内容提要

本文介绍了一种通用的安全服务应用程序接口 GSS-API,并在此基础上设计并实现了一种网络安全服务模型。

首先，详细阐述了 GSS-API 的概念、原理、基本元素和工作流程。

其次，介绍了应用于 GSS-API 的若干底层安全机制，包括 Kerberos V5 和公开密钥机制，并讲述了它们与 GSS-API 的结合。

最后，设计实现了一种基于 GSS-API 的安全服务模型，为应用程序提供了最大的可移植性，并且应用在河南省驻马店市中原气化集团基于网络的管理和财务核算系统中。解决了该系统在远程数据传输中的安全问题。

第 1 章 引言

1.1 网络安全的意义和发展

随着科学技术的飞速发展，人们已经生活在信息时代。计算机技术和网络技术深入到社会的各个领域，因特网把“地球村”的居民紧密的连在了一起。给人们的日常生活带来了全新的感受，人类社会各种活动对信息网络的依赖程度已经越来越大。

然而，凡事“有一利必有一弊”。人们在得益于信息革命所带来的新的巨大机遇的同时，也不得不面对信息安全问题的严峻考验。在人们对网络技术的普及叫好声尚未消失的时候，黑客攻击战在现实生活中也愈演愈烈。国内外众多的网站相继被“黑”，病毒制造者们各显其能。从 CIH 噩梦难醒，到“爱虫”病毒狂吻全球，全球“中毒”者不计其数。这些给各行各业带来了巨大的经济和其它方面的损失。除此之外，“电子战”、“信息战”已成为国与国之间、商家与商家之间的一种重要的攻击与防卫手段。因此，信息安全、网络安全的问题已经引起各国、各部门、各行、各业以及每个计算机用户的充分重视。

也就是说，在当今这个到处充斥着病毒和电脑黑客、电子窃听和电子诈骗的全球电子连接时代，网络安全已经越来越重要了。

在过去的几十年中，各个组织机构对信息安全的需要经历了两次重大的变化。在数据处理设备广泛应用之前，组织中对有价值的信息的安全保护主要是通过物理的和管理的手段实现的。前一种方法的实例是：使用坚固的文档柜，对存储敏感文档的橱柜加锁，后者的实例是在聘用过程中使用人事审查步骤。

随着计算机的引入，对保护文件和其它存储在计算机上的信息的自动化工具的需求变得愈来愈明显。这对共享系统尤为重要，对通过公共电话和数据网络就可以访问的系统来讲，此需求更为迫切。设计用来保护数据和阻止黑客的工具集合通称计算机安全措施。

互联网的安全性十分复杂：

1. 通信和网络的安全性要求似乎很直接，确实，大多数主要的安全服务要求都可以使用一个词来表示：机密性、身份验证、不

可否认性和完整性。但实现这些要求的方法却很复杂。
2．在制订特定的安全方法和算法中，一定要考虑到潜在问题的对策。很多情况下，设计对策时需要从完全不同的角度来观察问题，这样才能发现机制中未预料到的薄弱环节。
3．由于第二点的原因，用来提供服务的步骤通常不是直观的，它并不是直接来自需要安全措施的特定需求的陈述，而只是在考虑各种对策时找出起作用的对策。
4．设计好各种安全机制后，就要决定它们的使用位置。这不仅是指物理位置，例如在网络中的哪个位置需要哪种安全机制，还含有逻辑意义，例如，在结构中的哪一层或哪些层需要哪种安全机制。
5．通常，安全机制所牵涉的内容不仅仅是特定的算法和协议，还需要拥有保密信息的参与者（如加密密钥），是它们带来了创建、分布和保护秘密消息的问题。也存在着对通信协议的依赖性，协议的行为可能使制定安全机制的任务复杂化。

1.2 GSS-API

GSS-API 也就是 Generic Security Service Application Program Interface，通用安全服务应用程序接口，为保护对端应用程序之间传递的数据提供了一种方法。GSS-API 的优点就在于它能够使编程者编制的应用程序具有通用性，它的安全处理不依赖于任何特定的操作平台、安全机制、安全服务的类型以及传输协议。应用 GSS-API 具有良好的可移植性。这种可移植性是通用安全服务 API 的最重要的特点。

实际上，GSS-API 本身并不提供安全服务，它是一种以通用的方式为调用者提供安全服务的框架结构，并且被一定范围的底层机制和技术所支持。

1.3 本文的工作

本文介绍了一种通用的安全服务应用程序接口 GSS-API，并在此基础上设计并实现了一种网络安全服务模型。

首先，详细阐述了 GSS-API 的概念、原理、基本元素和工作流程。

其次，介绍了应用于 GSS-API 的若干底层安全机制，包括

Kerberos V5 和公开密钥机制，并讲述了它们与 GSS-API 的结合。

最后，设计实现了一种基于 GSS-API 的安全服务模型，为应用程序提供了最大的可移植性，并在河南省驻马店市中原气化集团基于网络的管理和财务核算系统。

1.4 本文的组织

第 1 章为引言；第 2 章介绍了安全服务的概念、攻击的类型；第 3 章详细介绍了 GSS-API 的原理、概念、基本元素和工作流程；第 4 章分析了若干用于 GSS-API 的底层安全机制；第 5 章设计并实现了一个具体的 GSS-API 接口，提供了身份验证和数据完整性校验功能；第 6 章为结束语。

第 2 章 网络安全

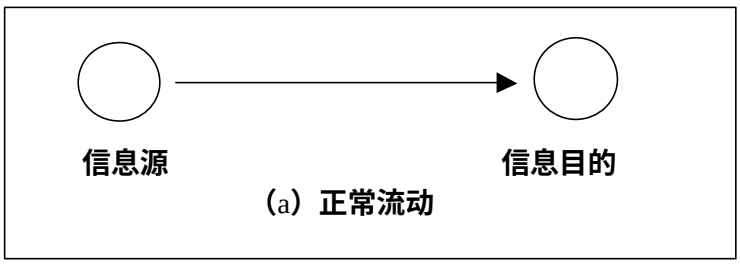
2.1 概述

为了有效地实现组织安全性的需要，有效地评估并选择各种不同的安全性产品和策略，需要能够定义安全需要的系统方法，每种方法要从 3 个方面考虑信息的安全性：

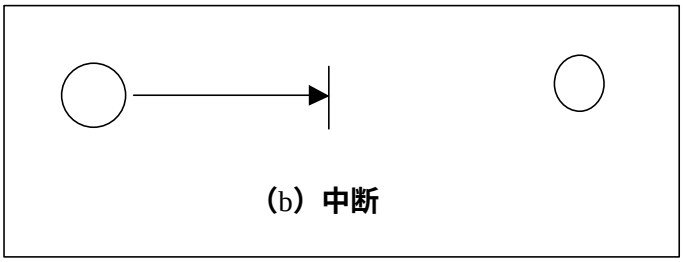
1. 安全攻击（security attack）：有损于组织机构所拥有的信息安全性的所有操作。
2. 安全机制(security mechanism)：用来检测、预防或从安全攻击中恢复的机制。
3. 安全服务(security service)：提高数据处理安全系统和组织中信息传递安全性的服务，这些服务可能会遇到安全攻击，一般使用一种或多种安全机制来提供此项服务。

2.2 安全攻击

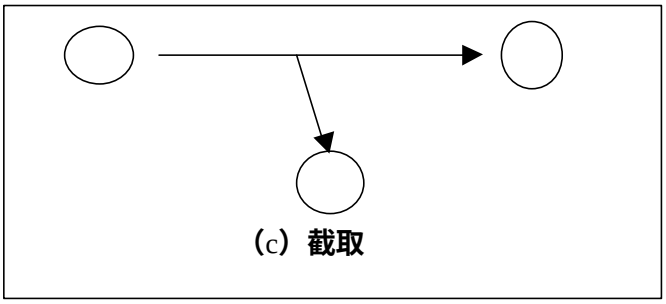
对于计算机或网络安全性的攻击，最好通过在提供信息时查看计算机系统的功能来记录其特性。一般来讲，从源（例如文件和主存区域）到目的地（例如另一个文件或用户）之间有信息的流动。在图 2－1（a）中描述了这种正常流动。图中其余部分展示的是以下 4 种类型的攻击：



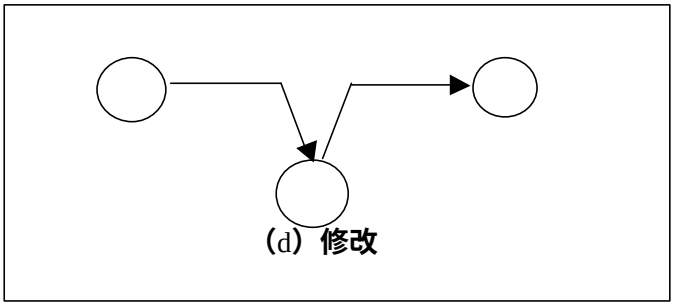
（a）正常流动



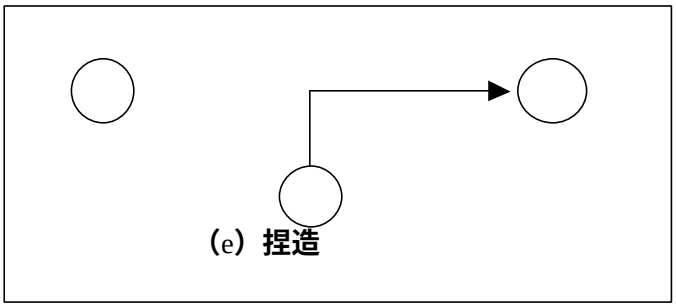
（b）中断



（c）截取



（d）修改



（e）捏造

图 2-1 安全攻击类型

1． 中断：系统资产遭到破坏或变得不能使用。这是对可用性的攻击。示例中包含了对一些硬件（如硬盘）的破坏、切断通信线路或禁用文件管理系统。
2．截取：未授权的团体得到了资产的访问权。这是对机密性的攻击。未授权的团体可能是一个人、一个程序或一台计算机。示例中包含了为捕获网络数据的窃听行为以及在未授权的情况下复制文件或程序的行为。
3．修改：未授权的团体不仅得到了访问权，而且还篡改了资产。这是对完整性的攻击。示例中包含了在数据文件中改变数值、改动程序使它按不同的方式运行、修改网络中传送消息的内容等。
4． 捏造：未授权的团体向系统中插入伪造的对象。这是对真实性的攻击。示例中包含了向网络中插入欺骗性消息或是向文件中插入补充记录。

这些攻击可以按照被动攻击和主动攻击来区分（图 2－2）。

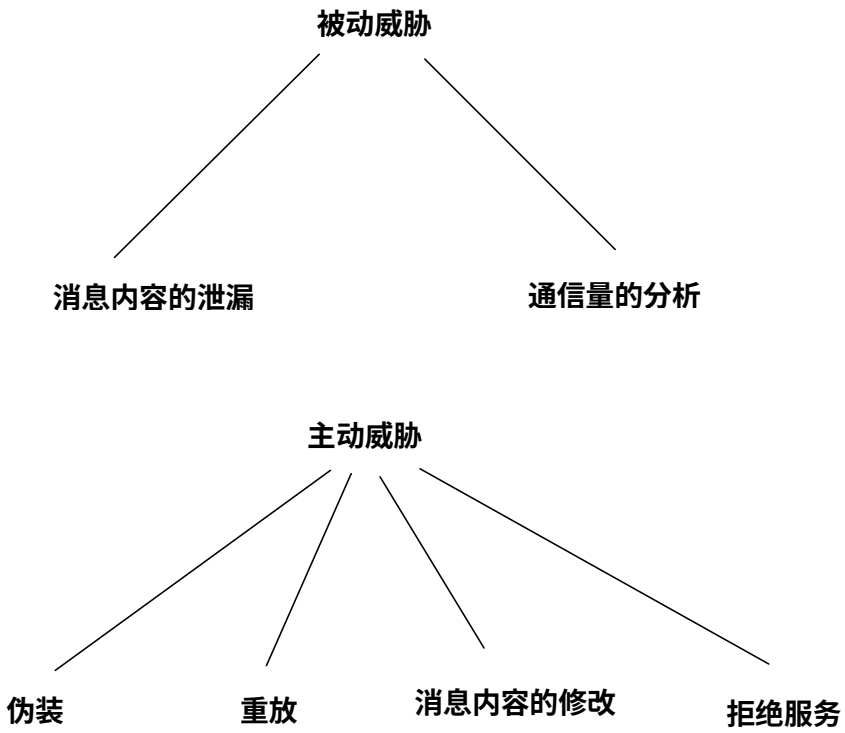


图 2-2 安全攻击的分类

被动攻击

被动攻击的特点是偷听或监视传送。其目标是获得正在传送的信息。有两种被动攻击：泄漏消息的内容和通信量的分析。

泄漏消息的内容很容易理解。电话对话、电子邮件消息、传递到文件中可能包含有敏感或机密信息。我们要防止对手从传送中下手获得这些内容。

第二种被动攻击即通信量分析则比较微妙。我们用某种方法将消息或信息的内容隐藏起来，这样即使对手捕获了消息，也不能从中提取信息。隐藏内容的常见技术是加密。如果在适当的位置有密码保护，对手仍然有可能观察这些消息的模式。对手可以确定位置和通信机主的身份，可以观察交换消息的频率和长度。这些信息对于猜测正在进行的通信特性很有帮助。

被动攻击很难被检测出来，因为它们不牵涉到数据的改动。但是，防止此类攻击却是可能的。因此处理被动攻击的重点是预防而不是检测。

主动攻击

第二类攻击类型是主动攻击。这些攻击涉及到一些数据流的修改或创建错误流，可以细分为 4 类：伪装、重放、修改消息和拒绝服务。

伪装是一个实体假装成另一个实体。伪装攻击通常包括一种其它形式的主动攻击，例如，在发生身份验证序列时，可以捕获身份验证序列并重新执行，这样，通过扮演具有特权的实体，几乎没有特权的实体获得了额外的特权。

重放涉及到被动捕获数据单元及其后来的重新传送，以产生未经授权的影响。

修改消息意味着改变旧的真实消息的部分内容，或将消息延迟或重新排序，导致未授权的操作。

拒绝服务防止或禁止对通信工具的正常使用或管理。这种攻击拥有特定的目标 :例如,实体可以取消送向特定目的地的所有消息（例如安全审核服务）。另一种拒绝服务的形式是整个网络的中断，可以通过使网络失效或通过消息过载来降低性能。

主动攻击具有与被动攻击相反的特点。虽然被动攻击很难检测出来，可采取措施防止它的成功。相反，很难绝对预防主动攻击，因为这样需要在任何时候对所有的通信工具和路径进行的保护。目标反而应该改为检测攻击，并从它引起的中断或延迟中恢复过来。因为检测具有威慑的效果，它也可以对预防作出贡献。

2.3 安全服务

安全服务的一种实用的分类如下：

1．保密性
2．身份验证
3．完整性
4．不可否认性
5．访问控制
6．可用性

2 .3 .1 保密性

保密性是为了防止被动攻击而对传送数据的保护。根据分布消息的内容不同，可以使用几种保护级别。最为广泛的服务可以保护两个用户之间在一段时间内传送的所有用户数据。

保密性的另一方面是保护通信流，以防止分析。这就要求攻击者看不到通信设备上通信流的来源和目的地、频率、长度或其它特性。

2 .3 .2 验证

验证服务致力于保证信息的可靠性。在只有一条消息的情况下，如警告和警报信号，验证服务的功能就是要保证信息接收方接收的消息确实是从它声明的来源发出的。在进行交互的过程中，如将终端连接到主机上，需要牵涉到两个方面。首先，在连接的初始化阶段，此服务应保证两个实体的可靠性（也就是说，每个实体都是所声明的实体）；其次，此服务还应保证第三方不能靠伪装成两个合法实体中的一个来干扰连接，执行未授权的传送或接收。

2 .3 .3 完整性

与保密性一样，完整性可以应用到信息流、单个信息或消息中选定的字段。而且，最有用和最直接的保护方式是对整个信息流保护。

面向连接的完整性服务可以处理消息流，保证接收与发出的内容一致，没有经过复制、插入、修改、记录或重放。数据破坏也属于本服务的管辖范围。因此，面向连接的完整性服务可以解决信息流修改和拒绝服务的问题。另一方面，无连接的完整性服务只适于处理与其他大型文本无任何关系的单个信息，通常提供对信息修改的保护。

可以区分为有恢复和无恢复的服务。因为完整性服务与主动攻击，与预防相比，我们更注意检测。如果检测到对完整性的破坏，服务可能只能报告出破坏的存在，而需要软件的其它部分或人工介入以从破坏中恢复过来。

2 .3 .4 不可否认性

不可否认性防止发送方或接收方否认消息的传送。因此，当消息发出时，接收方可以证实消息确实从声明的发送方发出。与此类似，当接收到消息时，发送方也能证实消息确实由声明的接收方接收了。

2 .3 .5 访问控制

在网络安全环境中，访问控制能够限制和控制通过通信链路对主机系统和应用的访问。为了达到这种控制，每个想获得访问的实体都必须经过鉴别或身份验证，这样才能根据个体来定制访问权力。

2 .3 .6 可用性

各种攻击都会导致可用性的损失或降低。一些攻击可以通过自动对策进行管理，如机密性和加密，但其他攻击则需要用某种形式的物理操作来防止或恢复分布式元素可用性的损失。

2.4 本章小结

本章主要介绍了网络安全服务的基本内容，包括安全攻击和一些安全服务，这正是本文要解决的一些问题，在应用中实现网

络安全服务中的一部分服务。

第 3 章 GSS-API

3.1 简介

3 .1 .1 GSS-API 的概念

通用安全服务应用程序接口（GSS-API）为保护对端应用程序之间传递的数据提供了一种方法。这种传输最典型的模型就是一台主机上的客户端和另一台主机上的服务器。而 GSS-API 的优点就在于它能够使编程者编制的应用程序具有通用性，它的安全处理不依赖于任何特定的操作平台、安全机制、安全服务的类型以及传输协议。虽然 GSS-API 使应用程序能够控制安全方面的事务，但是应用 GSS-API 的编程者可以忽略保护网络数据的一些细节。因此应用 GSS-API 具有良好的可移植性。这种可移植性是通用安全服务 API 的最重要的特点。

实际上，GSS-API 本身并不提供安全服务，它是一种以通用的方式为调用者提供安全服务的框架结构，并且被一定范围的底层机制和技术所支持，例如 Kerberos v5 或公共密钥技术。其层次结构如下图 3－1 所示：

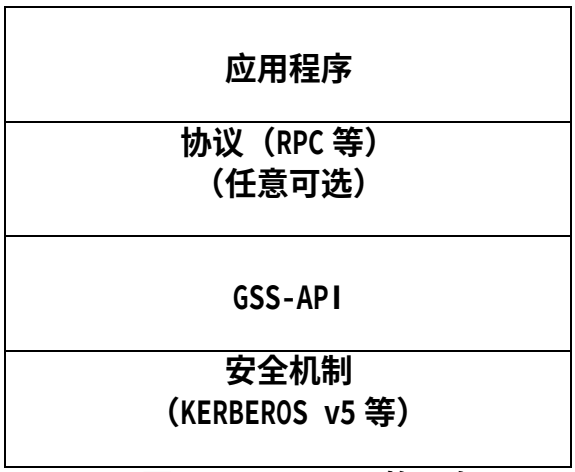


图 3－1 GSS-API 的层次

总的来讲，GSS-API 主要做了以下两件事情：

1． 创建了安全上下文,数据在安全上下文中传递于两个应用程序之间。一个安全上下文可以在两个应用程序之间被认为是一种可信任的状态。共享同一个安全上下文的两个应用程序相互知道对方的身份，并且因此在安全上下文存在期间，允许彼此间数据的传输。

2． 为要传输的数据提供了一种或多种保护类型,也就是安全服务。

当然 GSS-API 所能够做的远比这些复杂,它的其它功能包括：数据转换；错误检测；用户权力的授权；信息显示以及身份比较。GSS-API 包括许多支持或转换的功能。

3 .1 .2 应用程序的可移植性

正如前面所介绍的 GSS-API 为应用程序提供了以下几种类型的可移植性：

1． 机制无关。GSS-API 为机制的执行提供了一个通用的接口，如果已指定了一个缺省的安全机制，那么应用程序无需知道它所使用的是什么机制（例如，Kerberos v5），甚至无需知道它所用机制的类型。例如，当一个应用程序给服务器提交了一个用户信任状,它既不需要知道信任状是否具有 Kerberos 格式或是否具有能够被其它机制所使用的格式，也不需要知道信任状如何被机制存储和如何被应用程序所访问。(如果必要，应用程序可以指定使用一个特定的机制。)

2． 协议无关。GSS-API 独立于任意通信协议和协议组，例如，sokets，RPC 或 TCP/IP。

3． 操作平台无关。GSS-API 完全不用考虑应用程序运行的操作系统平台。

3 .1 .3 GSS-API 提供的安全服务

GSS-API 所提供的最基本的安全服务是鉴别。鉴别就是身份的验证。如果底层机制支持，那么 GSS-API 将提供另外两种安全服务：

1．完整性。通常我们并不知道接收的数据是否就是应用程序

传来的数据，这些数据有可能已经被破坏或篡改。GSS-API 为数据提供了一个加密标志，称为信息完整性代码（MIC），以保证接收到的数据与发送者传输的数据完全相同。这种数据的有效性验证被成为完整性。

2．私密性。鉴别和完整性服务都未对数据本身做什么，如果数据在中途被截获，那么其它人就能够阅读它。如果底层机制支持，那么 GSS-API 允许数据被加密。这种数据的加密被称为私密性。

3 .1 .4 GSS-API 不能提供的功能

虽然 GSS-API 简化了数据保护，但是为了使它通用性的特点充分体现，有些功能是它所不能提供的。如下：

1．为用户或应用程序提供安全信任状。这些必须由底层安全机制自己提供。GSS-API 不允许应用程序直接的或自动获得信任状。

2．在应用程序之间传输数据。处理对端应用程序之间的数据传输是由应用程序来实现的，无论该数据是安全相关的还是普通数据。

3．辨别传输数据的不同的类型（例如，了解或确定是否是一个普通数据并且非 GSS-API 相关）。

4．指出由远程（异步）错误引起的状态。

5．自动保护在一个多重处理过程程序中的两个过程间传输的信息。

6．分配传送给 GSS-API 函数的字符串缓冲区。

7．释放 GSS-API 数据空间。这些功能必须由 gss_release_buffer()和 gss_delete_name()直接完成。

3 .1 .5 语言绑定

GSS-API 可以与 C 语言进行绑定,另外可以与 JAVA 语言绑定。本文用的是与 C 语言绑定。

3.2 相关概念

3 .2 .1 主体

在网络安全术语中，一个主体就是一个用户，一段程序和一台主机。主体可以是客户也可以是服务器。主体的例子有：一个登录到另一主机上的用户（joe@machine）；一个网络安全服务（ nfs@machine ）；一个运行应用程序的主机 (swim2birds@eng.company.com).

3 .2 .2 信任状

3 .2 .2 .1 信任状的结构和概念

信任状是能够为应用程序声明的主体的名字提供证明的数据结构。一个应用程序用信任状来建立它的全局标识符。你可以认为信任状是一个主体的身份证明，由一系列能够证明一个人、机器或程序的身份以及他所拥有的权利的信息组成。

GSS-API 不提供信任状，信任状由 GSS-API 下层的机制在 GSS-API 功能被调用前产生。例如,在大多数情况下当用户登录一个系统时，他会得到信任状。

一个给定的信任状对一个独立的主体是有效的。一个独立的信任状能够包括该主体的几个元素，每个元素都由不同的机制产生。这就意味着在同时拥有几种安全机制的机构中获得到信任状，当它被传输到只有这些机制的子集机制的机构时也是有效的。GSS-API 通过 gss_cred_id_t 访问信任状，该结构被称为信任状句柄。信任状对于应用程序来讲是不透明的，应用程序无须知道给定信任状的细节。

信任状以以下三种形式出现：

1. GSS_C_INITIATE:这种类型的信任状表明应用程序只初始化安全上下文。
2. GSS_C_ACCEPT:这种类型的信任状表明应用程序只接收安全上下文。
3. GSS_C_BOTH:这种类型的信任状表明应用程序能够初始化或

接收安全上下文。

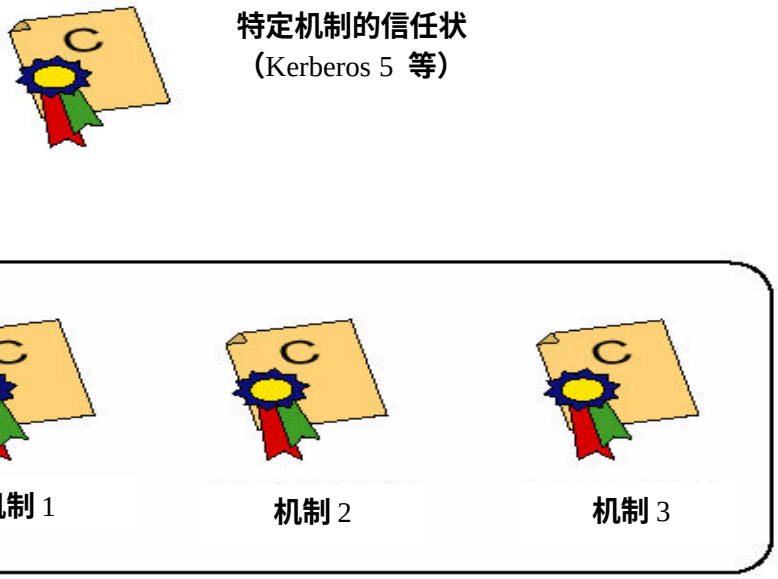


图 3-2 信任状的构成

信任状提供一个先决条件，允许 GSS-APID 对端相互建立安全上下文。调用者可以指定缺省情况下是信任状元素被用到安全上下文初始化还是接收。那些需要对信任状结构作出选择的调用者需要引用由 GSS-API 提供信任状句柄引导的信任状。

在所有的情况下，信任状的引用是非直接的，需要 GSS-API 实现来作为媒介，不要求调用者直接访问信任状元素。

单个信任状结构可能被用来初始化输出上下文并接收输入上下文。当使用时有必要指明是哪种模式(初始化还是接收用)，允许底层的机制去优化它们的处理和储存要求。由某种特殊机制定义的信任状元素可能包含多重加密密钥，如认证和信息加密采用不同的算法。

信任状结构可能包含了多种信任状元素，每个都包含了特定的机制信息，但是在给定的信任状结构下信任状元素的集合代表一个常规实体。一个信任状的结构的内容会随着特定 GSS-API 支持的机制类型的变化而变化。每个信任状元素标识了机制在代表特定的主体建立安全上下文时所需要的数据,并且可能在安全上下文的初始化和接收时包含不同的信任状。在一个给定的信任状

中的多个信任状元素不允许有重复的机制，使用模式和有效期。

3 .2 .2 .2 信任状的获得

在一个安全上下文建立之前，服务器端和客户端必须分别获得一个信任状。信任状一旦被获得，那么在它期满之前可以被重复使用，期满时必须重新获得信任状。客户端用的信任状同服务器端用的信任状的生存期可能有所不同。

以 GSS-API 为基础的应用程序用以下两种方法来获得信任状：

1. 用 gss_acquire_cred()功能（在某些情况下可能用到 gss_add_cred()功能）。
2. 当建立一个安全上下文时，由值 GSS_C_NO_CREDENTIAL 指定一个缺省信任状。

在大多数情况下，gss_acquire_cred()只被安全上下文接收者（服务器）所调用，那是因为上下文发起者（客户端）在登录时已经获得信任状。因此发起者通常只能够指定缺省的信任状。安全上下文接收者也能够不用 gss_acquire_cred()而直接使用缺省的信任状。

发起者的信任状向其它处理过程证实了自己的身份，而接收者获得信任状使它能够接收安全上下文。考虑下面的情况，客户端对服务器端提出了一个 ftp 请求，客户端在登录时已经有了一个信任状，并且当客户端试图初始化上下文时 GSS-API 自动找到该信任状。而服务器端为要求的服务（ftp）直接获得信任状。

3 .2 .2 .3 信任状的管理

确保获取和使用信任状的能力是底层特定系统机制和操作系统功能的职责。这种责任需要被实施者重视起来，因为一个实体使用一个主体信任状的能力与其成功判断该主体身份的能力是等价的。

一旦 GSS-API 信任状集合被建立,信任状的可转移性(在一个系统内可以用于其他处理和类似结构)就算本地的事情了,不需要再由 GSS-API 来定义。例如：在本地策略中任何接收到的信任状是登录到一个给定用户帐号的结果，或帐号的授权，是可以在那个帐号下传递、处理、使用的。

在信任状建立过程中，访问口令或其它量应在本地被保护并

尽可能暴露的时间最短。经常可以在用户注册时先建立一个初步的信任状，并储存起来以备以后引用。后期调用GSS_Acquire_cred()，得到引用那个初步信任状的明确的句柄，包含缺省的要被安装的信任状元素，并且当一个处理过程要求缺省的信任状处理时使用它。

3.2.2.4 缺省信任状的确定

gss_init_sec_context 和 gss_accept_sec_context 允许GSS_C_NO_CREDENTIAL的值被指定作为它们的信任状句柄的参数。这个特别的信任状句柄表明应用程序希望使用缺省主体。而特定的GSS-API实施对于决定适应机制的缺省处理是自由的,为了实现可移植性推荐使用下面的例程提供的缺省处理：

GSS_Init_sec_context:

1. 如果只有一个主体能够初始化安全上下文，并且该应用程序被授权代表这个主体，那么这个主体就可被使用。
2. 如果平台支持一个缺省的网络标识的概念，而应用程序又被授权代表该标识来初始化安全上下文，那么与这个标识相应的主体就可以被使用。
3. 如果平台支持一个缺省的本地标识的概念，并提供一种方法把本地标识来影射成网络标识；并且如果应用程序被授权代表缺省的本地表示的网络标识映象来初始化安全上下文，那么对应该标识的主体就可被使用。
4. 用户配置的缺省标识可被使用。

GSS_Accept_sec_context:

1. 如果只有一个被授权的主体标识能够接收安全上下文，那么该主体将被使用。
2. 如果机制能够通过检查安全上下文建立的token来确定目标主体的标识，并且接收到应用程序被授权作为该主体接收安全上下文，那么这个主体标识就将被使用。
3. 如果机制支持安全上下文可以被任何主体接收，并且不要求交互认证，那么在其下应用程序被授权接收安全上下文的任何主体都可以被使用。
4. 用户配置的缺省标识可以被使用。

上述规则的目的是允许在任何可能的情况下发起者和接收者都能够利用缺省的处理建立安全上下文。应用程序要求缺省的处理比使用GSS_Acquire_cred来要求一个特定的标识在机制和平台

上更具有可移植性。

3 .2 .3 token

3 .2 .3 .1 概念

在 GSS-API 中，传输的基本单元是 token。采用 GSS-API 的应用程序使用 token 进行通信，用来交换数据和进行安全的设置。token 被声明是 gss_buffer_t 数据类型并且对于应用程序来讲是不透明的。

token 分成两类：一类是上下文层 token，主要是在上下文建立时使用，进行交换的目的是为了建立和管理两端之间的安全上下文。另一类是每信息 token，主要是在安全上下文已经建立之后使用，每信息 token 与一个已建立的安全上下文有关，被交换的目的是为了向相应的数据提供保护的安全服务(如数据源发性认证，完整性和可选择的私有性)。例如，如果一个应用程序想要传输一个信息给另一个应用程序，它可能用 GSS-API 来生成一个加密的标识同信息一起传送，那么这个标识就将被存储在 token 当中。

一个信息是应用程序传给对端实体的一条数据；例如，传递给一个 ftp 服务器的 ls 命令。一个每信息 token 是 GSS-API 为该信息生成的对象，例如一个加密的标签或该信息的加密的形式。

不同的GSS-API token用于不同的目的（例如，上下文初始化，上下文接收，在一个建立的上下文中保护信息数据），接受token来分辨他们的类型是GSS-API调用者的任务,把它们与相应的安全上下文相联系起来，并且将它们传送给适当的GSS-API处理程序。

3 .2 .3 .2 应用程序的职责（非 GSS-API 提供的）

1． 收发 token。开发者为了要执行这些操作通常需要编写通用的读和写的功能。“ send_token() ”和“ recv_token() ”是这样功能的例子。

2．分辨两种 token 类型并对它们进行处理。

由于 token 对于应用程序来讲是不透明的,因此对于应用程序来讲 token 之间是没有区别的。因此在将它们传递给适当的

GSS-API 函数之前，应用程序必须能够在不能清楚地知道 token 的内容的情况下，将它们区别开。应用程序区别 token 的方法包括以下几种：

1．通过状态－也就是通过程序的控制流。例如，如果一个应用程序正在等待接收一个安全上下文，它能够假定它接收到任意一个 token 是与上下文建立相关的上下文层 token，因为它希望对端等到安全上下文完全建立好了以后再传递信息 token。在上下文建立以后，应用程序能够假定它接收的任意 token 都是信息 token。这是处理 token 最为普通的办法。

2． 一个应用程序可以在发送和接收一个 token 时来区分它的类型。例如，如果一个应用程序有一个它自己的函数发送 token 给对端，那么它就能够包含一个标识发送 token 种类的标签：

```
gss_buffer_t token; /* declare the token */
OM_uint32 token_flag /* flag for describing the type
        of token */
<get token from a GSS-API function>
token_flag = MIC_TOKEN; /* specify what kind of token
        it is */
send_a_token(&token, token_flag);
```

那么接收的应用程序会有接收的函数（“get_a_token()”）来检查 token_flag 参数。

3． 第三种途径是通过直接的标签。例如，应用程序可能使用它自己的“meta-tokens”:用户定义的结构，包含从 GSS-API 函数接收的 token,以及用户定义的字段来表示 GSS-API 提供的 token 如何使用。

3 .2 .3 .3 进程间 Token

在一个多进程的应用程序当中,GSS-API 允许一个安全上下文从一个进程传递到另一个进程。很有代表性的，一个应用程序已经接收了一个客户的安全上下文并且希望在它的进程集合中共享该安全上下文。

gss_export_context()函数创建一个包含允许上下文被另一个进程重建的信息的进程间 token。将进程间 token 在进程之间传递同将 token 传递给其它应用程序一样，是应用程序的职责。

由于这个进程间 token 可能包括密钥或其它敏感信息，并且由于不能确保所有的 GSS-API 实现加密保护进程间 token，因此

应用程序在它们被交换前应对其进行保护。如果加密技术有效的话，可能包括用 gss_wrap()来加密它们。

值得注意的是，进程间 token 在不同的 GSS-API 实现之间不具备可移动性。

通常在token内部封装的数据包括内部机制指定的标志信息，使机制层处理模块能够区分机制中用于不同目的的token。推荐机制设计者使用这种内部机制层标志，并且使机制可以确定一个调用者是否已经通过不合适的程序传送了一个特定的token。

GSS-API的开发支持基于特定底层加密技术和协议的模型,但并不意味着使用GSS-API机制的GSS-API调用者能够与使用相同技术和协议但不使用GSS-API的对端相互操作,或者与基于相同的底层技术而执行不同的GSS-API机制的对端进行交互操作。GSS-API token的格式的定义与特定的机制相关，通常把这些token集成到调用者的协议中,可能不能与使用相同低层技术的非GSS-API调用者互操作。

3 .2 .3 .4 介绍 token 的格式（format）

第一个上下文层token由GSS_Init_sec_context()来获取，它被要求在开始时就表明一个全局说明的机制标识，例如，安全机制的对象标识。这个token的剩余部分同所有其他token的整体内容一样，对支持GSS-API的特定的底层机制来讲是特定的。为GSS-API的设计者支持机制做了规定，即第一个上下文层token的报头是在机制规定信息的前面。

第一个安全上下文token:头描述|特定机制信息

token的内容对于GSS-API调用者来讲是不透明的。它们是由一个系统两端的GSS-API实现产生的，供一个GSS-API调用者传递给在一个远程系统中的对端GSS-API调用者，并且由对端GSS-API的实现来处理。token是GSS-API调用的输出（并且将被传送给对端的GSS-API）无论调用状态指示器是否显示成功执行。token传送可能采取带内的方式,集成于由GSS-API调用者为其它数据传送使用的相同的协议流中，或在通过逻辑分离信道的带外方式。

本小节规定了在GSS-API建立过程当中最初的token的与机制无关的封装表示，并结合了一个机制类型的标识符，使得token可以被对端清楚地解释。Internet标准的GSS-API机制要求在最初建立的上下文token中使用该格式；在非最初的token中使用是可选的。

token标志定编码格式源于ASN.1和DER，但是它具体的表示法是依据八位位组直接定义，而不是依据ASN.1层，目的是为了简化不使用通用的ASN.1处理代码的交互操作实现。token标志由下列元素组成，按顺序如下：

1．0x60 -- [APPLICATION 0] SEQUENCE的标志；表明构造的格式，确定长度编码。

2．Token 长度字节,指定了下面数据的长度，该元素由可变的字节数组成。

1）如果指示的值小于128，应用单字节表示，且高位置0，其余位表示该值。

2）如果指示的值等于或大于128,应用两个或多个字节表示，第一个字节的第八位置1，其它位表示余下的字节数。后面的字节存放值，每个字节8位，高位在前。应用最小的字节数来编码长度。

3．0x06 –标志 OBJECT IDENTIFIER

4．对象标识符长度，元素5中包括的编码的对象标识符的长度（字节数），编码规则如2中所述。

5．对象标识符字节，变长的字节数，按ASN.1 BER规则编码。

1）第一个字节包括两个值的和：（1）首层对象标识符部分，被40（十进制）乘，和（2）第二层对象标识符部分。在对象标识符编码中只有这里是特殊情况，用一个字节表示多余一个部分的内容。

2）后续的字节如果需要的话,在表示的对象标识符中编码后续的低层的部分。一个部分的编码可能跨越多个字节，每字节中编码7位和所有的第8位置为‘1’，除了在成分编码中的最后一个字节。字节的最小数目将被用于编码每个部分（例如没有首位为0的字节包括在成分编码中）。

（注意：在许多实现中，元素3－5可能是作为连续的串常量来存储和引用的。）

token标签后紧跟着机制定义的token目标。注意无独立尺寸指定器干涉下列指定机制定义token目标的尺寸的目标标识符的值。当允许在定义的机制token中应用ASN.1时，并不要求特定机制innerContextToken，innerMsgToken，和 sealedUserData的数据元素必须应用ASN.1 BER/DER编码规定。

3 .2 .4 安全上下文

3 .2 .4 .1 建立安全上下文

GSS-API在提供安全服务方面所作的最为重要的两件事情就是创建安全上下文和保护数据。一个应用程序在拥有它所需要的信任状以后就要建立一个安全上下文。为了完成这一工作，一个应用程序（通常是客户机）初始化一个安全上下文而另一个应用程序（通常是服务器）接收它。

两个正在通信的应用程序通过交换认证token来建立共同的安全上下文。安全上下文是一对GSS-API数据结构，其中包含了两个应用程序之间共享的信息。这些信息描述了每个应用程序的规定（在安全方面）。一个安全上下文被要求用来保护数据。

在一对端点之间可能同时存在多个上下文，使用相同或不同的信任状集合。使用不同信任状的多个上下文共存允许当信任状到期时轮换。区别基于相同信任状的多个安全上下文，使应用程序在安全的意义上来区分不同的信息流。

GSS-API与低层的协议和地址结构无关，而依赖于传送GSS-API提供的数据元素的调用者。基于此，解析通信的信息，从调用者提供的数据中提取出GSS-API相关的数据元素是调用者的职责。GSS-API与底层通信服务的连接性和非连接性无关。

在通信协议和安全上下文之间并没有一个确定的相互关系。这种分离允许GSS-API用于广泛的通信环境,简化了单独调用的调用顺序。在大多数情况下，上下文建立要求的信息能够与用户的初始化数据一同发出，无需额外的信息交换。

3 .2 .4 .2 安全上下文的初始化（客户端）

在一个应用程序和远程对端之间的安全上下文初始化通过函数gss_init_sec_context()来完成。如果成功，该函数将返回一个安全上下文被建立的上下文句柄和一个上下文层票根给接收者。在调用gss_init_sec_context()之前，客户端应该完成如下几步：

1. 获取信任状，如果有必要需调用gss_acquire_cred().而通常
   如前面讲过，在客户登录时已获得一个信任状，那么就可以

越过这一步。

2. gss_import_name()将服务器的名字输入到GSS-API内部的格式。

如果 gss_init_sec_context()完全成功完成，那么它将返回值GSS_S_COMPLETE。如果对端应用程序要求上下文建立票根，它将返回GSS_S_CONTINUE_NEEDED。如果出错那么返回错误代码。如果安全上下文初始化失败那么客户端应与服务器端重新建立连接。

3 .2 .4 .3 安全上下文的接收（服务器端）

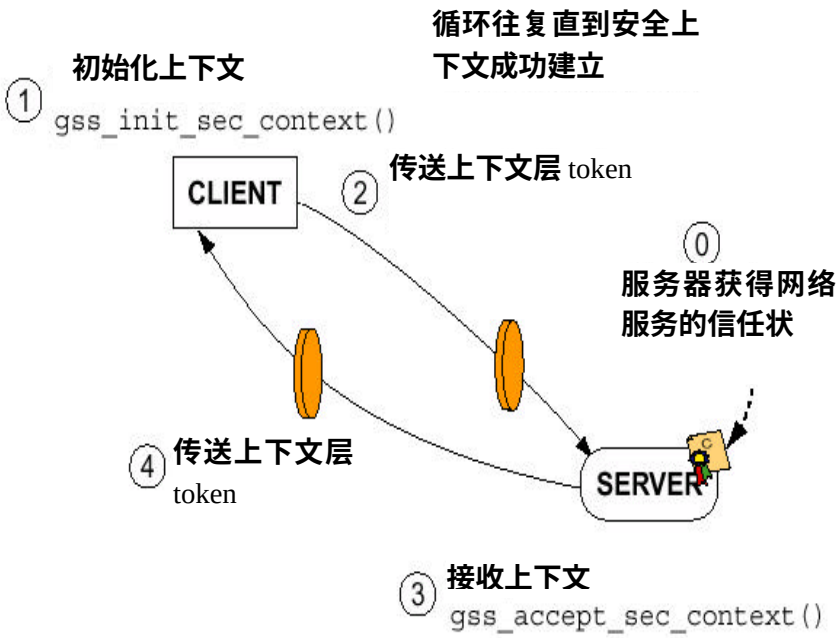
安全上下文建立的另一半就是上下文的接收，通过调用gss_accept_sec_context()函数来完成。在一个典型的假定中，服务器端接收一个由客户端初始化的安全上下文。gss_accept_sec_context()将由发起者传送来的输入票根作为它的主要输入，它将返回一个上下文句柄和输出token给发起者。在gss_accept_sec_context()被调用之前，应获取客户要求服务的信任状。

3.3 工作流程

使用GSS-API的基本步骤：

1. 每个应用程序，发送者和接收者，在没有自动获取信任状的情况下，直接获取信任状。
2. 发送者初始化一个安全上下文，接收者接收该安全上下文。
3. 发送者为它要发送的数据信息提供安全性保护。也就是说发送者或者加密信息，或者用一个认证标志标记信息。发送者传输被保护的信息。(发送者也可以选择不提供这两种安全保护，在这种情况下该信息只具有缺省的GSS-API安全服务，也就是鉴别，以便使接收者确认发送者。)
4. 接受者解密信息（如果需要）并进行校验。
5. （可选的）接受者返回一个鉴定标志给发送者确认。
6. 两端的应用程序破坏共享的安全上下文。如果必要的话，它们也能够释放任意剩余的GSS-API数据。

第一阶段：安全上下文的建立



第二阶段：数据传送

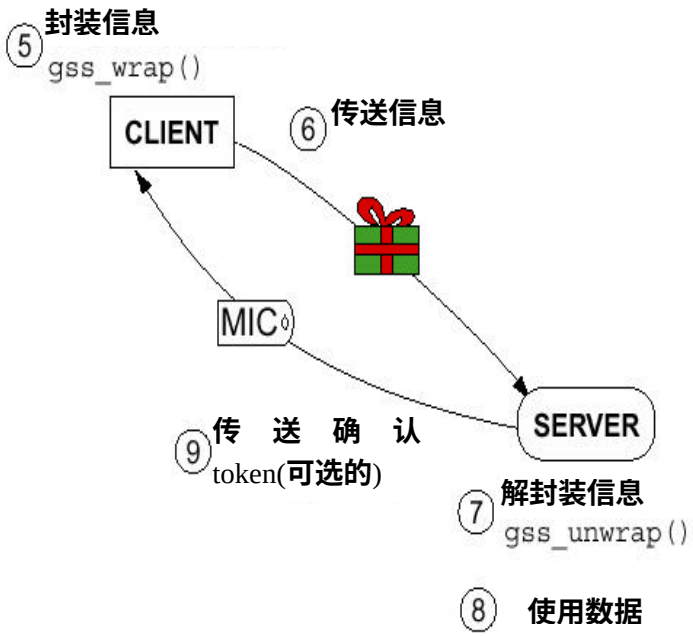


图 3-3 工作流程

图3-3介绍了这个简单的过程，这只是GSS-API的一种用法。

3.4 接口描述

这部分描述了GSS-API的服务接口，将GSS-API提供的调用分为四组。信任状管理调用与主体获得和释放信任状相关。上下文层调用与主体间的安全上下文的管理相关。每信息调用与在已建立的安全上下文上的单个信息的保护相关。支持调用为GSS-API调用者提供了一些有用的辅助函数。

下表简单介绍了这些调用。

信任状管理

GSS_Acquire_cred	获得要用的信任状
GSS_Release_cred	用后释放信任状
GSS_Inquire_cred	显示关于信任状的信息
GSS_Add_cred	增加构造信任状
GSS_Inquire_cred_by_mech	显示每信息信任状资料
	上下文级调用
GSS_Init_sec_context	初始化外发的安全上下文
GSS_Accept_sec_context	接收进入的安全上下文
GSS_Delete_sec_context	冲掉不再需要的安全上下文
GSS_Process_context_token	处理在上下文上接收的控制
	票根
GSS_Context_time	指明在上下文上剩余的有效
	时间
GSS_Inquire_context	显示关于上下文的信息
GSS_Wrap_size_limit	确定 GSS_Wrap 票根尺寸限
	制
GSS_Export_sec_context	将上下文传递给其它处理过
	程
GSS_Import_sec_context	输入传递来的上下文

每信息调用

GSS_GetMIC	申请完整性检查接收作为票
	根从信息中分离出来
GSS_VerifyMIC	对票根和信息一起进行有效

	的完整性检测
GSS_Wrap	标记，任意加密,压缩封装
GSS_Unwrap	解压缩，如果需要就解密，有效的完整性检测

支持调用

GSS_Display_status	将状态码转换成可印刷格式
GSS_Indicate_mechs	指出在本地系统中支持的机制类型
GSS_Compare_name	比较两个名字是否等同
GSS_Display_name	将名字转化成可印刷格式
GSS_Import_name	将可印刷名字转化成规范化格式
GSS_Release_name	释放规范化格式名字的存储空间
GSS_Release_buffer	释放可印刷名字的存储空间
GSS_Release_OID	释放OID目标存储空间
GSS_Release_OID_set	释放OID设置目标存储空间
GSS_Create_empty_OID_set	建立空 OID 设置
GSS_Add_OID_set_member	向 OID 设置中添加新选项
GSS_Test_OID_set_member	检测 OID 是否为 OID 设置的成员
GSS_OID_to_str	以字符串形式显示 OID
GSS_Str_to_OID	由字符串构造OID
GSS_Inquire_names_for_mech	指出机制支持的名字类型
GSS_Inquire_mechs_for_name	指出支持名字类型的机制
GSS_Canonicalize_name	将名字转化为每机制格式
GSS_Export_name	具体化每机制名字
GSS_Duplicate_name	复制名字目标

3 .4 .1 信任状管理调用

在这部分详细介绍几个重要的与信任状管理相关的函数。

3 .4 .1 .1 GSS_Acquire_cred 调用

输入：

- o desired_name INTERNAL NAME
- o lifetime_req INTEGER
- o desired_mechs SET OF OBJECT IDENTIFIER
- o cred_usage INTEGER -0=INITIATE-AND-ACCEPT, 1=INITIATE-ONLY, 2=ACCEPT-ONLY

输出：

- o major_status INTEGER,
- o minor_status INTEGER,
- o output_cred_handle CREDENTIAL HANDLE,
- o actual_mechs SET OF OBJECT IDENTIFIER,
- o lifetime_rec INTEGER

主状态返回值代码：

- o GSS_S_COMPLETE 表明要求的信任状已被成功的建立，适应在lifetime_rec中指定的时间,适合在cred_usage中要求的用法，适合在actual_mechs中指定的机制类型集合,并且应用output_cred_handle 中返回的句柄可以在以后的使用中引用这些信任状。
- o GSS_S_BAD_MECH 表明要求的mech_type不为GSS-API实现所支持，导致了信任状建立的失败。
- o GSS_S_BAD_NAMETYPE 表明提供的desired_name无法解释或属于不被适用的底层GSS-API机制所支持的类型,所以不能依靠这个desired_name建立信任状。
- o GSS_S_BAD_NAME表明提供的desired_name与捆绑合成类型指示器信息不一致，所以不能依靠这个desired_name建立信任状。
- o GSS_S_FAILURE表明信任状建立失败的原因在GSS-API层未详细说明，包括在输入desired_name的参数中建立和使用与身份命名相关的信任状缺乏认证。

GSS_Acquire_cred()是用来获取一个信任状，以便一个主体能够在由desired_name输入参数表示的标识下初始化和接收安全上下文。在成功完成的情况下，返回的output_cred_handle结果为以后对该获取的信任状的引用提供了一个句柄。在典型的情况下，单用户客户端过程要求为安全上下文的建立提供缺省的信任

状，那么它就无需启用该调用。

一个调用者可能会为desired_name提供NULL值，这就表示要求一个与为调用者选定的缺省主体标识相关的信任状。由GSS-API实现使用的来选择合适的主体标识的过程是本地的事情。有可能当GSS_Acquire_cred()被调用时，为同一个主体标识预建立的多个信任状已经存在；在这种情况下选择一个特定的信任状的方法就是本地的事情了。输入到GSS_Acquire_cred()中的lifetime_req参数为本地的GSS-API的实现提供了有用的信息,使在最好的满足调用者要求的方法中消除歧义。

lifetime_rec结果指明了获取的信任状有效期。如果对信任状的有效期没有规定,那么机制可能返回一个保留值表示INDEFINITE。一个GSS_Acquire_cred()的调用者能够要求一个时间长度作为获取信任状的有效期（lifetime_req参数),或可以要求一个带有缺省时间间隔的信任状。(GSS-API不支持对过期信任状的要求。)特定的机制和实现可能与信任状有效期指定器绑定在调用GSS_Acquire_cred()之前（例如，与用户登录过程相关)。因此，要求lifetime_req非缺省值的调用者必须承认这样的要求

GSS_Acquire_cred()的调用者能够清楚地指定机制类型的集合，包含在返回的信任状中（desired_mechs参数）或者能够要求系统定义的缺省机制类型集合。系统指定缺省的集合的选择应该符合应用程序的可移植性。调用者可能会查询actual_mechs的返回值以确定返回到信任状可能使用的机制的集合。

3 .4 .1 .2 GSS_Release_cred 调用

输入：

- cred_handle CREDENTIAL HANDLE

输出：

- major_status INTEGER,
- minor_status INTEGER

主状态返回值代码：

- GSS_S_COMPLETE表明由输入cred_handle引用的信任状被释放，以便调用者以后的访问。对被授权共享访问这些信任状的其它进程的影响就是本地的事情了。
- GSS_S_NO_CRED表明没有执行释放操作,或者是由于输入的cred_handle无效,或者是由于调用者缺乏对被引用的信任状访问的授权。

- GSS_S_FAILURE表明由于GSS-API 层没有指定而使释放操作失败。

GSS_Release_cred（）为调用者提供了一种方法，当信任状的应用不再需要时将其释放。注意系统指定的信任状管理函数也存在，例如确保当所有相关的进程结束时，将这些进程共享的信任状正确的删除，即使这些进程没有提出明确的释放要求。假定多个调用者没有从对同一信任状的获得的访问授权中被排除，那么对GSS_Release_cred()的调用不能在系统范围的基础上删除特定的信任状集合。

3 .4 .2 上下文层调用

这组调用用来在对端建立和管理安全上下文。一个上下文发起者调用GSS_Init_sec_context()，生成一个调用者要传递给目标端的token。在目标端，token被传递给GSS_Accept_sec_context()。根据底层机制类型和指定的选项，在安全上下文建立的期间执行附加的token交换 这样的交换被包含在GSS_S_CONTINUE_NEEDED状态中,从GSS_Init_sec_context()和 GSS_Accept_sec_context()中返回。

当一个安全上下文不再需要时，建立上下文的任一部分都可能调用GSS_Delete_sec_context()来冲刷掉上下文信息。GSS_Process_context_token()用来处理携带上下文层控制信息的接收token。GSS_Context_time()允许调用者确定一个建立的上下文的有效期的长度。GSS_Inquire_context()返回描述上下文特性的状态信息。GSS_Wrap_size_limit()允许调用者确定由GSS_Wrap()操作生成的token的尺寸。GSS_Export_sec_context()和 GSS_Import_sec_context()使激活的安全上下文在端系统的进程间能够传输。

3 .4 .2 .1 GSS_Init_sec_context 调用

输入:

- claimant_cred_handle CREDENTIAL HANDLE
- input_context_handle CONTEXT HANDLE
- targ_name INTERNAL NAME,
- mech_type OBJECT IDENTIFIER

- deleg_req_flag BOOLEAN,
- mutual_req_flag BOOLEAN,
- replay_det_req_flag BOOLEAN,
- sequence_req_flag BOOLEAN,
- anon_req_flag BOOLEAN,
- lifetime_req INTEGER
- chan_bindings OCTET STRING,
- input_token OCTET STRING

输出:

- major_status INTEGER,
- minor_status INTEGER,
- output_context_handle CONTEXT HANDLE,
- mech_type OBJECT IDENTIFIER
- output_token OCTET STRING
- deleg_state BOOLEAN,
- mutual_state BOOLEAN,
- replay_det_state BOOLEAN,
- sequence_state BOOLEAN,
- anon_state BOOLEAN,
- trans_state BOOLEAN,
- prot_ready_state BOOLEAN
- conf_avail BOOLEAN,
- integ_avail BOOLEAN,
- lifetime_rec INTEGER

主状态返回值代码：

- GSS_S_COMPLETE表明上下文层信息被成功初始化，并且返回的output_token将为目的端口在新建的安全上下文上执行每信息处理提供重要的信息。
- GSS_S_CONTINUE_NEEDED 表明在返回的output_token中的控制信息必须传递给目标端口，并且在与该上下文相关的每信息处理被执行前，必须接收一个答复并将其作为继续调用GSS_Init_sec_context()的input_token参数。
- GSS_S_DEFECTIVE_TOKEN表明在input_token上执行的一致性检测失败了，并阻止以该token为基础的进一步处理。
- GSS_S_DEFECTIVE_CREDENTIAL 表明在由claimant_cred_handle引用的信任状结构上执行的一致性检测失败，并阻止了使用该信任状结构进行的进一步处理。

o GSS_S_BAD_SIG表明接收的input_token包括一个不正确的完整性检测，所以不能完成上下文的建立。

o GSS_S_NO_CRED表明没有建立安全上下文，或者是因为输入cred_handle是无效的，或者因为引用的信任状只对接受者有效，还可能因为调用者缺乏访问引用的信任状的授权。

o GSS_S_CREDENTIALS_EXPIRED 表明通过输入claimant_cred_handle参数提供的信任状不再有效，所以不能完成安全上下文的建立。

o GSS_S_BAD_BINDINGS表明调用者提供的chan_bindings与从input_token中选取出来的不匹配,表示一个安全相关的事件并阻止安全上下文的建立。

o GSS_S_OLD_TOKEN表明对于检测完整性来讲输入票根已经过时。这是上下文建立中一个致命的错误。

o GSS_S_DUPLICATE_TOKEN表明输入票根有正确的完整性检测，但它是一个已经处理过的票根的副本。这是上下文建立中的一个重大错误。

o GSS_S_NO_CONTEXT表明在context_handl提供的输入中无有效的上下文经过验证；只有当后续者调用下列GSS_S_CONTINUE_NEEDED状态返回时，主状态才会返回。

o GSS_S_BAD_NAMETYPE 表明提供的targ_name无法翻译或不为适用的底层GSS-API机制所支持,所以不能完成安全上下文的建立。

o GSS_S_BAD_NAME表明提供的targ_name同一体绑定类型指示符信息矛盾，所以不能完成安全上下文的建立。

o GSS_S_FAILURE表明由于GSS-API 层未说明的原因上下文的建立未能完成，并且没有利用定义界面防御措施。

该程序由上下文发起者使用，并且通常在被选择的机制类型协议中发送一个适合目的端口使用的output_token。利用claimant_cred_handle 引用的信任状结构中的信息，GSS_Init_sec_context()初始化了一个与目的端targ_name相关的安全上下文所需的数据结构。

3 .4 .2 .2 GSS_Accept_sec_context 调用

输入:

o acceptor_cred_handle CREDENTIAL HANDLE

o input_context_handle CONTEXT HANDLE

- chan_bindings OCTET STRING,
- input_token OCTET STRING

输出:

- major_status INTEGER,
- minor_status INTEGER,
- src_name INTERNAL NAME
- mech_type OBJECT IDENTIFIER,
- output_context_handle CONTEXT HANDLE,
- deleg_state BOOLEAN,
- mutual_state BOOLEAN,
- replay_det_state BOOLEAN,
- sequence_state BOOLEAN,
- anon_state BOOLEAN,
- trans_state BOOLEAN,
- prot_ready_state BOOLEAN,
- conf_avail BOOLEAN,
- integ_avail BOOLEAN,
- lifetime_rec INTEGER,
- delegated_cred_handle CREDENTIAL HANDLE,
- output_token OCTET STRING

主状态返回值代码:

- GSS_S_COMPLETE表明上下文层数据结构成功初始化并且与该上下文相关的每信息处理可以执行。
- GSS_S_CONTINUE_NEEDED 表明在返回的output_token 中的控制信息必须传递给发起者，并且必须接收一个回复，并在与该上下文相关的每信息处理执行前，将其作为调用GSS_Accept_sec_context()的input_token 参数。
- GSS_S_DEFECTIVE_TOKEN
- GSS_S_DEFECTIVE_CREDENTIAL
- GSS_S_BAD_SIG
- GSS_S_DUPLICATE_TOKEN
- GSS_S_OLD_TOKEN
- GSS_S_NO_CRED
- GSS_S_CREDENTIALS_EXPIRED
- GSS_S_BAD_BINDINGS
- GSS_S_NO_CONTEXT
- GSS_S_BAD_MECH

- GSS_S_FAILURE

GSS_Accept_sec_context()程序由上下文目的端口使用。利用输入acceptor_cred_handle引用的信任状结构中的信息，它验证了引入的input_token并返回鉴别的src_name和 mech_type。

3 .4 .3 每信息调用

这组调用用来执行在一个建立的安全上下文上的每信息保护处理。这些调用可能被一个上下文的发起者或目的端调用。这组中的四个成员可以分成两组， GSS_GetMIC() 的输出是GSS_VerifyMIC()的输入,GSS_Wrap()的输出是GSS_Unwrap()的输入。

GSS_GetMIC()和 GSS_VerifyMIC()支持数据的源发性认证和数据的完整性服务。当在输入信息中调用GSS_GetMIC()，它将产生一个每信息token,其中包括允许底层机制提供指定的安全服务的数据项。原始信息同生成的每信息token一起被传送到远程对端；这两个数据元素由GSS_VerifyMIC()进行处理，它会对与这个单独token相关的信息进行验证。

GSS_Wrap()和GSS_Unwrap()支持调用者要求的私有性安全服务。GSS_Wrap()输出一个单独的数据元素，其中封装了可选加密的用户数据和相关的token数据项。从GSS_Wrap()输出的数据元素被传送到远程对端并由系统中的GSS_Unwrap()进行处理。GSS_Unwrap()包含了解密和与鉴定和完整性相关的数据项的确认。

3 .4 .3 .1 GSS_GetMIC 调用

输入:

- context_handle CONTEXT HANDLE,
- qop_req INTEGER
- message OCTET STRING

输出:

- major_status INTEGER,
- minor_status INTEGER,
- per_msg_token OCTET STRING

主状态返回值代码：

- GSS_S_COMPLETE表明与建立的安全上下文相适应的完整性检查被成功的提供，并且信息和相关的per_msg_token可以传输。
- GSS_S_CONTEXT_EXPIRED表明上下文相关的数据项已经期满，所以要求的操作不能被执行。
- GSS_S_CREDENTIALS_EXPIRED表明上下文已经被验证,但是与它相关的信任状已经期满，所以要求的操作不能被执行。
- GSS_S_NO_CONTEXT表明输入的context_handle提供的安全上下文没有被验证有效的。
- GSS_S_BAD_QOP表明提供的QOP值未被验证和不为上下文所支持。
- GSS_S_FAILURE表明上下文被验证，但由于GSS-API层未指明的原因导致要求的操作不能被执行。

3 .4 .3 .2 GSS_VerifyMIC 调用

输入:
- context_handle CONTEXT HANDLE,
- message OCTET STRING,
- per_msg_token OCTET STRING

输出:
- qop_state INTEGER,
- major_status INTEGER,
- minor_status INTEGER,

主状态返回值代码:
- GSS_S_COMPLETE表明信息被成功验证。
- GSS_S_DEFECTIVE_TOKEN表明在接收的per_msg_token上的一致性检测执行失败，阻止了应用该token进行进一步的处理。
- GSS_S_BAD_SIG表明接收的per_msg_token包含一个不正确的关于信息的完整性检查。
- GSS_S_DUPLICATE_TOKEN, GSS_S_OLD_TOKEN, GSS_S_UNSEQ_TOKEN, GSS_S_GAP_TOKEN
- GSS_S_CONTEXT_EXPIRED
- GSS_S_CREDENTIALS_EXPIRED
- GSS_S_NO_CONTEXT
- GSS_S_FAILURE

3 .4 .3 .3 GSS_Wrap 调用

输入:
- o context_handle CONTEXT HANDLE,
- o conf_req_flag BOOLEAN,
- o qop_req INTEGER
- o input_message OCTET STRING

输出:
- o major_status INTEGER,
- o minor_status INTEGER,
- o conf_state BOOLEAN,
- o output_message OCTET STRING

主状态返回值代码:
- o GSS_S_COMPLETE 表明input_message被成功处理，output_message准备发送。
- o GSS_S_CONTEXT_EXPIRED
- o GSS_S_CREDENTIALS_EXPIRED
- o GSS_S_NO_CONTEXT
- o GSS_S_BAD_QOP
- o GSS_S_FAILURE

3 .4 .3 .4 GSS_Unwrap 调用

输入:
- o context_handle CONTEXT HANDLE,
- o input_message OCTET STRING

输出:
- o conf_state BOOLEAN,
- o qop_state INTEGER,
- o major_status INTEGER,
- o minor_status INTEGER,
- o output_message OCTET STRING

主状态返回值代码:
- o GSS_S_COMPLETE表明input_message被成功处理并且结果的output_message可用。
- o GSS_S_DEFECTIVE_TOKEN

- GSS_S_BAD_SIG
- GSS_S_DUPLICATE_TOKEN, GSS_S_OLD_TOKEN, GSS_S_UNSEQ_TOKEN,和 GSS_S_GAP_TOKEN
- GSS_S_CONTEXT_EXPIRED
- GSS_S_CREDENTIALS_EXPIRED
- GSS_S_NO_CONTEXT
- GSS_S_FAILURE

3.5 本章小结

本章主要介绍了 GSS-API 的基本概念，工作流程和接口描述。正是由于 GSS-API 的可移植性和其它一些特性，本文在 GSS-API 上开发网络安全服务。

第 4 章 底层安全机制

GSS-API 被一定范围的底层机制所支持，这里简单介绍几种常用的底层安全机制。

4.1 Kerberos

Kerberos 是一种验证协议，它是建立在已经得到广泛支持的常规加密的基础之上，现在已经应用到各种系统上。现代的 Kerberos 有 3 个组成部分，用来守护网络的大门：身份验证、记帐和审核。后两个组成还没有实现。Kerberos 提出的问题是：假定一个开放的分布环境，其中工作站的用户想要从分布在网络中的服务器上获得服务。我们希望服务器能够限定仅能被授权用户访问，能够验证服务的请求。在此环境中，工作站不能够确保它的用户就是网络服务的正确的用户。特别是存在着以下 3 种威胁：

1．用户可以访问特定的工作站并伪装成其它工作站的用户。

2．用户可以改动工作站的网络地址，这样，改动过的工作站发出的请求就像是从伪装工作站发出的一样。

3．用户可以根据交换窃取消息，并使用重放攻击来进入服务器或破坏操作。

在以上任何一种情况下，未授权用户有可能获得他没有权利访问的服务或数据。与每个服务器上详细的身份验证协议相比，Kerberos 提供了集中的身份验证服务器，它的功能是用来对服务器验证用户以及对用户验证服务器。Kerberos 只依赖于常规加密，根本不用公钥加密。

有两个版本的 Kerberos 是最常用的。版本 4 仍然广泛使用，版本 5 更正了版本 4 中的一些安全缺陷，已经发布为 Internet 提议标准（Proposed Internet Standard）。

4 .1 .1 Kerberos 5 的身份验证对话

表 4－1

（a） 身份验证服务交换：获得票据授予的票据

（1） C → AS: Options ‖ ID_c ‖ $Realm_c$ ‖ ID_{tgs} ‖ Times ‖ $Nonce_1$

（2） AS → C: $Realm_c$ ‖ ID_c ‖ $Ticket_{tgs}$ ‖ $E_{kc}[K_{c,tgs}$ ‖ Times ‖ $Nonce_1$ ‖ $Realm_{tgs}$ ‖ $ID_{tgs}]$

$Ticket_{tgs}=E_{ktgs}[Flags ‖ K_{c,tgs} ‖ Realm_c ‖ ID_c ‖ AD_c ‖ Times]$

(b)票据授予的服务交换：获得服务授予的票据

（3） C → TGS: Options ‖ ID_v ‖ Times ‖ $Nonce_2$ ‖ $Ticket_{tgs}$ ‖ $Authenticator_c$

(4) TGS → C: $Realm_c$ ‖ ID_c ‖ $Ticket_v$ ‖ $E_{kc,tgs}[K_{c,v}$ ‖ Times ‖ $Nonce_2$ ‖ $Realm_v$ ‖ $ID_v]$

$Ticket_{tgs}=E_{ktgs}[Flags ‖ K_{c,tgs} ‖ Realm_c ‖ ID_c ‖ AD_c ‖ Times]$

$Ticket_v=E_{Kv}[Flags ‖ K_{c,v} ‖ Realm_c ‖ ID_c ‖ Times]$

$Authenticator_c =E_{Kc,tgs}[ID_c ‖ Realm_c ‖ TS_1]$

(c) 客户机/服务器身份验证交换；获得服务

（5） C → TGS: Options ‖ $Ticket_v$ ‖ $Authenticator_c$

（6） TGS → C: $E_{Kc,v}[TS_2 ‖ Subkey ‖ Seq\#]$

$Ticket_v=E_{Kv}[Flags ‖ K_{c,v} ‖ Realm_c ‖ ID_c ‖ Times]$

$Authenticator_c =E_{Kc,tgs}[ID_c ‖ Realm_c ‖ TS_1 ‖ Subkey ‖ Seq\#]$

其中：

C = 客户机

AS = 身份验证服务器

TGS = 票据授予服务器

V = 服务器

AD_c = C 的网络地址

‖ = 连接

基本原理

(a) 身份验证服务交换

Message(1)	客户机请求票据授予的票据
Options	请求在返回票据中设置一些标记
ID_c	告诉 AS 来自客户机的用户身份
$Realm_c$	告诉 AS 客户机所在的域
ID_{tgs}	告诉 AS 用户请求访问 TGS
Times	客户机用来请求在票据中进行的时间设置: (1)起始时间：请求票据的开始时间 (2)结束时间：请求票据的期满时间 (3)更新时间：要求更新的时间
$Nonce_1$	现时值：在 Message(2)中重复的随机数值，用来保证回应是最新的，没有被对手重放。
Message(2)	AS 返回票据授予的票据
$Realm_c$	确认客户机所在的域
ID_C	确认客户机的身份
$Ticket_{tgs}$	客户机访问 TGS 要使用的票据
E_{kc}	加密以用户的口令为基础，使 AS 和客户机验证口令，并保护消息（2）中的内容
$K_{c,tgs}$	可访问客户机的会话密钥的副本，由 AS 创建，用来允许客户机与 TGS 之间的保密交换，而不需要它们共享一个永久密钥
Times	客户机要求的时间设置
$Nonce_1$	现时值

$Realm_{tgs}$	确认 TGS 所在的域
ID_{tgs}	确认此票据是用于 TGS 的

票据本身包括了识别客户机信息的会话密钥、请求的时间值、反应票据状态度的标记和请求的选项。

(b) 票据授予的服务交换

Message(3)：	客户机请求服务授予的票据
Options	请求在返回票据中设置一些标记
ID_v	告诉 TGS 用户要访问的服务器 V
Times	客户机要求的时间设置
$Nonce_2$	现时值
$Ticket_{tgs}$	向 TGS 保证此用户已经由 AS 进行过身份验证
$Authenticator_c$	由客户机生成，用来验证票据
Message(4)：	TGS 返回服务授予的票据
$Realm_c$	确定客户机 C 所在的域
ID_C	确定客户机的身份
$Ticket_V$	客户机访问服务器 V 使用的票据
$E_{kc,tgs}$	只由 C 和 TGS 共享的密钥，保护 Message(4)中的内容
$K_{c,v}$	可访问客户机的会话密钥的副本，由 TGS 创建，用以允许保护客户机和服务器之间的交换，而不需要它们共享一个永久的密钥
Times	客户机要求的时间设置
$Nonce_2$	现时值
$Realm_v$	确认服务器 V 所在的域
ID_v	确认此票据用于服务器 V
$Ticket_{tgs}$	可重用，以便用户不用重新输入口令
E_{Ktgs}	使用只有 AS 和 TGS 知道的密钥加

	密的票据，用以防止被篡改
Flags	在返回票据中设置的一些标记
$K_{c,tgs}$	可访问 TGS 的会话密钥的副本，用于解密身份验证码，因而验证票据
$Realm_c$	指出客户机所在的域
ID_C	指出此票据的合理拥有者
AD_C	防止票据由不是初始请求票据的工作站使用
Times	一些时间设置
$Authenticator_c$	向 TGS 保证票据的推荐者与发出票据的客户机相同其生命周期非常短，以防重放
$E_{Kc,tgs}$	使用只有客户机和 TGS 知道的密钥来加密身份验证码，以防被篡改
ID_C	必须与票据中的 ID 相匹配，以验证票据
$Realm_c$	必须与票据中的 Realm 相匹配，以验证票据
TS_1	通知 TGS 生成此身份验证码的时间

(c) 客户机/服务器身份验证交换

Message(5)	客户机请求服务
Options	请求在返回票据中设置一些标记
$Ticket_v$	向服务器保证此用户已经由 AS 进行了身份验证
$Authenticator_c$	由客户机生成，用于验证票据
Message(6)	服务器对客户机可选的身份验证
$E_{Kc,v}$	向 C 保证此消息是来自 V
TS_2	向 C 保证这不是重放旧的应答
Subkey	子密钥。客户机对加密密钥的选择可以保护这种特定的应用会话。如果忽略了此字段，就使用来自票据（$K_{c,v}$ ）的会话密钥

Seq#	序列号。指定起始序列号的可选字段是服务器用来在会话中向客户机发送消息的，这些消息可以标上序列号来检测重放
$Ticket_v$	可重用，以便对于相同服务器的每个访问，客户机不需要向 TGS 请求新的票据
E_{Kv}	票据只使用 TGS 和服务器知道的密钥进行加密，以防止被篡改
Flags	客户机请求设置的一些标记
$K_{c,v}$	可访问客户机的会话密钥的副本，用于解密身份验证码，因而验证票据
$Realm_c$	指出客户机所在的域
ID_C	指出此票据的合理拥有者
Times	一些时间设置
$Authenticator_c$	向 TGS 保证票据的推荐者与发出票据的客户机相同生命周期非常短，以防重放
$E_{Kc,tgs}$	身份验证码使用只有客户机和服务器知道的密钥进行加密，以防止篡改
ID_C	必须与票据中的 ID 相匹配，以验证票据
$Realm_c$	必须与票据中的 Realm 相匹配，以验证票据
TS_1	通知服务器，此验证码被生成的时间
Subkey	子密钥。客户机对加密密钥的选择可以保护这种特定的应用会话。如果忽略了此字段，就使用来自票据（$K_{c,v}$）的会话密钥
Seq#	序列号。指定起始序列号的可选字段是服务器用来在会话中向客户机发送消息的，这些消息可以标上序列号来检测重放

1．用户登录到工作站，并请求主机上的服务。
2． AS 在数据库中验证用户的访问权，创建票据授权的票据和会话密钥。使用从用户口令中导出的密钥对结果加密。
3． 工作站提示用户输入口令，并使用口令来解密传入的消息，然后发送票据和认证码，其中包含用户名、网络地址和 TGS 的时间。

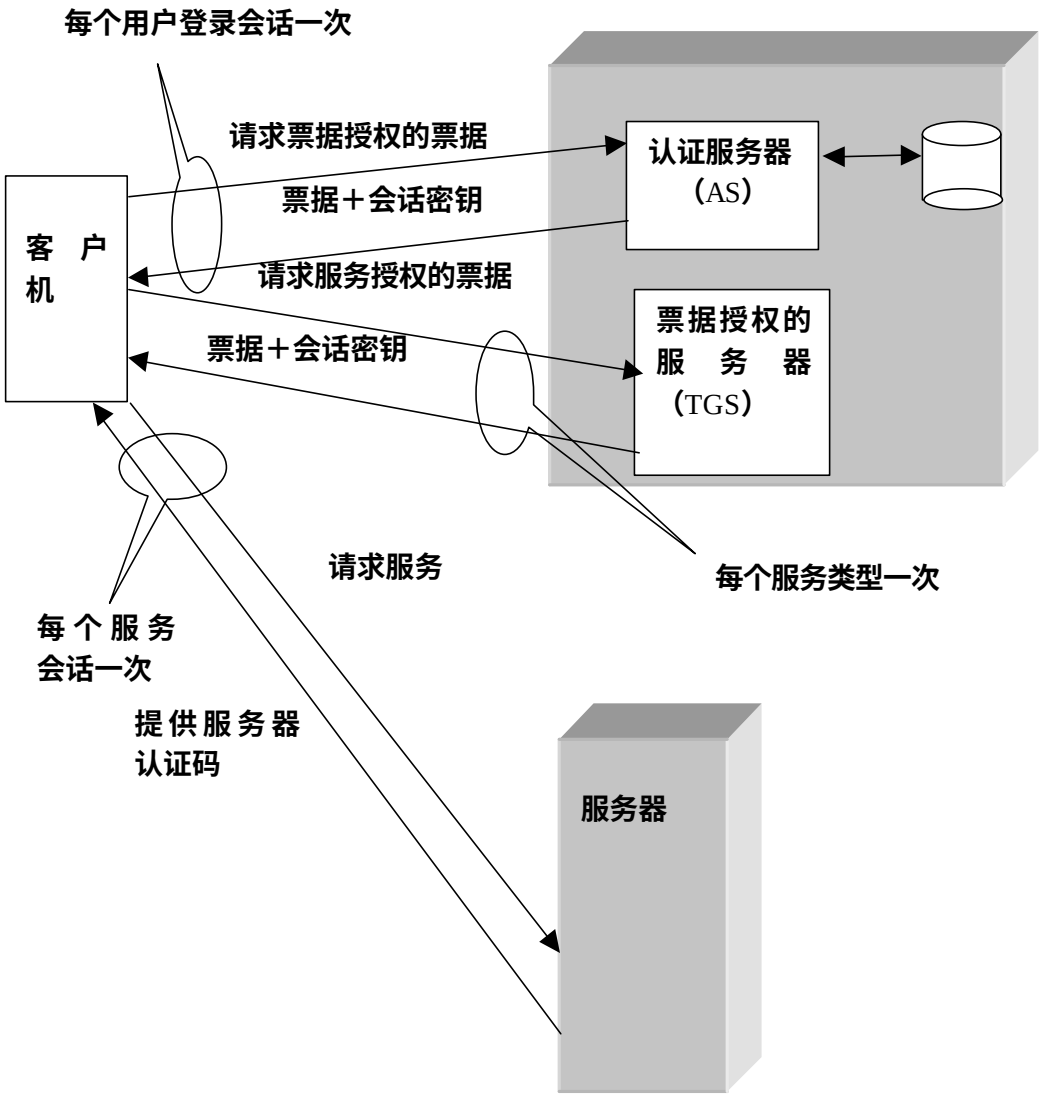


图 4-1Kerberos 概述

4． TGS 解密票据和认证码，验证请求，然后为请求的服务器创建票据。
5．工作站对服务器发送票据和认证码。
6． 服务器验证票据和认证码是否匹配，然后授予服务的访问。如果需要相互的认证；则服务器返回一个认证码。

4 .1 .2 Kerberos 5 与 GSS-API 的结合

kerberos 安全上下文的数据结构

```
struct _krb5_context {
    krb5_magic    magic;
    krb5_enctype  FAR *in_tkt_ktypes;    AS 请求加密密钥类型
    int           in_tkt_ktype_count;    AS 请求加密密钥类型数量
    krb5_enctype  FAR *tgs_ktypes;       TGS 请求加密密钥类型
    int           tgs_ktype_count;       TGS 请求加密密钥类型类型
    char          FAR *default_realm;    缺省区域
    profile_t     profile;               初始配置文件
    void          FAR *db_context;       与数据库相关的上下文
    int           ser_ctx_count;         服务器的上下文数量
    void          FAR *ser_ctx;          服务器的上下文
    krb5_deltat   clockskew;             可误差时间
    krb5_cksumtype kdc_req_sumtype;      KDC 请求校验类型
    krb5_flags    kdc_default_options;   KDC 缺省选项
    krb5_flags    library_options;       数据库选项
    krb5_Boolean  profile_secure;        身份选用安全选项
      int     fcc_default_format;   文件缓存的缺省格式
      int     scc_default_format;       缓存的缺省格式
     #ifdef   KRB5_DNS_LOOKUP
```

```
        krb5_boolean    profile_in_memory;是否支持 DNS
    #endif /* KRB5_DNS_LOOKUP */
};
krb5_creds
typedef struct _krb5_creds {
        krb5_magic magic;
        krb5_principal client;              客户端主体
        krb5_principal server;              服务器端主体
        krb5_keyblock keyblock;             会话密钥
        krb5_ticket_times times;            票根生存期
        krb5_boolean is_skey;               /*true    if
ticket is encrypted in another ticket's skey */
        krb5_flags ticket_flags;            票根属性
        krb5_address FAR * FAR *addresses;票根中主机地
                                            址
        krb5_data ticket;                   票根数据部分
        krb5_data second_ticket;
/*      second      ticket,      if      related      to
ticket (via DUPLICATE-SKEY or ENC-TKT-IN-SKEY) */
        krb5_authdata FAR * FAR *authdata;  认证数据
} krb5_creds;
```

4.2 公钥密码

公钥加密最初是由 Diffie 和 Hellman 在 1976 年提出的，是几千年来文字加密的第一次真正革命性的进步。因为公钥是建立在数学函数基础上的，而不是建立在位方式的操作上的。更重要的是，公钥加密是不对称的，与只使用一种密钥的对称常规加密相比，它涉及到两种独立密钥的使用。这两种密钥的使用已经对机密性、密钥的分发和身份验证领域产生了深远的影响。

4 .2 .1 公钥加密的组成与步骤

公钥加密方案由 6 个部分组成：

1. 明文：作为算法输入的可读消息或数据。

2． 加密算法：加密算法对明文进行各种各样的转换。
3． 公共的和私有的密钥：选用的一对密钥，一个用来加密，一个用来解密。解密算法进行的实际转换取决于作为输入提供的公钥或私钥。
4． 密文：作为输出生成的杂乱的消息。它取决于明文和密钥。对于给定的消息，两种不同的密钥会生成两种不同的密文。
5． 解密算法：这种算法以密文和对应的密钥为输入，生成原始明文。

顾名思义，密钥对中的公钥是要公开使用的，而私钥只有所有者知道。通常公钥加密算法在加密时使用一个密钥，在解密时使用一个密钥。

基本步骤如下：

（1） 每个用户都生成一对加密和解密时使用的密钥。

（2） 每个用户都在公共寄存器或其它可访问的文件中放置一个密钥，这个就是公钥。另一个密钥为私钥。每个用户都要保持从他人那里得到的公钥集合。

（3） 如果 A 要向 B 发送私有消息，A 可以用 B 的公钥加密消息。

（4） 当 B 受到消息时，他可以用自己的私钥进行解密。

其他接收方不能解密消息，因为只有 B 知道他自己的私钥。

用这种方法，所有的参与者都可以访问公钥，而生成的私钥却由每个参与者个人生成并拥有，不需传送。只要用户能够保护好他的私钥，接收到的消息就是安全的。用户可以随时改变私钥并发布新的公钥来替换旧公钥。

4 .2 .2 公钥密码系统的应用

公钥系统的特点是：使用了两种密钥的密码术类型的算法，一个是个人拥有的私钥，一个是大众可用的公钥。依据应用要求的不同，发送方或者使用发送方的私钥，或者使用接收方的公钥，或者两者都用，来运行某种类型的密码函数。广义上，可以将公共加密系统的使用分为 3 种：

1． 加密/解密：发送方可以用接收方的公钥加密消息。
2． 数字签名：发送方用其私钥“ 签署 ”消息，通过对消息或作为消息函数的小块数据应用加密算法来进行签署。
3． 密钥交换：两方互相合作可以进行会话密钥的交换。

4.3 一次性口令认证系统（OTP）

4 .3 .1 简介

一次性口令认证系统（one-time password authentication system OPT）为系统访问提供了认证，同时也为其它应用程序提供了防止以重放捕获重用口令为基础的被动攻击所需要的认证。

在联网的计算机系统中一种形式的攻击就是在网络连接上窃听以获取像登录 ID 和合法用户的口令这样的认证信息。一旦这些信息被捕获，就会被重用以访问系统，一次性口令认证系统的设计正是阻止了这种重放攻击。该系统利用一个秘密通行短语来产生一个一次性口令队列。在该系统中用户秘密通行短语无需在网络上传输无论是在认证的时候还是在修改时候，因此不会受到重放攻击。

4 .3 .2 工作原理

在一次性口令系统的操作中有两个实体，发起者必须从用户的秘密通行短语和服务器的 challenge 中提供的信息中产生一个合适的一次性口令。服务器必须发送一个包含合适的生成参数的 challenge 给发起者，必须验证接收的一次性口令并且必须存储它接收的上一次有效的一次性口令以及相应一次性口令的序列号。

OTP 系统的发起者将用户的秘密通行短语同从服务器接收到作为 challenge 的一部分的种子一起经过安全杂凑函数的多次迭代产生一个一次性口令。在每次成功的认证以后，杂凑函数迭代的次数减一，因此一个唯一点口令序列就产生了。服务器将从发起者处接收的一次性口令经过杂凑函数的一次计算的结果同以前收到的一次性口令相比较来进行验证。

4.4 本章小结

本章主要介绍了适用于 GSS-API 的几种底层机制,Kerberos 5 的概念及工作原理，以及同 GSS-API 的结合。公钥系统的概

念和应用。一次性口令认证的概念和工作原理。本文采用了一次性口令验证系统。

第 5 章 应用实现

5.1 应用背景

在为河南省驻马店市中原气化集团开发的基于网络的管理和财务核算系统中，需要通过远程的数据传输以进行财务核算，实现实时的统一管理。在远程的数据传输中，为了防止非法用户和恶意破坏者的行为对系统或集团的利益造成不必要的损失，需要采取相应的安全措施。

5.2 应用方案

由于 GSS-API 的优点就在于它能够使编程者编制的应用程序具有通用性，它的安全处理不依赖于任何特定的操作平台、安全机制、安全服务的类型以及传输协议，所以本文决定将 GSS-API 与底层安全机制相结合，来实现相应的网络安全服务。目前，主要解决了两个问题，也即身份验证和数据完整性，在将来可以方便地采用更加强有力的安全机制和更丰富的安全服务。

身份验证采用了一次性口令验证方式，采用单向函数 MD5 算法进行完整性校验。

5.3 实现

5 .3 .1 总体结构

总体结构如图 5-1 所示。

5 .3 .2 工作过程

1． 启动客户端程序登录服务器，自动获取一个信任状，信任状中包括用户名和口令。
2． 客户端调用 GSS_Init_sec_context（）初始化一个安全上下文。在此过程中客户端给服务器端发一个 challenge request。
3． 此时，服务器端建立一个信任状。并且接收客户端的安全上下文。并回应给客户端一个 challenge。
4． 客户端收到 challenge 后，计算它的一次性口令，并将其发给服务器。
5． 服务器端对口令进行验证，并将结果返回给客户端。
如果验证成功，是合法用户，下面就进入数据传输阶段，否则的话，重新登录。
6． 客户端调用 GSS_GetMIC（）函数形成信息完整性编码。
7． 调用 GSS_Wrap()封装数据，发送给服务器。
8． 服务器收到封装的信息以后，调用 GSS_UnWrap()将数据解开封装。
9． 服务器调用 GSS_VerifyMIC（）来对接收到信息进行校验，根据返回值来判断接收到数据是否有效，如有效则进行应用程序的处理，否则丢弃。

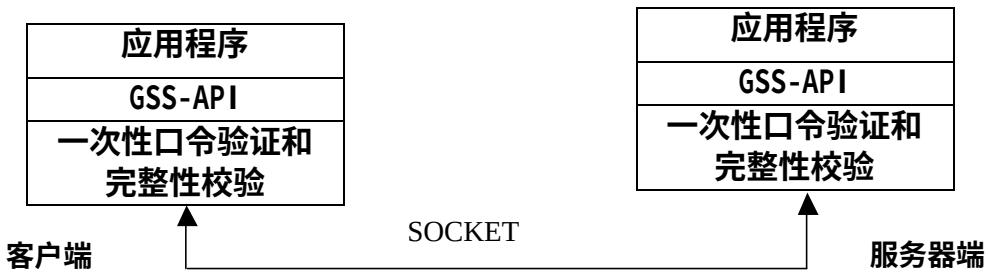


图 5-1 设计模型

5.4 本章小结

本章主要介绍了基于 GSS-API 的网络安全服务的具体设计与实现。

第 6 章 结束语

计算机系统及网络相互连接的爆炸性增长增加了组织机构和个人对使用这些系统进行信息存储和信息交流的依赖性。这也必然会带来人们对保护数据和资源以防泄漏的高度重视，保证数据和信息的可靠性，保护系统免受网络攻击。其次密码术和网络安全的规则已经成熟，导致实际的、即时可用的应用程序的开发，来加强管理。

由于 GSS-API 的优点就在于它能够使编程者编制的应用程序具有通用性，它的安全处理不依赖于任何特定的操作平台、安全机制、安全服务的类型以及传输协议，所以本文决定将 GSS-API 与底层安全机制相结合，来实现相应的网络安全服务。目前，主要解决了两个问题，也即身份验证和数据完整性，在将来可以方便地采用更加强有力的安全机制和更丰富的安全服务。

摘要

计算机系统及网络相互连接的爆炸性增长增加了组织机构和个人对使用这些系统进行信息存储和信息交流的依赖性。这也必然会带来人们对保护数据和资源以防泄漏的高度重视，保证数据和信息的可靠性，保护系统免受网络攻击。其次密码术和网络安全的规则已经成熟，导致实际的、即时可用的应用程序的开发，来加强管理。在为河南省驻马店市中原气化集团公司开发的基于网络的管理与财务核算系统的开发过程中，需要通过远程的数据传输得到原始数据，以进行财务核算，实现实时的统一管理。在远程的数据传输中，为了防止非法用户和恶意破坏者的行为对系统或集团的利益造成不必要的损失，需要采取相应的安全措施。本文正是根据这一需要，设计实现了一种基于 GSS-API 的安全服务模型，为应用程序提供了最大的可移植性。

安全服务包括保密性、身份验证、完整性、不可否认性、访问控制、可用性。身份验证服务致力于保证信息的可靠性。在只有一条消息的情况下，如警告和警报信号，验证服务的功能就是要保证信息接收方接收的消息确实是从它声明的来源发出的。在进行交互的过程中，如将终端连接到主机上，需要牵涉到两个方面。首先，在连接的初始化阶段，此服务应保证两个实体的可靠性（也就是说，每个实体都是所声明的实体）；其次，此服务还应保证第三方不能靠伪装成两个合法实体中的一个来干扰连接，执行未授权的传送或接收。

与保密性一样，完整性可以应用到信息流、单个信息或消息中选定的字段。而且，最有用和最直接的保护方式是对整个信息流保护。

面向连接的完整性服务可以处理消息流，保证接收与发出的内容一致，没有经过复制、插入、修改、记录或重放。数据破坏也属于本服务的管辖范围。因此，面向连接的完整性服务可以解决信息流修改和拒绝服务的问题。另一方面，无连接的完整性服务只适于处理与其他大型文本无任何关系的单个信息，通常提供对信息修改的保护。

通用安全服务应用程序接口（GSS-API）为保护对端应用程序之间传递的数据提供了一种方法。这种传输最典型的模型就是一台主机上的客户端和另一台主机上的服务器。而 GSS-API 的优点

就在于它能够使编程者编制的应用程序具有通用性，它的安全处理不依赖于任何特定的操作平台、安全机制、安全服务的类型以及传输协议。虽然 GSS-API 使应用程序能够控制安全方面的事务，但是应用 GSS-API 的编程者可以忽略保护网络数据的一些细节。因此应用 GSS-API 具有良好的可移植性。这种可移植性是通用安全服务 API 的最重要的特点。

实际上，GSS-API 本身并不提供安全服务，它是一种以通用的方式为调用者提供安全服务的框架结构，并且被一定范围的底层机制和技术所支持，例如 Kerberos v5 或公共密钥技术。

总的来讲，GSS-API 主要做了以下两件事情：

1．创建了安全上下文，数据在安全上下文中传递于两个应用程序之间。一个安全上下文可以在两个应用程序之间被认为是一种可信任的状态。共享同一个安全上下文的两个应用程序相互知道对方的身份，并且因此在安全上下文存在期间，允许彼此间数据的传输。

2．为要传输的数据提供了一种或多种保护类型，也就是安全服务。

当然 GSS-API 所能够做的远比这些复杂，它的其它功能包括：数据转换；错误检测；用户权力的授权；信息显示以及身份比较。GSS-API 包括许多支持或转换的功能。

GSS-API 所提供的最基本的安全服务是鉴别。鉴别就是身份的验证。如果底层机制支持，那么 GSS-API 将提供另外两种安全服务：也即完整性和私密性。

本文决定将 GSS-API 与底层安全机制相结合，来实现相应的网络安全服务。目前，主要解决了两个问题，也即身份验证和数据完整性，在将来可以方便地采用更加强有力的安全机制和更丰富的安全服务。

身份验证采用了一次性口令验证方式，采用单向函数 MD5 算法进行完整性校验。

工作过程如下：

（1）启动客户端程序登录服务器，自动获取一个信任状，信任状中包括用户名和口令。

（2）客户端调用 GSS_Init_sec_context（）初始化一个安全上下文。在此过程中客户端给服务器端发一个 challenge request。

（3）此时，服务器端建立一个信任状。并且接收客户端的安全上下文。并回应给客户端一个 challenge。

（4）客户端收到 challenge 后，计算它的一次性口令，并将其发给服务器。

（5）服务器端对口令进行验证，并将结果返回给客户端。如果验证成功，是合法用户，下面就进入数据传输阶段，否则的话，重新登录。

（6）客户端调用 GSS_GetMIC（）函数形成信息完整性编码。

（7）调用 GSS_Wrap()封装数据，发送给服务器。

（8）服务器收到封装的信息以后，调用 GSS_UnWrap()将数据解开封装。

（9）服务器调用 GSS_VerifyMIC（）来对接收到信息进行校验，根据返回值来判断接收到数据是否有效，如有效则进行应用程序的处理，否则丢弃。

Abstract

Recent years, with the rapid development of computer system and the connection of internet, the dependency on information's storage and information's intercommunion which organizations and individual used the system to realize had increased. It makes people to attach importance to data and information's protect from being leaked. Secondly encryption and the regulation of network security has been mature. Under this condition, development of actual and instant used applications is needed to strengthen the management. In the course of the development of the network-based management and finance computing system for Midland Qing Hua Group Corporation in He Nan Province Zhu Ma Dian City, remote data transmit is needed to compute the original data and realize instant and consolidated management. In the period of remote data transmit, security measure must be adopted to prevent illegal users and spiteful destroyers from doing harm for the system and the benefit of the group. For this request, a security service model based on GSS-API has been designed in this thesis in order to provide the application portability.

Security service includes confidentiality, authentication, integrity, nonrepudiation, access control and usability. Authentication is use to ensure the reliability of the information. In the condition of there is only one piece of message such as admonition or admonition signal, Authentication service is used to ensure the information the accepter had received is from the origin which it had declared. In the course of the alternating , for example, connect terminal to the mainframe ,it need two sides. Firstly, at the stage of initialization of the connection, the service assure the two entities' dependability; secondly, the service assure another side can not pretend to be one of the two legal entities to disturb

the connection or perform the transmit or accept that are not authorized.

Just like the confidentiality service , Integrity can be used in message stream, single message or the fields chosen in the message. And that the most useful and direct protect fashion is to protect the whole message stream.

Link oriented Integrity service can deal with message stream in order to ensure the content received according to the content sent without copying, inserting, modifying , recording and replaying. This service also can avoid disturbing data. Therefore link oriented integrity service can solved the problems of modifying message stream and rejecting service. On the other side, non-link integrity service only adapts to perform the single message the has nothing with other great text, and commonly protect the message from being modified.

The Generic Security Standard Application Programming Interface (GSS-API) provides a way for applications to protect data that is sent to peer applications; typically, this might be from a client on one machine to a server on another. As its name implies, the GSS-API enables programmers to write applications that are generic with respect to security; that is, they do not have to tailor their security implementations to any particular platform, security mechanism, type of protection, or transport protocol. Although the GSS-API enables applications control over security aspects, a programmer using GSS-API can write a program that is ignorant of the details of protecting network data. Therefore, a program that takes advantage of GSS-API is more portable as regards network security. More than anything else,this portability is the hallmark of the Generic Security Standard API.

The GSS-API does not actually provide security services itself. Rather, it is a framework that provides security services to callers in a generic fashion, supportable with a range of underlying mechanisms and technologies such as Kerberos v5 or public key technologies。

Broadly speaking, the GSS-API does two main things:
1. It creates a security context in which data can be passed between applications. A context can be thought of as a sort of "state of trust" between two applications. Applications that share a context know who each other are and thus can permit data transfers between them as long as the context lasts.
2. It applies one or more types of protection, known as security services, to the data to be transmitted.

Of course, the GSS-API is more complex than that. Some of the other things that the GSS-API does include: data conversion; error checking; delegation of user privileges; information display; and identity comparison. The GSS-API includes numerous support or convenience functions.

The most basic security service that GSS-API provided is Authentication. GSS-API also can provide the other two type of security service if underlying mechanism sustains: confidentiality and Integrity .

In order to realize the corresponding network security service , GSS-API was combined with the underlying mechanism in this thesis. Now it mainly solved two problem, integrity and authentication. More strongly security mechanism and more abundant security service can be adopted expediently in the future.

One time password (OTP) validation mode is used to achieve authentication, and an unilateralism function MD5 arithmetic is used to achieve integrity validation.

Work procedure as following:

(1) Start up application login server in client, credentials which include username and password are acquired automatically .

(2) Client sent a challenge request to server while client calls GSS_Init_sec_context () to initiates a security context.

(3) At the same time credentials are established in server and accept the security context from the client. After that server responses a challenge to client.

(4) After client had received the challenge, it calculates its one time password and pass it to the server.

(5) Server validates the password and return the result to the client.

If it is legal user, data transmission begins, otherwise login again.

(6) Client calls GSS_GetMIC () to generate information integrality code.

(7) Client calls GSS_Wrap() to wrap data and sent it to server.

(8) Server calls GSS_UnWrap() to unwrap the data after it received the wraped data.

(9) Server calls GSS_VerifyMIC () to verify the information received. Server judges that if the data is available from the return value. If the data are available server will perform the application ,otherwise ,data will be casted off.

参考文献

1. ISBN 7-111-07588-9，Bruce Schneier,《应用密码学》，机械工业出版社，第 1 版，2000 年
2. ISBN 7-115-08791-1/TP·1826，William Stallings,《网络安全要素——应用与标准》，人民邮电出版社，第 1 版，2000 年
3. ISBN 7-115-08813-6/TP·1847，Merike Kaeo,《网络安全性设计》，人民邮电出版社，第 1 版，2000 年
4. “ Generic Security Service Application Program Interface, Version 2 ”, John Linn , OpenVision Technologies
5. [RFC1510] “ The Kerberos Network Authentication Service (V5) ” ,J. Kohl& C. Neuman，1993 年 9 月
6. [RFC1964] “ The Kerberos Version 5 GSS-API Mechanism ” , J. Linn，OpenVision Technologies,1996 年 6 月
7. [RFC2744] “ Generic Security Service API Version 2: C-bindings ” ,J. Wray，Iris Associates,2000 年 1 月
8. [RFC2473] “ Generic Security Service Application Program Interface Version 2,Update 1 ” ,J.Linn，RSA Laboratories ,2000 年 1 月

致　谢

首先要衷心感谢我的导师刘淑芬教授，在刘老师的悉心指导和帮助下，本文才得以完成。三年来，刘老师的言传身教使我获益匪浅；她在生活上给予我的无微不至的关怀令我感激不尽；她对知识的不懈追求，对科学的严谨态度，对工作的一丝不苟，对事业的执着更是值得我毕生学习。

其次，我要向本教研室的李庆新、李兵两位老师及韩兵、陈志雨、金浩、姚志林、方英兰、冯伟等几位同学表示我诚挚的谢意，感谢他们在我三年研究生学习期间对我的关心和帮助。

此外，还要感谢计算机系的高瑛老师和计算机系的硕士生李强同学感谢他们对我的鼓励和帮助。

感谢我的父母对我成长的关怀和支持。

最后，谨向其他所有帮助我的老师、同学们表示我最为衷心的谢意。