安全

我们讨论的分布式系统的最后一个原则是安全性。安全决不是最不重要的原则。然而，有人可能会说这是最困难的原则之一，因为安全性需要在整个系统中无处不在。与安全性有关的单个设计缺陷可能会使所有安全措施失效。在本章中，我们将重点介绍分布式系统中通常包含的各种支持安全性的机制。

我们首先介绍安全的基本问题。将各种安全机制构建到系统中实际上没有意义，除非知道如何使用这些机制，以及针对什么使用这些机制。这要求我们了解要执行的安全策略。首先讨论安全策略的概念，以及帮助执行此类策略的机制的一些一般设计问题。我们还简要介绍了必要的密码学。

分布式系统中的安全性大致可以分为两部分。其中一部分涉及用户或进程之间的通信，可能驻留在不同的机器上。确保安全通信的主要机制是安全通道。安全通道，更具体地说，身份验证、消息完整性和机密性，将在单独的部分中讨论。

另一部分涉及授权，它处理如何确保流程只获得对其有权访问的分布式系统中的资源的那些访问权限。授权将在处理访问控制的单独部分中讨论。除了传统的访问控制机制之外，当我们必须处理诸如代理之类的移动代码时，我们还将重点放在访问控制上。

我们还将回到命名，并注意相当棘手的问题，即确保用于检索对象的名称属于该对象，以及如何将安全命名与对人友好的名称结合起来。

安全通道和访问控制需要分发加密密钥的机制，还需要从系统中添加和删除用户的机制。

这些主题由所谓的安全管理涵盖。在另一节中，我们将讨论如何管理加密密钥、安全组管理和分发证明所有者有权访问指定资源的证书。

9.1安全简介

我们通过研究一些一般的安全问题来开始描述分布式系统中的安全性。首先，有必要定义什么是安全系统。我们将安全*策略*与安全*机制*区分开来。我们的第二个关注点是考虑一些安全系统的一般设计问题。最后，简要讨论了在安全协议设计中起关键作用的密码算法。

**安全威胁、策略和机制**

计算机系统中的安全性与可靠性的概念密切相关。非正式地说，可靠的计算机系统是我们有理由相信能够提供其服务的系统[Laprie, 1995]。可靠性包括可用性、可靠性、安全性和可维护性。然而，如果我们要信任计算机系统，那么也应该考虑机密性和完整性。**保密**是指计算机系统的属性，其信息仅向授权方披露。**完整性**是系统资产的变更只能以授权的方式进行的特性。换句话说，在一个安全的计算机系统中，不适当的更改应该是可检测和可恢复的。任何计算机系统的主要资产都是它的硬件、软件和数据。另一种看待计算机系统安全性的方法是，我们试图保护它提供的服务和数据免受**安全威胁**。有四种类型的安全威胁需要考虑[Pfleeger, 2003]：

1. 拦截

2. 中断

3. 修改

4. 制造

截取的概念是指未经授权的一方获得了对服务或数据的访问权。截取的一个典型例子是，双方之间的通信被其他人偷听了。截取同样啊发生在非法复制中，例如，在闯入私人文件系统中的私人目录中进行复制。

中断的一个例子是文件损坏或丢失。更一般地说，中断是指服务或数据变得不可用、被销毁等等的情况。从这个意义上讲，拒绝服务攻击(denial of service attack)是一种安全威胁，可以将其分类为中断。

修改涉及未经授权的数据更改或篡改服务，使其不再遵循其原始规范。修改的例子包括截取并随后更改传输的数据、篡改数据库条目，以及更改程序以便秘密地记录用户的活动。

伪造是指产生了通常不存在的额外数据或活动的情况。例如，入侵者可能试图将条目添加到密码文件或数据库中。同样，有时可以通过重播以前发送的消息来闯入系统。我们将在本章后面遇到这样的例子。

注意，中断、修改和伪造都可以看作是数据伪造的一种形式。

简单地说一个系统应该能够保护自己免受所有可能的安全威胁，并不是真正构建一个安全系统的方法。首先需要的是安全需求的描述，即安全策略。**安全策略**精确地描述了允许系统中的实体执行哪些操作，以及禁止执行哪些操作。实体包括用户、服务、数据、机器等等。一旦制定了安全策略，就可以将精力集中在可以执行策略的**安全机制**上。重要的安全机制包括：

1. 加密

2. 身份验证

3. 授权

4. 审计

加密是计算机安全的基础。加密将数据转换成攻击者无法理解的东西。换句话说，加密提供了实现数据机密性的方法。此外，加密允许我们检查数据是否被修改。因此，它还提供了对完整性检查的支持。

身份验证用于验证用户、客户机、服务器、主机或其他实体的声明身份。对于客户机，基本的前提是，在服务开始代表客户机执行任何工作之前，服务必须了解客户机的标识(除非服务对所有人都可用)。通常，用户通过密码进行身份验证，但是还有许多其他方法可以对客户机进行身份验证。

客户机经过身份验证后，需要检查该客户机是否被授权执行请求的操作。访问医疗数据库中的记录就是一个典型的例子。根据访问数据库的人员，可以授予读取记录、修改记录中的某些字段或添加或删除记录的权限。

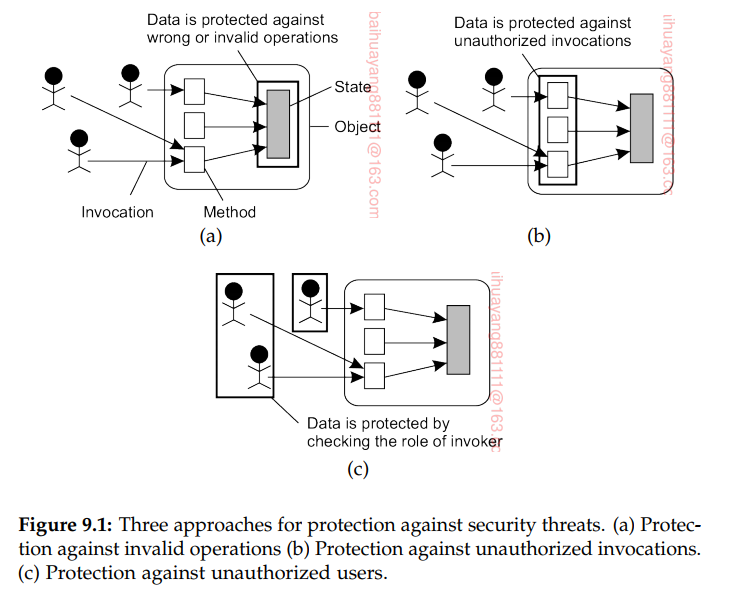
审计工具用于跟踪哪些客户机访问了哪些内容，以及以何种方式访问。虽然审计并不能真正提供任何针对安全威胁的保护，但是审计日志对于分析安全漏洞以及随后对入侵者采取措施非常有用。由于这个原因，攻击者通常不愿意留下任何可能最终导致暴露其身份的痕迹。 从这个意义上说，日志访问有时使攻击成为一项风险更大的业务。

**设计问题**

分布式系统或任何计算机系统都必须提供安全服务，通过这些服务可以实现各种各样的安全策略。在实现通用安全服务时，需要考虑许多重要的设计问题。在接下来的几页中，我们将讨论其中的三个问题:控制焦点、安全机制的分层和简单性(参见Gollmann [2006])。

**控制的重点**

在考虑保护(可能是分布式的)应用程序时，可以遵循三种不同的方法，如图9.1所示。第一种方法是直接集中精力保护与应用程序相关的数据。所谓直接，我们的意思是，不管可能对数据项执行什么操作，主要的关注点都是确保数据完整性。通常，这种类型的保护发生在数据库系统中，其中可以制定各种完整性约束，这些约束在每次修改数据项时自动检查。

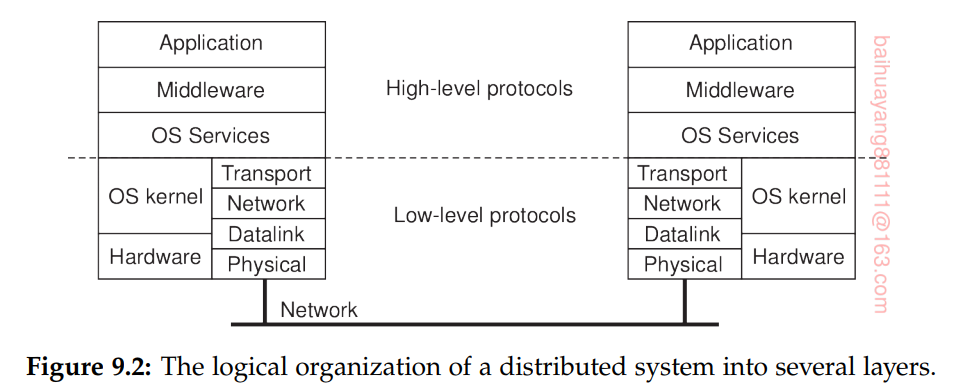


第二种方法是专注于保护，具体指定在访问某些数据或资源时可以调用哪些操作，以及由谁调用这些操作。在这种情况下，控制的重点与访问控制机制密切相关，我们将在本章后面详细讨论。例如，在基于对象的系统中，可以决定为客户机可用的每个方法指定允许哪些客户机调用该方法。或者，访问控制方法可以应用于对象提供的整个接口，或者应用于整个对象本身。因此，这种方法允许访问控制的各种粒度。

第三种方法是直接关注用户，采取措施，使只有特定的人能够访问应用程序，而不管他们希望执行什么操作。例如，银行中的数据库可以通过拒绝访问除银行高层管理人员和特定授权人员之外的任何人来保护。另一个例子是，在许多大学，某些数据和应用程序被限制只供教职员工使用，而不允许学生访问。实际上，控制的重点是定义用户所具有的**角色**，一旦验证了用户的角色，就会授予或拒绝访问资源。因此，作为设计安全系统的一部分，有必要定义人们可能具有的角色，并提供支持基于角色的访问控制的机制。本章稍后我们将回到角色。

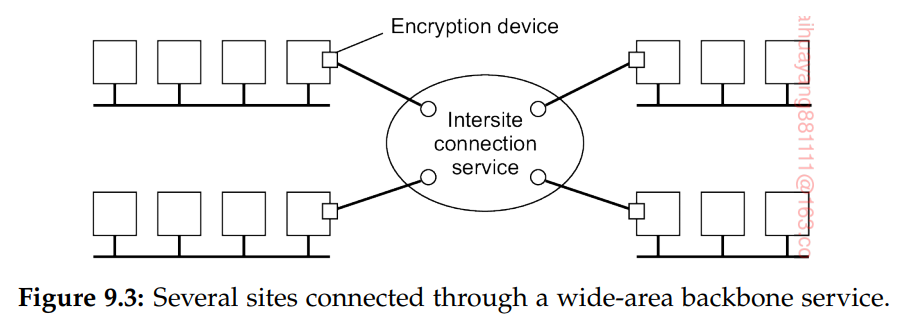
**安全机制的分层**

设计安全系统的一个重要问题是确定应该将安全机制放在哪个级别。在此上下文中，级别与系统逻辑组织成若干层有关。例如，正如我们在第4章中讨论的，计算机网络经常按照一些参考模型组织成层。在第1章中，我们介绍了由应用程序、中间件、操作系统服务和操作系统内核的独立层组成的分布式系统的组织。将这两个组织组合起来大致可以得到如图9.2所示的结果



实际上，图9.2将通用服务与通信服务分开。这种分离对于理解分布式系统中的安全分层非常重要，尤其是信任的概念。信任和安全之间的区别很重要。系统是安全的还是不安全的(考虑到各种概率度量)，但是客户端是否认为系统是安全的是一个信任的问题。安全是技术，信任是情感[毕晓普，2003]。在哪个层中放置安全机制取决于客户端对特定层中服务的安全性的信任程度。

例如，考虑一个位于不同站点的组织，该组织通过一个低层主干连接可能位于地理上分散的站点上的各种局域网，如图9.3所示。可以使用多协议标签交换(MPLS)或虚拟专用网(VPN)技术来配置这种连接。



可以通过在每个骨干交换机上放置加密设备来提供安全性，如图9.3所示。这些设备自动加密和解密站点之间发送的数据包，但不提供同一站点上主机之间的安全通信。如果站点A的Alice向站点B的Bob发送一条消息，并且她担心自己的消息被拦截，那么她至少必须相信站点间通信的加密能够正常工作。例如，这意味着她必须相信两个站点的系统管理员已经采取了适当的措施来防止篡改设备。

现在假设Alice不相信站点间通信的安全性。然后，她可能决定采取自己的措施，例如使用传输层安全性(Transport Layer Security, TLS)服务，该服务可用于跨TCP连接安全地发送消息。这里需要注意的重要一点是，TLS允许Alice设置到Bob的安全连接。所有传输级别的消息都将加密——在我们的示例中，链接级别也是加密的，但这与Alice无关。在这种情况下，Alice必须信任TLS。换句话说，她相信TLS是安全的。

在分布式系统中，安全机制通常位于中间件层。如果Alice不信任TLS，她可能希望使用本地安全RPC服务。同样，她将不得不信任这个RPC服务来做它承诺的事情，例如不泄漏信息或正确验证客户机和服务器。

只有在分布式系统中间件层中放置的安全服务所依赖的服务确实是安全的情况下，才可以信任它们。例如，如果安全RPC服务部分是通过TLS实现的，那么对RPC服务的信任取决于对TLS的信任程度。如果TLS不受信任，那么就不能信任RPC服务的安全性。

**安全机制的分布**

服务之间关于信任的依赖关系导致了可信**计算基础(TCB)**的概念。TCB是(分布式)计算机系统中执行安全策略所需的所有安全机制的集合，因此需要得到信任。TCB越小越好。如果分布式系统作为中间件构建在现有网络操作系统上，那么它的安全性可能取决于底层本地操作系统的安全性。换句话说，分布式系统中的TCB可能包括不同主机上的本地操作系统。

考虑分布式文件系统中的文件服务器。这样服务器可能需要依赖其本地操作系统提供的各种保护机制。这些机制不仅包括保护文件不被文件服务器以外的进程访问的机制，还包括保护文件服务器不被恶意关闭的机制。

因此，基于中间件的分布式系统需要信任它们所依赖的现有本地操作系统。如果不存在这种信任，那么可能需要将本地操作系统的部分功能合并到分布式系统中。考虑一个微内核操作系统，其中大多数操作系统服务作为普通用户进程运行。在这种情况下，例如，文件系统可以完全替换为适合分布式系统的特定需求的文件系统，包括它的各种安全措施。

与此方法相一致的是，根据所需的安全级别将服务分布在不同的机器上，从而将安全服务与其他类型的服务分离开来。例如，对于安全的分布式文件系统，通过将服务器放在具有可信操作系统的机器上(可能运行专用的安全文件系统)，可以将文件服务器与客户机隔离开来。客户机及其应用程序被放置在不受信任的机器上。

这种分离有效地将TCB减少到相对较少的机器和软件组件。通过随后保护这些机器免受来自外部的安全攻击，可以提高对分布式系统安全性的总体信任。

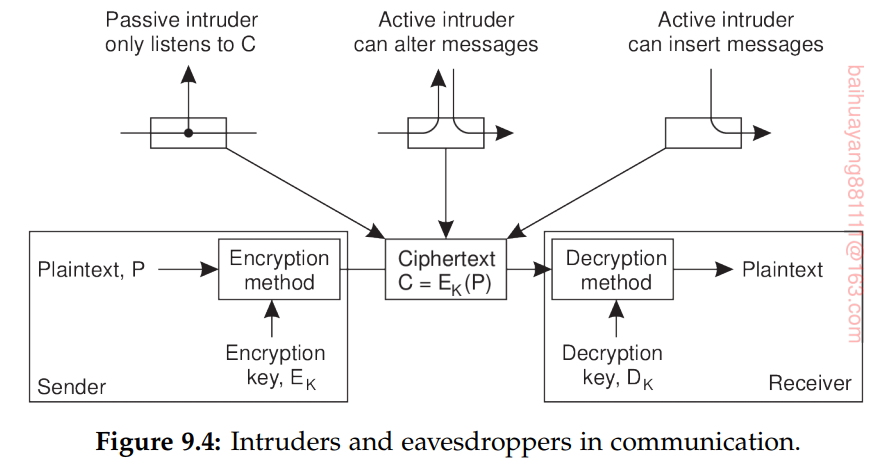
**简单**

与决定将安全机制放置在哪个层相关的另一个重要设计问题是简单性。设计一个安全的计算机系统通常被认为是一项困难的任务。因此，如果系统设计人员能够使用一些易于理解和信任的简单机制来工作，效果就会更好。不幸的是，简单的机制并不总是足以实现安全策略。再次考虑上面讨论过的Alice想要发送消息给Bob的情况。链路级加密是一种简单易懂的机制，可以防止截获站点间的消息流量。然而，如果Alice想确保只有Bob才能接收到她的消息，还需要做更多的工作。在这种情况下，需要用户级身份验证服务，Alice可能需要知道这些服务是如何工作的，以便信任这些服务。因此，用户级身份验证可能至少需要密码密钥的概念和证书等机制的意识，尽管许多安全服务是高度自动化的，并且对用户隐藏。

在其他情况下，应用程序本身就很复杂，引入安全性只会使情况变得更糟。数字支付系统是一个涉及复杂安全协议的应用领域。数字支付协议的复杂性通常是由多方需要通信才能进行支付这一事实造成的。在这些情况下，重要的是用于实现协议的底层机制是相对简单的并且易于理解。简单性将有助于最终用户对应用程序的信任，更重要的是，它将有助于使设计人员相信系统没有安全漏洞。

**密码学**

分布式系统安全的基础是使用密码技术。应用这些技术的基本思想很简单。假设发送方想要将消息m发送给接收方R。为了保护消息不受安全威胁，发送方首先将其加密为难以理解的消息m0，然后将m0发送给R。反过来，R必须将接收到的消息解密为其原始形式m。加密和解密是通过使用密钥参数化的加密方法来完成的，如图9.4所示。发送的消息的原始形式称为明文，如图9.4中P所示。加密的形式称为密文，如图C所示。



要描述用于为分布式系统构建安全服务的各种安全协议，有一个将明文、密文和密钥关联起来的符号是很有用的。按照常见的标记约定，我们将使用C = EK(P)表示密文C是通过使用密钥K加密明文P获得的。同样，P = DK(C)用于表示密钥K对密文C的解密，从而得到明文P。回到图9.4所示的示例中，在以密文C传输消息时，我们需要防范三种不同的攻击，而加密对此有所帮助。首先，**入侵者**可能在发送方或接收方都不知道正在发生窃听的情况下拦截消息。当然，如果传输的消息以一种没有正确密钥就无法轻松解密的方式进行加密，则截取是无用的:入侵者将只看到无法理解的数据。

需要处理的第二种攻击类型是修改消息。修改明文很容易;修改已正确加密的密文要困难得多，因为入侵者必须先解密消息，然后才能对其进行有意义的修改。此外，他还必须对其进行正确的加密，否则接收者可能会注意到消息已被篡改。

第三种攻击是入侵者将加密的消息插入通信系统，试图使R相信这些消息来自S。同样，加密可以帮助防范此类攻击。注意，如果入侵者可以修改消息，他也可以插入消息。

根据加密密钥和解密密钥是否相同，不同加密系统之间有一个基本的区别。在对称密码系统中，相同的密钥用于加密和解密消息：



**对称密码系统**也称为秘钥或共享密钥系统，因为发送方和接收方需要共享相同的密钥，为了确保保护有效，这个共享密钥必须保密;任何人都不许看这把钥匙。我们将使用符号KA,B来表示a和B共享的键。

在**非对称密码系统**中，用于加密和解密的密钥是不同的，但它们组合在一起形成一对惟一的密钥。换句话说，有一个用于加密的单独密钥KE和一个用于解密的密钥KD，诸如此类。



非对称密码系统中的一个密钥是私有的;另一个是公开的。因此，非对称密码系统也称为**公钥系统**。在接下来的代码中，我们使用符号KA+表示属于a的公钥，而KA -表示它对应的私钥。

哪一个加密或解密密钥实际上是公开的，这取决于如何使用这些密钥。例如，如果Alice想向Bob发送一条机密消息，她应该使用Bob的公钥加密该消息。因为Bob是惟一持有关联和私有解密密钥的人，所以他也是惟一能够解密消息的人。

另一方面，假设Bob想确定他刚刚收到的消息实际上来自Alice。在这种情况下，Alice可以保持她的私有加密密钥，以便加密她发送的消息。如果Bob能够使用Alice的公钥成功解密消息（消息中的明文包含足够的信息，使其对Bob有意义），他知道消息一定来自Alice，因为解密密钥与加密密钥是唯一绑定的。

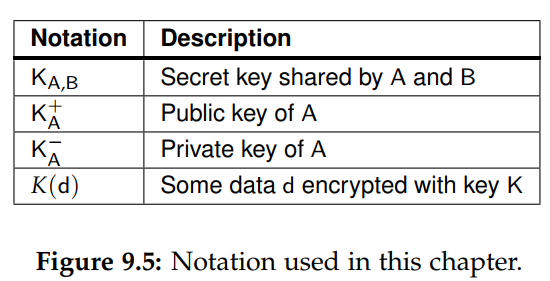
密码学在分布式系统中的最后一个应用是**哈希函数**的使用。哈希函数H接受任意长度的消息m作为输入，并生成一个长度固定的位串H作为输出：



哈希h在某种程度上类似于通信系统中为检测错误而附加到消息后面的额外位，比如循环冗余检查(CRC)。在密码系统中使用的哈希函数具有许多基本属性。首先，它们是**单向函数**，这意味着在计算上不可能找到对应于已知输出h的输入m。另一方面，从m计算h很简单。其次，它们具有较弱的抗碰撞性，这意味着给定一个输入m及其相关输出h = h (m)，在计算上不可能找到另一个不同的输入m' = m，比如h (m) = h (m')。最后，加密哈希函数也具有很强的抗碰撞性，这意味着，当只给定H时，在计算上不可能找到任意两个不同的输入值m和m'，比如H(m) = H(m')。

类似的属性必须应用于任何加密函数E和所使用的密钥。此外，对于任何加密函数EK，当给定明文P和相关密文C = EK(P)时，在计算上都不可能找到密钥K。同样，与碰撞电阻类似，同样，与碰撞抗力类似，当给定一个明文P和一个密钥K时，实际上不可能找到另一个密钥K'，使得EK(P) = EK'(P)。

为密码系统设计算法的艺术和科学有着悠久而迷人的历史[Kahn, 1967]，而构建安全系统通常是惊人地困难，甚至是不可能的[Schneier, 2000]。详细讨论这些算法超出了本书的范围。关于密码算法的信息可以在[Ferguson et al.， 2010]， [Menezes et al.， 1996]和[Schneier, 1996]中找到。图9.5总结了我们在本书的数学表达式中使用的符号和缩写。



9.2安全通道

客户机-服务器模型是组织分布式系统的一种方便方法。在这个模型中，服务器可能是分布式的和复制的，但是也可以作为相对于其他服务器的客户机。在考虑分布式系统中的安全性时，从客户机和服务器的角度考虑同样很有用。特别是，使分布式系统安全本质上归结为两个主要问题。第一个问题是如何确保客户机和服务器之间的通信安全。安全通信需要通信各方的身份验证。在许多情况下，还需要确保消息的完整性和保密性。作为这个问题的一部分，我们还需要考虑保护一组服务器中的通信。

第二个问题是授权问题:一旦服务器接受了来自客户机的请求，它如何确定客户机是否被授权执行该请求？授权与控制对资源的访问有关。

保护客户端和服务器之间的通信的问题，可以从在通信方之间建立一个**安全通道**的角度来考虑[Voydock和Kent, 1983]。安全通道保护发送方和接收方不受截取、修改和制造消息的影响。它也不一定能防止中断。保护消息不被截取是通过确保机密性来实现的:安全通道确保其消息不会被入侵者窃听。

**身份验证**

在详细讨论各种身份验证协议之前，值得注意的是，身份验证和消息完整性不能相互独立。 例如，考虑一个分布式系统，它支持两个通信方的身份验证，但不提供确保消息完整性的机制。在这样的系统中，Bob可以确定Alice是消息m的发送者。但是，如果不能保证Bob在传输期间没有修改m，那么知道Alice发送了(原始版本)m对他有什么用呢。

同样，假设只支持消息完整性，但不支持身份验证机制。当Bob收到一条消息说他刚刚中了100万美元的彩票时，如果他不能证实这条消息是由彩票的组织者发出的，他会高兴吗？

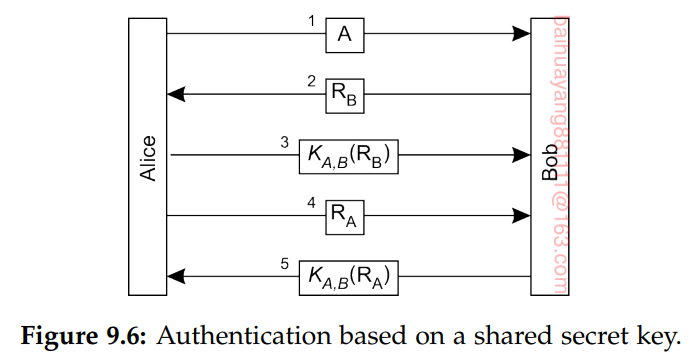
因此，身份验证和消息完整性应该同时进行。在许多协议中，这种组合的工作原理大致如下。同样，假设Alice和Bob想要通信，并且Alice在设置通道时采取主动。Alice首先向Bob发送一条消息，或者向将帮助设置通道的可信第三方发送一条消息。一旦通道设置好，Alice确定她在和Bob说话，Bob也确定他在和Alice说话，他们可以交换消息。

为了确保在进行身份验证之后交换的数据消息的完整性，通常使用会话密钥来使用密钥加密。会话密钥是一种共享(秘密)密钥，用于加密消息，以确保消息的完整性和保密性。这种键通常只在通道存在时使用。当通道关闭时，其关联的会话密钥将被销毁。

**基于共享密钥的身份验证**

让我们从查看基于Alice和Bob之间已经共享的密钥的身份验证协议开始。目前，这两者如何以安全的方式获得共享密钥并不重要。在协议描述中，Alice和Bob分别缩写为A和B，共享密钥记为KA,B。协议采用一种共同的方法，即一方向另一方发出挑战，要求对方做出只有在另一方知道共享密钥的情况下才能正确的响应。这种解决方案也称为**质询-响应协议**。

在基于共享密钥的身份验证的情况下，协议继续执行，如图9.6所示。首先，Alice将她的身份发送给Bob(消息1)，表示她想在两者之间建立一个通信通道。Bob随后向Alice发送一个**质询RB**，如消息2所示。这样的**质询**可以以随机数的形式出现。Alice需要使用她与Bob共享的密钥KA B加密挑战，并将加密的挑战返回给Bob。这个响应如图9.6中的消息3所示，其中包含KA,B(RB)。



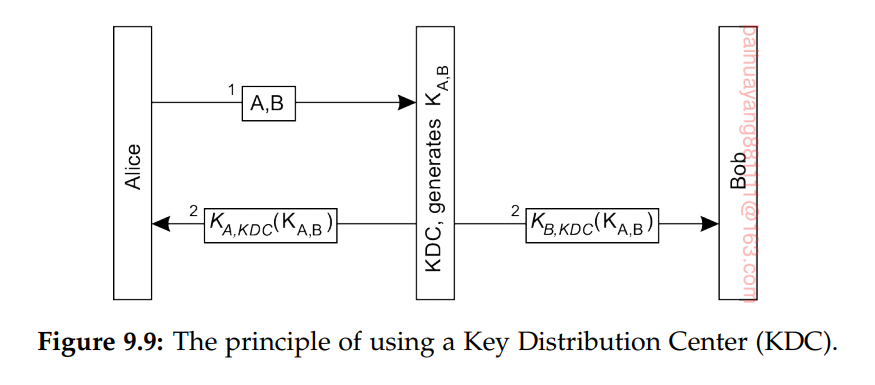
当Bob收到对他的质询RB的响应KA,B(RB)时，他可以再次使用共享密钥解密消息，看看它是否包含RB。如果是这样，那么他就知道Alice在另一边，因为还有谁可以在一开始就用KA B加密RB。换句话说，Bob现在已经证实他确实在和Alice说话。但是，请注意，Alice还没有验证它确实是通道另一端的Bob。因此，她发送一个challenge RA (message 4)， Bob通过returnin KA,B(RA)对其进行响应，如message 5所示。当Alice用KA,B解密它并看到她的RA时，她知道她在和Bob说话。（这么做的前提是，A和B已经共享了 KAB）。

多年来，密码协议的开发人员逐渐学会了许多原则。一个重要的教训是，设计能够完成它们应该完成的任务的安全协议通常比看起来要困难得多。此外，调整现有协议以提高其性能，很容易影响其正确性。关于协议设计原则的更多信息可以在[Abadi和Needham, 1996]中找到。

**使用密钥分发中心进行身份验证**

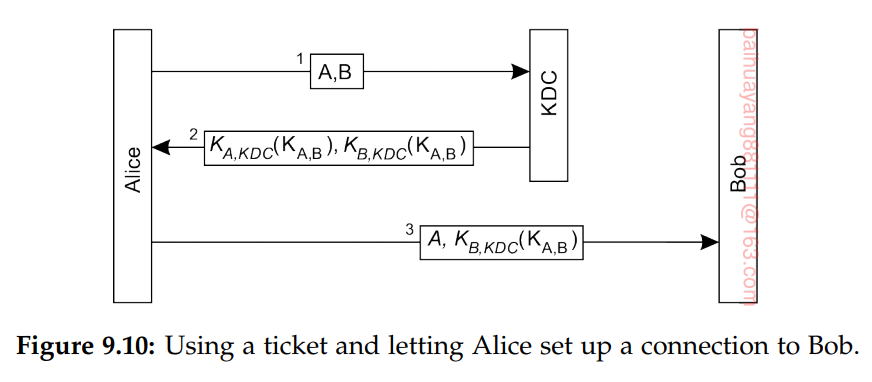
使用共享密钥进行身份验证的一个问题是可伸缩性。如果一个分布式系统包含N个主机，并且每个主机都需要与其他N - 1个主机共享一个密钥，那么整个系统需要管理N(N - 1)/2个密钥，每个主机必须管理N - 1个密钥。当N很大时，这会有问题。另一种方法是通过**密钥分发中心(KDC)使用集中式方法**。此KDC与每个主机共享一个密钥，但是不需要任何一对主机也具有共享密钥。换句话说，使用KDC需要管理N个键，而不是N(N - 1)/2，这显然是一个改进。

如果Alice想要与Bob建立一个安全通道，她可以在(可信的)KDC的帮助下这样做。整个想法是KDC向Alice和Bob分发一个密钥，用于通信，如图9.9所示。



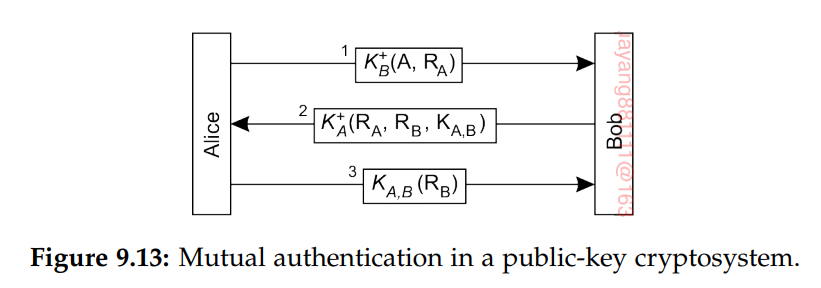
Alice首先向KDC发送一条消息，告诉它她想和Bob通话。KDC返回一条包含共享密钥KA,B的消息，她可以使用该密钥。消息使用密钥KA KDC加密，Alice与KDC共享密钥KDC。此外，KDC还将KA、B发送给Bob，但现在使用密钥KB加密，KDC与Bob共享KDC。

这种方法的主要缺点是，Alice可能希望在Bob从KDC接收到共享密钥之前就开始与Bob建立一个安全通道。此外，KDC还需要通过将密钥传递给Bob使其进入循环。如果KDC只是将KB、KDC(KA,B)传递回Alice，并让她负责连接到Bob，则可以避免这些问题。这导致了如图9.10所示的协议。消息KB,KDC(KA,B)也称为**票据**。Alice的工作是把这张票递给Bob。注意，Bob仍然是惟一能够合理使用票据的人，因为除了KDC之外，他是惟一知道如何解密票据中包含的信息的人。



Alice首先向Bob发送一个使用公钥KB+加密的挑战RA。Bob的工作是解密消息并将挑战返回给Alice。因为Bob是唯一能够解密消息的人(使用与Alice使用的公钥相关联的私钥)，所以Alice将知道她正在与Bob对话。注意，确保Alice使用Bob的公钥，而不是模仿Bob的人的公钥，这一点很重要。本章后面将讨论如何提供这种保证。

当Bob收到Alice设置通道的请求时，他返回解密的质询，以及他自己的质询RB来验证Alice。此外，他还生成一个**会话密钥KA,B**，可用于进一步的通信。Bob对Alice的挑战、他自己的挑战和会话密钥的响应被放入到一个消息中，该消息使用属于Alice的公钥KA+加密，如图9.13中的消息2所示。只有Alice能够使用她的私钥KA -解密这个消息。



最后，Alice使用Bob生成的会话密钥KA B返回她对Bob的挑战的响应。通过这种方式，她将证明她可以解密消息2，因此她实际上就是Bob正在与之交谈的Alice。

**消息完整性和机密性**

除了身份验证之外，安全通道还应该为消息的完整性和机密性提供保证。消息完整性是指消息不受秘密修改的保护;机密性确保消息不会被窃听者截获和读取。通过在发送消息之前对消息进行加密，可以很容易地建立机密性。加密可以通过与接收方共享的密钥进行，也可以通过使用接收方的公钥进行。然而，保护消息不受修改的影响稍微复杂一些，我们将在接下来讨论。

**数字签名**

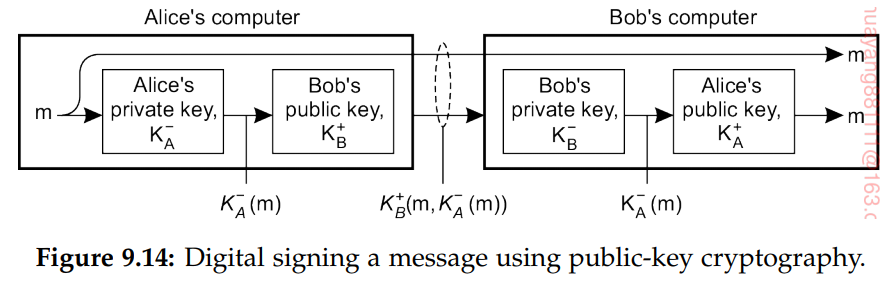
消息完整性常常超出通过安全通道进行的实际传输。考虑这样一种情况，Bob刚刚以500美元的价格将Alice的一些黑胶唱片卖给了她。整个交易都是通过电子邮件完成的。最后，爱丽丝给鲍勃发了一条信息，确认她将以500美元的价格购买这张唱片。除了身份验证之外，关于消息的完整性至少还需要注意两个问题。

1. Alice需要确保Bob不会恶意地将她消息中提到的500美元换成更高的值，并声称她承诺了超过500美元。

2. Bob需要确信Alice不能否认曾经发送过这个消息，例如，因为她有了新的想法。

如果Alice对消息进行数字签名，使其签名与消息内容惟一地绑定在一起，就可以处理这两个问题。消息与其签名之间的惟一关联可以防止对消息的修改被忽略。此外，如果Alice的签名可以被验证为真实的，那么她就不能否认她在消息上签名的事实。

有几种方法可以放置数字签名。一种流行的形式是使用公钥密码系统，如图9.14所示。当Alice向Bob发送一条消息m时，她使用自己的私钥KA -加密它，然后将它发送给Bob。如果她还想对消息内容保密，她可以使用Bob的公钥发送KB+(m, K - a (m))，它结合了m和Alice签名的版本。



当消息到达Bob时，他可以使用Alice的公钥解密它。如果他可以确信公钥确实属于Alice，那么解密签名版本的m并成功地将其与m进行比较，只能意味着它来自Alice。Alice受到Bob对m的任何恶意修改的保护，因为Bob总是必须证明m的修改版本也是由Alice签名的。换句话说，仅解密消息本身根本不能算作证据。保留m的签名版本以保护自己不被Alice拒绝，这也符合Bob自己的利益。

尽管协议本身是正确的，但是这个方案有很多问题。首先，只要Alice的私钥仍然是一个秘密，Alice签名的有效性就有效。如果Alice想退出交易，即使在向Bob发送了她的确认信息之后，她也可以声称她的私钥在消息发送之前就被偷走了。

另一个问题发生在Alice决定更改她的私钥时。这样做本身可能不是一个坏主意，因为不时地更换密钥通常有助于防止入侵。然而，一旦Alice更改了她的密钥，她发送给Bob的语句就变得毫无价值了。在这种情况下，除了在签名消息时使用时间戳之外，可能还需要一个中央权威机构来跟踪密钥何时更改。

**会话密钥**

在建立安全通道期间，身份验证阶段完成后，通信各方通常使用唯一的共享会话密钥进行保密。当不再使用通道时，会话密钥将被安全地丢弃。另一种选择是使用与用于设置安全通道的密钥相同的密钥来进行机密性。然而，使用会话密钥有许多重要的好处[Kaufman et al.， 2003]。

首先，当钥匙经常被使用时，它就更容易被发现。从某种意义上说，加密密钥和普通密钥一样，也会受到“磨损”。其基本思想是，如果入侵者能够拦截使用相同密钥加密的大量数据，就有可能发动攻击，以发现所使用密钥的某些特征，并可能显示明文或密钥本身。出于这个原因，尽可能少地使用身份验证密钥要安全得多。此外，这类密钥通常使用一些相对昂贵的带外机制(如普通邮件或电话)进行交换。以这种方式交换密钥应该保持在最低限度。

为每个安全通道生成唯一密钥的另一个重要原因是，确保防止重播攻击，就像我们以前多次遇到的那样。通过在每次设置安全通道时使用唯一的会话密钥，通信各方至少可以避免重播整个会话。为了防止重复播放前一个会话中的单个消息，通常需要额外的措施，比如将时间戳或序列号作为消息内容的一部分。假设消息完整性和机密性是通过使用用于会话建立的相同密钥实现的。在这种情况下，无论何时密钥被泄露，入侵者都可能能够解密在旧会话期间传输的消息，这显然不是一个理想的特性。相反，使用每个会话密钥要安全得多，因为如果这样的密钥被破坏，在最坏的情况下，只会影响一个会话。在其他会议期间发送的信息是保密的。

与最后一点相关的是，Alice可能想要与Bob交换一些机密数据，但是她不太信任Bob，所以她会以数据的形式给Bob提供信息，这些数据都是用长时间密钥加密的。她可能想为她与她真正信任的人交流的高度机密信息保留这样的密钥。在这种情况下，使用相对便宜的会话密钥与Bob对话就足够了。

总的来说，身份验证密钥的建立方式通常是这样的，替换它们的成本相对较高。因此，将这种持久密钥与更便宜、更临时的会话密钥组合在一起通常是实现用于交换数据的安全通道的好选择。

**安全组通信**

到目前为止，我们一直致力于在双方之间建立一个安全的沟通渠道。然而，在分布式系统中，常常需要在不止两方之间启用安全通信。一个典型的例子是复制服务器，对于复制服务器，应该保护副本之间的所有通信不受修改、制造和截取的影响，就像在两方安全通道的情况下一样。在本节中，我们将进一步研究安全组通信。

**保密组通信**

首先，考虑保护一组N个用户之间的通信免受窃听的问题。为了确保机密性，一个简单的方案是让所有组成员共享相同的密钥，该密钥用于加密和解密组成员之间传输的所有消息。 由于此方案中的密钥由所有成员共享，因此必须信任所有成员以确保密钥的机密性。与两方安全通道相比，仅使用一个共享密钥进行机密组通信更容易受到攻击。

另一种解决方案是在每对组成员之间使用单独的共享密钥。一旦一个成员泄露了信息，其他成员就可以停止向该成员发送消息，但仍然使用他们用来相互通信的密钥。但是，现在必须维护N(N - 1)/2个键，而不是必须维护一个键，这本身可能是一个难题。

使用公钥密码系统可以改善这种情况。在这种情况下，每个成员都有自己的(公钥、私钥)对，所有成员都可以使用公钥发送机密消息。在这种情况下，总共需要N对密钥。如果其中一个成员不再值得信任，则将其从组中删除，而不会影响其他密钥。

**安全复制服务器**

现在考虑一个完全不同的问题:客户机向一组复制的服务器发出请求。由于容错或性能的原因，服务器可能已经被复制，但是在任何情况下，客户机都希望响应是可信的。换句话说，无论服务器组是否受到拜占庭式故障的影响，客户机都希望返回的响应不会受到损坏。

保护客户机不受这种情况影响的解决方案是从所有服务器收集响应并对每个响应进行身份验证。如果大多数响应来自未损坏的经过身份验证的服务器，客户机也可以相信响应是正确的。不幸的是，这种方法暴露了服务器的复制，违反了复制透明性。

**例子:Kerberos 可参考网页答案** <https://www.jianshu.com/p/fc2d2dbd510b>

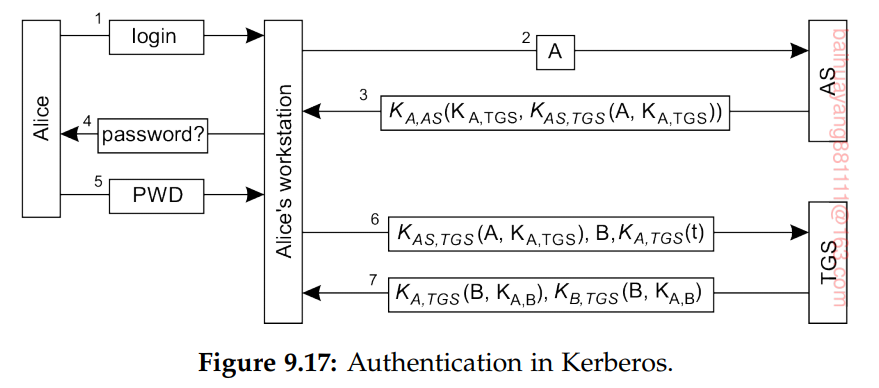
现在应该很清楚，将安全性合并到分布式系统中并不简单。问题是由于整个系统必须是安全的;如果某些部分不安全，整个系统可能会受到危害。为了帮助构建能够执行大量安全策略的分布式系统，已经开发了一些支持系统，可以作为进一步开发的基础。一个广泛使用的重要系统是Kerberos [Steiner et al.， 1988;科尔等人，1994]。

Kerberos是在mit开发的，基于**Needham-Schroeder身份验证协议**。Kerberos系统的详细描述可以在[Neuman et al.， 2005]中找到，而运行Kerberos的实际信息由Garman[2003]描述。Kerberos的一种公共实现(称为Shishi)在[Josefsson, 2015]中进行了描述。

Kerberos可以看作是一个安全系统，它可以帮助客户机与任何属于分布式系统的服务器一起设置一个安全通道。安全性基于共享密钥。有两个不同的组成部分。**身份验证服务器**(AS)负责处理来自用户的登录请求。AS对用户进行身份验证，并提供一个密钥，该密钥可用于使用服务器设置安全通道。设置安全通道由票证授予服务(TGS)处理。TGS发出特殊的消息，称为票据，用于说服服务器客户端确实是他或她所声称的那个人。我们在下面给出了具体的例子。

让我们看看Alice如何登录到使用Kerberos的分布式系统，以及她如何使用服务器Bob设置一个安全通道。要让Alice登录到系统，她可以使用任何可用的工作站。工作站将她的名字以明文发送到AS, AS返回一个会话密钥KA、TGS和一张票据，她需要将该票据交给TGS。

由AS返回的票据包含Alice的身份，以及生成的密钥，Alice和TGS可以使用该密钥彼此通信。票据本身将由爱丽丝交给TGS。因此，重要的是，只有TGS才能读取它。由于这个原因，票是用秘密密钥K AS，TGS加密的，由AS和TGS共享。



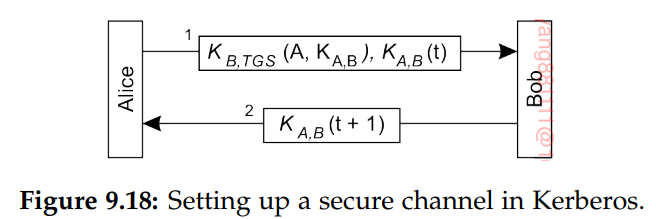
登录过程的这一部分在图9.17中分别显示为消息1、消息2和消息3。消息1并不是真正的消息，而是对应于Alice在工作站上输入她的登录名。消息2包含该名称，并被发送到AS。消息3包含会话密钥KA、TGS和票证KA、TGS(A、KA、TGS)。为了确保隐私，消息3使用密钥KA加密，由Alice和AS共享。

当工作站接收到来自AS的响应时，它会提示Alice输入密码(如message 4所示)，然后使用该密码生成共享密钥KA,AS。(取一个字符串密码，应用一个密码散列，然后取前56位作为密钥相对简单。) 注意，这种方法的优点不仅是Alice的密码永远不会在网络上以明文形式发送，而且工作站甚至不必临时存储它。此外，一旦生成了共享密钥KA (as)，工作站就会找到会话密钥KA (TGS)，并且可以忘记Alice的密码，只使用共享密钥KA (as)。

完成这部分身份验证之后，Alice可以考虑通过当前工作站登录到系统。从AS接收的票据是临时存储的(通常为8-24小时)，并将用于访问远程服务。当然，如果Alice离开她的工作站，她应该销毁所有缓存的票据。如果她想与Bob对话，她请求TGS为Bob生成一个会话密钥，如图9.17中的message 6所示。Alice有票的事实证明她是Alice。TGS使用会话密钥KA,B进行响应，同样封装在一个票据中，稍后Alice必须将该票据传递给Bob。

消息6还包含一个时间戳t，使用Alice和TGS之间共享的密钥加密。此时间戳用于防止Chuck再次恶意重播消息6，并尝试设置通向Bob的通道。TGS将在将票返回给Alice之前验证时间戳。如果与当前时间相差超过几分钟，则拒绝请求购票。

这个方案建立了所谓的**单点登录**。只要Alice不更改工作站，她就不需要向分布式系统中的任何其他服务器进行身份验证。当必须处理分布在多台机器上的许多不同服务时，此功能非常重要。原则上，服务器以某种方式将客户机身份验证委托给AS和TGS，并将接受任何具有有效票证的客户机的请求。当然，远程登录之类的服务需要关联的用户有一个帐户，但这与Kerberos身份验证无关。



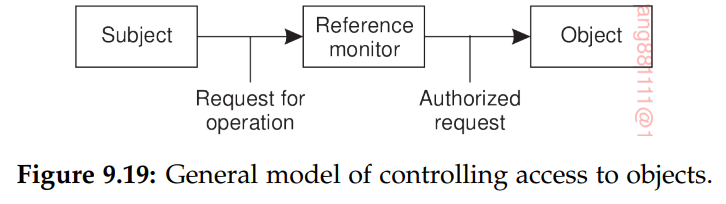
使用Bob设置安全通道现在非常简单，如图9.18所示。首先，Alice向Bob发送一条消息，其中包含她从TGS获得的票据，以及一个加密的时间戳。当Bob解密票据时，他注意到Alice正在与他交谈，因为只有TGS才能构造票据。他还找到了密钥KA,B，允许他验证时间戳。此时，Bob知道他是在和Alice说话，而不是在恶意地重播message 1。通过用KA (B(t + 1))来回应，Bob向Alice证明了他确实是Bob。

9.3访问控制

在我们目前使用的客户机-服务器模型中，一旦客户机和服务器设置了安全通道，客户机就可以发出将由服务器执行的请求。请求包括对服务器控制的资源执行操作。一般情况下，对象服务器在其控制下有多个对象。来自客户机的请求通常涉及调用特定对象的方法。只有当客户机对调用具有足够的访问权限时，才可以执行这样的请求。在形式上，验证**访问权限**被称为访问控制，而**授权**则是授予访问权限。这两个术语紧密相关，经常以可互换的方式使用。

**访问控制中的一般问题**

为了理解访问控制中涉及的各种问题，通常采用图9.19所示的简单模型。它包含的**主题**是请求一个**对象**。对象与我们目前讨论的对象非常相似。可以认为它封装了自己的状态并实现了对该状态的操作。这些对象的操作可以通过接口来提供。主题可以被认为是代表用户的进程，但也可以是需要其他服务才能完成自身工作的对象。



控制对对象的访问完全是为了保护对象不受不允许执行特定(甚至任何)方法的主题的调用（就是不合法的调用）。此外，保护可能包括对象管理问题，例如创建、重命名或删除对象。保护通常由一个称为**参考监视器**的程序来执行。参考监视器记录哪个主题可以做什么，并决定是否允许某个主题执行特定的操作。每次调用对象时都会调用此监视器(例如，由底层可信操作系统调用)。因此，引用监控器本身是防篡改的，这一点非常重要:攻击者不能随意摆弄它。

**访问控制矩阵**

对主题相对于对象的访问权限建模的一种常见方法是构造**访问控制矩阵**。每个主题由矩阵中的一行表示;每个对象都由一列表示。如果矩阵表示为M，则条目M[s, o]精确地列出了s可以请求对o对象执行哪些操作。换句话说，每当主题s请求调用对象o的方法m时，引用监视器应该检查m [s, o]中是否列出了m。如果m [s, o]中没有列出m，则调用失败。

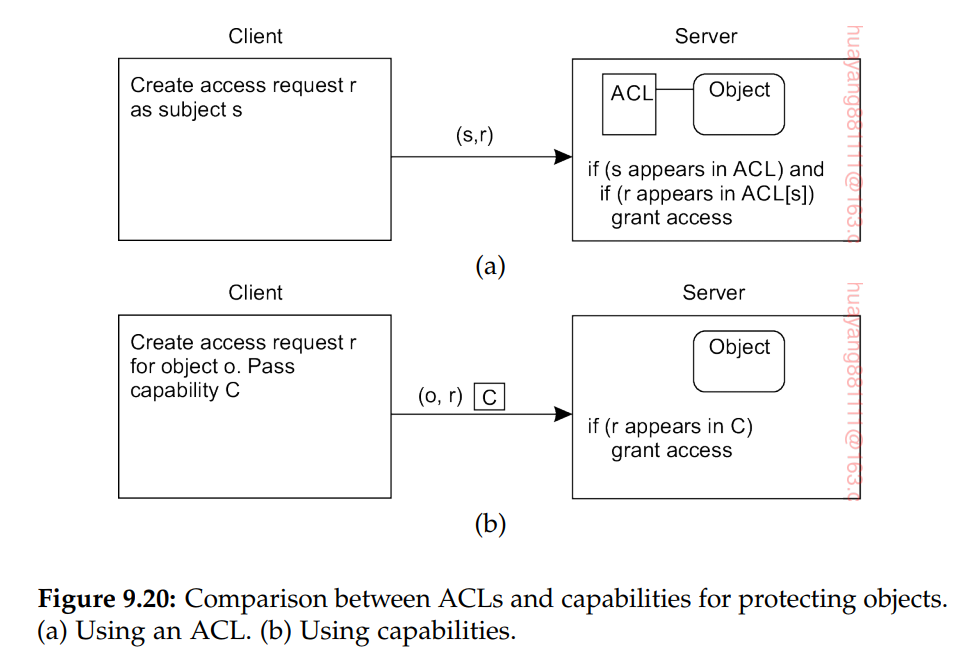
考虑到一个系统可能很容易需要支持数千个用户和数百万个需要保护的对象，将访问控制矩阵实现为一个真正的矩阵不是正确的方法。矩阵中的许多条目将是空的:一个主题通常可以访问相对较少的对象。因此，遵循其他更有效的方法来实现访问控制矩阵。

一种广泛应用的方法是让每个对象维护希望访问对象的主题的访问权限列表。本质上，这意味着矩阵是按列向分布在所有对象上的，而空条目被省略了。这种类型的实现将导致所谓的**访问控制列表(ACL)**。假设每个对象都有自己关联的ACL。

另一种方法是按行分配矩阵，方法是给每个主题一个它对每个对象具有的**功能列表**。换句话说，功能对应于访问控制矩阵中的条目。没有特定对象的功能意味着主体没有该对象的访问权限。

可以将功能与票证进行比较:它的持有者被赋予与票证相关的某些权利。同样清楚的是，一张票应该保护其持有人不受修改。一种特别适合于分布式系统的方法是使用签名保护(一组)功能。稍后在讨论安全管理时，我们将回到这些和其他问题。

使用acl（客户端维护）和功能（服务器维护）来保护对对象的访问的区别如图9.20所示。使用acl，当客户机向服务器发送请求时，服务器的引用监视器将检查它是否认识客户机，以及是否知道该客户机并允许执行所请求的操作，如图9.20(a)所示。

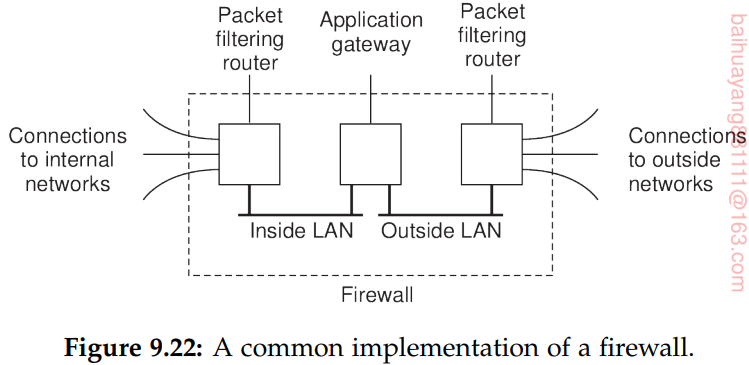


然而，当使用功能时，客户机只需将其请求发送到服务器。服务器不关心它是否知道客户端。能力就足够了。因此，服务器只需要检查功能是否有效，以及所请求的操作是否列在功能中。这种通过功能来保护对象的方法如图9.20(b)所示。

**防火墙**

到目前为止，我们已经展示了如何使用密码技术结合访问控制矩阵的一些实现来建立保护。 只要所有的通信方都遵循相同的规则，这些方法就可以很好地工作。当开发一个或多或少与世界其他地方隔离的独立分布式系统时，可以执行这些规则。然而，当需要与世界其他地方通信时，事情就变得更加复杂了，比如发送邮件、访问Web站点或提供本地资源。

为了在这种情况下保护资源，需要采取不同的办法。实际上，对分布式系统任何部分的外部访问都是由一种称为**防火墙**的特殊参考监视器控制的[Cheswick and Bellovin, 2000;Zwicky等人，2000]。防火墙是网络系统中最常用的保护机制之一。本质上，防火墙将分布式系统的任何部分与外部世界断开连接，如图9.22所示。所有的出站包，尤其是所有的入站包都是通过一台特殊的计算机路由的，并在它们通过之前进行检查。未经授权的通信将被丢弃，并且不允许继续。一个重要的问题是，防火墙本身应该受到严格保护，以抵御任何类型的安全威胁:它永远不应该失败。同样重要的是，规定什么可以通过的规则是一致的，并确定什么是打算通过的。正如Wool[2010]所报道的，正确配置防火墙是一个相当大的挑战。



防火墙通常有两种不同的风格。防火墙的一个重要类型是**包过滤网关**。这种类型的防火墙像路由器一样工作，并根据包头中包含的源地址和目标地址来决定是否传递网络包。通常，图9.22中所示的外部LAN上的包过滤网关将对传入的包进行保护，而内部LAN上的包过滤传出的包。

更微妙的情况是公司的网络由多个局域网组成。每个LAN都可以通过包过滤网关来保护，该网关被配置为仅当传入的流量来自其他LAN上的主机时才传递传入的流量。这样就可以建立一个私有的虚拟网络。

另一种类型的防火墙是应用**程序级网关**。与包过滤网关(只检查网络包的头部)相反，这种防火墙实际上检查传入或传出消息的内容。一个典型的例子是一个邮件网关，它丢弃超过一定大小的传入或传出邮件。更复杂的邮件网关存在，例如，能够过滤垃圾邮件。

应用程序级网关的另一个示例是允许外部访问数字图书馆服务器，但只提供文档摘要。 如果外部用户需要更多，则启动电子支付协议。防火墙内的用户可以直接访问库服务。

一种特殊的应用程序级网关称为**代理网关**。这种类型的防火墙作为特定类型应用程序的前端，确保只传递符合特定标准的消息。以上网为例。许多Web页面包含要在用户浏览器中执行的脚本或applet。为了防止这些代码被下载到局域网内部，所有的Web流量都可以通过Web代理网关进行定向。此网关接受来自防火墙内外的常规HTTP请求。换句话说，在用户看来，它是一个普通的Web服务器。但是，它过滤所有传入和传出的流量，方法是丢弃某些请求和页面，或者修改包含可执行代码的页面。

**安全的移动端代码**

在现代分布式系统中，一个重要的问题是能够在主机之间迁移代码，而不仅仅是迁移被动数据。然而，移动代码引入了一些严重的安全威胁。首先，需要保护主机免受恶意代理程序或下载程序的攻击。鉴于智能手机的普及，后者正变得越来越重要。分布式系统的大多数用户都不是系统技术方面的专家，也无法判断他们从其他主机获取的程序是否可信，不会破坏他们的设备。在许多情况下，即使是专家也很难检测到程序实际上是被下载的。

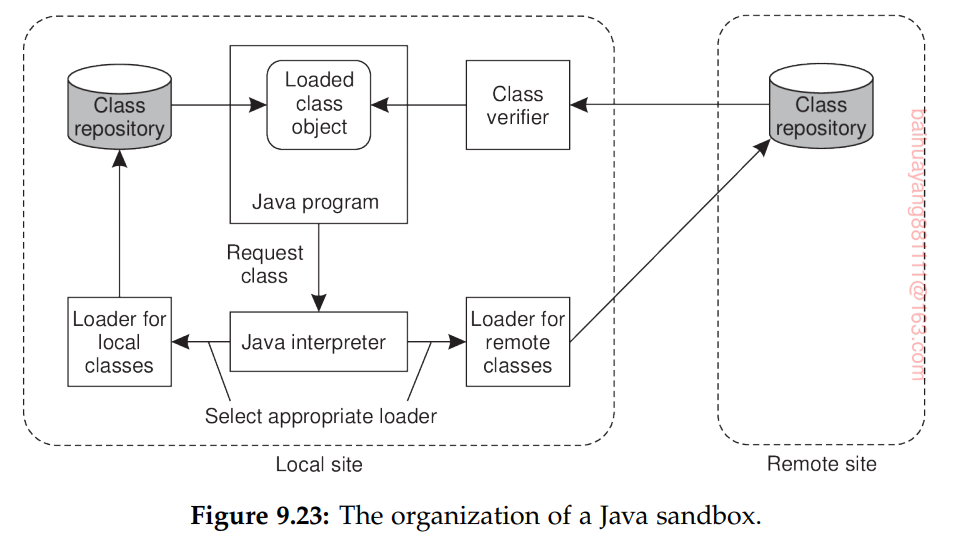
除非采取安全措施，否则一旦恶意程序在计算机中安顿下来，就很容易破坏主机。我们面临一个访问控制问题:不应该允许程序未经授权访问主机的资源。我们将看到，保护主机免受下载的恶意程序的攻击并不总是那么容易。问题不在于如何避免下载程序。相反，我们正在寻找的是支持移动代码，使我们能够以灵活的、完全受控的方式访问本地资源。

防止潜在恶意代码的一种方法是构造一个沙箱。沙箱是一种技术，通过这种技术，下载的程序可以以完全控制其每条指令的方式执行。如果试图执行主机禁止执行的指令，则程序将停止执行。同样，当一条指令访问主机不允许的内存中的某些寄存器或区域时，执行将停止。

实现沙箱的一种方法是在下载可执行代码时检查它，并为只能在运行时检查的情况插入额外的指令[Wahbe et al.， 1993]。在处理解释代码时，事情会变得简单得多。让我们简要考虑一下在Java中采用的方法(参见Oaks [2001])。每个Java程序都由许多创建对象的类组成。没有全局变量和函数；所有内容都必须声明为类的一部分。程序执行从一个名为main的方法开始。Java程序被编译成一组指令，这些指令由所谓的**Java虚拟机(JVM)解释**。因此，对于要下载和执行编译后的Java程序的客户机，客户机进程必须运行JVM。JVM随后将通过解释每个指令来处理下载程序的实际执行，从组成main的指令开始。请注意，该模型对应于**进程虚拟机**，如第3.2节所述。

在Java沙箱中，保护首先要确保处理向客户机传输程序的组件是可信的。Java中的下载由一组**类加载器**负责。每个类加载器负责从服务器获取指定的类，并将其安装到客户机的地址空间中，以便JVM可以从中创建对象。类加载器派生自现有的可信加载器，这些加载器将自动执行与沙箱关联的特定策略。

Java沙箱的第二个组件由字节码验证器组成，它检查下载的类是否遵守沙箱的安全规则。特别地，验证器检查类是否包含非法指令或可能以某种方式破坏堆栈或内存的指令。并不是所有类都被选中。图9.23显示只验证从外部服务器下载到客户机的文件。在这种情况下，位于客户机机器上的类是可信的。是否验证本地类是可以设置的另一个策略。



最后，当安全下载并验证了类之后，JVM可以从该类实例化对象并执行这些对象的方法。为了实际防止对象未经授权访问客户机的资源，**安全管理器**用于在运行时执行各种检查。 拟下载的Java程序被迫使用安全管理器；他们没有办法绕过它。因此，安全管理器扮演我们前面讨论过的引用监视器的角色。在配置沙箱方面，Java非常灵活:安全管理器使用了客户端可以设置的一组广泛的权限。安全管理器的角色是检查权限，例如文件操作、网络操作、类加载操作、运行时检查等等。

提高灵活性的下一步是要求每个下载的程序都可以进行身份验证，然后根据程序的来源强制执行特定的安全策略。要求程序能够被认证相对容易:移动代码可以被签名，就像任何其他文档一样。实际上，只接受来自受信任服务器的代码。

**拒绝服务**

访问控制通常是谨慎地确保只有授权的进程才能访问资源。与访问控制相关的一种特别烦人的攻击类型是恶意阻止授权进程访问资源。随着分布式系统通过Internet开放，对这种(拒绝服务攻击)的防御变得越来越重要。当来自一个或几个源的DoS攻击通常可以被有效地处理时，当不得不处理**分布式拒绝服务(DDoS)**时，事情就变得困难得多。

在DDoS攻击中，大量进程联合起来试图击垮网络服务。在这些情况下，我们经常看到攻击者成功地劫持了一大批机器，这些机器在不知不觉中参与了攻击。Specht和Lee[2004]区分了针对带宽损耗的攻击和针对资源损耗的攻击。

带宽消耗可以通过简单地向一台机器发送许多消息来实现。其结果是，普通的消息将很难到达接收者。资源耗尽攻击集中于让接收者耗尽其他无用消息上的资源。一个著名的资源消耗攻击是TCP同步洪泛。在这种情况下，攻击者试图启动大量的连接(即，发送SYN包作为三方握手的一部分)，但永远不会响应来自接收者的确认。其结果是服务器将很快耗尽套接字描述符，从而防止建立任何进一步的连接。

没有一种方法可以防止DDoS攻击。一个问题是，攻击者利用无辜的受害者，在他们的机器上秘密安装软件，有效地创建了所谓的僵尸网络(Silva et al.， 2013)（肉机？？）。在这些情况下，唯一的解决方案是让机器通过检查污染文件来持续监视它们的状态。考虑到病毒很容易在互联网上传播，仅仅依靠这种对策是不可行的。

更好的方法是连续监视网络流量，例如，从数据包离开组织网络的出口路由器开始。经验表明，通过删除源地址不属于组织网络的数据包，我们可以防止许多破坏。通常，可以在接近源的地方过滤的数据包越多越好。

另外，也可以将注意力集中在入侵路由器上，即流量流入组织网络的路由器。问题是检测入侵路由器的攻击已经太迟了，因为网络可能已经无法正常通信。更好的办法是让Internet上的路由器(如isp网络中的路由器)在怀疑正在进行攻击时开始丢弃数据包。一般来说，需要部署大量的技术，而新的攻击则会不断出现。Zargar等人[2013]对拒绝服务攻击和解决方案的最新技术进行了实用概述，重点关注应用程序级洪泛攻击(这种攻击越来越普遍)。Peng等人[2007]给出了另一种更侧重于网络级解决方案的概述。

9.4安全命名

安全命名是一个越来越受到关注的话题。简单地说:当客户端基于某个名称检索对象时，它如何知道它返回了正确的对象。整个问题是相当基本的:当解析DNS中的名称时，客户机如何知道它返回了正确的地址? 当使用URL和数据库查询的组合查找对象时，接收者如何知道它返回了请求的内容? 更准确地说，我们有三个问题需要担心[Smetters and Jacobson, 2009]：

1. **有效性**:返回的对象是存储在服务器上的完整的、未修改的副本吗?

2. **出处**:返回对象的服务器是否可以被信任为真正的供应商?例如，客户端可能返回原始对象的缓存版本。

3.**相关性**:考虑到所询问的内容，返回的内容是否相关?

全命名的部分且众所周知的解决方案是通过散列将对象的名称安全地绑定到其内容。简单地说:以hash(O)作为对象O的名称。这是**自认证名称**的一种形式，至少允许客户端检查有效性。自认证名称在SFS中已经率先出现[Fu et al.， 2000]。如果我们假设每个对象O都有一个已知的、经过认证的公钥KO+，那么我们还可以使用hash(KO+)作为它的名称，这样做的好处是在一定程度上可以检查出处。GlobeDoc [Popescu et al.， 2005]已经探索了这种方法。

这些自认证名称的问题在于，它们充当纯粹的标识符，并且在原则上对人类不太友好。 此外，如果我们只是将对象的散列作为其名称，那么该对象的每次更改都会导致不同的名称。 GlobeDoc中的方法在一定程度上纠正了这个问题，但是接下来我们需要确保使用了正确的公钥，并确保返回了对象的当前版本。在实践中，假设存在某种形式的公钥基础设施(我们将在本章后面讨论)，再加上能够有效地检查对象的原始服务器，应该可以解决后一个问题。结果是，如果我们想使用人性化的命名，则无法自己验证这个名字的正确定；而如果使用了 hash后的名字，由于这个名字会随时更改，将不会是我们第一次访问的名称，而且也不人性化。

Ghodsi等人[2011]广泛讨论的一种解决方法是让对象O的名称采用的形式 <hash(KO+), label>，其中label是一个对人友好的名称，可以用来查找对象。标签可以只是一个标签，也可以是全局惟一的层次结构名称，比如URL。当检索到对象时，服务器将像前面讨论的那样对其签名。接收方接受对象的公钥，验证这就是名称中使用的公钥，然后检查对象是否为真。显然，标签应该是对象内容的一部分，否则就不可能验证标签是否也属于对象。例如，源服务器可能需要单独使用与对象本身相同的键对标签进行签名。

9.5安全管理

现在我们来仔细看看安全管理。首先，我们需要考虑加密密钥的一般管理，特别是公钥分发的方式。事实证明，证书在这里扮演着重要的角色。其次，我们通过集中讨论添加当前成员信任的新组成员的问题来讨论安全管理一组服务器的问题。显然，面对分布式和复制的服务，重要的是不能通过向组承认恶意进程而损害安全性。第三，我们通过查看功能和所谓的属性证书来关注授权管理。在分布式系统中，关于授权管理的一个重要问题是，一个进程可以将其部分或全部访问权限委托给另一个进程。以一种安全的方式授权有它自己的微妙之处，我们也将在本节中讨论。

**密钥管理**

到目前为止，我们已经描述了各种密码协议，其中我们(隐式地)假设各种密钥随时可用。例如，在公钥密码系统中，我们假设消息的发送方拥有接收方的公钥，以便它可以加密消息以确保机密性。同样，在使用密钥分发中心(KDC)进行身份验证的情况下，我们假设各方已经与KDC共享了一个密钥。

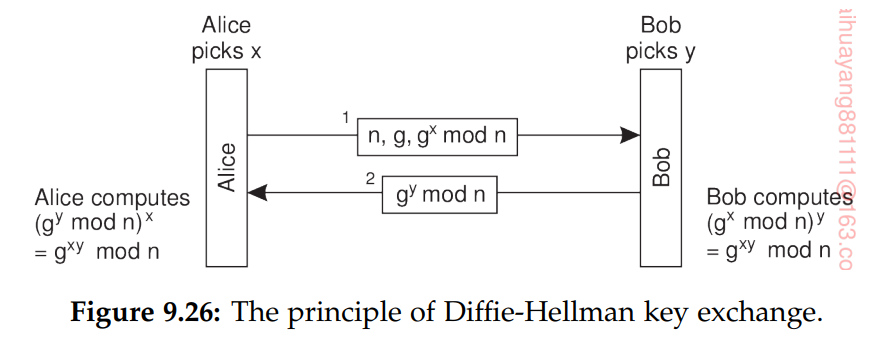
然而，建立和分发密钥并不是一件小事。例如，通过不安全的通道分发密钥是不可能的，在许多情况下，我们需要使用带外方法。此外，还需要一些机制来撤销密钥，即防止密钥在被破坏或失效后被使用。

**密钥建立**

让我们首先考虑如何建立会话密钥。当Alice想要与Bob建立一个安全通道时，她可以首先使用Bob的公钥来启动通信，如图9.13所示。如果Bob接受，他可以随后生成会话密钥并将其返回给Alice，使用Alice的公钥加密。通过在共享会话密钥传输之前对其进行加密，可以在网络上安全地传递该密钥。

当Alice和Bob已经共享密钥时，可以使用类似的方案生成和分发会话密钥。然而，这两种方法都要求通信方已经具备了建立安全通道的可用手段。换句话说，某种形式的密钥建立和分发肯定已经发生了。当通过可信的第三方(如KDC)建立共享密钥时，也可以使用相同的参数。

跨不安全通道建立共享密钥的一种优雅而广泛应用的方案是Diffie-Hellman密钥交换(Diffie and Hellman, 1976)。协议的工作原理如下。假设Alice和Bob想建立一个共享密钥。 第一个条件是他们同意两个大的数字，n和g，它们受一些数学性质的约束(我们不在这里讨论)。n和g都可以公开;没有必要向外人隐瞒他们。Alice选择了一个很大的随机数，比如说x，这是她保密的。同样地，Bob选择自己的秘密大数，比如y。此时，有足够的信息来构造一个密钥，如图9.26所示。



Alice首先发送gx mod n给Bob，同时还有n和g。需要注意的是，该信息可以以明文形式发送，因为在给定gx mod n的情况下，几乎不可能计算x。当Bob收到消息时，他随后计算(g^x mod n)y，它在数学上等于g^xy mod n。此外，他将gy mod n发送给Alice, Alice可以计算(gy mod n)x = gxy mod n。因此，Alice和Bob，而且只有这两个，现在将共享密钥gxy mod n。注意，它们都不需要让对方知道它们的私有号码(分别是x和y)。

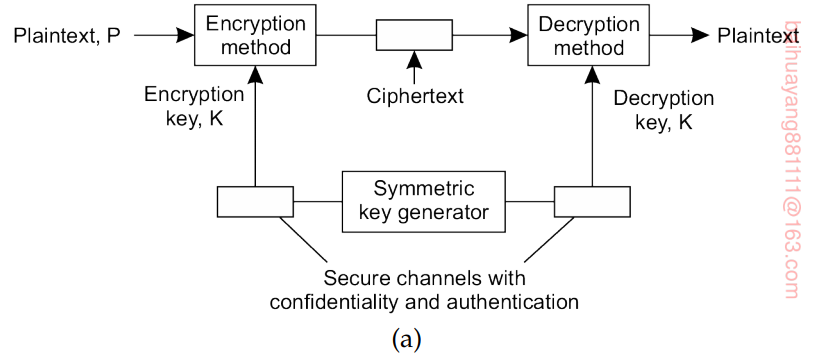
Diffie-Hellman可以看作是一个公钥密码系统。在Alice的例子中，x是她的私钥，而gx mod n是她的公钥。安全地分发公钥对于Diffie-Hellman的工作至关重要。

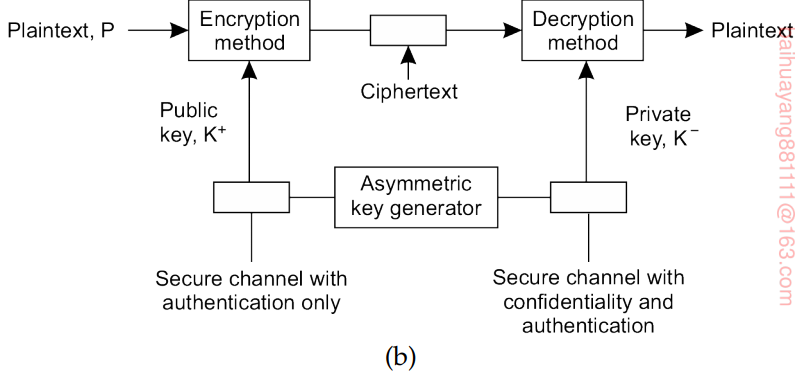
**密钥分发 这段看不太懂，可以参考网络**

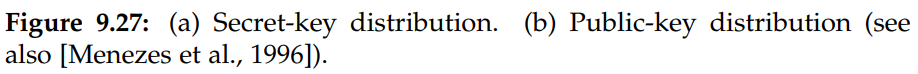
<https://www.jianshu.com/p/658334496bb9>

密钥管理中比较困难的部分之一是初始密钥的实际分布。在对称密码系统中，初始共享密钥必须通过提供身份验证和保密性的安全通道进行通信，如图9.27所示。如果Alice和Bob没有可用的密钥来设置这样一个安全通道，那么就需要在带外分发密钥。换句话说，Alice和Bob必须通过其他的通信渠道联系。

在公钥密码系统的情况下，我们需要以这样一种方式分发公钥，即接收方能够确保密钥确实与声明的私钥配对。换句话说，如图9.27所示，虽然公钥本身可以作为明文发送，但是发送它的通道必须能够提供身份验证。当然，私钥需要通过提供身份验证和机密性的安全通道发送。







当涉及到密钥分发时，经过身份验证的公钥分发可能是最有趣的。在实践中，公钥分发是通过**公钥证书**进行的。这样的证书由一个公钥和一个字符串组成，该字符串标识与该密钥关联的实体。实体可以是用户，也可以是主机或某些特殊设备。公钥和标识符已由认证机构一起签名，该签名也已放在证书上。签名通过属于认证机构的私钥KCA-进行。假设相应的公钥KCA +是众所周知的。例如，各种认证机构的公钥内置在大多数Web浏览器中，并随二进制文件一起提供。

使用公钥证书的工作原理如下。假设客户机希望确定证书中找到的公钥确实属于标识的实体。然后，它使用关联证书颁发机构的公钥来验证证书的签名。如果证书上的签名与(公钥、标识符)匹配，客户端就接受公钥确实属于标识的实体。重要的是要注意，通过按顺序接受证书，客户端实际上相信证书没有被伪造。特别是，客户端必须假设公钥KCA +确实属于关联的。特别是，客户端必须假设公钥KCA +确实属于相关的认证机构。如果有疑问，应该可以通过来自另一个不同的、可能更可信的认证机构的证书来验证K+ CA的有效性。

**安全组管理**

许多安全系统使用特殊服务，如密钥分发中心(KDCs)或认证机构(CAs)。这些服务展示了分布式系统中的一个难题。首先，他们必须被信任。为了增强对安全服务的信任，有必要对各种安全威胁提供高度的保护。例如，一旦CA受到攻击，就不可能验证公钥的有效性，从而使整个安全系统变得毫无价值。

另一方面，许多安全服务提供高可用性也是必要的。例如，在KDC的情况下，每当两个进程想要设置一个安全通道时，至少其中一个进程需要联系KDC以获得共享密钥。如果KDC不可用，则无法建立安全通信，除非可用另一种密钥建立技术，例如Diffie-Hellman密钥交换。

高可用性的解决方案是复制。另一方面，复制使服务器更容易受到安全攻击。我们已经讨论了如何通过在组成员之间共享秘密来实现安全的组通信。实际上，没有任何一个组成员能够破坏证书，从而使组本身高度安全。

最后一段省略了，可以重读密码学的书