路对象本身中包括适当的控制程序来实现，所述控制程序当执行时会强制实施期满日期或其它有效期）。在一个实施例中，DRM 引擎是通用的，并且与任何适当的加密、数字签名、撤销和 / 或由主机应用和 / 或环境所使用的其它安全机制一起工作。从而，例如如果 DRM 引擎需要确定特定链路是否已经被适当的签名，那么它可以简单地调用主机应用（和 / 或主机或系统密码服务）以便依照由系统设计者所选择的特定签名方案来验证所述签名，DRM 引擎本身可以不知道所述方案的细节。在其它实施例中，DRM 引擎本身依赖主机简单地表明所使用的适当签名算法，来执行实际的签名评估。

[0096] 1.2.3. 内容保护和管理支配

[0097] 再次参照图 3，在典型的方案中，内容供应者 300b 使用应用 304b，所述应用 304b 包括用于加密或用密码来保护电子内容 308 的封装引擎，并且所述内容供应者 300b 创建用于管理支配对该内容的访问或其它使用的许可 306。在一个实施例中，许可 308 包括用于指定可以怎样使用内容 308 的一组对象 305，并且还包括内容的加密密钥和 / 或为获得它们所需要的信息。在一个实施例中，内容 308 和许可 306 在逻辑上是独立的，但是通过内部引用

（例如，使用对象 ID 310）被绑定在一起。在许多情况中，把内容和许可一起存储和 / 或一起递送可能是方便的；然而在优选实施例中并不要求这样。在一个实施例中，许可可以应用于一个以上的内容项，并且一个以上许可可以应用于任何单个的内容项。

[0098] 如图 3 所示，当在客户端设备 300a 上运行的主机应用 304a 想要对特定的内容 308 执行动作时，它请求 DRM 引擎 303a 检查它旨在执行的动作（例如“播放”）是否被允许。在一个实施例中，DRM 引擎 303a 从在包括内容许可 306 的对象 305 中所包含的信息中加载并执行与内容 308 相关联的控制程序，并且根据所述控制程序所返回的结果来准许或拒绝用于执行该动作的权限。权限一般要求满足一些条件，诸如节点可从用于表示请求实体 / 设备 300a 的节点到达的条件。

[0099] 图 5 是用于图示 DRM 引擎的实施例可以怎样确定所请求的动作（例如，观看内容）是否被授权的流程图。如图 5 所示，接收用于评估对给定动作的许可的请求 (500)。例如，在主机从用户接收用于执行所指定动作的请求之后，可以从主机应用接收此请求。如图 5 所示，DRM 引擎评估所指定的许可 (502)，并且确定所请求的动作是否被授权 (504)。例如，所述许可可以包含 DRM 引擎执行的控制程序，其输出用来进行授权判定。如果许可授权所请求的动作（即，从块 504“是”退出），那么 DRM 引擎向主机应用表明所述请求被准许 (506)。否则，DRM 引擎向主机应用表明所述请求被拒绝 (508)。在一些实施例中，DRM 引擎还可以向主机应用返回各种元数据，所述元数据例如用于把条件与准许授权相关联（例如，义务 (obligation) 和 / 或回调 (callback)），或者提供关于拒绝授权原因的附加信息。例如，DRM 引擎可以表明只有主机应用记录关于所请求动作执行的某些信息才允许所请求的动作，或者只要所述主机应用以预定义的时间间隔回调 DRM 引擎以便例如重新评估许可，才允许所请求的动作。下面提供了由 DRM 引擎所返回的关于这种义务、回调的附加信息及其它元数据。如果所请求的动作被授权，那么内容密钥将被（例如，从许可的内容密钥对象中）获取，并且用来就所请求的使用而发布所述内容。

[0100] 1.2.4. 许可 DRM 对象

[0101] 如图 6 所示，在优选实施例中，许可 600 是对象集合。在图 6 所示出的例子中，许可 600 包括内容密钥对象 602、保护器对象 604、控制器对象 606 和控制对象 608。如图 6 所示，内容密钥对象 602 包括加密的密钥数据 610（例如，为解密所加密的内容项 612 所需要的密钥的加密版本）和关于用于加密所述密钥数据的密码系统的信息。保护器对象 604 把内容密钥对象 602 绑定到一个或多个内容对象 614。如图 6 所示，控制对象 608 包括并保护用于指定怎样管理支配内容对象 614 的控制程序 616。在优选实施例中，控制程序 616 是在由 DRM 引擎所操作的虚拟机上运行的可执行字节代码。控制程序通过检查是否满足在所述控制程序中所指定的条件来管理支配是否可以对所述内容执行某些动作，所述条件诸如某些节点是否可使用有效链路对象到达，某些状态对象是否已经被存储，主机环境是否具有某些特征等。再次参照图 6，控制器对象 606 用来把一个或多个内容密钥对象 602 绑定到控制对象 608。

[0102] 许可 600 还可以包括另外的对象，诸如用于提供所述许可所要求的内容访问条件的机器或人类可读描述的元数据。作为选择或另外，可以包括这种元数据作为一个其它对象（例如，控制对象 608）的资源扩展。在图 6 所示出的实施例中，控制对象 608 和控制器对象 606 都被签名，使得该系统可以在使用控制信息来进行内容访问判定之前来验证所述

控制信息是否来自信任源。在一个实施例中，还可以通过验证在控制器对象 606 中所包括的安全散列来检查控制对象 608 的有效性。控制器对象 606 还可以包含在它引用的内容密钥对象 602 中包含的每个密钥或其它密钥数据的散列值，由此使攻击者篡改在密钥数据和内容密钥对象之间的绑定变得相对困难。

[0103] 如图 6 所示，在一个实施例中，内容 612 被加密并且包括在内容对象 614 中。所使用的解密密钥被包括在内容密钥对象 602 内（或由其引用），并且在这两者之间的绑定由保护器对象 604 来表示。如图 6 所示，唯一的 ID 用来使在内容对象 614 和内容密钥对象 602 之间的绑定便于进行。用于管理支配使用密钥 610 来解密内容 612 的规则包括在控制对象 608 内，并且在控制对象 608 和内容密钥 602 之间的绑定由控制器对象 606 再次使用唯一的 ID 来表示。

[0104] 应当理解，虽然图 6 示出了在一个优选实施例中包括许可的对象，不过这里所描述的 DRM 系统和方法不限于此许可结构的使用。例如而并非限制，可以使用许可，其中在图 6 中所示出的各个对象的功能依照不同的方式可以被组合到少量对象中，或传播到另外的对象上，或者在对象之间分散。作为选择或另外，可以利用许可来实施这里所描述的系统和方法的实施例，所述许可缺乏由在图 6 中所示出的许可结构具有的一些功能，和 / 或提供另外的功能。从而应当理解，依照这里所描述的原理可以使用用于使许可与内容相关联的任何适当机制，不过在优选实施例中使用在图 6 中所示出的有益结构。

[0105] 1.3. 状态数据库

[0106] 在一个实施例中，DRM 引擎包括或有权访问可以用来提供安全状态存储机制的安全持久的对象存储装置。这种机构对于使控制程序能够读取和写入从调用到调用是持久的状态信息来说是有用的。这种状态数据库可以用来存储状态对象以及会员资格状态和 / 或任何其它适当的数据，所述状态对象诸如播放计数、首次使用日期、累积的再现时间等。在一些实施例中，在第一系统上执行的 DRM 引擎可能不具有访问本地状态数据库的权利，并且可操作来例如使用 web 网络和 / 或主机服务来访问远程状态数据库。在一些情况中，在第一系统上执行的 DRM 引擎可能必须访问在远程系统上的数据库中所存储的状态信息。例如，第一系统可能不包括状态数据库，或者可能在其自己的状态数据库中没有它所需要的信息。在一些实施例中，当 DRM 引擎面临这种情况时，它可以经由服务接口和 / 或通过使用代理程序来访问远程状态数据库，如下面所更详细地描述。

[0107] 1.4. 关于控制程序

[0108] 这里所描述的系统和方法在各种情境中利用控制程序。例如，在控制对象中所包含的控制程序可以用来表达用于管理支配使用受保护内容的规则和条件。另外，链路对象中的控制程序可以用来表达用于确定对于给定目的（例如，节点可达性分析）所述链路是否有效的规则和条件。这种控制程序有时这里被称作为链路约束 (link constraints)。其中可以使用控制程序的又一情境是在代理或委托对象中，其中控制代码用来代表另一实体（在代理控制程序的情况下）或代表另一控制（在委托控制程序的情况下）执行动作。

[0109] 在一个实施例中，和直接由物理处理器执行相反，控制程序由为 DRM 引擎主控的虚拟机来执行或解释。然而应当理解，可以容易地构造物理处理器或其它硬件逻辑单元来执行控制程序。在一个实施例中，控制程序采用字节代码格式，其使跨平台进行的相互操作便于进行。

[0110] 在优选实施例中，控制程序用汇编语言编写并且被汇编程序转换为字节代码。在其它实施例中，模板和 / 或高级权利表达式语言可以用来提供权利、规则和 / 或条件的初始表达式，并且可以使用编译器来把所述高级表达式转换为字节代码以供这里所描述 DRM 引擎的实施例执行。例如，用专有 DRM 格式编写的权利表达式可以利用适当的编译器被转换或翻译为在功能上等效的字节代码表达式以供在这里所描述的 DRM 引擎实施例上执行，从而使受保护的内容能够依照由内容供应者所指定的条件在理解所述专有 DRM 格式的系统上以及在包括诸如这里所描述的 DRM 引擎的系统上使用。还应当理解，这里所描述的数字权利管理引擎系统和方法不限于使用由虚拟机所解释的字节代码权利表达式。作为替代在一些实施例中，可以依照任何适当的方式（例如，使用高级权利表达式语言 (rights expression language REL)、模板等）来表示权利，并且这里所描述的授权图和 / 或其它技术使用被设计成用于识别并评估这种权利表达式的应用程序来执行。

[0111] 1.4.1. 条件

[0112] 如先前所表明，控制程序一般表达为了对内容的使用请求被准许、为了链路被确认为有效等所必须满足的一个或多个条件。取决于内容供应者或系统设计者的要求和 / 或所述系统所提供的功能，可以使用任何适当的条件。

[0113] 在优选实施例中，由 DRM 引擎所使用的虚拟机支持任意复杂的程序，所述程序能够测试诸如以下一些或全部的条件 :

[0114] ●基于时间的条件 :把客户端时间值与在控制程序中所指定的一个或多个值相比较。

[0115] ●以特定节点为目标 :检查某个节点是否可从另一节点到达。此原理提供了对如域、预订、成员资格等这种模型的支持。

[0116] ●测试某些节点属性是否匹配所指定的值 :检查节点的任何属性，诸如与节点相关联的设备的再现能力是否满足保真度要求。

[0117] ●测试在客户端与安全相关的元数据是否是最新的 :例如检查客户端是否具有可接受版本的客户端软件以及准确的时间量度。在一些实施例中，这种检查例如可以依赖于来自数据证明服务的一个或多个证书中的声明。

[0118] ●基于状态的条件 :检查状态数据库中的信息。例如，状态数据库可以包含作为控制程序先前执行的结果所产生的信息和 / 或令牌，及关于所记录事件和条件的其它信息，所述令牌用于证明预订的所有权、成员资格等，由此使得可以评估涉及计数器的条件（例如，播放次数、输出次数、所过去的时间限制等）。

[0119] ●环境特性 :例如，检查主机环境中的硬件和 / 或软件是否具有某些特征，诸如识别并强制义务的能力 ;检查诸如安全输出信道之类的某些软件或硬件组件的存在或不存在 ;检查邻近度信息，诸如请求设备与另一设备或应用的邻近度 ;使用网络服务和 / 或代理来检查远程系统和 / 或在其上所存储数据的特征 ;等。

[0120] 使用这些或任何其它适当的条件，控制对象可以表达用于管理支配可以怎样再现、转送、输出等内容的规则。应当理解，以上条件列表实际上是说明性的，而且可以通过例如执行用于测试所想要条件的系统调用来定义并使用任何适当的条件。例如而并非限制，如果希望要求设备位于特定的子网络上，那么可以定义可操作来返回主机设备的 IPConfig 信息（或远程设备的 IPConfig 信息，如果使用代理在远程设备上运行系统调用的话）的系

统调用（例如，GetIPConfig），其可以由控制程序用来测试所述设备是否位于所规定的子网络上。

[0121] 1.4.2. 代理

[0122] 这里所描述的 DRM 引擎相关的系统和方法的优选实施例提供了对携带控制程序的独立对象的支持。这种“代理”可以被分发给在远程系统上运行的 DRM 引擎以便实现所指定的功能，诸如写入到远程 DRM 引擎的安全状态存储中。例如，可以作为联系远程服务或执行远程控制程序的结果来发送代理。还可以使用代理来实现内容移动操作、初始化计数器、撤销节点的注册等。作为又一例子，可以使用代理来执行从远程节点到另一节点的可达性分析。这种代理例如在强制实施用于禁止被注册给第一用户的设备再被注册给第二用户的策略方面是有用的。如果第二用户请求注册，那么代理可以被第二用户或者代表他或她活动的注册服务发送到该设备，，以便确定所述设备是否已经被注册到第一用户，在这种情况下第二用户的注册请求可能会被拒绝。

[0123] 图 7A 和 7B 图示了在一个实施例中代理的使用。如图 7A 所示，假定两个实体——系统 A 700 和系统 B 702——想要彼此经由计算机网络 703 通信，并且 DRM 系统正被使用且能够描述并强制实施用于某些操作的规则，诸如访问受保护的内容或创建可以用来表示成员资格、注册状态等的 DRM 对象。在一些情况下，规则会在系统 A 700 上被评估，但是规则要求取决于系统 B 702 的状态的信息。该信息需要被 DRM 系统 704 信任，所述 DRM 系统 704 在系统 A 700 上强制实施所述规则。

[0124] 例如，系统 A 700 上的 DRM 系统 704 可以评估 / 强制实施用于执行内容从系统 A 700 到系统 B 702 的远程再现的规则，并且所述规则可以表明只有当系统 B 702 是确定设备组的一部分时才允许这种操作，其中借助可在系统 B 702 上访问的安全状态数据库 716 中的状态对象 711 的存在来声明该组中的成员资格。

[0125] 在优选实施例中用来处理这种情况的方法利用代理。例如，如果系统 A 700 需要来自系统 B 702 的信息，那么系统 A 700 准备代理 705，所述代理 705 在一个实施例中是从系统 A 700 向系统 B 702 发送的控制程序（例如，可以由 DRM 引擎执行的指令序列）。在一个实施例中，系统 A 700 经由认证的通信信道 720 向系统 B 702 发送代理代码 705，使得系统 A 700 可以确信代理 705 的确是运行在系统 B 702 上。在一些实施例中，连同代理代码 705 一起，系统 A 700 还可以向系统 B 702 传送可以由代理代码 705 用来执行其工作的一个或多个参数。

[0126] 如图 7B 所示，系统 B 702 接收代理 705 和任何相关联的代理参数，并且运行代理代码 705。当代理 705 在系统 B 702 上运行时，它访问系统 B 的状态数据库 716，获取状态信息 711 和 / 或用其执行一个或多个计算，并且把结果 713 发送回到系统 A 700，优选经由认证的通信信道 710。在这一点上，系统 A 700 具有需要用来继续进行其评估的信息。

[0127] 1.4.3. 链路约束

[0128] 在一个实施例中，用于表示用来管理支配对内容项执行某个操作（诸如“播放”）的规则的例程集被称作“动作控制”。用于表示关于链路对象的有效性约束的例程集被称作“链路约束”。像动作控制一样，在优选实施例中，链路约束可以表达任何适当的条件组合。还像动作控制一样，链路约束可以被本地评估和 / 或使用服务接口或代理来远程评估。

[0129] 1.4.4. 义务和回调

[0130] 在一个实施例中，某些动作当被准许时要求来自主机应用的进一步参与。义务表示当使用主机应用请求的内容密钥时或在此之后需要由所述主机应用执行的操作。回调表示对一个或多个控制程序例程的调用，所述控制程序例程当使用主机应用请求的内容密钥时或在此之后需要被主机应用执行。义务的例子包括但不限于当内容正被再现时关闭某些输出和 / 或控制的要求（例如，以便防止把所述内容写入到不受保护的输出或防止快速转发通过某些重要的内容段）；关于内容使用的信息被记录（例如，计量或审计信息）和 / 或被发送到远程站点（例如，交换所，服务供应商等）的要求；本地或远程执行代理程序的要求；等。回调的例子包括但不限于在某个绝对时间主机向回调用控制程序的要求，在某个消逝的时间（例如，使用内容所消逝的时间）之后，在发生某个事件（例如，完成试用内容再现时段）之后，当已经停止使用内容时等。例如，在某个过去时间之后的回调可以用来增加或减少预算、播放计数等（例如，只是在用户使用一段内容至少达一定量时间才把用户预算记入帐），从而保护用户免得在他或她偶然按压播放按钮但是立即按下停止时就从他或她的帐户中记帐。

[0131] 在一个实施例中，存在不同类型的义务和回调，并且如果应用遇到它不支持或者不理解的任何关键的义务或回调（例如因为所述义务类型可能在执行应用之后才定义），那么就需要所述应用拒绝继续要为其返回此义务或回调参数的动作。

[0132] 1.4.5. 例子

[0133] 图 8-12 示出了 DRM 引擎的说明性实施例可以怎样控制使用内容的例子。参照图 8，假定 DRM 引擎已经接收用于播放内容项 802、804 的组 800 的请求。例如，内容项 802、804 可以包括从预订服务、电子邮件附件等中获得的不同的内容段、多媒体展示的不同子部分和曲集的不同曲目。所述请求可以由 DRM 引擎从主机应用接收，所述主机应用随后接收来自计算设备的用户的请求，所述主机应用运行在所述计算设备上。来自主机应用的请求一般标识所请求的动作、要对其采取动作的一个或多个内容以及管理支配所述内容的许可。DRM 引擎按照在图 5 中所图示的过程来确定是否应当准许请求。

[0134] 图 8 和 9 提供了在图 5 中所示出的过程的更详细的非限制性例子。参照图 9，当接收对访问内容项 802 和 804 的请求时（块 900），DRM 引擎检查在所述请求中所标识的或其拥有的许可来看看是否存在有效许可。例如，DRM 引擎可以首先标识保护器对象 806 和 808，其包含内容项 802 和 804 的唯一标识符（即分别为 NS :007 和 NS :008）（图 9 中的块 902）。接下来，DRM 引擎定位在保护器对象 806 和 808 中所标识的内容密钥对象 810 和 812（图 9 中的块 904），所述保护器对象 806 和 808 随后使所述 DRM 引擎能够标识引用内容密钥对象 810 和 812 的控制器 814（图 9 中的块 906）。在优选实施例中，控制器 814 被签名，并且 DRM 引擎验证其签名（或请求主机服务来验证它）。DRM 引擎使用控制器 814 来标识用于管理支配对内容密钥对象 810 和 812（从而内容项 802 和 804）的使用的控制对象 816（图 9 中的块 908）。在优选实施例中，DRM 引擎验证控制对象 816 的完整性（例如，通过计算控制对象 816 的摘要 (digest) 并且把它与在控制器 814 中所包含的摘要相比较）。如果完整性验证取得成功，那么 DRM 引擎执行在控制对象 816 中所包含的控制代码（块 910），并且把结果返回到主机应用（块 912），所述主机应用使用它来准许或拒绝用户对访问内容的请求。控制代码的结果还可以选择性地指定主机应用将需要满足的一个或多个义务或回调。

[0135] 图 10 是 DRM 引擎可以怎样执行在图 9 的块 910 和 912 中所指定的动作的更详细例

子（即，执行控制程序并且返回结果）。如图 10 所示，当标识相关的控制对象时，DRM 引擎把在控制对象中所包含的字节代码加载到优选由所述 DRM 引擎主控的虚拟机中（块 1000）。DRM 引擎和 / 或虚拟机典型情况下还初始化虚拟机的运行时环境（块 1002）。例如，虚拟机可以分配为执行控制程序所需要的存储器、初始化寄存器及其它环境变量，和 / 或（例如，如下所述通过进行 System.Host.GetObject 调用）获得关于虚拟机操作的主机环境的信息。应当理解，在一些实施例中，块 1000 和 1002 可以被有效地组合或交织，和 / 或次序颠倒。如图 10 所示，虚拟机接下来执行控制程序的字节代码（块 1004）。这里如其它地方所描述，这可以涉及调用其它虚拟机代码、从安全存储装置获取状态信息等。当控制程序已经完成执行时，它提供输出（例如在优选实施例中为 ExtendedStatusBlock），其例如可以由调用应用用来确定请求是否已经被准许，并且如果是的话，确定任何义务或回调是否与其相关联；请求是否已经被拒绝，并且如果是的话，确定拒绝的原因；或者确定在执行期间是否出现任何错误（块 1006）。

[0136] 如先前所表明，在控制对象 816 中所包含的控制代码指定为了对内容项 802 和 804 进行所请求的使用而必须满足的条件或其它要求。这里所描述的系统和方法能够说明任意复杂的条件组；然而为了此例子目的，假定控制程序被设计成要求为了播放内容项 802 和 804，(a) 给定用户的节点必须可从对其作出播放内容请求的设备到达，和 (b) 当前日期必须在所规定日期之后。

[0137] 图 11 示出了在设备 1102 上运行的 DRM 引擎 1100 的说明性实施例可以怎样执行上述示例性控制程序，并且图 12 是在执行过程中所涉及的步骤的流程图。如图 11 所示，DRM 引擎 1100（例如，通过调用 System.Host.SpawnVm）创建虚拟机执行环境 1104 并且加载控制程序。虚拟机 1104 在由 DRM 引擎 1100 所指定的入口点（例如，在 Control.Actions.Play.perform 例程的位置）开始执行控制程序。在此例子中，控制程序需要确定给定节点是否可从其上运行 DRM 引擎 1100 的设备 1102 的个性 (personality) 节点到达。为了进行此确定，控制程序向 DRM 引擎 1100 所提供的链路管理器服务 1106 作出调用 1105，指定要求链接到的节点（图 12 中的块 1200）。链路管理器 1106 负责评估链路对象来确定一个节点是否可从另一节点到达。为了有效地完成这点，链路管理器 1106 可以预先计算从设备 1102 的个性节点 1110 到在该设备 1102 所拥有的任何链路对象中所指定的各个节点 1114 是否存在路径。即，链路管理器 1106 可以通过检查它有权访问的链路的“去往”和“来自”字段来简单地确定哪些节点潜在地可从设备 1102 的个性节点 1110 到达。当链路管理器 1106 接收来自虚拟机 1104 的调用 1105 时，它通过首先确定从个性节点 1110 到所指定的节点 1112 是否存在路径（例如，通过检查在它先前确定在理论上可到达的节点的列表中的节点 ID）来确定所指定的节点 1112 是否是可到达的（图 12 中的块 1202）。如果存在路径，那么链路管理器 1106 评估在链路中所包含的任何控制程序来看看所述链路是否有效（图 12 中的块 1204-1210）。为了评估链路对象中的控制程序（图 12 中的块 1206），链路管理器 1106 可以使用其自己的虚拟机 1108，所述链路管理器 1106 在其上执行在所述链路对象中所包括的控制程序。链路管理器 1106 将其确定结果（即，给定节点是否是可到达的）返回到在虚拟机 1104 中执行的控制程序，其中它被用于全面评估播放内容的请求是否将被准许。当确定所指定的节点 1112 可从设备 1102 的个性节点 1110 到达时，在虚拟机 1104 上执行的控制程序接下来确定是否满足所指定的日期限制（图 12 中的块 1212）。如果已经满足日期

限制（即，从块 1212 以“是”退出），那么控制程序返回用于表明已经满足所指定条件的结果（图 12 中的块 1214）；否则，控制程序返回用于表明不满足所指定条件的结果（图 12 中的块 1216）。

[0138] 下面示出了诸如上述控制程序的例子：

[0139] ;样本控制

[0140] ;

[0141] ;此控制检查用户节点是否是可到达的

[0142] ;以及日期是否在特定开始日期之后

[0143] ;以及在特定结束日期之前

[0144] ;从控制中的属性获取值

[0145] ;==========

[0146] ;常量

[0147] ;==========

[0148]

```
.equ DEBUG_PRINT_SYSCALL,                1
.equ FIND_SYSCALL_BY_NAME,               2
.equ SYSTEM_HOST_GET_OBJECT_SYSCALL,     3
.equ SUCCESS,                            0
.equ FAILURE,                            -1
```

[0149] ;==========

[0150] ;数据

[0151] ;==========

[0152]

```
.data

ControlTargetNodeIdAttributePath:
    .string "Octopus/Control/Attributes/TargetNodeId"
ControlStartDateAttributePath:
    .string "Octopus/Control/Attributes/StartDate"
ControlEndDateAttributePath:
    .string "Octopus/Control/Attributes/EndDate"
TargetNodeId:
     .zeros 256
StartDate:
    .long 0
EndDate:
```

[0153]

```
        .long -1
    IsNodeReachableFunctionName:
        .string "Octopus.Links.IsNodeReachable"
    IsNodeReachableFunctionNumber:
        .long    0
    GetTimeStampFunctionName:
        .string "System.Host.GetLocalTime"
    GetTimeStampFunctionNumber:
        .long    0
```

[0154] ;==========

[0155] ;代码

[0156] ;==========

[0157] .code

[0158] Global.OnLoad：

[0159] ;加载全局函数

[0160] ;获取用于 Octopus.Links.IsNodeReachable 的 syscall 数目

[0161] PUSH@IsNodeReachableFunctionName

[0162] PUSH FIND_SYSCALL_BY_NAME

[0163] CALL

[0164] DUP

[0165] PUSH@IsNodeReachableFunctionNumber

[0166] POKE

[0167] BRN OnLoad_Fail

[0168] ;获取用于 System.Host.GetTimeStamp 的 syscall 数目

[0169] PUSH@GetTimeStampFunctionName

[0170] PUSH FIND_SYSCALL_BY_NAME

[0171] CALL

[0172] DUP

[0173] PUSH@GetTimeStampFunctionNumber

[0174] POKE

[0175] BRN OnLoad_Fail

[0176] ;好

[0177] PUSH 0

[0178] RET

[0179] OnLoad_Fail：

[0180] PUSH FAILURE

```
[0181]      RET
[0182]  Control.Actions.Play.Init：
[0183]      ;从属性中获取值
[0184]      ;获取目标节点（保证在那）
[0185]      PUSH 256 ;返回缓冲区大小（256 个字节）
[0186]      PUSH@TargetNodeId ;返回值
[0187]      PUSH@ControlTargetNodeIdAttributePath ;名称
[0188]      PUSH 0 ;亲代＝根容器
[0189]      PUSH SYSTEM_HOST_GET_OBJECT_SYSCALL
[0190]      CALL
[0191]      ;获取开始日期
[0192]      PUSH 4 ;返回缓冲区大小（4 个字节）
[0193]      PUSH@StartDate ;返回值
[0194]      PUSH@ControlStartDateAttributePath ;名称
[0195]      PUSH 0 ;亲代＝根容器
[0196]      PUSH SYSTEM_HOST_GET_OBJECT_SYSCALL
[0197]      CALL
[0198]      ;获取结束日期
[0199]      PUSH 4 ;返回缓冲区大小（4 个字节）
[0200]        PUSH@EndDate ;返回值
[0201]        PUSH@ControlEndDateAttributePath ;名称
[0202]        PUSH 0 ;亲代＝根容器
[0203]        PUSH SYSTEM_HOST_GET_OBJECT_SYSCALL
[0204]        CALL
[0205]        ;成功
[0206]        PUSH 0
[0207]        PUSH SUCCESS
[0208]        STOP
[0209]  Control.Actions.Play.Perform：
[0210]  Control.Actions.Play.Check：
[0211]  ;检查目标节点是否是可到达的
[0212]  PUSH@TargetNodeId
[0213]  PUSH@IsNodeReachableFunctionNumber
[0214]  PEEK
[0215]  CALL
[0216]  BRN Play_Fail
[0217]  ;把当前时间放入堆栈
[0218]  PUSH@GetTimeStampFunctionNumber
[0219]  PEEK
```

[0220] CALL

[0221] ;检查日期是否在结束日期之前

[0222] DUP ;当前时间

[0223] PUSH@EndDate

[0224] PEEK

[0225] SWAP

[0226] CMP

[0227] BRN Play_Fail

[0228] ;检查日期是否在所述开始日期之后

[0229] ;当前时间处于堆栈上

[0230] PUSH@StartDate

[0231] PEEK

[0232] CMP

[0233] BRN Play_Fail

[0234] ;成功

[0235] PUSH 0

[0236] PUSH SUCCESS

[0237] STOP

[0238] Play_Fail :

[0239] PUSH 0

[0240] PUSH FAILURE

[0241] STOP

[0242] .export Global.OnLoad

[0243] .export Control.Actions.Play.Init

[0244] .export Control.Actions.Play.Check

[0245] .export Control.Actions.Play.Perform

[0246] 在附录 E 中包括了控制程序的附加例子。

[0247] 3. 内容消费和封装应用

[0248] 以下是消费 DRM 保护的内容的应用（例如，媒体播放器、文字处理器、电子邮件客户端等，诸如图 3 中的应用 303a、303c 和 303d）以及诸如应用 303b 之类的封装应用的更详细的说明性实施例，所述封装应用用于封装其目标为消费应用的内容。

[0249] 1.5. 内容消费应用体系结构

[0250] 内容消费应用典型情况下集中于访问受保护的内容，或者可以是通用应用的一部分，所述通用应用还执行诸如封装内容之类的其它功能。在各个实施例中，内容消费应用可以执行以下一些或全部：

[0251] ●提供用户可以用来请求访问受保护的内容对象并且接收关于所述内容的信息或错误信息的接口；

[0252] ●管理与文件系统的交互；

[0253] ●识别受保护内容对象的格式；

[0254] ●请求 DRM 引擎评估对内容的许可以查看访问所述内容的权限是否可以准许；

[0255] ●验证数字签名并且处理 DRM 引擎需要执行的其它通用密码功能；

[0256] ●请求 DRM 引擎提供为解密受保护内容所需要的密钥；和 / 或

[0257] ●解密受保护内容并且与媒体再现服务相交互以便再现所述内容。

[0258] 在一个实施例中，DRM 客户端引擎评估与内容相关联的许可，确认或拒绝用于使用内容的权限，并且向内容消费应用提供解密密钥。DRM 客户端引擎还可以向内容消费应用发布一个或多个义务和 / 或回调，要求该应用执行某些动作，原因在于已经被给与了访问内容的权利。

[0259] 图 13 示出了在一个实施例中构成内容消费客户端应用 1300 的元项。如图 13 所示，主机应用 1302 是客户端的逻辑中心点。它负责驱动在其它模块之间的交互模式以及通过用户接口 1304 与用户的交互。主机应用 1302 经由主机服务接口 1308 向 DRM 引擎 1306 提供一组服务。主机服务接口 1308 允许 DRM 引擎 1306 访问由主机应用 1302 所管理的数据，以及由所述主机应用 1302 执行的某些库函数。在一个实施例中，主机服务接口 1308 是 DRM 引擎 1306 唯一的出站接口。

[0260] 在一个实施例中，DRM 引擎 1306 并不与由主机应用 1302 所管理的多媒体内容直接交互。主机应用 1302 在逻辑上与用于访问多媒体内容的内容服务 1310 相交互，并且只向 DRM 引擎 1306 传递必须由其处理的数据部分。由媒体再现引擎 1312 来执行与内容的其它交互。例如，在一个实施例中，内容服务 1310 负责从媒体服务器获取内容，存储并管理客户端的持久性存储装置上的内容，而媒体再现引擎 1312 是负责访问多媒体内容并且（例如在视频和 / 或音频输出上）再现它的子系统。在一个实施例中，媒体再现引擎 1312 从 DRM 引擎 1306 接收一些信息（诸如内容解密密钥），但是在一个实施例中，所述 DRM 引擎 1306 并不直接与媒体再现引擎 1312 相交互，而是通过主机应用 1302 与之交互。

[0261] DRM 引擎 1306 所需要的一些信息可在多媒体内容带内获得，并且可以通过内容服务 1310 来获取和管理，但是该信息的一些可能需要经由诸如个性化服务或成员资格服务之类的其它服务手段（未示出）来获得。

[0262] 在图 13 所示出的实施例中，由密码服务块 1314 来处理密码运算（例如，加密，签名验证等）。在一个实施例中，DRM 引擎 1306 并不与密码服务块 1314 直接相交互，而是作为替代经由主机 1302（使用主机服务接口 1308）间接地交互，主机 1302 转发它的请求。密码服务 1314 还可以例如由媒体再现引擎 1312 用来执行内容解密。

[0263] 应当理解，提供了图 13 是为了说明而已，并且在其它实施例中在图 13 中所示出的各个组件可以被重新排列、合并、分离、消除，和 / 或可以增加新的组件。例如但不限于，在图 13 中的 DRM 引擎和主机应用之间的逻辑功能划分只是说明一种可能实施例，并且在实际的实现方式中可以作出变化。例如，可以把 DRM 引擎与主机应用整个地或部分地集成。从而应当理解，可以在主机应用和 DRM 引擎之间使用任何适当的功能划分。

[0264] 1.6. 封装器体系结构

[0265] 下面提供了封装引擎可以为用于封装电子内容的主机应用所执行的功能的例子。在实践中，封装应用可以特别地集中于封装，或者可以是在用户系统操作的通用应用的一部分，所述用户系统也访问受保护的内容（在本地或者例如在网络上的其它地方封装的）。

[0266] 在各个实施例中，封装主机应用可以执行以下一些或全部：

[0267] ●提供可以通过其指定内容和许可信息的用户接口；

[0268] ●加密内容；

[0269] ●创建构成许可的 DRM 对象；和 / 或

[0270] ●创建内容对象，所述内容对象包含或引用所述内容并且包含或引用许可

[0271] 图 14 示出了在一个实施例中构成内容封装应用 1400 的元项。DRM 封装引擎 1416 负责封装诸如这里所描述的那些许可（例如，包括诸如控制、控制器、保护器等 DRM 对象的许可）。在一些实施例中，DRM 封装引擎 1416 还可以使元数据与许可相关联以便依照人类可读形式来解释所述许可作什么。

[0272] 在一个实施例中，主机应用 1402 提供了用户接口 1404 并且负责获得诸如内容引用和用户（典型情况下为内容所有者或供应者）所想要执行的动作（例如，把内容绑定给谁，在许可中包括什么内容使用条件等）之类的信息。用户接口 1404 还可以显示关于封装过程的信息，诸如所发出的许可的文本，以及如果发生失败，那么所述失败的原因。在一些实施例中，主机应用 1402 所需要的一些信息可以通过服务访问点 (Service Access Point SAP) 要求使用其它服务，诸如认证或授权服务和 / 或成员资格。从而在一些实施例中，封装应用 1400 和 / 或主机应用 1402 可能需要执行以下一些或全部：

[0273] ●媒体格式服务 1406：在一个实施例中，此元项负责管理诸如代码转换和封装之类的媒体格式操作。它还负责内容加密，内容加密经由内容加密服务模块 1408 来实现。

[0274] ●通用密码服务 1410：在一个实施例中，此元项负责发出 / 验证签名以及加密 / 解密一些数据。对这种操作的请求可以由服务访问点 1414 或 DRM 封装引擎 1416 经由主机服务接口 1412 来发出。

[0275] ●内容加密服务 1408：在一个实施例中，此模块在逻辑上与通用密码服务 1410 相分离，这是因为它并不知道所述应用。它由媒体格式服务在内容封装时利用由 DRM 封装引擎 1416 先前发出的一组密钥来驱动。

[0276] 4. 密钥推导

[0277] 下面描述了密钥推导系统，所述密钥推导系统自然地配有这里所描述的 DRM 引擎和系统体系结构的优选实施例，和 / 或可以在其它情境中使用。在下面部分中的一些例子取自被称为“Scuba（水肺）”的密钥推导系统的优选实施例的参考实现方式。在‘551 申请中描述了另外的实施例。

[0278] 如图 15 所示，在一些实施例中，链路对象 1530a、1530b 除了它们在节点 1500a、1500b、1500c 之间建立关系的主要目的之外还用来分发密钥。如上所述，控制对象可以包含控制程序，所述控制程序可以用来判定用于执行动作的请求是否应当被准许。为了完成这点，控制程序可以检查特定节点是否可经由一链路链到达。这里所描述的密钥推导技术利用此链路链的存在来使密钥分发便于进行，以致可以使所述密钥可用于正在执行控制程序的 DRM 引擎。

[0279] 在一个说明性实施例中，在使用可选密钥分发系统的给定部署中的每个节点对象 1500a、1500b、1500c 具有用于加密内容密钥及其它节点密钥的一组密钥。被创建来用于相同部署中的链路对象 1530a、1530b 包含一些密码数据作为有效载荷，所述密码数据使得当 DRM 引擎处理链路链时能够导出密钥信息。

[0280] 节点和链路依照这种方式携带密钥，假定从节点 A 1500a 到节点 C1500c 的链路链

1530a、1530b，有权访问节点 A 1515a、1525a 的秘密共享密钥的实体（例如，客户端主机应用的 DRM 引擎）也有权访问节点 C 1515c、1525c 的秘密共享密钥。有权访问节点 C 的秘密共享密钥允许该实体访问利用那些密钥加密的任何内容密钥。

[0281] 1.7. 节点、实体和密钥

[0282] 1.7.1. 实体

[0283] 在 DRM 系统的一个实施例中，节点是数据对象，而不是系统中的活动参与者。在此情境下，活动参与者被称作实体。实体的例子是媒体播放器、设备、预订服务、内容封装器等。实体一般使得节点与它们相关联。消费内容的实体使用 DRM 引擎并且管理构成其个性 (personality) 的至少一个节点对象。在一个实施例中，实体被假定有权访问它所管理的节点对象的所有数据，包括那些对象的所有私有信息。

[0284] 1.7.2. 节点

[0285] 参与密钥推导系统的说明性实施例中的节点对象包含密钥，作为它们数据的一部分。在一个实施例中，节点可以包含两种一般类型的密钥 :共享密钥和机密密钥。以下部分列出了可以在各个实施例中使用的不同密钥类型。然而应当理解，具体部署可以只使用这些密钥的一个子集。例如，系统可以被配置为只利用密钥对工作，省略使用秘密对称密钥。或者，系统如果它只需要使用共享密钥的话可以在不向节点提供机密密钥的情况下来加以部署。

[0286] 1.7.2.1. 共享密钥

[0287] 共享密钥是公钥 / 私钥对和 / 或由节点 N 和所有节点 Px 共享的对称密钥，为此从 Px 到 N 存在包含密钥推导扩展的链路。

[0288] 共享公钥 :Kpub-share[N]。这是公钥密码的公钥 / 私钥对的公共部分。此密钥一般与证书一起供给，使得其凭证可以由想要把机密信息用密码绑定到它的实体来验证。

[0289] 共享私钥 :Kpriv-share[N]。这是公钥 / 私钥对的私有部分。管理节点的实体负责确保此私钥是保密的。由于这个原因，此私钥通常被独立于其余的节点信息来存储或传输。此私钥可以通过链路的密钥推导扩展与下游 (downstream) 的其它节点共享。

[0290] 共享对称密钥 :Ks-share[N]。这是以对称密码方式使用的密钥。就像私钥一样，此密钥是机密的，并且管理节点的实体负责保持它是秘密的。此秘密密钥可以通过链路的密钥推导扩展与其它节点下游地共享。

[0291] 1.7.2.2. 机密密钥

[0292] 机密密钥是密钥对和 / 或对称密钥，只为管理它们所属节点的实体所知。在上述这些密钥和共享密钥之间的差异在于它们并不通过链路中的密钥推导扩展而与其它节点共享。

[0293] 机密公钥 :Kpub-conf[N]。这是公钥密码的公钥 / 私钥对的公共部分。此密钥一般与证书一起供给，使得其凭证可以由想要把机密信息用密码绑定到它的实体来验证。

[0294] 机密私钥 :Kpriv-conf[N]。这是公钥 / 私钥对的私有部分。管理节点的实体负责确保此私钥是保密的。由于这个原因，此私钥通常被独立于其余的节点信息来存储或传输。

[0295] 机密对称密钥 :Ks-conf[N]。这是以对称密码方式使用的密钥。就像机密私钥一样，此密钥是被保密的。

[0296] 1.8. 密码元项

[0297] 这里所描述的密钥推导和分发系统的优选实施例可以使用各种不同的密码算法来实现，并且不局限于对密码算法的任何特定选择。尽管如此，对于给定部署或简档来说，所有参与实体通常需要一致同意一组支持的算法（其中术语简档通常指的是在特定实现方式中所使用的一组实际技术的规范（例如，用于密钥推导的 RSA ；用于编码对象的 XML ；用于文件格式的 MP4 等）和 / 或当在实际的部署中定义对象时存在的语义上下文的其它表示）。

[0298] 在一个实施例中，部署包括对至少一个公钥密码（诸如 RSA）和一个对称密钥密码（诸如 AES）进行支持。

[0299] 当涉及密码功能时使用以下符号 ：

[0300] ■ Ep (Kpub[N]，M) 意指“消息 M 使用公钥密码利用节点 N 的公钥 Kpub 来加密”

[0301] ■ Dp(Kpriv[N]，M) 意指“消息 M 使用公钥密码利用节点 N 的私钥 Kpriv 来解密”

[0302] ■ Es(Ks[N]，M) 意指“消息 M 使用对称密钥密码利用节点 N 的对称密钥 Ks 来加密”

[0303] ■ Ds(Ks[N]，M) 意指“消息 M 使用对称密钥密码利用节点 N 的对称密钥 Ks 来解密”

[0304] 1.9. 内容密钥的目标

[0305] 在优选实施例中，使用两种类型的密码目标。把内容密钥的目标定为目标节点的共享密钥意味着使该密钥可用于共享该目标节点的秘密共享密钥的所有实体。把内容密钥的目标定为节点的机密密钥意味着使该密钥只可用于管理该节点的实体。通过使用以下方法中的一个或两个加密在内容密钥对象中所携带的内容密钥 CK 来完成内容密钥的定目标 ：

[0306] ●公共绑定 ：创建包含 Ep(Kpub[N]，CK) 的内容密钥对象

[0307] ●对称绑定 ：创建包含 Es(Ks[N]，CK) 的内容密钥对象

[0308] 在优选实施例中，在可能的情况下使用对称绑定，这是因为它只涉及较小的计算强度算法，由此使得对接收实体来说不那么繁重。然而，创建内容密钥对象的实体（典型情况下为内容封装器）可能不总是有权访问 Ks[N]。如果封装器没有 Ks[N]，那么它可以使用公共绑定，这是由于 Kpub[N] 并不是机密信息由此可用于需要进行公共绑定的实体。通常使 Kpub[N] 可用于需要为附有证书的内容密钥定目标的实体，所述证书可以由所述实体检查以便依照某个商定策略来判定 Kpub[N] 是否确实是可以信任其处理内容密钥的节点的密钥（例如，所述节点对应于运行 DRM 引擎和主机应用的实体，所述 DRM 引擎和主机应用符合系统的功能、操作和安全策略）。

[0309] 1.10. 使用链路的密钥推导

[0310] 为了允许实体有权访问可从其个性节点到达的所有节点的共享密钥，在一个实施例中，链路对象包含可选的密钥扩展有效载荷。此密钥扩展有效载荷允许有权访问链路来源节点的私有 / 秘密密钥的实体也有权访问链路去往节点的私有 / 秘密共享密钥。依照这种方式，实体可以解密其目标为可从其个性节点到达的节点的任何内容密钥（如果使用目标节点的共享密钥来定目标的话）。

[0311] 在一个实施例中，当 DRM 引擎处理链路对象时，它处理每个链路的密钥扩展有效载荷以便更新它有权访问的内部密钥链。在一个实施例中，从节点 F 到节点 T 的链路 L 的

密钥扩展有效载荷包括：

[0312] ●公共推导信息：Ep(Kpub-share[F]，{Ks-share[T]，Kpriv-share[T]} 或

[0313] ●对称推导信息：Es(Ks-share[F]，{Ks-share[T]，Kpriv-share[T]}

[0314] 其中 {Ks-share[T] 和 Kpriv-share[T]} 是包含 Ks-share[T] 和 Kpriv-share[T] 的数据结构。

[0315] 公共推导信息用来向有权访问节点 F 的私有共享密钥 Kpriv-share[F] 传达节点 T 的秘密共享密钥 Ks-share[T] 和 Kpriv-share[T]。

[0316] 对称推导信息用来向有权访问节点 F 的对称共享密钥 Ks-share[F] 传达节点 T 的秘密共享密钥 Ks-share[T] 和 Kpriv-share[T]。

[0317] 对于把内容密钥的目标定到节点来说，在链路中包括的优选有效载荷为对称推导信息。这在链路创建者有权访问 Ks-share[F] 时是可能的。如果不可能的话，那么所述链路创建者会回退到包括公共推导信息来作为链路的有效载荷。

[0318] 假定处理链路的 DRM 引擎已经在其内部密钥链中具有 Ks-share[F] 和 Kpriv-share[F]，那么在处理所述链路之后，L[F → T]，它还将具有 Ks-share[T] 和 Kpriv-share[T]。

[0319] 由于在一个实施例中可以依照任何次序来处理链路，所以在处理给定链路 L 时，DRM 引擎可能无法进行密钥推导计算。这可能是由于以下事实，在那时 DRM 引擎的密钥链可能仍不包含该链路的“来自”节点的密钥。在这种情况下，所述链路被记录，并且当新的信息可用于 DRM 引擎时 - 诸如在处理新的链路 P 之后，再次被处理。如果链路 P 的“去往”节点与链路 L 的“来自”节点相同，并且链路 P 的“来自”节点是可到达节点，那么链路 L 的“来自”节点也是可到达的，并且密钥推导步骤把链路 L 的“来自”节点的私有共享密钥添加到该密钥链。

[0320] 5. 实现方式例子

[0321] 下面提供了几个例子来用于图示在实践中可以怎样应用这里所描述的系统和方法的各个实施例。这里所描述的系统和方法可以实现大范围的权利管理及其它功能，从而应当理解，这里所给出的具体例子并不旨在穷举，而是说明所发明工作主体的范围。

[0322] 1.11. 例子：用户、PC 和设备

[0323] 假定你想要实现把播放内容的权利与特定用户联系在一起的 DRM 系统，并且你想要使用户在他或她所拥有的所有播放设备上播放内容都很容易。假定你决定将向用户提供用于使他们能够按照需要来增加播放设备（例如，移动播放器）的软件。然而，还假定你想要设置某个策略来限制用户可以向其转送内容的通用设备的数目，使得所述用户没有充当分发代理的能力。

[0324] 根据这些系统需求，例如把你创建的许可联系到用户并且在用户和他们所使用的设备之间建立关系可能是有意义的。从而在此例子中，你可以首先决定你需要什么类型的节点来建立你要求的关系种类。例如，你可以定义以下类型的节点：

[0325] ●用户（例如，拥有使用内容权利的个人）

[0326] ● PC（例如，在个人计算机上运行的软件应用，可以播放内容并且指定另外的播放设备）

[0327] ●设备（例如，便携式内容再现设备）

[0328] 每个节点对象可以包括类型属性，用于表明对象是表示用户、PC 还是设备。

[0329] 比如说例如：你决定把在特定时间可以附着到任何一个用户的 PC 节点对象的最大数目限制为四个 (4)。你决定只要你提供对 PC 数目的限制就不必限制被附着到用户的设备数目。据此，控制程序可以被设置为如果能够在用户节点和请求访问的节点之间建立关系那么允许访问。于是该节点可以是 PC 或设备。

[0330] 图 16 示出了被设计用来满足上述要求的系统。服务器 1600 向每个新用户 1604a、1604b 分配用户节点对象 1602a、1602b，并且管理用户 1604a、1604b 为了访问受保护内容的目的而把设备 1606、1608 和 PC1610、1612 与之相关联的能力。当用户 1604a 想要把新的设备 1606 与他或她的用户节点 1602a 相关联时，服务器 1600 确定所述设备 1606 是否已经包含个性化信息 1614，这也许与所述设备 1606 在制造时就被个性化的情况一样。如果该设备的确包含个性化信息 1614，那么服务器 1600 使用该个性化信息 1614 来创建从设备 1606 到用户节点 1602a 的链路 1616，并且向用户设备 1606 发送链路 1616。当用户 1604a 获得受保护的内容 1618 时（例如，从服务器 1600 或从其它内容供应者），该内容 1618 其目标在于用户节点 1602a（例如，通过利用与用户节点 1602a 相关联的一个秘密共享密钥来加密内容的解密密钥）并且使许可 1619 与其相关联，指定能够访问内容的条件。当用户 1604a 试图在设备 1606 上播放内容 1618 时，在设备 1606 上运行的 DRM 引擎 1620 评估许可 1619，所述许可 1619 表明只要用户节点 1602a 是可到达的那么就可以播放所述内容 1618。DRM 引擎 1620 评估链路 1616（所述链路 1616 示出了用户节点 1602a 可从设备 1606 到达），并且例如通过授权解密在许可 1619 内所包含的内容解密密钥，来准许用户 1604a 对访问内容 1618 的请求，。

[0331] 由于在此例子中，使用与用户节点 1602a 相关联的秘密密钥来加密内容解密密钥，所以需要获得此秘密密钥以便解密所述内容解密密钥。如果已经使用在此的其它地方所描述的可选密钥推导技术，那么可以通过使用设备 1606 的秘密密钥之一解密在链路 1616 中所包含的密钥推导信息来简单地获得用户节点密钥。所解密的密钥推导信息包含用于解密在许可 1619 中所包含的内容解密密钥所需要的密钥（或者能够从其导出或获得密钥的信息）。

[0332] 再次参照图 16，假定用户 1604a 想要使新的 PC 1610 与他或她的用户节点 1602a 相关联。服务器 1600 验证尚没有把最大数目的 PC 与用户节点 1602a 相关联，并且授权 PC 1610 与用户节点 1602a 相关联。然而为了执行所述关联，服务器 1600 需要从 PC 1610 获得个性化信息（例如，密码密钥、唯一标识符等）。然而如果 PC 1610 先前尚未被个性化（这可能与用户只是下载 PC 软件拷贝的情况相同），那么服务器 1600 会执行个性化过程（例如，通过使用在此的其它地方所描述的引导协议来创建 PC 节点对象）或者把用户引向可以执行个性化过程的服务供应商。当完成个性化过程时，服务器 1600 可以创建从 PC 1610 到用户节点 1602a 的链路 1624 并且向 PC 1610 发送所述链路，只要该链路保持有效，那么所述 PC 1610 可以继续使用该链路。

[0333] 用户稍后可以请求增加另外的 PC，并且服务器可能会强制实施用于把每个用户的 PC 节点对象的数目限制为 4 的策略（典型情况下，它可能还会向用户提供按照需要从其活动列表中移除 PC 的能力）。

[0334] 作为又一例子，现在假定服务供应商已经决定用户应当能够在他们拥有的任何设

备上播放他们拥有的任何内容。服务供应商还可能想要允许用户的 PC 软件创建到他或她每个设备的链路，而并不要求所述用户联系服务器 1600。在这种实施例中，当用户想要在新的设备上播放内容时，用户的 PC 软件可能会访问新的设备机密个性化信息并且使用它来创建用于该设备的新链路（例如，从所述新的设备到用户节点 1602a 的链路）。如果所述设备未被个性化，那么 PC 软件有权访问远程服务，或者指示设备访问所述远程服务来执行个性化过程。然后 PC 软件会向新设备发送链路，在该点，只要内容保持有效，那么新设备就能播放所述内容，这是由于在一个实施例中一旦链路对象存在，就不必在创建另一个，除非所述链路对象期满或被无效。

[0335] 在上面所示出的例子中，内容的目标在于用户。为此，封装器应用为内容选择新的 ID，或者使用现有的，创建加密密钥和相关联的内容密钥对象，以及用于绑定内容对象和内容密钥对象的保护器对象。然后封装器利用控制程序（例如，用 DRM 引擎的虚拟机可执行的字节代码编译的）来创建控制对象，用于当且仅当用户节点可从请求“播放”动作的 PC 或设备节点到达时才允许进行所述“播放”动作。典型情况下，如果适当的话，控制、控制器、保护器和内容密钥对象被嵌入到封装的内容中，使得 PC 和设备不必分别获得它们。

[0336] 在一个实施例中，当设备或 PC 想要播放内容时，它遵循诸如先前结合图 9 所描述的过程。即，DRM 引擎找到用于所述内容的内容 ID 的保护器对象，然后是由保护器所引用的内容密钥对象，然后是引用该内容密钥对象的控制器对象，并且最后是由该控制器引用的控制对象。DRM 引擎执行控制对象的控制程序，所述控制程序检查所述用户节点是否是可到达的。如果设备或 PC 节点具有必要的链路对象来验证在其节点和用户节点之间存在路径，那么满足条件并且控制程序允许使用在内容密钥对象中所表示的密钥。然后设备或 PC 的媒体再现引擎可以解密并播放内容。

[0337] 1.12. 例子：临时登录

[0338] 图 17 是这里所描述的 DRM 系统和方法的潜在应用的另一例子。此例子类似于在先前部分中的例子，除了这里用于管理支配在 PC 节点对象和用户节点对象之间创建链路对象的策略允许只是 12 小时的临时登录，只要用户尚未在另一 PC 上临时登录的话。此特征会允许用户 1700 把他的内容 1702 带到朋友的 PC 1704，登录该 PC 1704 一段时间，并且在朋友的 PC 1704 上播放所述内容 1702。

[0339] 为了实现这点，可能在有限有效期的情况下创建链路对象 1710。在一个实施例中，这可以如下进行：

[0340] 为了便于解释，假定用于播放 DRM 保护的内容 1702 所要求的具有 DRM 功能的消费软件 1714 已经存在于朋友的 PC 1704 上。包含内容 1702 和许可 1708 的文件被转送到朋友的 PC 1704。当用户试图播放内容 1702 时，软件 1714 认识到并不存在用于把本地 PC 节点与拥有所述内容的用户节点相链接的有效链路对象。软件 1714 提示用户需要他的凭证 1712（这可以经由用户名 / 密码、移动电话认证协议、智能卡或在系统策略下所允许的任何认证系统来提供）并且与后端系统 1706 通信。后端系统 1706 检查所请求链路的用户节点对象和 PC 节点对象的属性，并且检查到并不存在仍然有效的活动临时登录链路对象。如果满足那些条件，那么后端服务 1706 创建用于链接朋友的 PC 节点对象和用户节点的链路对象 1710，其有效期限于所请求的登录持续时间（例如小于 12 小时，以便符合此例子中的策略）。现在具有链路对象 1710 使朋友的 PC 1704 能够播放用户的内容 1702 直到链路 1710

期满。

[0341] 1.13. 例子 :企业内容管理

[0342] 图 18 示出了用于管理企业文档（例如，电子邮件、字处理文档、展示幻灯片、即时消息发送文本等）的说明性系统 1800 的高级体系结构。在图 18 所示出的例子中，文档编辑应用（例如文字处理器）1802、电子邮件客户端 1804 和目录服务器（例如，活动目录服务器）1806 利用数字权利管理 (DRM) 插件 1808、网络服务组织层 1810、注册服务 1812 和策略服务 1816 来便于依照策略管理文档、电子邮件消息等。在优选实施例中，使用在此的其它地方以及‘551 申请中所描述的 DRM 引擎和服务组织技术来实现 DRM 插件 1808、网络服务组织层 1810、策略服务 1816 和注册服务 1812。例如在一个实施例中，DRM 插件 1808 可以包括上述 DRM 引擎的实施例。应当理解，虽然图 18 示出了其中经由应用可以调用的插件来把诸如文字处理器 1802 和电子邮件客户端 1804 之类的现有应用与 DRM 引擎集成的实施例，但是在其它实施例中所述 DRM 引擎可以作为应用本身任意或者两者的组成部分来包括。还应当理解在图 18 中所示出的说明性系统可以在单个企业内实现或者可以跨越多个企业实现。

[0343] 在图 18 所示出的图解中，目录服务器 1806 例如可以包含用户简档和组定义。例如，被称作“特别工程队 (special project team)”的组可以被公司系统管理员设置来标识公司特别工程队的成员。

[0344] 在一个实施例中，目录服务器 1806 可以包括用于运行诸如在‘551 申请中所描述（并且例如利用在Windows®平台上基于标准 IIS 技术实现）的那些 web 网络服务的活动目录服务器，所述网络服务根据所访问的内容向特别工程队组中的人们发出节点、链路和许可。如果在组中成员资格改变，那么可以发出新的令牌。对于权利撤销来说，目录服务器 1806 可以根据诸如在‘551 申请中所描述的技术（这里有时被称作为“NEMO”技术）来运行安全元数据服务。在一些实施例中，可以要求客户端具有到期时间值或时间概念（根据公司选择定义的任何新鲜值（例如，1 周、1 天、1 小时、每 5 分钟等））以便使用 DRM 许可。例如，安全元数据服务提供的令牌可以包括信任和认证的时间值。在一些实施例中，客户端可以在安全元数据服务交互中标识用户节点 ID。可以在许可控制情境中直接评估安全元数据以便确定用户是否仍然具有给定的成员资格。安全元数据还可以返回代理，所述代理可以确定诸如作为特别工程队中成员的关系是否有效。从而在一些实施例中，可以利用只是添加几个明确定义的网络服务来平衡公司现有的授权和认证基础结构（例如，公司的活动目录服务器）。

[0345] 图 19 示出了诸如在图 18 中所示出的系统可以怎样用来管理对文档的访问或其它使用的例子。在此例子中，特定的雇员（约翰）可能频繁致力于高度机密的战略工程，并且可能已经安装了 DRM 插件 1908 来用于其应用（例如，字处理程序 1902、电子邮件程序 1904、日程表程序、集成这种程序的程序或程序套件等）。在创建他文档期间的某个时间点，约翰访问已经被添加到其应用工具条的“权限”下拉菜单项（动作 1913）。出现权限对话框，用于联系他公司的活动目录服务器 1906 以便查找已经在系统上设置的个人和组的目录。他从列表中选择“特别工程队”，并且选择向队中的每个人给予查看、编辑和打印文档的权限。使用在‘551 申请中所描述的 NEMO 服务组织技术，DRM 插件 1908 把具有 NEMO 功能的策略服务扩展 1916 联系到活动目录 1906 并且请求要用来保护特别工程队的文件的策略的拷贝

（动作 1914）。当约翰保存文档时，DRM 插件自动加密文件 1912，并且创建许可对象 1910，其目标在于并且绑定到被称为“特别工程队”的组。许可 1910 允许任何设备访问（例如，查看、编辑、打印等）文件 1912，其中所述设备可以生成从其设备节点到特别工程队组节点的有效链路链。

[0346] 约翰有权访问文档 1912，这是因为他的设备具有到约翰用户节点的链路，并且它还具有从约翰的用户节点到“特别工程队”组节点的链路。同样，如果他把此文档转发给其它人，那么只有在他们能够生成到“特别工程队”组节点的有效链路链才能访问它（例如，通过要求特别工程队节点可由该设备到达）。

[0347] 约翰可以把（已经受保护的）文件保存在他的计算机上，并且稍后把它附到电子邮件消息上（动作 1920）。例如，他可以打开给他老板（乔治）的旧电子邮件，像通常那样附加文件，并且发送消息。如图 20 所示，乔治还在他的计算机 2014 上安装有 DRM 插件 2000。当他登录到他的计算机 2014 时，插件 2000 有机会检查他已经添加的所有组（动作 2006），并且下载新组，刷新已经期满的任何链路（动作 2012）。如果他自其上次登录而已经被添加到“特别工程队”，那么他的插件 2000 能会下载用于把他的用户节点链接到“特别工程队”组节点的链路对象 2008。此链路 2008 意味着用户节点“乔治”是组节点“特别工程队”的成员。在此例子中，假定链路对象 2008 具有截止日期（例如 3 天），在此之后该链路对象 2008 不再有效。

[0348] 如图 21 所示，当乔治试图打开文档时（动作 2130，2132），DRM 插件 2108 检查所嵌入的（或附加的）许可，并且得知“特别工程队”节点必须是可到达的。他的插件 2108 构造（并验证）从他计算机设备节点到用户节点“乔治”以及从用户节点“乔治”到组节点“特别工程队”的链路链 2120、2122（动作 2134）。由于设备具有有效的链路链 2120、2122，所以他的插件 2108 允许访问文件。

[0349] 如在此的其它地方所描述，在一些实施例中，链路还可以携带安全的密钥链。从而在一些实施例中，通过生成到特别工程队节点的链路链，插件不仅可以证明它被允许访问内容，而且还能够解密密钥链，所述密钥链使其能够解密所述内容。

[0350] 例如如果另一雇员（“卡罗尔”）意外地接收约翰的电子邮件，并且试图打开文档，那么她的 DRM 插件会获取与所述文件绑定的许可并且评估所述许可的条款。她的 PC 具有到她用户节点“卡罗尔”的链路；但是由于她不是该团队的成员，所以不存在从“卡罗尔”到“特别工程队”组节点的链路。由于“特别工程队”是不可到达的，所以不允许她访问文件。

[0351] 如果卡罗尔最终被添加到组“特别工程队”。在下次她的 DRM 插件刷新她的成员资格时，会检测此新组，并且下载用于把她的用户节点链接到特别工程队节点的链路对象。现在她的插件具有需要用来构造从她的设备节点到她的用户节点再到特别工程队节点的链所需要的所有链路。现在特别工程队节点“是可到达的”并且她可以打开目标为特别工程队的任何文档或电子邮件——即便是在她加入该团队之前所创建的那些文档或电子邮件。

[0352] 假定一个月后，乔治转换为新角色并且被从活动目录中的特别工程队组中移除。下次乔治登录时，他的插件不接收新的、刷新的链路对象，其中所述链路对象把他的用户节点“乔治”关联到“特别工程队”。当数周以后他再试图打开约翰的文件时，他的插件试图构造到特别工程队的链路链。他的 PC 仍然具有到用户节点“乔治”的链路（乔治的 PC 仍然属于他）；但是从“乔治”到“特别工程队”的链路已经期满。由于“特别工程队”是不可到

达的，所以不允许他访问文件。

[0353] 假定公司具有这样的策略，所述策略要求对所有的机密信息的访问被记录为日志。在一个这种实施例中，用于特别工程队的策略指示为此组所创建的所有许可也需要要求收集使用信息并把它报告给例如中央储存库。从而在此例子中，当评估（例如，执行）许可中的控制程序时，插件执行把访问记录为日志的要求并且记录所述访问为日志。例如，结果活动可以被记录在诸如这里所描述的本地保护的状态数据库中作为日志，并且当重新建立网络连接时，可以经由先前描述的服务来报告相关内容。

[0354] 图 22 示出了用于管理企业内的电子内容的另一说明性系统 2200。在图 22 所示出的例子中，LDAP 服务器 2206 用来管理用户简档、组定义和角色分配，并且包含被称作“特别工程队”的组定义以及“代理人”的角色定义。

[0355] 假定约翰是代理人并且想要向特别工程队的其它成员发送具有附件的电子邮件。当约翰安装用于他应用的 DRM 插件程序 2208 时，它还把项目安装到他的电子邮件工具条。在他创作电子邮件消息期间的某个点，约翰从下拉菜单访问已被添加到他工具条的“设置权限”。DRM 插件程序 2208 联系策略服务 2216 并且显示要从中进行选择的公司消息发送策略列表。约翰选择“特别工程 DRM 模版”并且 DRM 插件程序 2208 使用 NEMO 协议来请求并确保它接收的策略对象的可靠性、完整性和机密性。所述策略描述了应当怎样创建使用此模板的许可，包括怎样对它们定目标和绑定。

[0356] 当约翰击中“发送”时，DRM 插件 2208 加密消息和附件，并且产生相关联的许可。该许可要求为了访问电子邮件或附件，特别工程队组节点或“代理人”组节点必须是可到达的。

[0357] 把许可与加密的消息有效载荷和加密的附件相绑定。随后使用标准的电子邮件功能来向接收者列表发送所述消息。由于许可规则和加密不取决于电子邮件的寻址，所以可能错误地包括不正确的电子邮件接收者的事实不会使电子邮件或附件的内容置于风险之中。

[0358] 由于这种无意的接收者没有用于把他的用户节点链接到特别工程队的有效链路对象，所以如果或者当他试图访问内容时不允许他这样做。此外，由于他的设备没有必要的链路链（以及它们包含的密钥），所以他的设备也不具有解密所述内容的能力。

[0359] 然而，如果无意的接收者随后使用标准的电子邮件功能把相同的、未经修改的电子邮件转送到特别工程队的成员。该成员具有用于把他的用户节点链接到“特别工程队”组节点的链路对象，并且能够访问电子邮件的内容。

[0360] 假定公司的另一代理人（“比尔”）也接收到用于把他与“特别工程队”组节点相关联的链路对象。比尔还可以查看所述文件。如果他把消息转送给律师助手（“特伦特”），他既不是代理人也不与特别工程队相关联，那么特伦特没有用于把他与“特别工程队”组节点连接的链路对象，并且他不能访问所述文档。

[0361] 如果特伦特随后被添加到 LDAP 目录 2206 中的特别工程队组中，那么他会被给予必要的链路对象并且将能访问先前转发的电子邮件。

[0362] 如果如先前所论述，公司具有用于表明在所有许可中包括报告要求的策略，那么在一个实施例中，每当执行这些许可之一内的控制程序时（例如当某人试图访问所述文件时），可以触发报告事件。报告步骤另外可以包括关于访问是被准许还是拒绝的指示符——

这是实现方式选择的问题。如果使用这种指示符，那么可以维护尝试访问特定文档的次数以及每次的状态和其它信息（例如，成功、失败等）的日志。

[0363] 作为又一例子，假定特别工程队的成员之一（“斯蒂芬”）出差到另一公司来致力于该特别工程。在前往该另一公司公司之前，斯蒂芬的电子邮件客户端把所有电子邮件的本地拷贝下载到他的收件箱中。被附着到这些电子邮件之一的受保护报告也包括嵌入式（或附加）的许可。此许可对象包括用于访问内容以及访问加密的内容密钥的规则。访问内容所要求的唯一“缺少的链路”是用于到达“特别工程队”组节点所必要的链路对象。

[0364] 由于在此例子中公司策略在于允许链路对象保持有效达 3 天，所以用于把斯蒂芬的用户节点链接到特别工程队节点的链路对象将在他出差和断开连接时保持有效。如果他试图在离线时访问文件，那么特别工程队组节点仍然是可到达的，并且允许他访问所述文件。

[0365] 然而如果斯蒂芬保持离线达三天以上，那么用于把他链接到特别工程队的链路对象会期满。特别工程队组节点于是不再可到达，并且不允许他访问所述文件。

[0366] 如果斯蒂芬最终出差到他可以（例如经由 VPN）连接到公司系统的位置，那么他的 DRM 插件会向他所属的每个组请求所刷新的链路对象拷贝。由于他仍然是“特别工程队”组的一部分，所以他接收从他的用户节点到特别工程队组节点的新链路对象。此链路替换已经期满并不再有效的“旧”链路。

[0367] 由于“特别工程队”节点现在可使用此新刷新的链路到达，所以他能够再次访问受保护的报告。新的链路对象在 3 天时间内是有效的，在此之后所述链路对象也期满。

[0368] 作为又一例子，假定特别工程队的成员“萨莉”想要经由即时信使与另一队员通信，保存通信拷贝，并且把它给予团队的另一成员（例如，经由电子邮件附件、盘片、软件狗 (dongle) 等）。在此例子中，即时信使客户端（以及可能还有公司向其雇员所提供的任何其它消息发送和通信产品）被链接到 DRM 插件，和在先前例子中一样，所述 DRM 插件访问策略“特别工程 DRM 模板”，所述策略“特别工程 DRM 模板”规定将怎样为许可定目标及绑定。当萨莉试图保存她的即时消息发送会话时（例如，通过选择“另存为”），该插件选择加密密钥（例如，随机地）并且封装（加密）会话的文本。依照公司策略，然后 DRM 插件产生其目标在于并且被绑定到特别工程队组节点的许可对象。

[0369] 把包含受保护 IM 副本 (transcript) 的文件与所述许可绑定以便访问副本内容。和在先前的例子中一样，许可包含用于管理支配对内容以及所加密的内容密钥拷贝进行访问的规则。萨莉可以使用标准的‘拖放’过程来把此绑定的文件转送到电子邮件、USB 软件狗、盘片等，并且把它发送给其他人。如果接收方设备可以生成到特别工程组节点的有效链路，那么允许并且可以访问内容。

[0370] 假定萨莉把文件给了约翰，约翰也是特别工程队的成员。如果约翰具有最近刷新的链路对象，所述链路对象把他标识为特别工程队的成员，那么他能够访问所述文件。按照公司策略，此链路对象包含会导致它在三天后期满的截止日期。因此，即便约翰保持断开连接，只要该链路保持有效他仍然有权访问。

[0371] 如果在以后的某一时间约翰离开特别工程队而从事另一分配的工作，并且在他的袋子中发现来自萨莉的 USB 软件狗并且试图使用他的台式计算机打开文件，那么用于把他的用户节点关联到特别工程队的链路对象将已经期满。由于他不再是团队的一部分，所以

他设备上的 DRM 插件不再可以获取新的、刷新的链路。由于“特别工程队”组节点不再可通过他的设备到达，所以不允许访问。

[0372] 假定自他改变了工作以来他的膝上计算机一直没有被连接到网络，他还试图利用该设备打开所述文件。由于最大分配的时间已经过去，所以该链路也不再有效。在一些实施例中，每当他试图访问所述文件时，可以产生报告并对其排队以发送到中央储存库。

[0373] 中央储存库经由电子邮件接收多次试图访问文件未成功的报告并且标记管理器。管理器提醒约翰不再允许他访问机密材料并且请求毁掉所有文件（即便系统表明该访问尚未被准许也是如此）。

[0374] 作为又一例子，假定政府机构或外部审计员想要调查或审计特别工程队对机密信息的处理。为了支持调查，公司想要表明审计记录以供访问与特别工程相关的敏感信息。

[0375] 为此，公司首先扫描用于与特别工程相关的任何消息的所有明文消息存档。使他们欣慰的是，他们发现在坚持公司策略的情况下，在没有适当 DRM 保护的情况下，没有雇员发送论述特别工程的消息（例如向系统之外发送消息）。

[0376] 然后公司使用 DRM 访问记录来生成详细描述向谁以及何时给予对受保护信息的访问权的审计跟踪细节。

[0377] 按照公司规程，当建立特别工程队组时，默认时它还包括首席服从官 (Chief Compliance Officer CCO)。用于首席服从官的链路对象被创建并保存到存档服务器，所述存档服务器允许他或她如果将来需要的话回顾所有消息的内容。

[0378] 在此例子中，为特别工程队所定义的策略表明该团队所产生的所有许可必须包括用于报告任何试图对所述文件进行访问的要求，包括日期和时间、用户节点以及访问是否被准许。这些报告被保存在中央储存库上的访问日志中。

[0379] CCO 检查在已经出现可疑的任何泄漏或舞弊的日期之前与特别工程队相关联的所有访问的访问日志。CCO 还搜索电子邮件、IM 和网络备份存档以查找在该日期或之前的所有消息通信业务和系统文件。由于每个文件具有附加的许可（具有内容密钥），并且 CCO 具有必要的链路对象来满足所述许可的要求，所以允许他或她访问在所关注时间之前所访问的每一个消息的内容。

[0380] 使访问日志和未加密的消息内容完全可用于代理机构 / 审计员作为调查的一部分。

[0381] 在一些实施例中，特别工程队的策略还可以包括用于为与特别工程相关的所有许可设置截止日期的要求。例如，如果只是法定要求公司保持此天然记录达 1 年，那么它们可以在策略中表明许可在发出日之后一年期满。在该情况下，公司可以只是按照法律的要求来保持记录。在该时间之后，即便 CCO 也无访问权。

[0382] 在上面论述中，偶尔会引用“定目标”和“绑定”。在优选实施例中，定目标和绑定表示两种不同的、却紧密相关的过程。在优选实施例中，“绑定”主要是密码过程，涉及保护用于加密内容的密钥。当许可被‘绑定’到节点（例如“特别工程队”节点）时，例如它可以意指利用与该节点相关联的公钥来加密内容密钥。从而，只有有权访问节点私钥的设备具有必要的密钥来解密该内容（并且在优选实施例中，获得访问节点私钥权利的唯一方式是解密到该节点的链路链）；然而，简单地具有正确私钥只表明所述设备如果被允许这样做的话具有解密该内容的能力。

[0383] 在优选实施例中，由许可内的控制程序来确定是否允许设备访问内容，并且特别地是，确定怎样来给它定目标。“定目标”指的是在控制程序中增加用于指定特定节点（或多个节点）“是可到达的”以便使用内容的要求。在上面所示出的例子中，控制程序典型情况下指定特定的节点“特别工程队”可由消费设备到达。

[0384] 在一些情况下，可能希望使许可目标为一个以上节点，诸如在公司的新产品开发团队（“公司”），其与多个供应商合作来投标用于新绝密产品的组件。假定在工程早期阶段期间，供应商 A 和供应商 B（竞争者）都具有到“SecretProjectX”的链路。供应商 A 想要与 SecretProjectX 的所有成员共享其思想，但是不希望他们把它无意间泄漏给供应商 B。供应商 A 可以为这些许可定目标以致：（“SecretProjectX 是可到达的”）并且（“供应商 A 是可到达的”或者“公司是可到达的”）。如果公司无意间把此信息共享给秘密工程 X 中的每个人（包括供应商 B），那么不允许在供应商 B 的那些人查看它，限制对公司的任何非公开的风险并且消除供应商 A 丢失其商业秘密的可能。

[0385] 1.14. 例子：保健记录

[0386] 图 23 图示了可以怎样应用这里所描述的系统和方法来管理保健记录。假定医疗记录具有不同的机密级别，而且希望对系统中不同的实体（例如，病人、医生、保险公司等）准许不同的访问权利。例如，可能希望只允许某些记录被病人查看，只允许某些记录被病人的医生查看，允许某些记录可被病人查看但是只可由病人的医生编辑，允许某些记录可被所有医生查看，允许某些记录被所有保险公司查看，允许某些记录只可被病人的保险公司查看等。

[0387] 如图 23 所示，可以使用像诸如在此的其它地方所描述的那些节点和链路式的 DRM 对象来建模此保健生态系统 2300。例如，可以把节点分配给病人 2302、病人的医生 2304、病人的保险公司 2306、病人的设备 (2308，2310)、病人的一个特定医生 2312、医生的计算设备 2314、2316、所有医生组 2318、某个专业的医生组 2320、医疗机构 2322、保险公司 2324、由保险公司所使用的计算设备 2326、所有保险公司组 2328 等。

[0388] 假定病人的医生使用他或她的 PC 来创建关于病人的医疗记录。例如，医疗记录可以包括具有用于他或她的附注、诊断、处方指令、所述病人的指令等的多个字段的文档模板。所述模板还可以允许医生选择用于管理支配文档和 / 或其个人字段的安全策略。例如，医生的应用可以给出一组标准的安全策略选择，并且当获得医生的选择时，可以根据那些选择来自动地产生许可并且与医疗记录受保护的（例如，加密的）内容相关联。

[0389] 为了此例子的目的，假定许可准许查看对病人、对治疗所述病人的所有保健提供者以及对向所述病人提供保险的所有保险公司的访问。进一步假定，为了图示说明，所述许可只向医疗机构 x 的心脏病专家准许编辑权利。

[0390] 封装应用接受医生的策略规范输入（其可以简单地包括标准模板上的鼠标点击）并且产生许可，所述许可包括诸如下面所示出的控制程序：

[0391]

```
        Action.Edit.Perform() {
            if (IsNodeReachable("MedicalFoundationX") &&
                IsNodeReachable("Cardiologist")) {
                return new ESB(ACTION_GRANTED);
            } else {
                return new ESB(ACTION_DENIED);
            }
        }

        Action.View.Perform() {
            if (IsNodeReachable("PatientY") ||
                IsNodeReachable("HCPsPatientY") ||
                IsNodeReachable("ICsPatientY") {
                return new ESB(ACTION_GRANTED);
            } else if (EmergencyException == TRUE) {
```

[0392]

```
            return new ESB(ACTION_GRANTED, new
NotificationObligation());  }
        else {
            return new ESB(ACTION_DENIED);
        }
    }
```

[0393] 然后可以把医疗记录及其相关联的许可存储在医疗记录的中央数据库、由特定医学基金会所操作的数据库等中。如果病人 Y 随后拜访另一保健提供者，并且授权保健提供者作为他认可的保健提供者之一（例如通过签名授权表单），该保健提供者会获得到病人 y 认可的保健提供者节点的链路，所述保健提供者可能会把它存储在他的计算机系统上。如果该保健提供者然后获得由医生 x 所创建的医疗记录，那么他能够获得对该医疗记录的查看访问权，这是由于病人 y 认可的保健提供者节点可从新的保健提供者计算机系统到达。如果另一方面未经认可的保健提供者获得（加密的）医疗记录的拷贝，那么他可能不能访问它，这是因为没有任何一个所要求的节点（即，病人 y 的节点、所有病人 y 认可的保健提供者的节点以及所有病人 y 认可的保险公司的节点）可从其计算系统到达。

[0394] 然而注意，上面所示出的示例性控制程序包括重载特征，所述重载特征例如在紧急情况下可以被调用，例如如果保健提供者需要访问受保护的医疗记录，但是不能满足控制程序的条件（例如，因为试图紧急访问医疗记录的保健提供者先前尚未被注册为病人 Y 的保健提供者）。然而还要注意，紧急访问异常的调用会导致关于调用和 / 或其它环境的信

息被自动记录，并且在此例子中还导致发送通知（例如，向病人优选的保健提供者——即，由病人明确授权的实体－和／或病人自己）。这种义务与紧急异常的关联可以阻止对异常的滥用，这是因为可能会存在滥用的记录。

[0395] 应当理解，已经提供此示例性程序以便于解释这里所描述的系统和方法的某些实施例。例如，系统是否包括对紧急异常的支持一般取决于系统设计者的要求和愿望。从而例如一些实施例可以不支持紧急异常，其它实施例可以支持紧急异常，但是把可以调用这种异常的实体类限制为‘所有医生’类（例如，通过要求 EmergencyException 标志被设置为“真”并且所有医生节点都是可到达的），并且其它实施例仍然可以支持紧急异常，但是不把强制性义务与之相关联（由于不能符合所述义务在优选实施例中可能会使内容不可访问），作为替代依赖于用于实施的非技术性的、法律或制度手段（例如，通过信任保健提供者不会滥用调用异常的能力，和／或依赖工业证明和法律体系来防止滥用）。

[0396] 可以对上面所提供的例子进行的又一变化可能要求更有力的证明，即医生或具体姓名的医生实际上是访问医疗记录的人，而不是坐在医生用来访问记录的计算机（并且从而潜在地包含为满足可达性分析所必须的链路的计算机）边上的其他人。可以依照任何适当的方式来强制实施这种更有力的认证形式。例如，通过使用口令、软件狗、生物测定标识机制等来保护用于访问医疗记录的医生的计算机和／或软件，可以在应用或系统级整个地或部分地强制实施这点。作为选择或另外，与某些医疗记录相关联的控制程序本身可以包括要求这种更有力标识的义务或条件，诸如检查软件狗的存在，要求主机去获得口令等。

[0397] 1.15. 例子：预订

[0398] 图 24 是在电子预订服务的情境中可以怎样使用这里所给出的系统和方法的图解。例如假定用户（爱丽丝）想要从因特网服务供应商 (XYZISP) 获得对爵士音乐的预订。因特网服务供应商可以提供各种不同的预订选项，包括免费的试订，但是只可以用来在期满之前播放预订内容五次（例如，通过播放一首歌曲五次，通过播放五首不同的歌曲各一次等）。试订还使内容只可用于略微变差的形式（例如，下降的保真度或分辨率）。爱丽丝使用她的个人计算机来访问服务供应商的因特网网点，并且选择试订。然后服务供应商发出链路对象 2400 和代理 2401 并且把它们发送到爱丽丝的个人计算机 2406。代理 2401 可操作来初始化爱丽丝安全状态数据库中的状态，所述状态将用来跟踪爱丽丝已经使用试用内容的次数。链路 2400 是从爱丽丝的帐户节点 (Alice@XYZ ISP)2402 到预订节点 2404 并且包括控制程序，当爱丽丝请求播放内容时，所述控制程序检查由代理 2401 所设置的状态变量的当前值以便看看是否允许额外的播放。

[0399] 当爱丽丝把内容下载到她的 PC 并且试图播放它时，她的 PC 上的 DRM 引擎评估与所述内容相关联的许可，其表明为播放所述内容，预订节点 2404 必须是可到达的。爱丽丝先前已经向她的 ISP 注册了她的 PC，此时她接收了从她的 PC 节点 2406 到她的帐户节点 2402 的链路 2405。从而 DRM 引擎拥有用于把 PC 节点 2406 连接到预订节点 2404 的链路对象 2405、2400；然而在准许爱丽丝播放内容的请求之前，DRM 引擎首先通过执行所述链路包含的任何控制程序来确定所述链路是否有效。当执行链路 2400 中的控制程序时，DRM 引擎检查状态数据库条目以便确定是否已经进行了 5 次播放，并且如果尚未到达 5 次，那么准许她播放所述内容的请求，并且还向主机应用发出义务。所述义务要求主机在再现之前使内容降级。主机应用确定它能够满足此义务，并且继续再现内容。为了使爱丽丝能够在相对于

她五次免费试用播放来计数该内容之前预览内容，控制程序还可以包括回调，所述回调例如在已经准许用于播放内容的请求之后检查 20 秒以便看看所述内容是否仍然在播放。如果所述内容仍然被播放，那么减少播放计数，否则不减少。从而，爱丽丝可以从由预定服务所提供的任何内容项中选择，并且在她试订期满之前播放它们中的五个。

[0400] 一旦爱丽丝试订期满，爱丽丝就决定购买完全的按月预订，这使她能够播放与她希望按月付费一样多的内容项。爱丽丝使用她的 PC 来对预订签名，并且接收从她的帐户节点 2402 到预订节点 2404 的链路 2410。所述链路包括用于表明所述链路只有效一个月的控制程序（例如，所述控制程序检查状态数据库中的条目以便看看自从发出链路以来是否已经过去一个月）。此链路 2410 被连同代理程序一起发送到爱丽丝的 PC，所述代理程序可操作来初始化 PC 的 DRM 引擎的状态数据库中的适当条目，所述条目表明发出链路的日期。当爱丽丝从预订服务下载内容并且试图播放它时，她 PC 的 DRM 引擎确定到预订节点的路径由链路 2405、2410 组成。DRM 引擎执行在链路 2405、2410 中所包含的任何控制程序来确定链路是否有效。如果自从发出链路 2410 以来还没有过去一个月，那么链路 2410 中的控制程序返回用于表明链路 2410 仍然有效的结果以及爱丽丝播放所述内容的请求。如果爱丽丝试图播放她先前在她免费试用期间所获得的内容，那么她 PC 上的 DRM 引擎将执行相同的分析并且准许她的请求。由于与在试用期间所获得的内容相关联的许可表明如果安全数据库中的 TrialState 变量未被设置，所以唯一条件在于预订节点必须是可到达的，现在爱丽丝可以再次访问该内容，这是因为所述预订节点此时可经由链路 2410 而不是不再有效的链路 2400 来从爱丽丝的 PC 再次到达。从而，爱丽丝不必获得内容项的第二拷贝来替换她在免费试用提供期间所获得的拷贝。类似地，如果爱丽丝从她的朋友鲍勃获得预订内容，所述鲍勃也是相同服务的订户，那么在此例子中爱丽丝也能够播放该内容，这是因为内容许可只是要求预订节点是可到达的，而并不要求它可经由鲍勃的 PC 或帐户到达。

[0401] 应当理解，以上例子只是旨在图示可以由这里所描述的系统和方法启用的一些功能，并且并不旨在建议必须依照上述精确的方式来实现所述预定。例如在其它实施例中，与预订内容相关联的许可可以被绑定到用户节点而不是预订节点，从而防止两个订户共享内容，像在上述例子中鲍勃和爱丽丝那样。应当理解，可以对以上例子进行许多其它变化。

[0402] 下表提供了用于上述例子中的代理、链路和许可控制程序的一些说明性伪码：

[0403]

==========

预订试用允许你访问多达5个预订内容。内容只在再现20秒之后才被标记为已再现。在试用情境下再现的内容必须被再现应用降级。

真正预订每月更新并且对再现的次数或质量没有这种限制。

代理的代码如下：

==========

```
TrialAgent() {
    SetObject("TrialState", 5);
}
```

==========

试用链路中的控制程序代码为：

==========

```
Control.Link.Constraint.Check() {
```

[0404]

```
        if (GetObject("TrialState", 5) > 0) {
            return SUCCESS;
        } else {
            return FAILURE;
        }
    }
    ==========
```

当爱丽丝真的注册预订服务时，她取回链路（从：爱丽丝，到：预订）和代理

代理的代码如下：

```
    ==========
    RealSubscriptionAgent() {
        // 如果存在的话,那么擦除TrialState
        trialState = GetObject("TrialState");
        if (trialState != NULL) {
            SetObject("TrialState", NULL);    // 擦除
        }
    }
    ==========
```

链路的代码为：

```
    ==========
    Control.Link.Constraint.Check() {
        if (GetTrustedTime() < ExpirationDate) {
            return SUCCESS;
        } else {
            return FAILURE;
        }
    }
    ==========
```

[0405]

```
目标在于预订的内容许可都具有相同的控制程序:
==========

Action.Play.Perform() {
    // 首先检查预订节点是否是可到达的
    if (!IsNodeReachable("SubscriptionNode")) {
        return new ESB(ACTION_DENIED);
    }

    // 现在检查是否存在TrialState
    if (GetObject("TrialState) != NULL) {
        // 我们处于试用模式中: 我们需要回调和义务
        return new ESB(ACTION_GRANTED,
            new OnTimeElapsedCallback(20, DecrementCounter),
            new DegradeRenderingObligation());
    } else {
      // 我们处于付费预定模式: 只是返回ACTION_GRANTED
        return new ESB(ACTION_GRANTED);
    }
}

// OnTimeElapsed的回调功能的代码
DecrementCounter() {
    SetObject("TrialState",   GetObject("TrialState")-1)  ;       }
==========
```

[0406] 再次参照图 24,爱丽丝在她的移动服务供应商下还具有帐户 2420,只要她保持连接到网络,所述移动服务供应商就保持有效。不要求爱丽丝对预订进行特别付款,以换取她获取发送链路 ;作为替代当她连接到网络时更新链路 2424 被自动地发送到她的电话。这些链路使她能够访问由移动服务供应商所提供的任何内容项或服务,所述内容项或服务具有只要求预订节点 2422 是可到达的许可。如果爱丽丝改变移动服务供应商,那么一旦她的链路 2424 期满,她就不能访问先前获取的内容。

[0407] 图25示出了服务供应商可以怎样与家庭网络域2500相交互的例子。在此例子中，设备被注册到家庭网络域，所述家庭网络域强制实施允许多达5个设备在任一时刻属于该域的策略。尽管史密斯家庭的电缆服务供应商没有提供用于设置家庭网络域2500的域管理器软件，不过电缆服务供应商知道域管理器已经由所证实的家庭网络域管理器软件的提供者实现，从而信任域管理器软件来按照打算那样操作。如图25所示，史密斯家庭把爱丽丝的电话和PC、卡罗尔的PVR和乔的PSP连接到域2500，导致从这些设备中的每个向域节点2500发出链路。当例如在PVR接收新的内容时，诸如在‘551申请中所描述的那些发现服务使域中的其它设备能够自动地获得内容和任何必要的链路。从域节点2500向服务供应商帐户节点2502发出链路。一些电缆服务供应商的内容具有许可，所述许可的义务是必须禁止快进和回倒使得广告将被观看。卡罗尔的PVR和PC、爱丽丝的PC能够强制实施所述义务，从而可以播放内容。爱丽丝的移动电话不能强制实施所述义务并且从而拒绝对内容进行访问。

[0408] 1.16. 附加例子：内容和权利共享

[0409] 如先前例子所图示，这里所给出的系统和方法的实施例使得能够依照自然方式来共享电子内容。例如，这里所描述的系统和方法可以用来使消费者能够与他们的朋友和家庭成员共享娱乐内容，和/或在所有他们的家庭设备上欣赏它，同时防止过宽的、未经授权的分发。例如，可以使用自动化对等发现和通知服务，以致当一个设备获得内容或相关联权利时，其它设备可以自动地知道该内容，由此提供可以自动更新的虚拟分布式库。例如在一个实施例中，如果一个用户在一个位置在便携式设备上获得内容或权利，然后回家，那么用户的家庭设备可以自动地发现并利用那些权利。相反地，如果用户在他或她的家庭网络的设备上获得权利，那么他或她的便携式设备可以发现并带走该内容以便在其它地方使用。这里所描述的系统和方法的优选实施例可以用来创建服务和权利对象，所述服务和权利对象允许例如使用在‘551申请中所描述的服务发现和检查技术来使上述方案完全地自动化。例如，被注册到特定域的设备可以彼此提供服务（例如，权利和内容的共享），和/或可以调用远程服务以便于本地内容共享。所描述的系统和方法能够创建这样的DRM架构，所述DRM架构不集中在防止创建拷贝本身，而是被设计成用于利用网络技术协调地工作以便允许共享内容，同时防止消费者变为内容的非法分发者。

[0410] 这里所描述的DRM系统和方法的优选实施例还能够在没有其它DRM系统的冗长类型的权利表达特征的情况下确定权利。作为替代，优选实施例使用可以上下文交互的一组精巧的(crafted)权利对象。这些对象描述了在诸如用户、设备、内容及其组之类的实体之间的关系和控制。例如，这种上下文交互可以允许设备确定可以播放给定内容，这是因为(a)内容从用户目前预订的合法内容服务获得，(b)所述用户是特定家庭组的一部分，并且(c)所述设备与此特定家庭组相关联。存在诸如在此例子中所描述的那些多种类型的关系，用户可以直观地理解所述关系，并且这里所描述的系统和方法的优选实施例能够创建自然理解这种关系的系统。在实体之间的关系可以随时间创建、销毁和改变，并且优选实施例提供了用于在动态联网环境中——消费者可以自然理解的环境——确定权利的自然方式。尽管如此，如果内容部署者想要使用更传统的权利表达方式，那么优选实施例也可以适应该方式。例如，可以使用工具来把传统的权利表达转换为诸如上述的那些对象组，和/或可以实现用于直接对这种权利表达操作的DRM引擎。作为选择，在一些实施例中，设备不必理解

这种传统的权利表达，并且不受它们的局限性的约束。

[0411] 这里所描述系统和方法的优选实施例还具有媒体服务的十分一般的概念。广播服务和因特网下载或预订服务是媒体服务的例子。与这些服务相关联的限制可以使内容难于共享。利用这里所描述的系统和方法的优选实施例，内容可以在广播、宽带和移动服务上获得，并且在包括便携式设备的家里的一组网络设备上共享。作为选择或另外，可以由单个设备经由无线连接依照对等方式来提供服务。例如，具有新一代 WiFi 功能的手机可以向其它设备提供内容范畴服务。这种服务允许其它设备‘看看’什么内容可用来与所述设备共享。所述服务提供了能够用来确定权利以使得可以接受或容易地消除任何限制的信息。

[0412] 这里所描述的系统和方法的优选实施例不限于一个服务或一个平台。如上所述，优选实施例能够与包括‘个人服务’的很多服务一起工作。随着家庭和个人网络变得更加普遍，这变得越来越重要。例如，数字照相机现在可与 WiFi 连接一起使用，使得经由网络共享照片非常方便。能够使照片共享自动化很好，但是照相机因为被四处携带而遇到许多不同的网络。自动化共享是便利的，但是个人照片当然是个人的。这里所描述的系统和方法的实施例使得在家庭内在家庭设备上共享照片很容易，而不与碰巧在网络上遇到所述照相机的任意设备共享。通常，随着更多设备变得可联网，管理在那些设备上的所有内容的权利变得越来越重要。尽管联网目的在于允许联网的设备上的信息被共享，不过网络会彼此重叠和合并。网络使内容能够容易地被共享，但是该内容不应当被任意地共享。从而，希望具有这样的 DRM 系统，其能够觉察到网络并且可以使用由内容、用户、网络和设备特征所提供的上下文来确定是否应当共享并且怎样来共享内容。这里所描述的系统和方法的优选实施例能够实现这种方法。

[0413] 6. 用于内容消费和封装的参考体系结构

[0414] 下面描述了用于消费受 DRM 保护的内容的消费应用（例如，媒体播放器）和用于封装目标为消费应用的内容的封装应用（例如，位于服务器上的应用）的参考体系结构。

[0415] 1.17. 客户端体系结构

[0416] 下面提供了 DRM 引擎的说明性实施例可以为用于消费内容的主机应用而执行的功能的例子。

[0417] 1.17.1. 主机应用到 DRM 引擎的接口

[0418] 尽管在优选实施例中 DRM 引擎并不要求 API，不过下面是在一个说明性实施例中由说明性 DRM 引擎（被称为‘章鱼’DRM 引擎）向主机应用所提供的接口类型的高级描述：

[0419] Octopus::CreateSession(hostContextObj ect) → Session- 给定主机应用上下文，来创建会话。上下文对象由章鱼 DRM 引擎用来向应用进行回调。

[0420] Session::ProcessObject(drmObject)- 此函数当主机应用遇到媒体文件中的某些类型的对象时应当被主机应用调用，所述媒体文件可以被标识为属于 DRM 子系统。这种对象包括内容控制程序、成员资格令牌等。那些对象的语法和语义对主机应用来说是不透明的。

[0421] Session::OpenContent(contentReference) → Content——主机应用当需要与多媒体内容文件相交互时调用此函数。DRM 引擎返回内容对象，所述内容对象随后可以用来获取关于所述内容的 DRM 信息并且与之相交互。

[0422] Content::GetDrmInfo()- 返回关于内容的 DRM 元数据，所述 DRM 元数据另外在所

述文件的常规元数据中是不可得的。Content::CreateAction(actionInfo) → Action- 主机应用当想要与内容对象相交互时调用此函数。actionInfo 参数指定应用需要执行什么类型的动作（例如，播放）以及如果必要的话（任何相关联的参数）。所述函数返回动作对象，所述动作对象然后可以用来执行所述动作并且获取内容密钥。

[0423] Action::GetKeyInfo()- 返回为解密子系统解密所述内容所必须的信息。

[0424] Action::Check()- 检查 DRM 子系统是否授权执行此动作（即 Action::Perform() 是否会成功）。

[0425] Action::Perform()- 执行所述动作，并且按照管理支配此动作的规则所指定来执行任何结果（具有它们的副作用）。

[0426] 1.17.2.DRM 引擎到主机服务的接口

[0427] 下面是由 DRM 引擎的说明性实施例从主机应用的说明性实施例所需要的主机服务接口类型的例子。

[0428] HostContext::GetFileSystem(type) → FileSystem- 返回 DRM 子系统具有独占访问权的虚拟 FileSystem（文件系统）对象。此虚拟 FileSystem 用来存储 DRM 状态信息。此 FileSystem 内的数据应当只能由 DRM 子系统读取和写入。

[0429] HostContext::GetCurrentTime()- 返回由主机系统维护的当前日期 / 时间。

[0430] HostContext::GetIdentity()- 返回此主机的唯一 ID。

[0431] HostContext::ProcessObject(dataObject)- 向主机服务返回数据对象，所述数据对象可能已经被嵌入在 DRM 对象内，但是 DRM 子系统已经被标识为由主机（例如，证书）管理。

[0432] HostContext::VerifySignature(signatureInfo)- 检查数据对象上的数字签名的有效性。在一个实施例中，signatureInfo 对象包含等效于在 XMLSig 元项中所找到信息的信息。主机服务负责管理为验证所述签名所必需的密钥和密钥证书。

[0433] HostContext::CreateCipher(cipherType,keyInfo) → Cipher- 创建 DRM 子系统可以用来加密并解密数据的密码对象。最小的一组密码类型将被定义，并且每种类型定义有用于描述由密码实现方式所要求的密钥信息的格式。

[0434] Cipher::Encrypt(data)

[0435] Cipher::Decrypt(data)

[0436] HostContext::CreateDigester(digesterType) → Digester- 创建所述 DRM 子系统可以用来对一些数据计算安全散列的摘要对象。在一个实施例中，可以定义最小的一组摘要类型。

[0437] Digester::Update(data)

[0438] Digester::GetDigest()

[0439] 1.17.3.UML 顺序图

[0440] 图 26 图示了在先前部分中所阐明的说明性 API 的使用以及在示例性实施例中在主机应用和 DRM 客户端引擎之间进行的交互。

[0441] 1.18. 封装器参考体系结构

[0442] 下面提供了封装引擎可以为用于封装的内容的主机应用所执行的功能的例子。在实践中，封装应用可以特别集中于封装，或者可以是在用户系统操作的通用应用的一部分，

所述通用应用也访问受保护的内容（在本地或者例如在网络上的其它地方封装的）。

[0443] 1.18.1. 主机应用到封装引擎的接口

[0444] 本部分提供了在主机应用和结合被称为“章鱼”的参考 DRM 引擎所使用的封装引擎之间的说明性 API 的高级描述。

[0445] Octopus::CreateSession(hostContextObject) → Session。给定主机应用上下文来创建会话。由此函数所返回的上下文对象由封装引擎用来向应用进行回调。

[0446] Session::CreateContent(contentReferences[]) → Content。主机应用调用此函数以便创建将在随后的步骤中与许可对象相关联的内容对象。在 contentReferences（内容引用）数组中具有一个以上内容引用意味着这些在一捆中被绑定在一起（例如，一个音频和一个视频曲目）而且所发出的许可应当目标在于这些引用，作为一个不可分的组。

[0447] Content::SetDrmInfo(drmInfo)。drmInfo 参数指定将发出的许可的元数据。drmInfo 充当用于把许可转换为用于虚拟机的字节码的指南。

[0448] Content::GetDRMObjects(format) → drmObjects。当主机应用准备获取封装器引擎所创建的 drmObjects 时，此函数被调用。格式参数表明这些对象所希望的格式（例如，XML 或二进制原子）。Content::GetKeys() → keys[]。此函数当主机封装应用需要用于加密内容的密钥时被主机封装应用调用。在一个实施例中，每个内容引用存在一个密钥。

[0449] 1.18.2. 封装引擎到主机服务的接口

[0450] 下面是在一个实施例中说明性封装引擎需要主机应用提供的接口类型的例子。

[0451] HostContext::GetFileSystem(type) → FileSystem。返回 DRM 子系统具有独占访问权的虚拟 FileSystem 对象。此虚拟 FileSystem 可以用来存储 DRM 状态信息。此 FileSystem 内的数据应当只能由 DRM 子系统读取和写入。

[0452] HostContext::GetCurrentTime() → Time。返回由主机系统维护的当前日期 / 时间。

[0453] HostContext::GetIdentity() → ID。返回此主机的唯一 ID。

[0454] HostContext::PerformSignature(signatureInfo, data)。由封装引擎所创建的一些 DRM 对象必须被信任。由主机所提供的此服务用来为所指定的对象签名。

[0455] HostContext::CreateCipher(cipherType,keyInfo) → Cipher。创建封装引擎可以用来加密和解密数据的密码对象（能够加密和解密数据的对象）。在一个实施例中，密码对象用来加密在内容密钥对象中的内容密钥数据。

[0456] Cipher::Encrypt(data)。加密数据。

[0457] Cipher::Decrypt(data)。解密数据。

[0458] HostContext::CreateDigester(digesterType) → Digester。创建可以由封装引擎用来对一些数据计算散列的摘要对象。

[0459] Djgester::Update(data)。把数据馈送到摘要对象。

[0460] Digester::GetDigest()。计算摘要。

[0461] HostContext::GenerateRandomNumber()。产生可以用于产生密钥的随机数。

[0462] 图 27 是用于示出上面所阐明的说明性 API 的使用以及在一个说明性实施例中在主机应用和封装引擎之间进行的交互的例子的 UML 图。

[0463] 7. 对象

[0464] 此部分提供了关于 DRM 对象的更多信息，所述 DRM 对象充当 DRM 引擎的说明性实现方式的构建块。首先，给出了由 DRM 引擎用于内容保护和管理支配的对象类型的相对高级的概观。接下来，它们表达的这些对象和信息的更详细描述连同在一个说明性实施例中所使用的一些示例性数据结构一起被提供了。

[0465] 1.19. 内容保护和管理支配 DRM 对象

[0466] 如先前结合图 6 所描述，内容管理支配对象（有时与节点和链路对象一起被称作为‘DRM 对象’）用来把使用规则和条件与受保护的内容相关联。这些对象一起形成了许可。

[0467] 如图 6 所示，使用密钥来加密由内容对象 614 所表示的数据。解密所述内容所需要的该密钥由内容密钥对象 602 表示，并且在所述内容和用于加密该内容的密钥之间的绑定由保护器对象 604 来表示。用于管理支配对解密密钥的使用的规则由控制对象 608 来表示，并且在内容密钥 602 和控制对象 608 之间的绑定由控制器对象 606 来表示。在一个实施例中，受信系统只在由控制对象 608 中的字节代码所表达的规则的管理支配下利用内容解密密钥。图 28A 是诸如在图 6 中所示出的该许可的更详细图解，并且图示了在一个实施例中所使用的签名方案。

[0468] 1.19.1. 公用元项

[0469] 在一个实施例中，对象共享通用的基本特性：它们都可以具有 ID、属性列表和扩展列表。

[0470] 1.19.1.1. ID

[0471] 由其它对象所引用的对象具有唯一的 ID。在一个实施例中，ID 简单地是 URI，并且按照惯例，那些 URI 为 URN。

[0472] 1.19.1.2. 属性

[0473] 属性是有类型的值。属性可以是有名或无名的。有名属性的名称是简单的字符串或 URI。属性值具有简单类型（字符串，整数或字节数组）或复合类型（列表和数组）。类型‘列表’的属性包含有名属性的无序列表。类型‘数组’的属性包含无名属性的有序数组。

[0474] 对象的‘属性’字段是有名属性的无序集合（可能是空的）。

[0475] 1.19.1.3. 扩展

[0476] 扩展是可以被添加到对象以便携带可选的或强制性的额外数据的元项。扩展是有类型的，并且也具有唯一的 ID。扩展可以是内部或外部的。

[0477] 1.19.1.3.1. 内部扩展

[0478] 内部扩展包含在它们扩展的对象中。它们具有‘关键’标志，用于表明是否要求所述扩展的特定扩展数据类型为使用对象的实现方式所知。在一个实施例中，如果实现方式遇到具有关键扩展的对象，其中所述关键扩展具有它并不理解的数据类型，那么所述实现方式必须拒绝整个对象。

[0479] 在一个实施例中，内部扩展的 ID 需要是本地唯一的：对象无法包含具有相同 ID 的两个扩展，但是两个不同的对象都可以包含具有与另一对象的扩展相同的 ID 的扩展。

[0480] 对象的‘扩展’字段是有内部扩展的无序集合（可能是空的）。

[0481] 1.19.1.3.2. 外部扩展

[0482] 外部扩展并不包含在它们扩展的对象中。它们独立于对象出现，并且具有‘主题’字段，所述‘主题’字段包含它们扩展的对象的 ID。在一个实施例中，需要外部扩展的 ID 是

全局唯一的。

[0483] 1.19.2. 内容

[0484] 在一个实施例中，内容对象是‘外部’对象。其格式和存储不受 DRM 引擎的控制，但是受主机应用的内容管理子系统的控制（例如，所述内容可以是 MP4 电影文件、MP3 音乐曲目等）。在一个实施例中，内容的格式需要向把 ID 与内容有效载荷数据相关联提供支持。依照格式相关的方式来加密内容有效载荷（一般利用对称密码，诸如 AES）。

[0485] 1.19.3. 内容密钥

[0486] 内容密钥对象表示唯一的加密密钥，并且把 ID 与之相关联。ID 的目的在于使保护器对象和控制器对象能够引用内容密钥对象。在内容密钥对象中所封装的实际密钥数据自身被加密，使得它只可以由被授权来解密所述内容的接收者读取。内容密钥对象指定使用哪个密码系统来加密密钥数据。用于保护内容密钥数据的密码系统被称作密钥分发系统。可以使用不同的密钥分发系统。密钥分发系统的例子是上述 Scuba 密钥分发系统。

[0487] 1.19.4. 保护器

[0488] 保护器对象包含可以找出使用哪个密钥来加密内容对象的数据的信息。它还包含关于使用哪个加密算法来加密该数据的信息。在一个实施例中，保护器对象包含作为内容对象引用的一个或多个 ID 以及正好包含表示用于加密数据的密钥的内容密钥对象引用的一个 ID。如果保护器指向一个以上的内容对象，那么所有那些内容对象表示已经使用相同的加密算法和相同的密钥而加密的数据。在一个实施例中，除非所使用的密码系统允许一种对不同的数据项使用相同密钥的安全方式，否则并不建议保护器对象指向一个以上的内容对象。

[0489] 1.19.5. 控制

[0490] 控制对象包含用于允许 DRM 引擎判定当对内容的某些动作被主机应用请求时所述动作是否应当被允许的信息。在一个实施例中，用于管理支配对内容密钥使用的规则在控制对象中被编码为供虚拟机执行的字节代码。控制对象还具有唯一的 ID 使得它可以由控制器对象引用。在一个实施例中，控制对象被签名，使得 DRM 引擎可以验证所述控制字节代码在用来判定之前是有效的并被信任。选择性地，还可以通过验证在控制器对象中所包括的安全散列来导出控制对象的有效性。

[0491] 1.19.6. 控制器

[0492] 控制器对象包含用于允许 DRM 引擎找出哪个控制来管理支配对由内容密钥对象所表示的一个或多个密钥的使用的信息。控制器对象包含用于把它绑定到它引用的控制对象和内容密钥对象的信息。在一个实施例中，控制器对象被签名（例如，被具有用于允许它签名控制器对象的证书的封装应用签名），使得可以确立在内容密钥和管理支配它的控制对象之间的绑定的有效性以及在内容密钥 ID 和实际密钥数据之间的绑定的有效性。控制器对象的签名可以是公钥签名或对称密钥签名或两者的结合。当在控制器对象中包括由控制器对象所引用的控制对象摘要时，还可以在不必独立地验证控制对象的签名的情况下推导控制对象的有效性。

[0493] 1.19.6.1. 对称密钥签名

[0494] 在一个实施例中，这是控制器对象的优选类型签名，并且通过计算控制器对象的消息认证代码 (Message Authentication Code MAC) 来实现，利用与由相应的内容密钥对

象所表示的密钥相同的密钥来加密钥。在一个实施例中，用于此 MAC 的规范方法是利用与为在相同部署中所使用的 PKI 签名算法所选择的相同散列算法来使用 HMAC。

[0495] 1.19.6.2. 公钥签名

[0496] 当需要知道控制器对象的签名人身份时使用此类签名。此类签名利用公钥签名算法来实现，利用断言此对象有效性的委托人 (principal) 的私钥来签名。在一个实施例中，当使用此类签名时，对称密钥签名也存在，并且对控制器对象以及公钥签名进行签名，使得可以保证利用其私钥签名的委托人也知道在内容密钥对象中所携带的内容密钥的实际值。

[0497] 1.20. 身份和密钥管理 DRM 对象

[0498] 如前所述，节点对象表示 DRM 简档中的实体，并且不使用隐式的或显式的语义来定义节点对象表示什么。系统的给定部署 (DRM 简档）定义了存在什么类型的委托人，并且不同的节点对象表示什么角色和身份。典型情况下使用节点对象的属性来表达该语义信息。

[0499] 链路对象表示在节点之间的关系。选择性地，链路对象还可以包含一些密码数据，所述密码数据允许所述链路用于内容密钥推导。仅仅对于节点来说，在一个实施例中，不使用隐式的或显式的语义来定义链路关系意指什么。取决于链路的来自和去往节点在给定 DRM 简档中表示什么，链路关系的意义可以表达成员资格、所有权、关联性和 / 或许多其它类型的关系。在典型的 DRM 简档中，一些节点对象可以表示用户，其它节点可以表示设备，并且其它节点可以表示用户组或授权域 (authorized domains AD)。在这种情境中，在设备和用户之间的链路可以表示所有权关系，并且在用户和用户组或授权域之间的链路可以表示成员资格关系。图 28B 图示了在一个示例性实施例中在节点和链路之间的结构和相互关系。

[0500] 1.20.1. 节点

[0501] 节点对象表示系统中的实体。节点对象的属性定义了节点对象所表示的某些方面，诸如在 DRM 简档情境中由节点对象所表示的角色或身份。节点对象还可以具有用于把机密信息的目标定为子系统的机密非对称密钥对，所述子系统有权访问所述节点对象的机密部分（典型情况下，由节点所表示的实体或负责管理该节点的某个实体）。可以利用节点的机密公钥来加密目标在该节点的机密信息。节点对象还可以具有共享非对称密钥对，并且共享对称密钥当系统使用内容密钥推导系统来用于内容密钥分发时能够被结合链路对象使用，所述内容密钥推导系统诸如在此的其它地方所描述。在优选实施例中，只有需要被链路或控制对象引用或者接收密码目标的信息的实体需要具有相应的节点对象。

[0502] 1.20.2. 链路

[0503] 链路对象是签名的声明，在其顶点为节点对象的图中存在有向边。对于一组给定节点和链路来说，我们认为如果在图中的节点 X 顶点和节点 Y 顶点之间存在有向路径，那么在节点 X 和节点 Y 之间存在路径。当在节点 X 和节点 Y 之间存在路径时，我们认为节点 Y 可从节点 X 到达。使用由链路对象所表示的声明来表达哪些节点可从其它节点到达。用于管理支配内容对象的控制可以在它们允许执行动作之前检查某些节点可从与用于执行所述动作的实体相关联的节点到达。例如，如果节点 D 表示想要对内容对象执行‘播放’动作的设备，那么用于管理支配所述内容对象的控制可以测试用于表示某一用户的某一节点 U 是否可从节点 D 到达。为了确定节点 U 是否是可到达的，DRM 引擎可以检查是否存在可以在

节点 D 和节点 U 之间建立路径的一组链路对象。

[0504] 在一个实施例中，DRM 引擎在它使用链路对象来判定在节点图中存在路径之前验证所述链路对象。取决于用于对链路对象进行签名的证书系统（例如，x509v3）的特定特征，链路对象可以被给予有限的使用期、被撤销等。在一个实施例中，用于管理支配哪些密钥可以对链路对象签名、可以创建哪些链路对象以及链路对象的使用期的策略并不直接由 DRM 引擎来处理。作为替代，那些策略平衡节点的属性信息。为了使强制实施某些策略的任务便于进行，在一个实施例中，提供了一种用于利用附加约束检查来扩展标准的证书格式的方式。这些扩展使得可以表达对用于为链路签名的密钥的证书的有效性约束，以致可以在链路被认为有效之前检查诸如所述链路连接什么类型节点的约束以及其它属性。

[0505] 在一个实施例中，链路对象可以包含用来约束链路有效性的控制对象。另外，在一个实施例中，链路对象可以包含密码密钥推导数据，密码密钥推导数据用于向用户提供用于密钥分发的共享密钥。该密码数据除元数据之外包含“来自”节点的私有和 / 或对称共享密钥，其利用“去往”节点的共享公钥和 / 或共享对称密钥来加密。

[0506] 1.21. 数据结构

[0507] 以下段落更详细地描述了上述对象的说明性对象模型，用于定义在一个说明性实施例中每种对象所具有的字段。使用相对简单的对象描述语法来描述数据结构。每种对象类型由可以扩展父的类来定义类（这是“是 -(is-a)”关系）。类描述是采用简单的抽象类型‘string’（字符串）、‘int’（整数值）、‘byte’(8 位值）和‘boolean’（真或假）的术语，但是对于那些数据类型或者包含那些类型的复合结构不定义任何特定编码。编码或表示对象的方式可以根据引擎的实现方式改变。在实践中，DRM 引擎的使用的给定简档可以指定怎样表示字段（例如，使用 XML 模式）。

[0508] 在一个说明性实施例中，使用以下符号：

[0509]

class ClassName {
field1;
field2;
....
}
	定义类的类型。类的类型是不均匀的复合数据类型（也称作对象类型）。此复合类型由一个或多个字段组成，每个字段为简单或复合类型。每个字段可以具有不同类型。

type[]	定义均匀的复合数据类型（也称作列表或数组类型）。此复合类型由 0 或更多相同类型的元项组成（当列表为空时是 0）。
String	简单类型：表示字符串
Int	简单类型：表示整数值
Byte	简单类型：表示在 0 和 255 之间的整数值
Boolean	简单类型：表示布尔值（真或假）
class SubClass extends SuperClass {...}	定义用于扩展另一个类类型的类类型。扩展另一个类的类除其所拥有的字段之外还包含它所扩展的类（称作超类）的所有字段。
Abstract class {...}	定义抽象类的类型。抽象类类型是可以扩展的类型，但是从不被自身使用。
{type field; }	定义可选字段。可选字段是可以从包含它的复合数据类型中省略的字段。
(type field;)	定义当计算封闭复合字段的规范字节序列时跳过的字段
class SubClass extends SuperClass(field=value) {...}	定义类类型的子类并且为该子类的所有实例指定该子类，超类的某些字段值始终等于固定值。

[0510] 1.21.1. 公用结构

[0511] 在一个说明性实施例中，使用以下公用结构：

[0512]

```
abstract class Octobject {
        {string id;  }
        Attribute[] attributes;
        InternalExtension[] extensions;
}

class Transform {
        string algorithm;
}

class Digest {
        Transform[] transforms;
        string algorithm;
        byte[] value;
}
```

[0513]

```
class Reference {
        string id;
        {Digest digest;  }
}
```

[0514] 1.21.1.1. 属性

[0515] 在一个实施例中，存在四种属性：IntegerAttribute、StringAttribute、ByteArrayAttribute 和 ListAttribute，每种属性具有名字和类型。

[0516]

```
abstract class Attribute {
        {string name;  }
        string type;
}

class IntegerAttribute extends Attribute(type='int') {
        int value;
}

class StringAttribute extends Attribute(type='string') {
        string value;
}

class ByteArrayAttribute extends Attribute(type='bytes') {
        byte[] value;
}

Class ListAttribute extends Attribute(type='list') {
        Attribute[] attributes;    // 必须都有名称
}

Class ArrayAttribute extends Attribute(type='array') {
        Attribute[] attributes;    // 必须都没有名称
}
```

[0517] 1.21.1.2. 扩展

[0518] 在所讨论的说明性实施例中，存在两种类型的扩展：在 Octobject 内携带的内部扩展和在 Octobject 外携带的外部扩展。

[0519]

```
abstract class ExtensionData {
        string type;
}

abstract class Extension {
        string id;
}

class ExternalExtension extends Extension {
        string subject;
        ExtensionData data;
}

class InternalExtension extends Extension {
        boolean critical;
        {Digest dataDigest;  }
        (ExtensionData data;  )
}
```

[0520] 在一些实施例中，即便给定实现方式并不理解特定类型的 ExtensionData，也能够验证对象的签名，这将是重要的。从而在一个实施例中，添加与 dataDigest 字段的间接级别。如果 ExtensionData 的规范要求所述数据是特定对象的上下文内签名的一部分，那么 dataDigest 字段将存在。然后，理解此 ExtensionData 并且因此能够计算其规范表示的实现方式可以验证所述摘要。如果在这种实施例中此 ExtensionData 的规范要求所述数据不是签名的一部分，那么 dataDigest 字段将不存在。

[0521] 1.21.2. 节点对象

[0522]

```
        class Node extends Octobject {
        }
```

[0523] 1.21.3. 链路对象

[0524]

```
class Link extends Octobject {
    string fromId;
    string toId;
    {Control control; }
}
```

[0525] 1.21.4. 控制对象

[0526]

```
class Control extends Octobject {
    string protocol;
    string type;
    byte[] codeModule;
}
```

[0527] 1.21.5. 内容密钥对象

[0528]

```
abstract class Key {
    string id;
    string usage;
    string format;
    byte[] data;
}

abstract class PairedKey extends Key {
    string pairId;
}
class ContentKey extends Octobject {
    Key secretKey;
}
```

[0529] 在一个实施例中，每个密钥具有唯一的 id、格式、使用（可以为空）和数据。‘使用’字段如果不为空的话，那么指定所述密钥可以用于做什么。对于正常内容密钥来说，此字段为空。在使用诸如上述的密钥分发方案的实施例中，此字段可以指定这是共享密钥还是机密密钥。‘格式’字段指定‘数据’字段的格式（诸如对于对称密钥来说为‘RAW’，或者对于 RSA 私钥来说为‘PKCS#8’等）。‘数据’字段包含依照‘格式’字段格式化的实际密钥

数据。

[0530] 对于作为密钥对一部分的密钥（诸如 RSA 密钥）来说，额外的字段‘pairId’给出了用于该对的唯一标识符，使得可以从其它数据结构引用该对。

[0531] 在一个实施例中，密钥对象中的数据字段是实际密钥的明文值（即，它是将被散列的密钥的明文值），即便该对象的实际表示包含该密钥的加密拷贝也是如此。

[0532] 1.21.6. 控制器对象

[0533]

```
class Controller extends Octobject {
    Reference controlRef;
    Reference[] contentKeyRefs;
}
```

[0534] 8. 虚拟机

[0535] 这里所描述 DRM 引擎的优选实施例使用虚拟机（这里有时被称作为“控制虚拟机”、“控制 VM”或简单地为“VM”）来执行用于管理支配对内容进行访问的控制程序。下面描述这种虚拟机的说明性实施例，可以对此说明性实施例进行各种修改和设计考虑。还描述了把虚拟机（被称为“Plankton（浮游生物）”虚拟机）的说明性实施例与 DRM 引擎（被称为“章鱼”）的说明性实施例相集成。然而应当理解，这里所描述的数字权利管理引擎、体系结构及其它系统和方法的实施例可以以任何适当的虚拟机的方式来使用，或者在一些实施例中，根本没有虚拟机，从而应当理解，下面所提供的关于虚拟机示例性实施例的细节只是为了说明而并非是限制。

[0536] 在优选实施例中，控制 VM 是传统的虚拟机，被设计成用于易于使用具有很小代码脚印（footprint）的各种编程语言来实现。它是基于简单的、面向堆栈的指令集，其被设计成是最低限度的，而不会过度考虑执行速度或代码密度。在要求简洁代码的情况下，可以使用数据压缩技术来压缩虚拟机的字节代码。

[0537] 在优选实施例中，控制虚拟机被设计成用于适于作为用于低或高级程序设计语言的目标，并且支持汇编语言，C 和 FORTH。另外应当理解，可以依照相对简单的方式创建用于诸如 Java 或客户语言之类的其它语言的编译器来把代码编译为由虚拟机所使用的格式（例如，字节代码）。在一个实施例中，控制虚拟机被设计成用于在主机环境内被主控，而不是在处理器或硅中直接运行。在优选实施例中，用于虚拟机的自然主机环境是 DRM 引擎，不过应当理解，这里所描述的虚拟机体系结构作为选择或另外可以在其它情境中使用。

[0538] 图 29 图示了控制虚拟机 2902 的说明性实现方式的操作环境。如图 29 所示，在一个实施例中，虚拟机 2902 在其主机环境 2904 的情境下运行，当虚拟机执行程序 2906 时所述主机环境 2904 实现了所述虚拟机所需要的一些功能。典型情况下，控制 VM 在 DRM 引擎 2908 内运行，所述 DRM 引擎 2908 实现其主机环境。如图 29 所示，在优选的数据库中，虚拟机 2902 和 DRM 引擎 2908 有权访问安全数据库 2910 以便持久地存储状态信息。

[0539] 1.22. 体系结构

[0540] 1.22.1. 执行模型

[0541] 在优选实施例中，VM 通过执行在代码模块中用字节代码所存储的指令来运行程

序。一些指令可以通过进行系统调用来调用在程序本身之外实现的功能。系统调用可以由VM本身来实现或被委托给主机环境来实现。

[0542] 在一个实施例中，VM执行在代码模块中所存储的指令，所述指令作为被加载到存储器中的字节代码流。VM维持被称作程序计数器(Program Counter PC)的虚拟寄存器，当执行指令时使所述程序计数器增加。VM顺次地执行每个指令，直到遇到OP_STOP指令，在空调用栈的情况下遇到OP_RET指令，或者出现运行时异常。跳转被指定为相对跳转（被指定为从当前的PC值偏移的字节）或指定为绝对地址。

[0543] 1.22.2. 存储器模型

[0544] 在一个实施例中，VM使用相对简单的存储器模型，其中存储器被分成数据存储器和代码存储器。例如，数据存储器可以被实现为起始于地址0的单个、平面、连续的存储空间，并且可以被实现为在主机应用或主机环境的堆存储器内分配的字节数组。在一个实施例中，试图访问在所分配空间之外的存储器导致运行时异常，所述运行时异常会导致程序执行终止。

[0545] 在由虚拟机并发加载的几个代码模块之间潜在地共享数据存储器。数据存储器中的数据可以由存储器访问指令访问，所述存储器访问指令在一个实施例中可以为32比特或8比特访问。使用大端(big-endian)字节次序来执行32比特存储器访问。在优选实施例中，关于在虚拟机可见的存储器和主机管理的存储器（即，主机CPU虚拟或物理存储器）之间的对准并不做任何假定。

[0546] 在一个实施例中，代码存储器是起始于地址0的单个、连续的存储空间，并且可以被实现为在主机应用或主机环境的堆存储器内分配的字节数组。

[0547] VM可以支持加载一个以上代码模块。如果VM加载几个代码模块，那么在一个实施例中，所有代码模块共享相同的数据存储器（不过优选地，每个模块的数据被加载在不同的地址），但是每个代码模块具有其自己的代码存储器，从而防止一个代码模块中的跳转指令导致跳转到另一代码模块中的代码。

[0548] 1.22.3. 数据堆栈

[0549] 在一个实施例中，VM具有数据堆栈的概念，表示在数据存储器中所存储的32比特数据单元。VM维持被称作堆栈指针(Stack Pointer SP)的虚拟寄存器。在复位之后，SP指向数据存储器末尾，并且堆栈向下发展（当把数据压到数据堆栈上时，使SP寄存器减少）。堆栈上的32比特数据单元根据引用堆栈数据的指令而被解释为32比特地址或32比特整数。地址是无符号整数。在一个实施例中，数据堆栈上的所有其它32比特整数值除非另作说明否则被解释为带符号整数。

[0550] 1.22.4. 调用堆栈

[0551] 在一个实施例中，VM管理用于进行子例程调用的调用堆栈。在一个实施例中，被压到此堆栈上的值无法被存储器访问指令直接读取或写入。当执行OP_JSR、OP_JSRR和OP_RET指令时，此堆栈由VM内部使用。对于给定的VM实现方式来说，此返回地址堆栈的大小可以被固定为最大，这只允许一定数目的嵌套调用。

[0552] 1.22.5. 伪寄存器

[0553] 在一个实施例中，VM在数据存储器开始时保留小地址空间以便映射伪寄存器。在一个实施例中，这些伪寄存器的地址是固定的。例如，可以定义以下寄存器：

[0554]

地址	大小	名称	描述
0	4	ID	目前执行代码段的 32 比特 ID。当加载模块时由 VM 选择此 ID。如果 VM 从一个模块的代码段改变到另一模块的代码段，那么 VM 改变此寄存器
4	4	DS	32 比特值被设置为绝对数据地址，在所述绝对数据地址已经加载目前执行模块的数据段。此值由 VM 的模块加载器来确定
8	4	CS	32 比特值被设置为绝对代码地址，在所述绝对代码地址已经加载目前执行模块的代码段。此值由 VM 的模块加载器来确定。
12	4	UM	32 比特值被设置为在数据存储器空间的已经加载代码模块的数据段的区域之后第一字节的绝对数据地址。

[0555] 1.22.6. 存储器映象

[0556] 下面在说明性实施例中示出了数据存储器和代码存储器的布局：

[0557] 数据存储器

[0558]

地址范围	描述
0 到 15	伪寄存器
16 到 127	保留以供将来 VM/系统使用
128 到 255	保留以供应用使用
256 到 DS-1	未指定。VM 可以在 256 或以上的任何地址加载代码模块的数据段。如果它选择大于 256 的地址，那么在 256 和 DS 之间的地址空间的使用被留为未指定的。这意味着虚拟机实现方式依照看起来合适的任何方式来使用它。
DS 到 UM-1	由虚拟机加载的一个或多个代码模块的数据段的映像。
UM 到结束	共享地址空间。代码模块的数据和数据堆栈共享此空间。数据堆栈位于该空间的末尾并且向下发展。末尾表示数据存储器空间的最后地址。数据存储器空间的大小由 VM 根据在代码模块中所包含的存储器需求和实现方式需求来固定。

[0559] 代码存储器

[0560]

地址范围	描述
0 到 CS-1	未指定。虚拟机可以在 0 或以上的任何地址加载代码模块的代码段。如果它选择大于 0 的地址，那么在 0 和 CS 之间的地址空间的使用被留为未指定的。这意味着虚拟机依照看起来合适的任何方式来使用它。
CS 到 CS+size（代码段）-1	由虚拟机加载的代码模块的代码段的映像

[0561]

[0562] 1.22.7. 执行例程

[0563] 在执行代码例程之前，在一个实施例中，虚拟机实现方式复位数据堆栈指针以便指向初始化数据堆栈的顶部。初始化的数据堆栈包含例程的输入数据，并且扩展到数据存

储器的末尾。初始化的数据堆栈可以被用为向例程传递输入自变量的方式。当不存在初始化的数据堆栈时，数据堆栈指针指向数据存储器的末尾。在一个实施例中，初始调用堆栈是空的或者包含指向 OP_STOP 指令的单端返回地址，这在利用 OP_RET 指令完成例程的情况下，将强制执行例程以结束于 OP_STOP 指令。

[0564] 当因为已经执行具有空调用堆栈的最终 OP_RET 指令或已经执行最终 OP_STOP 指令所以执行停止时，留在数据堆栈上的任何数据被认为是例程的输出。

[0565] 1.22.8. 运行时异常

[0566] 在一个实施例中，任何以下条件被认为是造成执行立即停止的运行时异常：

[0567] ●试图访问在当前数据存储器地址空间之外的数据存储器。

[0568] ●试图把 PC 设置为或使所述 PC 到达在当前代码存储器的地址空间之外的代码地址。

[0569] ●试图执行未定义的字节代码。

[0570] ●试图执行具有等于 0 的栈顶操作数的 OP_DIV 指令。

[0571] ●试图执行具有等于 0 的栈顶操作数的 OP_MOD 指令。

[0572] ●调用堆栈的上溢或下溢。

[0573] 1.23. 指令集

[0574] 在一个实施例中，控制 VM 使用相对简单的指令集。尽管是有限的，不过指令数目足以表达任意复杂的程序。指令及其操作数由字节代码流来表示。在一个实施例中，指令集是基于堆栈的，并且除 OP_PUSH 指令之外，没有一个指令具有直接操作数。从数据堆栈中读取操作数，并且把结果压到数据堆栈上。在一个实施例中，VM 是 32 比特 VM：所有指令对用于表示存储器地址或带符号整数的 32 比特堆栈操作数进行操作。利用 2 的补码二进制编码来表示带符号整数。在以下表中示出了供控制 VM 使用的指令集的说明性实施例。在该表中，用于具有两个操作数的指令的堆栈操作数被列为“A，B”，其中在堆栈顶部的操作数被列在最后（即，“B”）。除非另作说明，否则在一个说明性实施例的下面描述中所使用的术语‘压入’指的是把 32 比特的值压入到数据堆栈的顶部上。

[0575]

操作码	名称	字节代码	操作数	描述
OP_NOP	No Operation	0		什么都不作
OP_PUSH	Push Constant	1	N (direct)	进栈 32 比特常数
OP_DROP	Drop	2		移除数据堆栈的顶部单元
OP_DUP	Duplicate	3		复制数据堆栈的顶部单元
OP_SWAP	Swap	4		交换顶部两个堆栈单元
OP_ADD	Add	5	A, B	进栈 A 和 B 的和（A+B）
OP_MUL	Multiply	6	A, B	进栈 A 和 B 的积（A * B）
OP_SUB	Subtract	7	A, B	进栈 A 和 B 间的差（A-B）
OP_DIV	Divide	8	A, B	进栈 A 除以 B（A/B）
OP_MOD	Modulo	9	A, B	进栈 A 按 B 模计算（A%B）

[0576]

操作码	名称	字节代码	操作数	描述
OP_NEG	Negate	10	A	进栈 A 的 2 进制补码求反（- A）
OP_CMP	Compare	11	A, B	如果 A 小于 B，那么进栈-1，如果 A 等于 B 那么进栈 0，并且如果 A 大于 B 那么进栈 1
OP_AND	And	12	A, B	进栈 A 和 B 的逐位与（A& B）
OP_OR	Or	13	A, B	进栈 A 和 B 的逐位 OR（A | B）
OP_XOR	Exclusive Or	14	A, B	进栈 A 和 B 的逐位异或（A ^ B）
OP_NOT	Logical Negate	15	A	进栈 A 的逻辑非（如果 A 为 0 那么为 1，并且如果 A 不为 0,那么为 0）
OP_SHL	Shift Left	16	A, B	进栈 A 在逻辑上左移 B 比特(A << B)
OP_SHR	Shift Right	17	A, B	进栈 A 在逻辑上右移位 B 比特（A >> B）
OP_JMP	Jump	18	A	跳转到 A
OP_JSR	Jump to Subroutine	19	A	跳转到在绝对地址 A 的子例程。PC 的当前值被压进调用堆栈。

操作码	名称	字节代码	操作数	描述
OP_JSRR	Jump to Subroutine (Relative)	20	A	跳转到在 PC+A 的子例程。PC 的当前值被压进调用堆栈。
OP_RET	Return from Subroutine	21		从子程序返回到从调用堆栈出栈的返回地址。
OP_BRA	Branch Always	22	A	跳转到 PC + A
OP_BRP	Branch if Positive	23	A, B	如果 A > 0 那么跳转到 PC+B
OP_BRN	Branch if Negative	24	A, B	如果 A < 0 那么跳转到 PC+B
OP_BRZ	Branch if Zero	25	A, B	如果 A 为 0 那么跳转到 PC+B
OP_PEEK	Peek	26	A	在地址 A 进栈 32 比特值
OP_POKE	Poke	27	A, B	在地址 B 存储 32 比特值 A
OP_PEEKB	Peek Byte	28	A	在地址 A，0 读取 8 比特值—把它扩展到 32 比特并且把它压入到数据堆栈
OP_POKEB	Poke Byte	29	A, B	在地址 B 存储值 A 的最低有效 8 比特
OP_PUSHSP	Push Stack Pointer	30		进栈 SP 的值

[0577]

[0578]

操作码	名称	字节代码	操作数	描述
OP_POPSP	Pop Stack Pointer	31	A	把 SP 的值设置为 A
OP_CALL	System Call	32	A	利用索引 A 来执行系统调用
OP_STOP	Stop	255		终止执行

[0579] 1.24. 代码模块

[0580] 在优选实施例中，依照与用于 MPEG-4 文件格式类似或完全相同的基于原子格式来存储代码模块，其中原子包含 32 比特大小（例如由依照大端字节次序的 4 字节表示），后面是 4 字节类型（例如，对应于字母表字母的 ASCII 值的字节），后面是有效载荷（例如，8 字节）。

[0581] 图 30 示出了说明性代码模块 3000 的格式。参照图 30，pkCM 原子 3002 是最高级的代码模块原子。它包含子原子的序列。在一个实施例中，pkCM 原子 3002 包含一个 pkDS 原子 3004、一个 pkCS 原子 3006、一个 pkEX 原子 3008 以及可能还包含一个 pkRQ 原子 3010。pkCM 原子 3002 还可以包含任意数目的其它原子，在一个实施例中所述其它原子如果存在的话那么被忽略。在一个实施例中，子原子的次序未被指定，因此实现方式不应当假定特定次序。

[0582] 1.24.1.pkDS 原子

[0583] 如图 30 所示，pkDS 原子 3004 包含可以被加载到数据存储器中的数据段的存储器映像 3005。如图 31A 所示，在一个实施例中，存储器映像 3005 由字节 3112 的序列来表示，由一个首部字节 3114 后面是零或更多数据字节 3116 组成。首部字节 3114 编码用于标识随后字节 3116 的格式的版本号。

[0584] 在一个实施例中，只定义了一个版本号（即，DataSegmentFormatVersion = 0），并且依照此格式，存储器映像的数据字节表示要被加载到存储器中的原始映像。虚拟机加载器只加载存储器映像 3105 的数据字节 3116，不包括首部字节 3114。在一个实施例中，虚拟机加载器可操作来拒绝加载采用任何其它格式的映像。

[0585] 1.24.2.pkCS 原子

[0586] 如图 30 所示，pkCS 原子 3006 包含可以被加载到代码存储器中的代码段的存储器映像 3007。如图 31B 所示，在一个实施例中，存储器映像 3007 由字节 3120 的序列来表示，其由一个首部字节 3122 后面是零或更多数据字节 3124 组成。首部字节 3122 编码用于标识随后字节 3124 的格式的版本号。

[0587] 在一个实施例中，只定义了一个版本号（即，CodeSegmentFormatVersion = 0），并且如图 31C 所示，在此版本中，在首部字节 3122 之后的字节包含另一首部字节 3130，所述另一首部字节 3130 包含用于标识随后字节 3132 的字节代码编码的版本号。在图 31C 所示出

的例子中，首部字节 3130 标识 ByteCodeVersion = 0，用于指定数据字节 3132 包含原始字节序列，所述原始字节序列具有诸如在上述示例性指令集中所定义的那些字节代码值。在优选实施例中，虚拟机加载器只加载数据字节的字节代码部分 3132，而不加载两个首部字节 3122、3130。

[0588] 1.24.3.pkEX 原子

[0589] 再次参照图 30，pkEX 原子 3008 包含一列输出入口。在图 30 所示出的例子中，pkEX 原子 3008 的第一组四个字节 3009 依照等于随后入口的数目的大端字节次序来编码 32 比特无符号整数。如图 31D 所示，每个随后的输出入口 3160 包括名称，其被编码为包含名称大小 S 的一个字节 3162，后面是包含该名称的 ASCII 字符的 S 字节 3164，包括终结零 3166，后面是依照大端字节次序的 32 比特无符号整数 3168，用于表示命名入口点的字节偏移，其从在 31CS 原子中所存储的字节代码数据开始测量。图 31E 示出了用于在偏移 64 的入口点 MAIN 的输出表入口 3170 的例子，其中第一字节 3172 指示名称（即“MAIN”）的大小加上末尾的零一共是五个字节，并且其中最后四个字节 3174 表明字节偏移是 64。

[0590] 1.24.4.pkRQ 原子

[0591] 如图 30 所示，pkRQ 原子 3010 包含为了执行代码模块中的代码，由虚拟机实现方式所需要满足的要求。在一个实施例中，此原子是可选的，并且如果不存在的话，那么虚拟机使用默认实现方式设置，诸如可以由实现方式简档来定义。

[0592] 在一个实施例中，pkRQ 原子由 32 比特无符号整数值的数组组成，每个字段一个值：

[0593]

字段名	描述
vmVersion	VM 规范的版本 ID
minDataMemorySize	可用于代码的数据存储器字节中的最小大小。这包括用于加载数据段映像的数据存储器以及由数据堆栈所使用的数据存储器。在一个实施例中，VM 必须拒绝加载模块，如果它无法满足此要求的话。
minCallStackDepth	VM 必须支持的嵌套子例程调用（OP_JSR 和 OP_JSRR）的最小数目。在一个实施例中，VM 必须拒绝加载模块，如果它无法满足此要求的话。
Flags	位标志组以便发信号表示 VM 所要求的特征。 在一个实施例中，VM 实现方式必须拒绝加载具有任何未知标志组的代码模块。例如，如果存在未定义的标志，那么在一个实施例中 VM 实现方式必须检查此标志被设置为 0。

[0594] 1.24.5. 模块加载器

[0595] 虚拟机负责加载代码模块。当加载代码模块时，在 pkDS 原子中所编码的数据段存储器映像被加载到在数据存储器中的存储器地址。该地址由 VM 加载器选择，并且当代码执行时被存储在 DS 伪寄存器中。

[0596] 在 pkCS 原子中所编码的代码段存储器映像被加载到代码存储器中的存储器地址。该地址由 VM 加载器选择，并且当代码执行时被存储在 CS 伪寄存器中。

[0597] 当加载代码模块时，如果在输出表的入口中找到名称为“Global.OnLoad”的特定例程，那么执行此例程。此例程不获取堆栈上的自变量，并且当返回时返回整数状态，0 表示成功，并且负错误代码表示错误条件。

[0598] 当卸载代码模块时（或当去掉已经加载模块的虚拟机时），如果在输出表中找到名称为“Global.OnUnload”的特定例程，那么执行该例程。此例程不获取堆栈上的自变量，并且当返回时返回整数状态，0 表示成功，并且负错误代码表示错误条件。

[0599] 1.25. 系统调用

[0600] 虚拟机的程序可以调用在它们的代码模块的代码段之外实现的函数。这通过使用OP_CALL 指令来完成，所述 OP_CALL 指令获取用于指定系统调用号的整数堆栈操作数来调用。根据系统调用，实现方式可以是在不同的代码模块（例如，实用函数库）中由 VM 依照VM 本机实现格式来直接执行或者被委托给外部软件模块（诸如 VM 的主机环境）来执行的字节代码例程。

[0601] 在一个实施例中，如果利用包含并不对应于任何系统调用的号的操作数来执行OP_CALL 指令，那么 VM 就好像 SYS_NOP 系统调用被调用一样来工作。

[0602] 1.25.1. 系统调用号分配

[0603] 在所讨论的说明性实施例中，系统调用号 0 到 1023 被保留给固定系统调用（这些系统调用在所有 VM 实现方式上具有相同的号）。系统调用号 1024 到 16383 可用于 VM 动态分配（例如，由 System.FindSystemCallByName 所返回的系统调用号可以由 VM 动态地分配，并且不必在所有 VM 实现方式上是相同的号）。

[0604] 在一个示例性实施例中，指定以下固定的系统调用号：

[0605]

记忆码	号	系统调用
SYS_NOP	0	System.NoOperation
SYS_DEBUG_PRINT	1	System.DebugPrint
SYS_FIND_SYSTEM_CALL_BY_NAME	2	System.FindSystemCallByName
SYS_SYSTEM_HOST_GET_OBJECT	3	System.Host.GetObject
SYS_SYSTEM_HOST_SET_OBJECT	4	System.Host.SetObject

[0606] 1.25.2. 标准系统调用

[0607] 在一个实施例中，支持可用于写入控制程序的几个标准系统调用。这些调用包括在上面表中所列出的固定号的系统调用，以及具有动态确定的号的系统调用（即通过调用System.FindSystemCallByName 系统调用来获取它们的系统调用号，它们的名称作为自变量来传递）。

[0608] 在一个实施例中，在可能返回负错误代码的此节中所指定的系统调用可以返回具有任何负值的错误代码。节 8.4.4 定义了特定的说明性值。在一个实施例中，如果返回未被预定义的负错误代码值，那么就像它们是通用的错误代码值 FAILURE 那样来解释。

[0609] System.NoOperation。此调用没有输入并且不返回输出，并且简单地返回而不作任何事情。它主要用于测试 VM。

[0610] System.DebugPrint。此调用从堆栈顶部获取包含空结束字符串的存储单元地址作为其输入，并且不返回输出。对此函数的调用使文本字符串被打印到调试输出，其在调试方面可能是有用的。如果 VM 实现方式不包括用于输出调试文本的手段（诸如可能是在非开发环境中的情况），那么 VM 可以忽略调用并且就好像已经调用 System.NoOperation 那样来处理它。

[0611] System.FindSystemCallByName。给定系统调用的名称，此调用寻找它的编号。所

述调用（从堆栈的顶部）把包含要查找的系统调用名称的空结束 ASCII 字符串的地址作为其输入，并且如果执行具有所指定名称的系统调用，那么（向所述堆栈的顶部）返回系统调用号，如果没有执行该系统调用，那么返回 ERROR_NO_SUCH_ITEM，并且如果出现错误，那么返回负错误代码。

[0612] System.Host.GetLocalTime。此调用没有输入并且向堆栈的顶部返回主机的当前本地时间值或负错误代码，所述当前本地时间值在一个实施例中被表示为等于自从 19701 月 1 日 00:00:00 以来过去的分钟数的 32 比特带符号整数。

[0613] System.Host.GetLocalTimeOffset。这调用没有输入，并且向堆栈的顶部返回主机的当前时间偏移（从UTC时间开始），所述当前时间偏移在一个实施例中被表示为其数目等于在本地时间和 UTC（即 LocalTime-UTC）时间之间分钟差的 32 比特带符号整数。

[0614] System.Host.GetTrustedTime。此调用没有输入，并且向堆栈顶部返回 TrustedTime（信任时间）和一个或多个标志值。在一个实施例中，信任时间是信任计时时钟的当前值（如果系统包括这种信任时钟），或者如果信任时间不可用那么返回负错误代码。在一个实施例中，信任时间值被表示为其数目等于自从 1970 年 1 月 1 日 00:00:00UTC 以来过去的分钟的 32 比特带符号整数，或者为负错误代码。在一个实施例中，所述标志是用于进一步定义信任时钟的当前状态的标志比特组。在一个实施例中，如果已经出现错误（例如，TrustedTime 值为负错误代码），那么对该标志所返回的值为 0。

[0615] 在一个实施例中，定义以下标志：

[0616]

比特索引（0 为 LSB）	名称	描述
0	TIME_IS_ESTIMATE	已知 TrustedTime 值并非是其最准确的值，由此应当被认为是估计值。

[0617] 此系统调用与用于执行信任时钟的系统相关，所述信任时钟可以与信任时间源同步并且维持单调的时间计数器。并不保证信任时间值始终是准确的，但是在一个实施例中要求以下特性为真：

[0618] ●信任时间值被表示为 UTC 时间值（信任时间并不处于本地时区中，这是因为通常无法安全地确定当前的位置）。

[0619] ●信任时间永远不会回退。

[0620] ●信任时钟不比实际时间走得更快。

[0621] 因此，在此示例性实施例中，TrustedTime 值在最后同步的时间值（与信任时间源同步）和当前实际时间值之间。如果系统能够确定自从最后与信任时间源同步以来其信任时钟已经连续地并且通常不中断地操作并更新，那么它可以确定 TrustedTime 值并不是估计值而是准确值，并且把 TIME_IS_ESTIMATE 标志设置为 0。

[0622] 在一个实施例中，如果信任时钟检测到已经出现硬件或软件故障条件，并且它甚至不能返回信任时间的估计值，那么返回错误代码，并且把所返回标志的值设置为 0。

[0623] System.Host.GetObject：此系统调用是用于允许程序访问由虚拟机的主机所提供的对象的通用接口。System.Host.GetObject 调用获取以下输入（从堆栈顶部向下列出）：Parent（亲代），Name（名称），ReturnBuffer（返回缓冲器）和 ReturnBufferSize（返回缓冲器大小）。其中“Parent”是亲代容器 (parent container) 的 32 比特句柄；“Name”是包含通向所请求对象的路径的空结束字符串相对于所述亲代容器的地址；“ReturnBuffer”是存储器缓冲器中将要存储所述对象值的的地址；并且“ReturnBufferSize”是用于表明其中将存储对象值的存储器缓冲器的字节大小的 32 比特整数。

[0624] System.Host.GetObject 调用生成以下输出（从堆栈顶部向下列出）：TypeID（类型 ID），Size（大小）。其中“TypeId”是对象类型 id，或者如果调用失败，那么是负错误代码。如果所请求的对象并不存在，那么所返回的错误为 ERROR_NO_SUCH_ITEM。如果向返回值所提供的缓冲器太小，那么所返回的错误为 ERROR_INSUFFICIENT_SPACE。如果正访问的对象树的一部分是被访问控制的，并且调用程序没有访问该对象的权限，那么返回 ERROR_PERMISSION_DENIED。可以返回其它错误代码。“Size”是 32 比特整数，用于表明在由调用方所提供缓冲器中所返回的数据字节大小或者如果调用方提供了太小的缓冲器，那么它表明所要求的大小。

[0625] 在一个实施例中，存在四种类型的主机对象：字符串，整数，字节数组和容器。

[0626]

对象类型	类型 Id 名称	类型 Id 值
容器	OBJECT_TYPE_CONTAINER	0
整数	OBJECT_TYPE_INTEGER	1
字符串	OBJECT_TYPE_STRING	2
字节数组	OBJECT_TYPE_BYTE_ARRAY	3

[0627] 在一个实施例中，字节数组对象值是 8 比特字节数组，字符串对象值是依照 UTF-8 编码 (incoded) 的空结束字符串，并且整数对象值是 32 比特带符号整数值。容器是包含任何类型组合的任意数目对象序列的通用容器。在容器中所包含的对象被称作该容器的孩子。容器值是在给定 VM 实例内唯一的 32 比特的容器句柄。在一个实施例中，根容器‘/’具有固定句柄值 0。

[0628] 在一个实施例中，用于主机对象的名称空间 (namespace) 是分级的，其中通过向亲代容器的名称追加孩子的名称来构造容器的孩子对象的名称，名称之间由‘/’符号来分隔。字符串和整数对象没有孩子。例如，如果容器被命名为‘/Node/Attributes’(/ 节点 / 属性），并且具有名称为‘Type（类型）’的字符串孩子，那么‘/Node/Attributes/Type(/ 节点 / 属性 / 类型）’指的是孩子字符串。

[0629] 名称空间的根为‘/’。所有绝对名称以‘/’开始。并非以‘/’开始的名称都为相对名称。相对名称是相对于亲代容器的。例如，相对于亲代‘/Node（节点）’的名称‘Attributes/Type（属性 / 类型）’是具有绝对名称‘/Node/Attributes/Type’的对象。

[0630] 在一个实施例中，容器对象还可以具有通过使用虚拟名称来访问的真正和虚拟的孩子对象。虚拟名称是并未被附着到主机对象的名称，但是惯例是标识未命名的孩子对象、具有不同名称的孩子对象或者虚拟孩子对象（并非是容器的真正孩子但是当请求时动态

创建的孩子对象）。

[0631] 在一个实施例中，对于对象来说，以下虚拟名称被定义为虚拟孩子对象名称：

[0632]

虚拟名称	描述
@Name	虚拟字符串对象：对象的名称。 如果对象是未命名的，那么值为空字符串。注意，未命名的对象只可经由容器对象的@<n>虚拟名称来访问（参见下文）
@Size	虚拟整数对象。整数值等于用于存储此对象所要求的字节大小。对于整数来说，此值为 4；对于字符串来说，它是用于存储 UTF-8 字符串所需要的字节数目加上空字节终结符的数目。对于字节数组来说，这是数组中的字节数目。
@Type	虚拟整数对象。整数值等于对象的类型 Id。

[0633] 对于容器来说，在一个实施例中，以下虚拟名称被定义为虚拟孩子对象名称：

[0634]

	虚拟名称	描述
虚拟索引	@<n>	虚拟对象：容器中的第<n>个对象。容器中的第一对象具有索引 0。<n>被表示为十进制数。 例子：如果‘Attributes’是包含 5 个孩子对象的容器，那么‘Attributes/@4’是容器的第五个孩子。
虚拟大小	@Size	虚拟整数对象。整数值等于容器中对象的数目。

[0635] 例子

[0636] 以下表示出了主机对象体系的例子：

[0637]

名称	值	孩子			
节点	1	名称	值	孩子	
		类型	“设备”		
		名称	值	孩子	
		属性	2	名称	值	孩子
				颜色	“红色”	
				名称	值	孩子
				大小	78	
				名称	值	孩子
				域	“TopLevel（顶级）”	

[0638] 在此例子中，调用 System.Host.GetObject(parent = 0,name =”Node”返回类型 ID 0(即，容器))，并且使句柄值 1 被写入到由调用方所提供的缓冲器中。值的大小为 4 字节。

[0639] 调用 System.Host.GetObject(parent = 0,name =”Node/Attributes/Domain”)返回类型 ID 2(即，字符串)，并且使字符串“TopLevel”被写入到由调用方所提供的缓冲器中。值的大小为 9 字节。

[0640] 调用 System.Host.GetObject(parent = 1,name =”Attributes/@1”) 返回类型 ID 1(即，整数)，并且使整数 78 被写入到由被调用方所提供的缓冲器中。值的大小为 4 字节。

[0641] 调用 System.Host.GetObject(parent = 0, name =”DoesNotExist”) 返回错误代码 ERROR_NO_SUCH_ITEM。

[0642] System.Host.SetObject。此系统调用是用于允许程序创建、写入并销毁由虚拟机的主机所提供对象的通用接口。对象名称和类型的描述与上述 System.Host.GetObject 调用的描述相同。并非所有主机对象支持被写入或销毁，并且并非所有容器支持创建孩子对象。当对不支持该操作的对象进行 SetObject 调用时，返回 ERROR_PERMISSION_DENIED。

[0643] System.Host.SetObject 系统调用获取从堆栈顶部向下列出的以下参数作为输入：

[0644] 堆栈顶部

[0645]

Parent
Name
ObjectAddress
ObjectType
ObjectSize

...

[0646] Parent(亲代):亲代容器的 32 比特句柄。

[0647] Name(名称):包含通向对象的路径的空结束字符串相对于亲代容器的地址。

[0648] ObjectAddress(对象地址):存储器缓冲器中存储对象值的地址。如果地址为 0,那么调用被解释为销毁对象的请求。在该地址的数据取决于对象的类型。

[0649] ObjectType(对象类型):对象的类型 ID。

[0650] ObjectSize(对象大小):用于表明其中存储对象值的存储器缓冲器的字节大小的 32 比特整数。在所讨论的说明性实施例中,所述大小对于整数对象来说被设置为 4,并且对于字符串对象来说被设置为存储器缓冲器的大小,包括空终结符。对于字节数组对象来说,大小为数组中字节的数目。

[0651] System.Host.SetObject 系统调用向堆栈顶部返回 ResultCode 作为输出。如果调用成功,那么 ResultCode 为 0,并且如果调用失败,那么 ResultCode 为负错误代码。如果调用是用于销毁对象的请求并且所请求的对象不存在,或者所述调用是用于创建或写入对象的请求并且所述对象的亲代不存在,那么所返回的错误代码为 ERROR_NO_SUCH_ITEM。如果正访问的对象树的一部分是受访问控制的,并且调用程序没有访问对象的权限,那么返回 ERROR_PERMISSION_DENIED。还可以返回其它错误代码。

[0652] 当对象指的是容器并且 ObjectAddress 不是 0 时存在一个特定情况。在这种情况下,ObjectSize 参数被设置为 0 并且 ObjectAddress 值被忽略。如果容器已经存在,那么不作任何事情,并且返回 SUCCESSResultCode。如果容器不存在,并且容器的亲代是可写的,那么创建空容器。

[0653] Octopus.Links.IsNodeReachable。此系统调用由控制程序用来检查给定节点是否可从与主控虚拟机的此实例的实体相关联的节点到达。调用从堆栈顶部获取 NodeId 作为其输入,其中 NodeId 是包含要测试可到达性的目标节点的 ID 的空结束字符串。作为输出,该调用向堆栈顶部返回 ResultCode 和 StatusBlockPointer。ResultCode 是一个整数值,如果该节点是可到达的,那么所述整数值为 0,或者如果所述节点是不可到达的,那么所述整数值为负错误代码。StatusBlockPointer 是标准 ExtendedStatusBlock 的地址,或者如果没有返回状态块,那么为 0。

[0654] System.Host.SpawnVm。此系统调用由控制程序用来请求创建虚拟机的新实例并且加载新的代码模块。在一个实施例中,新创建的虚拟机的主机暴露与被暴露于调用方相同的主机对象,除主机对象“/Octopus/Runtime/Parent/Id”被设置为调用方的身份之外。在一个实施例中,此主机对象是容器。此容器的孩子是类型字符串的对象,均具有用于表示名称的值。在一个实施例中,那些名称的语义和具体细节由虚拟机主机的规范指定。

[0655] 在一个实施例中，当运行用于调用方的代码的虚拟机终止时，尚未通过调用 System.Host.ReleaseVm 显式发布的任何产生的虚拟机由系统来自动发布，就好像已经调用 System.Host.ReleaseVm 一样。

[0656] System.Host.SpawnVm 调用从堆栈顶部获取 ModuleId（模块 ID）作为其输入。ModuleId 标识要被加载到新的虚拟机实例中的代码模块。在一个实施例中，虚拟机的主机的规范描述了用于定位对应于此模块 ID 的实际代码模块的机制。

[0657] System.Host.SpawnVm 调用向堆栈顶部返回 ResultCode 和 VmHandle。ResultCode 是一个整数值，如果调用是成功的，那么所述整数值为 0，并且如果所述调用失败，那么所述整数值为负错误代码。VmHandle 是用于标识已经创建的虚拟机的实例的整数值。如果调用失败，那么此句柄被设置为 0。在一个实施例中，只保证这句柄在其中做出此调用的虚拟机内是唯一的。

[0658] System.Host.CallVm。此系统调用由控制程序用来调用在代码模块中实现的例程，所述代码模块被加载到使用 System.Host.SpawnVm 系统调用创建的虚拟机实例中。此系统调用从堆栈顶部采取以下输入：

[0659] 堆栈顶部：

[0660]

VmHandle
EntryPoint
ParameterBlockAddress
ParameterBlockSize
ReturnBufferAddress
ReturnBufferSize

...

[0661] VmHandle：用于表示通过调用 System.Host.SpawnVm 而创建的虚拟机的句柄的整数值。

[0662] EntryPomt ：用于指定到调用的入口点名称的空结束字符串的地址。此名称需要匹配代码模块的输出表中的入口点之一，所述代码模块被加载到对应于 VmHandle 参数的虚拟机实例中。

[0663] ParameterBlockAddress ：包含要被传递到被调用方的数据的存储块的地址。如果没有向被调用方传递参数，那么此地址被设置为 0。

[0664] ParameterBlockSize ：在地址 ParameterBlockAddress 的存储块的字节大小，或者如果 ParameterBlockAddress 为 0，那么为 0。

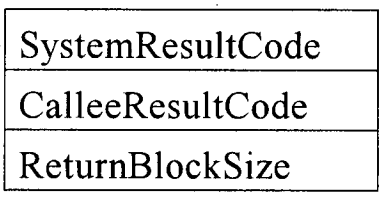
[0665] ReturnBufferAddress：其中调用方可以从被调用方接收数据的存储器缓冲器的地址。如果调用方并不期望从被调用方返回任何数据，那么此地址被设置为 0。

[0666] ReturnBufferSize ：在地址 ReturnBufferAddress 的存储器缓冲器的字节大小，或者如果 ReturnBufferAddress 为 0，那么为 0。

[0667] System.Host.CallVm 调用向堆栈顶部返回以下输出：

[0668] 堆栈顶部：

[0669]



[0670] SystemResultCode :整数值,如果调用成功,那么所述整数值为 0,并且如果所述调用失败,那么所述整数值为负错误代码。此值由系统而不是由被调用方来确定。成功只是表明系统能够成功地找到要调用的例程,执行所述例程并且从所述例程获取返回值。在 CalleeResultCode 值中返回来自该例程本身的返回值。

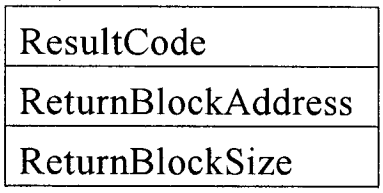
[0671] CalleeResultCode :由被调用方所返回的整数值。

[0672] ReturnBlockSize :在由调用方所提供缓冲器中返回的数据字节大小,或者如果调用方提供了太小的缓冲器,那么它表明所要求的大小。如果被调用方没有返回数据,那么该值为 0。

[0673] 在所讨论的说明性实施例中,被调用例程符合以下接口惯例 :当调用所述例程时,堆栈顶部包含由调用方所提供的值 ParameterBlockSize,后面是数据的 ParameterBlockSize 字节,所述值 ParameterBlockSize 表明参数块的大小。如果所述大小不是 4 的倍数,那么用零来填充堆栈上的数据以便确保堆栈指针保持为 4 的倍数。当返回时,被调用的例程在堆栈上提供了以下返回值 :

[0674] 堆栈顶部 :

[0675]



[0676] ReturnBlockAddress :包含要被返回到调用方的数据的存储块的地址。如果没有返回数据,那么此地址被设置为 0。

[0677] ReturnBlockSize :在地址 ParameterBlockAddress 的存储块的字节大小,或者如果 ReturnBlockAddress 为 0,那么为 0。

[0678] System.Host.ReleaseVm。此系统调用由控制程序用来发布由对 System.Host.SpawnVm 的先前调用所产生的虚拟机。递归地发布由所发布的虚拟机产生的任何虚拟机等。System.Host.ReleaseVm 调用从堆栈顶部获取 VmHandle 作为其输入,VmHandle 表示通过调用 System.Host.SpawnVm 所创建的虚拟机的句柄。System.Host.ReleaseVm 调用向堆栈顶部返回 ResultCode 作为输出。ResultCode 是个整数值,如果调用是成功的,那么所述整数值为 0,或者如果所述调用失败,那么所述整数值为负错误代码。

[0679] 1.25.3. 标准的数据结构

[0680] 下面是由一些标准的系统调用所使用的标准数据结构。

[0681] 1.25.3.1. 标准参数

[0682] ParameterBlock :

[0683]

名称	类型
Name	NameBlock
Value	ValueBlock

[0684] Name :参数的名称。

[0685] Value :参数的值

[0686] ExtendedParameterBlock :

[0687]

名称	类型
Flags	32 比特的比特字段
Parameter	ParameterBlock

[0688] Flags :布尔标志的向量。

[0689] Parameter :包含名称和值的参数块。

[0690] NameBlock :

[0691]

名称	类型
Size	32 比特整数
Characters	8 比特字符的数组

[0692] Size :32 比特无符号整数等于随后的“字符”字段的字节大小。如果此值为 0,那么字符字段为空(即,后面什么都没有)。

[0693] Characters :空结束 UTF-8 字符串。

[0694] ValueBlock :

[0695]

名称	类型
Type	32 比特整数
Size	32 比特整数
Data	8 比特字节的数组

[0696] Type :32 比特类型标识符。在一个实施例中,定义以下类型 :

[0697]

标识符	类型名称	描述
0	Integer	32 比特整数值，依照大端字节次序被编码为四个 8 比特字节。在一个实施例中，除非另作说明否则该值被认为是有符号的。
1	Real	32 比特浮点值，依照大端字节次序被编码为 IEEE-754。
2	String	空结束 UTF-8 字符串
3	Date	32 比特无符号整数值，表示自从 1970 年 1 月 1 日 00:00:00 以来所过去的分钟数。在一个实施例中，除非另作说明否则该值被认为是 UTC 日期，其最高有效位必须为 0。
4	Parameter	ParameterBlock 结构
5	ExtendedParameter	ExtendedParameterBlock 结构
6	Resource	该值为资源。这里资源用 ID 引用：值的数据字段是空结束 ASCII 字符串，包含应当被去引用以便生成实际数据的资源 ID。
7	ValueList	值数组（被编码为 ValueListBlock）
8	ByteArray	该值是 8 比特字节的数组

[0698] Size :32 比特无符号整数等于随后的“数据”字段的字节大小。如果此值为 0,那么数据字段为空（即,在 ValueBlock 中的大小字段后面什么都没有）。

[0699] Data :用于表示值的 8 比特字节的数组。实际字节取决于由类型字段所指定的数据编码。

[0700] ValueListBlock :

[0701]

名称	类型
ValueCount	32 比特整数
Value0	ValueBlock
Value1	ValueBlock
…	…

[0702] ValueCount :等于随后的 ValueBlock 结构数目的 32 比特无符号整数。如果此值为 0,那么后面没有 ValueBlocks。

[0703] Value0, Value1, ... :零或更多 ValueBlock 结构的序列。

[0704] 1.25.3.2. 标准的 ExtendedStatus

[0705] 标准的 ExtendedStatusBlock 是一般用于把扩展信息作为返回状态从调用传达到例程或系统调用的数据结构。它是可以在各种情境中使用的通用数据结构,对于其字段来说具有不同可能值的范围。在一个实施例中,如下定义 ExtendedStatusBlock :

[0706] ExtendedStatusBl ock :

[0707]

名称	类型
GlobalFlags	32 比特的比特字段
Category	32 比特整数
SubCategory	32 比特整数
LocalFlags	32 比特的比特字段
CacheDuration	CacheDurationBlock
Parameters	ValueListBlock

[0708] GlobalFlags :不管类别字段如何其语义都是相同的布尔标志。标志的位置和意义由使用标准 ExtendedStatusBlock 数据结构的简档来定义。

[0709] Category :此状态所属的类别的唯一整数标识符。类别标识符值由使用标准 ExtendedStatusBlock 数据结构的简档来定义。

[0710] SubCategory :用于进一步分类由此块所描述的状态类型的子类别的整数标识符

（在所述类别内是唯一的）。

[0711] LocalFlags：其语义局限于此状态块的类别和子类别局部的布尔标志。标志的位置和意义由用于定义并使用类别语义的简档来定义。

[0712] CacheDuration：表明此状态可以被高速缓存（即保持有效）的持续时间。对于怎样来定义实际的持续时间值，参见下面 CacheDurationBlock 类型的定义。

[0713] Parameter：零或更多 ValueBlocks 的列表。每个 ValueBlock 包含被编码为类型参数或 ExtendedParameter 值的参数。每个参数把名称绑定到类型值，并且用来编码灵活的变量数据，所述变量数据用于依照比只是类别、子类别、高速缓存持续时间和标志更详细的细节来描述状态块。

[0714] CacheDurationBlock：

[0715]

名称	类型
Type	32 比特整数
Value	32 比特整数

[0716] Type：用于值类型的整数标识符。在一个实施例中，定义以下类型：

[0717]

类型	描述
0	该值为表示当前时间的秒数的 32 比特无符号整数。值 0 意味着状态根本无法被高速缓存，由此只可以被使用一次。特定值 0xFFFFFFFF 被解释为无穷大的持续时间（即，所述状态可以被无限地高速缓存）。
1	该值为表示绝对本地时间的 32 比特无符号整数，其被表示为自从 1970 年 1 月 1 日 00:00:00 以来过去的分钟数。在一个实施例中，最高有效位必须为 0。

[0718] Value：32 比特整数，其意义取决于类型字段。

[0719] 1.25.4. 标准的结果代码

[0720] 在各个 API 中使用标准的结果代码。可以定义其它结果代码以供在更特定的 API 中使用。

值	名称	描述
0	SUCCESS	成功
-1	FAILURE	未指定的失败
-2	ERROR_INTERNAL	已经出现内部（执行）错误
-3	ERROR_INVALID_PARAMETER	参数具有无效值
-4	ERROR_OUT_OF_MEMORY	没有可用于成功完成的足够存储器
-5	ERROR_OUT_OF_RESOURCES	没有可用于成功完成的足够资源
-6	ERROR_NO_SUCH_ITEM	所请求的项不存在或未找到
-7	ERROR_INSUFFICIENT_SPACE	调用方没有提供足够的存储器（一般当返回缓冲器太小时使用）
-8	ERROR_PERMISSION_DENIED	拒绝向调用方授予用于执行调用的权限。
-9	ERROR_RUNTIME_EXCEPTION	在字节代码执行期间已经出现错误
-10	ERROR_INVALID_FORMAT	由具有无效格式的数据（例如代码模块中的无效数据）所引起的错误

[0721]

[0722] 1.26. 汇编器语法

[0723] 此节描述了用于把程序编译为在此的其它地方所描述的字节码格式的示例性语法。应当理解，这只是一个可能语法的一个例子，并且可以使用任何适当的语法。如先前所表明，还应当理解，这里所给出的字节码格式也只是一个例子，并且这里所描述的系统和方法可以被以任何其它适当的字节代码格式或其它代码格式的方式使用。

[0724] 汇编器读取包含代码、数据和处理指令的源文件，并且生成可以由控制虚拟机所加载的二进制代码模块。在一个说明性实施例中，汇编器逐行顺序地处理源文件。行可以是零或更多符号，后面是换行符。每行可以是以下之一：空行（只是空白的）、段命令、数据命令、汇编器命令、代码指令、标记或输出命令。另外，每行可以以注解结束，其以‘；’符号开始并且继续直到该行的末尾。

[0725] 从源文件所读取的数据和指令具有隐式的目的地段（即，其中当它们被 VM 加载时结束）。在解析过程期间的任何一点，汇编器具有“当前”段，其是用于数据和指令的隐式目的地段。可以使用段指令来改变当前段。

[0726] 1.26.1. 段命令

[0727] 段命令改变解析器的当前段。在一个实施例中，所支持的段命令是.code和.data。.code段保持字节代码指令，并且.data段保持全局变量。

[0728] 1.26.2. 数据命令

[0729] 数据命令指定将被加载到虚拟机的数据段中的数据（例如，整数和字符串）。在一个实施例中，所支持的数据命令是：

[0730] ●.string“<some chars>”——指定一串符号。在一个实施例中，汇编器在字符串末尾添加值为0的八位字节。

[0731] ●.byte<value>——指定8比特值。<value>可以被表示为十进制数或十六进制数（前缀为0x）。

[0732] ●.long<value>——指定32比特值。<value>可以被表示为十进制数或十六进制数（前缀为0x）。

[0733] 1.26.3. 汇编器命令

[0734] 在一个实施例中，所支持的汇编器命令是.equ<symbol>、<value>，用于把符号<symbol>设置为等于值<value>。符号一般被用为操作数或代码指令。

[0735] 1.26.4. 标签

[0736] 标签是指向段内位置的符号。指向代码段中指令的标签一般用于跳转/分支指令。指向数据段中数据的标签一般用于参照变量。在一个实施例中，用于标签的语法是<LABEL>：

[0737] 注意，除可选注解之外，在“：”之后什么都没有。标签指向下一数据或指令的位置。在一个实施例中，可以具有指向相同地址的一个以上标签。

[0738] 1.26.5. 输出命令

[0739] 输出命令用来在由汇编器所生成的代码模块的“输出”节中创建入口。输出节中的每个入口是（名称，地址）对。在所讨论的说明性实施例中，仅有代码段内的地址能够被在输出节中指定。

[0740] 输出命令的语法是:.export<label>，用于输出由<label>指向的地址，其名称为“<label>”。

[0741] 1.26.6. 代码指令

[0742] 当编译前往代码段的数据时，汇编器读取用于直接或间接映射成字节代码的指令。在上面所示出的示例性指令集中，大部分虚拟机字节代码没有直接的操作数，并且在一行上以简单的助记符形式出现。为了使汇编器语法更加可读，一些指令接受伪操作数，所述伪操作数看起来好像它们是字节代码操作数，但是实际上并不是；在这种情况下，汇编器产生一个或多个字节代码指令以便产生就像所述指令的确具有直接操作数那样相同的效果。例如，分支指令使用伪操作数。

[0743] 1.26.6.1. 分支操作数

[0744] 分支指令可以被逐字指定（没有任何操作数），或者被用被汇编器转换为相应的字节代码序列的可选操作数指定。可选操作数是整数常数或符号。当操作数是符号时，汇编器计算正确的整数相对偏移，使得分支结束于对应于所述符号的地址。

[0745] 1.26.6.2. 操作数入栈

[0746] 在一个实施例中，PUSH 指令始终获取一个操作数。该操作数可以是整数常数、符号或前缀“@”后面直接是标签名称之一。当该操作数是符号时，进栈的值是该符号的直接值，而不管所述符号是标签还是 .equ 符号（并不按照段偏移增加该值）。当该操作数是具有前缀“@”的标签名称时，进栈的值取决于标签指向什么。被压进堆栈的值是由标签所表示的绝对地址（即，被添加到段偏移的本地标签值）。

[0747] 1.26.7. 例子

[0748]

```
; 常数
.equ SOMECONST, 7

; 下面进入到数据段中
    .data
```

[0749]

```
VAR1:
        .byte 8
VAR2:
        .string “hello\0”
VAR3:
        .long 0xFFFCDA07
VAR4:
        .long 0

; 下面进入到代码段中
        .code
FOO:
        PUSH 1
        ADD
        RET
BAR:
        PUSH 2
        PUSH @FOO           ; 进栈标签 FOO 的地址
        JSR                 ; 跳转到标签 FOO 处的代码
        PUSH SOMECONST      ; 进栈值 7
        PUSH @VAR1          ; 进栈 VAR1 的地址
        PUSH VAR1           ; 进栈数据段内 VAR1 的偏移
        PUSH @VAR3          ; 进栈 VAR3 的地址
        PEEK                ; 进栈 VAR3 的值
        PUSH @VAR4          ; 进栈 VAR4 的地址
        POKE                ; 把堆栈顶部上的值存储到 VAR4 中
        PUSH @VAR2          ; 进栈字符串 “hello” 的地址
```

[0750] 1.26.8. 命令行语法

[0751] 在一个实施例中，汇编器是可以利用以下语法调用的命令行工具：“PktAssembler[options]<input_file_path><output_file_path>”，其中 [options] 可以是 :-cs int，-ds int，-xml id 或 -h，其中“-cs int”是代码段地址值（默认＝ 8），“-ds int”是数据段地址值（默认＝ 4），“-xml id”用来输出控制对象作为具有所指定 ID 的 XML 文件，并且“-h”用来显示帮助信息。

[0752] 9. 控制

[0753] 此节描述了控制对象的说明性实施例。控制对象可以用来表示用于通过准许或拒绝使用它们所控制的内容密钥对象来管理支配对内容进行访问的规则。它们还可以用来表示对它们被嵌入到的链路对象的有效性的约束。它们还可以被用为独立程序容器，代表诸如代理或委托之类的另一实体来运行所述独立程序容器。在一个实施例中，控制包含元数据和字节代码程序，用于执行特定的交互协议。控制协议的目的在于经由 DRM 引擎来指定在 DRM 引擎和控制程序之间或在主机应用和控制程序之间的交互。此节还描述了应用可以对内容执行的说明性动作，所述动作的参数应当被提供给控制程序，并且还描述所述控制程序怎样编码用于表明可以或不可以执行所请求动作的返回状态以及可以进一步描述所述返回状态的参数。

[0754] 在此节中，使用以下缩写和字母缩略词（：

[0755] ● ESB：扩展状态块

[0756] ● LSB：最低有效位

[0757] ●字节：8 比特值或八位字节

[0758] ●字节代码：用于编码可执行指令及其操作数的字节流

[0759] 1.27. 控制程序

[0760] 在一个实施例中，控制对象包含控制程序。控制程序包括代码模块，所述代码模块包含可由虚拟机执行的字节代码，以及命名例程列表（例如，输出表中的入口）。

[0761] 在一个实施例中，用于表示用来管理支配对内容项执行某个操作（诸如“播放”）的规则的例程集被称作‘动作控制’。用于表示对链路对象的有效性约束的例程集被称作“链路约束”。旨在代表远程实体执行的例程集（诸如在不同主机上运行的 DRM 引擎的情况下在协议会话期间）被称作“代理”。旨在代表另一控制执行的例程集（诸如当控制程序使用 System.Host.CallVm 系统调用时）被称作“委托”。

[0762] 1.27.1. 通向控制程序的接口

[0763] 在一个实施例中，由在主机环境中运行的虚拟机来执行控制程序。可以依照任何适当的方式来执行主机环境；然而，为了便于解释以及为了图示，在下面论述中假定虚拟机主机环境的实现方式在逻辑上可以被分成两个部分：主机应用和 DRM 引擎。然而应当理解，其它实施例可以具有不同的逻辑功能划分，其可以等价于上述逻辑结构。

[0764] 如在图 29 中所示出，在优选实施例中，DRM 引擎 2908 是在主机应用 2900 和控制程序 2906 之间的逻辑接口。主机应用 2900 向引擎 2908 做出逻辑请求，诸如为了某个目的（例如，播放或再现内容流）而请求访问内容密钥。在一个实施例中，引擎 2908 确保正确地实现下述交互协议，诸如通过确保关于控制程序的初始化、调用序列及其它交互细节的任何保证被满足来正确地实现。

[0765] 当主机应用 2900 请求使用用于一组内容 ID 的内容密钥时，DRM 引擎 2908 确定使用哪个控制对象。保护器对象允许引擎分析对于所请求的内容 ID 来说哪些内容密钥对象需要被访问。然后所述引擎寻找引用那些内容密钥对象的控制器对象。在一个实施例中，控制器对象可以引用一个以上内容密钥对象。这允许多个内容密钥对象由相同的控制对象来管理支配。当主机应用通过调用动作来请求访问内容密钥时，它可以请求作为一组的内容 ID，在这个意义上对应于它们的内容密钥对象由相同的控制器对象来引用。在一个实施

例中，不允许对访问由一个以上控制器对象引用的一组内容密钥的请求。

[0766] 在一个实施例中，DRM 引擎遵循用于把动作映射到例程名称的约定。例如在一个实施例中，对于下述每个例程来说，在代码模块的输出表入口中出现的名称是下面在节 9.1.4-9.1.7 中所示出的各自字符串。

[0767] 1.27.1.1. 控制加载

[0768] 在一个实施例中，在引擎可以进行调用以便控制例程之前，它需要把控制的代码模块加载到虚拟机中。在一个实施例中，每个 VM 只加载一个代码模块。

[0769] 1.27.1.2. 原子性 (Atomicity)

[0770] 在一个实施例中，该引擎确保对控制程序内例程的调用相对于它使所述例程可用的资源来说是原子的，所述资源诸如对象（或“状态”）数据库。从而，在这种实施例中，所述引擎需要确保在执行它所调用的任何例程期间那些资源保持不被修改。这可以通过在例程调用期间有效地锁定那些资源或者通过防止多个 VM 并发地运行来完成。然而，所述引擎不必保证那些资源在连续的例程调用之间都不被修改。

[0771] 1.27.2. 控制协议

[0772] 在一个实施例中，例程命名、输入 / 输出接口和用于代码模块中每个例程的数据结构一起构成控制协议。在控制对象的“协议”字段中表示由代码模块实现的协议。下述说明性控制协议被称作标准控制协议，并且其标识符（‘协议’字段值）是“http://www.octopus-drm.com/specs/scp-1_0”。

[0773] 在一个实施例中，在 DRM 引擎加载代码模块并且调用控制程序中的例程之前，它需要保证与控制程序的交互要符合在协议字段中表示的特定协议 id 的规范。这包括关于需要实现的虚拟机特征的任何保证、关于可用于控制程序的地址空间大小的保证等。

[0774] 对于诸如标准控制协议之类的控制协议来说，可以在不必创建新的协议规范的情况下随时间推移而演进。只要对协议所作出的改变符合规范的先前修订版，并且只要 DRM 引擎的现有实现方式以及符合该协议的现有控制程序继续依照所述规范来执行，那么所述改变就被认为是兼容的。这种改变例如可以包括新的动作类型。

[0775] 1.27.3. 字节代码类型

[0776] 在上述涉及标准控制协议的说明性实施例中，字节代码模块的类型是“Plankton 字节代码模块版本 1.0”。在此示例性实施例中，控制对象的“类型”字段值是“http://www.octopus-drm.com/specs/pkcm-1_0”。

[0777] 1.27.4. 通用控制例程

[0778] 通用例程是适用所述控制作为总体的例程，并且对于给定的动作或链路约束并不是专门的。在一个说明性实施例中使用以下通用控制例程：

[0779] 1.27.4.1.Control.Init

[0780] 此例程是可选的（即，在所有控制中都不要求此例程）。如果使用此例程，那么在任何其它控制例程被调用之前，该引擎调用此例程一次。所述例程没有输入，并且向堆栈顶部返回 ResultCode（结果代码）作为输出。ResultCode 在成功时为 0，或者在失败时为负错误代码。在一个实施例中，如果 ResultCode 不是 0，那么该引擎中止当前的控制操作并且不对用于此控制的例程进行任何进一步的调用。

[0781] 1.27.4.2.Control.Describe

[0782] 此例程是可选的。通常，当应用请求由控制程序（即并非是针对具体动作）表示的规则的意义描述时调用该例程。所述例程没有输入，并且向堆栈顶部返回 ResultCode 和 StatusBlockPointer（状态块指针）作为输出，其中 ResultCode 是整数值（如果例程成功完成那么为 0，否则为负错误代码），并且其中 StatusBlockPointer 是标准 ExtendedStatusBlock（扩展状态块）的地址。ExtendedStatusBlock 包含这样的信息，应用可以解释该信息并使用它来向用户提供关于由控制程序所表示规则的意义的信息。

[0783] 1.27.4.3.Control.Release

[0784] 此例程是可选的。如果此例程存在，那么 DRM 引擎在它不再需要调用用于控制的任何其它例程之后调用所述例程一次。对于控制来说将不调用其它例程，除非启动了该控制的新使用（在这种情况下，Control.Init 例程被再次调用）。所述例程没有输入，并且向堆栈顶部返回 ResultCode 作为输出。ResultCode 在成功时为 0，或者在失败时为负错误代码。

[0785] 1.27.5. 动作例程

[0786] 每个可能的动作具有名称（例如，播放，转送，输出等）。在一个说明性实施例中，对于给定动作 <Action> 来说，以下例程名称被定义（其中“<Action>”标示动作的实际名称（例如，“播放”，“转送”，“输出”等））：

[0787] 1.27.5.1.Control.Actions.<Action>.Init

[0788] 此例程是可选的。如果此例程存在，那么在为此动作调用任何其它例程之前，引擎调用此例程一次。所述例程没有输入，并且向堆栈顶部返回 ResultCode 作为输出。ResultCode 在成功时为 0，或者在失败时为负错误代码。在一个实施例中，如果 ResultCode 不是 0，那么该引擎中止当前动作并且不对用于此控制中此动作的例程做出任何进一步的调用。

[0789] 1.27.5.2.Control.Actions.<Action>.Check

[0790] 在所论述的说明性实施例中，此例程被要求，并且被调用来在实际上没有执行给定动作的情况下检查如果将为该动作而调用执行例程那么返回状态会是什么。对于此例程来说重要的是没有任何副作用。注意，如果执行例程也没有副作用，那么控制的入口表中的检查和执行入口可以指向相同的例程。此例程具有与下述执行例程相同的输入和输出。

[0791] 1.27.5.3.Control.Actions.<Action>.Perform

[0792] 在一个实施例中，此例程被要求，并且当应用即将执行动作时被调用。所述例程没有输入，并且向堆栈顶部返回 ResultCode 和 StatusBlockPointer 作为输出，其中 ResultCode 是整数值（如果例程成功完成那么为 0，否则为负错误代码），并且其中 StatusBlockPointer 是标准 ExtendedStatusBlock 的地址。注意，在一个实施例中，成功 ResultCode（即，0）并不意味着所述请求被准许了。它只意味着所述例程能够没有错误地运行。是 ExtendedStatusBlock 表明所述请求是被准许了还是拒绝了。然而，如果 ResultCode 表明失败的话，那么主机应用就好像所述请求被拒绝一样进行。例如在一个实施例中，StatusBlock 的类别应当为 ACTION DENIED，或者所返回的 ExtendedStatusBlock 被决绝，因而主机应用中止该动作。

[0793] 当执行动作时，只有执行例程需要被调用。所述引擎不必事先调用检查例程。执行例程的实现可以内部调用检查例程（如果它选择这样作的话），但是不应当假定所述系

统将已经事先调用了所述检查例程。

[0794] 1.27.5.4.Control.Actions.<Action>.Describe

[0795] 此例程是可选的，并且当应用请求由控制程序为给定动作所表示的规则和条件的意义描述时被调用。所述例程没有输入，并且向堆栈顶部返回 ResultCode 和 StatusBlockPointer 作为输出，其中 ResultCode 是整数值（如果例程成功完成那么为 0，否则为负错误代码），并且其中 StatusBlockPointer 是标准 ExtendedStatusBlock 的地址。

[0796] 1.27.5.5.Control.Actions.<Action>.Release

[0797] 此例程是可选的。如果此例程存在，那么在 DRM 引擎不再需要调用用于给定动作的任何其它例程之后所述例程被调用一次。对于给定动作来说不调用其它例程，除非启动所述动作的新使用（在这种情况下，Init 例程被再次调用）。所述例程没有输入，并且向堆栈顶部返回 ResultCode 作为输出。ResultCode 在成功时为 0，并且在失败时为负错误代码。如果 ResultCode 不为 0，那么所述引擎不对用于给定动作的例程进行任何进一步的调用。

[0798] 1.27.6. 链路约束例程

[0799] 在一个实施例中，当链路对象具有嵌入的控制时，DRM 引擎在该控制中调用链路约束例程以便验证所述链路对象的有效性。在一个说明性实施例中使用以下链路约束例程：

[0800] 1.27.6.1.Control.Link.Constraint.Init

[0801] 此例程是可选的，并且如果它存在的话，那么在为给定链路约束调用任何其它例程之前此例程被调用恰好一次。所述例程没有输入，并且向堆栈顶部返回 ResultCode 作为输出。ResultCode 在成功时为 0 并且在失败时为负错误代码。如果 ResultCode 不为 0，那么该引擎认为对链路对象的有效性约束未被满足，并且避免对用于所述链路控制的例程进行进一步调用。

[0802] 1.27.6.2.Control.Link.Constraint.Check

[0803] 在所论述的说明性实施例中，此例程被要求，并且被调用来检查是否满足对给定链路的有效性约束。所述例程没有输入，并且向堆栈顶部返回 ResultCode 和 StatusBlockPointer 作为输出，其中 ResultCode 是整数值（如果例程成功完成那么为 0，否则为负错误代码），并且其中 StatusBlockPointer 是标准 ExtendedStatusBlock 的地址。如果 ResultCode 不为 0，那么引擎认为对链路对象的有效性约束未被满足，并且避免对用于所述链路控制的例程做出进一步调用。即便 ResultCode 为 0（成功），这也不意味着所述约束已经满足；它只意味着所述例程能够没有错误地运行。是 StatusBlock 表明是否满足所述约束。

[0804] 1.27.6.3.Control.Link.Constraint.Describe

[0805] 此例程是可选的，并且当应用请求由控制程序为给定链路所表示的约束意义的描述时被调用。所述例程没有输入，并且向堆栈顶部返回 ResultCode 和 StatusBlockPointer 作为输出，其中 ResultCode 是整数值（如果例程成功完成那么为 0，否则为负错误代码），并且其中 StatusBlockPointer 是标准 ExtendedStatusBlock 的地址。

[0806] 1.27.6.4.Control.Link.Constraint.Release

[0807] 此例程是可选的，并且如果它存在的话，那么在引擎不再需要为给定约束调用任何其它例程之后由所述引擎调用一次。所述例程没有输入，并且向堆栈顶部返回

ResultCode 作为输出。ResultCode 在成功时为 0 并且在失败时为负错误代码。在所论述的实施例中，在调用此例程之后，除非开始新的循环（在这种情况下，再次调用 Init 例程），否则可以不为给定约束调用任何其它例程。类似地，如果 ResultCode 不是 0，那么引擎不对用于所述给定链路约束的例程进行进一步的调用。

[0808] 1.27.7. 代理例程

[0809] 在一个实施例中，代理是被设计成用于代表实体运行的控制对象。一般在两个端点之间的服务交互的情境下使用代理，其中一个端点需要在第二端点的情境内执行某个虚拟机代码，并且可能获得该执行的结果。在一个实施例中，控制可以包含多个代理，并且每个代理可以包含能被执行的任意数目的例程；然而在实践中代理一般具有单个例程。

[0810] 在一个说明性实施例中，为代理定义以下入口点，其中 <Agent> 是指代代理的实际名称的名称字符串。

[0811] 1.27.7.1.Control.Agents.<Agent>.Init

[0812] 此例程是可选的，并且如果它存在的话，那么在为给定代理调用任何其它例程之前所述引擎调用此例程一次。所述例程没有输入，并且向堆栈顶部返回 ResultCode 作为输出。ResultCode 在成功时为 0 并且在失败时为负错误代码。

[0813] 1.27.7.2.Control.Agents.<Agent>.Run

[0814] 在所讨论的说明性实施例中，此例程被要求并且是代理的主例程。所述例程没有输入，并且向堆栈顶部返回 ResultCode、ReturnBlockAddress（返回块地址）和 ReturnBlockSize（返回块大小）作为输出。ResultCode 是整数值（如果例程成功完成那么为 0，否则为负错误代码），ReturnBlockAddress 是包含代理代码被预期返回给调用方的数据的存储器块的地址，（如果例程不必返回任何东西，那么该地址为 0），并且 ReturnBlockSize 是在 ReturnBlockAddress 的存储块器块的字节大小。在一个实施例中，如果 ReturnBlockAddress 是 0，那么 ReturnBlockSize 值也是 0。

[0815] 1.27.7.3.Control.Agents.<Agent>.Describe

[0816] 此例程是可选的，并且当应用请求给定代理的描述时被调用。所述例程没有输入，并且向堆栈顶部返回 ResultCode 和 StatusBlockPointer 作为输出，其中 ResultCode 是整数值（如果例程成功完成那么为 0，否则为负错误代码），并且其中 StatusBlockPointer 是标准 ExtendedStatusBlock 的地址。

[0817] 1.27.7.4.Control.Agents.<Agent>.Release

[0818] 此例程是可选的，并且如果它存在的话，那么在引擎不再需要为此代理调用任何其它例程之后，所述引擎调用此例程一次。对于此代理来说不调用其它例程，除非开始新的循环（在这种情况下，Init 例程被再次调用）。所述例程没有输入，并且向堆栈顶部返回 ResultCode 作为输出。ResultCode 在成功时为 0 并且在失败时为负错误代码。

[0819] 1.28. 扩展的状态块

[0820] 以下示例性定义适用于由上述几个例程的说明性实施例所返回的 ExtendedStatusBlock（扩展状态块）数据结构。结合描述虚拟机来描述 ExtendedStatusBlock 数据结构的例子。

[0821] 在一个实施例中，不存在全局 ExtendedStatusBlock 标志。在此实施例中，控制程序把 ExtendedStatuBlock 的 GlobalFlag（全局标志）字段设置为 0。

[0822] 1.28.1. 类别

[0823] 依照一个实施例以下段落定义了用于 ExtendedStatusBlocks 的类别字段的值。在一个实施例中，这些类别都没有子类别，从而 ExtendedStatusBlocks 的子类别字段的值被设置为 0。

[0824] 在一个实施例中，定义以下类别代码：

[0825] 1.28.1.1. 动作检查和执行例程

[0826]

值	名称	描述
0	ACTION_GRANTED	为了所请求动作的目的，应用被授权使用由控制程序所控制的内容密钥。 所返回 ExtendedStatusBlock 的参数列表不应当包含任何约束参数，但是可以包含义务和/或回调参数。
1	ACTION_DENIED	为了所请求动作的目的，应用不被授权使用由控制程序所控制的内容密钥。 当动作被拒绝时，控制程序应当在所返回 ExtendedStatusBlock 的参数列表中包括未被满足并导致动作被拒绝的一个或多个约束（应当省略未被评估的约束以及没有导致动作失败的约束）。 在一个实施例中，所返回 ExtendedStatusBlock 的参数列表不能包含任何义务或回调参数。

[0827] 在一个实施例中，在由动作例程所返回的 ExtendedStatusBlock 参数的情境中，约束意指为了让例程的结果返回具有类别 ACTION_GRANTED 的 ExtendedStatusBlock 而需要满足的准则或者要求为真的条件。

[0828] 在一个实施例中，通用于上述两个类别的 LocalFlags（局部标志）字段的值包括：

[0829]

比特索引（0 为 LSB）	名称	描述
0	OBLIGATION_NOTICE	参数列表包含与义务相关的一个或多个参数
1	CALLBACK_NOTICE	参数列表包含与回调相关的一个或多个参数
2	GENERIC_CONSTRAINT	参数列表包含与通用约束相关的一个或多个参数
3	TEMPORAL_CONSTRAINT	参数列表包含与时间约束相关的一个或多个参数
4	SPATIAL_CONSTRAINT	参数列表包含与空间约束相关的一个或多个参数
5	GROUP_CONSTRAINT	参数列表包含与组约束相关的一个或多个参数
6	DEVICE_CONSTRAINT	参数列表包含与设备约束相关的一个或多个参数
7	COUNTER_CONSTRAINT	参数列表包含与计数器约束相关的一个或多个参数

[0830] 在上面所示出的表中，所涉及的参数列表是 ExtendedStatusBlock 数据结构的“参数”字段。

[0831] 1.28.1.2. 描述例程类别代码

[0832] 在一个实施例中，对于描述例程来说没有定义类别代码。在一个实施例中，与为动作例程所定义的局部标志相同的局部标志适用于描述例程，并且描述例程应当在它们返回的 ExtendedStatusBlock 中包括名称为如下面所指定的‘描述’的参数。在一个实施例中，描述例程在它们返回的 ExtendedStatusBlock 中不包含任何义务或回调参数；然而描述例程应当在它们返回的 ExtendedStatusBlock 中包含用于描述适用于相应动作或链路约束的一些或所有约束的参数。

[0833] 1.28.1.3. 链路约束例程类别代码

值	名称	描述
0	LINK_VALID	由此控制程序所约束的链路是有效的。所返回的ESB的参数列表不应当包含任何约束参数，并且在一个实施例中不能包含义务或回调参数
1	LINK_INVALID	由此控制程序所约束的链路是无效的。当链路为无效时，控制程序应当在所返回ESB的参数列表中包括未被满足并导致链路无效的一个或多个约束（应当省略未被评估的约束以及没有导致链路失败的约束）。 在一个实施例中，所返回的ESB的参数列表不能包含任何义务或回调参数。

[0834]

[0835] 在一个实施例中，与为动作例程所定义的局部标志相同的局部标志适用于这些类别中的每个。

[0836] 在一个实施例中，在由链路约束例程所返回的ExtendedStatusBlock参数的情境中，约束意指为了让例程的结果返回具有类别LINK_VALID的ExtendedStatusBlock而要求满足的准则或要求为真的条件。

[0837] 1.28.2. 高速缓存持续时间

[0838] ExtendedStatusBlock的CacheDuration（高速缓存持续时间）字段是在ExtendedStatusBlock中所编码信息的有效时段的指示。当ExtendedStatusBlock具有非零的有效时段时，这意味着ExtendedStatusBlock可以被存储在高速缓存器中，而且在该时段期间，对利用相同参数调用的确切相同的例程的调用会返回相同的ExtendedStatusBlock，因此高速缓存的值可以被返回到主机应用而不是调用例程。

[0839] 1.28.3. 参数

[0840] 一些参数用来传达关于返回状态以及用于模板处理的变量绑定的详细信息，（参见节9.4）。

[0841] 在一个实施例中，除义务和回调之外，这里所描述的所有约束严格地用于帮助主机应用分类和显示的目的，而并非用于强制实施使用规则。强制实施所述规则是控制程序的职责。

[0842] 在一个实施例中，在下面节中所定义的参数被编码为ParameterBlock（如果没有参数标志适用的话），或者被编码为ExtendedParameterBlock（如果一个或多个标志适用的话）。下面将描述代表性的标志：

[0843] 1.28.3.1. 描述

[0844] 参数名：Description（描述）

[0845] 参数类型 :ValueList(值列表)

[0846] 描述 :描述参数的列表。列表中的每个值是类型参数或扩展参数。在一个实施例中,定义以下参数 :默认、短和长。它们中的每个如果存在的话 . 那么具有控制资源之一的 ID 值。该资源应当包含文本有效载荷或模板有效载荷。如果资源是模板,那么它被处理以便获得结果的文本描述 (整个控制程序或特定动作的描述)。使用其中出现 '描述' 参数的列表中的其它参数作为变量绑定来处理模板。

[0847] 在一个实施例中,如果还包括 '默认' 描述,那么可以只包括 '短' 和 '长' 描述。

[0848]

名称	类型	描述
默认	资源	包含正常描述文本或模板的资源的 Id
短	资源	包含短描述文本或模板的资源的 Id
长	资源	包含长描述文本或模板的资源的 Id

[0849] 1.28.3.2. 约束

[0850] 在一个实施例中,在包含类似类型的约束的列表中把约束参数编组。在一个实施例中,为一些类型定义标准的约束。在一个实施例中,如果约束参数的名称在保证该名称唯一性的名称空间中为 URN,那么控制可以返回未包括在标准约束组内的约束参数。这可以包括销售商专用的约束或在其它规范中所定义的约束。

[0851] 1.28.3.2.1. 通用约束

[0852] 参数名 :GenericConstraints(通用约束)

[0853] 参数类型 :ValueList(值列表)

[0854] 描述 :可以适用的通用约束的列表。列表中的每个值属于类型参数或扩展参数。

[0855] 在一个实施例中,通用约束是不属于在此节中所定义的任何其它约束类型的约束。在一个实施例中,没有定义通用约束参数。

[0856] 1.28.3.2.2. 时间约束

[0857] 参数名 :TemporalConstraints(时间约束)

[0858] 参数类型 :ValueList(值列表)

[0859] 描述 :可以适用的时间约束的列表。列表中的每个值是类型参数或扩展参数。时间约束是与时间、日期、持续时间等相关的约束。在一个实施例中,定义以下时间约束参数 :

[0860]

名称	类型	描述
NotBefore	日期	在其之前动作被拒绝的日期
NotAfter	日期	在其之后动作被拒绝的日期
NotDuring	Value List	类型日期的 2 个值的列表。第一值是时段的开始，并且第二值是排除的那个时段的结束。
NotLongerThan	整数	在首次使用之后的最大秒数。在一个实施例中，此值是无符号的。
NotMoreThan	整数	累积使用时间的最大秒数。在一个实施例中，此值是无符号的。

[0861] 1.28.3.2.3. 空间约束

[0862] 参数名 :SpatialConstraints(空间约束)

[0863] 参数类型 :ValueList(值列表)

[0864] 描述 :可以适用的空间约束的列表。在一个实施例中，列表中的每个值属于类型参数或扩展参数。空间约束是与物理位置相关的约束。在一个实施例中，没有定义标准的空间约束。

[0865] 1.28.3.2.4. 组约束

[0866] 参数名 :GroupConstraints(组约束)

[0867] 参数类型 :ValueList(值列表)

[0868] 描述 :可以适用的组约束的列表。列表中的每个值是类型参数或扩展参数。组约束是与组、组成员资格、身份组等相关的约束。在一个实施例中，定义以下参数 :

[0869]

名称	类型	描述
MembershipRequired	资源	包含要求成员资格的组的名称或标识符的文本或模板的资源的 Id
IdentityRequired	资源	包含个人名称或标识符的文本或模板的资源的 Id

[0870] 1.28.3.2.5. 设备约束

[0871] 参数名 :DeviceConstraints(设备约束)

[0872] 参数类型 :ValueList(值列表)

[0873] 描述 :可以适用的设备约束的列表。列表中的每个值属于类型参数或扩展参数。设备约束是与设备的特性相关的约束，所述特性诸如特征、属性、名称、标识符等。在一个实施例中，定义以下参数 :

[0874]

名称	类型	描述
DeviceTypeRequired	资源	包含所要求的主机设备类型的文本或模板的资源的 Id
DeviceFeatureRequired	资源	包含主机设备必须具有的特征名称的文本或模板的资源的 Id
DeviceIdRequired	字符串	要求设备具有的 Id。此 Id 可以是可以用来标识设备的任何字符串（例如，设备名称、设备序列号、节点 id 等）。

[0875] 1.28.3.2.6. 计数器约束

[0876] 参数名 :CounterConstraints（计数器约束）

[0877] 参数类型 :ValueList（值列表）

[0878] 描述 :可以适用的计数约束的列表。列表中的每个值属于类型参数或扩展参数。计数约束是与计数值相关的约束，诸如播放计数、累积计数等。在一个实施例中，没有定义标准的计数约束。

[0879] 1.28.3.3. 参数标志

[0880] 在一个实施例中，以下标志当被编码为 ExtendedStatusBlock 时可以用于在节 9.2.3 中所描述的所有参数 :

[0881]

比特索引（0 为 LSB）	名称	描述
0	CRITICAL	与此参数相关联的语义需要被主机应用所理解。如果它们不被理解，那么整个 ExtendedStatusBlock 应当被当作不理解对待并且被拒绝。 在一个实施例中，此标志不应当用于实际上是描述性的参数。
1	HUMAN_READABLE	此参数表示其名称和值适于在文本或图形用户接口中显示的值。没有此标志集的任何参数应当被保留给主机应用，并且不显示给用户。对于类型资源的参数值来说，不是资源 ID 而是由所述 ID 所引用的资源数据有效载荷是人类可读的。

[0882] 1.29. 义务和回调

[0883] 在一个实施例中，某些动作当被准许时要求来自主机应用的进一步参与。义务表示当使用主机应用请求的内容密钥时或在此之后需要由所述主机应用执行的操作。回调表示对一个或多个控制程序例程的调用，所述控制程序例程当使用所请求的内容密钥时或在此之后需要被主机应用执行。

[0884] 在一个实施例中，如果应用遇到它不支持或者不理解的任何关键的义务或回调（例如因为所述义务类型可能在执行应用之后才定义），那么它必须拒绝继续执行为其返回此义务或回调参数的动作。在一个实施例中，通过为用于描述关键的义务或回调的参数设置 CRITICAL 参数来表明所述关键的义务或回调。

[0885] 如果控制具有副作用（诸如减小播放计数），那么它应当使用 OnAccept 回调来要求主机应用调用某一例程（如果它能够理解并符合所有关键的义务和回调的话）。所述副作用应当发生在回调例程中。在一个示例性实施例中，实现方式被要求来理解并实现 OnAccept 回调，这是因为它可以用于防止副作用（例如对状态数据库的更新）提前出现（例如在主机应用确定它不能符合给定的关键义务或回调并且需要终止动作执行之前），从而提供事务原子性的量度。

[0886] 1.29.1. 参数

[0887] 以下参数定义了可以在 ExtendedStatusBlock 数据结构中返回的几种类型的义务和回调。

[0888] 1.29.1.1. 义务

[0889] 参数名 :Obligations

[0890] 参数类型 :ValueList

[0891] 描述：义务参数的列表。列表中的每个值属于类型参数或扩展参数。在一个实施例中，定义以下义务参数：

[0892]

名称	类型	描述
RunAgentOnPeer	ValueList	主机应用需要发送代理控制以便在目前运行的协议会话的对等体上运行

类型	描述
字符串	包含要运行的代理的控制的 Id
字符串	要运行的代理的名称
整数	实例 Id。此值用来唯一地标识此代理义务实例。此 id 使系统能够把此代理义务与 OnAgentCompletion 回调参数相关。
字符串	上下文 Id。此 Id 可被在代理会话上下文主机对象路径：Octopus/Agent/Parameters/Session/ContextId 下的对等体上的运行代理看见。
ValueList	类型参数值的列表。所有那些参数可作为输入参数被代理看见。

[0893] 1.29.1.2. 回调

[0894] 参数名：Callbacks（回调）

[0895] 参数类型：ValueList

[0896] 描述：回调参数的列表。列表中的每个值属于类型参数或扩展参数。在一个实施例中，定义以下回调参数：

[0897]

<table>
<tr><th>名称</th><th>类型</th><th>描述</th></tr>
<tr><td>OnAccept</td><td>Callback</td><td>如果主机应用能够理解在此 ESB 中所包含的所有关键的义务和回调参数，那么主机应用必须进行回调。
在一个实施例中，在回调参数列表中存在至多一个 OnAccept 回调参数。如果在列表中指定其它回调参数，那么首先执行 OnAccept。</td></tr>
<tr><td>OnTime</td><td>ValueList</td><td>主机应用必须在所指定的日期/时间之后进行回调。
<table>
<tr><th>类型</th><th>描述</th></tr>
<tr><td>日期</td><td>在此之后主机应用需要执行回调的日期。</td></tr>
<tr><td>回调</td><td>用于回调的例程以及相关联的 cookie（点心文件）。</td></tr>
</table></td></tr>
</table>

[0898]

<table>
<tr><td>OnTimeElapsed</td><td>ValueList</td><td>主机应用必须在所指定的持续时间已经过去之后进行回调（当所述主机应用实际上执行被准许权限的动作时开始计数）。<table><tr><td>类型</td><td>描述</td></tr><tr><td>整数</td><td>秒数。该值是无符号的。</td></tr><tr><td>回调</td><td>用于回调的例程以及相关联的 cookie。</td></tr></table></td></tr>
<tr><td>OnEvent</td><td>ValueList</td><td>当出现某一事件时主机应用必须进行回调。<table><tr><td>类型</td><td>描述</td></tr><tr><td>字符串</td><td>事件名</td></tr><tr><td>整数</td><td>事件标志（整数 valus 被解释为比特字段）</td></tr><tr><td>整数</td><td>事件参数</td></tr><tr><td>回调</td><td>用于回调的例程以及相关联的 cookie。</td></tr></table>对于关于事件的更多细节，参见关于所述事件的段落。</td></tr>
<tr><td>OnAgentCompletion</td><td>ValueList</td><td>当在义务参数之一中所指定的代理已经完成或者没能运行时主机应用必须进行回调。<table><tr><td>类型</td><td>描述</td></tr><tr><td>整数</td><td>代理实例 Id。
在代理义务中所指定的实例 id。</td></tr><tr><td>回调</td><td>用于回调的例程以及相关联的 cookie。</td></tr></table>当进行回调时，主机应用必须提供以下 ArgumentsBlock（自变量块）：</td></tr>
</table>

[0899]

		<table><tr><td>类型</td><td>编码</td><td>描述</td></tr><tr><td>32 比特整数</td><td>依照大端次序的 4 个字节</td><td>完成状态代码</td></tr><tr><td>32 比特整数</td><td>依照大端次序的 4 个字节</td><td>代理结果代码</td></tr><tr><td>8 比特字节数组</td><td>字节序列</td><td>代理 ReturnBlock</td></tr></table> 。 如果代理能够运行那么完成状态代码值为 0，否则如果不能运行那么完成状态代码值为负错误代码。 代理 ReturnBlock 是代理所返回的数据。如果代理不能运行那么省略此数据（完成状态代码不为 0）。

[0900] 在一个实施例中，在上表中所提及的‘回调’类型是具有三个 ValueBlock（值块）元项的 ValueListBlock（值列表块）：

[0901]

<table>
<tr><td>值类型</td><td>描述</td></tr>
<tr><td>整数</td><td>回调类型的 ID。在一个实施例中，定义了两种类型的回调：
<table>
<tr><th>ID</th><th>描述</th></tr>
<tr><td>RESET = 0</td><td>当调用回调例程时取消所有待决的回调请求和活动义务。回调例程返回用于表明应用是否以及怎样继续当前操作的 ESB。</td></tr>
<tr><td>CONTINUE =1</td><td>当所有其它待决的回调请求和活动义务保持有效时调用回调例程。回调例程返回简单的结果代码。应用可以继续当前操作，除非该结果代码表明已经失败了。</td></tr>
</table></td></tr>
<tr><td>字符串</td><td>代码模块中用于调用的入口点。在一个实施例中，对于与包含返回具有此参数的 ESB 的例程的控制相同的控制来说，这必须是代码模块的输出表中的入口之一。</td></tr>
<tr><td>整数</td><td>Cookie。此值会被传递到所调用的例程的堆栈上。</td></tr>
</table>

[0902] 1.29.1.3. 参数标志

[0903] 在一个实施例中，使用与在先前节中所定义的相同的参数标志。在一个实施例中，要求调用方实现的回调和义务被标记为 CRITICAL，以避免向主机应用给予忽略这些参数的选择。

[0904] 1.29.2. 事件

[0905] 在一个实施例中，由名称来指定事件。取决于事件类型，可以定义用于进一步指定所述事件的一组标志。在一个实施例中，如果没有为具体事件定义标志，那么标志字段值被设置为 0。一些事件还可以指定当出现所述事件时向回调例程提供一些信息。在一个实施例中，如果并不从所述主机应用要求任何特殊信息，那么所述主机应用必须利用空 ArgumentsBlock（自变量块）来调用（参见下面节 3.3 中回调例程接口的描述）。

[0906] 在一个实施例中，如果主机应用不理解或不支持在被标记为 CRITICAL 的回调参数中的事件名称，那么所述主机应用必须把此参数认为是不理解的 CRITICAL 参数（并且不能执行对其请求权限的动作）。

[0907] 在一个实施例中，定义以下事件名：

[0908]

事件名	事件标志	事件参数	描述
OnPlay	无	无	当多媒体对象开始播放时主机应用必须进行回调。
OnStop	无	无	当多媒体停止播放（或暂停）时主机应用必须进行回调
OnTimecode	无	用自从开始呈现起的秒数来	当已经到达或超过所指定的呈现时间时（在正常的实时播放期间或在寻找之后）主机应用必须进行回调。呈现时间

[0909]

<table>
<tr><td></td><td></td><td>表示的呈现时间</td><td>的起点是再现开始时。呈现时间涉及源媒体时间，而不是墙上挂钟的时间（例如，当呈现暂停时，呈现时间并不改变）。</td></tr>
<tr><td>OnSeek</td><td>无</td><td>无</td><td>当发生对多媒体呈现中的任意点进行直接访问时主机应用必须进行回调。
在一个实施例中，当回调时，主机应用必须在 ArgumentsBlock 中提供以下数据：
<table>
<tr><th>类型</th><th>编码</th><th>描述</th></tr>
<tr><td>32 比特无符号整数</td><td>依照大端次序的 4 个字节</td><td>寻找位置偏移</td></tr>
<tr><td>32 比特无符号整数</td><td>依照大端次序的 4 个字节</td><td>寻找位置范围</td></tr>
</table>
多媒体呈现内的位置是在所述呈现中的总“标记”之外的偏移“标记”。例如，对于 327 秒长的呈现来说，可以利用偏移= 60、范围= 327 来表示寻找第 60 秒。如果在整个呈现过程中均匀地分布“标记”，那么要由调用方才选择对应于所述偏移和范围的测量值的单位（它可以是时间单位、字节大小单位或任何其它单位）。偏移值必须小于或等于范围值。</td></tr>
</table>

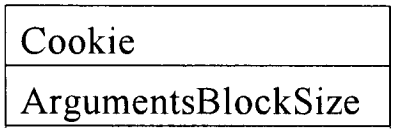
[0910] 1.29.3. 回调例程

[0911] 在一个实施例中，回调例程获取相同的输入：

[0912]

Input: 堆栈顶部:

[0913]



[0914] Cookie :在回调参数中所指定的 Cookie 字段的值。

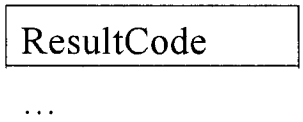
[0915] ArgumentsBlockSize :在此参数下在堆栈上所传递的数据字节数目。当调用例程时,所述堆栈包含由调用方所提供的值 ArgumentsBlockSize(自变量块大小),后面是数据字节 ArgumentsBlockSize,所述 ArgumentsBlockSize 表明在顶部的自变量块大小。在一个实施例中,如果所述大小不是 4 的倍数,那么堆栈上的数据用 0 值字节来填充以便确保堆栈指针保持为 4 的倍数。

[0916] 1.29.3.1. 继续回调

[0917] 在一个实施例中,具有类型 CONTINUE(继续)(类型 ID = 0) 的回调具有以下输出 :

[0918]

Output: 堆栈顶部:



[0919] ResultCode :整数值。如果例程能够执行那么结果值为 0,或者如果出现错误那么结果值为负错误代码。

[0920] Description :如果 ResultCode 表明回调例程能够运行 (即所述值为 0),那么主机应用可以继续当前操作。如果 ResultCode 表明出现错误,那么主机应用中止当前操作并且取消所有待决的回调和义务。

[0921] 1.29.3.2. RESET(复位) 回调

[0922] 当控制例程已经在从例程所返回的 ESB 中指定了一个或多个类型为 RESET 的回调时,当满足用于该回调的条件时主机应用会调用任何指定的回调例程。在一个实施例中,一旦满足任何回调的条件,主机应用就需要 :

[0923] ●取消所有其它待决的回调

[0924] ●取消所有当前义务

[0925] ●向该回调提供所要求的参数 (如果存在任何的话)

[0926] ●调用所指定的回调例程。

[0927] 来自例程的返回状态向主机应用表明它是否可以继续执行当前操作。在一个实施例中,如果权限被拒绝或者例程没能成功地执行,那么主机应用必须中止当前操作的执行。类似地,如果权限被准许,那么主机应用必须遵守可能在 ESB 中返回的任何义务或回调,就像它已经调用原始的 Control.Actions.<Action>.Perform routine 一样。先前的义务或回调规范不再有效。

[0928] 在一个实施例中,被指定为用于此类回调的回调入口点的所有例程具有以下输

出：

[0929]

Output: 堆栈顶部:

ResultCode
StatusBlockPointer
...

[0930] ResultCode：整数值。如果例程能够执行那么结果值为0，或者如果出现错误那么结果值为负错误代码。

[0931] StatusBlockPointer：标准的扩展状态块的地址。

[0932] 描述：此例程的返回语义等价于对Control.Actions.<Action>.Perform例程所描述的语义。

[0933] 1.30. 元数据资源

[0934] 在一个实施例中，控制对象可以包含元数据资源，其可以从在ExtendedStatusBlock数据结构中所返回的参数来引用。资源可以是简单的文本、文本模板或其它数据类型。每个资源由资源ID标识，并且可以包含一个或多个文本串或编码数据，不同语言的每种版本一个。不要求为所有语言都提供资源。由主机应用选择哪种语言版本是最适于其需要。

[0935]

资源		
字段	类型	描述
Id	ASCII 字符串	URI（典型情况下为 URN，它

[0936]

		的是控制对象的扩展的 Id，所述控制对象的扩展包含具有目前运行例程的代码模块）
Type	ASCII 字符串	如 IETF RFC 2046 中所述的资源数据的 MIME 类型
Data	LocalizedData 的列表	对于不同场所来说，资源的所有不同版本的列表

LocalizedData		
字段	类型	描述
Language	ASCII 字符串	如在 IETF RFC 3066 中所指定的语言代码
Data	类型取决于所指定的 mime 类型	用于资源（文本等）的实际数据

[0937] 资源通过作为扩展包括在控制对象中而伴随控制程序。资源 Id 映射到控制对象的内部扩展的 Id，所述控制对象的内部扩展包含具有目前运行的例程的代码模块。

[0938] 为了计算资源对象的规范字节序列，在一个实施例中数据结构描述如下：

[0939]

```
class LocalizedData {
        string language
        byte[] data;
}

class Resource {
   string id
   string type;
   LocalizedData data;
}
```

[0940] 1.30.1. 简单文本

[0941] 简单文本被指定为 MIME 类型‘文本’

[0942] 1.30.2. 文本模板

[0943] 除标准的文本资源之外，在一个实施例中，定义了文本模板类型。用于此文本模板

类型的 MIME 类型是‘text/vnd.intertrust.octopus-text-template’。

[0944] 在一个实施例中，文本模板包含依照 UTF-8 编码的文本符号以及命名占位符(placeholder)，所述占位符将由从在参数列表中所返回的参数所获得的文本值来代替，诸如 ExtendedStatusBlock 的。用于占位符的语法是‘\PLACEHOLDER\’，其中 PLACEHOLDER 指定参数块和可选格式化提示的名称。在一个实施例中，模板处理器必须利用该参数块的值字段的格式化表示来替换整个令牌‘\PLACEHOLDER\’，并且下面在节 4.2.1 指定了对值数据的格式化。

[0945] 在一个实施例中，如果符号‘\’出现在占位符之外的文本中，那么它必须被编码为‘\\’，并且在文本中所有出现的‘\\’会被模板处理器恢复为‘\’。

[0946] 用于占位符的语法为 :FORMAT|NAME，其中 NAME 是参数块的名称，并且 FORMAT 是用于把参数数据转换为文本的格式化提示。如果用于参数值数据类型的默认格式化规则就足够了，那么可以省略格式化提示，并且占位符简单地为 NAME。

[0947] 1.30.2.1. 格式化

[0948] 1.30.2.1.1. 默认格式化

[0949] 在一个实施例中，用于不同数值类型的默认格式化规则为 :

[0950]

类型	格式化
整数	整数值作为带符号十进制数的文本表示。文本只由字符数字“0”到“9”和符号“-”组成。如果值为 0，那么文本为字符串“0”。如果值不为 0，那么文本不以字符“0”开始。如果值为负，那么文本以字符“-”开始。如果值为正，那么文本以非零数字字符开始。
实数	浮点值用十进制的文本表示。使用与整数值相同的规则

[0951]

	来表示值的整数部分。利用主机应用的优选十进制分隔符来表示十进制分隔符。值的小数部分由多达 6 个“0”字符后面是多达 3 个非零数字字符组成。
字符串	字符串值本身
日期	依照主机优选的日期文本表示，人类可读的日期表示
参数	文本“<name> = <value>”，其中<name>为参数名，并且<value>为依照其类型的默认格式化规则格式化的参数值。
ExtendedParameter	与参数相同
资源	资源数据的文本串。在一个实施例中，由占位符所引用的资源必须具有基于文本的 MIMI 类型（例如，文本或 text/vnd.intertrust.octopus-text-template）。
ValueList	文本“<value>，<value>，...”，列表中具有所有值依照它们类型的默认格式化规则被格式化。

[0952] 1.30.2.1.2. 显式格式化

[0953] 显式格式名称可以被用为占位符标签的 FORMAT 部分。如果遇到未知的 FORMAT 名称，那么模板处理引擎使用默认格式化规则。

[0954]

名称	格式化
十六进制	被解释为无符号的整数值的十六进制表示。在一个实施例中，对于不是整数的数据类型，应当忽略此格式化提示。

[0955] 1.31. 上下文对象

[0956] 在一个实施例中，当执行控制例程时，它有权通过使用 System.Host.GetObject 系统调用来访问多个上下文对象。

[0957] 1.31.1. 通用上下文

[0958] 在一个实施例中，给出以下上下文情境以用于运行控制。

[0959]

名称	类型	描述
Octopus/Personality/Id	字符串	当前个性节点的 ID
Octopus/Personality/Attributes	属性的容器	当前个性节点的属性

[0960] 1.31.2. 运行时上下文

[0961] 在一个实施例中，对于在 VM 中运行的所有控制给出以下上下文情境，其中已经使

用 System.Host.SpawnVm 系统调用创建了所述 VM。在一个实施例中，此上下文情境必须是不存在的或者是用于控制的空容器，所述控制在不是使用 System.Host.SpawnVm 创建的 VM 中运行。

[0962]

名称	类型	描述
Octopus/Runtime/Parent/Id	未命名的字符串对象的容器	系统调用的调用方以之来运行的身份。

[0963] 1.31.3. 控制上下文

[0964] 在一个实施例中，每当运行控制的例程时给出以下上下文情境：

[0965]

名称	类型	描述
Octopus/Control/Id	字符串	运行控制的 Id
Octopus/Control/Attributes	容器	运行控制的属性。如果控制没有属性，那么可以省略此对象。

[0966] 1.31.4. 控制器上下文

[0967] 在一个实施例中，每当控制的例程正在运行并且由控制器对象指向所述控制时（例如，当访问内容密钥对象以便消费受保护的内容时），给出以下上下文情境。

[0968]

名称	类型	描述
Octopus/Controller/Id	字符串	指向目前运行的控制的控制器的 Id
Octopus/Controller/Attributes	容器	指向目前运行的控制的控制器的属性。如果控制器没有属性，那么可以省略此对象。

[0969] 在其中允许主机应用只编组由单个控制器对象所控制的内容密钥的实施例中，对于给定动作，只存在一个适用的控制器对象。

[0970] 1.31.5. 动作上下文

[0971] 在一个实施例中，每当为了控制动作目的调用控制时给出以下下上下文情境。

[0972]

名称	类型	描述
Octopus/Action/Parameters	容器	用于表示与当前动作相关的参数的名称/值对的数组，如果存在的话。在一个实施例中，每个动作类型定义了可选和要求的参数的列表。如果动作没有参数那么可以省略此容器。

[0973] 1.31.6. 链路上下文

[0974] 在一个实施例中，每当为了限制链路对象的有效性而调用控制时（例如被嵌入到链路对象中的控制对象），给出以下上下文情境：

[0975]

名称	类型	描述
Octopus/Link/Id	字符串	链路对象的 Id
Octopus/Link/Attributes	容器	包含运行的控制的链路对象的属性。如果链路没有属性，那么可以省略此对象。

[0976] 1.31.7. 代理上下文

[0977] 在一个实施例中，每当运行控制的代理例程时给出以下下上下文情境：

[0978]

名称	类型	描述
Octopus/Agent/Parameters	容器	用于表示代理的输入参数的名称/值参数对的数组。
Octopus/Agent/Session/ContextId	字符串	其中正运行代理的会话上下文情境的标识符。

[0979] 参数和会话容器通常用于允许这样的协议，所述协议要求一个实体发送并运行另一实体上的代理以便指定哪些输入参数传递到所述代理，以及在某些条件下主机需要设置哪些会话上下文对象。某些会话上下文对象的存在或不存在可以使代理代码能够判定它是否作为它被设计成所支持的协议的一部分运行，或者它是否在上下文之外运行，在这种情况下它可以拒绝运行。例如，其目的在于在它运行的主机上创建状态对象的代理可以拒绝运行，除非它在特定的协议交互期间被执行。

[0980] 1.32. 动作

[0981] 在一个实施例中，每个动作具有了名字和参数列表。在一个实施例中，需要一些参数－应用当执行此动作时必须提供它们－并且一些是可选的－应用可以提供它们或者可以省略它们。

[0982] 在一个实施例中，定义以下标准动作：

[0983] 1.32.1. 播放

[0984] 描述：多媒体内容的正常实时播放。

[0985] 1.32.2. 转送

[0986] 描述：转送到兼容的目标系统。

[0987] 当必须使内容可用于具有相同 DRM 技术的系统时，使用转送到兼容的目标系统，以致所述目标系统可以使用与包含此控制相同的许可，但是在源、信宿 (sink) 或它们二者上可能需要改变状态信息。从其进行转送的系统被称为源。向其进行转送的目标系统被称为信宿。

[0988] 此动作旨在结合服务协议来使用，所述服务协议使代理能够被从源转送到信宿以便在源和信宿持久状态（例如，这里所描述的状态数据库中的对象）中进行必要的更新。在一个实施例中，为了该目的，控制使用 RunAgentOnPeer 义务。下面结合论述状态数据库来提供关于此服务协议的说明性实施例的附加信息。

[0989] 参数：

[0990]

名称	类型	描述
Sink/Id	字符串	信宿的节点 Id
Sink/Attributes	容器	信宿节点的属性。如果节点没有属性，那么可以省略此容器。
TransferMo	字符串	用于表明正转送内容时所处于的模式的转送模式

[0991]

<table>
<tr><td>de</td><td></td><td>ID。此 ID 可以是如下面所定义的标准模式，或者用于系统专有模式的 URN。
在一个实施例中，定义以下标准模式:<table><tr><th>ID</th><th>描述</th></tr><tr><td>再现</td><td>信宿接收内容以便再现</td></tr><tr><td>拷贝</td><td>信宿正接收内容的拷贝</td></tr><tr><td>移动</td><td>内容正被移到信宿。</td></tr><tr><td>签出（Check Out）</td><td>所述内容正被签出到信宿。这与移动类似，但是区别在于信宿上所产生的状态可以防止除移回到源之外的任何其它移动。</td></tr></table></td></tr>
<tr><td>TransferCount</td><td>整数</td><td>用于表明需要把与此控制相关联的状态计数器的多少实例转送到信宿的整数值。
在一个实施例中，此参数是可选的。如果不存在的话，那么只转送一个实例。当转送模式为再现或拷贝时，它不应当存在。</td></tr>
</table>

[0992] 1.32.3. 输出

[0993] 描述：到外部目标系统的输出。

[0994] 输出到外部目标系统是当必须把内容输出到其中无法使用原始的内容许可的系统时所使用的动作。这可以是具有不同 DRM 技术的系统、没有 DRM 技术的系统或具有相同的技术但是处于要求不同于原始许可的许可的情况下的系统。从中进行转送的系统被称为源。向其进行转送的目标系统被称为信宿。

[0995] 在一个实施例中，在用于此动作的描述、检查和执行方法的扩展状态结果中，应当设置以下参数：

[0996]

名称	类型	描述
ExportInfo	任何	当向在动作参数中所指定的目标系统输出内容时相关的信息。此信息的实际类型和内容是每个目标系统专用的。例如对于基于 CCI 的系统来说，这会包含为所输出的内容设置的 CCI 比特。

[0997] 参数：

[0998]

<table>
<tr><th>名称</th><th>类型</th><th>描述</th></tr>
<tr><td>TargetSystem</td><td>字符串</td><td>向其进行输出的外部系统的系统 ID。此 ID 是 URN。</td></tr>
<tr><td>ExportMode</td><td>字符串</td><td>用于表明正输出内容时所处的模式的输出模式 ID。此 ID 可以是如下面所定义的标准模式，或者用于系统专有模式的 URN。
在一个实施例中，定义以下标准模式:
<table>
<tr><th>ID</th><th>描述</th></tr>
<tr><td>DontKnow</td><td>调用方不知道信宿的预计模式是什么。在这种情况下，控制程序应当假定可以由信宿来假定对于 TargetSystem 来说所允许的任何模式，并且应当在动作例程的返回状态中表明任何限制。例如，对于基于 CCI 的系统来说，控制可以返回 CCI 比特，所述 CCI 比特根据许可所允许的来允许再现或拷贝的等效物。</td></tr>
<tr><td>Render</td><td>信宿正接收内容以便再现，并且除如由每个目标系统所指定的高速缓存目的之外不再保持所述内容的可用拷贝</td></tr>
<tr><td>Copy</td><td>信宿正接收内容的拷贝</td></tr>
<tr><td>Move</td><td>内容正被移到信宿。</td></tr>
</table></td></tr>
</table>

[0999] 特定任务系统可以要求其它输入参数。

[1000] 1.32.3.1. 标准的目标系统

[1001] 1.32.3.1.1. 音频 CD 或 DVD

[1002] 在一个实施例中，当目标系统是其上写有未压缩 PCM 音频的未加密介质（诸如可写音频 CD 或 DVD）时，使用标准的 TargetSystem ID‘CleartextPcmAudio’。对于此目标系统来说，ExportInfo 参数是用于表示版权标志的单整数参数。用整数值的最低有效位来表明此标志。

[1003]

位索引	描述
0 (LSB)	当此标志被设置时，如果重新编码的音频的格式支持版权位或标志的表示，那么必须依照所述格式来设置所述位或标志。

[1004] 10. 状态数据库

[1005] 下面将描述安全对象存储装置，其可以由 DRM 引擎的优选实施例用来提供安全状态存储机制。这种机构对于使控制程序能够在从调用到调用保持不变的受保护的状态数据库中进行读取和写入来说是有用的。这种状态数据库可以用来存储状态对象，诸如播放计数、首次使用日期、累积的再现时间等。在优选实施例中，在非易失性存储器中实现安全数据库，所述非易失性存储器诸如便携式设备上的闪速存储器或 PC 上的硬盘驱动器的加密区域。然而应当理解，可以在任何适当的介质上实现安全数据库。

[1006] 如在此节中所使用的术语“对象”通常指的是在安全对象存储装置内所包含的数据对象，而不是在此的其它地方所论述的对象（例如控制、控制器、链路等）；如果必须区分这两种类别的对象，那么术语“DRM 对象”将用来指的是在此的其它地方所描述的对象（即，控制、控制器、保护器、内容密钥、链路、节点等），而术语“状态对象”用来指的是在状态数据库内所存储的对象。在下面论述中，有时参考被称作“Seashell（海贝）”的状态数据库的说明性实现方式，其结合在此的其它地方所描述的章鱼 DRM 引擎实施例来使用。然而应当理解；这里所描述的系统和方法的实施例可以在没有此说明性实现方式的一些或所有特征的情况下实施。

[1007] 1.33. 数据库对象

[1008] 对象存储装置（例如数据库）包含数据对象。在一个实施例中，对象被布置在逻辑分层体系中，其中容器对象是它们所包含的孩子对象的亲代。在一个实施例中，存在四种类型的对象：字符串，整数，字节数组和容器。每个对象具有相关联的元数据和类型。取决于其类型，对象还可以具有值。

[1009] 在一个实施例中，可以使用 System.Host.GetObject 和 System.Host.SetObject 系统调用从虚拟机程序中访问状态对象，并且如下面所更详细地描述，可以使用虚拟名称来访问对象元数据。在一个实施例中，一些元数据字段可以由数据库的客户端改变（即，它们是可读写 (RW) 访问的），而其它元数据字段是只读的 (RO)。

[1010] 在一个实施例中，定义了在以下表中所示出的元数据字段：

[1011]

字段	类型	可访问性	描述
Name	字符串	RO	对象的名称。在一个实施例中，只允许以下字符作为对象名称（所有其它字符保留）：a-z，A-Z，0-9，‘_’，‘-’，‘+’，‘:’，‘.’，‘$’，‘!’，‘*’，‘’
Owner	字符串	RW	该对象所有者的 Id
CreationDate	无符号的 32 比特整数	RO	创建所述对象的时间，被表示为自从 1970 年 1 月 1 日 00:00:00 本地时间起所经过的分钟数。
ModificationDate	无符号的 32 比特整数	RO	最后修改所述对象的时间，被表示为自从 1970 年 1 月 1 日 00:00:00 本地时间起所经过的分钟数。 对于容器对象来说，这是把孩子最后添加到容器或从中删除的时间。对于其它对象来说，这是它们的值最后一次被改变的时间。
ExpirationDate	无符号的 32 比特整数	RW	所述对象期满的时间，被表示为自从 1970 年 1 月 1 日 00:00:00 本地时间起所经过的分钟数。0 值意味着对象没有期满。
Flags	32 比特的比特字段	RW	表明对象特性的布尔标志组

[1012] 在一个实施例中，定义了在以下表中所示出的元数据标志：

[1013]

位索引	名称	意义
0 (LSB)	PUBLIC_READ	如果设置的话，那么表明用于此对象的访问控制是这样的以致任何客户端可以读取所述对象及其元数据。

[1014] 如先前所表明，在一个实施例中存在四种类型的状态对象：字符串、整数、字节数组和容器。在此实施例中，字符串对象值是 UTF-8 编码的字符串，整数对象值是 32 比特整数值，并且字节数组对象值是字节数组。在此实施例中，容器对象包含零或更多个对象。容器对象被认为是它所包含对象的亲代。所包含的对象被认为是所述容器的孩子。构成对象亲代、亲代的亲代等链的所有容器对象被称作对象的祖先。如果对象具有另一对象作为它的祖先，那么该对象被称作祖先对象的后裔。

[1015] 1.34. 对象生存期

[1016] 在一个实施例中，状态数据库中对象的生存期遵循多个规则。对象可以被显式地销毁或隐式地销毁。对象还可能由于数据库无用单元的收集而被销毁。不管怎样销毁对象，在一个实施例中应用以下规则：

[1017] ●用于该对象的亲代容器的 ModificationDate 被设置为当前本地时间。

[1018] ●如果对象是容器，那么当所述对象被销毁时其所有孩子被销毁。

[1019] 1.34.1. 显式对象销毁

[1020] 当数据库的客户端请求移除对象时（参见对象访问来更详细地了解这可以怎样使用 Host.SetObject 系统调用来进行），发生显式对象销毁。

[1021] 1.34.2. 隐式对象销毁

[1022] 当对象由于其祖先中的一个对象被销毁而被销毁时发生隐式的对象销毁。

[1023] 1.34.3. 无用单元收集

[1024] 在一个实施例中，状态数据库销毁已经期满的任何对象。当在实现数据库的系统上的本地时间迟于对象元数据的 ExpirationDate 字段时，所述对象被认为是已经期满。实现方式可以定期地扫描数据库来寻找期满的对象并销毁它们，或者它可以等待直到对象被访问以检查其期满日期。在一个实施例中，实现方式不能向客户端返回期满的对象。在一个实施例中，当容器对象被销毁时（例如，因为它已经期满），其孩子对象也被销毁（递归地，它们所有的后裔也被销毁），即便它们尚未期满也是如此。

[1025] 1.35. 对象访问

[1026] 在一个实施例中，可以经由以下一对系统调用来从虚拟机程序中访问状态数据库中的对象：用于读取对象值的 System.Host.GetObject 和用于创建、销毁或设置对象值的 System.Host.SetObject。

[1027] 在一个实施例中，为了作为主机对象树被可见，状态数据库被“安装”在主机对象树中某个名称之下。这样，数据库作为主机对象的更通用树中的子树是可见的。为了实现这点，在一个实施例中，状态数据库包含始终存在的顶层级的、内置的根容器对象。此根容器实质上是数据库的名称。数据库中的所有其它对象是根容器的后裔。多个状态数据库可以被安装在主机对象树中的不同位置（对于将被安装在相同主机容器下的两个数据库来说，

它们需要具有不同的名称来用于它们的根容器）。例如，如果其根容器名称为 Database1 的状态数据库包含名称为 Child1 的单整数孩子对象，那么该数据库可以被安装在主机对象容器“/SeaShell”下，在这种情况下，Child1 对象会做为“/SeaShell/Database1/Child1”而可见。在一个实施例中，按照访问策略来管理支配对状态数据库中对象的访问。

[1028] 1.35.1. 读取对象

[1029] 可以通过使用系统调用 System.Host.GetObject 来读取对象值。在状态数据库的一个实施例中，可以存在于数据库中的四个对象类型（整数，字符串，字节数组和容器）直接映射到它们虚拟机中的对应物上。可以采用正常方式来访问对象值，并且可以实现标准的虚拟名称。

[1030] 1.35.2. 创建对象

[1031] 对于已经不存在的对象名称可以调用 System.Host.SetObject 来创建对象。依照系统调用规范来进行对象创建。在一个实施例中，当创建对象时，状态数据库进行以下操作：

[1032] ●把对象元数据的“所有者”字段设置为亲代容器对象的元数据的“所有者”字段的值。

[1033] ●把元数据的 CreationDate 字段设置为当前的本地时间。

[1034] ●把元数据的 ModificationDate 字段设置为当前的本地时间。

[1035] ●把元数据的 ExpirationDate 字段设置为 0（没有期满）。

[1036] ●把元数据的标志字段设置为 0。

[1037] ●把亲代容器的 ModificationDate 设置为当前的本地时间。

[1038] 当在比现有容器分层体系更深的路径下创建对象时，在一个实施例中，状态数据库隐式地创建容器对象，所述容器对象需要存在以便创建通向被创建对象的路径。在一个实施例中，隐式的容器对象创建遵循与显式创建相同的规则。例如，如果存在没有孩子的容器“A”，那么在创建“A/B/C/SomeObject”之前用于设置“A/B/C/SomeObject”的请求会隐式地创建容器“A/B”和“A/B/C”。

[1039] 1.35.3. 写入对象

[1040] 对于已经存在的对象名称可以通过调用 System.Host.SetObject 来改变对象值。如果所指定的 ObjectType 不匹配现有对象的类型 ID，那么返回 ERROR_INVALID_PARAMETER。在一个实施例中，如果类型 ID 为 OBJECT_TYPE_CONTAINER，那么不需要指定值（ObjectAddress 必须是非零的，但是其值会被忽略）。当设置现有对象时，状态数据库把对象的 ModificationDate 设置为当前的本地时间。

[1041] 1.35.4. 销毁对象

[1042] 对已经存在的对象，可以通过利用 ObjectAddress 值 0 来调用 System.Host.SetObject 来显式地销毁对象。当对象被销毁时，状态数据库优选：

[1043] ●把亲代容器的 ModificationDate 设置为当前的本地时间。

[1044] ●如果所销毁的对象是容器，那么销毁所有其孩子对象。

[1045] 1.35.5. 对象元数据

[1046] 在一个实施例中，通过利用虚拟名称使用 System.Host.GetObject 和 System.Host.SetObject 系统调用来访问状态数据库对象的元数据。下表列出了在状态数据库的一

个实施例中可用于对象的标准和扩展虚拟名称，并且列出了它们怎样映射到元数据字段。

[1047]

虚拟名称	类型	描述
@Name	字符串	对象元数据的名称字段
@Owner	字符串	对象元数据的所有者字段
@CreationDate	32 比特无符号整数	对象元数据的 CreationDate 字段
@ModificationDate	32 比特无符号整数	对象元数据的 ModificationDate 字段
@ExpirationDate	32 比特无符号整数	对象元数据的 ExpirationDate 字段
@Flags	32 比特的比特字段	对象元数据的标志字段

[1048] 在一个实施例中，如果一个或多个未定义的标志被设置为 1，那么实现方式必须拒绝设置标志元数据字段的请求。在这种情况下，对 System.Host.SetObject 的返回值为 ERROR_INVALID_PARAMETER。在一个实施例中，当读取标志元数据字段时，客户端必须忽略未被预定义的任何标志，并且当设置对象的标志字段时，客户端必须首先读取其现有值并且保留未被预定义的任何标志的值（例如，依照系统规范）。

[1049] 1.36. 对象所有权和访问控制

[1050] 在一个实施例中，每当进行读取、写入、创建或销毁对象的请求时，状态数据库实现方式首先检查调用方是否有权限来执行所述请求。用于管理支配对对象进行访问的策略是基于委托人身份和委托的原理。为了要实现的该策略，实现方式操作时所处于的信任模型需要支持认证的控制程序的概念。这一般通过具有虚拟机代码模块并且具有用于使委托人名称与签名密钥相关联的名称证书来完成，所述虚拟机代码模块包含利用 PKI 密钥对的私钥来（直接或经由安全引用间接地）数字签名的程序；然而应当理解，用于确定控制程序身份的不同方式是可以的，可以使用其中的任何合适的一个。

[1051] 在一个实施例中，用于状态数据库中对象的访问策略由几个简单规则组成：

[1052] ●如果调用方身份与对象的所有者相同或者如果在对象的标志元数据字段中设置有 PUBLIC_READ 标志，那么准许对对象值的读取访问。

[1053] ●如果调用方有权对对象的亲代容器进行读取访问，那么准许对对象值进行读取访问。

[1054] ●如果调用方的身份与对象的所有者相同，那么准许对对象值进行写入访问。

[1055] ●如果调用方有权对对象的亲代容器进行写入访问，那么准许对对象值进行写入访问。

[1056] ●如果调用方有权对对象的亲代容器进行写入访问，那么准许创建或销毁对对象的访问。

[1057] ●（使用虚拟名称）对对象的元数据进行读取和写入访问遵循与对对象值进行读取和写入访问相同的策略，只是另外限制不能写入只读字段。

[1058] 在一个实施例中，当访问策略拒绝客户端请求时，对该请求的系统调用的返回值是 ERROR_PERMISSION_DENIED。

[1059] 优选，当创建数据库时，状态数据库的根容器是固定的。当创建对象时，其所有者元数据字段的值被设置为与其亲代容器所有者元数据字段的值相同的值。对象的所有权可以改变。为了改变对象的所有权，可以通过调用该对象的‘@Owner’虚拟名称的 Sytem.Host.SetObject 系统调用来设置所有者元数据字段的值（如果在访问控制规则下允许的话）。

[1060] 在控制程序不可以访问那些对象（所述对象并不由与控制程序正在其身份下运行的委托人是相同的委托人所拥有）的实施例中，控制程序需要把对“外部”对象的访问委托给从下述代码模块所加载的程序，所述代码模块具有在“外部”对象的所有者身份下运行的能力。为此，控制程序可以在控制虚拟机中使用 System.Host.SpawnVm、System.Host.CallVm 和 System.Host.ReleaseVm 系统调用。

[1061] 1.37. 许可转送协议

[1062] 把状态信息存储在诸如上述的数据库中使权利能够在设备之间移动或者从域中输出（例如，通过把状态信息转送到另一设备）。以下章节描述了协议的实施例，借此可以把数据库的状态从源转送到信宿。注意，尽管此过程被认为是许可转送协议，转送的是状态数据库的状态，这与纯粹的实际许可（例如控制对象等）的转送不同。所述协议被认为是许可转送协议，这是因为在一个实施例中，所述转送通过在控制程序中执行转送动作发起的，并且因为转送所述状态信息使信宿能够成功地执行与内容相关的许可。

[1063] 图 32 示出了由三个消息 3202、3204、3206 组成的许可转送 3200 的例子。在图 32 所示出的例子中，由信宿 3210 通过向源 3212 发送请求 3202 来启动该协议。在一个实施例中，请求 3202 保持要转送的内容的 ID。源 3212 向信宿 3210 发送响应 3204，包含 (i) 将在信宿 3210 的状态数据库中设置状态的代理，以及 (ii) 目标为信宿 3210 的内容密钥对象。如图 32 所示，信宿 3210 向源 3212 发送用于表示代理已经运行的确认 3206。当接收到内容密钥和 / 或内容时，信宿然后可以依照其相关联的控制来使用所述内容（例如，通过扬声器播放它，在视频屏幕上显示它和 / 或依照其它方式再现它）。

[1064] 虽然在一些实施例中可以使用在图 32 中所示出的方法，但是一些潜在的问题包括：

[1065] 没有办法主动告诉源再现已经结束了。在一个实施例中，在图 32 中所示出的协议支持两个模式，其中这是一个问题：(i) 再现（没有停止再现），和 (ii) 签出（没有签入 (checkin)）。由于此问题，可能导致控制发出方发出关于所转送的状态的超时。然而，这可能使消费者感觉很糟，例如当用户想要在一个设备上再现内容但是判定她实际上想要在另一个设备上再现此内容：利用当前的设计，很可能在她能够在另一个设备上再现内容之前必须等待在第一设备上再现整个内容。如果所述内容相对较长（例如电影），那么这可能会令人不快。

[1066] 可能很难分析与请求中的内容 ID 相关联的许可。在一个实施例中，所述请求只包含内容 ID，并且源从其许可数据库中获取与所述内容 ID 相关联的许可。然而，此过程可能

易于出错，这是由于许可可以被存储在可拆卸介质上，并且在约定协议时，如果所述介质已经被拆卸，那么特定的许可可能就是不可用的。此外，即便该许可是可用的，那么在许可存储装置中查找所述许可也可能是麻烦的。还因为可能存在与一组内容 ID 相关联的多个许可，所以可能难于确定所分析的许可是否与在请求中所想要的许可相同。

[1067] 控制程序没有办法主动请求邻近度检查 (proximity check)。在一个实施例中，该组系统调用 / 回调 / 义务不支持控制用来请求邻近度检查对等体的方式。作为替代，控制可以只读取主机对象 Octopus/Action/Parameters/Sink/Proximity/LastProbe 的值，该值由应用在转送期间利用它从先前的邻近度检查协议执行中所获得的值来填充。在如果不需要这种邻近度检查那么可能希望避免邻近度检查的情况下（例如，如果已知信宿在某一域内），这可能是一个问题。

[1068] 对于该协议来说只存在三个回合。在图 32 所示出的实施例中，所述协议被限制为三个回合。这可能是一个严重的限制，这是由于所述协议不能处理 OnAgentCompletion 回调返回具有另一 RunAgentOnPeer 义务的扩展状态块的情况。此外，在完成协议之后，信宿实际上不知道所述协议是否已经成功。另外，邻近度检查需要发生在发送响应之前（参见先前问题），但是在源和信宿处于相同域的情况下不必如此。另外，在图 32 所示出的协议中，源在不知道内容密钥将是否被使用的情况下向信宿给出此内容密钥。

[1069] 在 ESB 中没有办法暗示需要许可转送。在图 32 所示出的实施例中，当 DRM 客户端评估许可（例如 Control.Actions.Play.Check）时，为了获得能够成功评估所述控制的状态，控制写入方没有容易的方式来暗示需要许可转送。

[1070] 源无法发起转送。在图 32 所示出的协议中，许可转送由信宿发起。可能希望源也能够发起转送。

[1071] 改进的实施例

[1072] 下述实施例可以解决或改良上述一些或全部问题。

[1073] 释放问题的解决方案。在一个实施例中，引入新的释放操作。当在请求中指定此操作时，转送模式 ID 被设置为释放。按照顺序，为了让客户端在再现 / 签出和释放操作之间相关，把可选元项 SessionId（会话 ID）添加到请求（参见下面章节）。在一个实施例中，当此元项存在时，它被反映在 SessionId 下的转送动作上下文情境的主机对象树中。

[1074] 信宿知道如果它将在拆卸消息（参见下文）中获得的扩展状态块包含参数，那么它必须在释放请求中发送此 SessionId：

[1075] 参数名 :SessionId

[1076] 参数类型 :字符串

[1077] 此参数的标志被设置为 CRITICAL。

[1078] 许可解析问题的解决方案（重制 (refactoring) 请求）。在一个实施例中，所述解决方案包括使信宿设备把许可捆束（一个或者多个）放入请求中使得实质上保证所述信宿和源执行相同的许可。在图 32 所示出的实施例中，用于请求的 XML 模式如下：

[1079]

```
<xs:complexType name="LicenseTransferRequestPayloadType">
  <xs:sequence>
    <xs:element ref="ContentIdList"/>
    <xs:element ref="Operation"/>
    <xs:element ref="oct:Bundle"/>
  </xs:sequence>
</xs:complexType>
```

[1080] 在 ContentIdList（内容 ID 列表）包含标识内容的内容 ID 的列表的情况下（每个曲目 / 流一个），操作包含许可转送操作类型，并且捆束包含请求者的个性节点和相关联的签名。

[1081] 为了避免上述许可解析问题，例如可以通过如下修正模式来把许可捆束包括在请求中：

[1082]

```
<!—新元项 →
<xs:element name="LicensePart" type="LicensePartType"/>
<xs:complexType name="LicensePartType">
  <xs:sequence>
    <xs:element ref="oct:Bundle" minOccurs=”0”/>
  </xs:sequence>
  <xs:attribute name="contentId" use="optional"/>        </xs:complexType>
<xs:element name="License" type="LicenseType"/>
```

[1083]

```
<xs:complexType name="LicenseType">
  <xs:sequence>
    <xs:element ref="LicensePart" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<!—修改的许可转送请求有效载荷类型-->
<xs:complexType name="LicenseTransferRequestPayloadType">
 <xs:sequence>
  <xs:element ref="License"/> <!—定义见上-->
   <xs:element ref="Operation"/>
   <xs:element ref="oct:Bundle"/>
   <xs:element name="SessionId" type=”xs:string” minOccurs=”0”/>
   <xs:element name="NeedsContentKeys” type=”xs:boolean” minOccurs=”0”/>
   </xs:sequence>
 </xs:complexType>
```

[1084] 在此模式中，ContentIdList（内容 ID 列表）元项由许可元项来代替。此元项携带一组 LicensePart 元项。LicensePart 元项携带 oct :Bundle 元项，所述 oct :Bundle 元项包含许可对象以及用于表明所述许可对象被应用于此特定 ContentId（内容 ID）的可选 ContentId 属性。没有 ContentId 属性的 LicensePart 元项意味着在基础捆束中所包含的对象被应用于所有内容 ID（通常为控制器和控制对象）。

[1085] 在一个实施例中，SessionId 可选元项不能存在，除非如果所述操作为 urn : marlin :core :1-2 :service :license-transfer :release，在这种情况下如果在相应的再现或签出动作的扩展状态块中接收了 SessionId 参数那么它可以存在（参见上面）。

[1086] 在一个实施例中，如果信宿知道它已经能够解密内容密钥，那么 NeedsContentKeys（需要内容密钥）可选元项应当以假值存在。此元项不存在意味着在协议成功的情况下源必须重新加密信宿的内容密钥。

[1087] 在一个实施例中，当接收这种请求时，如下处理许可元项：

[1088] (1) 收集在 LicensePart 元项中所找到的所有 ContentId 属性。

[1089] (2) 处理在 LicensePart 元项中所找到的所有捆束属性。

[1090] (3) 打开上面所收集的该组内容 ID。

[1091] (4) 对相关对象验证适当签名。

[1092] (5) 对所处理的控制对象选择性地调用 Control.Actions.Transfer.Check 方法。

[1093] (6) 对所处理的控制对象调用 Control.Actions.Transfer.Perform。

[1094] 允许控制程序主动请求信宿的邻近度检查。为了允许控制程序这样作，可以定义

一对新的义务 / 回调。特别地是，控制可以把“ProximityCheckSink（邻近度检查信宿）”义务放入其扩展状态块中。这向应用表明必须检查与信宿的邻近度。当完成邻近度检查时，应用会使用“OnSinkProximityChecked”回调来回调所述控制。

[1095] 在一个实施例中，定义了只在许可转送的上下文情境内适用的ProximityCheck（邻近度检查）义务。在此实施例中，每个扩展状态块需要零或一个这种义务，并且如果存在的话，那么还需要存在 OnSinkProximityChecked 回调。

[1096]

<table>
<tr><th>名称</th><th>类型</th><th>描述</th></tr>
<tr><td>ProximityCheck</td><td>ValueList</td><td>主机应用需要与信宿设备执行邻近度检查协议。
<table>
<tr><th>类型</th><th>描述</th></tr>
<tr><td>字符串</td><td>必须进行邻近度检查的个性节点的 Id</td></tr>
</table>
</td></tr>
</table>

[1097] OnSinkProximityChecked 回调

[1098]

<table>
<tr><th>名称</th><th>类型</th><th>描述</th></tr>
<tr><td>OnProximityChecked</td><td>ValueList</td><td>当已经完成义务参数之一中的邻近度检查时主机应用需要回调。
<table>
<tr><th>类型</th><th>描述</th></tr>
<tr><td>回调</td><td>用于回调的例程以及相关联的 cookie（点心文件）。</td></tr>
</table>
</td></tr>
</table>

[1099] 在协议中允许多个往返行程。图 33 概述了会允许多个往返行程的协议的修改形式。在图 33 所示出的实施例中，设置消息 3302 例如可以与上面结合许可解析问题 / 解决方案描述的改进的许可转送请求消息相同。

[1100] 如图 33 所示，在设置 3302 之后，应用运行如上所述的控制并且获取扩展状态块 (ESB)。此 ESB 可以包含 RunAgentOnPeer 义务 /OnAgentCompletion 回调。在一个实施例中，RunAgentOnPeer 义务包含源 3312 应用需要用来构建 RunAgent 消息 3304 的所有参数。注意，在一个实施例中，如果应用在 OnAgentCompletion 回调的扩展状态块中遇到另一 RunAgentOnPeer/OnAgentCompletion 回调 / 义务对（在一个或多个 RunAgent/AgentResult 消息交换之后），那么还将发送 RunAgent 消息 3304。

[1101] 在一个实施例中，如果ESB没有包含RunAgentOnPeer义务/OnAgentCompletion回调，那么这意味着需要发送拆卸消息（参见下文）。注意，此 ESB 可以包含 ProximityCheck 义务 /OnSinkProximityChecked 回调，在这种情况下将执行邻近度检查协议并且在发送拆卸消息之前从 OnSinkProximityChecked 回调的 ESB 中读取结果。

[1102] 在一个实施例中，除并不携带ContentKeyList（内容密钥列表）之外，RunAgent消息3304的有效载荷与先前设计的响应消息完全相同。

[1103] 如图33所示，在信宿3310已经运行由源在RunAgent消息3304中发送的代理之后，所述信宿3310向源3312发送AgentResult（代理结果）消息3306。在一个实施例中，消息有效载荷与结合图32所描述的确认消息相同。

[1104] 如图33所示，当OnAgentCompletion的扩展状态块没有携带任何RunAgentOnPeer/OnAgentCompletion回调/义务对时（这意味着协议已经结束），由源应用3312发送拆卸消息3308。在一个实施例中，拆卸消息3308携带两个信息：(i)协议结果的描述，使得信宿3310知道所述协议已经成功还是没有成功，它为什么失败的指示（参见下文来获得更多细节），和(ii)在协议成功的情况下，所更新的内容密钥对象（先前消息中响应的ContentKeyList（内容对象列表）），设置消息的NeedsContentKey元项被设置为真还是不存在。

[1105] 在一个实施例中，协议结果的描述实际上是最后调用没有携带与代理相关的义务/回调对的控制的扩展状态块(ESB)。

[1106] 在失败的情况下，ESB的参数可以指向资源。在一个实施例中，这些资源位于在设置消息中所发送的控制的ResourceList（资源列表）扩展中。

[1107] 在成功的情况下，在一个实施例中，高速缓存持续时间表明在没有再次请求控制的情况下可以使用内容密钥多少时间。

[1108] 下面示出了这种ESB XML表示的例子，并且其可以被添加到虚拟机模式：

[1109]

```
<xs:element name="CacheDuration" type="CacheDurationType"/>
<!—高速缓存持续时间类型 -->
<xs:complexType name="CacheDurationType">
    <xs:attribute name="type" type="xs:int"/>
    <xs:attribute name="value" type="xs:int"/>
    </xs:complexType>

<xs:element name="ExtendedStatusBlock" type="ExtendedStatusBlockType"/>

<!—扩展状态块类型 -->
<xs:complexType name="ExtendedStatusBlockType">
 <xs:sequence>
      <xs:element ref="CacheDuration"/>
 <xs:element name="Parameters" type="ValueListBlockType" minOccurs="0"/>
 </xs:sequence>
 <xs:attribute name="globalFlags" type="xs:int" default="0" use="optional"/>
 <xs:attribute name="category" type="xs:int" use="required"/>
 <xs:attribute name="subcategory" type="xs:int" use="optional"/>
 <xs:attribute name="localFlags" type="xs:int" use="required"/>
</xs:complexType>
```

[1110] 下面是依照上述改进的许可转送机制的一个实施例的再现使用情况的例子。在此例子中,广播输入功能导入具有以下许可的内容 :

[1111] - 播放 :如果本地状态存在,那么 OK

[1112] - 转送 :

[1113] - 如果信宿处于域 X 中或者如果信宿在邻近,那么再现 OK。一次只可以再现一个并行流。

[1114] 假定核心 DRMClient1 请求再现内容流的权限。包含以下参数的设置请求被从信宿 (核心 DRMClient1) 发送到源 (BC 导入功能) :

[1115] - 许可 :与信宿想要再现的内容相关联的许可

[1116] - 操作 = urn :marlin :core :1-0 :service :license-transfer :render

[1117] - 捆束 = 信宿的个性节点

[1118] 当接收请求时,源应用填充相关的主机对象并且调用 Control.Actions.Transfer.Perform 方法。下面示出了用于管理支配再现转送的方法的说明性伪码 :

[1119]

```
/*管理支配方法的伪码
    再现转送*/
ESB* TransferRenderPerform(HostObjectTree* t) {
  // 检查锁定
  if (t->GetObject("SeaShell/.../lock") != NULL) {
    return new ESB(ACTION_DENIED);
  } else {
    // 时间限制锁定，在失败的情况下我们解锁
    t->SetObject("SeaShell/.../lock", 1);
    t->SetObject("SeaShell/.../lock@ExpirationTime,
                          Time.GetCurrent() + 180);
    // 返回包含 RunAgentOnPeer 的 ESB
    // 义务和 OnAgentCompleted 回调
    return new ESB(ACTION_GRANTED,
                   new Obligation(RUN_AGENT_ON_PEER,
                                  CheckDomainAgent),
                   new Callback(ON_AGENT_COMPLETED,
                                RenderAgentCompleted));
```

[1120]

```
                }
            }
```

[1121] 假定再现未被锁定，那么执行 RunAgentOnPeer 义务。利用包含 CheckDomainAgent 方法的控制来发送 RunAgent 消息。当接收此消息时，信宿将填充相关的主机对象并且调用 CheckDomainAgent 方法。下面示出了 CheckDomainAgent 的说明性伪码：

[1122]

```
/* CheckDomainAgent 的伪码*/
AgentResult* CheckDomainAgent(HostObjectTree* t) {
  // 检查域节点是否是可到达的
  if (IsNodeReachable("urn:marlin:...:domain2042x")) {
    return new AgentResult(SUCCESS);
  } else {
    return new AgentResult(FAILURE);
  }
```

[1123] 为了此说明目的，假定信宿的确在该域中。然后，信宿发送包含此代理结果的 AgentResult 消息。当接收 AgentResult 时，源将调用回调方法。下面示出了 RenderAgentCompleted 的说明性伪码：

[1124]

```
/ * RenderAgentCompleted 的伪码* /
ESB* RenderAgentCompleted(HostObjectTree* t,
                          AgentResult* ar)
{
  if (ar->IsSuccess()) {
    // 给出没有义务/回调的 ESB
    // 和高速缓存持续时间
    return new ESB(ACTION_GRANTED, new CacheDuration(0));
  } else {
    // 试图进行邻近度检查
    return new ESB(ACTION_GRANTED,
                   new Obligation(CHECK_PROXIMITY,
                     t->GetObject("…/Sink/Id"),
                   new Callback(ON_SINK_PROXIMITY_CHECKED,
                                ProximityCheckCompleted));
```

[1125]

```
                                }
                            }
```

[1126] 我们已经假定代理成功地检查了信宿上的域成员资格。发送拆卸消息，所述拆卸消息具有 (i)（使用在设置请求中向信宿节点提供的密钥）为信宿重新加密的内容密钥，和 (ii) 携带上面所指定的高速缓存持续时间的 ESB（在这种情况下为 0，意味着所述信宿下次想要访问内容时必须重新请求）。当信宿接收此消息时，它知道被允许再现内容并且具有所需要的内容密钥。

[1127] 现在假定用户想要在他的其它设备 DRMClient2 上再现内容。问题在于所述内容在该源上被锁定了 180 分钟。幸运地是，当用户在 DRMClient1 上按下 STOP 时，DRMClient1 利用操作 - 释放 - 来发起新的许可转送协议。当接收请求时，源应用将填充相关的主机对象并且调用 Control.Actions.Transfer.Perform 方法。下面示出了用于管理支配转送释放的方法的说明性伪码：

[1128]

```
/*管理支配方法的伪码
    转送释放*/
ESB* TransferReleasePerform(HostObjectTree* t) {
  // 检查锁定
  if (t->GetObject("SeaShell/.../lock") != NULL) {
    t->SetObject("SeaShell/.../lock, NULL);    //删除
    return new ESB(ACTION_GRANTED);
  } else {
    return new ESB(ACTION_DENIED);
  }
}
```

[1129] 由于在ESB中没有找到义务/回调，所以这意味着利用此ESB向回发送拆卸消息。

[1130] 从而此再现使用情况举例说明了在某些实施例中，再现操作的请求DRMClient不需要本地重新评估控制，状态不必从源转送到信宿，控制可以主动请求邻近度检查，并且当再现器利用内容时可以释放所述内容。

[1131] 11. 证书

[1132] 在一个实施例中，证书用来在根据利用那些密钥创建的数字签名进行判定之前检查与密码密钥相关联的凭证。

[1133] 在一些实施例中，DRM引擎被设计成用于与标准的证书技术兼容，并且可以平衡在这种证书的元项中所找到的信息，诸如有效时段、名称等。除那些基本约束之外，在一些实施例中，可以定义关于所证实的密钥能够用于和不能用于作什么的附加约束。这可以通过例如使用可用为证书的标准编码一部分的密钥使用扩展来实现。在这种扩展中所编码的信息使DRM引擎能够检查已经签名特定对象的密钥是否被授权以用于该目的。例如，某个密钥可能具有这样的证书，只有链路是从特定属性的节点到具有另一特定属性的节点而不是其它链路，所述证书才允许所述密钥对链路对象进行签名。由于用于表达证书的通用技术的语义通常不能够表达这种约束，这是因为没办法表达与诸如链路和节点之类DRM引擎专用的元项相关的条件，所以在一个实施例中这种DRM引擎专用的约束被作为基本证书的密钥使用扩展来传达，其由已经被配置为使用所述DRM引擎的应用来处理。

[1134] 在一个实施例中，密钥使用扩展中的约束由使用类别和VM约束程序来表达。使用类别指定了向密钥授权签名什么类型的对象。约束程序可以根据上下文情境来表达动态条件。在一个实施例中，被请求验证这种证书有效性的任何验证器被要求要理解DRM引擎语义，并且把对密钥使用扩展表示的评估委托给DRM引擎，所述DRM引擎使用虚拟机的实例来执行该程序。如果该程序执行的结果成功，那么该证书被认为是有效的。

[1135] 在一个实施例中，约束程序的任务是返回布尔值。“真”意味着满足约束条件，并且“假”意味着不满足约束条件。在一个实施例中，控制程序有权访问一些上下文信息，所述上下文信息可以用来作出决定，诸如经由虚拟机的主机对象接口可用于程序的信息。可被用为上下文的信息取决于当DRM引擎请求验证证书时试图做出什么类型的决定。例如，在使

用链路对象中的信息之前，在一个实施例中 DRM 引擎需要验证用于签名所述对象的密钥的证书允许该密钥用于该目的。当执行约束程序时，利用关于链路属性以及由所述链路所引用的节点属性的信息来填充虚拟机的环境。

[1136] 在一个实施例中，被嵌入到密钥使用扩展中的约束程序被编码为虚拟机代码模块，虚拟机代码模块用于输出名称为“Octopus.Certificate.<Category>.Check”的至少一个入口点，其中“类别”名称用于表明需要检查哪种类别的证书。在调用入口点之前，用于验证程序的参数被压入堆栈。传递到堆栈上参数的数目和类型通常取决于所评估的证书扩展的类别。

[1137] 12. 数字签名

[1138] 在优选实施例中，对由 DRM 引擎所使用的一些或全部对象进行签名。下面描述了在一个实施例中怎样使用 XML 数字签名规范 (http://www.w3.org/TR/xmldsig-core)（“XMLDSig”）来对对象进行数字签名。另外，还描述了与 XML 专用规范 (http://www.w3.org/TR/xml-exc-c14n/)（“c14n-ex”）兼容的 XML 的规范方法，其输出可以由非 XML 名称空间察觉 (non-XML-namespace-aware) 的解析器来处理。附录 D 提供了关于示例性对象串行化的更多信息，包括用于依照与编码无关的方式来为对象计算规范字节序列的说明性方式。

[1139] 如图 28、34 和 35 所示，在优选实施例中对 DRM 许可中的某些元项进行签名。诸如在图 28、34 和 35 中所示出的那些技术在防止或阻止篡改或替换许可组件上是很有用的。如图 34 所示，在优选实施例中，控制器对象 3402 分别包括内容密钥对象 3404 和控制对象 3406 的密码摘要或散列（或其它适当的绑定）3405、3407。控制器 3402 本身被利用 MAC（或优选为利用内容密钥的 HMAC) 和公钥签名（一般为内容或许可提供者的公钥签名）3412 而签名。在优选实施例中，控制器的公钥签名 3412 本身是使用内容密钥来利用 HMAC 3410 而被签名的。应当理解，在其它实施例中，取决于所想要的安全等级和 / 或其它系统需求，可以使用其它签名方案。例如，不同的签名方案可以用于控制器和 / 或控制的签名，诸如 PKI、标准 MAC 等。作为另一例子，可以为控制和控制器这二者计算独立的 MAC 签名，而不是在控制器中包括所述控制的摘要并计算所述控制器的单个 MAC 签名。在又一例子中，可以利用 MAC 和公钥签名来签名控制器。作为选择或另外，可以使用与上述那些不同的密钥来产生各个签名。从而虽然图 28、34 和 35 依照一些实施例图示了几种有益的签名技术，然而应当理解，这些技术是说明性并且不是限制性的。图 35 图示了其中控制器引用多个内容密钥的实施例。如图 35 所示，在一个实施例中，每个内容密钥用来产生控制器的 HMAC 和 PKI 签名。

[1140] 在一个实施例中，除移除名称空间前缀（使用默认的名称空间机制来表明名称空间）以及不支持外部实体只支持字符实体之外，用于 XML 规范的数据模式、处理、输入参数和输出数据与专用规范 XML(c14n-ex) 相同。第一限制意味着属性及其元项需要处于相同的名称空间中。

[1141] 图 42 示出了在一个实施例中在 c14n-ex 和说明性 XML 规范化之间的关系，其中 <xml> 是任何有效的 XML，并且其中只有在 <xml> 没有外部实体并且没有名称空间前缀时 <xml>’ = <xml>”。

[1142] 下面提供了简化签名方案的简单例子 :然而，在优选实施例中，使用标准的 XML 规范。

[1143]

原始的	<n1:elem2 id="foo" xmlns:n0="foo:bar" xmlns:n1="http://example.net" xmlns:n3="ftp://example.org"> <n3:stuff/> </n1:elem2>
处理的	<elem2 xmlns="http://example.net" id="foo"> <stuff xmlns="ftp://example.org"/> </elem2>

[1144] 在此节中所论述的签名元项属于XMLDSig名称空间(xmlns = http://www.w3.org/2000/09/xmldsig#)并且在XMLDSig规范中所定义的XML模式中定义。在一个实施例中,DRM对象的XML表示的容器元项是<Bundle(捆束)>元项。

[1145] 在一个实施例中,需要签名以下对象:

[1146] ■节点

[1147] ■链路

[1148] ■控制器

[1149] ■控制(可选)

[1150] ■扩展(取决于它们所携带的数据)

[1151] 在一个实施例中,签名需要被分离并且<Signature>元项需要存在于<Bundle>对象中,所述<Bundle>对象包含需要被签名的对象的XML表示。

[1152] 在一个实施例中,<Signature(签名)>块包含:

[1153] ■<SignedInfo>元项

[1154] ■<SignatureValue>元项

[1155] ■<KeyInfo>元项

[1156] 在一个实施例中,<SignedInfo>嵌入以下元项:

[1157] <CanonicalizationMethod>-在一个实施例中,<CanonicalizationMethod>元项是空的并且其算法属性具有以下值:http://www.w3.org/2001/10/xml-exc-c14n#

[1158] <SignatureMethod>-在一个实施例中,<SignatureMethod>元项是空的并且其算法属性可以具有以下值:

[1159] ■http://www.w3.org/2000/09/xmldsig#hmac-sha1(HMAC签名)

[1160] ■http://www.w3.org/2000/09/xmldsig#rsa-sha1(公钥签名)

[1161] <Reference>-在一个实施例中,如果一个以上对象需要由相同的密钥来签名(例如,对于控制和控制器对象,这可能就是这种情况),那么在<SignedInfo>块内就可以存在一个或多个<Reference(引用)>元项。

[1162] 在一个实施例中,当对对象签名时,<Reference>元项的'URI'属性值为引用对象

的 ID。当对本地 XML 元项签名时（例如在用于控制器对象的公共签名方法的多个签名情况中），URI 值为所引用的元项的‘Id’属性的值。

[1163] 在一个实施例中，当引用指向对象时，在所述引用中所作的摘要不是对象的 XML 表示而是其规范的字节序列。借助于 <Tranforms（变换）> 块在 XMLDSig 中表明此对象变换。因此在一个实施例中，<Reference> 元项嵌入此块：

[1164]

```
<Tranforms>
    <Transform Algorithm="http://www.intertrust.com/octopus/cbs-1_0"/>
</Tranforms>
```

[1165] 附录 D 提供了附加信息。在一个实施例中，对于对象引用来说不允许其它 <Tranform>。

[1166] 在一个实施例中，<DigestMethod> 元项是空的并且其算法属性具有以下值：http://www.w3.org/2000/09/xmldsig#sha1

[1167] <DigestValue（摘要值）> 元项包含摘要的 base64 编码值。

[1168] <SignatureValue（签名值）>——在一个实施例中，签名值是利用在 <KeyInfo> 元项中所描述的密钥来 base64 编码的规范化 (ex-c14n)<SignedInfo> 元项的签名值。

[1169] <KeyInfo>

[1170] ■用于控制器对象签名的 HMAC-SHA1 情况

[1171] 在一个实施例中，在这种情况下 <KeyInfo> 只有一个孩子：<KeyName>，用于表明已经为 HMAC 签名使用的密钥 ID。

[1172] 例子：

[1173]

```
<KeyInfo>
    <KeyName>urn:x-octopus:secret-key:1001</KeyName>
</KeyInfo>
```

[1174] ■ RSA-SHA1 情况

[1175] 在一个实施例中，在这种情况下，在 X.509v3 证书中携带用于验证签名的公钥并且所述公钥可以由其它证书附带，所述其它证书对完成通向 CA 根的证书路径来说是可能是必要的。

[1176] 这些证书被依照 base64 编码并携带在 <X509Certificate> 元项中。这些 <X509Certificate> 元项被嵌入到 <KeyInfo> 元项的 <X509Data> 元项孩子中，并且依照顺序次序出现，从签名密钥的证书开始。通常省略根的证书。

[1177] 例子（为了简洁起见，尚未再生整个示例性证书的值；已经被删除的材料由省略符号来表明）：

[1178]

```
<KeyInfo>
[1179]   <X509Data>
           <!—签名的公钥的证书 -->
           <X509Certificate>MIICh...</X509Certificate>
           <!—到信任根的中间证书 -->
           <X509Certificate>MIICo...</X509Certificate>
         </X509Data>
         </KeyInfo>
```

[1180] 在一个实施例中，控制器对象对于在它们的受控目标列表中所引用的每个内容密钥来说需要具有至少一个 HMAC 签名。用于那些签名中每个的密钥是在所引用的内容密钥对象中所包含的内容密钥值。

[1181] 控制器还可以具有 RSA 签名。在一个实施例中，如果这种签名存在，那么此签名还作为 <Reference> 在用于所述对象的每个 HMAC 签名中出现。为了实现这点，在一个实施例中，用于 RSA 签名的 <Signature> 元项必须在封入的 XML 文档内具有唯一的 'Id' 属性，其在每个 HMAC 签名的 <Reference> 元项之一中被用为 'URI' 属性。在一个实施例中，验证器必须拒绝未被 HMAC 签名确证的 RSA 签名。

[1182] 例子：

[1183]

```
<Signature Id="Signature.0" xmlns="http://www.w3.org/2000/09/xmldsig#">
<SignedInfo>
 <CanonicalizationMethodAlgorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
 <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
 <Reference
URI="urn:x-octopus.intertrust.com:controller:37A50262EE3389A14ABC0BC7BE5D
43E5">
    <Transforms>
    <Transform Algorithm="http://www.intertrust.com/Octopus/xmldsig#cbs-1_0"/>
    </Transforms>
    <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
    <DigestValue>G1zXF9Sz/zCwH6MaFm0ObOQcxuk=</DigestValue>
    </Reference>
 </SignedInfo>

<SignatureValue>mjoyW+w2S9iZDG/ha4eWYD1RmhQuqRuuSN977NODpzwUD02
```

FdsAICVjAcw7f4nFWuvtawW/clFzYP/pjFebESCvurHUsEaR1/LYLDkpWWxh/LlEp4
r3yR9kUs0AU5a4BDxDxQE7nUdqU9YMpnjAZEGpuxdPeZJM1vyKqNDpTk94=</Si
gnatureValue>
<KeyInfo>
<X509Data><X509Certificate>MIICh...</X509Certificate></X509Data>
</KeyInfo>
</Signature>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
<SignedInfo>
<CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
<SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
<Reference URI="#Signature.0">
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<DigestValue>AqPV0nvNj/vc51IcMyKJngGNKtM=</DigestValue>
</Reference>
<Reference URI="urn:x-octopus.intertrust.com:controller:1357">
<Transforms>
<Transform
Algorithm="http://www.intertrust.com/Octopus/xmldsig#cbs-1_0"/>
</Transforms>
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<DigestValue>G1zXF9Sz/zCwH6MaFm0ObOQcxuk=</DigestValue>
</Reference>
</SignedInfo>
<SignatureValue>TcKBsZZy+Yp3doOkZ62LTfY+ntQ=</SignatureValue>
<KeyInfo>
<KeyName>urn:x-octopus.intertrust.com:secret-key:2001</KeyName>
</KeyInfo>
</Signature>

[1184] 13. 邻近度检查协议

[1185] 在一些实施例中，可能希望根据请求实体的物理邻近度来限制对内容、服务和 /

或其它系统资源的访问（例如，帮助强制实施用于表明受保护的内容无法被拷贝到用户家庭网络、办公室建筑物等之外的规则）。下面将描述邻近度检查协议的实施例，所述实施例在不过于妨碍执行邻近度检查本身的情况下提供安全性。邻近度检查协议有助于在各式各样的宽的上下文情境中应用，如上所指出其中之一是在数字权利管理控制的上下文情境中；然而应当理解，下述邻近度检查系统和方法并不限制应用于数字权利管理上下文情境。例如而并非限制，这里所给出的邻近度检查技术还可以在网络服务组织系统上下文情境中使用，诸如在‘551 申请描述的上下文情境中和 / 或任何其它适当的上下文情境中。

[1186] 在一个实施例中，通过测量第一计算节点从第二计算节点接收对所述第一计算节点请求的响应所花费的时间量来执行邻近度检查。如果所述时间量小于预定义阈值（通常表明第二计算节点在第一计算节点的某一物理距离内），那么邻近度检查被认为是成功的。

[1187] 应当理解，由于存在各种可以用来发送请求和 / 或响应的不同网络连接的缘故，所以给定时间量可以对应于可能的距离范围。在一些实施例中，此变化被简单地忽略，并且如果请求 / 响应交换的往返时间小于预定义的阈值（例如，8 毫秒或任何其它适当的时间量），而不管例如是否正在使用快速网络连接，这可能意味着请求和响应节点实际上彼此相对较远，那么邻近度检查被认为是成功的。在其它实施例中，可以确定所使用的网络连接类型，并且不同的往返时间要求可以被应用于每个不同的网络连接。

[1188] 在优选实施例中，邻近度检查允许锚（例如，客户端）检查目标（例如，服务）的邻近度。在一个实施例中，协议是不对称的，这是由于锚产生了秘密种子，所述种子被使用并且是利用安全计时器的唯一一个。此外，目标不必信任所述锚。邻近度检查的优选实施例还是密码高效的：在一个实施例中只利用两个公钥操作。

[1189] 从种子 S 产生对 Q 的组 R

[1190] 在一个实施例中，依照以下公式从种子 S 获得组 R：$R_i = H^{2Q-i}(S)$。其中 H(M) 是对消息 M 的散列函数 H 的摘要值，并且对于 n ＞＝ 1 并且 $H^0(M) = M$ 来说，$H^n(M) = H(H^{n-1}(M))$。应当理解，这只是用于产生共享秘密的一个说明性技术，并且在不脱离其原理的情况下在其它实施例中可以使用其它技术。

[1191] 在一个实施例中，用于散列函数 H 的算法是 SHA1（例如参见 FIPSPUB 180-1。安全散列标准。美国标准和技术的商业 / 国家协会部 (U.S.Department of Commerce/National Institute of Standards andTechnology)），不过应当理解，在其它实施例中，可以使用其它散列、消息摘要或函数。

[1192] 在一个实施例中，按如下执行邻近度检查，其中“A”是锚（例如，客户端）并且“B”是目标（例如，服务）：

[1193] (a) 如上所示，A 产生 Q 对随机数的组 R：$\{R_0, R_1\}$，$\{R_2, R_3\}$...$\{R_{2Q-2}, R_{2Q-1}\}$。

[1194] (b)A 向 B 发送 E(PubB，{Q，S})，其中 E(Y，X) 标示了利用密钥 Y 加密 X，并且 PubB 标示了 B 在公钥 / 私钥对中的公钥。

[1195] (c) 如上所示，B 解密 {Q，S} 并且预先计算 R。

[1196] (d)B 向 A 发送确认以便表明它准备继续。

[1197] (e)A 把循环计数器 k 设置为零。

[1198] (f)A 测量 T_0 ＝当前时间。

[1199] (g)A 向 B 发送：$\{k, R_{2*k}\}$。

[1200] (h) 如果 R_{2*k} 值是正确的，那么 B 用 R_{2*k+1} 作出响应。

[1201] (i)A 测量 D ＝新的当前时间 -T0。

[1202] (j) 对于 R_{2*k+1} 来说，如果 B 利用正确值向 A 作出响应，并且 D 小于预定义的阈值，那么邻近度检查被认为是成功的。

[1203] 如果 k+1 ＜ Q，那么 A 可以通过增加 k 并且转到步骤 (f) 来重新尝试新的测量。如果需要执行 Q 次以上的测量，那么 A 可以以新的组 R 从步骤 (a) 开始。例如在一些实施例中，可以重复地（或预定义次数地）执行邻近度检查，直到在预定义的阈值内接收正确的响应（或者如果在大于质询 / 响应序列的预定义百分比的预定义阈值内接收了正确的响应），这是因为即便两个计算节点彼此在所要求的邻近度内，异常缓慢的网络连接、大通信量、噪声等也可能导致 B 的响应延迟。

[1204] 图 36 图示了上述协议的实施例，其中锚 (A) 确定目标 (B) 是否在可接受的锚 (A) 的邻近度内。例如如图 36 所示，A 可以包括计算节点 3602，所述计算节点 3602 包含受保护的内容（例如，音乐、视频、文本、软件等）和 / 或远程计算节点 (B)3606 访问受保护的内容所需要的内容访问材料（例如，链路、密钥等），所述受保护的内容存储在计算节点 B 3606 或可由其访问。与内容或内容访问材料相关联的控制可以表明它只可以与节点 A 3602 某一邻近度内的设备共享（例如，把内容分发近似地限制在家庭网络）。作为选择或另外，可以在计算节点 A3602 的系统级强制实施这种策略（所述计算节点 A 3602 例如可以包括家庭或企业网的域管理器）。即，邻近度检查不必是由虚拟机所执行的控制程序中的条件；作为替代它可以只是计算节点 A 3602 在向计算节点 B 3606 发送内容或内容访问材料之前作为操作策略所要求的一些东西。为了强制实施这种控制和 / 或策略，每当请求向计算节点 B 3606 分发受保护的内容或内容访问材料时，在计算节点 A 3602 上运行的软件和 / 或硬件可以执行上述邻近度检查协议。作为选择或另外，可以以预定义间隔执行邻近度检查（例如，一天一次）以便确定节点 B 是否在所要求的邻近度内，以及所述邻近度检查是否成功，对于预定义周期来说节点 B 3606 可以被视为在所要求的邻近度内（例如，直到执行下一次检查，直到预定义的时间量过去等）。

[1205] 如图 36 所示，一旦 A 和 B 完成任何初始设置步骤（例如，上面的步骤 (a) 到 (e))3604、3608，A 和 B 就从事安全、计时的质询 - 响应交换（例如，上面的步骤 (f) 到 (i))3610，所述交换使 A 能够确定 B 是否在可接受的邻近度内。

[1206] 如图 36 所示，在一个实施例中 A 3602 向 B 3606 发送设置请求 3604，所述设置请求 3604 包括 E(PubB，{Q，S})——即，对 Q 的数目，以及利用 B 的公开加密密钥（例如，由 B 在服务组织上下文情境内使用的密钥）加密的秘密对种子 S。在一个实施例中，{Q，S} 是依照网络字节次序的 Q(1 字节）和 S(16 字节）的字节流拼接。在一个实施例中，使用 RSA 公钥加密（例如在 B.Kaliski、J.Staddon，PKCS#1 的 RSA Cryptography Specifications Version 2.0 中所描述，IETFRFC2437，1998 年 10 月）来执行加密。在优选实施例中，PubB 将事先已经由 A 通过审查访问了，并且其证书将已经被验证。尽管在图 36 中示出了从 B 3606 到 A 3602 的设置响应 3608，不过在其它实施例中，不使用设置响应 3608。如先前所表明，在接收设置请求 3604 之后，B3606 优选预先计算组 R，以便对来自 A 3602 的随后质询进行快速响应。

[1207] 如图 36 所示，A 36-2 向 B 发送质询请求 3612，所述质询请求 3612 包括 [k，

R_{2*k}]——即，索引 k 和根据种子所计算的相应秘密。在一个实施例中，[k，R_{2*k}] 是用 base64 编码的、依照网络字节次序的 k(1 字节）和 R_{2*k}(20 字节）的字节流拼接以供传输。如图 36 所示，在一个实施例中，B 3606 可操作来向 A 3602 发送质询响应 3614，所述质询响应 3614 由 R_{2*k+1} 组成——即，来自质询请求 3612 的相应秘密。在一个实施例中，R_{2*k+1} 是用 base64 编码的、依照网络字节次序的 R_{2*k+1}(20 字节）的字节流以供传输。

[1208] 图 37 示出了可以怎样使用上述邻近度检查协议的实施例来控制对受保护内容访问的例子。参照图 37，假定电缆或卫星内容供应者具有用于允许用户个人录像机(PVR)3702 的预定义邻近度 3708 内的所有设备经由所述 PVR 访问内容的策略。从而例如在 PVR 3702 上运行的域管理器软件可以对用于请求经由 PVR 3702 对内容进行访问的设备 3704 和 3706 执行邻近度检查。在图 37 中所示出的例子中，设备 3706 不在由服务供应者策略所定义的邻近度 3708 内，并且会被 PVR 3702 拒绝访问。相比之下，设备 3704 在所述邻近度内，并且会被提供有访问（例如，通过连同从设备 3704 到 PVR 3702 的期满链路一起接收内容）。作为选择或另外，所述链路可以包含控制程序，所述控制程序本身可操作来利用 PVR 3702 发起邻近度检查，并且如果设备 3704 移出 PVR3702 的预定义邻近度 3708 之外，那么拒绝设备 3704 进一步对内容进行访问。

[1209] 安全性考虑

[1210] 在优选实施例中，应当注意坚持以下一些或全部：

[1211] ●对于任何组 R 来说，并不利用相同的值 k 来重复包括步骤 (f) 到 (i) 的循环。

[1212] ●如果任何一方接收了意料之外的消息，那么中止协议，所述意料之外包括：

[1213] ○如果在步骤 (g) 中对于 R_{2*k} 来说 B 接收了不正确的值

[1214] ○如果在步骤 (a) 中 Q 不在所指定的距离内

[1215] ○如果在循环中重复 k

[1216] ○如果 k 超过 Q

[1217] 作为选择或另外如果在步骤 (h)A 接收了不正确的值 R_{2*k+1}，那么中止协议。在其它实施例中，可以容许某些来自 B 的不正确响应。

[1218] 应当理解，Q 的最优值和预定义的时间阈值一般取决于手边的应用的唯一环境（例如，网络速度、确保相对紧密邻近度的重要性等）。因此，实现方式优选应当在配置这些值中提供灵活性。在一个实施例中，假定实现方式对于 Q 来说支持最小值 64 并且对于阈值来说支持值 8ms（其中以今天的某个网络速度，8ms 可对应于几英里的邻近度）。

[1219] 协议安全策略

[1220] 在优选实施例中，对于交换请求和响应来说不需要附加的安全性。由于所交换消息的大小（例如，20 字节）以及它们有效随机性的原因（通过使用 SHA1 散列算法或其它方法），即便攻击者设法拦截所述请求，那么所述攻击者要确定正确的响应在密码上也是不可行的。

[1221] 应当理解，上述实施例是说明性的，并且在不脱离这里所给出发明原理的情况下可以进行许多修改。例如，虽然上面描述了递归散列的秘密种子，不过任何适当的共享秘密可以用于质询 / 响应。在一个实施例中，共享秘密可以只是包括从 A 到 B 发送的加密的数目 / 消息，并且质询 / 响应可以只是包括数目 / 消息的 A 和 B 交换部分（例如，A 向 B 发送消息的第一字符，并且 B 向 A 发送所述消息的第二字符等）。尽管这种技术可能缺乏结合图

36 所描述的实施例的安全性（由于消息中的字符会比 20 字节散列更易于猜测），不过在一些实施例中，这种安全等级可能是足够的（特别是例如在网络延迟可变性使邻近度检查机制相当粗略地控制实际邻近度的情况下），并且在其它实施例中，可以通过多次执行邻近度检查来增强安全性，其中尽管任何特定的数字或比特可能相对易于猜测，不过攻击者能够正确猜测给定数字或比特序列的可能性随着序列长度增加而迅速减小。在这种实施例中，只有当 B 能够提供预定义数目以上个连续的正确响应（或正确响应的预定义百分比以上）时，邻近度检查才可以被认为是成功的。

[1222] 为了说明和解释，下面提供了邻近度检查协议的补充说明性例子。在此例子中，第一设备 SRC 经由通信信道（例如计算机网络）与第二设备 SNK 通信。我们想要能够安全地确定 SRC 和 SNK 是否彼此在邻近度（按照 SNK 对来自 SRC 的通信请求作出响应所花费的时间来测量）以内。从 SRC 向 SNK 发送质询或探测消息，并且 SNK 利用响应消息来应答。在发出质询和接收响应之间的时段被称作往返时间或 RTT。为了避免在 SNK 计算并送回对质询的响应所花费的时间中引入不必要的开销，通常希望实际上尽可能少量地进行质询 / 响应通信。特别地是，一般希望避免要求 SRC 或 SNK 在发出质询和接收响应之间进行密码运算。

[1223] 为了确保只有 SNK 能够对来自 SRC 的质询生成有效响应（例如，为了避免中间人的攻击，其中第三方可以拦截来自 SRC 的质询并且向回发送响应，就好像 SNK 已经响应一样），所述协议还可以按如下继续：

[1224] (1)SRC 创建秘密。此秘密由一对或多对随机或伪随机数组成。

[1225] (2)SRC 向 SNK 发送秘密。协议的此部分不是对时间敏感的。所述秘密由 SRC 和 SNK 来保持机密。还依照确保只有 SNK 知道所述秘密的方式来发送所述秘密。这一般涉及经由在 SRC 和 SNK 之间的安全认证信道来发送秘密（例如，SRC 可以利用它知道只有 SNK 具有相应私钥的公钥来加密秘密数据）。秘密数据不必是上述的随机或伪随机数对。即便在使用这种对的实施例中，在此步骤中所发送的秘密数据只需要是使 SNK 足以计算或推算数目对值的信息。例如，秘密数据可以是随机种子数目，据此可以使用种子伪随机数发生器来产生一对或多对伪随机数。

[1226] (3) 一旦 SRC 知道 SNK 准备接收质询（例如，SNK 可以在接收和处理秘密数据之后发送 READY 消息），SRC 就创建质询消息。为了创建质询消息。例如在优选实施例中，SRC 选择随机数对之一。如果使用一对以上，那么质询消息数据包含用于表明选择哪对以及在该对中两个数目之一的信息。

[1227] (4)SRC 测量当前时间值 T0。在此之后，SRC 立即向 SNK 发送质询消息（不需要加密或数字签名）并且等待响应。作为选择，SRC 可以在发送质询消息之前（不过优选在已经执行任何伴随的密码运算（例如，加密、签名等）之后）立即测量当前时间 T0。

[1228] (5)SNK 接收质询，据此它可以标识它先前已经接收的对之一。SNK 检查质询中的随机数是所述对的一部分，并且构造响应消息，所述响应消息包含了该对的另一随机数的值。

[1229] (6)SNK 向 SRC 发送响应消息（不需要加密或数字签名）。

[1230] (7)SRC 接收响应消息，并且测量当前时间 T1 的值。往返时间 RTT 等于 T1-T0。

[1231] (8)SRC 验证在响应中所接收的数目等于为质询所选择的对中的另一值。如果所述

数目匹配，那么质询响应成功，并且 SRC 可以确信 SNK 在由往返时间所表明的邻近度内。如果所述数目不匹配，那么 SRC 可以中止协议，或者如果共享一个以上的对，并且存在尚未使用的至少一对，那么返回到步骤 (3) 并且使用不同的对。

[1232] 应当理解，在不脱离其原理的情况下可以对上述说明性邻近度检查协议进行许多改变。例如而并非限制，可以使用不同的密码算法，可以使用不同的共享秘密等。

[1233] 14. 安全性

[1234] 在实际应用这里所描述的系统和方法中，可以以各种不同的级别并且使用各种不同的技术来提供安全性。这里的论述主要集中于 DRM 引擎和相关主机应用的设计和操作以用于有效地调整可能复杂的业务关系。当 DRM 引擎和主机应用按照预计的那样操作时，通过强制实施与内容相关联的许可条款来保护所述内容免受未经授权的访问或其它使用。

[1235] 保护 DRM 引擎和 / 或所述 DRM 引擎运行的环境（例如，应用和其与之交互的硬件）免受恶意篡改或修改可以使用任何适当的安全性技术组合来进行。例如，可以使用诸如加密、数字签名、数字证书、消息认证代码等密码机制，例如在此的其它地方所描述，以保护 DRM 引擎、主机应用和 / 或其它系统软件或硬件免受篡改和 / 或其它攻击，也可以是结构和 / 或策略上的安全措施，诸如软件模糊化、自检、定制、水印、反调试和 / 或其它机制。例如可以在美国专利号 6,668,325B 1 的 Obfuscation Techniques for Enhancing Software Security 中以及在共同受让的美国专利申请号 11/102,306，作为 US-2005-0183072-A1 公开；美国专利申请号 09/629,807；美国专利申请号 10/172,682，作为 US-2003-0023856-A1 公开；美国专利申请号 11/338,187，作为 US-2006-0123249-A1 公开；以及美国专利号 7,124,170B 1 的 SecureProcessing Unit Systems and Methods 中找到这种技术的代表性例子，这里在此通过全部引用加以结合以供参考。作为选择或另外，可以使用物理安全技术（例如，使用相对难于访问的存储器、安全处理器、安全存储器管理单元、硬件保护的操作系统模式等）来进一步增强安全性。这种安全性技术为本领域普通技术人员之一所知，并且应当理解，根据所想要的保护级别和 / 或手头特定应用的细节可以使用一些或所有这些技术的任何适当的组合，或者不使用这些技术。从而应当理解，虽然这里结合某些实施例描述了某些安全机制（例如，密钥推导技术、数字签名技术、加密技术等），但是在所有实施例中并不要求使用这些技术。

[1236] 可以借助系统的规定设计和操作以及其中参与者的法律和社会规章来提供又一形式的安全性。例如，为了获得个性节点、密钥材料、受保护内容等，可以要求设备或实体在契约上同意遵循系统规范和要求，可以需要服从证明 (certification) 过程等，其中在所述证明过程期间可以验证实体与系统要求的顺应性。例如，可以要求设备或应用依照与环境中的其它实现方式兼容的方式来实现 DRM 引擎，和 / 或要求所述设备或应用提供某种类型或级别的抗窜改或其它安全性。可以发出用于证明设备或其它实体与这种要求顺应性的数字证书，并且可以在允许所述设备或实体参与系统之前或作为允许继续访问的条件来验证这些证书。

[1237] 下面提供了关于安全性技术的另外的非限制性信息，其可以结合所发明的工作主体使用。

[1238] 系统安全性

[1239] 在一些实施例中，系统设计员可以选择使用更新、拒绝和 / 或补救技术的组合来

管理风险并缓和威胁，所述威胁可能起因于对设备、应用和服务的攻击和危害。下面给出了可以用来缓和威胁的各种技术机制的例子。

[1240] 可以使用更新机制来服务至少两个不同目的。首先，它们可以用来向信任的系统实体传达最新信息，所述系统实体允许它们拒绝访问或服务于不信任的系统实体。第二，更新机制使不信任的实体能够通过更新任何有危害的组件来返回到信任状态。拒绝对策可以进一步被表征为表现为以下一个或多个行为：

[1241] ●撤销或废除凭证（一般通过把一些实体放入黑名单）

[1242] ●通过应用密码或策略强制机制来排除或拒绝访问

[1243] ●根据被绑定到凭证的身份或其它属性来回避或拒绝访问或服务

[1244] ●根据时间事件来截止或废除凭证或特权。

[1245] 例如，可以使用拒绝机制来对诸如设备克隆、假冒攻击、协议失败、策略实施失败、应用安全性失败和失效或可疑信息之类的威胁进行计数。

[1246] 下表提供了潜在威胁、它们造成的一些风险和用于补救所述威胁并且更新系统安全的机制的例子。

[1247]

威胁	风险	补救机制	更新机制
克隆设备	自由访问设备。	广播加密	BKB 更新。
有危害的证实的密钥	未经授权的许可、链路、设备状态、身份、服务访问。	证书撤销	CRL 分发。 密钥更新。
实现失败	设备被黑的诀窍。	规范版本声明	软件升级
协议失败	有危害的密钥。 对许可内容未经管理的访问。	安全元数据声明	软件升级
失效安全元数据	伪造的服务交互。 时钟回退，依赖有危害的信息。	安全元数据声明	安全元数据更新服务。 软件升级。

[1248] 撤销

[1249] 撤销可以被视为依赖把实体列入黑名单的补救机制。典型情况下，所撤消的是诸如公钥证书之类的凭证。当撤消凭证时，需要更新黑名单并且使用更新机制来传达所述更新使得依赖方可以从中获益。

[1250] 从而在向设备、用户和 / 或其它实体给予用于消费内容或服务所必须的信息之前，例如可以要求它们给出身份证书、其它凭证和各种安全数据。类似地，为了使客户端信任服务，所述服务可能需要向所述客户端提供其凭证。

[1251] 实体可以有效地使为访问服务所必须的信息无效的方式的例子包括 :

[1252] ●证书撤销列表 (CRL)

[1253] ●凭证和数据有效性服务，诸如在线证书状态协议 (OnlineCertificate Status Protocol OCSP) 应答者

[1254] ●用于自毁凭证和数据的命令

[1255] 证书撤销列表 (CRL)

[1256] 撤销列表可以由不同的实体用来撤消身份证书、许可、链路及其它安全声明。此机制对于补救源于服务受到危害的情况来说是最有效的。可以使用多种技术来分发 CRL。例如，一些系统可以使用间接 CRL，使得存在用于管理整个生态系统 (escosystem) 的单个 CRL。另外，实体可以通告 (或公开) 它们拥有的 CRL 和 / 或预订更新服务。可以依照病毒引起的方式来对等地分发 CRL，和 / 或便携式设备当被束缚时可以接收公开的 CRL。在 '551 申请中所描述的服务组织技术也可以用于此目的。

[1257] 有效性服务

[1258] 可以使用有效性服务来提供关于凭证及其它安全相关数据状态的最新信息。有效性服务可以执行代表依赖方来执行活动的证实操作，或者它们可以用来代表依赖方管理安全信息。活动有效性服务的例子是可以检查凭证或属性有效性的服务。用于管理安全信息的有效性服务的例子是传播 CRL 或安全策略更新或提供安全时间服务的那些服务。使用有

效性服务可以帮助确保依赖方具有用于通知管理决定的当前数据。

[1259] 典型情况下，并非所有系统实体需要关于凭证和安全数据有效性的最新信息。例如，每当使用许可或获得新的许可时，并非所有消费者设备将使用在线证书状态协议(Online Certificate Status Protocol OCSP)服务来证实许可服务器的证书链。然而，许可服务器可以一定频率地使用OCSP服务来检查订户凭证的有效性。策略（其可以被容易地更新）可以确定必须何时使用服务以及使用什么服务。通过提供动态地更新策略的机会，许可服务器可以适于操作上的改变。从而，安全策略可以根据经验、技术进展和市场因素来发展。

[1260] 安全对象的有向自毁

[1261] 当实体的安全处理的完整性并非是不可信的时，由实体自毁凭证和数据是适当的。当此选项是可用的时，它常常是最直接的、迅速的和高效的撤销方法。在几乎不怀疑完整性被破坏时它是特别有用的，并且双向通信支持用于允许销毁以及销毁已经完成的验证的具体方向的协议。

[1262] 存在常常对已经销毁或禁止来说是有用的多个安全对象。例如，当设备离开域或者内容许可超时时，相关联的对象包含密钥并且可以用来访问要被销毁的内容是有用的。在此的其它地方更详细描述的代理控制程序对于自毁机制的实现来说是很合适的。代理可以被设计来销毁安全存储装置（例如，状态数据库）中的状态以便影响域成员资格中的变化或者移除不再可用的密钥（例如，由于成员资格或策略的改变）。

[1263] 排除

[1264] 排除是用于阻挡危险分子（或危险分子组）参与将来消费商品及服务的补救机制。由于排除所强加的严重结果，所以它一般只是被用为当环境批准时的最后手段。排除依赖于用于把危险分子有效地列入黑名单由此禁止它们消费媒体和媒体相关服务的机制。黑名单的传播依赖于用于启用此补救的更新机制。然而，排除不必提供用于把危险分子恢复为信任状态的更新机制。

[1265] 密钥排除

[1266] 密钥排除是用于采用在任何给定时间可以判定在逻辑上把一些接收器子集从解密将来内容的能力中排除在外的方式来向一组接收器广播密钥信息。这通过使用高效技术构造广播密钥块(Broadcast KeyBlock BKB)来激活，所述广播密钥块包括为巨大的接收器组的每个成员解密内容所必须的信息。BKB采用可以被容易更新的方式来构造，把该组的一个或多个成员从解密所述内容的能力中排除。换句话说，BKB的设计允许权威机构利用新的BKB来更新系统，使得内容供应者可以特别地排除目标设备组利用所述BKB，即便她／他可能有权访问它也是如此。

[1267] 此机制对克隆攻击来说是特别有效的，其中盗版者对合法设备进行反向工程，提取其密钥，继而把那些密钥的拷贝散布到克隆设备。克隆除了不一定遵循管理模型之外，表面像原始那样做动作。一旦发现受到危害，就可以部署更新的BKB，所述BKB排除受到危害的设备以及其所有的克隆。然而，密钥排除承担一些存储、传输和计算开销，在一些情况中这使与其它方法相比没那么高效。当没有广播内容时或当存在暗道时特别如此。

[1268] 回避(shunning)

[1269] 回避是与排除行为非常类似的补救机制，但是没有那么严重的影响。实质上，它是

用于由于运行时策略判定而拒绝服务的手段。代替通过有向自毁或经由密钥排除访问拒绝来禁止设备能力的更苛刻的方法，回避通过使服务供应者拒绝向其提供服务来提供更简单的禁止设备的方法。在当前趋向于通过使用外部提供的服务来扩展设备的价值的情况下，回避变为更加有效的安全机制。

[1270] 设备回避由策略来驱动并且可以用来排斥没有生成策略所要求的所有适当凭证的实体（例如，客户端、服务器和特定任务播放器）。策略例如可以要求实体证明它已经执行了最新的安全更新。因此回避可以是撤销的结果或无法采取某个特定动作。可以使用审查服务和诸如在‘551 申请中所描述的那些服务依照对等方式来使回避便于进行。数据证明服务（例如，有效性服务的实例）还可以在策略强制实施时执行回避。在系统实体已经被回避之后，可以向它通知没能符合服务策略的具体凭证或对象。这可以触发所回避的实体通过适当的服务接口来更新对象。

[1271] 期满

[1272] 期满是依赖于一些时间事件来使凭证或对象无效的补救机制。期满在允许对媒体或媒体服务进行临时访问时是有效的；一旦这些已经期满，那么管理模型确保不再允许访问。有效的使用期满可以要求更新机制，借此可以刷新凭证或对象以便能够继续访问媒体或媒体服务。

[1273] 凭证的期满

[1274] 证实的密钥可以具有被分配来用于保护依赖方的各种期满属性。凭证的期满可以用来确保其证书已经期满的实体被拒绝服务并且结合密钥翻滚（rollover）和密钥更新过程来使用。当预计实体频繁地连接到广域网时，最佳实施方式指示定期地更新凭证及其它安全数据。另一最佳实施方式在于保持这些对象的有效时段尽量合理地短。在有效性检查策略中诸如重叠有效时段和宽限时段之类的各种技术可以用来确保在转变期间的平滑操作。短的有效时段还有助于减小 CRL 的大小。

[1275] 链路的期满

[1276] 如前所述，可以向链路对象分配有效时段。当期满时，链路被认为是无效的并且 DRM 引擎在构成它的图时不考虑该链路。此机制可以用来允许对商品及服务进行临时访问。链路可以被更新，使得只要策略允许就可以准许对媒体继续访问。因为在一个实施例中链路是相对轻便的自保护对象，所以它们可以经由对等协议被容易地分发。

[1277] 可更新机制：应用和策略可更新性

[1278] 高效的可更新性一般需要对协议失败迅速展开补救，所述协议失败常常是在安全应用中（包括在 DRM 系统中）所常见的主要安全问题。然后可以使用软件更新来更新企业逻辑和安全协议。当应用被设计成用于把安全策略和信任策略与应用逻辑相分离时，可以使用独立的机制来更新策略；这是不那么有风险的方法。实际上，可以使用对等公开机制来迅速地更新策略。否则，可以使用应用部署者的软件更新方法更新安全和信任策略。

[1279] 为权利作业使用权利工具

[1280] 通常当可能时希望使用相对轻便的工具。使用具有有限有效时段的凭证和检查有效日期的策略可以帮助把实体的整个群体保持为可管理的大小并且不必太过迅速地发展 CRL。避免实体而不是把它从对密钥的访问中排除可以扩展 BKB 的使用期；此外，它还有允许细粒度策略的优点，所述策略可以是临时的并且随环境而改变。可以使用不同的 CRL 而

不是 BKB，其中所述 CRL 跟踪不同任务播放器感兴趣的特定类型的凭证，所述 BKB 可以被部署在它们最有效的地方（诸如处理克隆接收者）。当可以预期在线有效性服务对时间和精力投资能提供合理回报时，策略可以引导使用那些服务，其中未用过的凭证是非常重要的，并且其中较慢的撤销机制是不够的。当节点可能具有完整性并且可以被预计为做正确的事时，并且当需要撤消许可或安全对象（诸如用于预订的链路或域链路）时，那么合理的方法一般是告诉节点销毁所述对象。在这种情况下，不必公开宣布许可是无效的并且不必部署 BKB 或为域重新设定密钥。由本地策略或权威命令驱动的自毁是用于撤销的最有效的方法之一。

[1281] 应当理解，虽然已经描述了各种撤销、更新、补救及其它技术和实施方式，然而应当理解，不同的情况要求不同的工具，而且可以使用这些技术中的任何适当组合或不使用这些技术来实施这里所描述系统和方法的优选实施例。

[1282] 网络服务安全

[1283] 以下论述举例说明了一些可能与以下实施例有关的安全性考虑和技术，在所述实施例中结合诸如在‘551 申请中所描述的那些联网的服务组织系统和方法使用上述 DRM 引擎和应用。

[1284] 用于使用诸如这里所公开的那些 DRM 引擎和体系结构的 DRM 系统的实际实现方式常常执行联网事务来访问内容和 DRM 对象。在这样的上下文情境中，在‘551 申请中所描述的系统和方法尤其可以用来使消息层安全标准化，包括实体认证和授权属性（角色）的格式。

[1285] 为了论述，在 DRM 系统中出现的事务可以根据所访问、获取或操纵的信息类型被分成至少两个一般类别：

[1286] 内容访问事务涉及直接访问或操纵受 DRM 系统保护的媒体或企业内容或其它敏感信息。内容访问事务的例子包括再现受保护的视频剪辑，把受保护音频曲目的拷贝烧录到紧凑盘上，把被保护文件移动到便携式设备，电子邮件发送机密文档等。内容访问事务一般涉及直接访问内容保护密钥并且在用户指导下在消费点执行。

[1287] 对象事务是其中用户或系统获取由 DRM 系统所定义的对象或与之相交互的事务，所述对象采用某种方式来管理对受保护内容的访问。这种对象包括 DRM 许可、成员资格令牌、撤销列表等。通常在为执行内容访问事务所必须的所有担保都是可用的之前，要求一个或多个对象事务。对象事务一般其特征在于使用某种类型的通信网络来在消费点汇编 DRM 对象。

[1288] 这两种类型的事务定义了通常与大部分 DRM 系统有关的两个管理点。图 38 示出了一对典型的交互，其中具有 DRM 功能的客户端 3800 从适当的 DRM 许可服务 3804 中请求 DRM 许可 3802。在图 38 所示出的例子中，DRM 许可 3802 从 DRM 许可服务 3804 被发送到客户端 3800，在那里它被评估以便提供对内容 3806 的访问。

[1289] DRM 系统一般要求依照防止对内容未经授权的访问以及创建保护所述内容的对象的方式来执行内容访问和对象事务。然而，用于两种类型事务的安全问题自然是不同的。例如：

[1290] 内容访问事务可以要求认证人类委托人，检查安全再现计数，评估 DRM 许可以便导出内容保护密钥等。对合法执行内容访问事务的主要威胁是破坏了用于保护对象和其中

数据的抗篡改界限。

[1291] 对象事务通常涉及在要求 DRM 对象的实体和可以提供它的实体之间的通信信道。因而，对象事务面临基于通信的威胁，诸如中间人攻击、重播攻击、拒绝服务攻击和其中未经授权的实体获取它们不应当合法拥有的 DRM 对象的攻击。

[1292] 通常，对象事务涉及两个相交互实体的认证，在它们之间传递的消息的保护以及所述事务的授权。这种事务的主要目的在于从合法源中采集完整保护的 DRM 对象使得可以执行内容访问事务。从内容访问事务的视角看，用于获得合法 DRM 对象的机制以及在获得它们过程中所使用的担保信息在本质上是无关的；这些机制可以（并且优选应当）对于内容访问本身来说是不可见的。此问题的自然分离在优选实施例中导致了分层通信模型，分层通信模型用于把信任通信框架与在其之上所构建的应用区分开来。

[1293] 在图 38 中所示出的简化的许可获取和消费例子模糊了通常在实际应用中是重要的一些细节。例如，没有示出 DRM 许可服务怎样验证请求 DRM 许可的实体实际上是合法的 DRM 客户端而不是恶意的实体，所述恶意的实体试图获得未经授权的许可或通过消耗网络带宽和处理能力来拒绝对合法客户端服务。也没有示出当敏感信息通过连接客户端和服务的通信信道移动时怎样保护所述敏感信息的机密性和完整性。

[1294] 在图 39 中示出了此示例性事务的更详细的视图。参照图 39，从应用层内容再现客户端 3800 和 DRM 许可服务器 3804 的观点来，虚线表示逻辑事务。下面的堆栈 3900 表示用于确保在两个端点之间的受信任和受保护递送的处理层。

[1295] 在图 39 中，再现客户端 3800 从 DRM 许可服务器 3804 请求许可 3802。图中的虚线表明信息原始的源和最终消费者是内容再现客户端 3800 和 DRM 许可服务器 3804。然而，在实践中消息有效载荷实际上可以由在应用层逻辑和连接两个端点的不安全通信信道 3902 之间插入的几个处理层来处理。

[1296] 用于把应用层组件与不安全的通信信道相分离的处理层一块被认为是安全堆栈。安全堆栈可以被认为是用于确保在信任端点之间对消息进行完整性保护、机密递送的安全消息发送架构。分层堆栈模型提供了以下优点，诸如：

[1297] (1) 应用层逻辑的设计者不必花费精力来开发用于连接端点的基础安全通信机制。信任的消息发送基础结构是共用的设计模式，其一旦被设计就可以部署在许多不同的情形下，而不管它们所支持的应用层逻辑如何。

[1298] (2) 消息发送架构本身可以保持不知道它正传达消息的精确语义，并且把它的精力集中在防止与通信相关的攻击和对消息发送端点可靠性的攻击。

[1299] 在一个实施例中，安全堆栈由几个不同的处理层组成，如下所述。在一个实施例中，在‘551 申请中所描述的服务组织系统和方法可以用来提供安全堆栈的一些或全部操作。

[1300] 认证

[1301] 在一个实施例中，可以认证消息发送端点。认证是给定端点用来向另一个端点证明由为此目的所信任的权威机构已经向该端点给予有效名称的过程。命名权威机构应当被事务中的依赖端点所信任；一般由用于部署信任技术的组织来承担建立这种权威机构。

[1302] 用于表明有效名占有情况的通用机制使用公钥密码和数字签名。使用此方法，向实体提供三个信息：

[1303] (1) 用于向实体提供标识符的可区别名称；

[1304] (2) 由公钥和秘密私钥组成的非对称密钥对；和

[1305] (3) 用于断言私钥的持有者具有给定的可识别名称的数字签名证书。

[1306] 所述证书把可区别名称和私钥绑定在一起。使用私钥来签名信息的实体被信任为具有给定可区别名称。可以只使用公钥来验证签名。例如，认证可以是基于 X.509v3 标准的。

[1307] 由于在一个实施例中，可以证明拥有所证实私钥的实体被信任为具有在证书中所表明的可区别名称，所以保护用于签名信息的私钥变为重要的考虑事项。实际上，使用私有签名密钥的能力定义了由可区别名称所标识的实体界限。在应用层，发送方和接收方必须知道消息来源于信任的对方。因而在一个实施例中，重要的是应用层逻辑本身是被认证的实体的一部分。为此，在一个实施例中安全堆栈和依赖于它的应用层优选被封闭在信任界限内，以致在信任界限内所包含的子系统被认为共享对实体的私有消息签名密钥的访问。

[1308] 授权

[1309] 上述认证机制向分布式消息发送端点证明它们的通信方的身份是值得信任的。在许多应用中，此信息太过粗略 - 做出关于某些事务的策略判定，可能需要关于端点能力和特性的更详细的信息。例如在图 38 的上下文情境中，内容再现客户端可能不仅需要知道它正与被认证的端点通信，而且需要知道它是否与已经被认为有能力提供有效 DRM 许可对象的服务通信。

[1310] 安全堆栈的实施例经由授权机制提供了用于声明、传达和应用策略的机制，所述策略是基于关于被认证的实体的更细粒度的属性的。使用此机制，已经拥有认证凭证的实体被分配有角色声明，所述角色声明用于使一组命名的能力与所述实体的可区别名称相关联。例如，可以为 DRM 客户端和 DRM 许可服务器定义角色名称。

[1311] 命名的角色旨在传达由实体所保持的特定能力。在实践中，可以通过声明在实体的可区别名称和角色名称之间的关联来把角色附着到实体。就像用于使密钥与可区别名称相关联的认证证书一样，在一个实施例中用于授权的角色声明由可以不同于名称发行者的信任角色权威机构来签名。在实体内，角色声明连同认证凭证一起被验证，作为用于准许对消息发送端点的应用层进行访问的条件。

[1312] 实体可以保持所构建的应用要求的那么多的角色属性。图 40 中的例子示出了具有多种角色的实体：用于表明作为 DRM 客户端起作用的能力的一个角色和两个服务角色。例如，一个实体同时可以是 DRM 客户端、DRM 对象供应者和安全数据供应者。在一个实施例中，SAML1.1 用于关于实体属性的声明。

[1313] 消息安全性

[1314] 安全堆栈的底层是消息安全层，用于向消息提供完整性、机密性和新鲜 (freshness) 保护，并且缓和对通信信道的攻击的风险，诸如重播攻击。在消息安全层中：

[1315] ●使用实体的私有消息签名密钥来签名在应用层处理之间的消息，提供了完整性保护以及对中间人攻击的抵抗。

[1316] ●使用由目的地实体所保持的公钥来加密消息。这保证无意的接收者无法读取在运输中拦截的消息。

[1317] ●把特殊时间和时间戳添加到消息，提供了对重播攻击的免疫性并且便于证明在

消息发送端点之间的实况性。

[1318] ●使用服务器时间戳来更新 DRM 引擎的信任时间

[1319] 在一个说明性实施例中，提供了对 AES 对称加密、RSA 公钥密码、SHA-256 签名摘要和用于在消息中发信号表示其它算法的机制的支持。

[1320] 15. 引导协议

[1321] 在一些实施例中，使用引导 (bootsrap) 协议来向诸如设备和软件客户端之类的实体递送初始机密的配置数据。例如，当实体想要加入更大网络或系统并且使用密码协议与其它实体通信时，它可能需要利用包括一组（共享、秘密和公共）密钥而被配置。当利用个性化数据来预先配置实体是不可能的或不切实际的时，它需要使用密码协议来“引导”本身。

[1322] 下述示例性协议使用共享秘密作为用于利用一组密钥和其它配置数据来引导实体的基础。在下面章节里，使用以下符号：

[1323] ■ E(K, D) 是利用密钥 K 来加密某个数据 D。

[1324] ■ D(K, D) 是利用密钥 K 来解密某个加密的数据 D。

[1325] ■ S(K, D) 是利用密钥 K 来签名某个数据 D。这可以是公钥签名或 MAC。

[1326] ■ H(D) 是数据 D 的消息摘要。

[1327] ■ V(K, D) 是利用密钥 K 来验证某个数据 D 上的签名。它可以是公钥签名或 MAC 的验证。

[1328] ■ CertChain(k) 是与公钥 K 相关联的证书链。K 值被包括在链中的第一证书中。

[1329] ■ CertVerify(RootCert,CertChain) 是验证证书链 CertChain（包括在链的第一证书中找到的公钥）在根证书 RootCert 下是有效的

[1330] ■ A|B|C|... 是通过拼接单个字节序列 A、B、C、... 所获得的字节序列

[1331] ■ CN(A) 是 A 的规范的字节序列

[1332] ■ CN(A, B, C, ...) 是复合字段 A, B, C... 的规范字节序列

[1333] 1.38. 初始状态

[1334] 1.38.1. 客户端

[1335] 在一个实施例中，客户端具有以下引导令牌组（在制造时间预先加载和 / 或被加载到固件 / 软件中）：

[1336] ■一个或多个只读证书，其是引导过程的信任根 :BootRootCertificate

[1337] ■一个或多个秘密引导认证密钥 :BAK（共享）

[1338] ■可选的秘密引导种子产生密钥（对每个客户端来说是唯一的）BSGK。如果客户端具有良好的随机数据源，那么不需要此种子。

[1339] ■一些信息 ClientInformation，客户端需要向引导服务给予该信息以便获取其机密密钥（例如，ClientInformation 可以包括设备的序列号、制造商名称等）。此信息由一列属性组成。每个属性是（名称，值）对。

[1340] 客户端可以利用多个 BootRootCertificate 证书和 BAK 认证密钥来配置，以便能够利用可以要求不同信任域的不同引导服务器参与引导协议。

[1341] 1.38.2. 服务器

[1342] 在一个实施例中，服务器具有以下令牌：

[1343] ■至少一个客户端引导认证密钥:BAK(共享秘密)

[1344] ■用于签名的公钥/私钥对(Es,Ds)

[1345] ■证书链 ServerCertificateChain = CertChain(Es),其在根证书之一(BootRootCertificate)下是有效的

[1346] ■用于加密的公钥/私钥对(Ee,De)

[1347] 1.39.协议描述

[1348] 下面在图 41 中示出了并且描述了引导协议的说明性实施例。在该过程期间(例如当验证签名或证书链时)的失败会导致错误并且停止协议进展。

[1349] BootstrapRequestMessage(引导请求消息)

[1350] 客户端向服务器发送请求,用于表明它想要发起引导会话并且提供一些初始参数(例如,协议版本、简档等),以及会话 ID(用于防止重播攻击)和它可以参与到其中的信任域列表。下表示出了 BootStrapRequestMessage 的说明性格式:

[1351]

<table>
<tr><td>名称</td><td colspan="3">BootstrapRequestMessage</td></tr>
<tr><td>属性</td><td colspan="3"><table>
<tr><td>名称</td><td>描述</td></tr>
<tr><td>协议</td><td>协议的符号名</td></tr>
<tr><td>版本</td><td>协议版本</td></tr>
<tr><td>简档</td><td>用于此协议/版本的简档名称</td></tr>
</table></td></tr>
<tr><td>方向</td><td colspan="3">客户端 → 服务器</td></tr>
<tr><td>有效载荷</td><td colspan="3">BootstrapRequest（引导请求）<table>
<tr><td>名称</td><td>类型</td><td>描述</td></tr>
<tr><td>SessionId</td><td>字符串</td><td>由客户端所选择的唯一会话 ID</td></tr>
<tr><td>TrustDomains</td><td>字符串列表</td><td>客户端可以参与到其中的所有信任域的名称。</td></tr>
</table></td></tr>
<tr><td>预期响应</td><td colspan="3">ChallengeRequestMessage（质询请求消息）</td></tr>
</table>

[1352] 协议和版本消息属性指定了客户端使用哪个协议规范，并且简档字段标识了一组预定义的密码协议和用于交换消息和数据的编码格式。

[1353] 客户端选择 SessionId（会话 ID），所述 SessionId 应当对该客户端来说是唯一的并且不被重新使用。例如，该客户端的唯一 ID 和增加的计数器值可以被用为产生唯一会话 ID 的方式。

[1354] 在一个实施例中，客户端还发送它已经被配置的所有信任域的列表。

[1355] 在一个实施例中，服务器接收 BootstrapRequestMessage 并且执行以下步骤：

[1356] ■检查它支持由客户端所请求的指定协议、版本和简档。

[1357] ■产生特殊时间 (Nonce)（有力的随机数）。

[1358] ■选择性地产生 Cookie 以便携带诸如时间戳、会话令牌之类的信息或遍及会话期间保持不变的任何其它服务器端信息。cookie 值只对服务器有意义，并且被客户端认为是不透明的数据块。

[1359] ■从 BootstrapRequestMessage 提取 SessionId 值。

[1360] ■产生质询 (Challenge)：Challenge = [Nonce, Ee, Cookie, SessionId]。

[1361] ■计算 S(Ds, Challenge) 以便利用 Ds 来签名所述质询。

[1362] ■构造 ChallengeRequestMessage 并且作为响应把它发送回到客户端。

[1363] Challen geRequestMessage（质询请求消息）

[1364] 下表示出了用于 ChallengeRequestMessage 的说明性格式：

[1365]

<table>
<tr><td>名称</td><td>ChallengeRequestMessage（质询请求消息）</td></tr>
<tr><td>方向</td><td>服务器 → 客户端</td></tr>
<tr><td>有效载荷</td><td>
质询（Challenge）
<table>
<tr><td>名称</td><td>类型</td><td>描述</td></tr>
<tr><td>Nonce（特殊时间）</td><td>字节序列</td><td>服务器产生的随机特殊时间</td></tr>
<tr><td>ServerEncryptionKey</td><td>字节序列</td><td>用于消息有效载荷加密的所编码的公钥 Ee</td></tr>
<tr><td>Cookie</td><td>字节序列</td><td>服务器产生的不透明数据</td></tr>
<tr><td>SessionId</td><td>字符串</td><td>客户端产生的会话 ID</td></tr>
<tr><td>签名</td><td>字节序列</td><td>质询的规范字节序列的编码数字签名 S（Ds，CN（Challenge））= CN（CN（Nonce），CN（ServerEncryptionKey），CN（Cookie），CN（SessionId））</td></tr>
</table>
ServerCertificateChain（服务器证书链）
<table>
<tr><td>名称</td><td>类型</td><td>描述</td></tr>
<tr><td>TrustDomain</td><td>字符</td><td>证书链在其中有效的信任域</td></tr>
<tr><td>证书</td><td>字节序列的列表</td><td>用于形成链的编码证书列表：CertChain（Es）。数组中的第一证书证实公钥 Es，并且</td></tr>
</table>
</td></tr>
</table>

[1366]

			以下每个证书随后证实之前证书的公钥。数组中的最后证书具有被信任域的根 CA 证书证实的公钥	
响应于	BootstrapRequestMessage（引导请求消息）			

[1367] 在一个实施例中，在接收 ChallengeRequestMessage 之后，客户端执行以下步骤：

[1368] ■验证证书链 ServerCertificateChain 在根证书 BootRootCertificate 下有效：CertVerif(BootRootCertificate, ServerCertificateChain)。

[1369] ■从 ServerCertificateChain 提取公钥 Es。

[1370] ■验证质询的签名：V(Es, Challenge)。

[1371] ■当发送了 BootRequestMessage 时检查 SessionId 匹配为所述会话选择的一个 ID。

[1372] ■构造 ChallengeRequestMessage 并且把它发送到服务器。

[1373] ChallengeResponseMessage

[1374] 为了产生 ChallengeResponseMessage，客户端执行以下步骤：

[1375] ■使用两个以下方法之一来产生会话密钥 SK：

[1376] ○直接使用安全随机密钥发生器

[1377] ○间接地使用 Nonce 和 BSGK：计算 HSK = H(BSGK |Nonce)，并且设置 SK = HSK 的最初 N 个字节

[1378] ■ 产 生 包 含 [Challenge, ClientInformation, SessionKey] 的 ChallengeRepsonse 对象。这里，质询是来自先前接收的 ChallengeRequestMessage 的一个，ServerEncryptionKey 被省略了。

[1379] ■计算 S(BAK, ChallengeResponse) 以便利用 BAK 来签名响应。

[1380] ■利用 SK 加密所签名的 ChallengeReponse：E(SK, [ChallengeResponse, S(BAK, ChallengeResponse)]

[1381] ■利用服务器的公钥 Ee 来加密 SessionKey

[1382] 构造 ChallengeRequestMessage 并且把它发送到服务器

[1383]

名称	ChallengeResponseMessage（质询响应消息）
方向	客户端 → 服务器

[1384]

有效载荷

SessionKey [利用 Ee 加密]

名称	类型	描述
SessionKey	字节序列	利用服务器的公钥 Ee 加密的编码的会话密钥 SK

ChallengeResponse（质询响应） [利用 SK 加密]

名称	类型	描述
质询	对象	质询 名称 / 类型 / 描述 特殊时间 / 字节序列 / 服务器产生的随机特殊时间 Cookie / 字节序列 / 服务器产生的不透明数据 SessionId（会话 ID） / 字符串 / 唯一的会话 ID
ClientInformation（客户端信息）	属性数组	0 或更多属性对象的数组: 属性 名称 / 类型 / 描述 名称 / 字符串 / 属性的名称 值 / 字符串 / 属性值
SessionKey (会话密钥)	字节序列	秘密会话密钥 SK 的编码值
签名	字节序列	规范字节序列 CN（ChallengeResponse）的编码数字签名 S（BAK，CN（ChallengeResponse））= CN（CN

[1385]

			（Challenge），CN（ClientInformation），CN（SessionKey））
预计的响应	BootstrapResponseMessage(引导响应消息)		

[1386] 服务器接收 BootstrapChallengeResponse 并且执行以下步骤：

[1387] ○使用其私钥 De 来解密会话密钥 SK :D(De, SessionKey)

[1388] ○利用来自先前步骤的会话密钥 SK 解密 ChallengeResponse :D(SK, Challenge)

[1389] ○验证质询的签名 :V(BAK, ChallengeResponse)。

[1390] ○检查会话密钥 SK 匹配用于解密的密钥

[1391] ○如果需要那么检查 Cookie 和 Nonce 值（例如时间戳）

[1392] ○当发送了 BootRequestMessage 时检查 SessionId 匹配为所述会话选择的一个 ID。

[1393] ○构造 BootstrapResponseMessage 并且把它发送到服务器。

[1394] BootstrapResponseMessage

[1395] 为了产生 BootstrapResponseMessage，服务器执行以下步骤：

[1396] ○解析在 ChallengeResponseMessage 中所接收的 ClientInformation 并且查找或产生客户端配置数据，其中为了此引导请求需要发送所述客户端配置数据（对于表示客户端的节点，这可以包括机密密钥 (Ec/Dc))。服务器一般使用 Nonce 和 Cookie 值来帮助获取客户端的正确信息。

[1397] ○利用 SessionId 和配置数据来创建 BootstrapResponse

[1398] ○计算 S(Ds, BootstrapResponse) 以便利用 Ds 来签名数据

[1399] ○利用会话密钥 SK 加密所签名的 BootstrapResponse :E(SK, [BootstrapResponse, S(Ds, BootstrapResponse)]

[1400]

<table>
<tr><td>名称</td><td colspan="3">BootstrapResponseMessage(引导响应消息)</td></tr>
<tr><td>方向</td><td colspan="3">服务器 → 客户端</td></tr>
<tr><td>有效载荷</td><td colspan="3">BootstrapResponse （引导响应）[利用 SK 加密]<table><tr><td>名称</td><td>类型</td><td>描述</td></tr><tr><td>SessionID</td><td>字符</td><td>会话 ID</td></tr><tr><td>数据</td><td>字节序列</td><td>客户端的配置数据</td></tr><tr><td>签名</td><td>签名</td><td>规范字节序列 CN（BootstrapResponse）的数字签名
S（Ds，CN（BootstrapResponse））
= CN(CN(SessionID), CN(Data))</td></tr></table></td></tr>
<tr><td>响应于</td><td colspan="3">ChallengeResponseMessage</td></tr>
</table>

[1401] 1.40. 信任域

[1402] 在一个实施例中，每个信任域包括根证书权威机构和对该域来说唯一的名称。当客户端发送 BootstrapRequest 时，它标识它愿意接受的所有信任域（即，哪些证书它认为是有效的）。服务器从由客户端所发送的列表中选择信任域，如果它支持的话。

[1403] 1.41. 签名

[1404] 在一个实施例中，每当在消息有效载荷中使用签名时，对在消息的签名部分中所包含的数据字段的规范字节序列计算签名。根据字段值而不是根据字段值的编码来计算规范的字节序列。每个简档优选定义了用于为每种消息类型计算字段的规范字节序列的算法。

[1405] 1.42. 简档

[1406] 引导协议的简档是对各个密码和串行格式的一组选择。每个简档优选具有唯一的名称，并且包括选择：

[1407] ●公钥加密算法

[1408] ●公钥签名算法

[1409] ●密秘密钥加密算法

[1410] ●秘密密钥签名算法

[1411] ●公钥编码

[1412] ●摘要算法

[1413] ●规范的对象串行化

[1414] ●证书格式

[1415] ●最小特殊时间大小

[1416] ●消息编组

[1417] 附录 A

[1418] 下面是具有多个联锁签名的控制器对象的例子。

[1419] 注意 :在此例子中,内容密钥未被加密。

[1420]

<Controller xmlns="http://www.intertrust.com/Octopus/1.0"
id="urn:x-octopus.intertrust.com:controller:37A50262EE3389A14ABC0BC7BE5D43E5">
<ControlReference>
<Id>urn:x-octopus.intertrust.com:control:0001</Id>
<Digest>
<DigestMethod xmlns="http://www.w3.org/2000/09/xmldsig#"
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
<DigestValue
xmlns="http://www.w3.org/2000/09/xmldsig#">1z95n10V7CBiKs/rSQdXvKyZmfA=</DigestValue>
</Digest>
</ControlReference>
<ControlledTargets>
<ContentKeyReference>
<Id>urn:x-octopus.intertrust.com:content-key:2001</Id>
</ContentKeyReference>
<ContentKeyReference>
<Id>urn:x-octopus.intertrust.com:content-key:2002</Id>
</ContentKeyReference>
<ContentKeyReference>
<Id>urn:x-octopus.intertrust.com:content-key:2003</Id>
</ContentKeyReference>
</ControlledTargets>
</Controller>
<Signature Id="Signature.0" xmlns="http://www.w3.org/2000/09/xmldsig#">
<SignedInfo>
<CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
<Reference URI="urn:x-octopus.intertrust.com:controller:37A50262EE3389A14ABC0BC7BE5D43E5">

[1421]

<Transforms>

<Transform Algorithm="http://www.intertrust.com/Octopus/xmldsig#cbs-1_0" />

</Transforms>

<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />

<DigestValue>G1zXF9Sz/zCwH6MaFm0ObOQcxuk=</DigestValue>

</Reference>

</SignedInfo>

<SignatureValue>mjoyW+w2S9iZDG/ha4eWYD1RmhQuqRuuSN977NODpzwUD02FdsAICVjAcw7f4n
FWuvtawW/clFzYP/pjFebESCvurHUsEaR1/LYLDkpWWxh/LlEp4r3yR9kUs0AU5a4BDxDxQE7nUdqU9
YMpnjAZEGpuxdPeZJM1vyKqNDpTk94=</SignatureValue>

<KeyInfo>

<X509Data>

<X509Certificate>MIIC6jCCAlOgAwIBAgIBBjANBgkqhkiG9w0BAQUFADCBszELMAkGA1UEBhM
CVVMxEzARBgNVBAgTCkNhbGlmb3JuaWExFDASBgNVBAcTC1NhbnRhIENsYXJhMSAwHgYDVQ
QKExdJbnRlcnRydXN0IFRIY2hub2xvZ2llczEUMBIGA1UECxMLT2N0b3B1cyBEUk0xGDAWBgNVBA
MTD09jdG9wdXMgVGVzdCBDQTEnMCUGCSqGSIb3DQEJARYYb2N0b3B1cy10ZXN0LWNhQDhwdX
MubmV0MB4XDTA0MDQwODAwNTUyOVoXDTA0MDUwODAwNTUyOVowgcExCzAJBgNVBAYTA
lVTMRMwEQYDVQQIEwpDYWxpZm9ybmlhMRQwEgYDVQQHEwtTYW50YSBDbGFyYTEgMB4GA1
UEChMXSW50ZXJ0cnVzdCBUZWNobm9sb2dpZXMxFDASBgNVBAsTC09jdG9wdXMgRFJNMR8wHQ
YDVQQDExZPY3RvcHVzIFRlc3QgTm9kZSAwMDAxMS4wLAYJKoZIhvcNAQkBFh9vY3RvcHVzLXRl
c3Qtbm9kZS0wMDAxQDhwdXMubmV0MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDU8AJ
QArJg+VTuwUO2fMv5sCtfmZECyJjA0vbgQc+cPXpfeIdACiCL1n1eml/ZLlu7ZaRwQeo1yJSeK57bxv+zh
W14F1jnqS/IKLG84RG1eoMiOT1lhErb2nU3xT0KCgxsEXFAbfwAYnLX7hpy/1ho2mTmJbgksWoPrPw3x
MPCYwIDAQABMA0GCSqGSIb3DQEBBQUAA4GBAH1rHStXcQkFmcYhl5zck6twsNIRF+/1HZGuTGK
eb6+J2ZLk6sNUWXLOlD1oPRMde7X1RiqpDNkbG4xoPoxHiK9VdfBstjv9Q8iUceziMIXVV/q+XJMd7Hf
BJq25XqBScS9/RAKKKwuRRkQHEV3uBABvLSCzIRSJH9bFuYzNeVne</X509Certificate>

</X509Data>

</KeyInfo>

</Signature>

<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">

<SignedInfo>

<CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />

[1422]

<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1" />

<Reference URI="#Signature.0">

<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />

<DigestValue>AqPV0nvNj/vc51IcMyKJngGNKtM=</DigestValue>

</Reference>

<Reference URI="urn:x-octopus.intertrust.com:controller:37A50262EE3389A14ABC0BC7BE5D43E5">

<Transforms>

<Transform Algorithm="http://www.intertrust.com/Octopus/xmldsig#cbs-1_0" />

</Transforms>

<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />

<DigestValue>G1zXF9Sz/zCwH6MaFm0ObOQcxuk=</DigestValue>

</Reference>

</SignedInfo>

<SignatureValue>TcKBsZZy+Yp3doOkZ62LTfY+ntQ=</SignatureValue>

<KeyInfo>

<KeyName>urn:x-octopus.intertrust.com:secret-key:2001</KeyName>

</KeyInfo>

</Signature>

<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">

<SignedInfo>

<CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />

<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1" />

<Reference URI="#0">

<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />

<DigestValue>AqPV0nvNj/vc51IcMyKJngGNKtM=</DigestValue>

</Reference>

<Reference URI="urn:x-octopus.intertrust.com:controller:37A50262EE3389A14ABC0BC7BE5D43E5">

<Transforms>

<Transform Algorithm="http://www.intertrust.com/Octopus/xmldsig#cbs-1_0" />

</Transforms>

<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />

<DigestValue>G1zXF9Sz/zCwH6MaFm0ObOQcxuk=</DigestValue>

[1423]

```
</Reference>
</SignedInfo>
<SignatureValue>qAunQpXC18kl8Veo8UHbcXTqHCA=</SignatureValue>
<KeyInfo>
<KeyName>urn:x-octopus.intertrust.com:secret-key:2002</KeyName>
</KeyInfo>
</Signature>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
<SignedInfo>
<CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1" />
<Reference URI="#0">
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
<DigestValue>AqPV0nvNj/vc51IcMyKJngGNKtM=</DigestValue>
</Reference>
<Reference URI="urn:x-octopus.intertrust.com:controller:37A50262EE3389A14ABC0BC7BE5D43E5">
<Transforms>
<Transform Algorithm="http://www.intertrust.com/Octopus/xmldsig#cbs-1_0" />
</Transforms>
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
<DigestValue>G1zXF9Sz/zCwH6MaFm0ObOQcxuk=</DigestValue>
</Reference>
</SignedInfo>
<SignatureValue>bRxLSM82d4ktWsYz6uhBxzJfsOo=</SignatureValue>
<KeyInfo>
<KeyName>urn:x-octopus.intertrust.com:secret-key:2003</KeyName>
</KeyInfo>
</Signature>
</Bundle>
```

[1424] 附录 B

[1425] 此附录 B 给出了在使用在此的其它地方所描述的示例性章鱼 DRM 引擎的系统的一个实施例中对象的 XML 编码。对于特定应用来说，可以通过导入下面示出的 XML 模式（“章鱼 XML 模式”）并且添加应用专用的元项（例如，用于撤销的扩展）来创建应用专用的 XML

模式。在一个实施例中，用 XML 编码对象需要能够相对于应用专用的 XML 模式来证实。下面可以找到对这些 XML 编码的附加可能的约束。

[1426] 在此附录 B 所举例说明的例子中，用于所有 DRM 对象的基础 XML 模式类型为 OctopusObjectType。这意味着所有对象支持属性和扩展。从此基础类型来导出每个章鱼对象元项的类型。这些类型可以集合诸如 ContentKeyType（内容密钥类型）的 SecretKey（秘密密钥）元项之类的其它元项。

[1427] 在此示例性实施例中，按照扩展来描述 Scuba 密钥分发系统密钥：于是 ScubaKeys 元项是扩展元项的孩子。同样适用于具有 Torpedo（鱼雷）扩展的撤销密钥。

[1428] 如在此的其它地方所描述，存在各种章鱼对象（例如，内容密钥、保护器、控制器、控制、节点和链路）。可以使用 <Bundle> 元项把这些对象连同扩展一起捆绑。在一个实施例中，如果在 <Bundle> 内签名对象或扩展，那么 <Bundle> 包含如在此的其它地方所描述的 <Signature> 元项。

[1429] 章鱼 XML 模式 (Octopus.xsd)：

[1430]

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://intertrust.com/Octopus/1.0"
xmlns="http://intertrust.com/Octopus/1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <!—导入 -->
  <xs:import namespace="http://www.w3.org/2000/09/xmldsig#"
schemaLocation="xmldsig-core-schema.xsd"/>
  <xs:import namespace="http://www.w3.org/2001/04/xmlenc#"
```

[1431]

```
schemaLocation="xenc-schema.xsd"/>
  <!—顶层级元项-->
  <xs:element name="RootLevelObject" type="RootLevelObjectType" abstract="true"/>
  <xs:element name="OctopusObject" type="OctopusObjectType" abstract="true"/>
  <!—基本元项-->
  <xs:element name="Bundle" type="BundleType"/>
  <xs:element name="Link" type="LinkType" substitutionGroup="RootLevelObject"/>
  <xs:element name="Node" type="NodeType" substitutionGroup="RootLevelObject"/>
  <xs:element name="Control" type="ControlType" substitutionGroup="RootLevelObject"/>
  <xs:element name="Controller" type="ControllerType" substitutionGroup="RootLevelObject"/>
  <xs:element name="Protector" type="ProtectorType" substitutionGroup="RootLevelObject"/>
  <xs:element name="ContentKey" type="ContentKeyType"
substitutionGroup="RootLevelObject"/>
  <!—密钥元项 -->
  <xs:element name="SecretKey" type="KeyType"/>
  <xs:element name="PublicKey" type="PairedKeyType"/>
  <xs:element name="PrivateKey" type="PairedKeyType"/>
  <xs:element name="KeyData" type="KeyDataType"/>
  <!—其它元项-->
  <xs:element name="AttributeList" type="AttributeListType"/>
  <xs:element name="Attribute" type="AttributeType"/>
  <xs:element name="ExtensionList" type="ExtensionListType"/>
  <xs:element name="Extension" type="ExtensionType" substitutionGroup="RootLevelObject"/>
  <xs:element name="LinkFrom" type="OctopusObjectReferenceType"/>
  <xs:element name="LinkTo" type="OctopusObjectReferenceType"/>
  <xs:element name="Id" type="xs:string"/>
  <xs:element name="Digest" type="DigestType"/>
  <xs:element name="ControlProgram" type="ControlProgramType"/>
  <xs:element name="CodeModule" type="CodeModuleType"/>
  <xs:element name="ControlReference" type="OctopusObjectReferenceType"/>
  <xs:element name="ContentKeyReference" type="OctopusObjectReferenceType"/>
  <xs:element name="ContentReference" type="OctopusObjectReferenceType"/>
```

[1432]

```
<xs:element name="ProtectedTargets" type="ProtectedTargetsType"/>
<xs:element name="ControlledTargets" type="ControlledTargetsType"/>
<!—水肺 -->
<xs:element name="ScubaKeys" type="ScubaKeysType"/>
<!—章鱼对象的基本类型-->
<xs:complexType name="RootLevelObjectType"/>
<xs:complexType name="OctopusObjectType">
  <xs:complexContent>
    <xs:extension basc="RootLevclObjcctType">
      <xs:sequence>
        <xs:element ref="AttributeList" minOccurs="0"/>
        <xs:element ref="ExtensionList" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:string" use="optional"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="AnyContainerType">
  <xs:complexContent>
    <xs:extension base="RootLevelObjectType">
      <xs:sequence>
        <xs:any processContents="lax"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="ExtensionType">
  <xs:complexContent>
    <xs:extension base="AnyContainerType">
      <xs:sequence minOccurs="0">
        <xs:element ref="Digest" minOccurs="0"/>
      </xs:sequence>
```

[1433]

```
        <xs:attribute name="id" type="xs:string" use="required"/>
        <xs:attribute name="subject" type="xs:string"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="ExtensionListType">
    <xs:sequence>
      <xs:element ref="Extension" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="AttributeListType">
    <xs:sequence>
      <xs:element ref="Attribute" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="AttributeType">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="name" type="xs:string" use="required"/>
        <xs:attribute name="type" type="xs:string" default="string"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <xs:complexType name="DigestType">
    <xs:sequence>
      <xs:element ref="ds:DigestMethod"/>
      <xs:element ref="ds:DigestValue"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="OctopusObjectReferenceType">
    <xs:sequence>
      <xs:element ref="Id"/>
```

[1434]

```
        <xs:element ref="Digest" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ProtectedTargetsType">
      <xs:sequence>
        <xs:element ref="ContentReference" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ControlledTargetsType">
      <xs:sequence>
        <xs:element ref="ContentKeyReference" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
    <!—捆束类型-->
    <xs:complexType name="BundleType">
      <xs:sequence>
        <xs:element ref="RootLevelObject" maxOccurs="unbounded"/>
        <xs:element ref="ds:Signature" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
    <!—节点类型-->
    <xs:complexType name="NodeType">
      <xs:complexContent>
        <xs:extension base="OctopusObjectType"/>
      </xs:complexContent>
    </xs:complexType>
    <!—链路类型 -->
    <xs:complexType name="LinkType">
      <xs:complexContent>
        <xs:extension base="OctopusObjectType">
          <xs:sequence>
            <xs:element ref="LinkFrom"/>
```

[1435]

```
            <xs:element ref="LinkTo"/>
            <xs:element ref="Control" minOccurs="0"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
    <!—保护器类型 -->
    <xs:complexType name="ProtectorType">
      <xs:complexContent>
        <xs:extension base="OctopusObjectType">
          <xs:sequence>
            <xs:element ref="ContentKeyReference"/>
            <xs:element ref="ProtectedTargets"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
    <!—控制类型 -->
    <xs:complexType name="CodeModuleType">
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute name="byteCodeType" use="required"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
    <xs:complexType name="ControlProgramType">
      <xs:sequence>
        <xs:element ref="CodeModule"/>
      </xs:sequence>
      <xs:attribute name="type" use="required"/>
    </xs:complexType>
    <xs:complexType name="ControlType">
```

[1436]

```
    <xs:complexContent>
      <xs:extension base="OctopusObjectType">
        <xs:sequence>
          <xs:element ref="ControlProgram"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <!—控制器类型-->
  <xs:complexType name="ControllerType">
    <xs:complexContent>
      <xs:extension base="OctopusObjectType">
        <xs:sequence>
          <xs:element ref="ControlReference"/>
          <xs:element ref="ControlledTargets"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <!—密钥类型 -->
  <xs:complexType name="KeyType">
    <xs:sequence>
      <xs:element ref="KeyData"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:string" use="required"/>
    <xs:attribute name="usage" type="xs:string" use="optional"/>
  </xs:complexType>
  <xs:complexType name="PairedKeyType">
    <xs:complexContent>
      <xs:extension base="KeyType">
        <xs:attribute name="pair" type="xs:string" use="required"/>
      </xs:extension>
```

[1437]

```
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="KeyDataType" mixed="true">
    <xs:sequence>
      <xs:element ref="xenc:EncryptedData" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="encoding" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="xmlenc"/>
          <xs:enumeration value="base64"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="format" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="PKCS#8"/>
          <xs:enumeration value="X.509"/>
          <xs:enumeration value="RAW"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
  <!—内容密钥类型 -->
  <xs:complexType name="ContentKeyType">
    <xs:complexContent>
      <xs:extension base="OctopusObjectType">
        <xs:sequence>
          <xs:element ref="SecretKey"/>
        </xs:sequence>
      </xs:extension>
```

[1438]

```
    </xs:complexContent>
  </xs:complexType>
  <!—水肺扩展 -->
  <xs:complexType name="ScubaKeysType">
    <xs:sequence>
      <xs:element ref="SecretKey" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="PublicKey" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="PrivateKey" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

[1439] 说明性的应用专用模式：

[1440]

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://intertrust.com/kformat/1.0"
xmlns="http://intertrust.com/kformat/1.0" xmlns:oct="http://intertrust.com/Octopus/1.0"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" elementFormDefault="qualified"
attributeFormDefault="unqualificd">
  <!—导入 -->
  <xs:import namespace="http://intertrust.com/Octopus/1.0" schemaLocation="Octopus.xsd"/>
  <!—元项 -->
  <xs:element name="Torpedo" type="TorpedoType"/>
  <xs:element name="BroadcastKey" type="BroadcastKeyType"/>
  <xs:element name="BroadcastKeyMethod" type="BroadcastKeyMethodType"/>
  <!—类型 -->
  <xs:complexType name="TorpedoType">
    <xs:sequence>
      <xs:element ref="BroadcastKey"/>
    </xs:sequence>
```

[1441]

```
    </xs:complexType>
    <xs:complexType name="BroadcastKeyType">
      <xs:sequence>
        <xs:element ref="BroadcastKeyMethod"/>
        <xs:element ref="oct:KeyData"/>
      </xs:sequence>
      <!-- id是MNK的名称 -->
      <xs:attribute name="id" type="xs:string"/>
      <!—源是MKT的名称 -->
      <xs:attribute name="source" type="xs:string"/>
    </xs:complexType>
    <xs:complexType name="BroadcastKeyMethodType">
      <xs:attribute name="Algorithm" fixed="http://marlin-drm.com/mangrove/1.0"/>
    </xs:complexType>
  </xs:schema>
```

[1442] B.1. 附加约束

[1443] B.1.1. 节点

[1444] 在一个实施例中，定义以下类型的节点：

[1445] ■章鱼个性节点，它们是给定 DRM 引擎的根节点（例如，设备节点或 PC 软件节点）。

[1446] ■其它类型的节点，诸如用户节点或用户组的节点，诸如预订节点或成员资格节点。

[1447] 在一个实施例中，节点包含密钥（例如，在诸如 ScubaKeys 之类的扩展中）并且必须能够分离所述节点的公共信息（例如，id、属性和公钥）及其私有扩展（例如其携带有秘密密钥和私钥）。此外，每个部分（公共和私有）存在一个签名，使得具有其签名的公共节点可以被原样导出（例如作为对许可服务的请求的参数）。

[1448] 在一个实施例中，在 ExternalExtension 中携带私有扩展并进行签名。公共节点及其私有扩展可以被封装在相同的 <bundle> 元项中或者可以独立地到达。下面在附录 B 的附件 A 中给出了所签名的章鱼个性节点的例子。

[1449] B.1.1.1 属性

[1450] 在一个实施例中，节点对象的每个 XML 编码携带具有以下 <Attribute> 的 <AttributeList>：

[1451] 对于章鱼个性来说：

[1452]

<AttributeList xmlns="http://intertrust.com/Octopus/1.0">

<Attribute name="urn:x-marlin.intertrust.com:type">...</Attribute>

<Attribute name="urn:x-marlin.intertrust.com:dnk_id">...</Attribute>

<Attribute name="urn:x-marlin.intertrust.com:manufacturer">...</Attribute>

<Attribute name="urn:x-marlin.intertrust.com:model">...</Attribute>

<Attribute name="urn:x-marlin.intertrust.com:version">...</Attribute>

</AttributeList>

[1453] 对于其它类型的节点来说：

[1454]

<AttributeList xmlns="http://intertrust.com/Octopus/1.0">

<Attribute name="urn:x-marlin.intertrust.com:type">...</Attribute>

</AttributeList>

[1455] B.1.1.2 扩展

[1456] 如附录 B 的附件 A 所示，在一个实施例中，章鱼个性节点携带用于 ScubaKeys（共享和机密密钥）和 Torpedo（广播秘密密钥）的扩展。其它类型的节点只携带 Scuba 共享密钥。

[1457] 所有公钥被携带在 <ExtensionList> 的 <Extension> 元项中的 <Node> 元项内。其它密钥被携带在 <Node> 元项之外的独立 <Extension> 元项中。

[1458] 在一个实施例中，在 <Node> 中签名 <ScubaKeys> 扩展。在此实施例中，在 <Node> 内携带 <ScubaKeys> 的内部 <Extension>（公钥）需要包括 <ds :DigestMethod> 元项以及 <ds :DigestValue> 元项。在外部 <Extension> 中携带的私钥需要被签名并且这通过签名整个扩展来进行。同样，<Torpedo> 扩展也被签名。

[1459] B.1.2 链路

[1460] 在一个实施例中，<Link> 元项的 <LinkTo> 和 <LinkFrom> 元项只包含 <Id> 元项并且不包含 <Digest> 元项。<Control> 元项是可选的。附录 B 的附件 C 包含所签名的链路对象的例子。

[1461] B.1.1.1 属性

[1462] 在一个实施例中，链路没有强制属性。这意味着 <AttributeList> 不被要求并且将被适应的实现方式忽略。

[1463] B.1.1.2 扩展

[1464] 在此附录 B 所示出的示例性实施例中，链路具有在 <Link> 内所携带的 <ScubaKeys> 内部扩展，从而 <ExtensionList> 元项是强制性的。另外，链路中的 <ScubaKeys> 扩展未被签名，从而在 <Extension> 元项内没有携带 <ds :DigestMethod> 和 <ds :DigestValue> 元项。此 <ScubaKeys> 扩展包含利用“来自节点”的公共或秘密 Scuba 共享密钥加密的“去往节点”的公共 / 秘密 Scuba 共享密钥（在 <PrivateKey> 和 <SecretKey> 元项中）的版本。使用 XML 加密语法来发信号表示此加密。在此附录 B 所图示的实施例中，<KeyData> 元项、<PrivateKey> 和 <SecretKey> 元项的孩子的“编码”属性被设置为“xmlenc”。此 <KeyData> 元项的孩子是 <xenc :EncryptedData> 元项。在

<KeyInfo>/<KeyName> 元项中通告加密密钥的名称。

[1465] 在一个实施例中，如果加密密钥是公钥，那么：

[1466] ■ <KeyName> 元项是密钥所属对的名称。

[1467] ■如果加密数据（例如，私钥）太大而不能直接利用公钥加密，那么产生中间的 128 位秘密密钥。然后例如使用 aes-128-cbc 利用此中间密钥来加密数据，并且（使用 <EncryptedKey> 元项）利用公钥来加密中间密钥。

[1468] 于是 XML 组块看起来好像：

[1469]

```
<!—E(I, data) -->
<EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#">
  <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
  <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
    <!-- E(PUBa, I) -->
    <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
      <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
      <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
```

[1470]

<KeyName>urn:x-octopus.intertrust.com:key-pair:300a</KeyName>
</KeyInfo>
<CipherData>
<CipherValue>
fFeGD4KAPEmESz/jW6CkbRegpM5kyH0Oy/o/uDQ78PaShtvUMoozeO4a0b785YnB
13Qa1ZUEYqR9V5TCUaOcH7wxxvBEIsd1nYKkVOgW/kFnRr98UDFvU90PRqaEP/SABb+Ju
AUmvxYX47qOVQqBQGGqzFssBDKmUk+s98dkPR8=
</CipherValue>
</CipherData>
</EncryptedKey>
</KeyInfo>
<CipherData>
<CipherValue>
c8LBj4BLzGOYv/GT3Y4w2XcwTYbr8fHNJhCOQjULuvoha/QYvZKKCpUY+nuCXC/st9TU+
8tMtaMt1GUpkCZQhSaTNcluCSxOyBoA6Xh/bmyZLDJ78+aJ/slTmfNpJGdb
vTaI7x9DD1Mp1mvFEjpAUjTTvruN32g4bxsF7FD8C1RWNAc4hS96nFDgrmzoO5pR
dda6mswFKG5B0kY7mYbhacb1owXkAk1Wc/OuXA+QLHdUthxeajoXNPfAGRz9FM3bpuJxb
xDAaaAJDxoReiTtS1nGaHhqa1hvLCpKk1zHBowHyvTvDLElLjHYEPeG6xSHBbzpT298tdK
UhXfaY6vvdceMdVXuBVL3eZP1jkJHDxeaBy1ce8xlQKZpo6Pjuxlb
bn5KUMt/PxWp7rLa5s786S740cwuN63+ZRgienxPK1CnYO3htMJ7hh/agvO9IyUD
RvcgnSEY9KA5Exy/6gIS/gouIjFU8r7056XcE4/IBodTWDkfyli/y8q5QA/0VaD9
Y3oER1p3pYuHwn/IeXM4gsBD31cgd7nvfK7lKYkZjowR9P6pSy57a+K4LZKDmfUH
zG/gZs2XcoPb9o6mVAEEej7+aLwqmoileykkR+0pkFntvqvXYRPkphhcVdzjzlMV
scpXBXfWx7wbQURXkiew7R4RihQy3wcv+ZFJpl9NsAE1yqyWy4rBobzZ7cTNMtfR
znhVlt+Wwq5G0IBxzU9WIFzFd/Rn2H9L4TI71LCa4VR3uNpf+XM8lp9LjLPRUnNh
28KrMdAddceyopYyiIF5p8idfh0//a/LKdE7JAk0q9ewk19ryqfl6CFeKI5oOMjh
kzNx3BR/iHxm31HIe3ZKtA==</CipherValue>
</CipherData>
</EncryptedData>

[1471] B.1.3 许可对象

[1472] 此附录 B 的附件 C 提供了所签名许可的例子（在已经出现第一撤销之前，下面参见内容密钥节）。

[1473] B.1.3.1 保护器

[1474] 在此附录B所示出的示例性实施例中，<ContentKeyReference>元项和<ContentReference>元项（例如在<ProtectedTargets>元项内）只包含<Id>元项并且不包含<Digest>元项。在此说明性实施例中，保护器对象不包含强制的属性或扩展；<AttributeList>和<ExtensionList>元项是可选的并且被忽略。

[1475] B.1.3.2 内容密钥

[1476] 在此附录B所示出的示例性实施例中，内容密钥对象包含非强制的属性或扩展。因此，<AttributeList>和<ExtensionList>元项是可选的并且将被忽略。

[1477] 在一个实施例中，<ContentKey>元项包含<SecretKey>元项，用于表示用来解密内容的实际密钥。加密与<SecretKey>相关联的<KeyData>。在一个实施例中，<KeyData>的“编码”属性被设置为“xmlenc”是强制性的。

[1478] 在一个实施例中，内容密钥对象存在两种不同的情况：(1) 在设备或PC应用的首次撤销之前：在这种情况下，由<SecretKey>元项所表示的内容密钥Kc只被实体的Scuba密钥（公共的或秘密的）加密，其中把内容绑定到所述实体（例如用户）。(2) 在首次撤销之后，其中依照Mangrove广播加密方案来加密内容密钥。然后利用内容被绑定到的实体的Scuba密钥（公共的或秘密的）加密。在这种情况下，我们具有超级加密。

[1479] 在此的其它地方描述了用于在超级加密的情况下加密<EncryptedData>元项的说明性方法。下面解释了怎样将其应用于情况b。

[1480] 在一个实施例中，用于利用Mangrove广播加密方案来加密内容密钥Kc的xmlenc语法是：

[1481]

```
<EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#">
  <EncryptionMethod Algorithm="see (*)"/>
  <KeyInfo xmlns="'http://www.w3.org/2000/09/xmldsig#">
    <KeyName>see (**)</KeyName>
  </KeyInfo>
  <CipherData>
    <CipherValue>see (***)</CipherValue>
  </CipherData>
</EncryptedData>
```

[1482] ■ (*) 是用于标识Mangrove广播加密方案的URL，在一个实施例中所述Mangrove广播加密方案还是在应用专用的xml模式调用“kformat.xsd”中<Torpedo>扩展的<BroadcastKeyMethod>算法。

[1483] ■ (**) 是Mangrove密钥树的名称。在一个实施例中，此值必须与在kformat.xsd中所定义的<BroadcastKey>元项的源属性相同。

[1484] ■ (***) 是用于表示依照Mangrove广播密钥算法加密内容密钥Kc的ASN.1序列的base64编码值：

[1485]

```
SEQUENCE {
    tags BIT STRING
    keys OCTET STRING
}
```

[1486] 在一个实施例中，利用许可被绑定到的实体的scuba共享密钥（公共的或秘密的）来加密上面所提及的<EncryptedData>的字节序列。如果使用公钥，如果<EncryptedData>的字节序列对于RSA 1024公钥来说太大，那么与如在下面所描述应用相同的约定（例如，参见利用公钥加密）并且需要中间密钥。可以在此附录B的附件D中找到这种内容密钥对象的XML编码的例子。

[1487] B.1.3.3控制器

[1488] 在一个实施例中，控制器对象包含非强制的属性或扩展。因此，<AttributeList>和<ExtensionList>元项是可选的并且将被适应的实现方式忽略。

[1489] 在一个实施例中，<DigestMethod>元项的算法属性值始终为http://www.w3.org/2000/09/xmldsig#sha1。

[1490] 在一个实施例中，<ControlReference>必须具有<Digest>元项。<DigestValue>元项必须包含引用控制的摘要的base64编码。

[1491] 在一个实施例中，如果控制器上的签名是PKI签名(rsa-sha1)，那么(<ControlledTargets>元项内的)<ContentKeyRefence>元项需要包括<Digest>元项并且<DigestValue>元项必须包含被嵌入到内容密钥对象中的明文内容密钥的摘要。

[1492] B.1.3.4控制

[1493] 在一个实施例中，控制对象包含非强制的属性或扩展。因此，<AttributeList>和<ExtensionList>元项是可选的并且将被适应的实现方式忽略。

[1494] 在一个实施例中，<ControlProgram>元项的类型属性被设置为“plankton”，并且<CodeModule>元项的byteCodeType属性被设置为“Plankton-1-0”。

[1495] 附录B——附件A：所签名章鱼个性节点的例子

[1496]

```
<Bundle xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" xmlns="http://intertrust.com/Octopus/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://intertrust.com/kformat/1.0
C:¥DOCUME~1¥julien¥Desktop¥kformat¥kformat.xsd">
  <!—首先，具有公开信息的节点-->
  <Node id="urn:kformat:device:0001">
    <AttributeList>
      <Attribute name="urn:x-marlin.intertrust.com:type">device</Attribute>
      <Attribute
name="urn:x-marlin.intertrust.com:dnk_id">urn:kformat:mangrove:0001</Attribute>
      <Attribute name="urn:x-marlin.intertrust.com:manufacturer_id">SONY</Attribute>
      <Attribute name="urn:x-marlin.intertrust.com:model">urn:sony:walkman</Attribute>
      <Attribute
name="urn:x-marlin.intertrust.com:version">urn:sony:walkman:002a</Attribute>
    </AttributeList>
    <ExtensionList>
      <Extension id="urn:kformat:device:0001:scuba:public">
        <ScubaKeys>
          <PublicKey id="urn:kformat:device:0001:scuba:public:sharing"
                     pair="urn:kformat:device:0001:scuba:pair:sharing">
            <KeyData encoding="base64" format="X509">MIIC...MEbB</KeyData>
          </PublicKey>
```

[1497]

```
            <PublicKey id="urn:kformat:device:0001:scuba:public:confidentiality"
                       usage="confidentiality"
                       pair="urn:kformat:device:0001:scuba:pair:confidentiality">
              <KeyData encoding="base64" format="X.509">MIIChDCC... vh8BM52</KeyData>
            </PublicKey>
          </ScubaKeys>
          <Digest>
            <DigestMethod xmlns="http://www.w3.org/2000/09/xmldsig#"
                          Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <DigestValue
xmlns="http://www.w3.org/2000/09/xmldsig#">OGZGBY8OpQXs</DigestValue>
          </Digest>
        </Extension>
      </ExtensionList>
    </Node>
    <!—然后，私有Scuba扩展 -->
    <Extension id="urn:kformat:device:0001:scuba:private" subject="urn:kformat:device:0001">
      <ScubaKeys>
        <PrivateKey id="urn:kformat:device:0001:scuba:private:sharing"
                    pair="urn:kformat:device:0001:scuba:pair:sharing">
          <KeyData encoding="base64" format="PKCS8">MIICdgIBADAN...
DXywQLg==</KeyData>
        </PrivateKey>
        <PrivateKey id="urn:kformat:device:0001:scuba:private:confidentiality"
                    usage="confidentiality"
                    pair="urn:kformat:device:0001:scuba:pair:confidentiality">
          <KeyData encoding="base64" format="PKCS8">MIICdwIBADAN...
q4olog34=</KeyData>
        </PrivateKey>
        <SecretKey id="urn:kformat:device:0001:scuba:secret:sharing">
          <KeyData encoding="base64" format="RAW">Z1n2/2cbz1oO/fZo9xtmyA==</KeyData>
        </SecretKey>
```

[1498]

```
        <SecretKey id="urn:kformat:device:0001:scuba:secret:confidentiality"
                    usage="confidentiality">
          <KeyData encoding="base64"
format="RAW">0CJ8bcORW6GLX4GzT7XKvg==</KeyData>
        </SecretKey>
      </ScubaKeys>
    </Extension>
    <!—然后，私有Torpedo扩展 -->
    <Extension id="urn:kformat:device:0001:torpedo" subject="urn:kformat:device:0001">
      <Torpedo xmlns="http://intertrust.com/kformat/1.0">
        <BroadcastKey id="urn:kformat:mangrove:0001">
          <BroadcastKeyMethod Algorithm="http://marlin-drm.com/mangrove/1.0"/>
          <KeyData xmlns="http://intertrust.com/Octopus/1.0" encoding="base64"
format="RAW">....</KeyData>
        </BroadcastKey>
      </Torpedo>
    </Extension>
    <!—然后，在公开部分上的签名-->
    <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
      <SignedInfo>
        <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
        <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
        <Reference URI="urn:kformat:device:0001">
          <Transforms>
            <Transform
Algorithm="http://www.octopus-drm.com/2004/07/format-independent-cano#"/>
          </Transforms>
          <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
          <DigestValue>gI5QoD7MUAgjcpkPiciZhoSHbEQ=</DigestValue>
        </Reference>
      </SignedInfo>
      <SignatureValue>gI5QoD7MUAgjcpkPiciZhoSHbEQ=</SignatureValue>
```

[1499]

```
      <KeyInfo>
        <X509Data>
          <!—把签名密钥的公钥放在此-->
          <X509Certificate>...</X509Certificate>
          <!—如果需要，没有根的证书链-->
          <X509Certificate>...</X509Certificate>
        </X509Data>
      </KeyInfo>
    </Signature>
    <!—然后，在私有部分上签名 -->
    <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
      <SignedInfo>
        <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
        <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
        <Reference URI="urn:kformat:0001:scuba:private">
          <Transforms>
            <Transform
Algorithm="http://www.octopus-drm.com/2004/07/format-independent-cano#"/>
          </Transforms>
          <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
          <DigestValue>gI5QoD7MUAgjcpkPiciZhoSHbEQ=</DigestValue>
        </Reference>
        <Reference URI="urn:kformat:device:0001:torpedo">
          <Transforms>
            <Transform
Algorithm="http://www.octopus-drm.com/2004/07/format-independent-cano#"/>
          </Transforms>
          <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
          <ds:DigestValue>97mDfnw0vF/ECQHcvDk</ds:DigestValue>
        </Reference>
      </SignedInfo>
      <SignatureValue>gI5QoD7MUAgjcpkPiciZhoSHbEQ=</SignatureValue>
```

[1500]

```
    <KeyInfo>
      <X509Data>
        <!—把签名密钥的公钥证书放在这里-->
        <X509Certificate>...</X509Certificate>
        <!-- 如果需要，没有根的证书链 -->
        <X509Certificate>...</X509Certificate>
      </X509Data>
    </KeyInfo>
  </Signature>
  </Bundle>
```

[1501] 附录 B——附件 B：所签名章鱼链路的例子

[1502]

```
<?xml version="1.0" encoding="UTF-8"?>
<!--XMLSPY v2004 rel. 2 U生成的样本XML文件(http://www.xmlspy.com)-->
<Bundle xmlns="http://intertrust.com/Octopus/1.0"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://intertrust.com/Octopus/1.0
C:¥ws¥Octopus¥Source¥Xml¥Schemas¥Octopus.xsd">
  <Link id="urn:kformat:link:device:0001:to:user:1234">
    <ExtensionList>
      <Extension id="urn:kformat:link:device:0001:to:user:1234:scuba">
        <ScubaKeys>
          <!-- E(PUBdevice, PRIVuser) -->
          <PrivateKey id="urn:kformat:user:1234:scuba:private:sharing"
                      pair="urn:kformat:user:1234:scuba:pair:sharing">
            <KeyData encoding="xmlenc" format="PKCS8">
              <!-- E(I, PRIVuser) I: 中间密钥-->
              <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#">
                <EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
                <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
```

[1503]

```
                    <!-- E(PUBdevice, I) -->
                    <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
                      <EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
                      <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
                        <KeyName>urn:kformat:device:0001:scuba:pair:sharing</KeyName>
                      </KeyInfo>
                      <CipherData>
                        <CipherValue>fFeGD4K... s98dkPR8=</CipherValue>
                      </CipherData>
                    </EncryptedKey>
                  </KeyInfo>
                  <CipherData>
                    <CipherValue>
c8LBj4BLzGOYv...HIe3ZKtA==</CipherValue>
                  </CipherData>
                </EncryptedData>
              </KeyData>
            </PrivateKey>
            <!-- E(PUBdevice, Suser) -->
            <SecretKey id="urn:kformat:user:1234:secret:sharing">
              <KeyData encoding="xmlenc" format="RAW">
                <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#">
                  <EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
                  <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
                    <KeyName>urn:kformat:device:0001:scuba:pair:sharing</KeyName>
                  </KeyInfo>
                  <CipherData>
                    <CipherValue>OHVaH... kjLA=</CipherValue>
                  </CipherData>
                </EncryptedData>
```

[1504]

```
            </KeyData>
          </SecretKey>
        </ScubaKeys>
      </Extension>
    </ExtensionList>
    <LinkFrom>
      <Id>urn:kformat:device:0001</Id>
    </LinkFrom>
    <LinkTo>
      <Id>urn:kformat:user:1234</Id>
    </LinkTo>
  </Link>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <Reference URI="urn:kformat:link:device:0001:to:user:1234">
        <Transforms>
          <Transform
Algorithm="http://www.octopus-drm.com/2004/07/format-independent-cano#"/>
        </Transforms>
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <DigestValue>gI5QoD7MUAgjcpkPiciZhoSHbEQ=</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>gI5QoD7MUAgjcpkPiciZhoSHbEQ=</SignatureValue>
    <KeyInfo>
      <X509Data>
        <!—把签名密钥的公钥证书放在此处 -->
        <X509Certificate>...</X509Certificate>
        <!—如果需要，没有根的证书链 -->
        <X509Certificate>...</X509Certificate>
```

[1505]

```
            </X509Data>
        </KeyInfo>
    </Signature>
    </Bundle>
```

[1506] 附录 B——附件 C：所签名章鱼许可的例子（在没有撤销的情况下）

[1507]

```
<Bundle xmlns="http://intertrust.com/Octopus/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://intertrust.com/Octopus/1.0
C:¥ws¥Octopus¥Source¥Xml¥Schemas¥Octopus.xsd">
  <ContentKey id="urn:x-octopus.intertrust.com:content-key:2002">
    <SecretKey id="urn:x-octopus.intertrust.com:secret-key:2002">
      <KeyData encoding="xmlenc" format="RAW">
        <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#">
          <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
          <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
            <KeyName>urn:x-octopus.intertrust.com:secret-key:303c</KeyName>
          </KeyInfo>
          <CipherData>
            <CipherValue>
MCR0LGaoyuO2o6zsIW9IrOOSMfhuZCtV20o94/OfQ5dHbIJ3q2vZrgwRbJepLvRa
            </CipherValue>
          </CipherData>
        </EncryptedData>
      </KeyData>
    </SecretKey>
  </ContentKey>
  <ContentKey id="urn:x-octopus.intertrust.com:content-key:2001">
    <SecretKey id="urn:x-octopus.intertrust.com:secret-key:2001">
      <KeyData encoding="xmlenc" format="RAW">
        <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#">
          <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
          <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
```

[1508]

<KeyName>urn:x-octopus.intertrust.com:key-pair:300c</KeyName>
</KeyInfo>
<CipherData>
<CipherValue>
LD51cJ71Bswwb2GttPoPjMytFn3oocI7vhZPA5mKY06R82KZjxFDtcCmbOlYZ5Hv
6ldqQ3hy74/mQF3AJ1jRXa9/ymmasVBxsJnv426B9/JkzTT4CGqNjS+WPOKL9NZC
qnRWguJmk8dQ+jaxW51SQSjp4MCpGZB63zfvcuBD7qE=
</CipherValue>
</CipherData>
</EncryptedData>
</KeyData>
</SecretKey>
</ContentKey>
<Control id="urn:x-octopus.intertrust.com:control:0001">
<ControlProgram type="Plankton">
<CodeModule byteCodeType="Plankton-1-0">
AAABUnBrQ00AAAA2cGtFWAAAAAIOR2xvYmFsLk9uTG9hZAAAAAAAEkFjdGlvbi5QbGF5L
kNoZWNrAAAAAFgAAACmcGtDUwEAAAAEGgEAAAAABQEAAAACIAMBAAAABBoBAAA
AHgUbAQAAACwYAQAAAAQaAQAAACIFAQAAAAIgAwEAAAAEGgEAAAA7BRsBAAAAB
hgBAAAAABUB/////xUBAAAABBoBAAAAPwUBAAAABBoBAAAAHgUaIAEAAAAgGAEAAA
AEGgEAAAA7BRogAQEOX3oLAQAAAAYYAQAAAAAVAf////8VAAAAAbnBrRFNPY3RvcHVzL
kxpbmtzLklzTm9kZVJlYWNoYWJsZQAAAAAAU3lzdGVtLkhvc3QuR2V0VGltZVN0YW1wAAA
AAAB1cm46eC1vY3RvcHVzLmludGVydHJ1c3QuY29tOm5vZGU6MDAwMwA=
</CodeModule>
</ControlProgram>
</Control>
<Protector>
<ContentKeyReference>
<Id>urn:x-octopus.intertrust.com:content-key:2002</Id>
</ContentKeyReference>
<ProtectedTargets>
<ContentReference>

[1509]

```
            <Id>urn:x-octopus.intertrust.com:content:2001</Id>
          </ContentReference>
          <ContentReference>
            <Id>urn:x-octopus.intertrust.com:content:2002</Id>
          </ContentReference>
        </ProtectedTargets>
      </Protector>
      <Protector>
        <ContentKeyReference>
          <Id>urn:x-octopus.intertrust.com:content-key:2001</Id>
        </ContentKeyReference>
        <ProtectedTargets>
          <ContentReference>
            <Id>urn:x-octopus.intertrust.com:content:2003</Id>
          </ContentReference>
          <ContentReference>
            <Id>urn:x-octopus.intertrust.com:content:2004</Id>
          </ContentReference>
        </ProtectedTargets>
      </Protector>
      <Controller id="urn:x-octopus.intertrust.com:controller:0001">
        <ControlReference>
          <Id>urn:x-octopus.intertrust.com:control:0001</Id>
          <Digest>
            <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"
xmlns="http://www.w3.org/2000/09/xmldsig#"/>
            <DigestValue
xmlns="http://www.w3.org/2000/09/xmldsig#">02ACF5674287FF45CFA5A66D70125FF5601A63
F7</DigestValue>
          </Digest>
        </ControlReference>
        <ControlledTargets>
```

[1510]

```
            <ContentKeyReference>
              <Id>urn:x-octopus.intertrust.com:content-key:2002</Id>
            </ContentKeyReference>
            <ContentKeyReference>
              <Id>urn:x-octopus.intertrust.com:content-key:2001</Id>
            </ContentKeyReference>
          </ControlledTargets>
        </Controller>
        <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
          <SignedInfo>
            <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
            <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
            <Reference URI="urn:x-octopus.intertrust.com:controller:0001">
              <Transforms>
                <Transform
    Algorithm="http://www.octopus-drm.com/2004/07/format-independent-cano#"/>
              </Transforms>
              <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
              <DigestValue>A42CZFK4DQvb/M0wqOLZRnyiS1Y=</DigestValue>
            </Reference>
          </SignedInfo>
          <SignatureValue>gI5QoD7MUAgjcpkPiciZhoSHbEQ=</SignatureValue>
          <KeyInfo>
            <KeyName>urn:x-octopus.intertrust.com:secret-key:2002;
    urn:x-octopus.intertrust.com:secret-key:2001</KeyName>
          </KeyInfo>
        </Signature>
      </Bundle>
```

[1511] 附录 B——附件 D：在撤销情况下内容密钥的例子

[1512]

```
<ContentKey id="urn:x-octopus.intertrust.com:content-key:2001">
  <SecretKey id="urn:x-octopus.intertrust.com:secret-key:2001">
    <KeyData encoding="xmlenc" format="RAW">
      <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#">
        <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
        <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
          <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
            <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
            <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
              <KeyName>urn:kformat:user:0001:scuba:pair:sharing</KeyName>
            </KeyInfo>
            <CipherData>
              <CipherValue>E(PUBuser, I)</CipherValue>
            </CipherData>
          </EncryptedKey>
        </KeyInfo>
        <CipherData>
          <CipherValue>
          Encryption of the EncryptedData element containing
          the encryption of Kc with the broadcast encryption
          scheme (see note on xmlenc and broadcast key encryption
          in the ContentKey section) with the intermediate key I.
          </CipherValue>
        </CipherData>
      </EncryptedData>
    </KeyData>
  </SecretKey>
</ContentKey>
```

[1513] 15. 附录 C

[1514] 此附录 C 示出了供上述引导协议使用的简单简档的例子。还提供了简单的规范串行、示例性 XML 编组以及用于章鱼引导 SOAP 网络服务的示例性 WSDL。

[1515] 简单简档

[1516] 在一个实施例中，使用由以下内容组成的简单简档：

[1517]

简档名称	SimpleProfile
公钥加密算法	http://www.w3.org/2001/04/xmlenc#rsa-1_5
公钥签名算法	http://www.w3.org/2000/09/xmldsig#rsa-sha1
秘密密钥加密算法	http://www.w3.org/2001/04/xmlenc#aes128-cbc
秘密密钥签名算法	http://www.w3.org/2000/09/xmldsig#hmac-sha1
摘要算法	http://www.w3.org/2000/09/xmldsig#sha1
证书格式	X.509 (版本 3)
消息编组	简单的 XML 编组 1.0
最小特殊时间大小	16 字节
规范的对象串行	简单的规范串行 1.0

[1518] 简单的规范串行 1.0

[1519] 在一个实施例中，在上述简单简档中使用的简单的规范字节序列包括从消息中对象的字段值来构造字节序列。每个消息和每个对象由一个或多个字段组成。每个字段是简单字段或复合字段。

[1520] 简单字段可以是以下四种类型之一：整数、字符串、字节序列或字段数组。复合字段由一个或多个子字段组成，每个子字段是简单的或复合的。

[1521] 在一个实施例中，用于构造每个字段类型的规范字节序列的规则如下：

[1522] 复合字段

[1523]

字段 0	字段 1	字段 2	…

[1524] 规范的字节序列是每个子字段的规范字节序列的拼接（可选字段不被跳过，但是依照可选字段的规则来串行化）。

[1525] 字段数组

[1526]

字段计数	字段 0	字段 1	…

[1527] 被依照大端次序编码为 4 字节序列的字段计数，后面是每个字段的规范字节序列。如果字段计数为 0，那么在 4 字节字段计数后面什么都没有（在这种情况下，所有 4 个字节都具有值 0）。

[1528] 整数

[1529]

I0	I1	I2	I3

[1530] 32 比特带符号的值，依照大端次序被编码为 4 字节的序列。

[1531] 字符串

[1532]

字节计数	字节 0	字节 1	…

[1533] 字符串由 UTF-8 编码的 8 比特字节序列来表示。所编码的字节序列的字节计数被依照大端次序编码为 4 字节序列。字节计数后面是 UTF-8 编码的字符串的字节序列。

[1534] 字节序列

[1535]

字节计数	字节 0	字节 1	…

[1536] 字节计数被依照大端次序编码为 4 字节序列（如果字节序列为空，或者已经省略相应的字段，那么字节计数为 0，并且在 4 字节的字节计数之后没有字节值）。每个字节被按原样编码。

[1537] 简单的 XML 编组 1.0

[1538] 模式 SimpleBootProtocol.xsd

[1539]

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
    <xs:element name="BootstrapRequestMessage">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="BootstrapRequest"/>
            </xs:sequence>
            <xs:attribute name="Protocol" type="xs:string" use="required"/>
            <xs:attribute name="Profile" type="xs:string" use="required"/>
            <xs:attribute name="Version" type="xs:decimal" use="required"/>
        </xs:complexType>
```

[1540]

```
</xs:element>
<xs:element name="BootstrapRequest">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="SessionId"/>
            <xs:element ref="TrustDomain" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="ChallengeRequestMessage">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="ChallengeRequest"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="ChallengeRequest">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="Challenge"/>
            <xs:element ref="Signature"/>
            <xs:element ref="CertificateChain"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="ChallengeResponseMessage">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="SessionKey"/>
            <xs:element ref="EncryptedChallengeResponse"/>
        </xs:sequence>
    </xs:complexType>
```

[1541]

```
</xs:element>
<xs:element name="EncryptedChallengeResponse" type="xs:base64Binary"/>
<xs:element name="ChallengeResponse">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="ClientInfo"/>
            <xs:element ref="Challenge"/>
            <xs:element ref="SessionKey"/>
            <xs:element ref="Signature"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="Challenge">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="Cookie"/>
            <xs:element ref="Nonce"/>
            <xs:element ref="SessionId"/>
            <xs:element ref="EncryptionKey" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="BootstrapResponseMessage">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="EncryptedBootstrapResponse"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="EncryptedBootstrapResponse" type="xs:base64Binary"/>
<xs:element name="BootstrapResponse">
    <xs:complexType>
```

[1542]

```
            <xs:sequence>
                <xs:element ref="SessionId"/>
                <xs:element ref="Data"/>
                <xs:element ref="Signature"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="ErrorResponseMessage">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="ErrorResponse"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="ErrorResponse" type="xs:string"/>
    <xs:element name="CertificateChain">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="Certificate" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="TrustDomain" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="Certificate" type="xs:base64Binary"/>
    <xs:element name="ClientInfo">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="Attribute"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="Attribute" type="xs:string"/>
```

[1543]

```
            <xs:element name="Cookie" type="xs:base64Binary"/>
            <xs:element name="Data" type="xs:base64Binary"/>
            <xs:element name="EncryptionKey" type="xs:base64Binary"/>
            <xs:element name="Nonce" type="xs:base64Binary"/>
            <xs:element name="SessionId" type="xs:string"/>
            <xs:element name="SessionKey" type="xs:base64Binary"/>
            <xs:element name="Signature" type="xs:base64Binary"/>
            <xs:element name="TrustDomain" type="xs:string"/>
        </xs:schema>
```

[1544] 例子：

[1545]

<BootstrapRequestMessage Protocol="OctopusSimpleBoot" Profile="SimpleProfile" Version="1.0">
<BootstrapRequest>
<SessionId>some-unique-session-id-0008</SessionId>
<TrustDomain>urn:x-octopus.intertrust.com:scuba:boot:trust-domain:test001</TrustDomain>
</BootstrapRequest>
</BootstrapRequestMessage>

<ChallengeRequestMessage>
<ChallengeRequest>
<Challenge>
<Cookie>c29tZS11bmlxdWUtc2Vzc2lvbi1pZC0wMDA4</Cookie>
<Nonce>Mv5VIv73cxo5b+gisQJP8Q==</Nonce>
<SessionId>some-unique-session-id-0008</SessionId>
<EncryptionKey>
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCpMY4wvgTJvVPTufNVbdIfTUwOi4FZPtzi3ez
etY9gx51O6dfRn+LKPq1nJsSXCR5ZIvRUyoNZC0Qc3SLobUhXD6uTsrV5xtRKOSxZTLt5DZ15AtddSrA
AfF9baDGMi5KQP9w7qB2Ci/MmYha4Jix1iUltv0zWIKmSpytgHC8i/QIDAQAB
</EncryptionKey>
</Challenge>
<Signature>
GsWP3yPT36r3e1jZfulUS7xp5w2ei7iTsAJ/YD13fX+pSJrpeKAtq2BTzHQ1AclOorPJwzWHDanc
cui9/rinlg3Drw52bQXLzhZbZLXadIGFP3YP1gTKPuazUCYCLAjYTJbdulWlnTKDtmf34/66H0sz
DCCyxQsdFZhSNk6pyQE=
</Signature>
<CertificateChain TrustDomain="urn:x-octopus.intertrust.com:scuba:boot:trust-domain:test001">
<Certificate>
MIID...<!—终端实体证书 -->
</Certificate>
MIID...<!—中间人证书 -->
<Certificate>
MIIE...<!-- 中间人证书 -->

[1546]

```
        </Certificate>
        <Certificate>
MIID...<!—字节链到信任锚的证书 -->
        </Certificate>
      </CertificateChain>
    </ChallengeRequest>
</ChallengeRequestMessage>

<ChallengeResponseMessage>
    <SessionKey>
PtzJcFT2s1sW7oRZ1a+HASdRmZer4pk4QArFZWYlkUWZcIZTN2g2YeCQwORq2J9QXOksU6utKmOmg
fEHY151UdcMFake3CwquvVN6w/7mFH0qtDoc+GhuKe9eQXN2RHa3SlhfR5ShF2A/cwZHd4Nknt4w8M
WMDDn3SUDd6aS/ZI=
    </SessionKey>
    <EncryptedChallengeResponse>
mQCkPL560D00o...
    </EncryptedChallengeResponse>
</ChallengeResponseMessage>

<ChallengeResponse>
    <ClientInfo>
      <Attribute Name="SomeAttribute">Bla Bla</Attribute>
    </ClientInfo>
    <Challenge>
      <Cookie>c29tZS11bmlxdWUtc2Vzc2lvbi1pZC0wMDA4</Cookie>
      <Nonce>Mv5VIv73cxo5b+gisQJP8Q==</Nonce>
      <SessionId>some-unique-session-id-0008</SessionId>
    </Challenge>
    <SessionKey>bbBG8JsGaApFdNJq6hFrIQ==</SessionKey>
    <Signature>WYMULPpF4lOJ6MiAxd1lueN7p/4=</Signature>
</ChallengeResponse>
```

[1547]

```
<BootstrapResponseMessage>
  <EncryptedBootstrapResponse>
chXTp20+yI7/i1pHLawFOLXdGb...
  </EncryptedBootstrapResponse>
</BootstrapResponseMessage>

<BootstrapResponse>
  <SessionId>some-unique-session-id-0008</SessionId>
  <Data>
PD94bWwgdmVyc...
  </Data>
  <Signature>
XqCeVRb4YaYAK9Ilj60B5R1hQ03tFpHPw3wMMATbeUfqCpEXfAB7u2/qnjs9jLgWTOOvLDE5C5aVV
MvzlnRnDv0GHL1s6g43HusVx7fpazwHoFrb3M3eKwXMoYsI6xpdYy2BX1bs5QT2xdwBv2ClBjo7
KzQfmb/3bYEO+xGdg48=
  </Signature>
</BootstrapResponse>

<ErrorResponseMessage>
  <ErrorResponse Code="6">Some Error Info</ErrorResponse>
</ErrorResponseMessage>
```

用于引导 SOAP 网络服务的 WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<!--此wsdl文件描述无状态多回合的引导协议的接口
  该协议这样工作:
  1. C->S: BootstrapRequestMessage
  2. S->C: ChallengeRequestMessage
  3. C->S: ChallengeResponseMessage
  4. S->C: BootstrapResponseMessage
-->
```

[1548]

```
<wsdl:definitions name="OctopusBootstrap"
targetNamespace="http://www.intertrust.com/services/OctopusBootstrap"
xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://www.intertrust.com/services/OctopusBootstrap"
xmlns:intf="http://www.intertrust.com/services/OctopusBootstrap"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tnstype="http://www.intertrust.com/services/OctopusBootstrap"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ob="http://www.intertrust.com/Octopus/Bootstrap/1.0"
xmlns:nc="http://www.intertrust.com/core">
  <wsdl:types>
    <schema targetNamespace="http://www.intertrust.com/services/OctopusBootstrap"
xmlns="http://www.w3.org/2001/XMLSchema">
      <!—导入-->
      <import namespace="http://www.intertrust.com/Octopus/Bootstrap/1.0"
schemaLocation="./SimpleBootProtocol.xsd"/>
      <!—元项-->
      <element name="requestdata">
        <complexType>
          <!—这是多回合无状态协议(归因于点心文件):
              客户端能够发送BootstrapRequestMessage或者
              ChallengeReponseMessage -->
          <choice>
            <element ref="ob:BootstrapRequestMessage"/>
            <element ref="ob:ChallengeResponseMessage"/>
          </choice>
        </complexType>
      </element>
      <element name="responsedata">
        <complexType>
          <!-- 这是多回合无状态协议(归因于点心文件):
```

[1549]

```
                the server can send back a ChallengeRequestMessage or
                BootstrapResponseMessage or an ErrorResponseMessage -->
              <choice>
                <element ref="ob:ChallengeRequestMessage"/>
                <element ref="ob:BootstrapResponseMessage"/>
                <element ref="ob:ErrorResponseMessage"/>
              </choice>
            </complexType>
          </element>
        </schema>
      </wsdl:types>
      <!—消息声明-->
      <wsdl:message name="invokeRequest">
        <wsdl:part element="tnstype:requestdata" name="invokeRequest"/>
      </wsdl:message>
      <wsdl:message name="invokeResponse">
        <wsdl:part element="tnstype:responsedata" name="invokeResponse"/>
      </wsdl:message>
      <!—端口类型声明 -->
      <wsdl:portType name="OctopusBootstrap">
        <wsdl:operation name="invoke">
          <wsdl:input message="impl:invokeRequest" name="invokeRequest"/>
          <wsdl:output message="impl:invokeResponse" name="invokeResponse"/>
        </wsdl:operation>
      </wsdl:portType>
      <!—绑定声明-->
      <wsdl:binding name="OctopusBootstrapSoapBinding" type="impl:OctopusBootstrap">
        <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
        <wsdl:operation name="invoke">
          <wsdlsoap:operation soapAction=""/>
          <wsdl:input name="invokeRequest">
            <wsdlsoap:body encodingStyle=""
```

[1550]

```
namespace="http://www.intertrust.com/services/OctopusBootstrap" use="literal"/>
        </wsdl:input>
        <wsdl:output name="invokeResponse">
          <wsdlsoap:body encodingStyle=""
namespace="http://www.intertrust.com/services/OctopusBootstrap" use="literal"/>
        </wsdl:output>
      </wsdl:operation>
    </wsdl:binding>
    <!—服务声明-->
    <wsdl:service name="OctopusBootstrapService">
      <wsdl:port binding="impl:OctopusBootstrapSoapBinding" name="OctopusBootstrap">
        <wsdlsoap:address location="http://localhost:8080/OctopusBootstrap/services/OctopusBootstrap"/>
      </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

[1551] 16. 附录 D

[1552] 下面给出了用于计算对象的规范字节序列 (CBS) 的编码中性方式，并且在优选实施例中用于计算用来对对象进行数字签名的摘要。此字节序列与表示或发送对象的方式无关，从而使相同的摘要和签名值能够遍及系统使用，在所述系统中使用多种编码格式（例如，XML、ANS1）、编程语言等。

[1553] 规范的字节序列算法

[1554] 规范的字节序列算法包括根据字段值来构造字节序列。每个字段具有简单类型或复合类型的值。一些字段可以被指定为可选的（所述字段可以存在或被省略）。

[1555] 在一个实施例中，简单类型是：整数、字符串、字节和布尔。

[1556] 复合类型由一个或多个子字段组成；每个子字段具有简单或复合类型的值。复合类型是异构的或同构的，这意味着存在一个或多个不同类型（即，异构）的一个或多个子字段值（简单或复合），或者存在全部相同类型（同构）的一个或多个子字段值（简单或复合）。

[1557] 通过当字段始终存在时向字段值应用编码规则或者当所述字段被指定为可选时应用可选字段的编码规则来获得字段的规范字节序列。在下面的编码规则描述中，术语字节意指 8 比特值（八位字节）：

[1558] 1.1. 可选字段

[1559] 如果可选字段存在，那么其值被串行化为字节值 1 后面是字段值的规范字节序列。如果它被省略，那么其值被串行化为字节值 0。

[1560] 1.2. 异构的复合

[1561] 规范的字节序列是每个子字段值的规范字节序列的拼接（可选字段值不被跳过，

但是依照可选字段值的规则来串行化）。

[1562] 1.3. 同构的复合

[1563] 规范的字节序列是子字段计数，被依照大端次序编码为 4 字节的序列，后面是每个子字段值的规范字节序列的拼接。如果子字段计数为 0，那么在 4 字节字段计数后面什么都没有（在这种情况下，所有 4 个字节都具有值 0）。

[1564] 1.4. 整数

[1565] 32 比特整数的值，依照大端次序被编码为 4 字节的序列。

[1566] 1.5. 字符串

[1567]

字节计数	字节 0	字节 1	…

[1568] 字符串由 UTF-8 编码的字节序列（不是空结束的）表示。字符串的规范字节序列包括 (1) 字符串的字节计数，被依照大端次序编码为 4 字节的序列，后面是 (2) 字符串的字节序列。

[1569] 1.6. 字节

[1570] 8 比特值

[1571] 1.7. 布尔

[1572] 8 比特值 ：对于假来说为 0，并且对于真来说为 1

[1573] 2. 章鱼对象的应用

[1574] 在一个实施例中，章鱼对象的规范字节序列是其每个字段的规范字节序列依照它们在对象模型中所定义的次序的拼接。

[1575] 对于异构的复合类型来说，字段的次序是在类型定义中所指定的次序。对于同构的复合类型来说，在下面段中指定了元项的次序。

[1576] ○属性

[1577] 对象的“属性”字段被当做类型“列表”的未命名属性（它是命名属性的未分类容器）。在类型“列表”的属性值中所包含的命名属性按照它们的“名称”字段来依照词典方式分类。在类型“数组”的值属性中所包含的未命名属性未被分类（它们依照它们的数组次序来串行化）。

[1578] ○扩展

[1579] 对象的内部扩展按照它们的‘id’字段依照词典方式分类。在一个实施例中，对于内部扩展来说，在计算规范的字节序列中不使用‘extensionData’字段。对于这种扩展来说，如果为了签名目的需要把它们包括在摘要计算中，那么它们包含用于表示在‘extensionData’中携带的实际数据摘要的‘摘要’字段。对于每种类型的扩展数据来说，给出用于允许计算其规范字节序列的定义。

[1580] ○控制器

[1581] 内容密钥引用按照它们的‘id’字段依照词典方式分类。

[1582] 3.ScubaKeys

[1583] ‘publicKeys’、‘privateKeys’和‘secretKeys’字段中的密钥按照它们的‘id’字段依照词典方式分类。

[1584] 4. 例子

[1585]

```
Class X {
    int i;
    int j;
}

class A {
    int a[];
    string s;
}

class B extends A {
    {X optional_x; }
    X x;
    (string toDiscardInCano; )
    string s2;
}
```

[1586] 类B的实例的规范字节序列，其中a[] = {7,8,9}，s = “Abc”，x = {5,4}，s2 = “”，并且optional_x不存在被串行化为：

[1587]

3	7	8	9	3	“Abc” as UTF-8	0	Cano(X)	0
4字节	4字节	4字节	4字节	4字节	3字节	1字节	8字节	4字节

[1588] 其中Cano(X)为：

[1589]

5	4
4字节	4字节

[1590] 17. 附录E

[1591] 下面提供了控制程序的例子。在此例子中，许可表明如果可以在状态数据库（在此示例性实施例中被称为“Seashell”数据库）中找到成员资格状态（在注册期间准备）或许可状态（在许可转送期间准备），那么可以准许播放动作。所述许可还允许对等体请求

许可转送。如果两个对等体处于给定的邻近度内，那么准许此转送。许可包含用于在对等体上设置许可状态的代理。

[1592] 在下面的代码文件中，“MovableDomainBoundLicense.asm”是主控制，“LicenseUtils/*”是所述许可的帮助程序，“GenericUtils/*”是用于执行诸如计算字符串长度、比较字符串、操纵堆栈等功能的通用帮助程序，并且“ExtendedStatusBlockParameters/*”包含扩充状态块参数的 XML 描述和相应的表示作为根据 XML 所编译的字节系列。

[1593] E.1MovableDomainBound.asm

[1594]

```
; ********************************************************
;       文件名称: MovableDomainBoundLicense.asm
;       描述: 可移动许可的例子
; ********************************************************;


;  =========================
;  常数
;  =========================
.equ DEBUG_PRINT_SYSCALL,                1
.equ FIND_SYSCALL_BY_NAME_SYSCALL,       2
.equ SYSTEM_HOST_GET_OBJECT_SYSCALL,     3
.equ SYSTEM_HOST_SET_OBJECT_SYSCALL,     4
.equ SUCCESS,                            0
.equ FAILURE,                           -1
.equ ERROR_NO_SUCH_ITEM,                -6
```

[1595]

```
.equ CONTAINER_IGNORED_ADDRESS,          1

;  ========================
;  包括
;  ========================
.include "StrCmp.asm"
.include "PrintInt.asm"
.include "MembershipUtils.asm"
.include "LicenseStateUtils.asm"

; ========================
;  数据
; ========================
.data

GetTrustedTimeFunctionName:
    .string "System.Host.GetTrustedTime"
GetTrustedTimeFunctionNumber:
    .long 0
ActionGrantedNoObligationXStatus:
    .long 0x00000000 ;  全局标志
    .long 0x00000000 ;  类别 = ACTION_GRANTED
    .long 0x00000000 ;  子类别
    .long 0x00000000 ;  局部标志
    .long 0x00000000 ;  高速缓存持续时间类型
    .long 0x00000000 ;  高速缓存持续时间值
    .long 0x00000000 ;  值列表大小 = 0
ActionDeniedXStatus:
    .long 0x00000000 ;  全局标志
    .long 0x00000001 ;  类别 = ACTION_DENIED
    .long 0x00000000 ;  子类别
    .long 0x00000000 ;  局部标志
```

[1596]

```
        .long 0x00000000 ;  高速缓存持续时间类型
        .long 0x00000000 ;  高速缓存持续时间值
        .long 0x00000000 ;  值列表大小 = 0
TransferGrantedProximityNotChecked:
        .long 0x00000000 ;  全局标志
        .long 0x00000000 ;  类别 = ACTION_GRANTED
        .long 0x00000000 ;  子类别
        .long 0x00000003 ;  局部标志: 义务和回调通知
        .long 0x00000000 ;  高速缓存持续时间类型
        .long 0x00000000 ;  高速缓存持续时间值
        .include "TransferXStatusProximityCheckFailed.asm"
TransferGrantedProximityChecked:
        .long 0x00000000 ;  全局标志
        .long 0x00000000 ;  类别 = ACTION_GRANTED
        .long 0x00000000 ;  子类别
        .long 0x00000003 ;  局部标志: 义务和回调通知
        .long 0x00000000 ;  高速缓存持续时间类型
        .long 0x00000000 ;  高速缓存持续时间值
        .include "TransferXStatusProximityCheckSucceed.asm"
AgentContextPath:
        .string "Octopus/Agent/Session/ContextId"
AgentContextDesiredValue:
        .string "MoveStateContent0023"
AgentContextValue:
        .zeros 32
SinkProximityLastProbePath:
        .string "Octopus/Action/Parameters/Sink/Proximity/LastProbe"
SinkProximityLastProbeResult:
        .long -1
AgentProximityCheckedPath:
        .string "Octopus/Agent/Parameters/ProximityChecked"
AgentProximityCheckedValue:
```

[1597]

```
        .long 0
    ControllerTimestampAttributePath:
        .string "Octopus/Controller/Attributes/Import-time"
    ControllerTimestampAttributeValue:
        .long 0

    ;  调试
    .ifdef DEBUG

    Controller.Timestamp.Query.Debug:
        .string "--------- Entering Controller.Timestamp.Query ----------\n"
    Control.Actions.Play.Perform.Debug:
        .string "--------- Entering Control.Actions.Play.Perform ----------\n"
    Control.Agents.SetStateContent0023.Run.Debug:
        .string "--------- Entering Control.Agents.SetStateContent0023.Run ----------\n"
    Control.Actions.Transfer.Perform.Debug:
        .string "--------- Entering Control.Actions.Transfer.Perform ----------\n"
    Control.Agents.SetStateContent0023.OnAgentCompletion.Debug:
        .string "--------- Entering Control.Agents.SetStateContent0023.OnAgentCompletion ----------\n"
    Transfer_OK_Proximity_Not_Checked.Debug:
        .string "####### Transfer_OK_Proximity_Not_Checked #######\n"
    Transfer_OK_Proximity_Checked.Debug:
        .string "####### Transfer_OK_Proximity_Checked #######\n"
    Agent_Failure.Debug:
        .string "####### Agent Failure #######\n"
    Agent_Success.Debug:
        .string "####### Agent Success #######\n"
    Action_Granted.Debug:
        .string "####### Action Granted #######\n"
    Action_Denied.Debug:
        .string "####### Action Denied #######\n"
```

[1598]

```
.endif

; ========================
;  代码
; ========================
.code

;
;  Global.OnLoad
;
Global.OnLoad:
    ;  获取 GetTrustedTime functionNumber
    PUSH @GetTrustedTimeFunctionName
    PUSH FIND_SYSCALL_BY_NAME_SYSCALL
    CALL
    DUP
    PUSH @GetTrustedTimeFunctionNumber
    POKE
    BRN OnLoad_Failed

    ;  ok
    PUSH SUCCESS
    STOP

    ;  失败
OnLoad_Failed:
    PUSH FAILURE
    STOP

;
;  Controller.Timestamp.Query
;
```

[1599]

```
Controller.Timestamp.Query:
.ifdef DEBUG
    ; 调试
    PUSH @Controller.Timestamp.Query.Debug
    PUSH DEBUG_PRINT_SYSCALL
    CALL
.endif

    ; 获取控制器对象中的时间戳属性
    PUSH 4                                  ; 返回缓冲器大小（4 字节）    PUSH
@ControllerTimestampAttributeValue          ; 返回缓冲器 (类型是长整型)
    PUSH @ControllerTimestampAttributePath  ; 名称
    PUSH 0                                  ; 亲代 =根容器
    PUSH SYSTEM_HOST_GET_OBJECT_SYSCALL
    CALL
    RET


;
;  Control.Actions.Play.Check
;
Control.Actions.Play.Check:
;
;  Control.Actions.Play.Perform
;
Control.Actions.Play.Perform:
    ; 查询状态路径
    JSR MembershipStatePath.Query
    BRN Action_Denied

    JSR LicenseStatePath.Query
    BRN Action_Denied
```

[1600]

```
.ifdef DEBUG
    ; 调试
    PUSH @Control.Actions.Play.Perform.Debug
    PUSH DEBUG_PRINT_SYSCALL
    CALL
.endif

    ;  如果成员资格状态的时间戳为>
    ;  控制器的时间戳
    JSR MembershipStateValue.Query
    BRN Action_Denied
    JSR Controller.Timestamp.Query
    BRN Action_Denied
    PUSH @MembershipStateValue
    PEEK
    PUSH @ControllerTimestampAttributeValue
    PEEK
    SUB
    BRP Action_Granted ;  在这种情况下
                          ;  我们不必检查许可状态

    ;  我们只是检查状态存在
    JSR LicenseStateValue.Query
    BRN Action_Denied

Action_Granted:
.ifdef DEBUG
    ; 调试
    PUSH @Action_Granted.Debug
    PUSH DEBUG_PRINT_SYSCALL
    CALL
```

[1601]

```
.endif

    PUSH @ActionGrantedNoObligationXStatus
    PUSH SUCCESS
    STOP

Action_Denied:
.ifdef DEBUG
    ; 调试
    PUSH @Action_Denied.Debug
    PUSH DEBUG_PRINT_SYSCALL
    CALL
.endif

    PUSH @ActionDeniedXStatus
    PUSH SUCCESS
    STOP

;
;   Control.Actions.Transfer.Check
;
Control.Actions.Transfer.Check:
;
;   Control.Actions.Transfer.Perform
;
Control.Actions.Transfer.Perform:
    ;   查询状态路径
    JSR MembershipStatePath.Query
    BRN Action_Denied

    JSR LicenseStatePath.Query
    BRN Action_Denied
```

[1602]

```
.ifdef DEBUG
    ; 调试
    PUSH @Control.Actions.Transfer.Perform.Debug
    PUSH DEBUG_PRINT_SYSCALL
    CALL
.endif

    ;  获取已经检查邻近度的最近时间
    PUSH 4  ;  返回缓冲器大小
    PUSH @SinkProximityLastProbeResult  ;  返回缓冲器
    PUSH @SinkProximityLastProbePath  ;  名称
    PUSH 0  ;  亲代 = 根容器
    PUSH SYSTEM_HOST_GET_OBJECT_SYSCALL
    CALL
    ;  如果对象是不可见的，那么尚未检查邻近度。
    ;  所产生的代理（检查行 23 上的 TransferXStatusProximityCheckFailed.xml 文件）。然后代理在设置状态之前确信所述代理是域的一部分。
    BRN Transfer_OK_Proximity_Not_Checked

    ;  检查在最近 10 分钟已经检查邻近度
    DROP  ;  我们知道类型 id 为长整型
    DROP ;  我们知道大小为 4
    PUSH @GetTrustedTimeFunctionNumber
    PEEK
    CALL
    SWAP
    DROP ;  我们只需要该值
    PUSH @SinkProximityLastProbeResult
    PEEK
    SUB
    PUSH 10
```

[1603]

```
        SWAP
        SUB
        ;  检查邻近度的上次时间是大于 10’ 以前：同样
        ;  就好像根本尚未检查邻近度一样（参见上面）
        BRN Transfer_OK_Proximity_Not_Checked

        ;  已经成功地检查邻近度
    .ifdef DEBUG
        ; 调试
        PUSH @Transfer_OK_Proximity_Checked.Debug
        PUSH DEBUG_PRINT_SYSCALL
        CALL
    .endif

        PUSH @TransferGrantedProximityChecked
        PUSH SUCCESS
        STOP

    Transfer_OK_Proximity_Not_Checked:
    .ifdef DEBUG
        ; 调试
        PUSH @Transfer_OK_Proximity_Not_Checked.Debug
        PUSH DEBUG_PRINT_SYSCALL
        CALL
    .endif

        ;  返回 RunAgentOnPeer 义务（用于表明尚未检查邻近度）和 OnAgentCompletion 回调
        PUSH @TransferGrantedProximityNotChecked
        PUSH SUCCESS
        STOP
```

[1604]

```
;   Control.Agents.SetStateContent0023.Run
;
Control.Agents.SetStateContent0023.Run:
    ;   查询状态路径
    JSR MembershipStatePath.Query
    BRN Agent_Run_Failed

    JSR LicenseStatePath.Query
    BRN Agent_Run_Failed

.ifdef DEBUG
    ; 调试
    PUSH @Control.Agents.SetStateContent0023.Run.Debug
    PUSH DEBUG_PRINT_SYSCALL
    CALL
.endif

    ;   如果对等体处于域中，那么我们不必作任何事情
    JSR Membership.Check
    BRZ Agent_Success

    ;   检查环境被设置
    PUSH 32                 ;   返回缓冲器大小
    PUSH @AgentContextValue ;   返回缓冲器
    PUSH @AgentContextPath  ;   名称
    PUSH 0 ;   亲代 = 根容器
    PUSH SYSTEM_HOST_GET_OBJECT_SYSCALL
    CALL
    ;   检查结果
    BRN Agent_Run_Failed
    DROP   ;   我们知道类型 id 为字符串
    DROP ;   我们不关心大小
```

[1605]

```
        PUSH @AgentContextValue
        PUSH @AgentContextDesiredValue
        JSR streq  ;  确信我们处于良好的环境中
        BRN Agent_Run_Failed

        ;  检查源是否已经成功地邻近度检查了信宿
        PUSH 4                                  ;  返回缓冲器大小
        PUSH @AgentProximityCheckedValue        ;  返回缓冲器
        PUSH @AgentProximityCheckedPath         ;  名称
        PUSH 0  ;  亲代 = 根容器
        PUSH SYSTEM_HOST_GET_OBJECT_SYSCALL
        CALL
        ;  检查结果
        BRN Agent_Run_Failed  ;  此当接收代理时此参数应当始终被应用设置

        DROP  ;  我们知道类型 id 为长
        DROP;  我们知道大小为 4
        PUSH @AgentProximityCheckedValue
        PEEK
        NOT
        BRZ Agent_Set_State

    Agent_Run_Failed:
    .ifdef DEBUG
        ; 调试
        PUSH @Agent_Failure.Debug
        PUSH DEBUG_PRINT_SYSCALL
        CALL
    .endif

        PUSH 0;  返回块大小
        PUSH 0 ;  返回块地址
```

[1606]

```
        PUSH FAILURE ;  结果代码
        STOP

    Agent_Set_State:
        ;  设置状态
        JSR LicenseState.Set
        BRN Agent_Run_Failed

    Agent_Success:
    .ifdef DEBUG
        ; 调试
        PUSH @Agent_Success.Debug
        PUSH DEBUG_PRINT_SYSCALL
        CALL
    .endif

        ;  成功
        PUSH 0  ;  返回块大小
        PUSH 0 ;  返回块地址
        PUSH SUCCESS    ;  结果代码
        STOP

    ;
    ;  （类型 RESET 的）Control.Agents.SetStateContent0023.OnAgentCompletion 回调
    ;
    Control.Agents.SetStateContent0023.OnAgentCompletion:
    .ifdef DEBUG
        ; 调试
        PUSH @Control.Agents.SetStateContent0023.OnAgentCompletion.Debug
        PUSH DEBUG_PRINT_SYSCALL
        CALL
    .endif
```

[1607]

```
        ;  检查代理结果代码为 OK
        ;  堆栈为:
        ;  ... AgentResultCode CompletionStatusCode ArgumentsBlockSize Cookie
        DROP                                    ;  我们不需要 cookie
        DROP ;  我们不需要自变量块大小
        BRN Action_Denied ;  如果代理不能运行，那么失败
        BRN Action_Denied    ;  如果代理不能在对等体上设置状态那么同样失败

        ;  成功
        PUSH @ActionGrantedNoObligationXStatus
        PUSH SUCCESS
        STOP

Agent_Completion_Failed:
        PUSH FAILURE
        STOP

; ========================
;  导出
; ========================
.export Global.OnLoad
.export Control.Actions.Play.Check
.export Control.Actions.Play.Perform
.export Control.Actions.Transfer.Check
.export Control.Actions.Transfer.Perform
.export Control.Agents.SetStateContent0023.Run
.export Control.Agents.SetStateContent0023.OnAgentCompletion
```

E.2 LicenseUtils

E.2.1 LicenseStateUtils.asm

```
; ********************************************************
```

[1608]

```
;    文件名称：LicenseStateUtils.asm
;    描述：许可状态的实用程序
; ********************************************************;

; =========================
;  数据
; =========================
.data
LicenseStatePathControlAttribute:
    .string "Octopus/Control/Attributes/LicenseStatePath"
LicenseStatePath:
    .zeros 256
LicenseStateValue:
    .long 0

;  调试
LicenseStatePath.Query.Debug:
  .string "--------- Entering LicenseStatePath.Query ----------\n"
LicenseStateValue.Query.Debug:
  .string "--------- Entering LicenseStateValue.Query ----------\n"
LicenseState.Erase.Debug:
  .string "--------- Entering LicenseState.Erase ----------\n"
LicenseState.Set.Debug:
  .string "--------- Entering LicenseState.Set ----------\n"

; =========================
;  代码
; =========================
.code

;
;  LicenseStatePath.Query
```

[1609]

```
;
LicenseStatePath.Query:
    ; 调试
    PUSH @LicenseStatePath.Query.Debug
    PUSH DEBUG_PRINT_SYSCALL
    CALL

    PUSH 256 ;  返回缓冲器大小
    PUSH @LicenseStatePath  ;  返回缓冲器
    PUSH @LicenseStatePathControlAttribute    ;  名称
    PUSH 0 ;  亲代 = 根容器
    PUSH SYSTEM_HOST_GET_OBJECT_SYSCALL
    CALL
    RET

;
;  LicenseStateValue.Query
;
LicenseStateValue.Query:
    ; 调试
    PUSH @LicenseStateValue.Query.Debug
    PUSH DEBUG_PRINT_SYSCALL
    CALL

    PUSH 4 ;  返回缓冲器大小 (4 字节)
    PUSH @LicenseStateValue ;  返回缓冲器 (类型为长整型)
    PUSH @LicenseStatePath  ;  名称
    PUSH 0  ;  亲代 = 根容器
    PUSH SYSTEM_HOST_GET_OBJECT_SYSCALL
    CALL
    RET
```

[1610]

```
;
;   LicenseState.Erase
;
LicenseState.Erase:
    ; 调试
    PUSH @LicenseState.Erase.Debug
    PUSH DEBUG_PRINT_SYSCALL
    CALL

    ;  擦除局部状态
    PUSH 0 ;  对象大小 (容器)
    PUSH 0 ;  对象类型 (容器)
    PUSH 0   ;  删除容器     PUSH @LicenseStatePath      ;  名称
    PUSH 0 ;  亲代 = 根容器     PUSH SYSTEM_HOST_SET_OBJECT_SYSCALL
    CALL
    RET

;
;   LicenseState.Set
;
LicenseState.Set:
    ; 调试
    PUSH @LicenseState.Set.Debug
    PUSH DEBUG_PRINT_SYSCALL
    CALL

    ;  设置状态
    PUSH 0 ;  对象大小（容器）
    PUSH 0   ;  对象类型（容器）
    PUSH CONTAINER_IGNORED_ADDRESS
    PUSH @LicenseStatePath      ;  名称
    PUSH 0 ;  亲代=根容器
```

[1611]

```
    PUSH SYSTEM_HOST_SET_OBJECT_SYSCALL
    CALL
    RET
```

E.2.2 MembershipUtils.asm

```
; ************************************************************
;       文件名称：MembershipUtils.asm
;       描述：用于广播成员资格的实用程序
;
; ************************************************************;

; ========================
;  数据
; ========================
.data
MembershipStatePathControlAttribute:
    .string "Octopus/Control/Attributes/MembershipStatePath"
MembershipStatePath:
    .zeros 256
MembershipStateValue:
    .long 0

;  调试
intStrOutput:
    .string "............"
MembershipStatePath.Query.Debug:
  .string "--------- Entering MembershipStatePath.Query ----------\n"
MembershipStateValue.Query.Debug:
   .string "--------- Entering MembershipStateValue.Query ---------\n"
Membership.Check.Debug:
    .string "--------- Entering Membership.Check -------------\n"
Membership_Check_Success.Debug:
```

[1612]

```
        .string "####### Membership Check Success #######\n"
    Membership_Check_Failure.Debug:
        .string "####### Membership Check Failure #######\n"
    MembershipPath.Debug:
        .string "MembershipState path: "
    MembershipGetObjOutput.Debug:
        .string "MembershipState get object returns: "
    Membership_Expired.Debug:
        .string "MembershipState has expired. Check the Value of the Membership SeaShell token against
    the local time."
    NewlineString:
        .string "\n"

    ; ========================
    ;  代码
    ; ========================
    .code

    ;
    ;  MembershipStatePath.Query
    ;
    MembershipStatePath.Query:
        ; 调试
        PUSH @MembershipStatePath.Query.Debug
        PUSH DEBUG_PRINT_SYSCALL
        CALL

        PUSH 256                    ;  返回缓冲器大小
        PUSH @MembershipStatePath ;  返回缓冲器
        PUSH @MembershipStatePathControlAttribute ;  名称
        PUSH 0    ;  亲代 = 根容器
        PUSH SYSTEM_HOST_GET_OBJECT_SYSCALL
```

[1613]

```
        CALL
        RET

;
;   MembershipStateValue.Query
;
MembershipStateValue.Query:
        ; 调试
        PUSH @MembershipStateValue.Query.Debug
        PUSH DEBUG_PRINT_SYSCALL
        CALL
        PUSH @MembershipPath.Debug
        PUSH DEBUG_PRINT_SYSCALL
        CALL
        PUSH @MembershipStatePath
        PUSH DEBUG_PRINT_SYSCALL
        CALL
        PUSH @NewlineString
        PUSH DEBUG_PRINT_SYSCALL
        CALL

        PUSH 4                          ;  返回缓冲器大小 (4 字节)
        PUSH @MembershipStateValue      ;  返回缓冲器 (类型是长整型)
        PUSH @MembershipStatePath       ;  名称
        PUSH 0                          ;  亲代 =根容器
        PUSH SYSTEM_HOST_GET_OBJECT_SYSCALL
        CALL

        PUSH @MembershipGetObjOutput.Debug
        PUSH DEBUG_PRINT_SYSCALL
        CALL
        ;  打印结果 -首先把整型转换成字符串
```

[1614]

```
        DUP
        PUSH @intStrOutput
        ADD
        SWAP
        JSR printInt
        ;  调用打印结果
        PUSH @intStrOutput
        PUSH DEBUG_PRINT_SYSCALL
        CALL
        PUSH @NewlineString
        PUSH DEBUG_PRINT_SYSCALL
        CALL

        RET

;
;  Membership.Check
;
Membership.Check:
        ; 调试
        PUSH @Membership.Check.Debug
        PUSH DEBUG_PRINT_SYSCALL
        CALL

        ;  查询成员资格路径
        JSR MembershipStateValue.Query
        ;  看看我们是否成功
        BRN Membership_Check_Failed

        ;  检查时间<在成员资格状态中所获取的时间
        PUSH @MembershipStateValue ;  时间戳
        PEEK
```

[1615]

```
        PUSH @GetTrustedTimeFunctionNumber
        PEEK
        CALL
        SWAP
        DROP ;  我们只需要该值（而不是评估值）
        SUB
        BRN Membership_Expired

        ;  成功
        ; 调试
        PUSH @Membership_Check_Success.Debug
        PUSH DEBUG_PRINT_SYSCALL
        CALL

        PUSH SUCCESS
        RET

    Membership_Expired:
        ; 调试
        PUSH @Membership_Expired.Debug
        PUSH DEBUG_PRINT_SYSCALL
        CALL
        BRA Membership_Check_Failed

    Membership_Check_Failed:
        ; 调试
        PUSH @Membership_Check_Failure.Debug
        PUSH DEBUG_PRINT_SYSCALL
        CALL

        PUSH FAILURE
        RET
```

[1616]

E.3 GlobalUtils

E.3.1 IntUtils.asm

```
; ************************************************************
;       文件名称: IntUtils.asm
;       描述: 用于比较 2 个整数的实用程序
; ************************************************************


; =========================
;   包括
; =========================
.include "StackUtils.asm"


; =========================
;   代码
; =========================
.code


;
;   min
;
;   在 2 个整数之间计算最小值
;
;   输入: ... a b
;   输出: ... a<b?a:b
;
min:
      DUP                ;   ... a b b
      PUSH 2
      JSR pick           ;   ... a b b a
      CMP                ;   ... a b cmp_result
      BRN Swap_Stack     ;   ... a b
```

[1617]

```
    DROP
    RET                 ;  a

Swap_Stack:
    SWAP
    DROP
    RET                 ;  b

;
;  max
;
;  在 2 个整数之间计算最大值
;
;  输入: ... a b
;  输出: ... a>b?a:b
;
max:
    DUP                 ;  ... a b b
    PUSH 2
    JSR pick            ;  ... a b b a
    CMP                 ;  ... a b cmp_result
    NEG
    BRN Swap_Stack
    DROP
    RET                 ;  a
```

E.3.2 PrintInt.asm

```
; ************************************************************
;       文件名称: PrintInt.asm
;       描述: 把整数(有符号或无符号)转换为字符串
; ************************************************************
```

[1618]

```
; ;  注意：要求已经包括“StackUtils.asm”

;  数据段
    .data

;  代码段
    .code

;  把整数转换为字符串表示
;  参数：dest，int
printInt:
    ;  复制 dest 参数
    SWAP
    DUP
    PUSH 2
    JSR pick
    ;  STACK: origval, startstring, startstring, unsignedval
    ;  现在我们最后以整数的额外拷贝结束
    ;  我们使用这来稍后测试原始的签名
printIntLoop:
    ;  获取单个数字
    DUP
    PUSH 10
    MOD
    ;  转换成 ascii
    PUSH 48 ;   '0'的 ASCII
    ADD
    ;  获取输出缓冲器的地址
    PUSH 2
    JSR pick
    POKEB ;  打印到缓冲器
    ;  向前移动我们的缓冲器指针
```

[1619]

```
    SWAP
    PUSH 1
    ADD
    SWAP
    ;  除以 10 并且看看我们在哪
    PUSH 10
    DIV
    DUP
    BRP printIntLoop
    DROP ;  去除顶部的 0
    ;  STACK = orignum, startofstring, endofstring
    ;  如果原始数字为负，那么加入负号
    ;  以空终止
    DUP
    PUSH 0
    SWAP
    POKEB
    ;  把字符串末尾向上移动 1，因此不翻转终结符
    PUSH 1
    SUB
    ;  我们完成：只需要反向字符串
fliploop:
    ;  获取第二字节
    DUP
    PEEKB
    ;  获取第一字节
    PUSH 2
    JSR pick
    PEEKB
    ;  把第一字节放到最后地方
    PUSH 2
    JSR pick
```

[1620]

```
POKEB
;  把最后字节放到第一地方
PUSH 2
JSR pick
POKEB
;  把末尾指针向上移动一
PUSH 1
SUB
;  把开始指针向下移动一
SWAP
PUSH 1
ADD
SWAP
;  看看是否已经满足指针
;  首先必须复制值
DUP
PUSH 2
JSR pick
SUB
BRP fliploop
;  除去堆栈上的一些残余
DROP
DROP
DROP
RET
```

E.3.3 StackUtils.asm

```
; ************************************************************
;      文件名称: StackUtils.asm
;      描述: 根据 FORTH 产生的堆栈实用程序函数
;
; ************************************************************
```

[1621]

```
.ifndef _STACK_UTILS_
.define _STACK_UTILS_

; ==========================
;  代码
; ==========================
.code

;
;  结束
;
;  拷贝堆栈上的第二项
;
;  输入: ... a b
;  输出: ... a b a
;
over:
    PUSH 1
    JSR pick
    RET

;
;  挑选
;
;  输入: ... v3 v2 v1 v0 N
;  输出: ... v3 v2 v1 v0 vN
;
pick:
    PUSH 1
    ADD
    PUSH 4
    MUL
```

[1622]

```
    PUSHSP
    ADD
    PEEK
    RET

.endif ;  _STACK_UTILS_
```

E.3.4 StdLib.asm

```
;   用于 Plankton 的标准库
    .equ HEAP_ADDR, 16

;  数据段
;  代码段
    .code
strlen:
    DUP
loop:
    DUP
    PEEKB
    BRZ done
    PUSH 1
    ADD
    BRA loop

done:
    SWAP
    SUB
    RET

    .export strlen
```

E.3.5 StrCmp.asm

```
; ************************************************************
;       文件名称: StrCmp.asm
```

[1623]

```
;       描述: 用于两个字符串等同的 streq 测试
; ********************************************************


.ifndef _STR_CMP_
.define _STR_CMP_


; =========================
;   包括
; =========================
.include "StackUtils.asm"


; =========================
;   代码
; =========================
.code


;
;   streq
;
;   用于两个字符串等同性的测试
;
;   输入: ... @str1 @str2
;   输出: ... res (如果字符串相同，那么 res = 0，否则为-1)
;
streq:
    ; 获取两个字符串之间的偏移
    JSR over
    SUB
    SWAP                ;   ...偏移 @str1


streqloop:
    ;   获取 str1 的 cur 字符
```

[1624]

```
    DUP
    PEEKB                ;  ... 偏移 @str1 char1

    ;  get the cur char of str2
    JSR over             ;  ... 偏移@str1 char1 @str1
    PUSH 3
    JSR pick             ;  ... off 偏移 set @str1 char1 @str1 偏移
    ADD
    PEEKB                ;  ... 偏移@str1 char1 char2

    ;  现在比较两个字符
    JSR over
    SUB                  ;  ... 偏移@str1 char1 char1-char2
    ;  如果 char1 != char2，则失败
    NOT
    BRZ streqfailure
    ;  如果 char1 为 0 (char1 == char2 == 0)，那么已经结束了
    BRZ streqsuccess     ;  ... 偏移@str1

    ;  增加@str1 指针并且循环
    PUSH 1
    ADD
    BRA streqloop

streqfailure:
    ;  ... 偏移@str1 char1
    DROP
    DROP
    DROP
    PUSH -1
    RET
```

[1625]

```
streqsuccess:
    ;   ... 偏移@str1
    DROP
    DROP
    PUSH 0
    RET

.endif ;   _STR_CMP_
```

E.4 ExtendedStatusBlock Parameters

E.4.1 TransferXStatusProximityCheckSucceeded.xml

```
- <ValueListBlock>
  - <ValueBlock type="Parameter">
    - <ParameterBlock name="Obligations">
      - <ValueBlock type="ValueList">
        - <ValueListBlock>
          - <ValueBlock type="ExtendedParameter">
            - <ParameterBlock name="RunAgentOnPeer" flags="1">
              - <ValueBlock type="ValueList">
                - <ValueListBlock>
                  -<!-- 控制 ID  -->
                    <ValueBlock type="String">urn:marlin:control:0023</ValueBlock>
                -<!-- 代理名称 -->
                    <ValueBlock type="String">SetStateContent0023</ValueBlock>
                -<!-- 实例 ID  -->
                  <ValueBlock type="Integer">240343</ValueBlock>
                -<!-- 上下文 ID  -->
                  <ValueBlock type="String">MoveStateContent0023</ValueBlock>
                -<!-- 附加参数 -->
                - <ValueBlock type="ValueList">
                  - <ValueListBlock>
                    - <ValueBlock type="Parameter">
                      -<!--随着 TransferXStatusProximityCheckFailed.xml 变化的仅有的事务
```

[1626]

```
-->
                                - <ParameterBlock name="ProximityChecked">
                                    <ValueBlock type="Integer">1</ValueBlock>
                                  </ParameterBlock>
                                </ValueBlock>
                              </ValueListBlock>
                            </ValueBlock>
                          </ValueListBlock>
                        </ValueBlock>
                      </ParameterBlock>
                    </ValueBlock>
                  </ValueListBlock>
                </ValueBlock>
              </ParameterBlock>
            </ValueBlock>
          - <ValueBlock type="Parameter">
            - <ParameterBlock name="Callbacks">
              - <ValueBlock type="ValueList">
                - <ValueListBlock>
                  - <ValueBlock type="ExtendedParameter">
                    - <ParameterBlock name="OnAgentCompletion" flags="1">
                      - <ValueBlock type="ValueList">
                        - <ValueListBlock>
                        -<!-- 代理实例 ID -->
                          <ValueBlock type="String">240343</ValueBlock>
                        -<!—回调例程 -->
                        - <ValueBlock type="ValueList">
                        - <ValueListBlock>
                        -<!-- 复位 -->
                          <ValueBlock type="Integer">0</ValueBlock>
                        -<!-- 名称 -->
                    <ValueBlock
```

[1627]

```
type="String">Control.Agents.SetStateContent0023.OnAgentCompletion</ValueBlock>
- <!-- 点心文件 -->
  <ValueBlock type="Integer">0</ValueBlock>
                    </ValueListBlock>
                  </ValueBlock>
                 </ValueListBlock>
                </ValueBlock>
               </ParameterBlock>
              </ValueBlock>
             </ValueListBlock>
            </ValueBlock>
           </ParameterBlock>
          </ValueBlock>
         </ValueListBlock>
```

E.4.2 TransferXStatusProximityCheckFailed.xml

```
- <ValueListBlock>
  - <ValueBlock type="Parameter">
    - <ParameterBlock name="Obligations">
- <ValueBlock type="ValueList">
  - <ValueListBlock>
    - <ValueBlock type="ExtendedParameter">
      - <ParameterBlock name="RunAgentOnPeer" flags="1">
        - <ValueBlock type="ValueList">
        - <ValueListBlock>
          -<!—控制 ID -->
  <ValueBlock type="String">urn:marlin:control:0023</ValueBlock>
- <!-- 代理名称 -->
  <ValueBlock type="String">SetStateContent0023</ValueBlock>
- <!-- 实例 ID -->
  <ValueBlock type="Integer">240343</ValueBlock>
- <!-- 上下文 ID -->
  <ValueBlock type="String">MoveStateContent0023</ValueBlock>
```

[1628]

```
- <!--  附加参数  -->
- <ValueBlock type="ValueList">
    - <ValueListBlock>
      - <ValueBlock type="Parameter">
            -<!—随 TransferXStatusProximityCheckSucceed.xml 改变的仅有的事务 -->
        - <ParameterBlock name="ProximityChecked">
<ValueBlock type="Integer">0</ValueBlock>
                    </ParameterBlock>
                  </ValueBlock>
  </ValueListBlock>
          </ValueBlock>
                  </ValueListBlock>
                </ValueBlock>
                </ParameterBlock>
              </ValueBlock>
            </ValueListBlock>
          </ValueBlock>
        </ParameterBlock>
      </ValueBlock>
  - <ValueBlock type="Parameter">
    - <ParameterBlock name="Callbacks">
      - <ValueBlock type="ValueList">
        - <ValueListBlock>
          - <ValueBlock type="ExtendedParameter">
            - <ParameterBlock name="OnAgentCompletion" flags="1">
              - <ValueBlock type="ValueList">
                - <ValueListBlock>
                -<!--  代理实例 ID  -->
                  <ValueBlock type="String">240343</ValueBlock>
                -<!—回调例程  -->
                    - <ValueBlock type="ValueList">
                      - <ValueListBlock>
```

[1629]

```
-<!-- 复位 -->
<ValueBlock type="Integer">0</ValueBlock>
-<!-- 名称 -->
<ValueBlock
type="String">Control.Agents.SetStateContent0023.OnAgentCompletion</ValueBlock>
-<!-- 点心文件 -->
<ValueBlock type="Integer">0</ValueBlock>
</ValueListBlock>
</ValueBlock>
</ValueListBlock>
</ValueBlock>
</ParameterBlock>
</ValueBlock>
</ValueListBlock>
</ValueBlock>
</ParameterBlock>
</ValueBlock>
</ValueListBlock>
```

E.4.3 TransferXStatusProximityCheckSucceeded.asm

```
0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x04 0x00 0x00 0x00 0x20 0x00 0x00
0x00 0x0C 0x4F 0x62 0x6C 0x69 0x67 0x61 0x74 0x69 0x6F 0x6E 0x73 0x00
0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x0C 0x00 0x00 0x00 0x01 0x00 0x00
0x00 0x05 0x00 0x00 0x00 0x04 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x0F
0x52 0x75 0x6E 0x41 0x67 0x65 0x6E 0x74 0x4F 0x6E 0x50 0x65 0x65 0x72
0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x92 0x00 0x00 0x00 0x05 0x00
0x00 0x00 0x02 0x00 0x00 0x00 0x18 0x75 0x72 0x6E 0x3A 0x6D 0x61 0x72
0x6C 0x69 0x6E 0x3A 0x63 0x6F 0x6E 0x74 0x72 0x6F 0x6C 0x3A 0x30 0x30
0x32 0x33 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x14 0x53 0x65 0x74
0x53 0x74 0x61 0x74 0x65 0x43 0x6F 0x6E 0x74 0x65 0x6E 0x74 0x30 0x30
0x32 0x33 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x04 0x00 0x03 0xAA
0xD7 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x15 0x4D 0x6F 0x76 0x65 0x53
0x74 0x61 0x74 0x65 0x43 0x6F 0x6E 0x74 0x65 0x6E 0x74 0x30 0x30 0x32
```

[1630]

0x33 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x25 0x00 0x00 0x00 0x01

0x00 0x00 0x00 0x04 0x00 0x00 0x00 0x1D 0x00 0x00 0x00 0x11 0x50 0x72

0x6F 0x78 0x69 0x6D 0x69 0x74 0x79 0x43 0x68 0x65 0x63 0x6B 0x65 0x64

0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x04 0x00 0x00 0x00 0x01 0x00

0x00 0x00 0x04 0x00 0x00 0x00 0x1E 0x00 0x00 0x00 0x0A 0x43 0x61 0x6C

0x6C 0x62 0x61 0x63 0x6B 0x73 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00

0x0C 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x05 0x00 0x00 0x00 0x04 0x00

0x00 0x00 0x01 0x00 0x00 0x00 0x12 0x4F 0x6E 0x41 0x67 0x65 0x6E 0x74

0x43 0x6F 0x6D 0x70 0x6C 0x65 0x74 0x69 0x6F 0x6E 0x00 0x00 0x00 0x00

0x07 0x00 0x00 0x00 0x6C 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x02 0x00

0x00 0x00 0x07 0x32 0x34 0x30 0x33 0x34 0x33 0x00 0x00 0x00 0x00 0x07

0x00 0x00 0x00 0x55 0x00 0x00 0x00 0x03 0x00 0x00 0x00 0x00 0x00 0x00

0x00 0x04 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x35

0x43 0x6F 0x6E 0x74 0x72 0x6F 0x6C 0x2E 0x41 0x67 0x65 0x6E 0x74 0x73

0x2E 0x53 0x65 0x74 0x53 0x74 0x61 0x74 0x65 0x43 0x6F 0x6E 0x74 0x65

0x6E 0x74 0x30 0x30 0x32 0x33 0x2E 0x4F 0x6E 0x41 0x67 0x65 0x6E 0x74

0x43 0x6F 0x6D 0x70 0x6C 0x65 0x74 0x69 0x6F 0x6E 0x00 0x00 0x00 0x00

0x00 0x00 0x00 0x00 0x04 0x00 0x00 0x00 0x00

E.4.4 TransferXStatusProximityCheckFailed.asm

0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x04 0x00 0x00 0x00 0x20 0x00 0x00

0x00 0x0C 0x4F 0x62 0x6C 0x69 0x67 0x61 0x74 0x69 0x6F 0x6E 0x73 0x00

0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x0C 0x00 0x00 0x00 0x01 0x00 0x00

0x00 0x05 0x00 0x00 0x00 0x04 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x0F

0x52 0x75 0x6E 0x41 0x67 0x65 0x6E 0x74 0x4F 0x6E 0x50 0x65 0x65 0x72

0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x92 0x00 0x00 0x00 0x05 0x00

0x00 0x00 0x02 0x00 0x00 0x00 0x18 0x75 0x72 0x6E 0x3A 0x6D 0x61 0x72

0x6C 0x69 0x6E 0x3A 0x63 0x6F 0x6E 0x74 0x72 0x6F 0x6C 0x3A 0x30 0x30

0x32 0x33 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x14 0x53 0x65 0x74

0x53 0x74 0x61 0x74 0x65 0x43 0x6F 0x6E 0x74 0x65 0x6E 0x74 0x30 0x30

0x32 0x33 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x04 0x00 0x03 0xAA

0xD7 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x15 0x4D 0x6F 0x76 0x65 0x53

0x74 0x61 0x74 0x65 0x43 0x6F 0x6E 0x74 0x65 0x6E 0x74 0x30 0x30 0x32

[1631]

0x33 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x25 0x00 0x00 0x00 0x01

0x00 0x00 0x00 0x04 0x00 0x00 0x00 0x1D 0x00 0x00 0x00 0x11 0x50 0x72

0x6F 0x78 0x69 0x6D 0x69 0x74 0x79 0x43 0x68 0x65 0x63 0x6B 0x65 0x64

0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x04 0x00 0x00 0x00 0x00 0x00

0x00 0x00 0x04 0x00 0x00 0x00 0x1E 0x00 0x00 0x00 0x0A 0x43 0x61 0x6C

0x6C 0x62 0x61 0x63 0x6B 0x73 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00

0x0C 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x05 0x00 0x00 0x00 0x04 0x00

0x00 0x00 0x01 0x00 0x00 0x00 0x12 0x4F 0x6E 0x41 0x67 0x65 0x6E 0x74

0x43 0x6F 0x6D 0x70 0x6C 0x65 0x74 0x69 0x6F 0x6E 0x00 0x00 0x00 0x00

0x07 0x00 0x00 0x00 0x6C 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x02 0x00

0x00 0x00 0x07 0x32 0x34 0x30 0x33 0x34 0x33 0x00 0x00 0x00 0x00 0x07

0x00 0x00 0x00 0x55 0x00 0x00 0x00 0x03 0x00 0x00 0x00 0x00 0x00 0x00

0x00 0x04 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x35

0x43 0x6F 0x6E 0x74 0x72 0x6F 0x6C 0x2E 0x41 0x67 0x65 0x6E 0x74 0x73

0x2E 0x53 0x65 0x74 0x53 0x74 0x61 0x74 0x65 0x43 0x6F 0x6E 0x74 0x65

0x6E 0x74 0x30 0x30 0x32 0x33 0x2E 0x4F 0x6E 0x41 0x67 0x65 0x6E 0x74

0x43 0x6F 0x6D 0x70 0x6C 0x65 0x74 0x69 0x6F 0x6E 0x00 0x00 0x00 0x00

0x00 0x00 0x00 0x00 0x04 0x00 0x00 0x00 0x00

[1632] 尽管为了理解清楚已经相当详细地描述了上述发明，然而在所附权利要求的范围内显然可以进行某些改变和修改。应当注意，存在用于实现这里所描述过程和设备的许多候选方式。据此，本实施例被认为是说明性的而不是限制性的，并且所发明的工作主体不限于这里所给出的细节，而是可以在所附权利要求的范围和等效方式内进行修改。

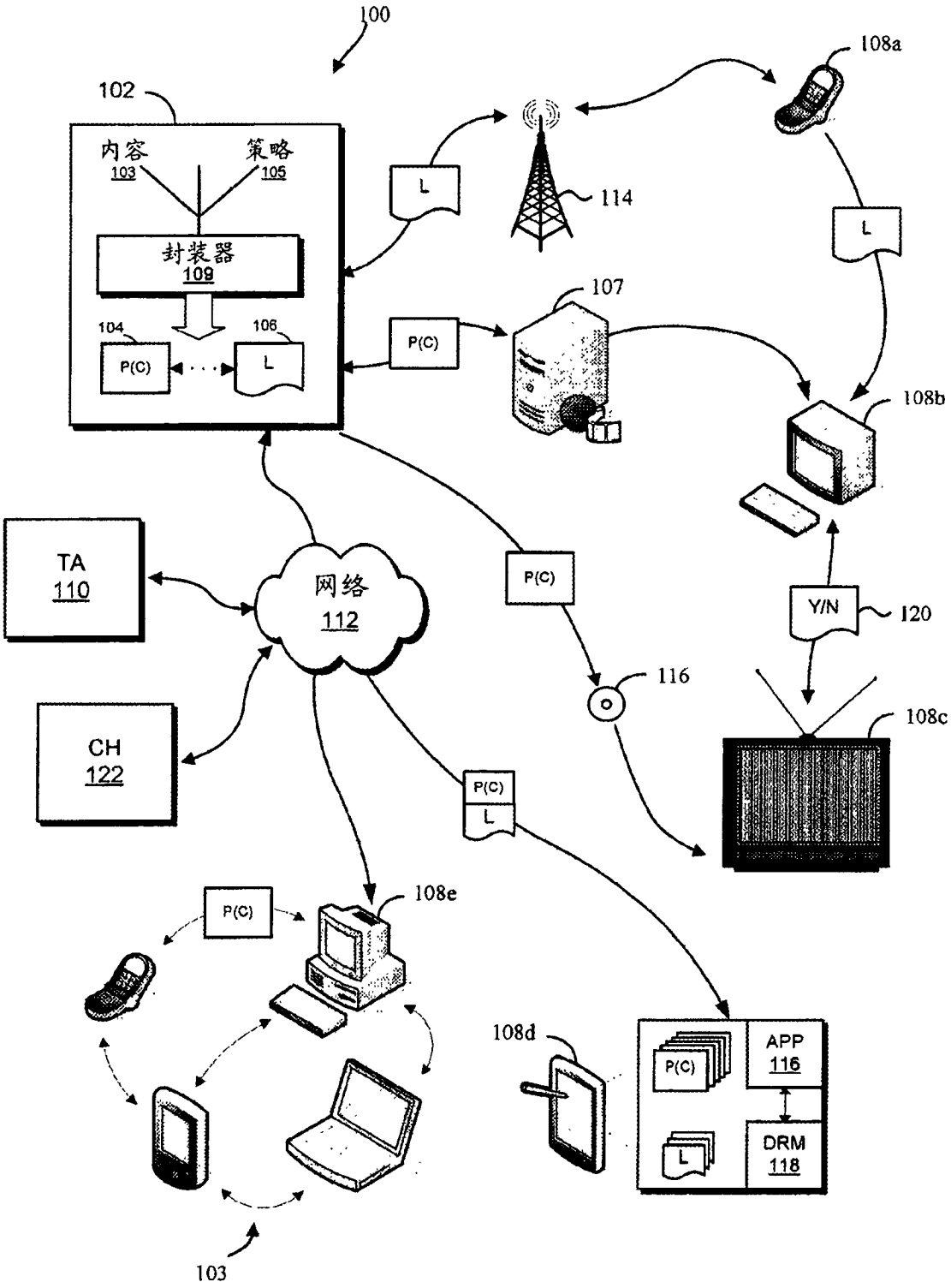

100
102
内容
103
策略
105
封装器
109
104
106
P(C)
L
108a
114
107
108b
TA
110
网络
112
CH
122
Y/N
120
116
108c
108e
108d
APP
116
DRM
118
103

图 1

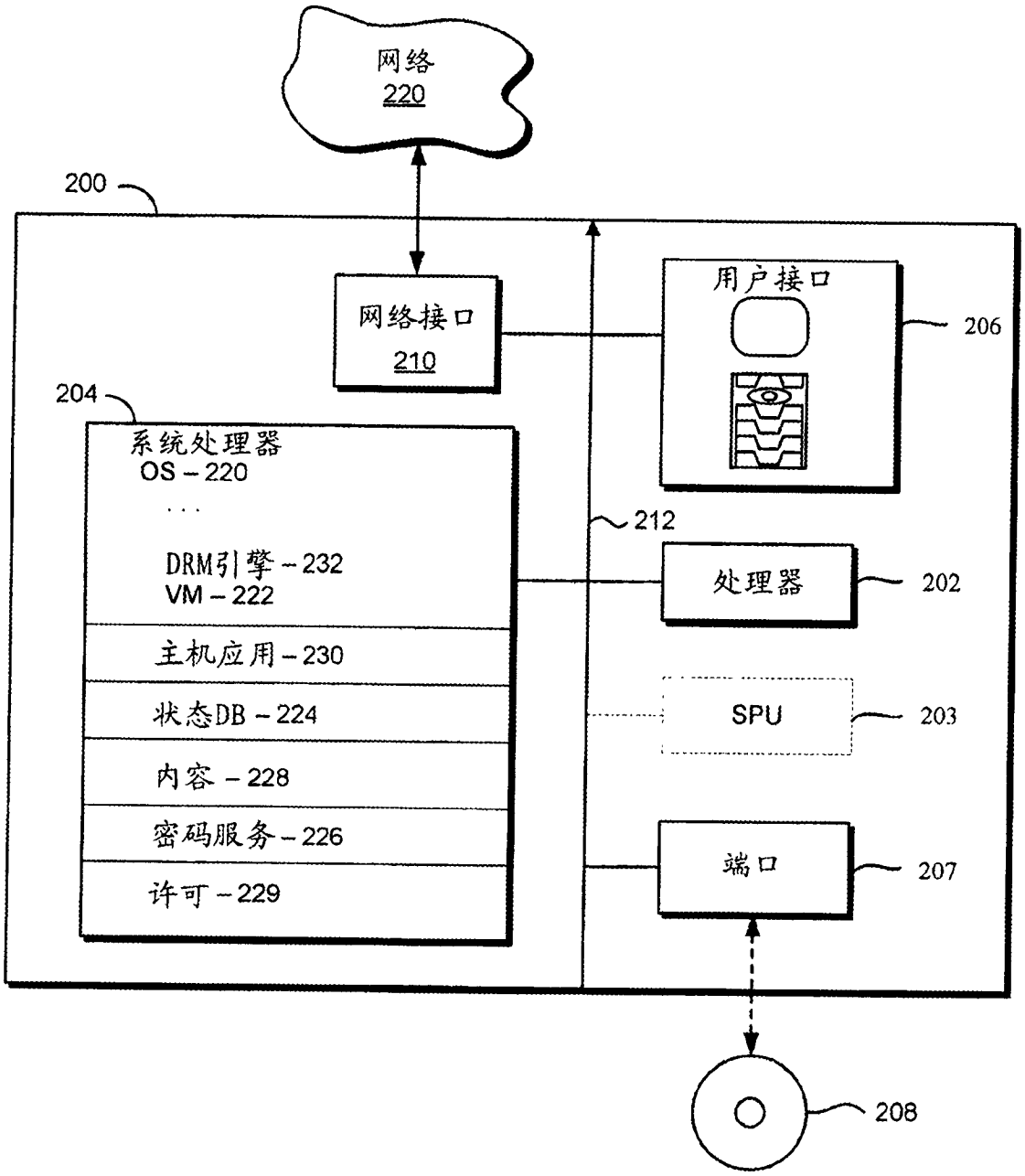

网络
220
200
网络接口
210
用户接口
206
204
系统处理器
OS – 220
...
DRM引擎 – 232
VM – 222
主机应用 – 230
状态DB – 224
内容 – 228
密码服务 – 226
许可 – 229
212
处理器
202
SPU
203
端口
207
208

图 2

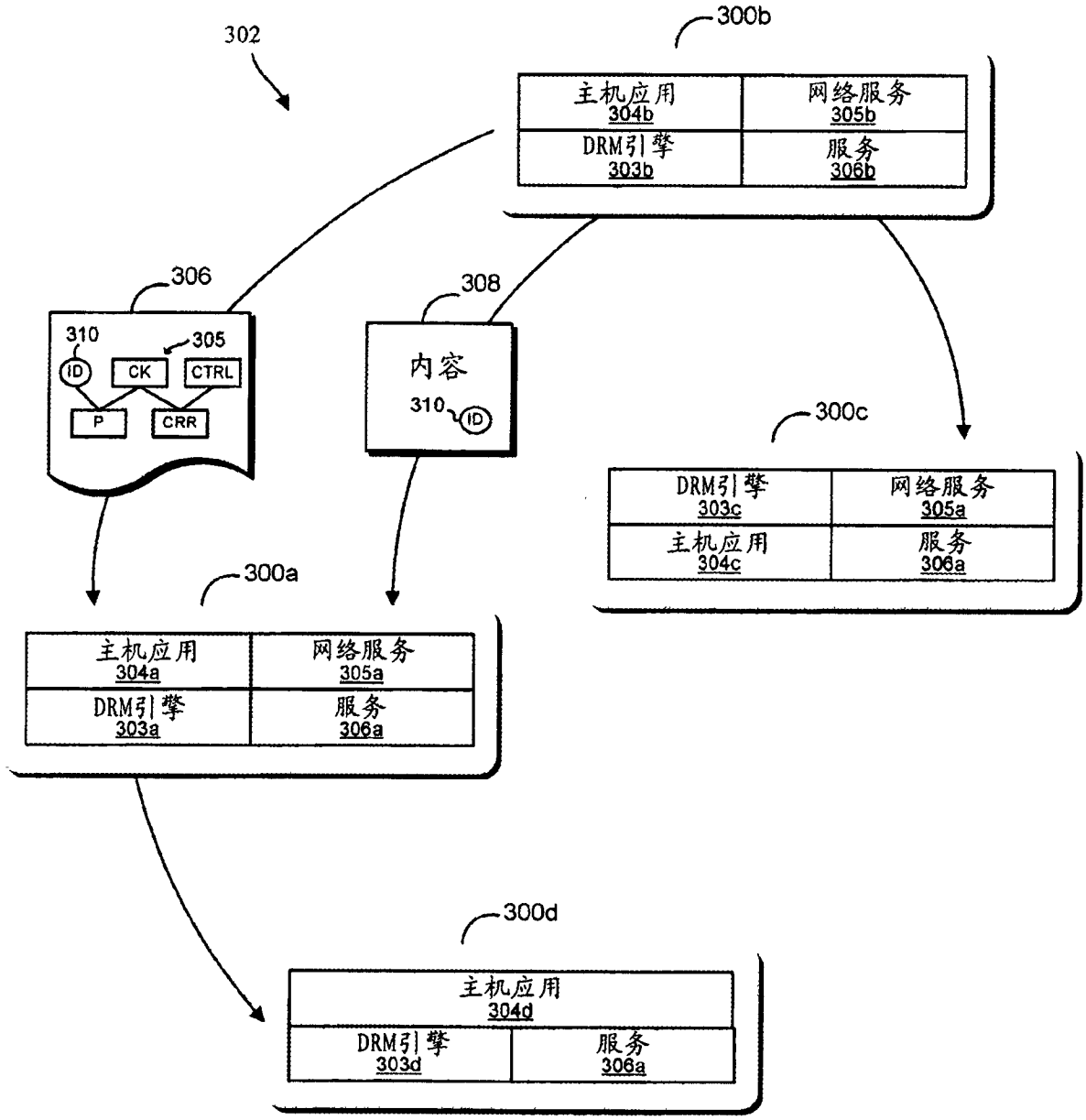

302
300b
主机应用
304b
网络服务
305b
DRM引擎
303b
服务
306b
306
310
ID
305
CK
CTRL
P
CRR
308
内容
310
ID
300c
DRM引擎
303c
网络服务
305a
主机应用
304c
服务
306a
300a
主机应用
304a
网络服务
305a
DRM引擎
303a
服务
306a
300d
主机应用
304d
DRM引擎
303d
服务
306a

图 3

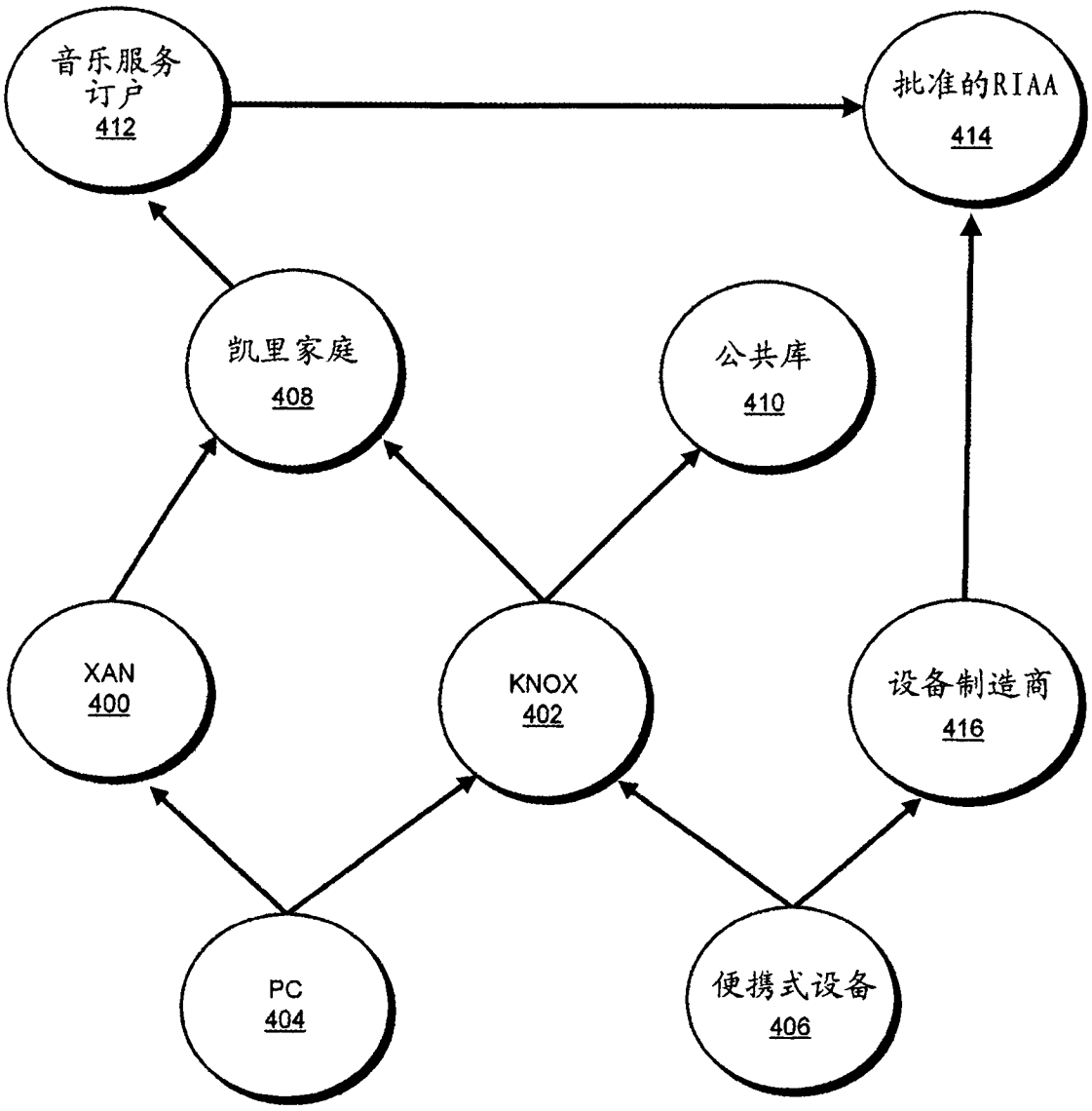

音乐服务
订户
412
批准的RIAA
414
凯里家庭
408
公共库
410
XAN
400
KNOX
402
设备制造商
416
PC
404
便携式设备
406

图 4

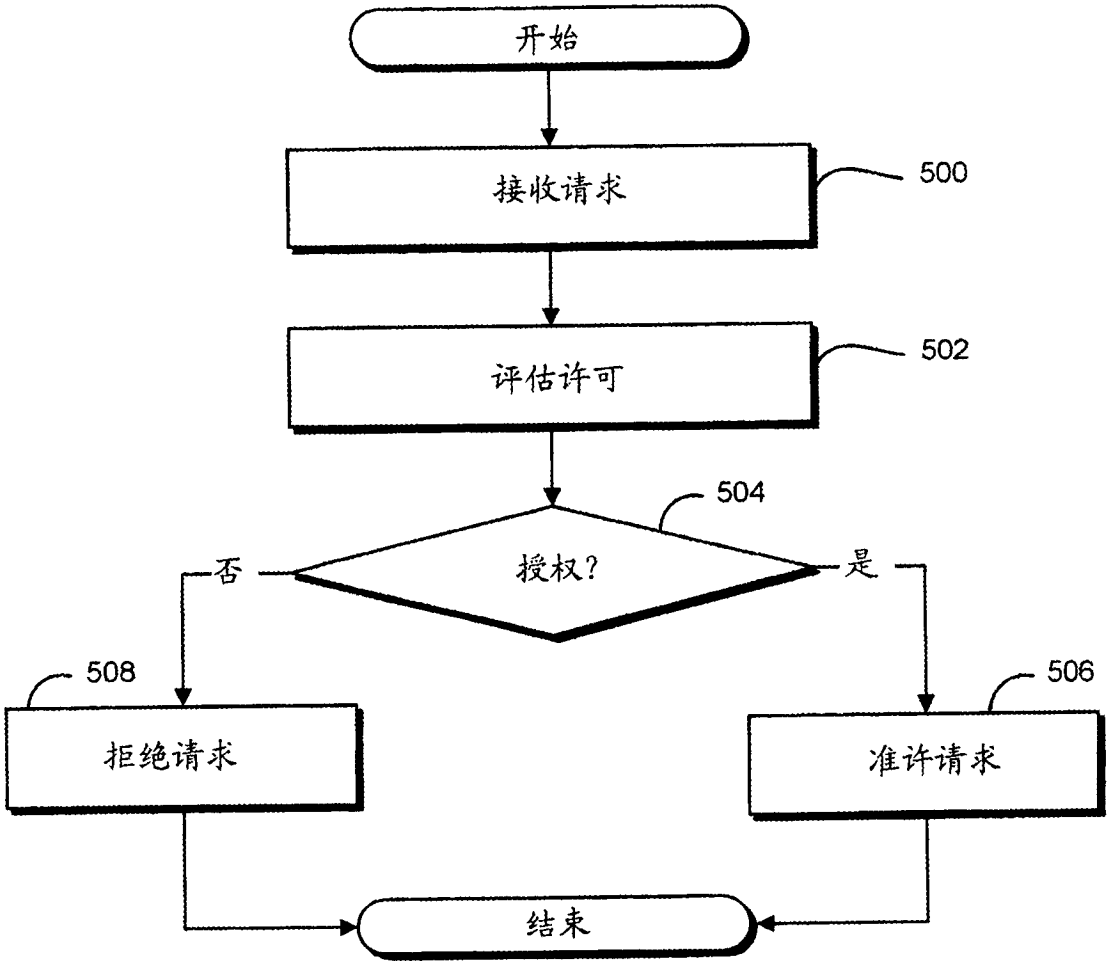


图 5

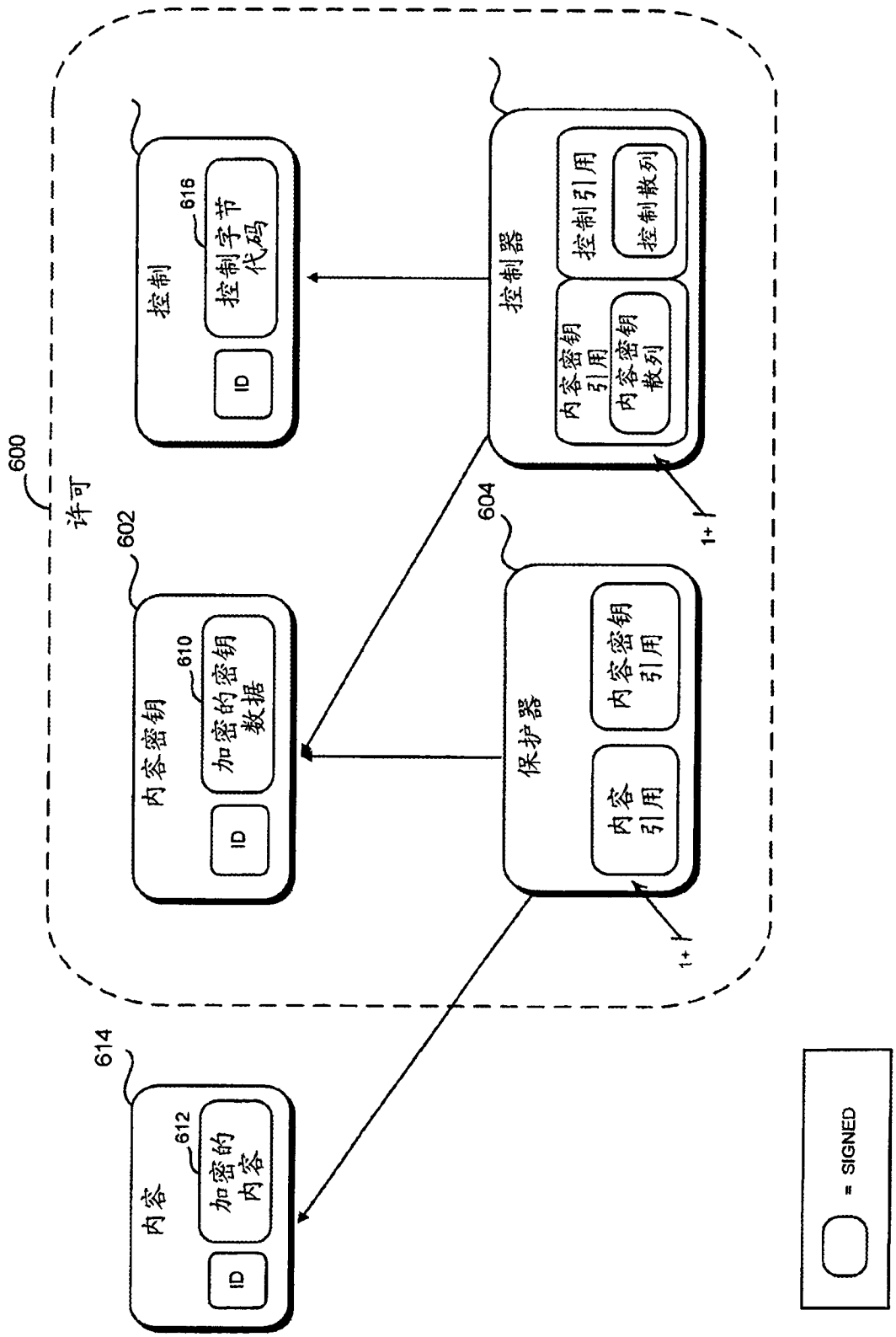

600
许可
602
内容密钥
ID
610
加密的密钥数据
604
保护器
内容引用
内容密钥引用
1+
控制
ID
616
控制字节代码
控制器
内容密钥引用
内容密钥散列
控制引用
控制散列
1+
614
内容
ID
612
加密的内容
= SIGNED

图 6

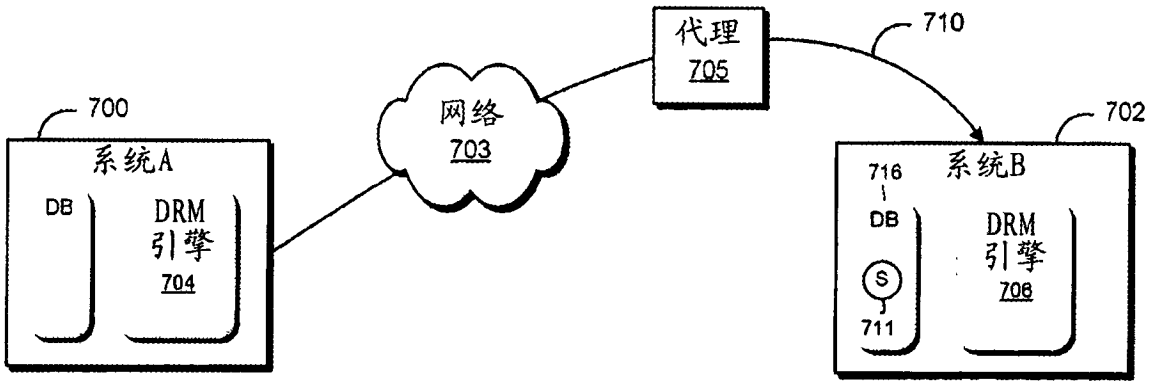

代理
705
710
网络
703
700
系统A
DB
DRM
引擎
704
716
系统B
DB
S
711
DRM
引擎
706
702

图 7A

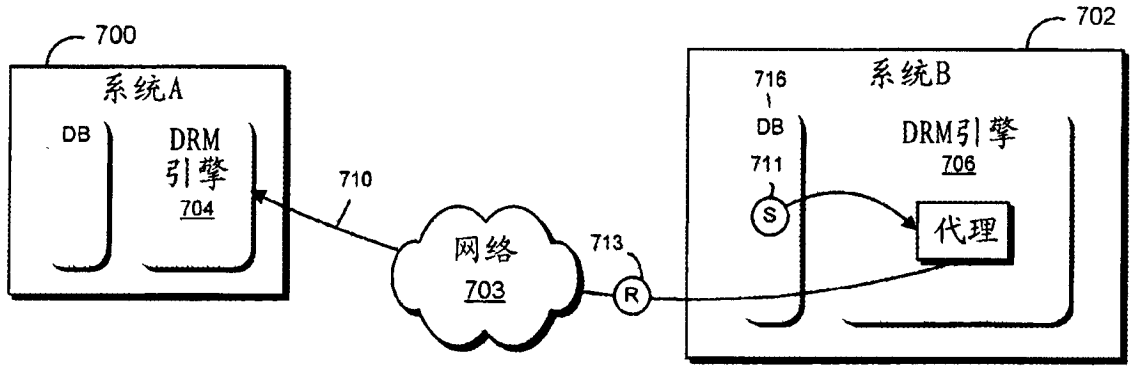

700
系统A
DB
DRM
引擎
704
710
网络
703
713
R
716
系统B
DB
711
S
DRM引擎
706
代理
702

图 7B

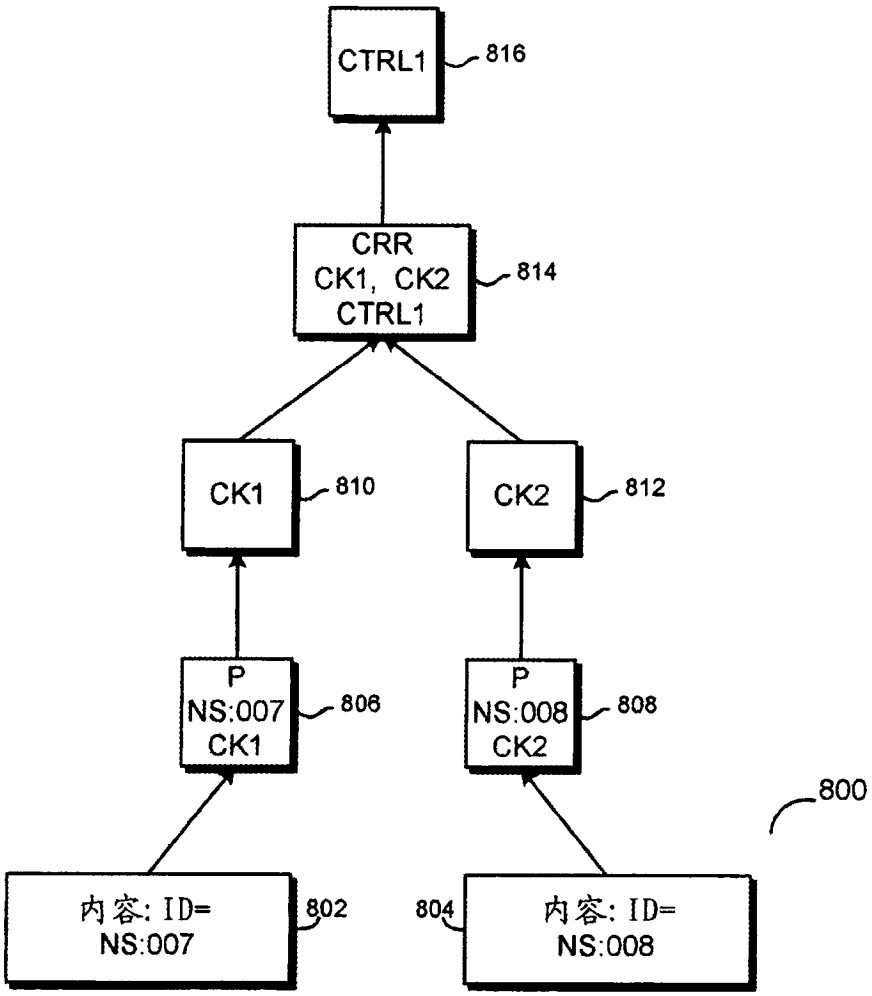

CTRL1
816
CRR
CK1, CK2
CTRL1
814
CK1
810
CK2
812
P
NS:007
CK1
806
P
NS:008
CK2
808
800
内容: ID=
NS:007
802
804
内容: ID=
NS:008

图 8

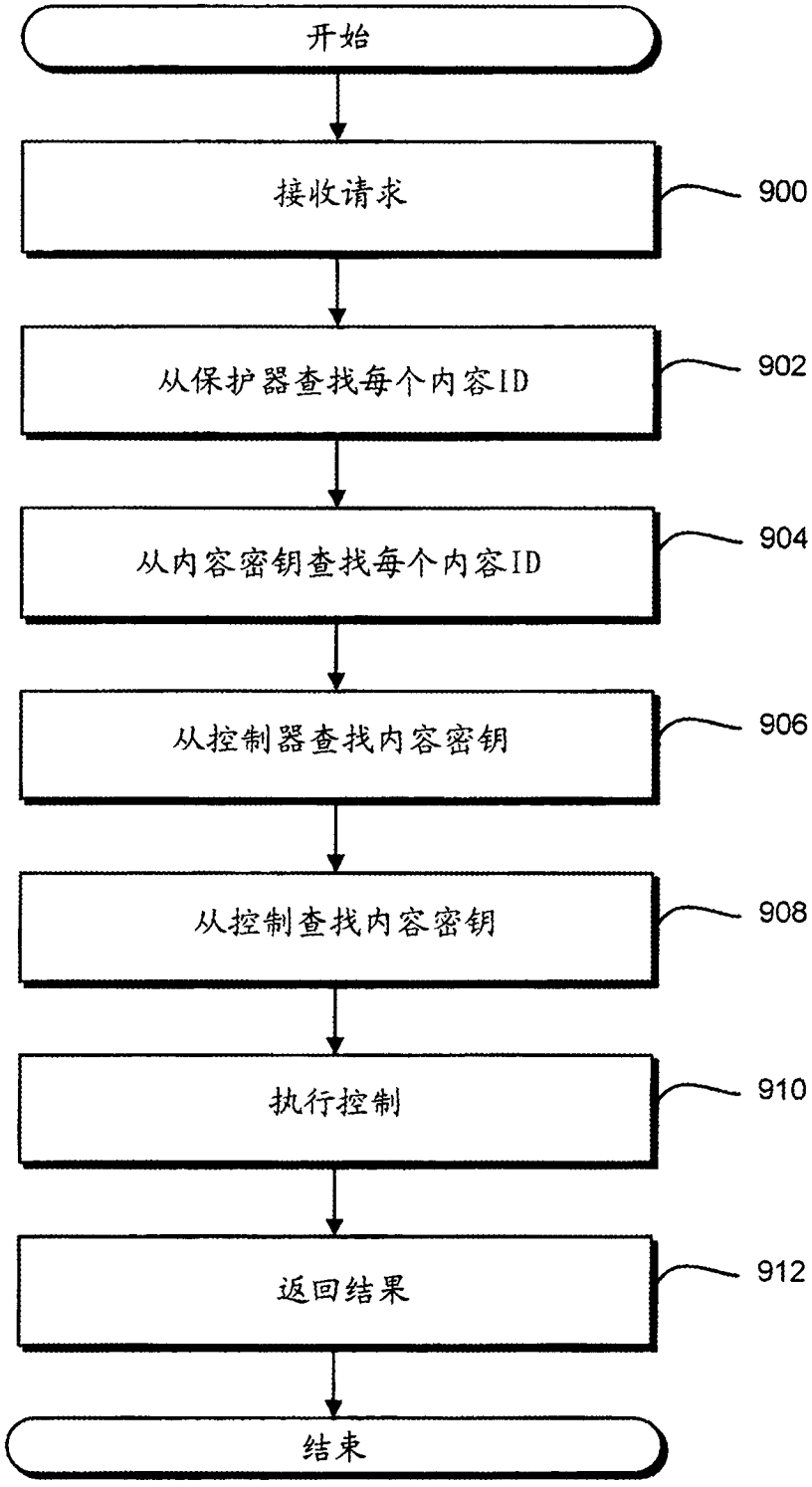

开始
接收请求
900
从保护器查找每个内容ID
902
从内容密钥查找每个内容ID
904
从控制器查找内容密钥
906
从控制查找内容密钥
908
执行控制
910
返回结果
912
结束

图 9

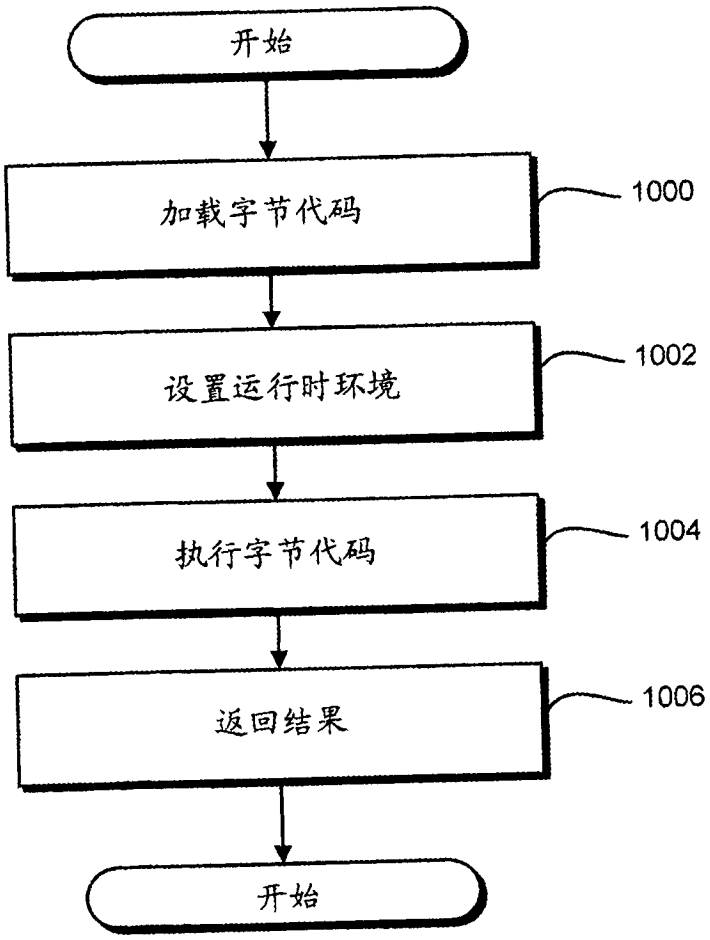


图 10

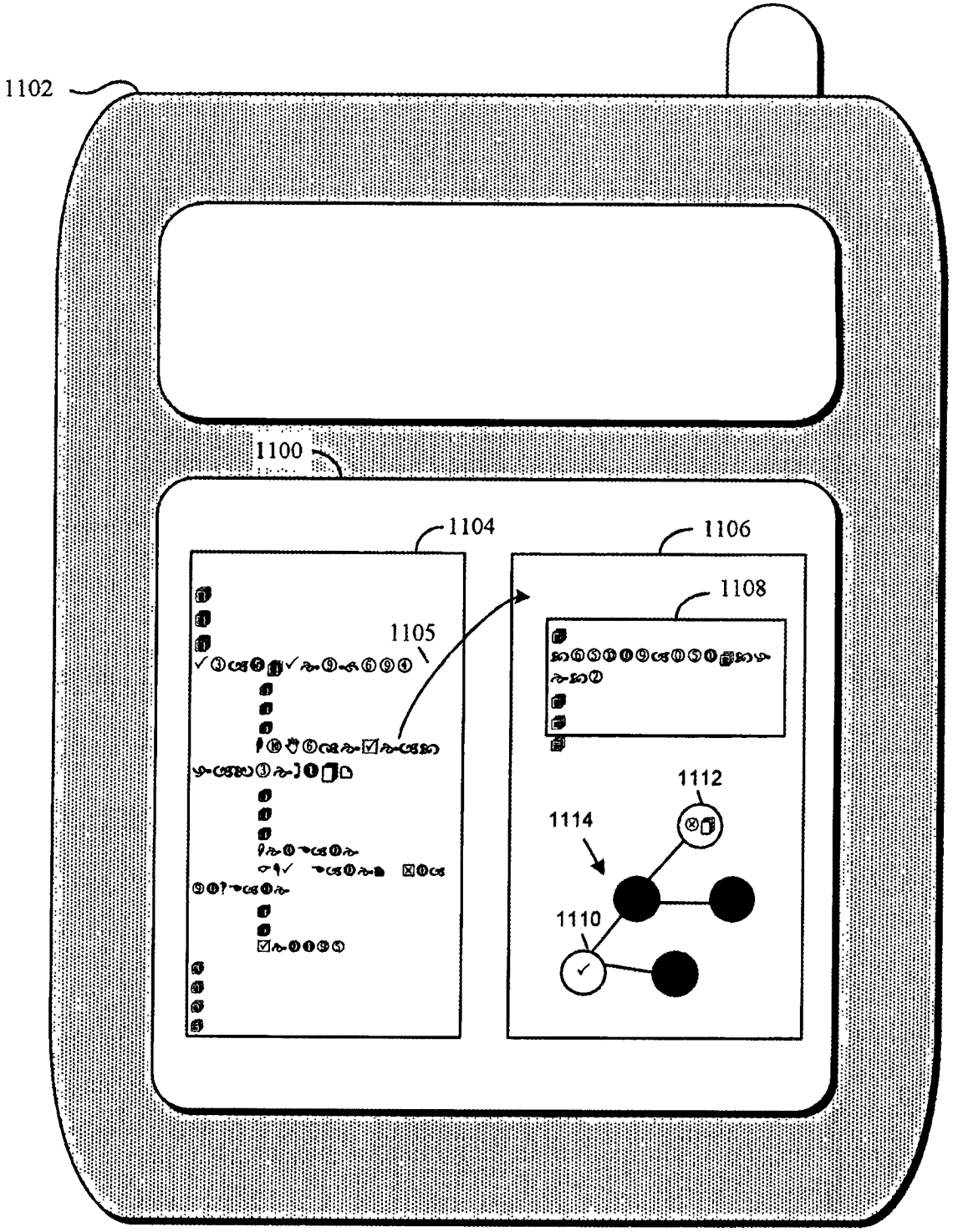


图 11

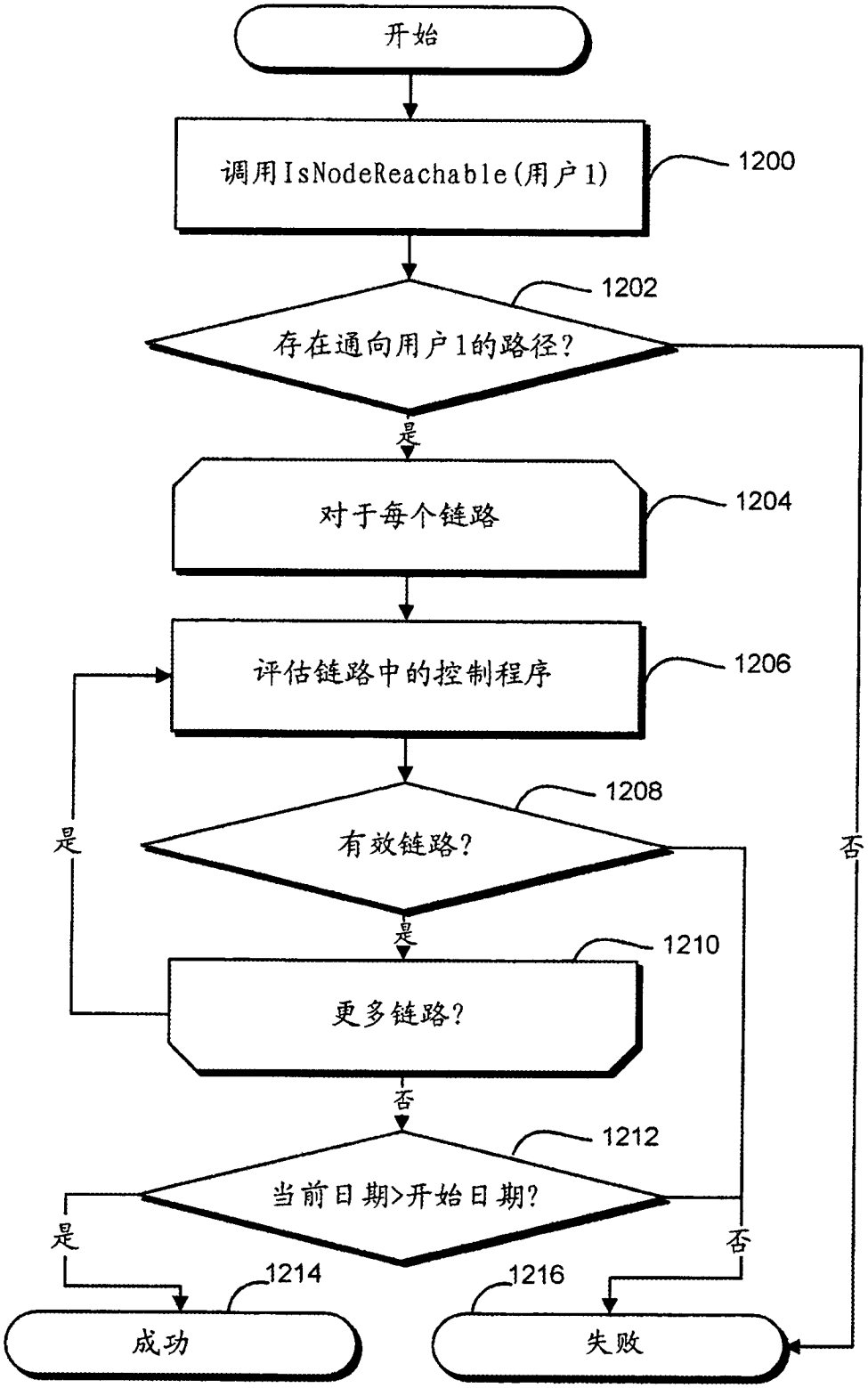

开始
调用IsNodeReachable(用户1)
1200
1202
存在通向用户1的路径?
是
否
对于每个链路
1204
评估链路中的控制程序
1206
1208
有效链路?
是
否
1210
更多链路?
否
是
1212
当前日期>开始日期?
是
否
1214
1216
成功
失败

图 12

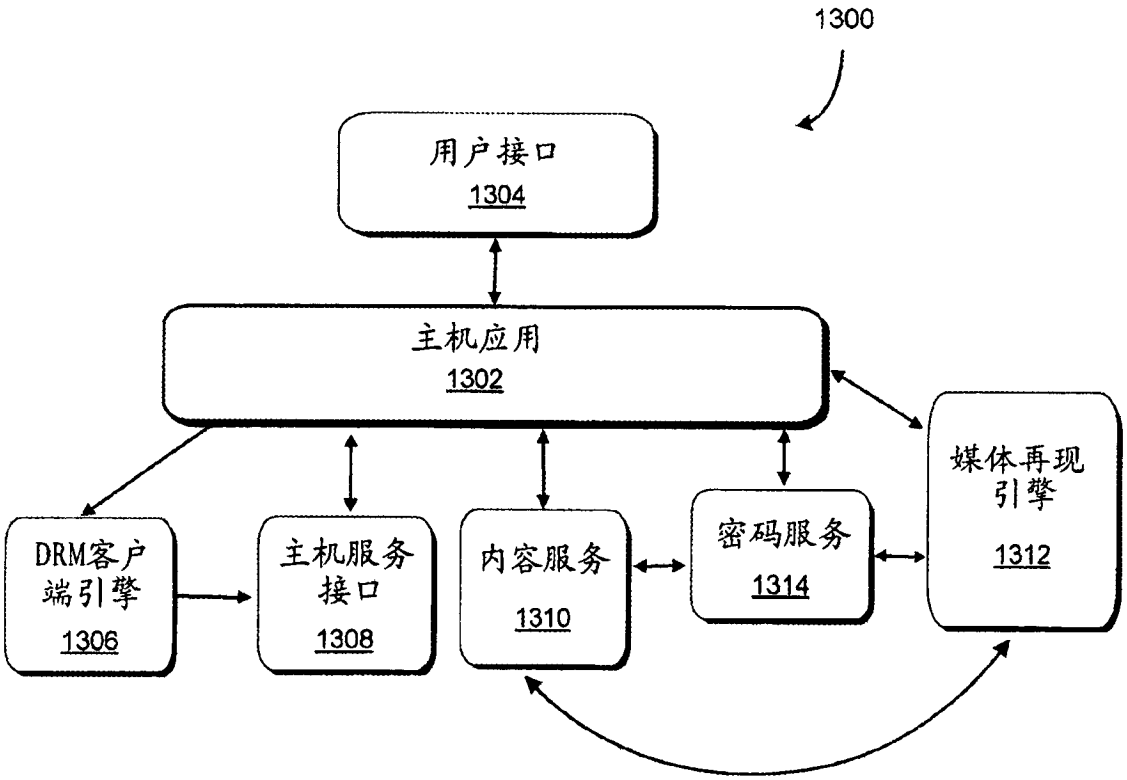

1300
用户接口
1304
主机应用
1302
DRM客户端引擎
1306
主机服务接口
1308
内容服务
1310
密码服务
1314
媒体再现引擎
1312

图 13

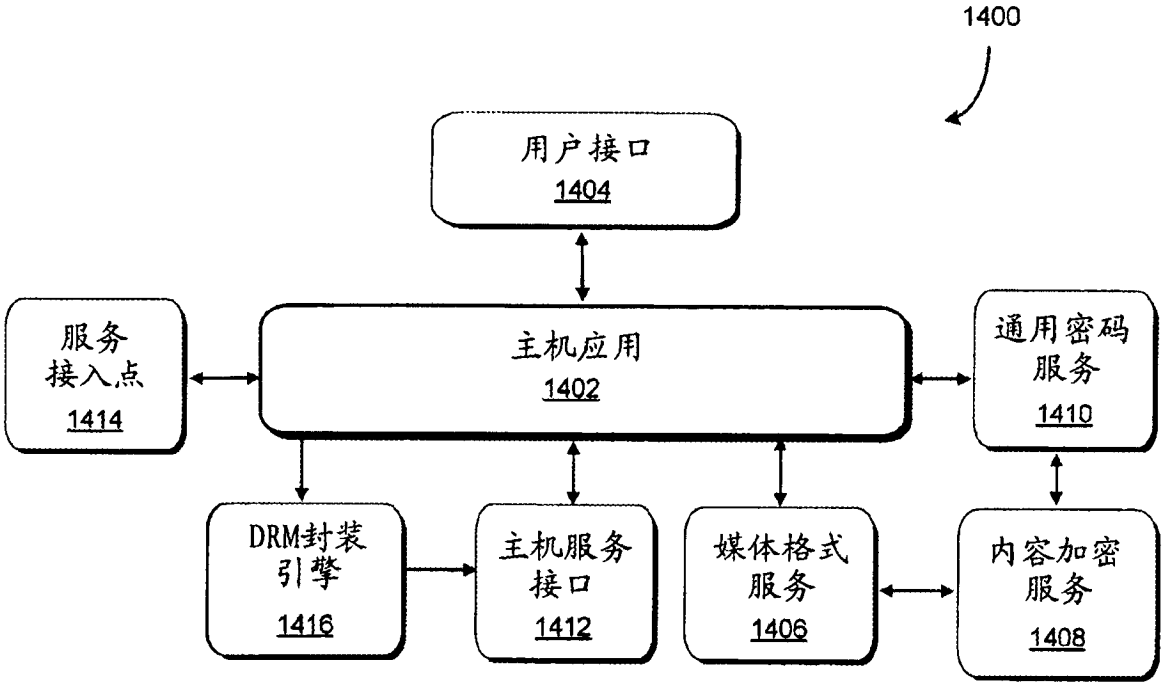

1400
用户接口
1404
服务接入点
1414
主机应用
1402
通用密码服务
1410
DRM封装引擎
1416
主机服务接口
1412
媒体格式服务
1406
内容加密服务
1408

图 14

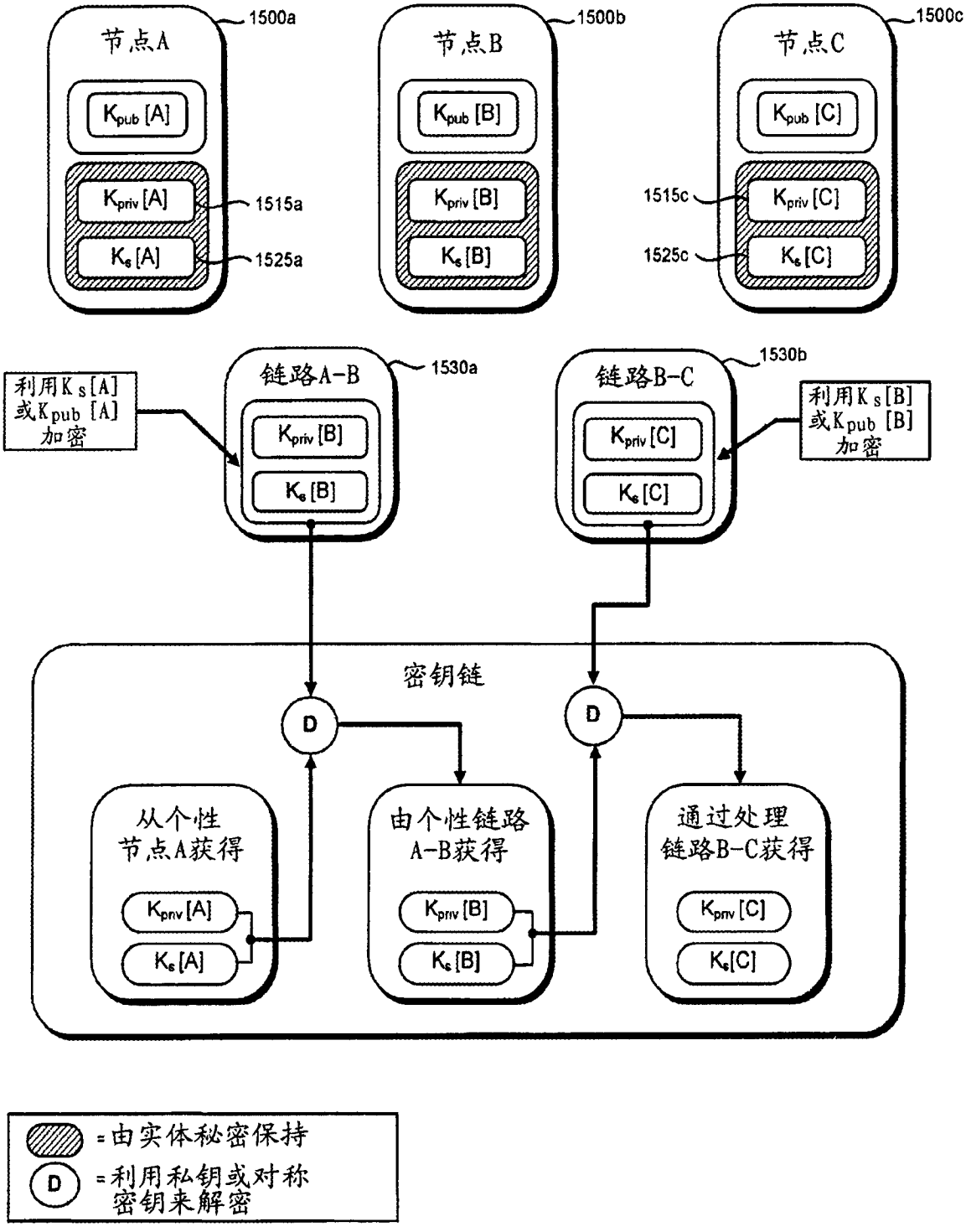

节点A
1500a
K_{pub} [A]
K_{priv} [A]
K_s [A]
1515a
1525a
节点B
1500b
K_{pub} [B]
K_{priv} [B]
K_s [B]
节点C
1500c
K_{pub} [C]
1515c
K_{priv} [C]
1525c
K_s [C]
利用K_s[A]或K_{pub}[A]加密
链路A-B
1530a
K_{priv} [B]
K_s [B]
链路B-C
1530b
K_{priv} [C]
K_s [C]
利用K_s[B]或K_{pub}[B]加密
密钥链
D
D
从个性节点A获得
K_{priv} [A]
K_s [A]
由个性链路A-B获得
K_{priv} [B]
K_s [B]
通过处理链路B-C获得
K_{priv} [C]
K_s [C]
=由实体秘密保持
D =利用私钥或对称密钥来解密

图 15

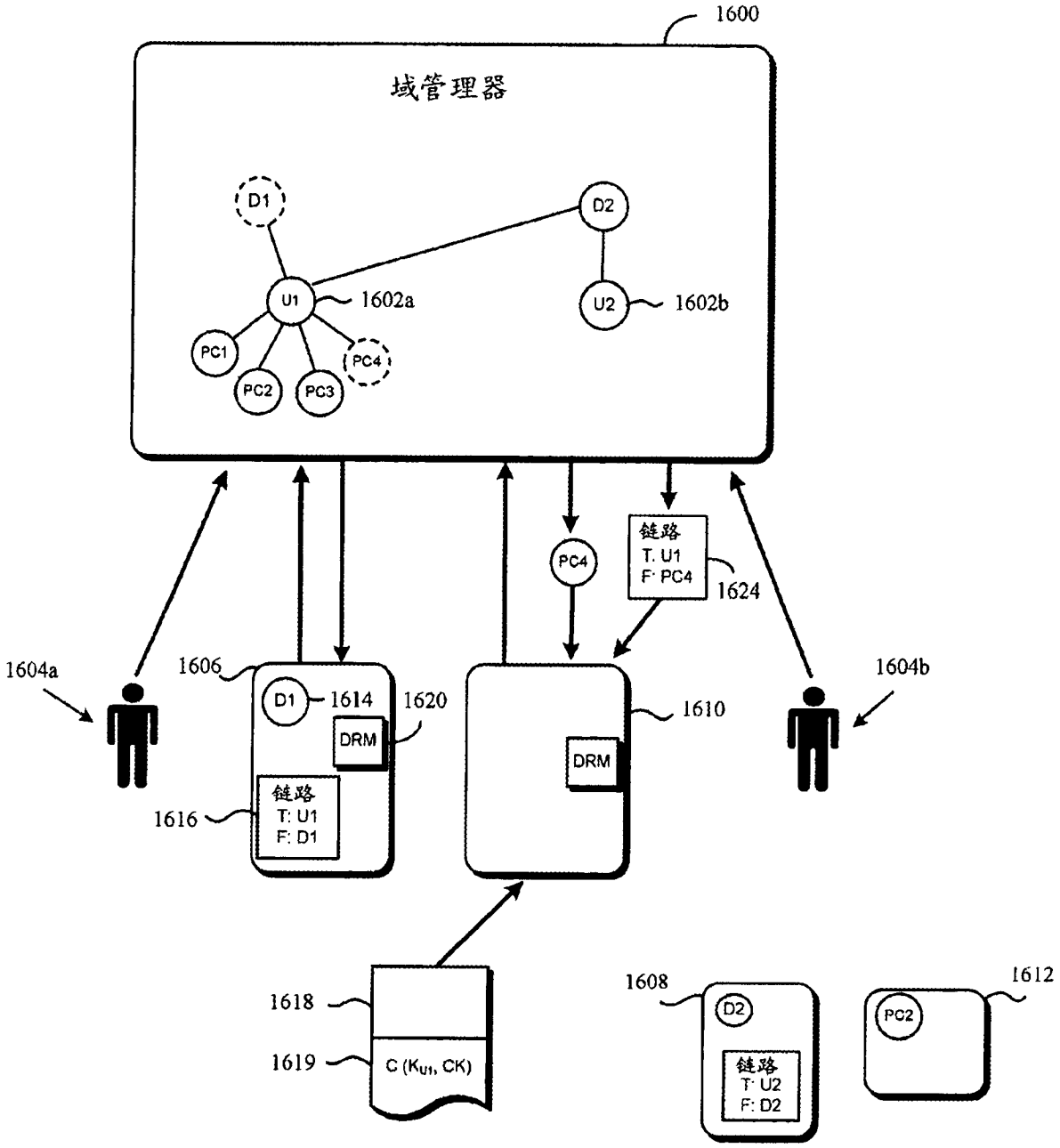

1600
域管理器
D1
D2
U1
1602a
U2
1602b
PC1
PC2
PC3
PC4
PC4
链路
T: U1
F: PC4
1624
1604a
1606
D1
1614
1620
DRM
链路
T: U1
F: D1
1616
1610
DRM
1604b
1618
1619
C (Ku1, CK)
1608
D2
链路
T: U2
F: D2
1612
PC2

图 16

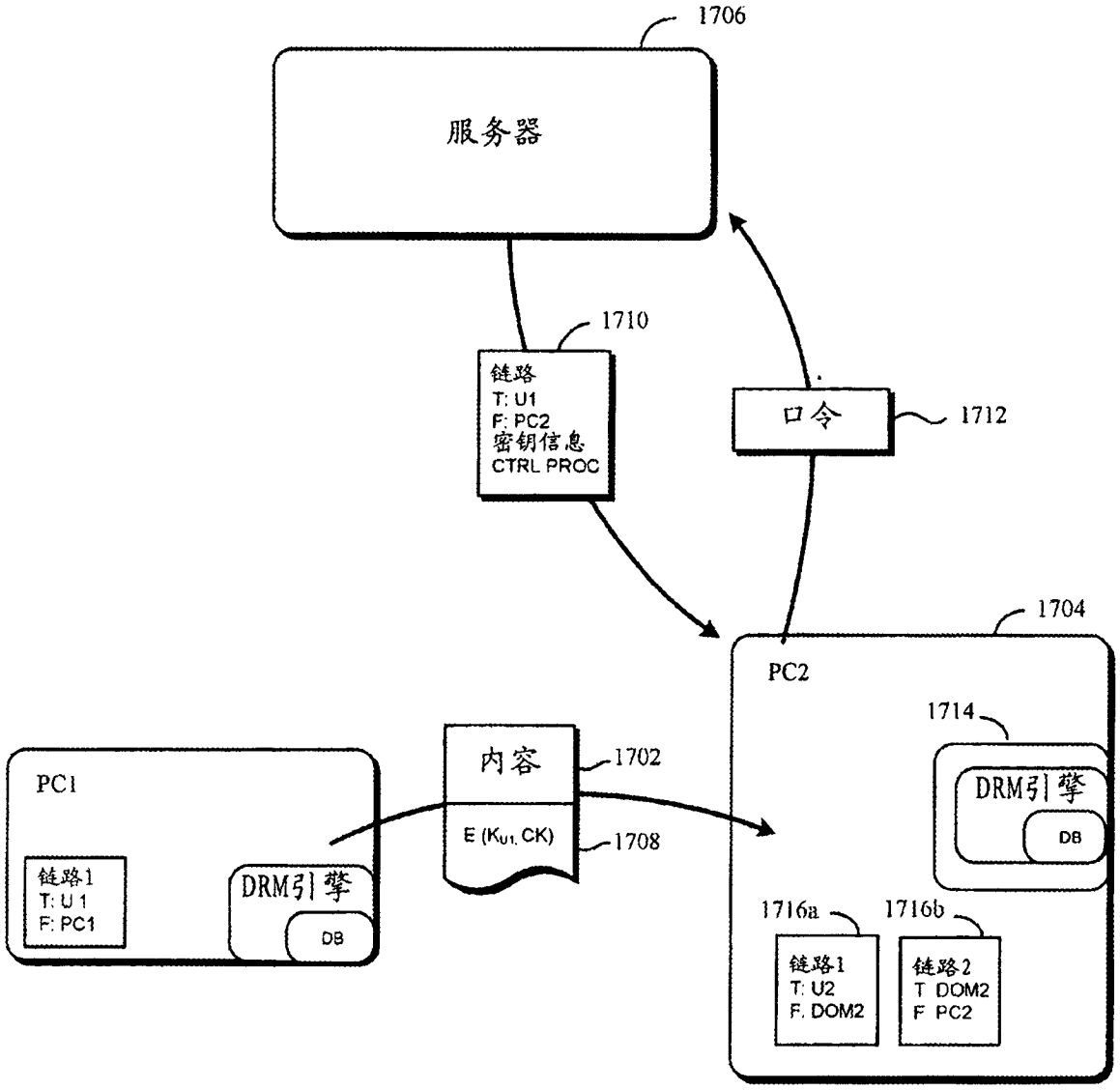

1706
服务器
1710
链路
T: U1
F: PC2
密钥信息
CTRL PROC
口令
1712
1704
PC2
1714
DRM引擎
DB
内容
1702
E (K_{U1}, CK)
1708
PC1
链路1
T: U 1
F: PC1
DRM引擎
DB
1716a
1716b
链路1
T: U2
F. DOM2
链路2
T DOM2
F PC2

图 17

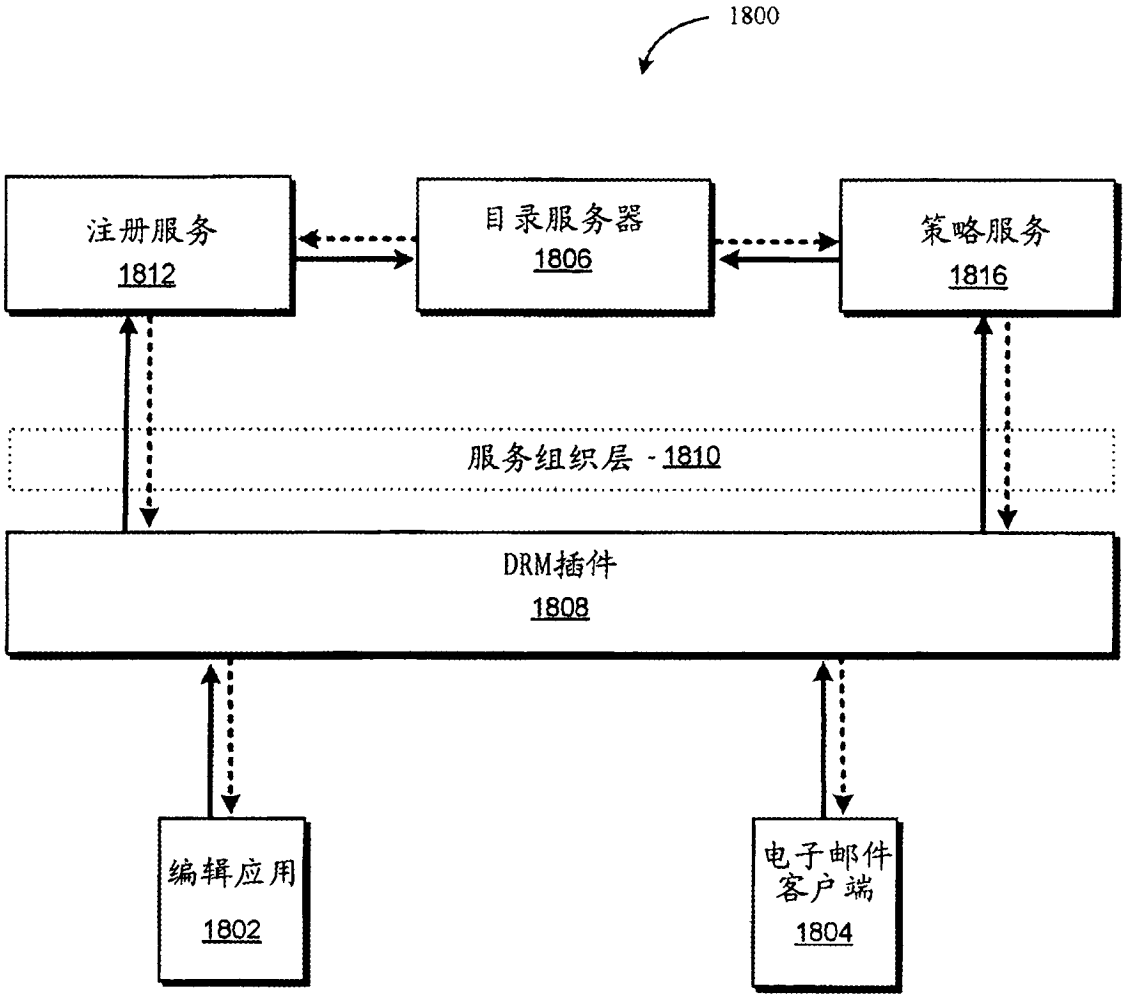

1800
注册服务
1812
目录服务器
1806
策略服务
1816
服务组织层 - 1810
DRM插件
1808
编辑应用
1802
电子邮件
客户端
1804

图 18

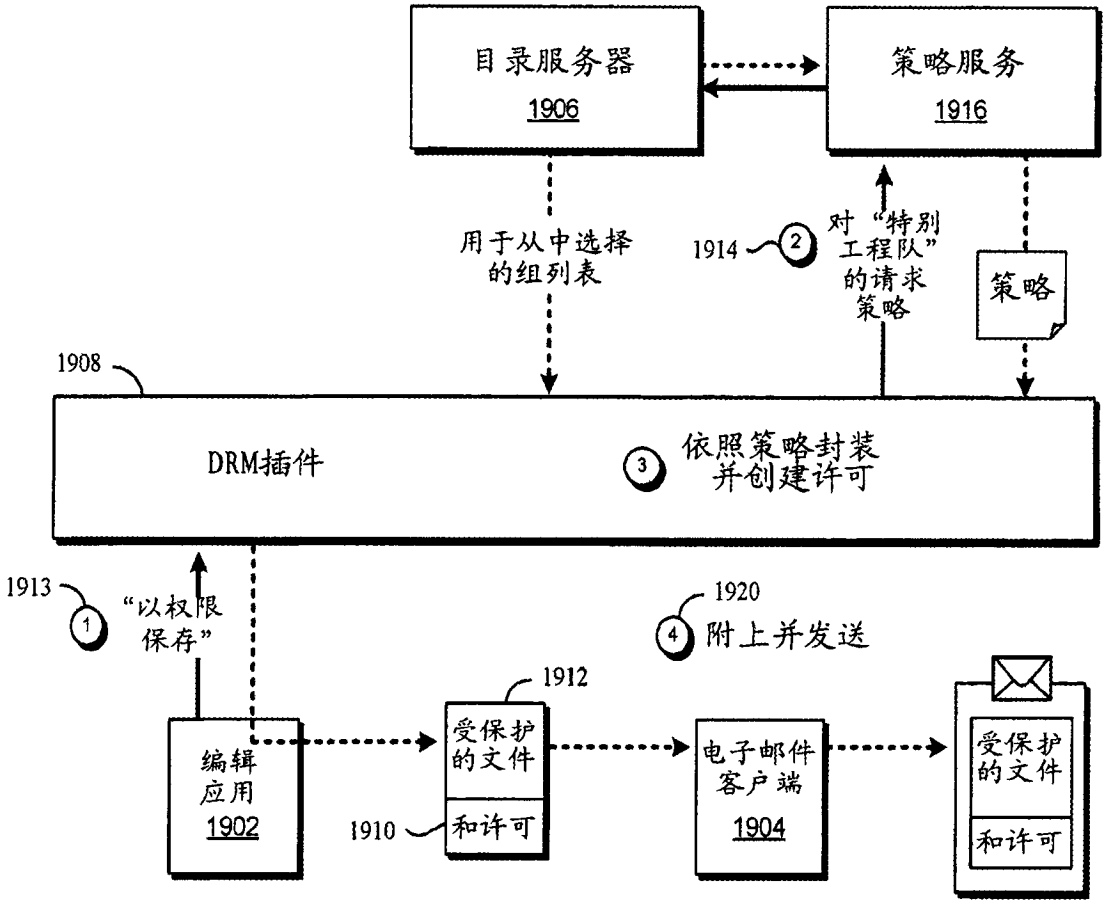

目录服务器
1906
策略服务
1916
用于从中选择
的组列表
1914
2
对“特别
工程队”
的请求
策略
策略
1908
DRM插件
3
依照策略封装
并创建许可
1913
1
“以权限
保存”
1920
4
附上并发送
1912
编辑
应用
1902
受保护
的文件
1910
和许可
电子邮件
客户端
1904
受保护
的文件
和许可

图 19

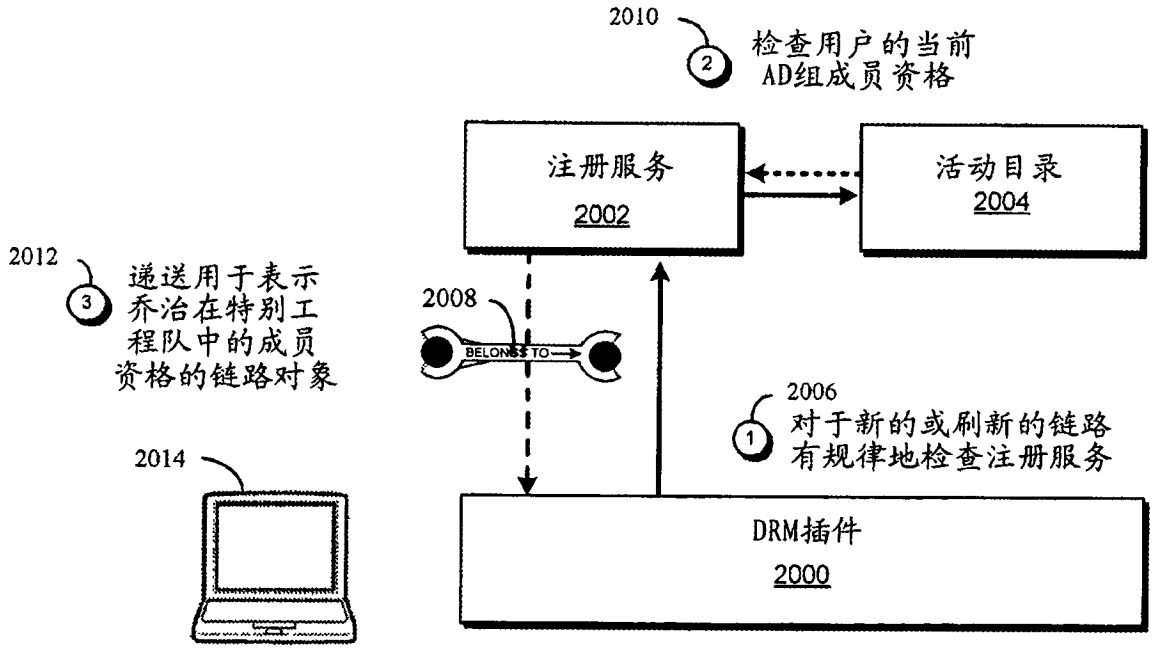

2010
2
检查用户的当前
AD组成员资格
注册服务
2002
活动目录
2004
2012
3
递送用于表示
乔治在特别工
程队中的成员
资格的链路对象
2008
BELONGS TO
2006
1
对于新的或刷新的链路
有规律地检查注册服务
2014
DRM插件
2000

图 20

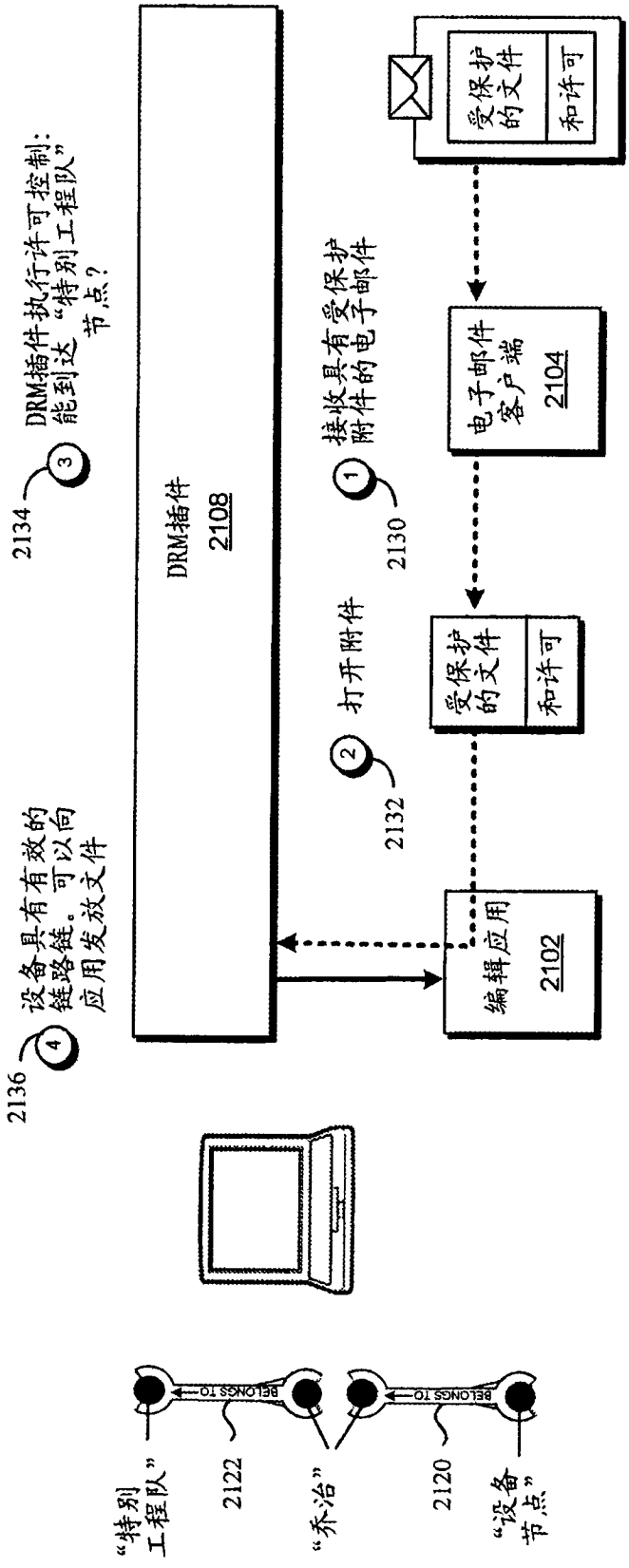

DRM插件
2108
编辑应用
2102
受保护的文件
和许可
电子邮件客户端
2104
受保护的文件
和许可
接收具有受保护附件的电子邮件
1
2130
打开附件
2
2132
DRM插件执行许可控制：能到达“特别工程队”节点？
3
2134
设备具有有效的链路链。可以向应用发放文件
4
2136
“特别工程队”
2122
“乔治”
2120
“设备节点”
BELONGS TO
BELONGS TO

图 21

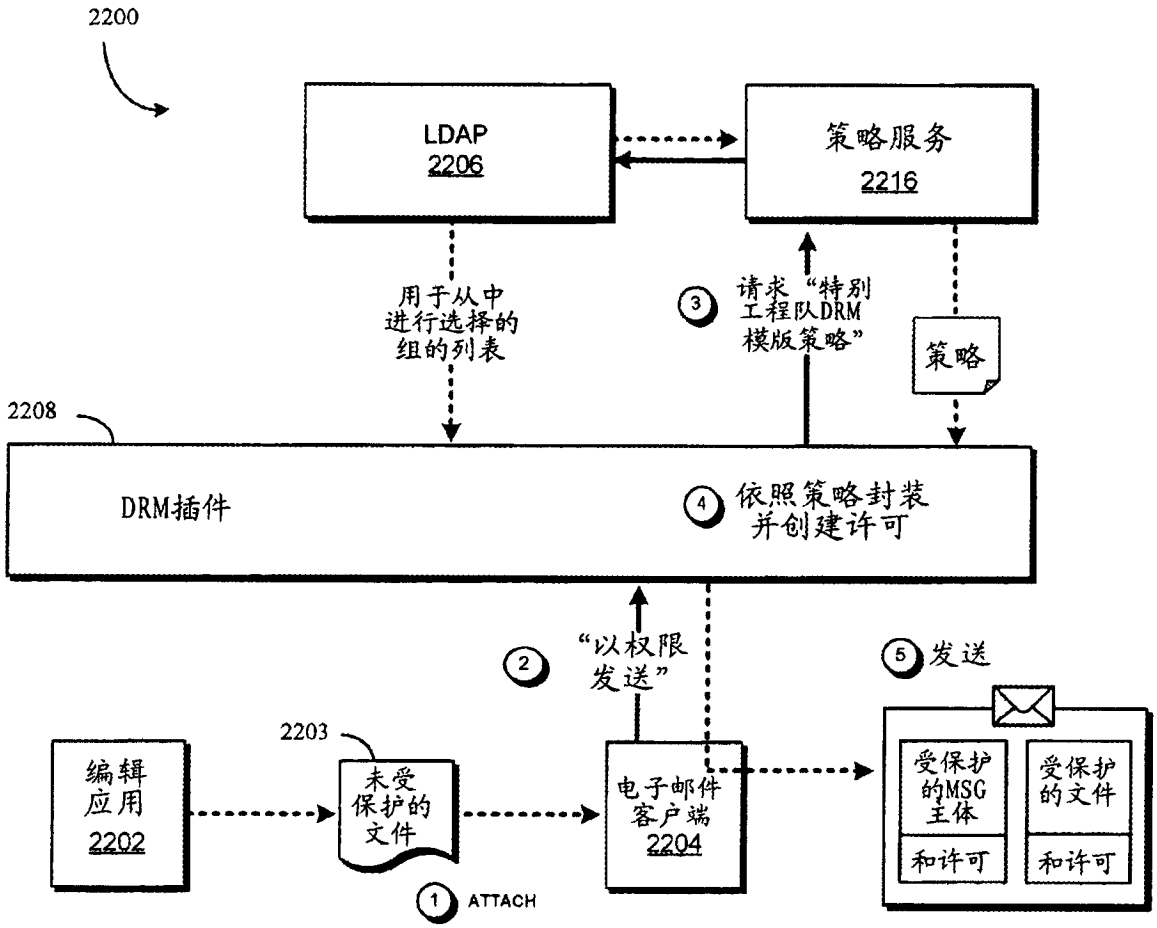

2200
LDAP
2206
策略服务
2216
用于从中
进行选择的
组的列表
3
请求“特别
工程队DRM
模版策略”
策略
2208
DRM插件
4
依照策略封装
并创建许可
2
“以权限
发送”
5
发送
2203
编辑
应用
2202
未受
保护的
文件
电子邮件
客户端
2204
受保护
的MSG
主体
受保护
的文件
和许可
和许可
1
ATTACH

图 22

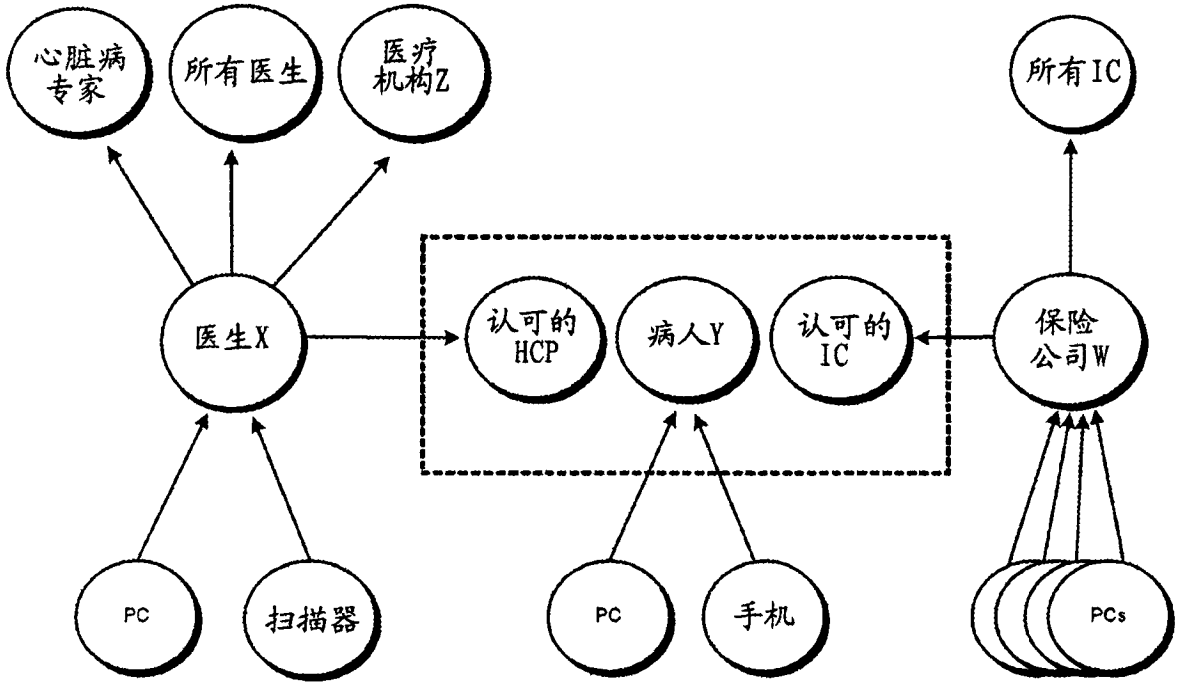

心脏病专家
所有医生
医疗机构Z
所有IC
医生X
认可的HCP
病人Y
认可的IC
保险公司W
PC
扫描器
PC
手机
PCs

图 23

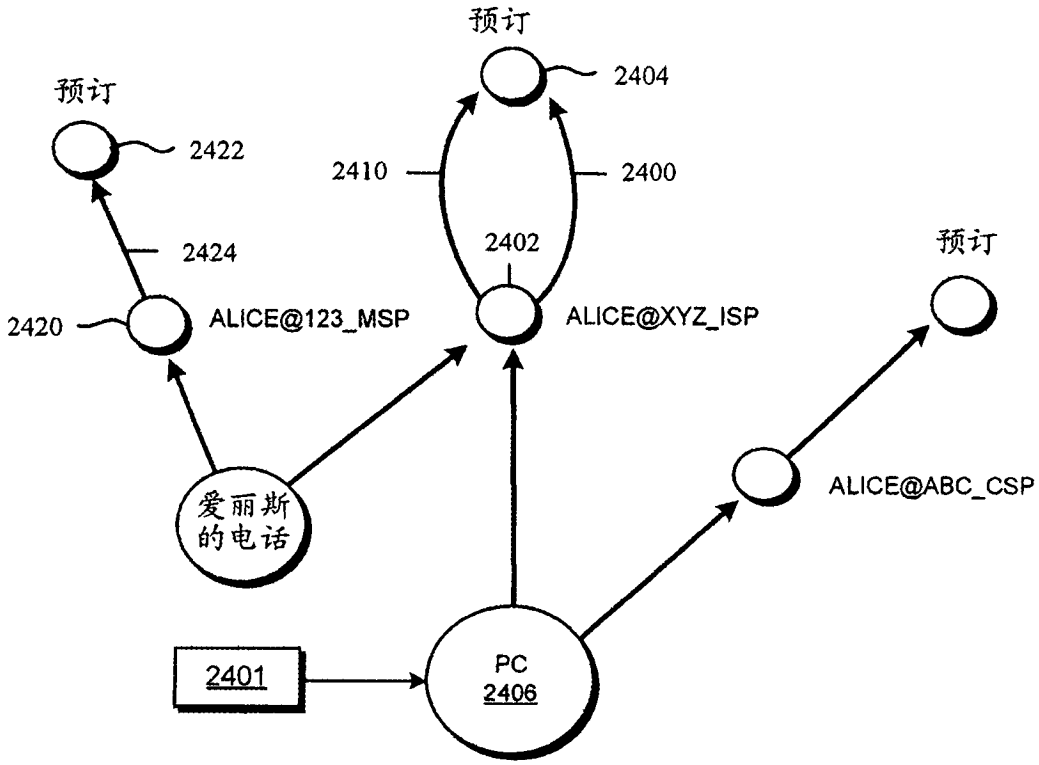

预订
2404
预订
2422
2410
2400
2424
2402
预订
2420
ALICE@123_MSP
ALICE@XYZ_ISP
爱丽斯的电话
ALICE@ABC_CSP
2401
PC
2406

图 24

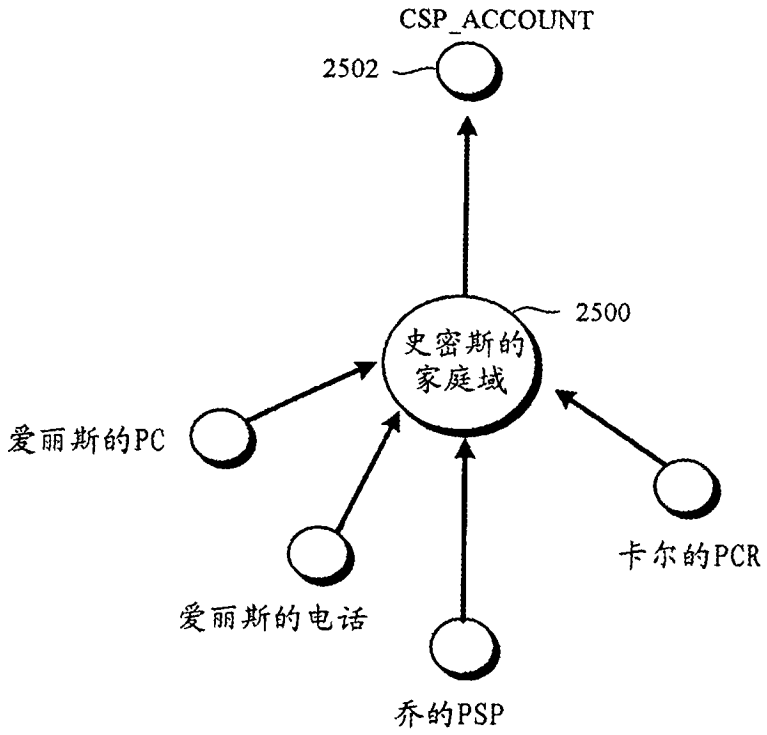


图 25

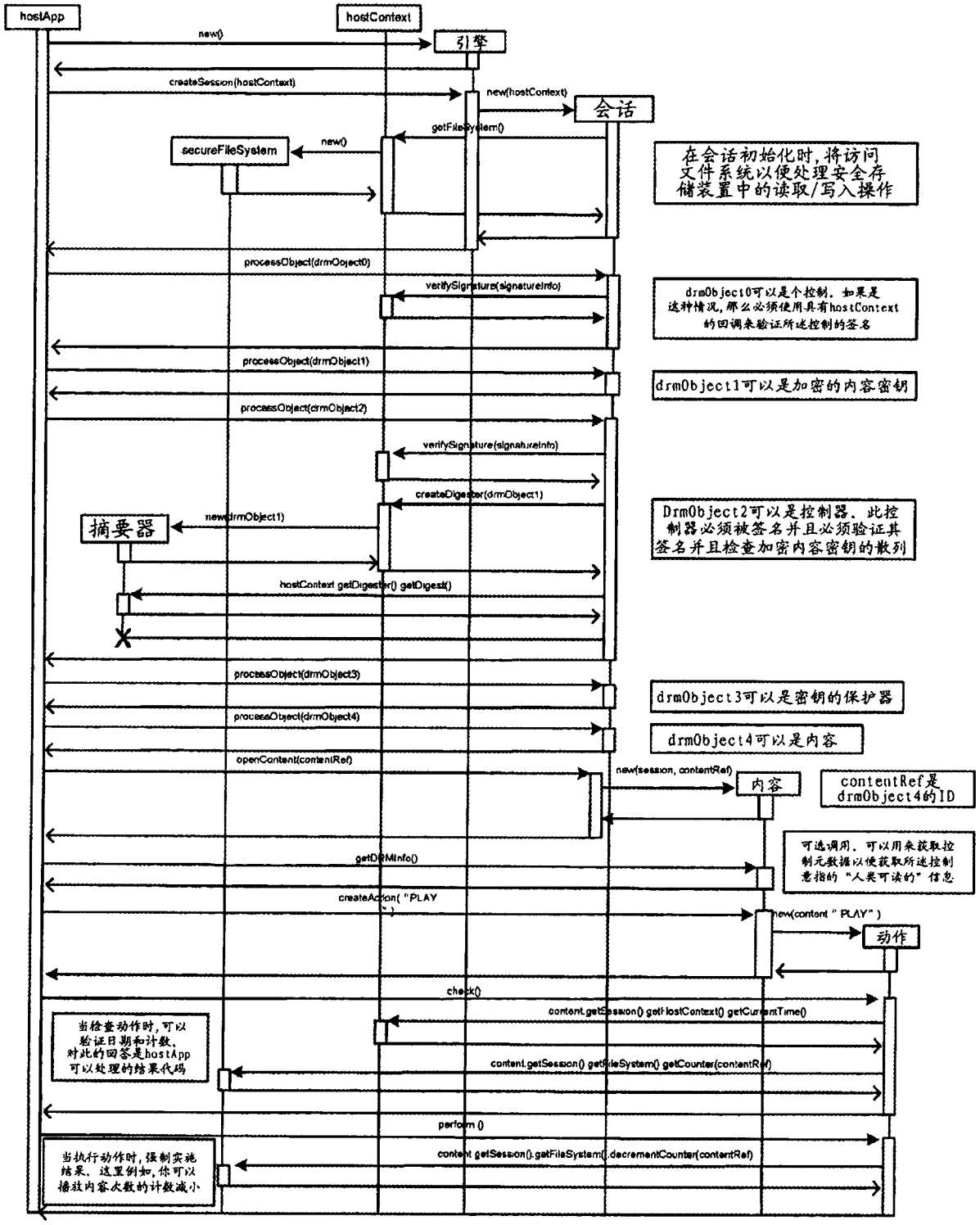

hostApp
hostContext
引擎
new()
createSession(hostContext)
new(hostContext)
会话
getFileSystem()
secureFileSystem
new()
在会话初始化时，将访问文件系统以便处理安全存储装置中的读取/写入操作
processObject(drmObject0)
verifySignature(signatureInfo)
drmObject0可以是个控制，如果是这种情况，那么必须使用具有hostContext的回调来验证所述控制的签名
processObject(drmObject1)
drmObject1可以是加密的内容密钥
processObject(drmObject2)
verifySignature(signatureInfo)
createDigester(drmObject1)
摘要器
new(drmObject1)
DrmObject2可以是控制器，此控制器必须被签名并且必须验证其签名并且检查加密内容密钥的散列
hostContext.getDigester().getDigest()
processObject(drmObject3)
drmObject3可以是密钥的保护器
processObject(drmObject4)
drmObject4可以是内容
openContent(contentRef)
new(session, contentRef)
内容
contentRef是drmObject4的ID
getDRMInfo()
可选调用，可以用来获取控制元数据以便获取所述控制意指的"人类可读的"信息
createAction("PLAY")
new(content, "PLAY")
动作
check()
content.getSession().getHostContext().getCurrentTime()
当检查动作时，可以验证日期和计数，对此的回答是hostApp可以处理的结果代码
content.getSession().getFileSystem().getCounter(contentRef)
perform()
content.getSession().getFileSystem().decrementCounter(contentRef)
当执行动作时，强制实施结果，这里例如，你可以播放内容次数的计数减小

图 26

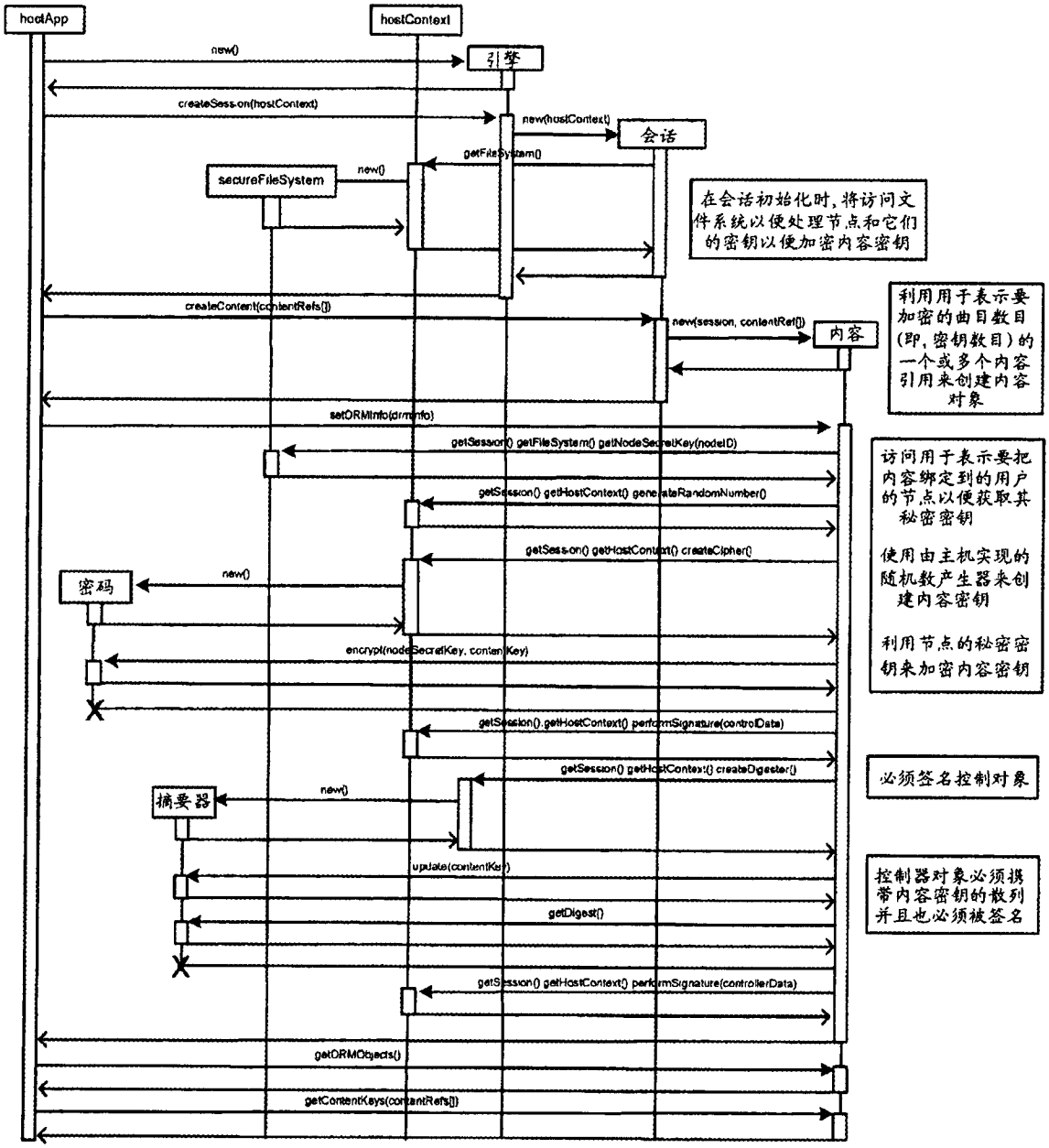

hostApp
hostContext
new()
引擎
createSession(hostContext)
new(hostContext)
会话
getFileSystem()
secureFileSystem
new()
在会话初始化时，将访问文件系统以便处理节点和它们的密钥以便加密内容密钥
createContent(contentRefs[])
new(session, contentRef[])
内容
利用用于表示要加密的由目数目(即，密钥数目)的一个或多个内容引用来创建内容对象
setORMInfo(ormInfo)
getSession() getFileSystem() getNodeSecretKey(nodeID)
访问用于表示要把内容绑定到的用户的节点以便获取其秘密密钥
getSession() getHostContext() generateRandomNumber()
使用由主机实现的随机数产生器来创建内容密钥
getSession() getHostContext() createCipher()
密码
new()
encrypt(nodeSecretKey, contentKey)
利用节点的秘密密钥来加密内容密钥
getSession() getHostContext() performSignature(controlData)
必须签名控制对象
getSession() getHostContext() createDigester()
摘要器
new()
update(contentKey)
控制器对象必须携带内容密钥的散列并且也必须被签名
getDigest()
getSession() getHostContext() performSignature(controllerData)
getORMObjects()
getContentKeys(contentRefs[])

图 27

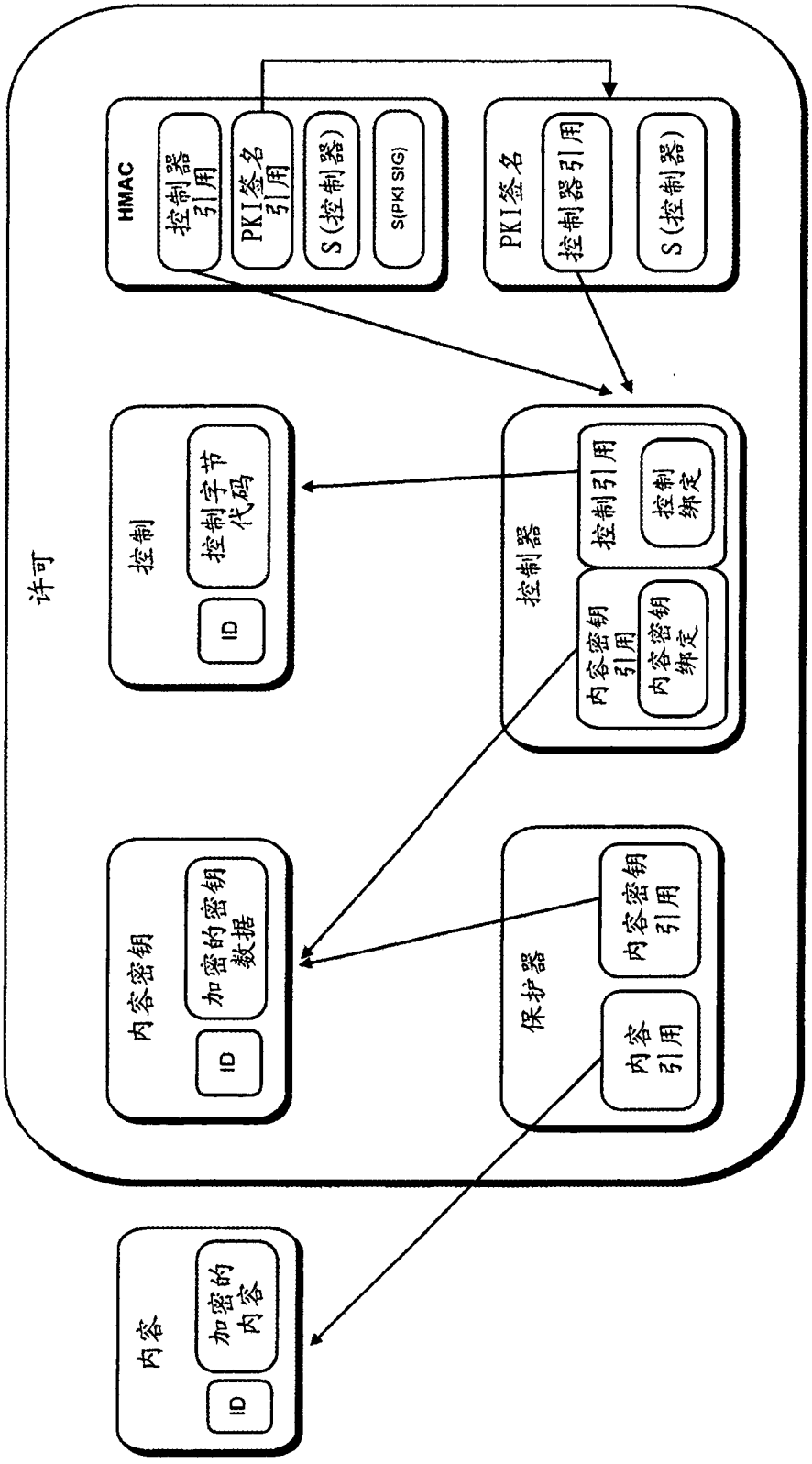

许可
HMAC
控制器引用
PKI签名引用
S(控制器)
S(PKI SIG)
PKI签名
控制器引用
S(控制器)
控制
ID
控制字节代码
控制器
控制引用
控制绑定
内容密钥引用
内容密钥绑定
内容密钥
ID
加密的密钥数据
保护器
内容密钥引用
内容引用
内容
ID
加密的内容

图 28A

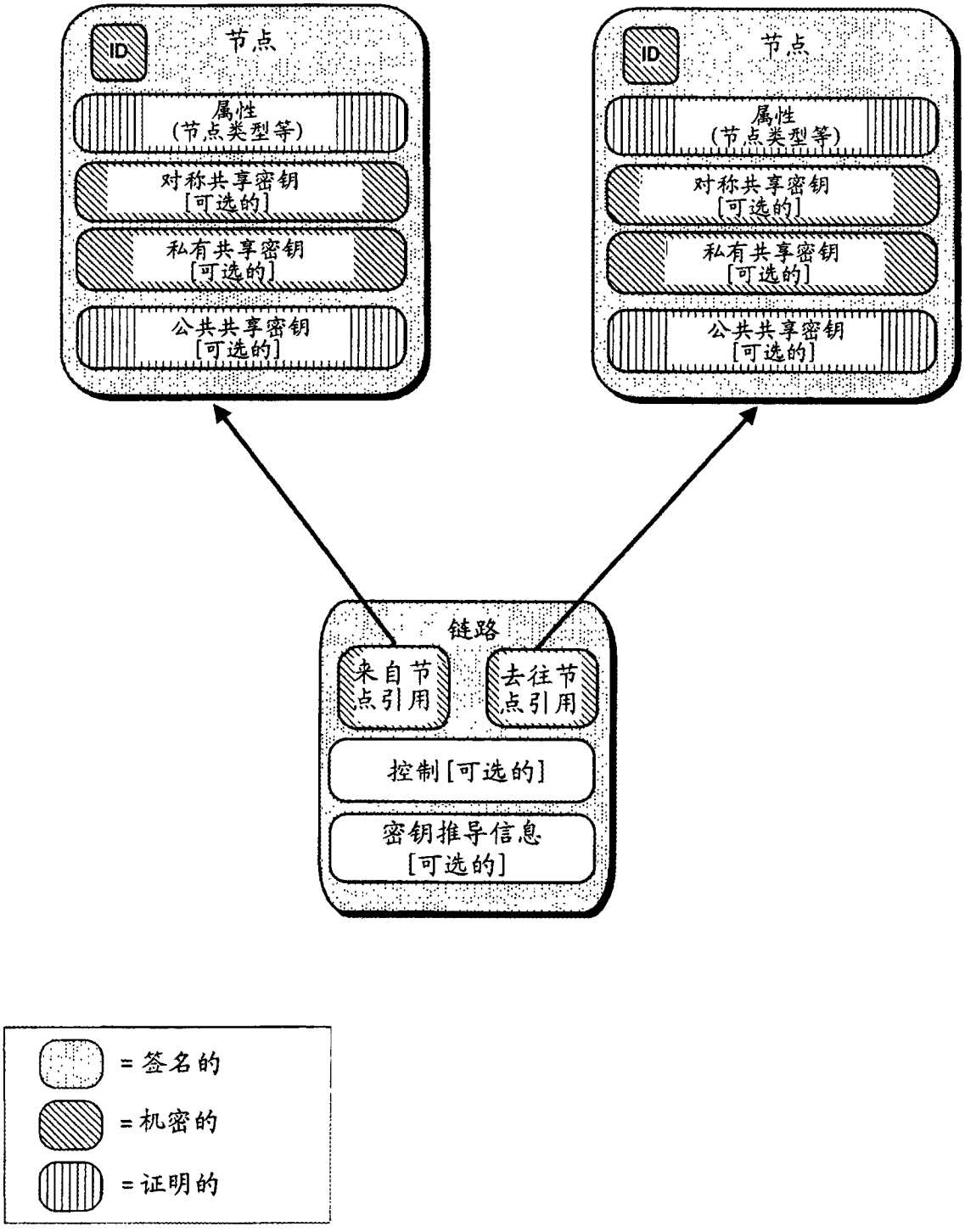

ID
节点
属性
(节点类型等)
对称共享密钥
[可选的]
私有共享密钥
[可选的]
公共共享密钥
[可选的]
ID
节点
属性
(节点类型等)
对称共享密钥
[可选的]
私有共享密钥
[可选的]
公共共享密钥
[可选的]
链路
来自节点引用
去往节点引用
控制[可选的]
密钥推导信息
[可选的]
= 签名的
= 机密的
= 证明的

图 28B

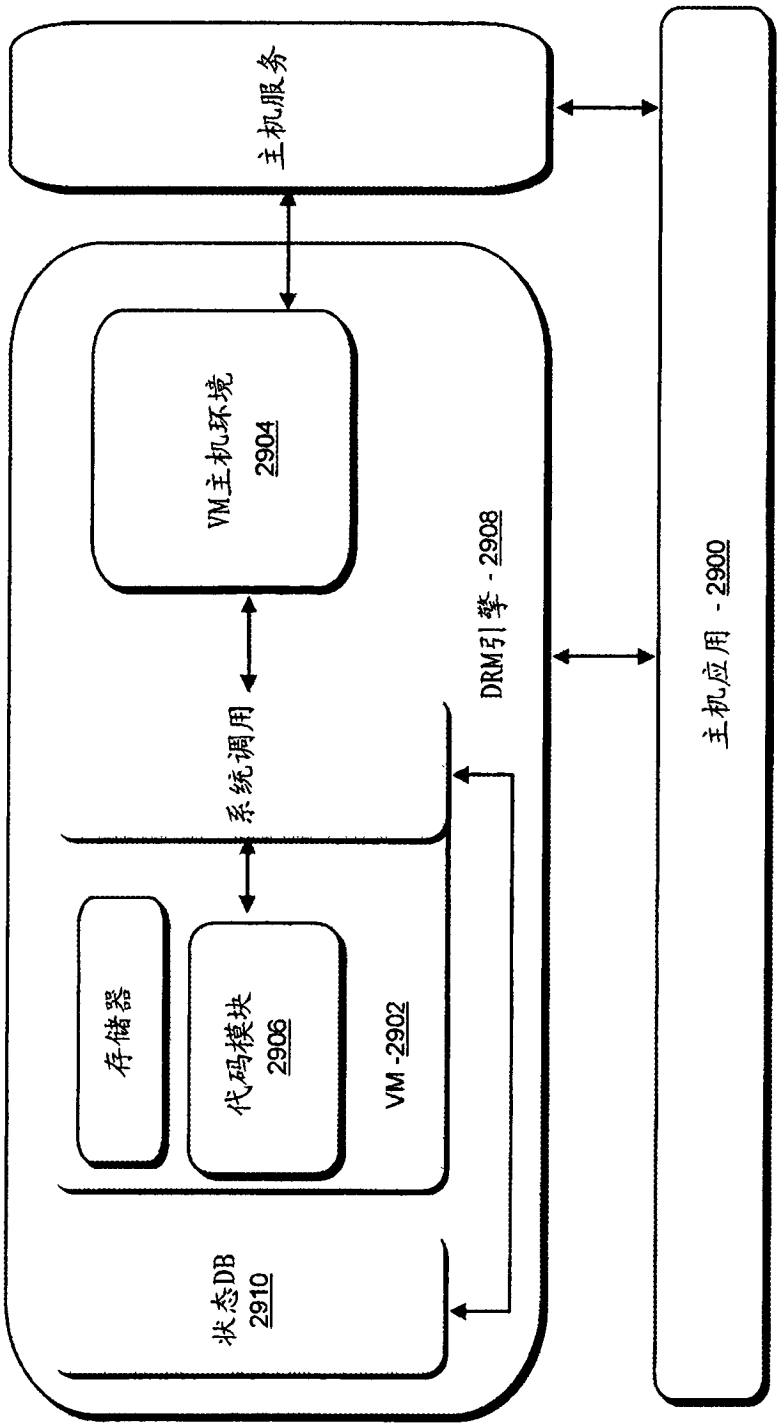

主机服务
VM主机环境
2904
系统调用
DRM引擎 - 2908
主机应用 - 2900
存储器
代码模块
2906
VM -2902
状态DB
2910

图 29

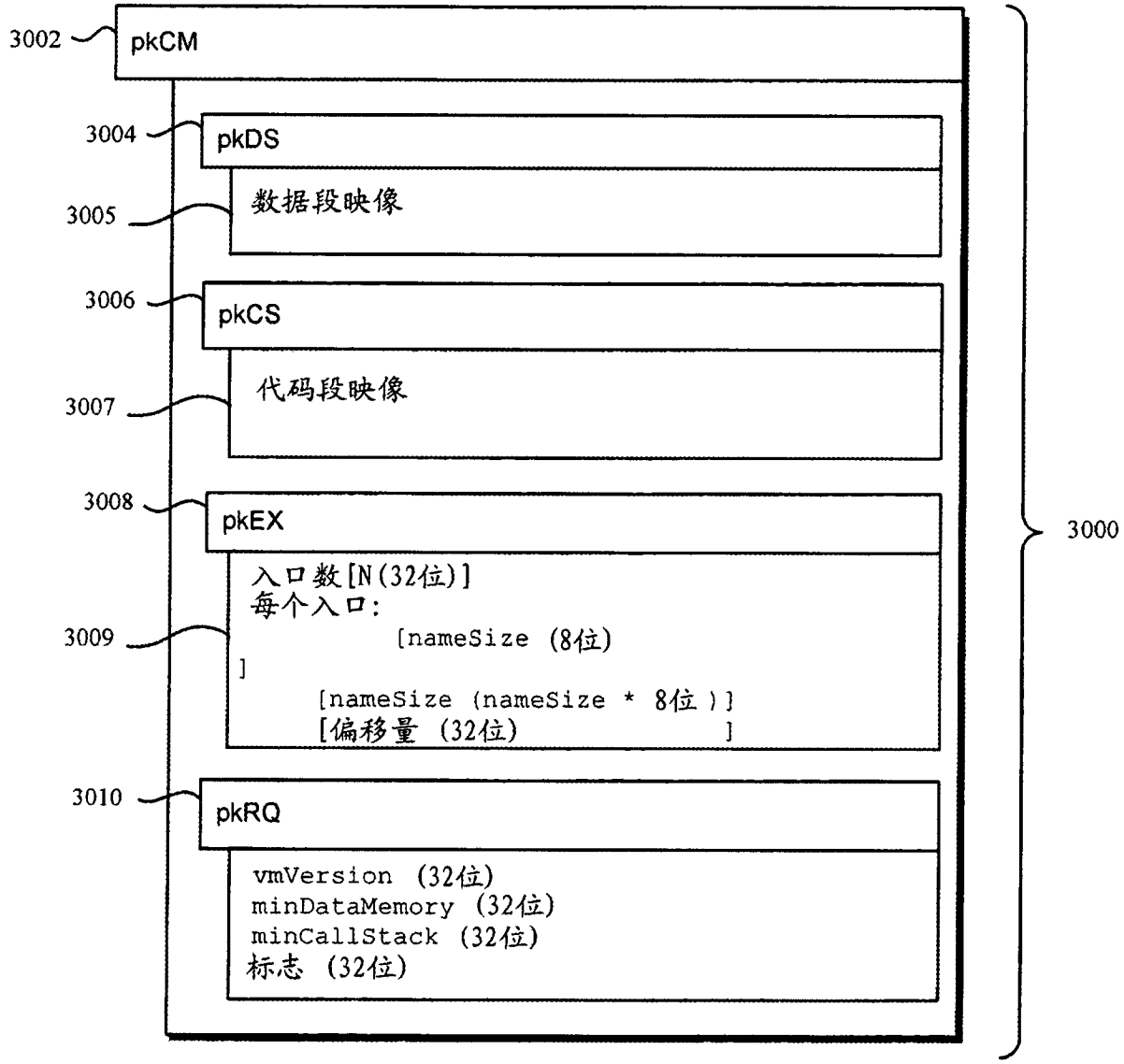


图 30

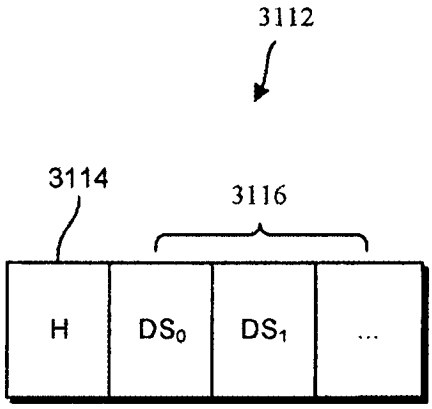


图 31A

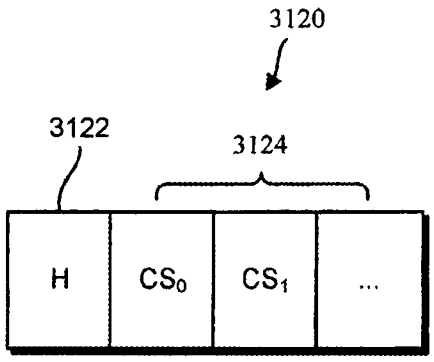


图 31B

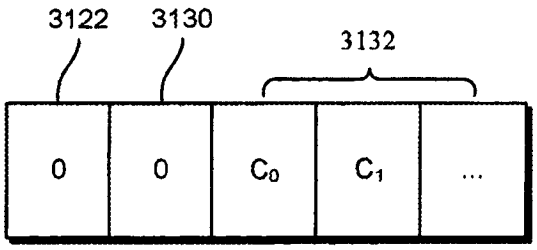


图 31C

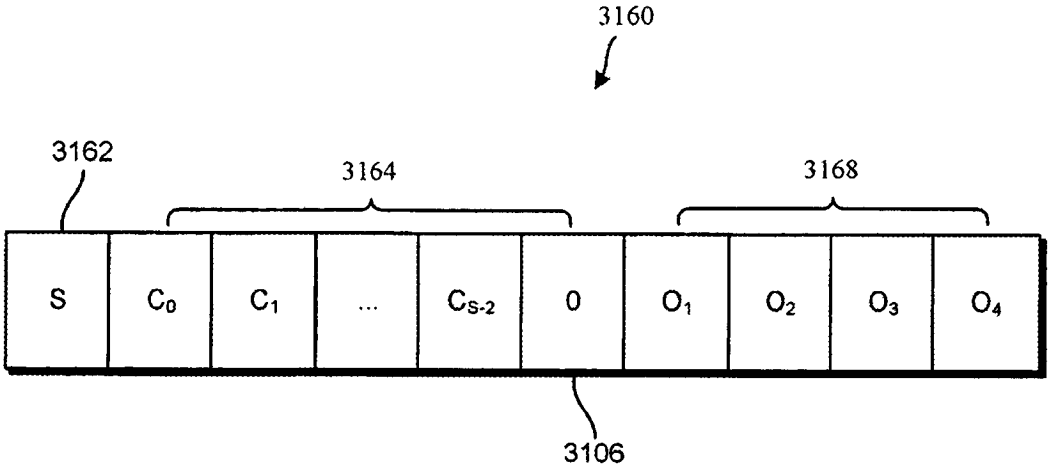


图 31D

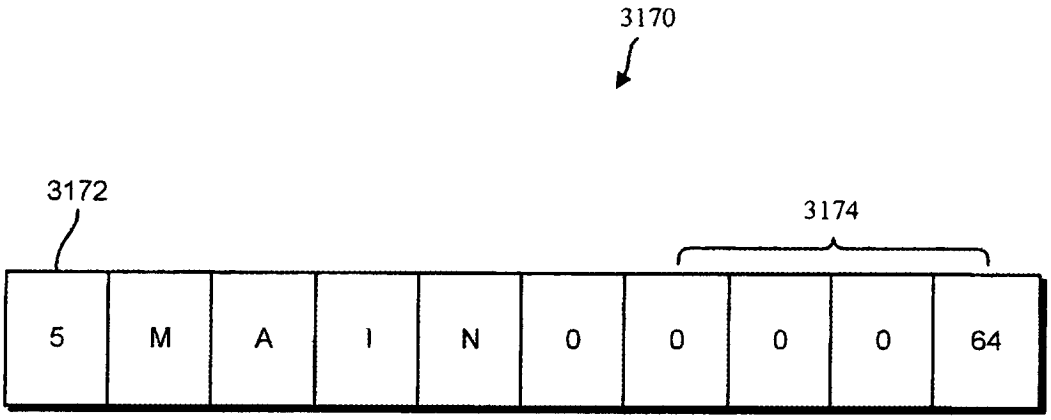


图 31E

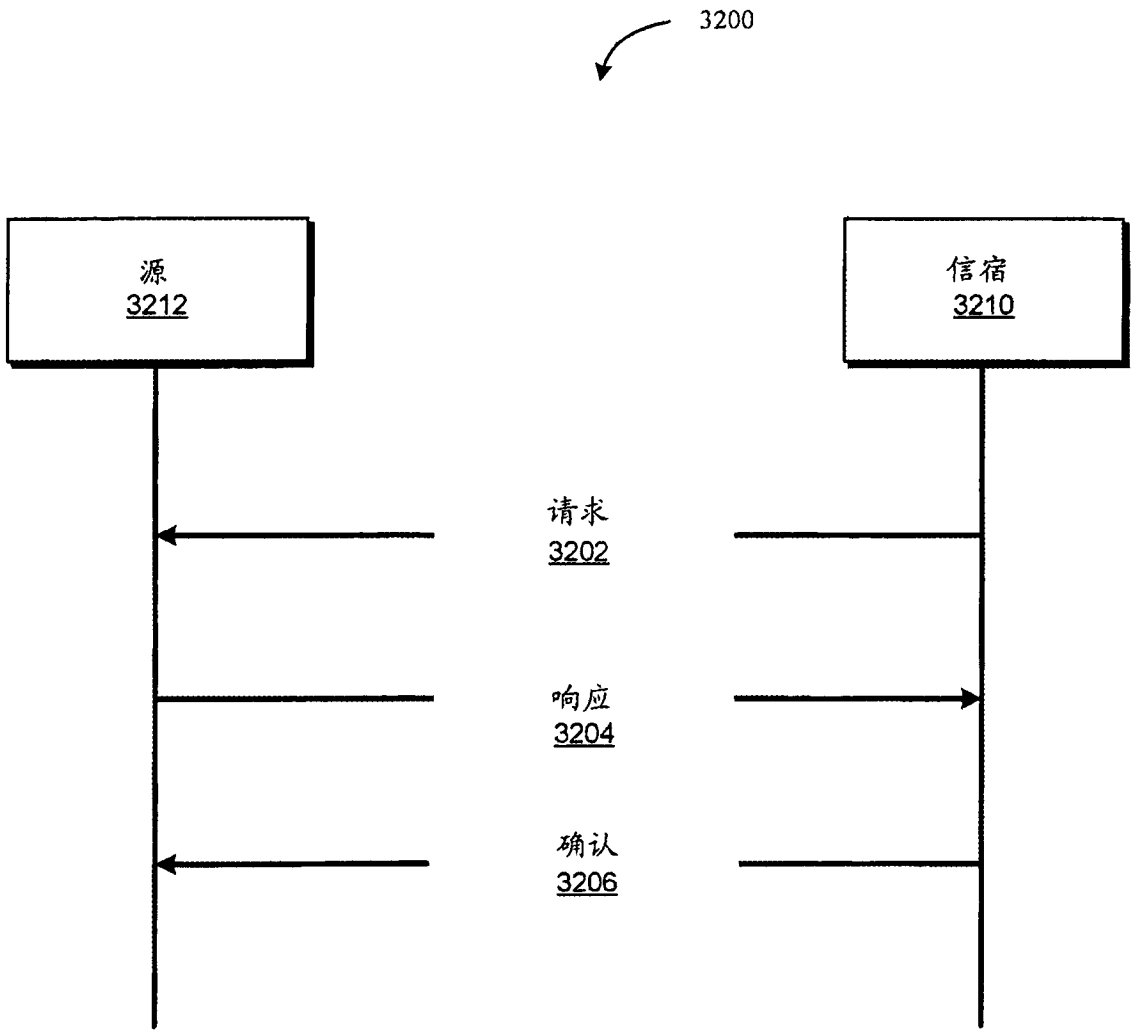


图 32

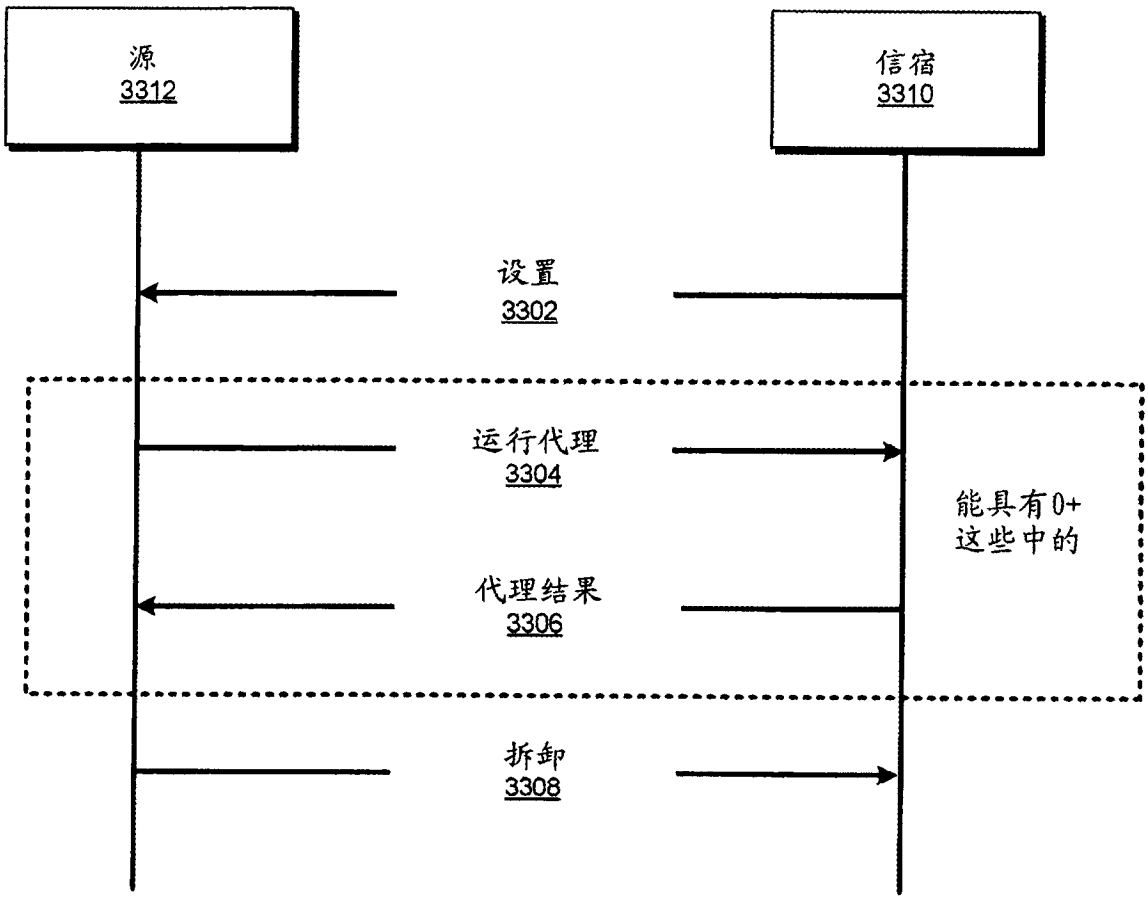


图 33

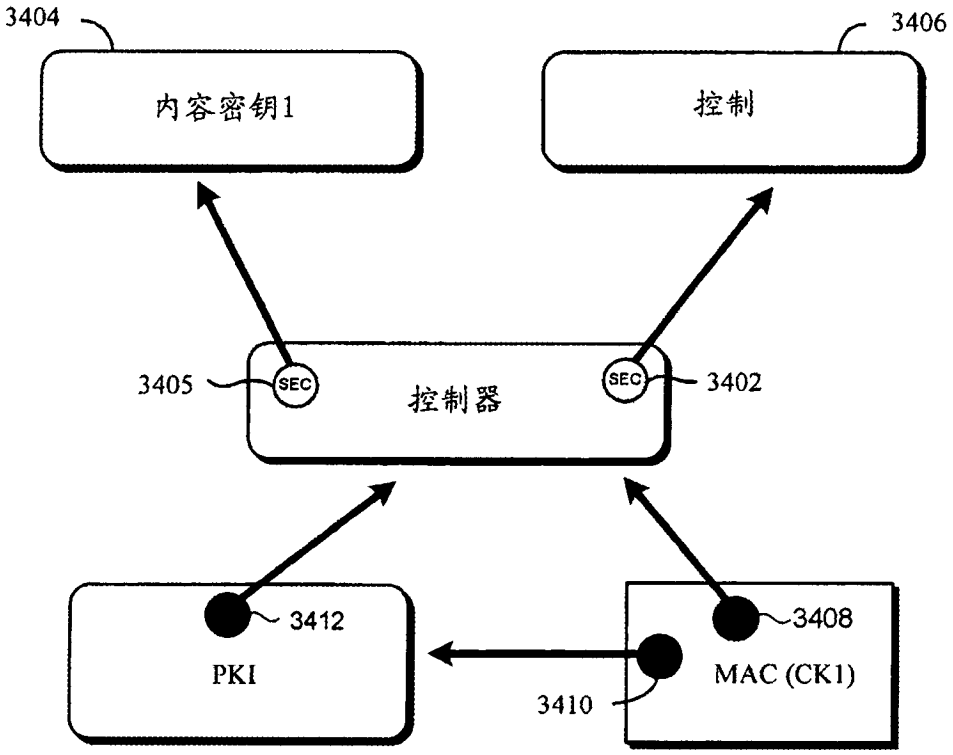

3404
内容密钥1
3406
控制
3405
SEC
控制器
SEC
3402
3412
PKI
3408
MAC (CK1)
3410

图 34

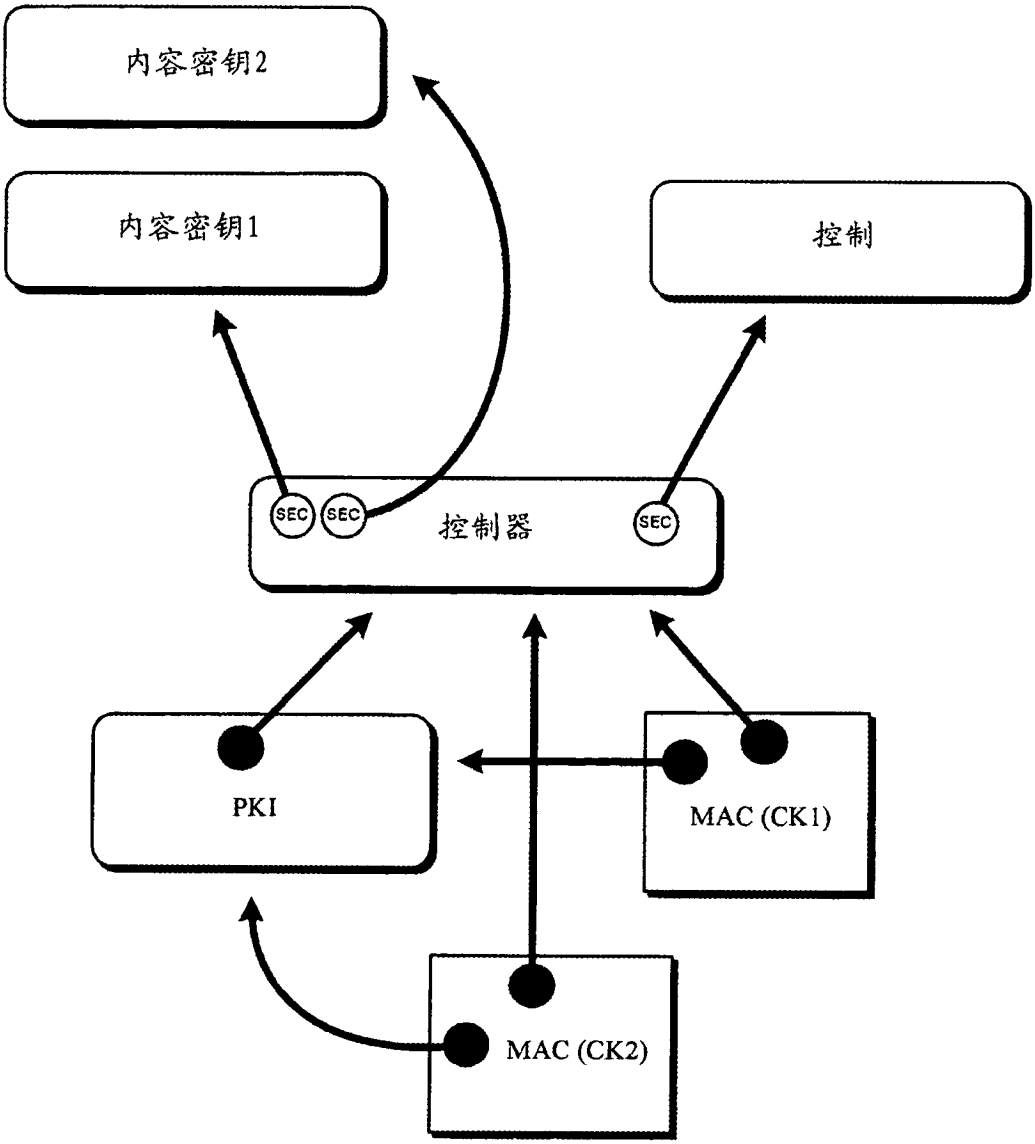


图 35

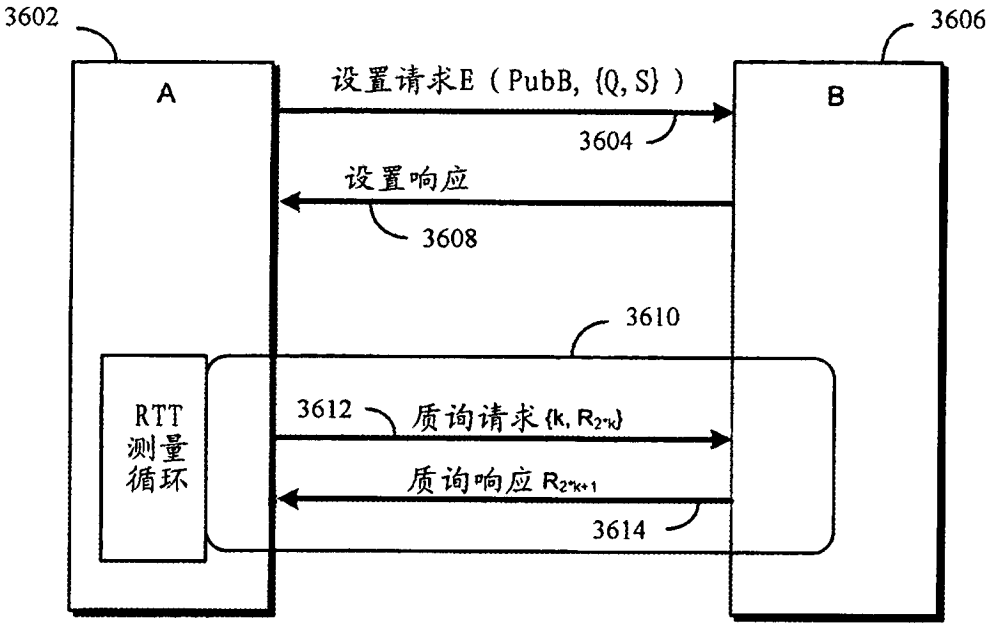

3602
3606
A
B
设置请求E（PubB, {Q, S}）
3604
设置响应
3608
3610
RTT
测量
循环
3612
质询请求{k, R2*k}
质询响应 R2*k+1
3614

图 36

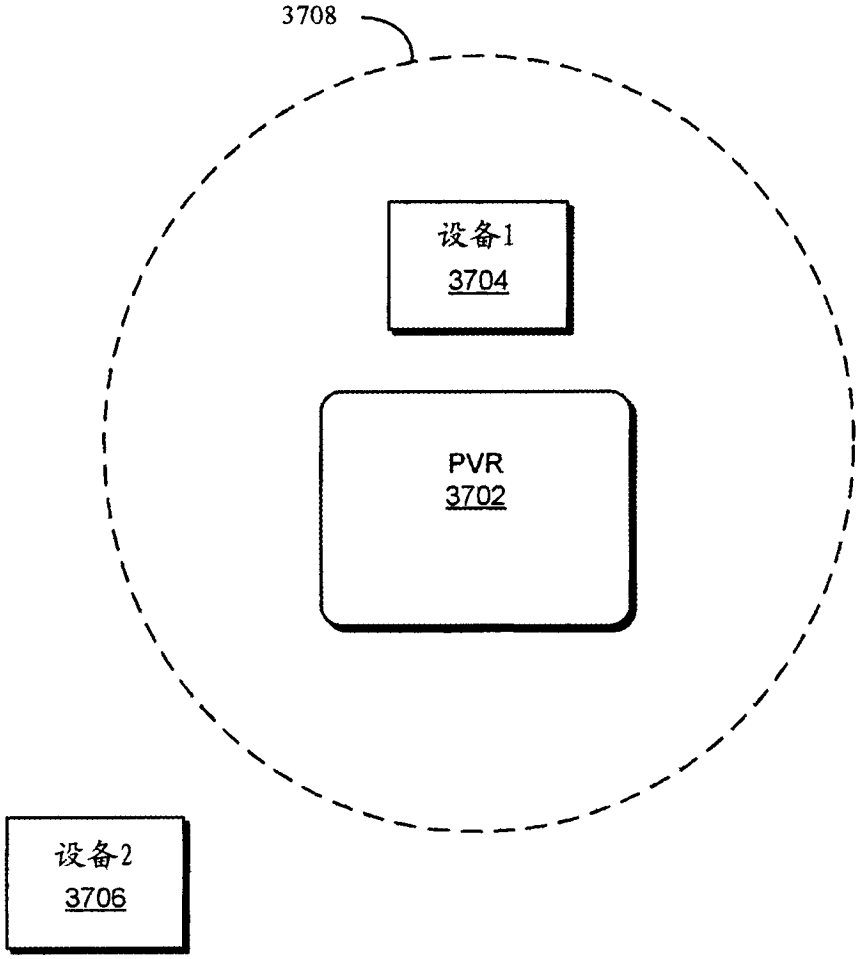

3708
设备1
3704
PVR
3702
设备2
3706

图 37

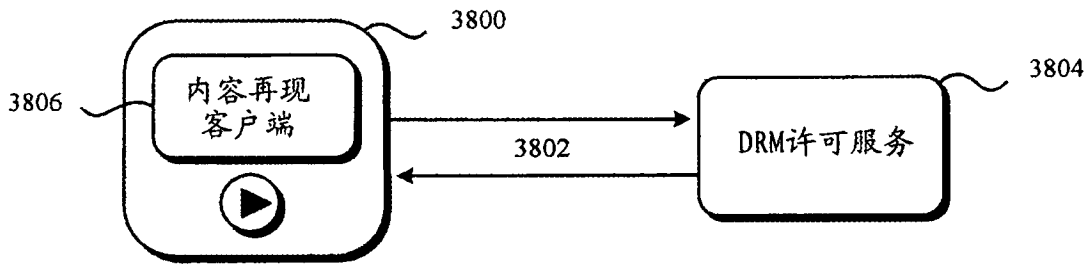

3800
3806
内容再现客户端
3802
3804
DRM许可服务

图 38

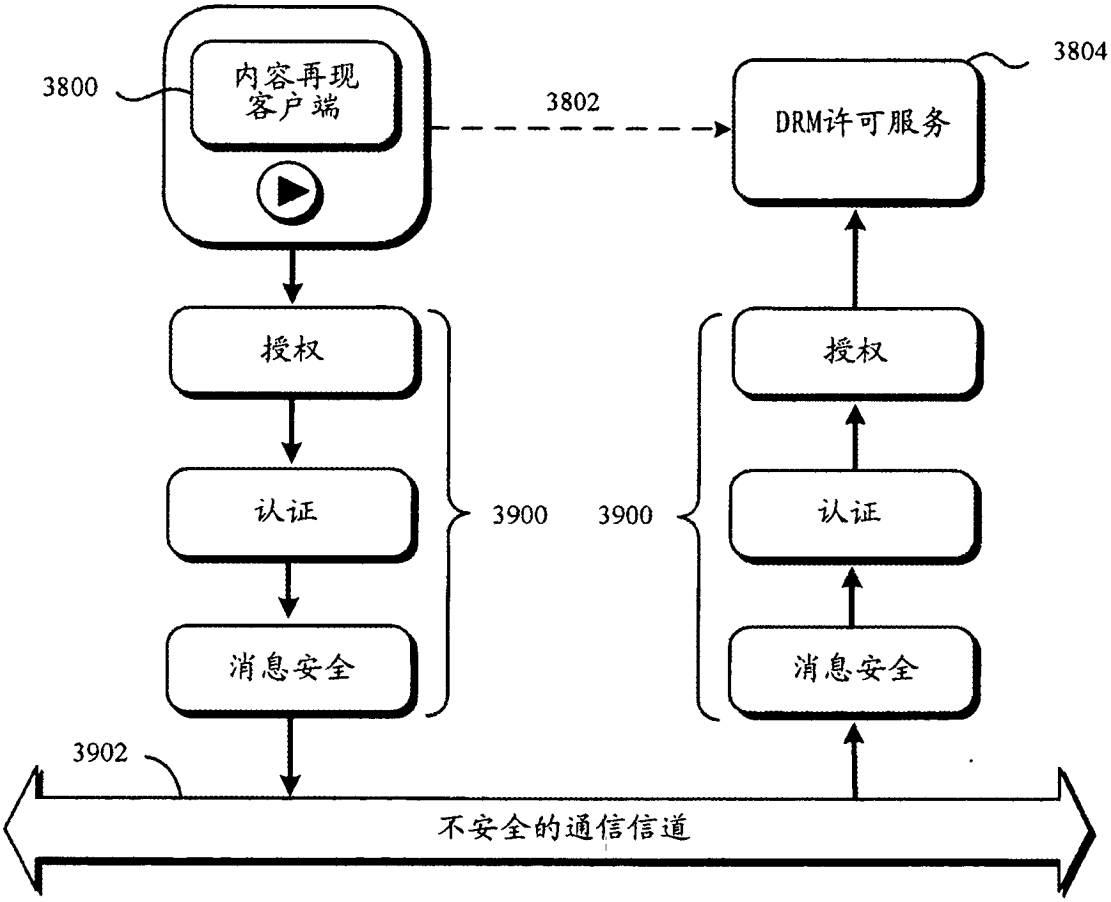

3800
内容再现客户端
3802
3804
DRM许可服务
授权
认证
消息安全
3900
3900
授权
认证
消息安全
3902
不安全的通信信道

图 39

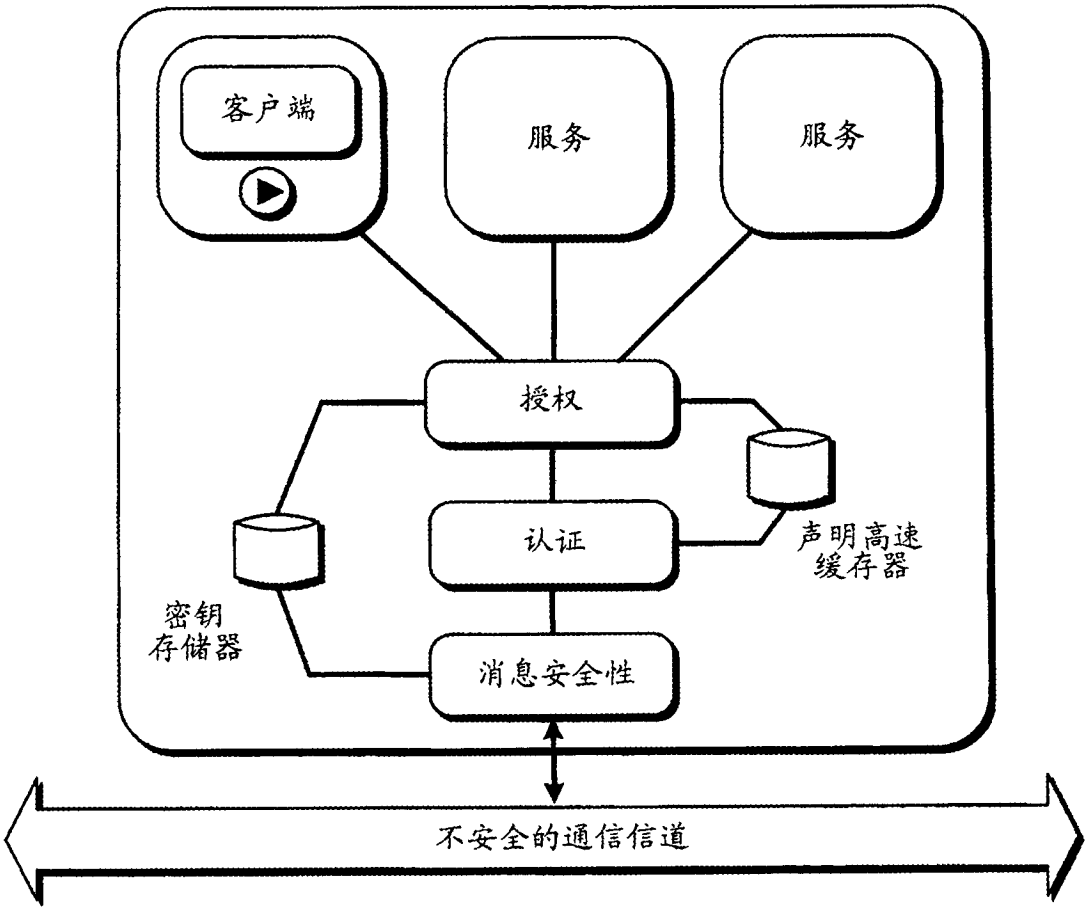


图 40

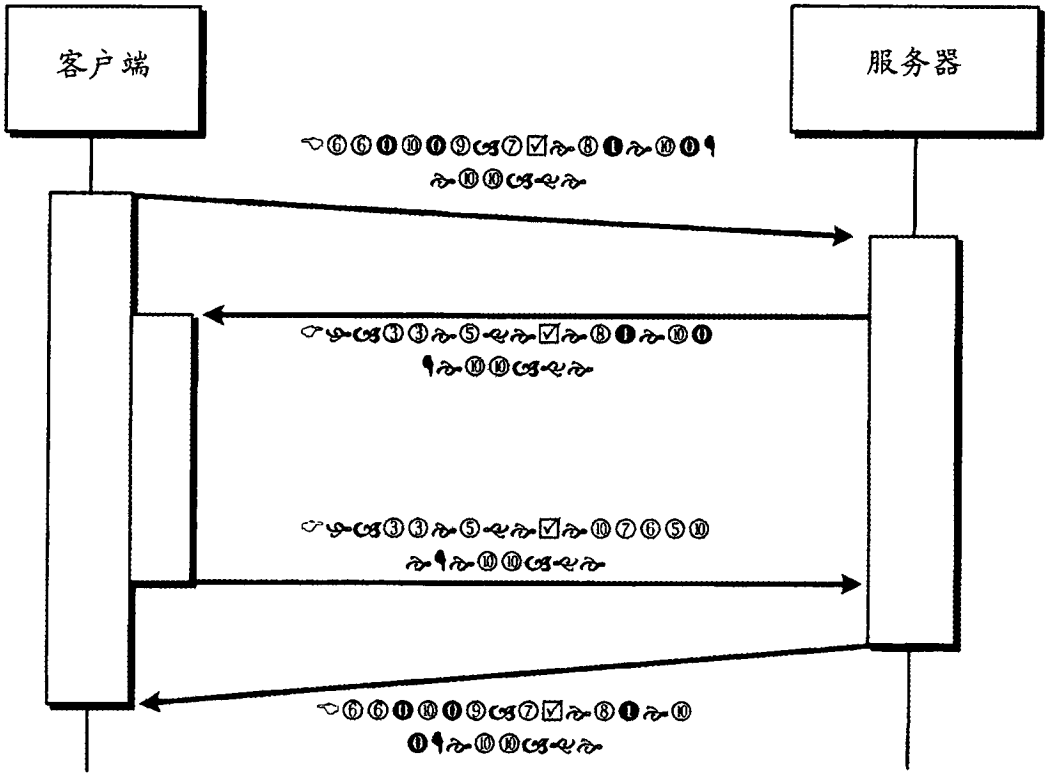


图 41

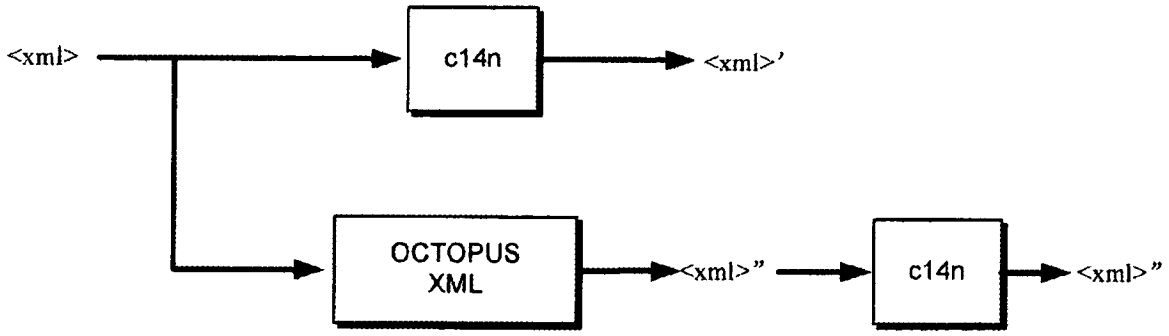


图 42