摘 要

在数字化时代，网络与信息安全的重要性与日俱增。安全服务（如认证、完整性和机密性等）已经成为数字化世界和网络应用中的一种最基本的服务。

公开密钥基础设施 PKI 是网络安全中的一项重要技术，它为安全服务提供基础框架，正成为整个安全体系结构的核心部分。基于PKI技术的CA认证中心及数字证书是目前互联网上各类安全应用系统的主要密钥管理方式，能很好地提供网络应用中的认证、加解密的密钥管理、数字签名等服务，是当前网络安全解决方案中比较有效可靠的方案之一。

密码学和加解密技术是 PKI 系统中的基础和核心之一。

论文介绍了 PKI 基本结构、功能及相关协议，CA 认证中心和数字证书的相关知识，密码学的基础理论。

论文详细阐述了 PKI 中常用的密码学算法 MD5、DES、RSA 的实现，base64 编解码的实现，以及 PKI 中加密/解密、数字签名、消息摘要、数字信封的实现。

论文对安全套接字层（SSL）协议进行了详细的分析，SSL 是 Internet 网络的一个安全通信协议，为通信双方建立秘密、可靠的连接方式，并提供防窃听、防篡改、防信息伪造的秘密通信手段。文章的最后提出了利用 OPENSSL 开放源代码实现 SSL 协议的方法。

关键词：公钥基础设施，加解密，安全套接层协议

Abstract

In the digital times, the importance of the network and information security is greater and greater. Security service, such as authentication, integrity and secrecy, has become the most essential service of digital world and network application.

PKI (Publice Key Infrastructure) is a kind of important network security technology, which is the basic framework of security framework. CA (Certificate Authority) and Certificate based on PKI is the main pivotal management scheme, which can offer the service such an authentication, key management and digital signature. It is one of the most affective solutions on network security.

Cryptography and encrypt/decrypt algorithm are one of the basises and keys of PKI system.

This article introduces the basic structure, function of PKI and the protocols involved, and gives a presentation of the knowledge about Certificate Authority, certification and cryptography.

This article specifies the realization of encrypt/decrypt algorithm used in PKI, includeing DES, MD5, RSA and Base64 Data Encodings, and the realization of cryptogram technology used in PKI such as encrypt and decrypt, digital signature, message digest, digital envelop.

This article analyzes the Secure Sockets Layer (SSL) protocol. SSL protocol is a security protocol that provides communications privacy over the Internet. The protocol provides privacy and reliability between two communicating applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery. At the final, the author provides the way realize SSL using OPENSSL.

Key words: PKI, Encrypt/Decrypt , SSL

第一章 PKI 概述及相关协议

1.1 公开密钥基础设施的定义

公开密钥基础设施（Public Key Infrastructure，简称 PKI）是一个用非对称密码算法原理和技术来实现并提供安全服务的具有通用性的安全基础设施，是一种遵循标准的利用公钥加密技术为网上电子商务、电子政务的开展提供一整套安全的基础平台。用户利用 PKI 平台提供的安全服务进行安全通信。PKI 这种遵循标准的密钥管理平台，能够为所有网络应用透明地提供采用加密和数字签名等密码服务所需要的密钥和证书管理。

用户使用基于公钥技术平台，建立安全通信信任机制的基础是：网上进行的任何需要提供安全服务的通信都是建立在公钥的基础之上的，而与公钥成对的私钥只掌握在它们与之通信的另一方。这个信任的基础是通过公钥证书的使用来实现的。公钥证书就是一个用户的身份与他所持有的公钥的结合，在结合之前，由一个可信任的权威机构 CA 来证实用户的身份，然后可信任的 CA 对该用户身份及对应公钥相结合的证书进行数字签名，以证明其证书的有效性。

PKI 首先必须具有可信任的权威认证机构 CA，在公钥加密技术基础上实现证书的产生、管理、存档、发放以及证书作废管理等功能，并包括实现这些功能的硬件、软件、人力资源、相关政策和操作规范以及为 PKI 体系中的各成员提供全部的安全服务。如：实现通信中各实体的身份认证、数据保密性、数字完整性以及不可否认性服务等。

PKI 必须具有认证机构 CA、证书库、密钥备份及恢复系统、证书作废处理系统，PKI 应用接口系统等主要组成部分。构建实施一个 PKI 系统也将围绕这些部分来进行。

（1） 认证机构——证书的签发机构，它是 PKI 的核心，是 PKI 应用中权威的、可信任的、公正的第三方机构。

（2） 证书库——是证书的集中存放地，提供公众查询。

（3） 密钥备份及恢复系统——对用户的解密密钥进行备份，当丢失时进行恢复，而签名密钥不能备份和恢复。

（4） 证书作废处理系统——证书由于某种原因需要作废，中止使用，这将通过证书作废列表 CRL 来完成。

（5） PKI 应用接口系统——是为各种各样的应用提供安全、一致、可信任的方式与 PKI 交互，确保所建立起来的网络环境安全可信，并降低管理成本。

1.2 公钥基础设施的内容

1.2.1 认证机构（Certificate Authority）

认证机构是 PKI 的核心组成部分，一般简称为 CA，在业界通常称为认证中心。它是数字证书的签发机构。

证书是公开密钥体制的一种密钥管理媒介，它是一种权威性的电子文档，形同网络计算环境中的一种身份证，用于证明某一主体（如：人、服务器等）的身份以及其公开密钥的合法性。在使用公钥体制的网络环境中，必须向公钥的使用者证明公钥的真实合法性。因此，在公钥体制环境中，必须有一个可信的机构来对任何一个主体的公钥进行公证，证明主体的身份以及它与公钥的匹配关系。CA 正是这样的机构。

1.2.2 证书库

证书库是 CA 颁发证书和撤消证书的集中存放地，它像网上的一种公共信息库，供广大公众进行开放式查询。

1.2.3 证书撤消

证书捆绑了用户的身份和公钥，必须存在一种机制来撤消这种捆绑关系，将现行的证书撤消。这种撤消的原因通常有：用户身份姓名的改变、私钥被窃或泄露、用户与其所属企业关系变更等。这样就必须存在一种方法警告其他用户不要再使用这个公钥。在 PKI 中，这种警告机制被称做证书撤消。所使用的手段为证书撤消列表或称 CRL。

证书撤消的实现方法有很多种。一种方法是在线查询机制，如在线证书状态协议 OCSP。

1.2.4 密钥备份和恢复

用户由于某种原因丢失了解密数据的密钥，则被加密的密文无法解开，造成数据丢失。为了避免这种情况的发生，PKI 提供了密钥备份与解密密钥的恢复机制。

1.2.5 自动更新密钥

一个证书的有效期是有限的，这样规定既有理论上的原因，又有实际操作的因素。在理论上诸如关于当前非对称算法和密钥长度的可破译性分析，同时在实际应用中，证明密钥必须有过期的措施，以便更换新的证书。这个过程被称为“密钥更新或证书更新”，它与证书作废撤消是两个概念。

1.2.6 密钥历史档案

在经过上述密钥更新概念的论述之后，我们很容易地会得出以下结论，即经过一段时间之后，每个用户都会形成多个“旧”证书和至少有一个“当前”的证书。这一系列旧证书和相应的私钥就组成了用户密钥和证书的历史档案，简称密钥历史档案。

1.2.7 交叉认证

在一个国家内或在全世界范围内，在一系列独立开发的 PKI 域中，至少其中有一部分互联是不可避免的。由于业务关系的改变或其他一些原因，不同 PKI 的用户团体之间必须进行安全通信。

为了以前没有联系的 PKI 之间建立信任关系，导致了“交叉认证”的概念。在没有一个统一的全球的 PKI 环境下，交叉认证是一个可以接受的机制，因为它能够保证一个 PKI 团体的用户验证另一个 PKI 团体的用户证书。

交叉认证是 PKI 中信任模型的概念。它是一种把以前无关的 CA 连接在一起的有用机制，从而使得它们在各自主体群之间实现安全通信。交叉认证的实际构成方法，就是具体的交换协议报文。

1.2.8 支持不可否认性

PKI 的不可否认性是用于从技术保证实体对他们行为的诚

实。通常指的是对数据来源的不可否认和接收后的不可否认，一个 PKI 用户经常执行与他们身份相关的不可否认的操作，如甲对一份文档进行数字签名申明该文档出自于他。

PKI 必须能够支持避免或组织这种否认——这就是不可否认性的特点。诚然，一个 PKI 本身无法提供真正的、完全的不可否认性服务。需要人为因素来分析、判断证据，并做出最后的抉择。然而，PKI 必须提供所需要的技术上的证据支持决策，提供数据来源认证和可信的时间戳数据的签名。

1.2.9 时间戳

时间戳或称安全时间戳，它是一个可信的时间权威用一段可认证的完整的数据表示的时间。最重要的不是时间本身的精确性，而是相关时间日期的安全性。支持不可否认性服务的一个关键因素就是在 PKI 中使用安全时间戳，就是说时间源是可信的，时间值必须被安全地传送。

1.2.10 客户端软件

客户端软件是一个全功能、可操作 PKI 的必要组成部分。熟悉客户/服务器结构的人都知道，只有客户端提出请求服务，服务器端才会为此请求做响应处理。这个原理同样适用于 PKI。客户端软件的功能有：

（1）询问证书和相关的撤消信息。
（2）在一定时刻为文档请求时间戳。
（3）作为安全通信的接收点。
（4）进行传输加密或数字签名操作。
（5）能理解策略，知道是何时和怎样去执行取消操作。
（6）证书路径处理等。

1.3 PKI 认证体系结构

目前，在 PKI 体系基础上建立起来的安全证书体系在世界各国得到了普遍关注和应用。为了使远程用户能相互信任和使用对方证书及公钥，各个国家以及国家与国家之间，需要构建大规模的、具有可伸缩性的 PKI 体系结构，使得寻找和检验路径非常方便。美国、加拿大等政府机构都提出了建立国家 PKI 体系的具体

实施方案。下面给出一种典型的 PKI 层次结构信任模式方案，如图 1-1 所示：

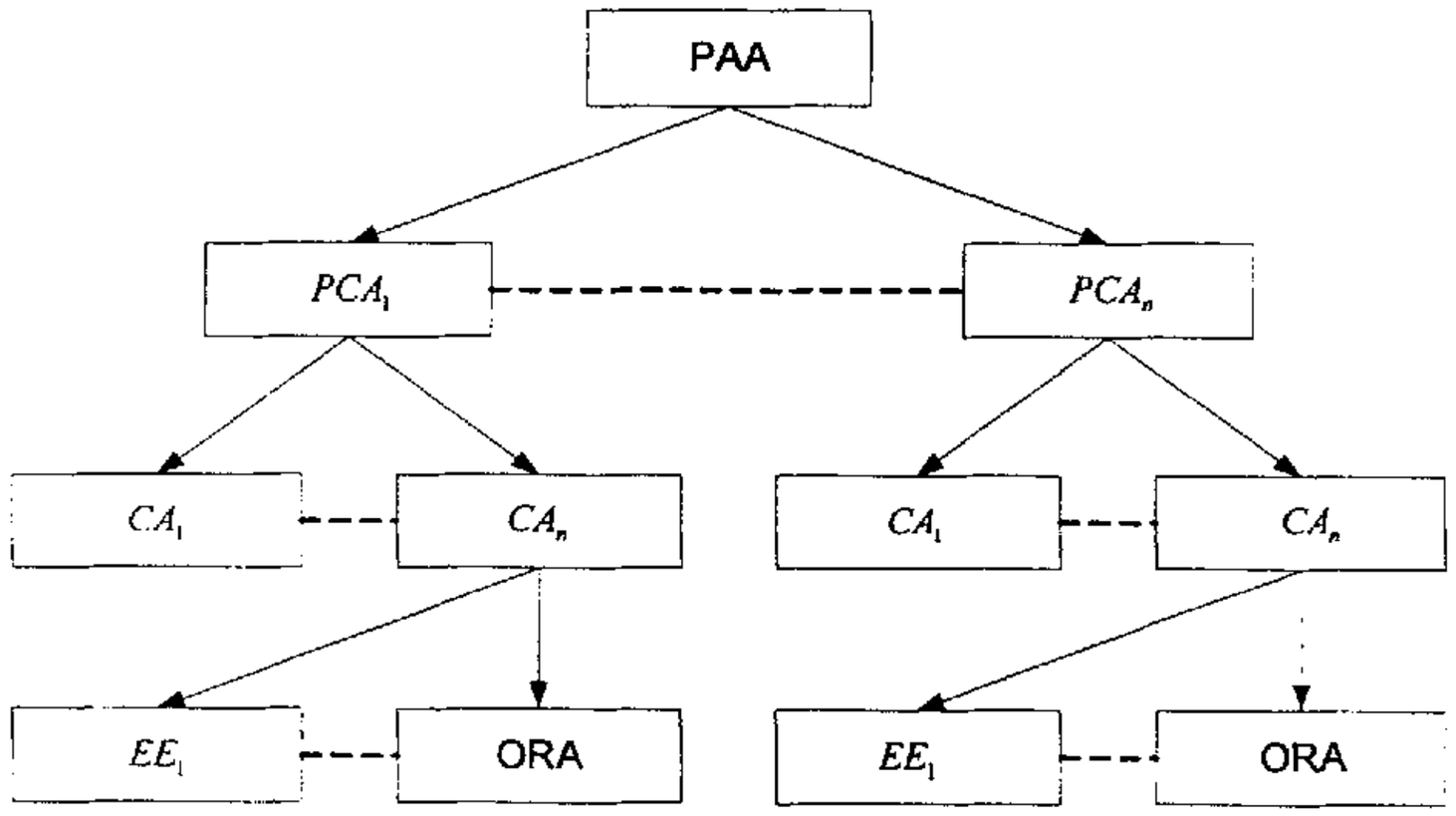


图 1-1 典型的 PKI 层次结构信任模式方案

层次信任体系结构中各实体功能描述如下：

1、政策批准机构 PAA

PAA 是政策批准机构，它创建各 PKI 系统的方针，批准 PAA 下属的 PCA 政策，为下属 PCA 签发公钥证书，建立整个 PKI 体系的安全策略，并监控各 PCA。

2、政策 PCA 机构

PCA 为政策 CA，制定本 PCA 的具体政策，可以是上级 PAA 政策的扩充或细化，但不能与之相背离。这些政策可能包括 PCA 范围内的密钥的产生、密钥的长度、证书的有效期规定及 CRL 的管理等。并为下属 CA 签发公钥证书。

3、认证中心 CA

CA 是认证中心，具备有限的政策制定功能，按照上级 PCA 制定政策，具体的用户公钥证书签发、生成和发布及 CRL 生成及发布。

4、在线证书申请 ORA

ORA 是在线执行 RA 服务器的功能，进行证书申请者的身份认证，向 CA 提交证书申请，验证接收 CA 签发的证书，并将证书发放给申请者。

5、证书使用实体 EE

EE 是证书使用实体，指具体的证书用户。EE 通过与认证中心

CA 的信息交互，向 CA 提出注册登记以及证书颁发等要求，获得数字证书。

1.4 目录协议

目录协议是建立 PKI 的基础。PKI 公钥证书系统的构成是基于 X.500 目录系列协议，在构造原理上 PKI 服务器就是一个 X.500 目录服务器或者一个 LDAP 服务器。

1.4.1 X.500

X.500 协议是 OSI 和 CCITT 的目录服务标准。X.500 是一个全局的分布数据库，一个 X.500 目录实体可以代表真实世界中任何一个实体。X.500 目录中的一个实体包含属主的特征信息，可以很好地区分不同的实体。

为了查找目录中的实体，X.500 采用一个特殊的方式给每个实体分配全局唯一的名字（DN）。X.500 目录按层次方式排列，因此叫做目录信息树即 DIT。树中的每个节点或向量都有一个父节点（根向量除外）和任意个子节点。每个向量除了根，均分配一个与身份相关的名字或 RDN，该节点的所有兄弟节点中 RDN 互不一样。每个节点的祖先的 RDN 与该向量的 RDN 连接起来形成实体的 DN。

X.500 名字 DN 的示意图如图 1-2 所示。

DN：{CN=CN, O=ERICST, CN=WANG}

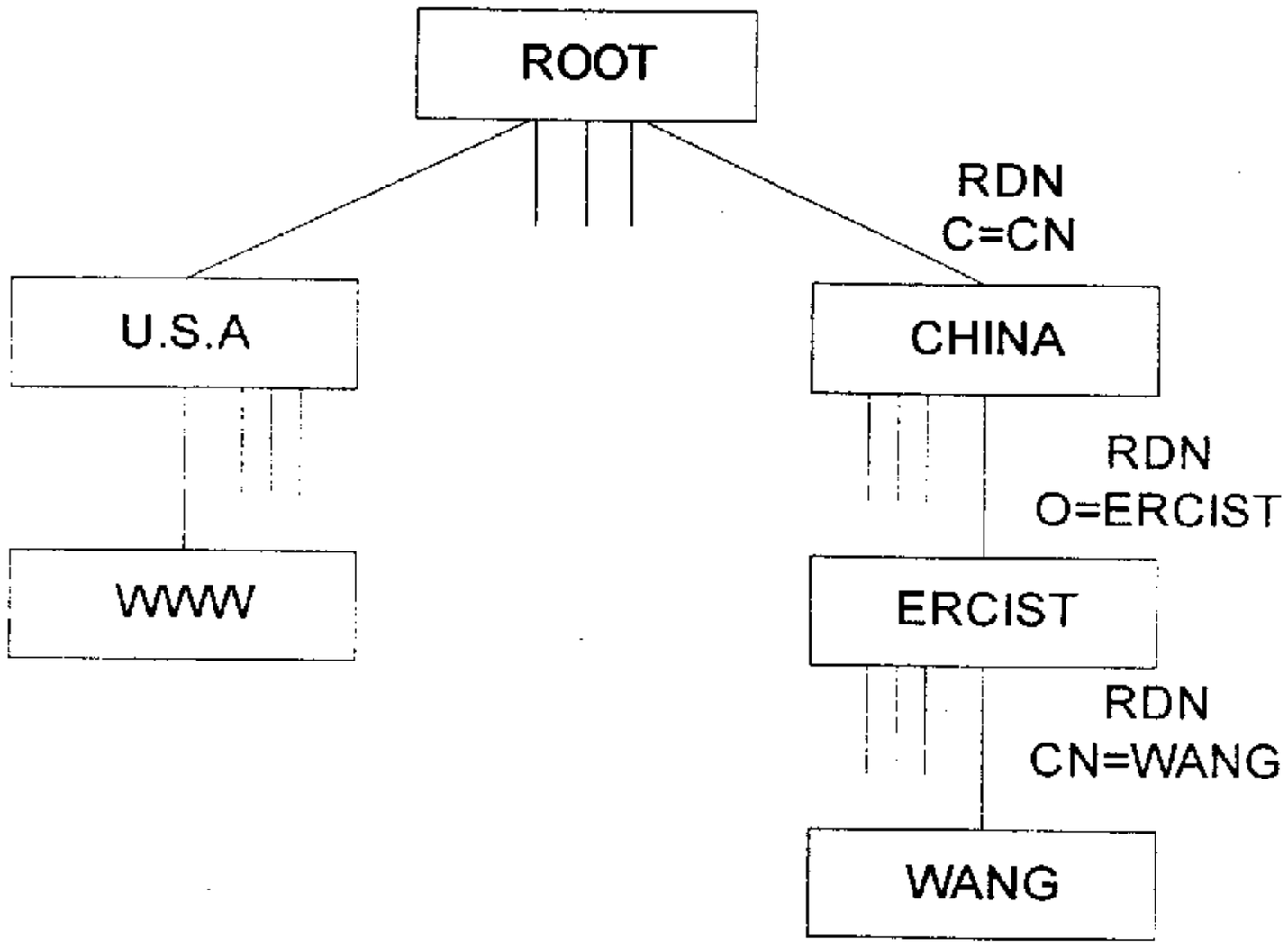


图 1-2 X.500 目录信息树

X.500 提供的服务是 OSI 目录服务，它主要由以下 5 部分组成：

1、组成目录的信息模型；
2、目录信息的网络名称和组织；
3、对目录信息进行操作，即：已知项的名称对象的属性检索、查询和读取，对属性和值的增加、修改和删除等；
4、目录信息的安全认证框架，支持口令模式和基于密码技术的认证；
5、分布式操作模型，客户—服务器方式，即：数据分布在网络中的多个目录服务器上，数据进操作在分布网络的环境下进行。

X.500 以客户—服务器方式运行，对于通过网络的目录操作信息交换，X.500 具有信息完整性验证功能。在系统组成上，用户通过目录用户代理 DUA 访问目录系统代理 DSA，DSA 之间通过目录通讯协议 DSP 通讯。当一个 DSA 不能找到所需信息时，它会自动链接另一台 DSA，直到得到结果，并将结果返回给 DUA，这样 X.500 系统可以提供分布式服务。

1.4.2 LDAP

证书系统发布证书或 CRL 到数据仓库，证书用户从数据仓库获取证书或 CRL 的过程称为证书服务的操作。证书管理里应提供多种操作途径，如：LDAP、HTTP、FTP、X.509 等。其中 LDAP 是最流行最方便的一种。

LDAP（Light Directory Access Protocol）即轻量级目录访问协议是基于 X.500 标准，但做了简化的存取控制访问协议。LDAP 直接运行在 TCP 之上，可以为目录用户提供快速的存取服务。LDAP 是为代替较为复杂的目录访问协议（DAP）而出现的，它最初的设计目的是为简单的管理或浏览应用提供对 X.500 目录简单的读写交互访问，即作为 X.500 的网关，但后来发展为独立的仅支持 LDAP 访问的目录服务器。

LDAP 的数据模型是基于 X.500 的，它的特点如下：

1、DIT 中每个条目（entry）以 DN 命名；
2、条目可归属一个或多个对象分类（object class）；
3、每个条目有一个或多个属性（attribute）；
4、每个属性有一个或多个值；
5、为最大限度采用可打印字符，采用了独特的编码方式。

LDAP 以 Clint—Server（客户—服务器）方式对 X.500 目录进行存取，基于 TCP 的默认端口 389。LDAP 服务器就是 PKIX 证书服务器，LDAP 客户即为证书用户。LDAP 客户可以对 PKI-X.509 服务器中的证书 CRL 进行下述操作：

1、LDAP 证书读操作

连接到 PKI 服务器根据证书用户名或发证 CA 的名称从相应的目录项中取回需要的信息。包括：BindRequest、BindResponse、SearchRequest、SearchResponse 和 UnbindRequest 五个 LDAP 操作。

2、PKI 服务器搜索

利用目录项证书的属性对 PKI 服务器中的证书或 CRL 进行搜索。包括 LDAP 的五个基本操作，在搜索中可以设置过滤条件。

3、PKI 证书或 CRL 修改

增加、删除和修改 PKI 服务器中的信息。主要包括：BindRequest、BindResponse、ModifyRequest、ModifyResponse、AddRequest 、 Addresponse 、 DelRequest 、 DelResponse 和 UnbindRequest 操作。

LDAP 允许管理员根据需要使用访问控制列表 ACI 控制对数

据读、写的权限。ACI 可以根据谁访问数据、数据存在什么地方以及其他对数据进行访问控制。因为这些都是由 LDAP 目录服务器完成的，所以不用担心在客户端的应用程序上是否要进行安全检查，从而很好地做到了操作安全、简便。

1.5 数字证书

公开密钥的用户将是对结合私有密钥被确定的远程主体（人或者系统），这些实体将使用加密或者签名算法。信任是通过证书中公开密钥的使用而得到，证书是绑定公开密钥到主题信息的数据结构,达到相当于实际生活中的身份证的功能。每张证书在它的签名内容中都有生命期。因为每张证书的签名和生命期在证书使用的客户端是独立检查的，证书能够（可以）在不受信任的通讯和服务器系统中传输也能在证书使用系统中非安全存储。

ITUT X.509（过去 CCITT X.509）首先在 1988 年作为 X.500 目录服务系统推荐的一部分出版。在 1988 标准中证书格式称为版本 1（v1）格式。当 X.500 在 1993 年修正的时候，在版本 2（v2）格式中增加一额外的两个字段。这两个字段可以使用来支持目录服务系统的存取控制。

后来面对各种新的要求，又相继开发了 X.509 版本 3（v3）证书格式。v3 格式在 v2 基础上通过扩展添加额外的字段（扩展字段）。特殊的扩展字段类型可以在标准中或者可以由任何组织定义和注册。在 1996 年 6 月，基本 v3 格式的标准化被完成。

1.5.1 证书结构和语义

尽管 X.509 已经定义了证书的标准字段和扩展字段的具体要求，仍然有很多的证书在颁发时需要一个专门的子集来进一步定义说明。Internet 工程任务组（IETF）PKI X.509 工作组就制定了这样一个协议子集，即 RFC2459（PKIX 的第一部分）。虽然 RFC2459 是专门为 Internet 的应用环境而制定的，但它里面的很多建议都可以应用于企业环境，并且它的前后一致性应该尽可能地得到贯彻执行。因此，我们会对 RFC2459 中那些有用的适当的建议给出参考。

下图所示为第 3 版的证书结构。

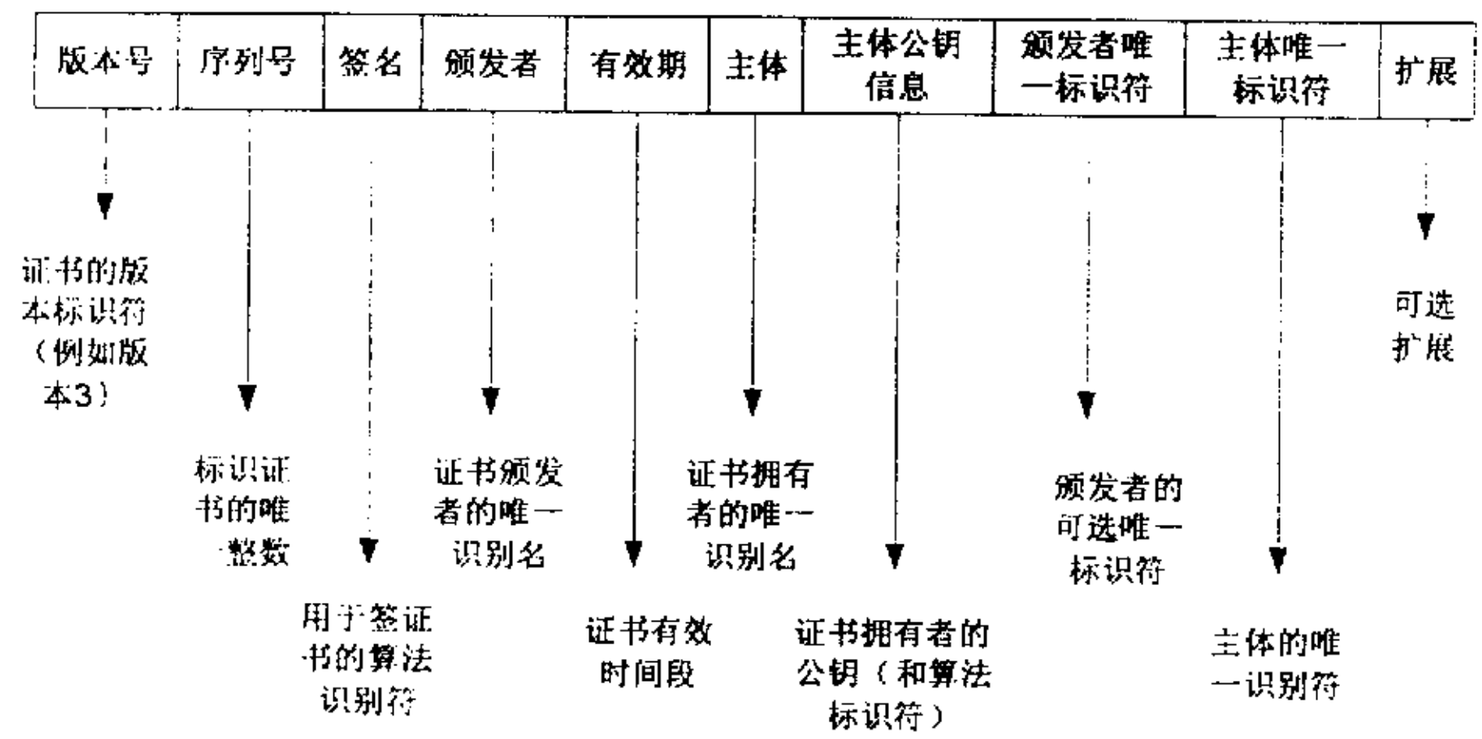


图 1-3 第三版证书结构

上图中的字段说明：

1、版本号

标示证书的版本（版本 1、版本 2 或是版本 3）。

2、序列号

由证书颁发者分配的本证书的唯一标识符。

3、签名

签名算法标识符，用于说明本证书所用的数字签名算法。

4、颁发者

证书颁发者的可识别名（DN），这是必须说明的。

5、有效期

证书有效时间段。本字段由“Not Valid Before”和“Not Valid After”两项组成，他们分别由 UTC 时间或一般的时间表示。

6、主体

证书拥有者的可识别名——此字段必须是非空的，除非使用了其他的名字形式。

7、主体公钥信息

主体的公钥（以及算法标识符）——这是必须说明的。

8、颁发者唯一标识符

证书颁发者的唯一标识符，仅在版本 2 和版本 3 中要求，属于可选项；该字段在实际应用中很少使用，并且不被 RFC2459 推荐使用。

9、主体唯一标识符

证书拥有者的唯一标识符，仅在版本 2 和版本 3 中要求，属于可选项；该字段在实际应用中很少使用，并且不被 RFC2459 推

荐使用。

10、扩展

可选的标准和专用扩展（仅在版本 3 里使用）。

1.5.2 证书验证

检查一份给定的证书是否可用的过程就称为证书验证。证书验证包括确定如下内容：

1、一个可信的 CA 已经在证书上签名（即 CA 的数字签名被验证是正确的）。注意这可能包括证书路径处理。

2、证书由良好的完整性。即证书上的数字签名与签名者的公钥和单独计算出来的证书杂凑值相一致。

3、证书处在有效期内。

4、证书没有被撤消。

5、证书的使用方式与任何声明的策略和/或使用限制相一致。

1.5.3 可选的证书格式

除了 X.509 第三版本的公钥证书以外，还有一些其他的证书类型。如：SPKI、PGP、SET 等。

1.6 ASN.1 编码介绍

1.6.1 ASN.1 简介

ASN.1 是 Abstract Syntax Notation One 的缩写，即“抽象语法标记”。简单地讲，ASN.1 是一种用来描述结构化信息的标记语言，而它所描述的结构化信息往往需要在通信介质中传递。ASN.1 现在已经成为国际标准，它主要用在通信协议的具体表示中。随着计算机技术的飞速发展，在计算机系统之间、各种应用程序之间都需要进行大量、复杂的信息交换与传递。对于这些需要传递交换的信息，人们需要用一种详细准确的规范来表示它们，以便不同的计算机系统或不同的应用程序之间能够传递这些信息。为此，人们需要制定各种应用协议。ASN.1 的作用就是具体描述各种协议标准，使人们既能清楚简洁地表示协议中要传递的各种信息，又能容易地实现相应的协议标准。目前，在电信、安全信息

交换以及多媒体传输等领域，ASN.1 都是描述协议的主要手段。此外，在大规模系统开发中，人们可能需要制定专门的通信协议，此时，如果用 ASN.1 来描述定义这些协议，然后再利用 ASN.1 编译器进行转化，就可以得到某些常见编程语言的相应数据结构表示形式，再通过相应的编/解码器，就可以生成可在通信媒介中传递的位模式。这种方式极大地方便了大规模系统的开发，充分保证了自定义协议相互之间具有互操作性和可扩展性，既有利用于系统进行良好的前期规划设计，又减轻了程序员的编程工作。PKI 系统中所有的数据都由 ASN.1 定义。

1.6.2 抽象语法的提取

ASN.1 提供了 20 多种内嵌缺省数据类型，任意复杂的结构都可利用构造标识符出简单的数据类型生成。需要描述的信息越复杂，就越能体现 ASN.1 的优越性。因为 ASN.1 支持用统一方式以很简单的成员构建出任意复杂的结构，它支持嵌套结构，并且这种嵌套结构可以扩展到任意深度以满足特殊需要。最终，所有复杂的类型都归为由简单类型定义而成。

1.6.3 类型和取值

ASN.1 最基本的概念是类型（Type）和取值（Value）。类型是该类型所有可能取值的集合，发送者通过选取某一类型的某个具体取值来传递相应的信息。一个类型可能只有几种取值，例如：BOOLEAN 类型只有两种取值而 INTEGER 类型的取值可以有无穷多。

根据实际需要，使用者可以自己定义新的类型和相应取值。这些自定义的类型和取值一样可以被别人使用，从而提供了极大的灵活性和可扩展性。

在 ASN.1 中，类型的定义要比取值的定义重要的多，这是因为对某个抽象语法标记来说，它本身就是某种类型。它的成分，以及成分的成分还是各种类型，如此嵌套下去。在一个协议标准中，类型是最重要的，它反映各种可能的取值；相反，在具体的信息通信中，取值更为重要。

1.6.4 编码方式

编码也称为传输语法（Transfer Syntax）位于网络的表示层，ASN.1 把抽象语法和传输语法这两个概念分开处理，也就是在具体传输时可以选择不同的编码方法，既可以选择传输效率最好的编码，也可以选择可靠性最好的编码还可以选择最易解码的编码。

1.6.4.1 BER 编码规则(Basic Encolding Rule)

所有用 ASN.1 定义的变量，其数值可以用基本编码规则编码，基本编码规则并不是 ASN.1 叙述变量唯一编码规则，还有其他编码方式，只是它是目前国际标准组织公认的标准编码方式。

利用 BER 编码的结果，可以在任何支持字节串（strings of octets）的通信介质中传输。正如 ASN.1 是层次结构化的一样，BER 也遵循这种层次嵌套结构。编码时按照类型（Type）、长度（Length）、值（Value）的顺序进行。在 ASN.1 中，这种编码表示法也叫做 ILC，即表示符（Identifier octets）、长度（Length octets）和内容（Contents octets）。在一个 Contents 中可以继续嵌套一系列 ILC。

1.6.4.2 DER 编码规则(distinguished encoding rules)

DER 编码规则(distinguished encoding rules)是 BER 编码规则的子集合，它可以表示任何 ASN.1 的值，编码数据使用明确长度的编码。

DER 对 BER 编码方法添加了如下一些限制：

1、长度在 0 与 127 之间，必须用“short”的形式。

2、长度大于等于 128，必须用“long”的形式。

3、对于简单字符串类型以及由其隐式置标所得类型必须用简单定长的编码方法。

对于许多具体的类型（例如二进制位串，序列，集合等）也添加了许多具体的限制。

第二章 密码理论

2.1 概述

2.1.1 对称密码算法

对称算法有时又叫传统密码算法，就是加密密钥能够从解密密钥中推算出来，反过来也成立。在大多数对称算法中，加/解密密钥是相同的。这些算法也叫秘密密钥算法或单密钥算法，它要求发送者和接收者在安全通信之前，商定一个密钥。对称算法的安全性依赖于密钥，泄漏密钥就意味着任何人都能对消息进行加/解密。只要通信需要保密，密钥就必须保密。

对称算法可分为两类。一次只对明文中的单个比特（有时对字节）运算的算法称为序列算法或序列密码。另一类算法是对明文的一组比特进行运算，这些比特组称为分组，相应的算法称为分组算法或分组密码。现代计算机密码算法的典型分组长度为 64 比特，这个长度大到足以防止分析破译，但又小到足以方便使用（在计算机出现前，算法普遍地每次只对明文的一个字符运算，可认为是序列密码对字符序列的运算）。

2.1.2 公开密码算法

公开密钥算法(也叫非对称算法)是这样设计的：用作加密的密钥不同于用作解密的密钥，而且解密密钥不能根据加密密钥计算出来(至少在合理假定的长时间内)。之所以叫做公开密钥算法，是因为加密密钥能够公开，即陌生者能用加密密钥加密信息，但只有用相应的解密密钥才能解密信息。在这些系统中，加密密钥叫做公开密钥（简称公钥），解密密钥叫做私人密钥（简称私钥）。私人密钥有时也叫秘密密钥。

用公开密钥 K 加密表示为：（注：M 是加密前的数据，C 是加密后的数据，EK-pub 是加密公钥，EK-pri 是解密私钥）

EK-pub（M）=C.

虽然公开密钥和私人密钥是不同的，但用相应的私人密钥解密可表示为：

DK-pri（C）=M

有时消息用私人密钥加密而用公开密钥解密，这用于数字签名，尽管可能产生混淆，但这些运算可分别表示为：

EK-pri（M）=C

DK-pub（C）=M

公开密钥既可以进行加密也可以进行数字签名和身份认证。常用的公钥系统加密算法有 RSA 算法，加密强度很高。采用公钥进行数字签名一般是发送方用自己的私钥将数据加密后传输给接收方。另外，公钥系统的计算复杂度较高，一般说来比基于私钥系统的计算量高两个数量级，大量数据的加密采用公钥系统开销太大，一般只用来加密少量关键数据，如秘密密钥系统的加密密钥等。因此公开密钥加密大多仅被用来 internet 用户之间建立相互信任的关系。由于公开密钥加密在对大量数据进行加密时速度太慢，因此在通信双方之间进行安全数据传输还是采用秘密密钥加密。在实用系统中，经常采用的是将公钥系统与秘密密钥系统结合的做法。

2.1.3 单向散列(hash)函数

单向散列函数是一种单向密码体制，它是一个从明文到密文的不可逆函数，也就是说，是无法解密的。单向散列函数通常用于只需要加密、不需要解密的特殊应用场合。例如，为保证数据的完整性（即保证信息不被非法修改）可以使用单向散列函数对数据生成并保存散列值，然后，每当使用数据时，用户将重新使用单向散列函数计算数据的散列值，并与保存的数值进行比较，如果相等，说明数据是完整的，没有经过改动；否则，则说明数据已经被改动了。

单向散列函数还可以应用于诸如口令表加密等场合，这时系统中保存的是口令的散列值，当用户进入系统时输入口令，系统重新计算用户输入口令的散列值并与系统中保存的数值相比较，当两者相等时，说明用户的口令是正确的，允许用户进入系统，反之则拒绝。使用单向函数保存口令表避免了以明文的形式保存口令，至今仍然在许多系统中得到广泛的使用。

另外，单向函数在数字签名等安全实践中具有很重要的作用。

目前使用比较广泛的几种散列函数有 MD5、SHA 等。

2.2 PKI 中常用的密码技术

2.2.1 加密/解密技术

加密是指使用密码算法对数据做变换，由明文变成密文，使得只有密钥的持有人，才能恢复数据的原貌，即解密，将密文还原为明文。主要目的是防止消息的非授权泄露。

现代密码学的基本原则是：一切秘密寓于密钥之中。算法是公开的，密钥是保密的。PKI 中常用的加密技术有：

2.2.1.1 对称密钥加密技术

对称密钥密码加密技术，也称单密钥密码技术。即加密密钥和解密密钥是相同的。设加密密钥为 K_e，解密密钥为 K_d，$K_e = K_d$，因此密钥必须特殊保管。对称密钥加密技术的特点是：保密强度高，计算开销时间少，处理速度快，适合大文件加/解密。在 PKI 中它与下述的非对称密钥加密技术相结合，实现了很多安全服务手段。对称密钥密码技术的缺点是密钥分发管理困难。其过程如下图所示：

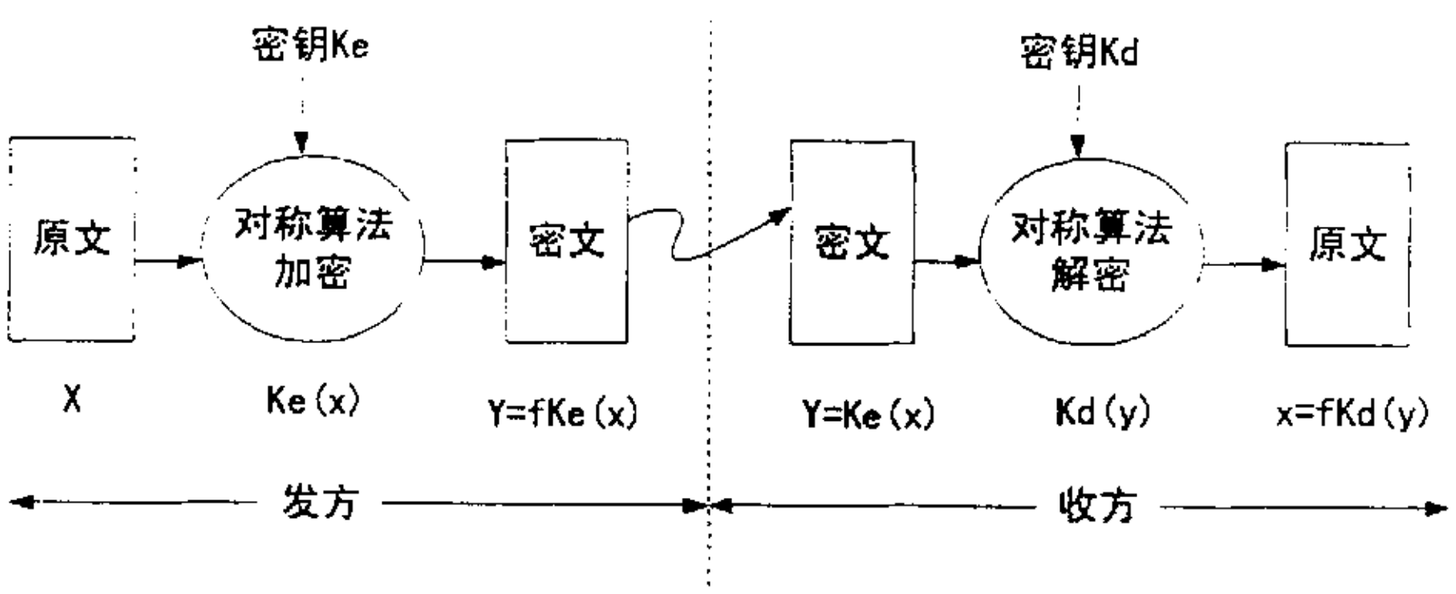


图 2-1 对称密钥加密过程

2.2.1.2 非对称密钥加密技术

非对称密钥加密技术也称双密钥密码技术。即每个用户都有两个密钥，一个是在信息集团内公开的，称为公开密钥；另一个

由用户秘密保存，称为私有密钥，简称私钥。因此，加密密钥与解密密钥是不相同的，由加密密钥解出解密密钥是不可能的。非对称密钥密码技术的特点是便于密钥管理和分发，便于通信加密和数字签名。它的缺点是处理速度较慢，特别是被加密的文件大时，计算开销要大。

非对称密钥加密技术分为两种情况：一种是是采用收方公钥加密数掘，而用收方的私钥解密；另一种是采用发方的私钥加密，而用发方的公钥解密。二者原理相同，但用途不同。

（1）收方公钥加密，收方私钥解密

设公钥为 K_{PB}，私钥为 K_{PV}，其过程如下图所示。

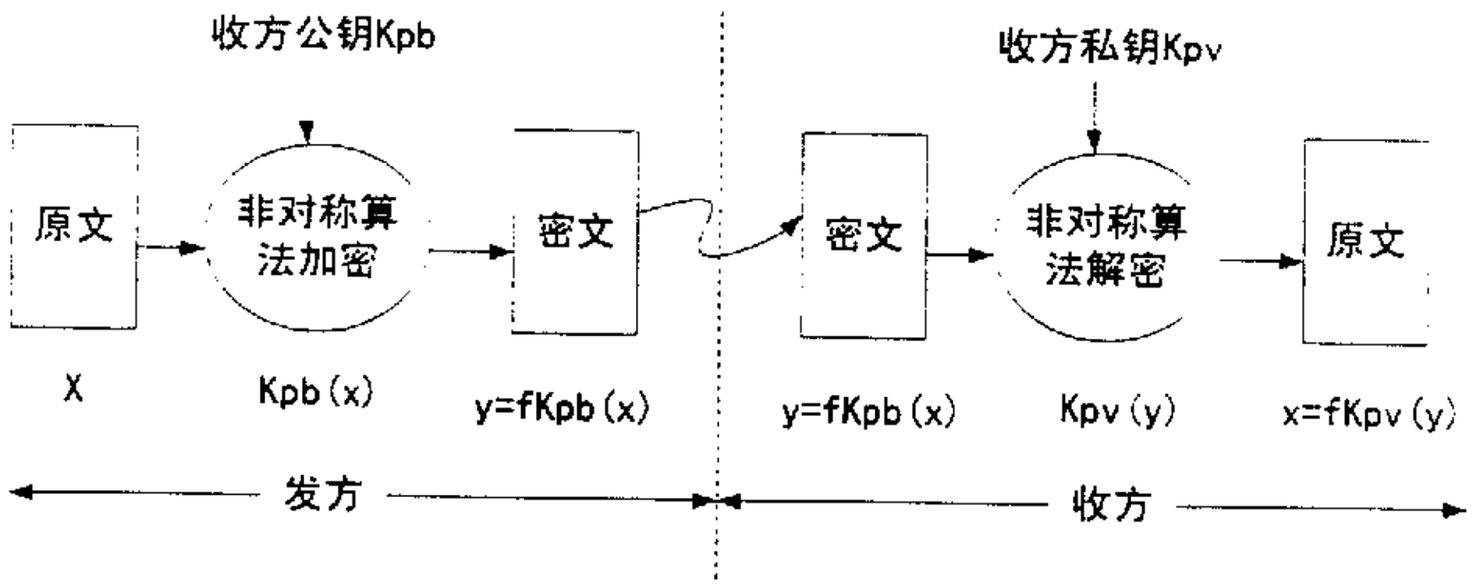


图 2-2 非对称算法公钥加密

这种以收方公钥 K_{PB} 加密原文，以收方私钥 K_{PV} 来解密的非对称密码算法，可以实现多个用户加密信息，只能由一个用户解读，这就实现了保密通信。PKI 中的加密机制，保证数据完整性服务。就是依据这种技术实现的。

（2）发方私钥加密，发方公钥解密

设公钥为 K_{PB}，私钥为 K_{PV}，其过程如下图所示。

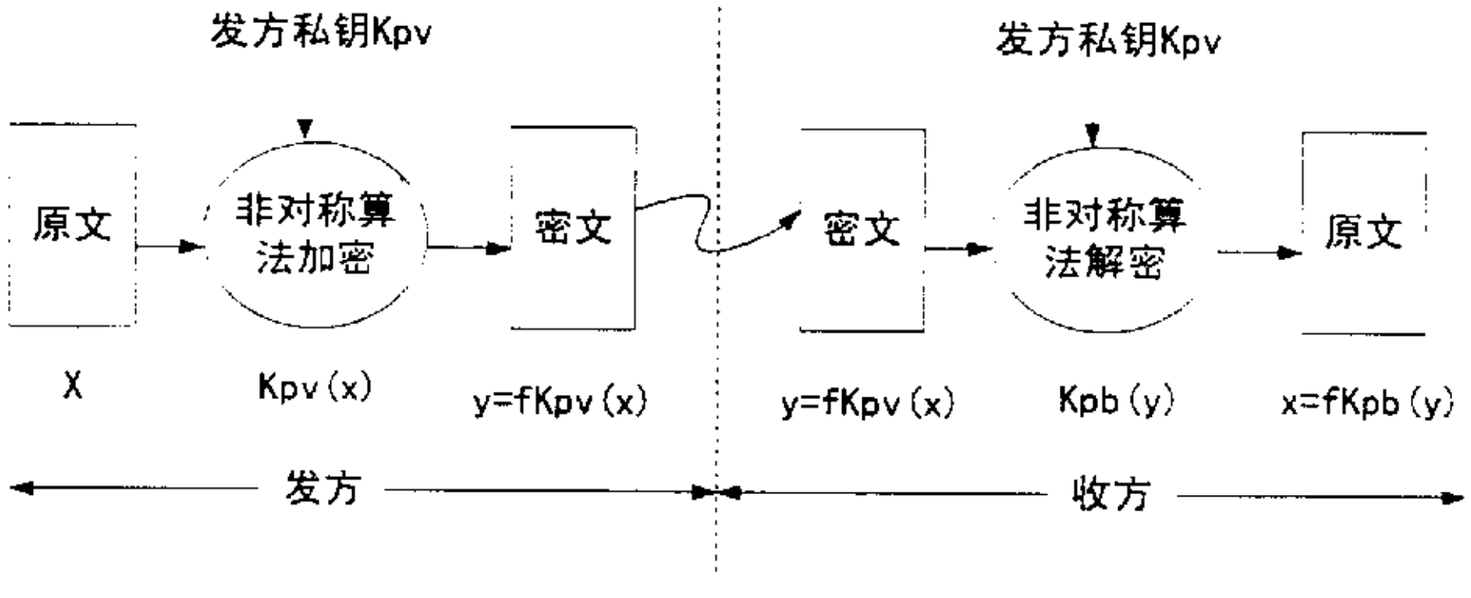


图 2-3 非对称算法私钥加密

这种以发方私钥 K_{pv} 加密原文，以发方公钥 K_{pB} 来解密的非对称，算法，可以实现由一个用户加密的信息，而由多个用户解读，这就是数字签名的原理。PKI 中的签名机制，保证不可否认性服务及数据完整性服务，就是依靠这种技术实现的。

2.2.2 数字签名

数字签名是指使用密码算法，对待发的数据（报文或票证等）进行加密处理，生成一段数据摘要信息附在原文上一起发送，这段信息类似现实中的签名或印章，接受方对其进行验证，判断原文真伪。这种适用于对大文件的处理，对于那些小文件的数据签名，则不预先做数据摘要，而直接将原文进行加密处理。

数字签名在 PKI 中提供数据完整性保护和提供不可否认性服务。

2.2.2.1 具有数据摘要的数字签名

先采用单向函数 Hash 算法，对原文信息进行加密压缩形成数据摘要，然后，对数据摘要用公开密钥算法进行加密和解密。原文的任何变化都会使数据摘要发生改变，所以，它是一种对压缩消息的签名。其过程如下图所示。

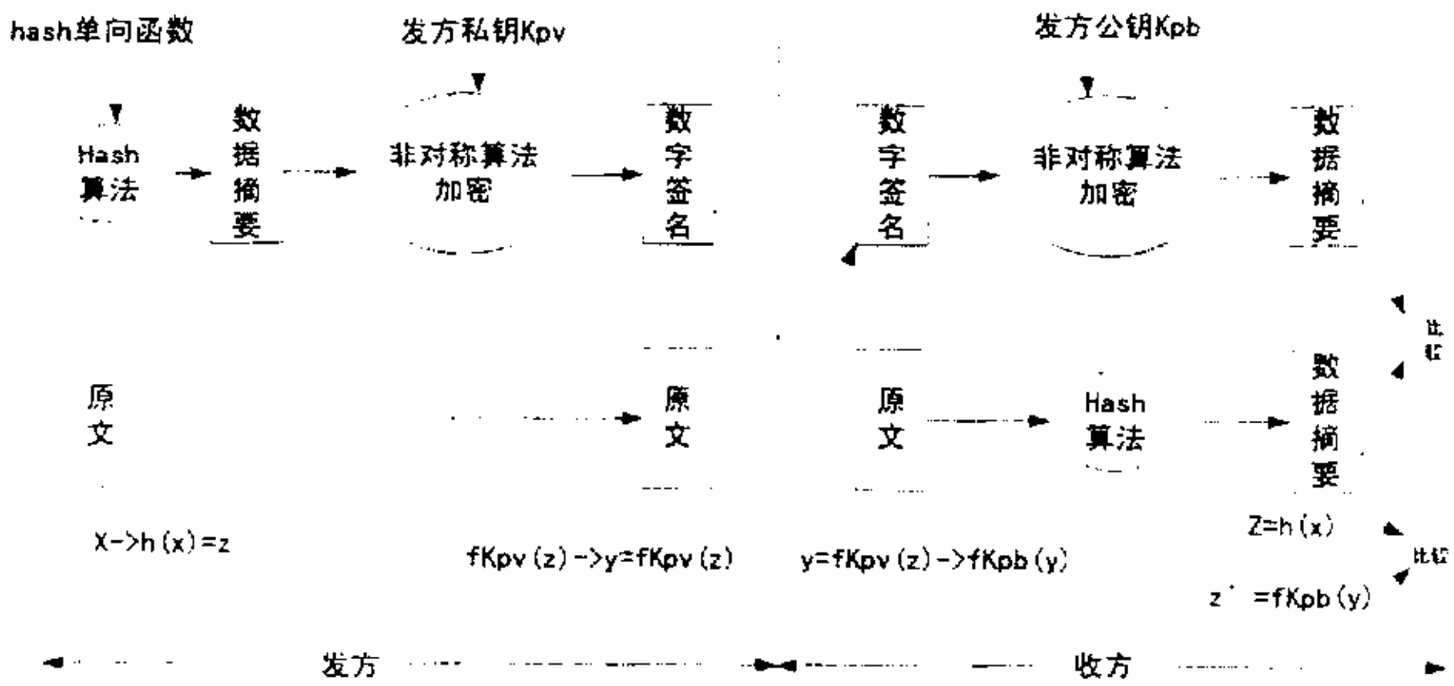


图 2-4 具有数据摘要的数字签名

这种数据签名适合对大文件的信息的数字签名。对一个数字签名要进行验证，就是最后对数据摘要的比较。

2.2.2.2 直接用私钥进行加密的数字签名

这种签名方法，是采用非对称算法中私有密钥对原文进行加密，而不用 hash 单向散列函数做数据摘要，是一种对整体消息的签名，适用于小文件信息。

其做法是：首先将原文用发方私钥加密，得到数字签名，然后将原文和数字签名一起发向接收方，接收方用发方的公钥进行解密，最后与原文进行比较，如图 2-6 所示。

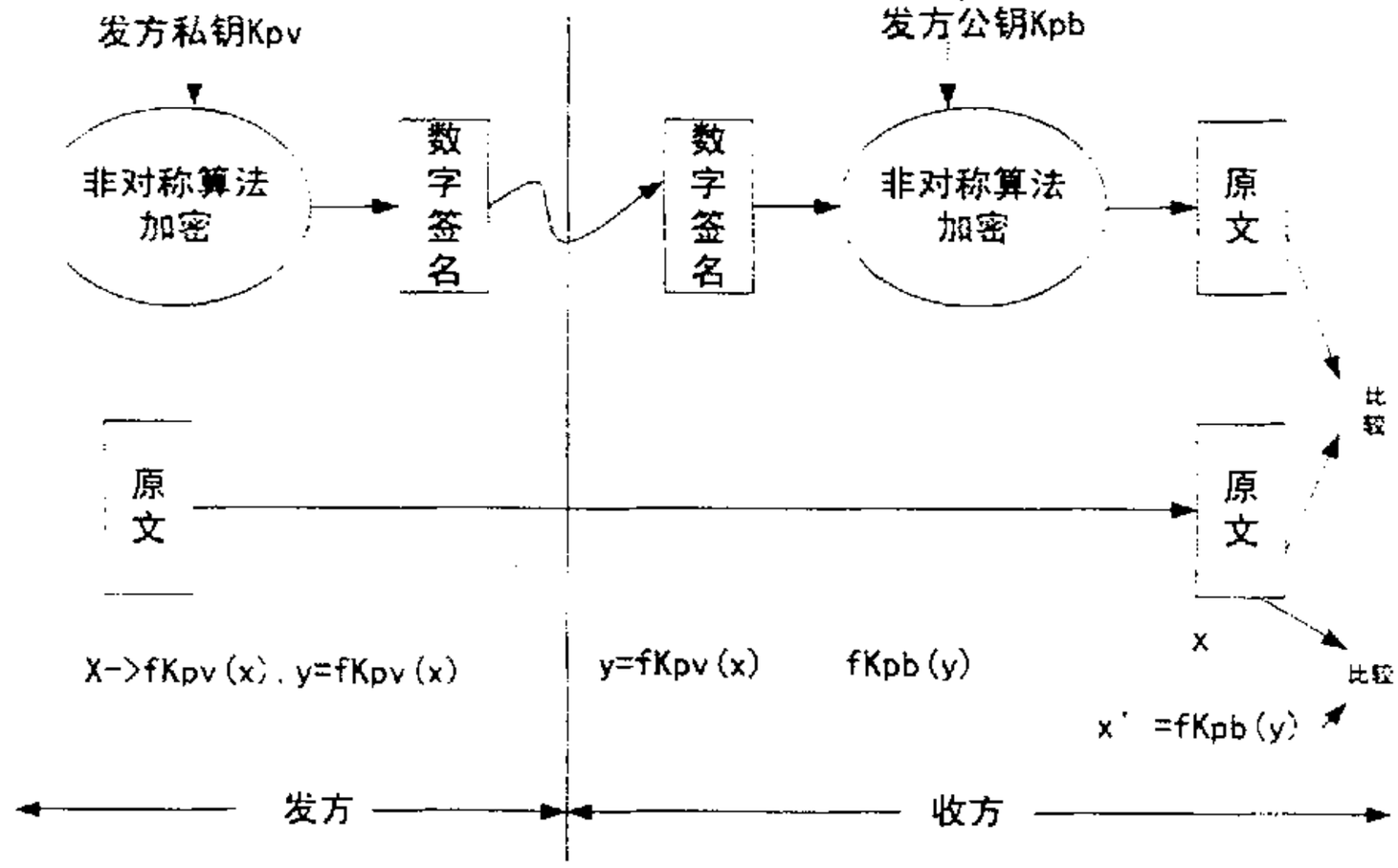


图 2-5 公钥加密原文的数字签名

通过验证，即 x' 与 x 的比较，$x = x'$？可以确定三件事：

（1） 消息 x 确实是由 A 方发出的。

（2） 签名 y 确实是由 A 方产生的。

（3） 收方 B 收到的信息是完整的。

所以，数字签名在信息安全中，包括身份认证，数据完整性、数据保密性及不可否认性等方面起重要作用。

2.2.3 消息摘要

消息摘要是一个惟一对应一个消息或文本的值，由一个单向散列加密函数对消息作用产生。在很多应用中，首先通过单向散列函数算法生成消息的文摘。用发送者的私有密钥加密摘要附在原文后面，一般称为消息的数字签名。数字签名的接受者可以确信消息确实来自谁，另外，如果消息在途中改变了，则接收者通过对收到消息的新产生的摘要与原摘要比较，就可知道消息是否被改变了。因此消息摘要保证了消息的完整性。消息摘要的示意图如下所示：

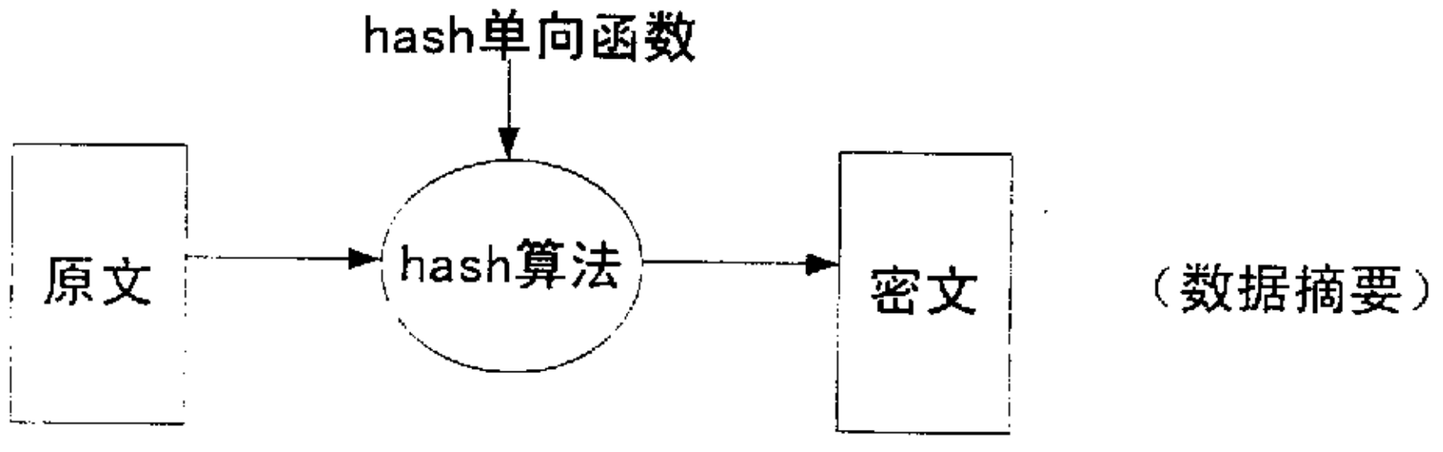


图 2-6 消息摘要示意图

2.2.4 数字信封

所谓数字信封就是信息发送端用接收端的公钥，将一个通信密钥（Symmentric Key），即对称密钥，给予加密，形成一个数字信封（DE）。然后传送给接收端，只有指定的接收方才能用自己的私钥打开数字信封，获取该对称密钥（SK），用它来解读传送来的信息。这就好比在实际生活中，将一把钥匙装在信封里，邮寄给对方，对方收到信件后，将钥匙取出，用它再去打开保密箱一样。其具体过程如图 2-7 所示。

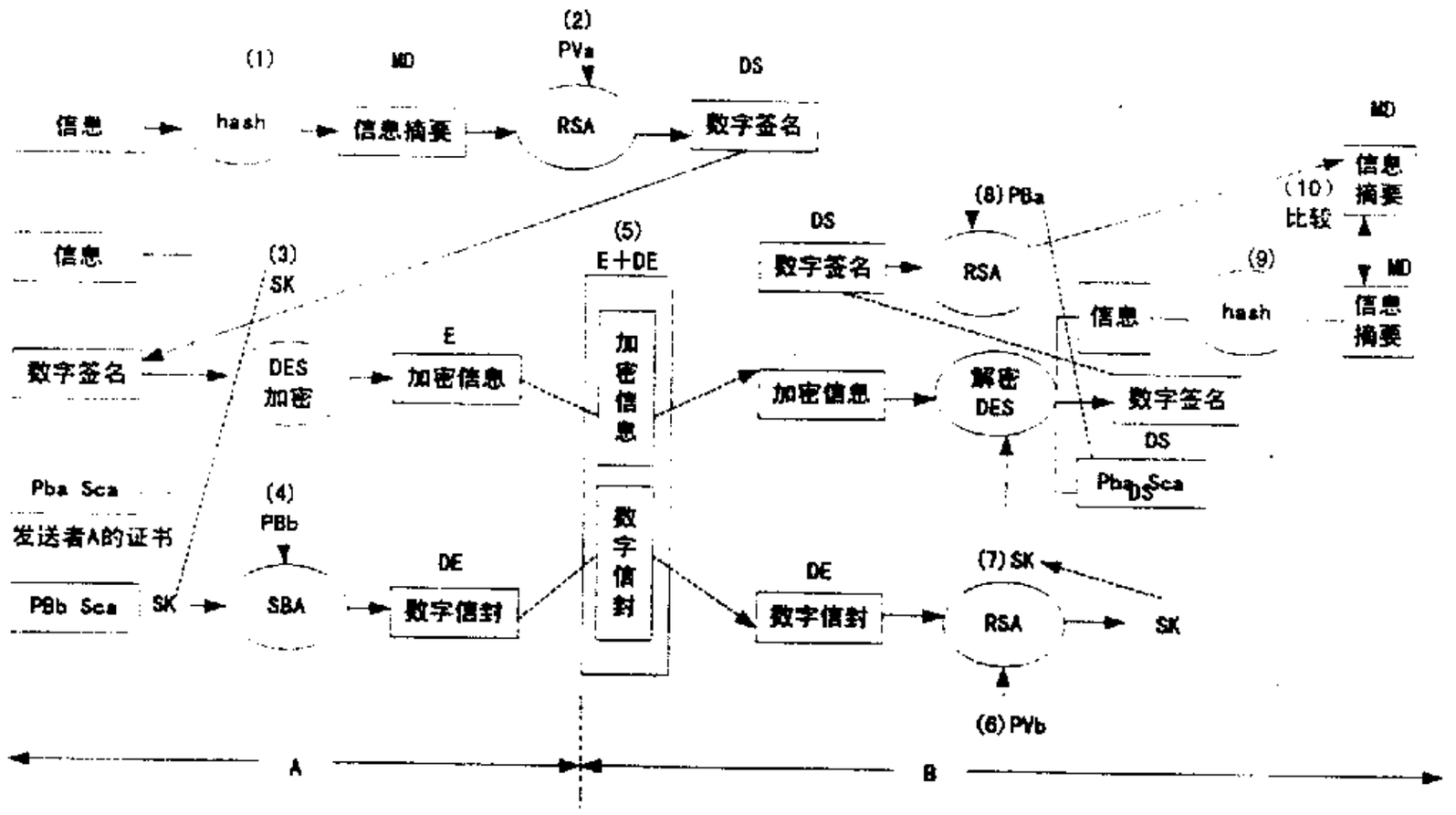


图 2-7 数字信封

其中：

（1） 将要传输的信息经杂凑函数（hash）运算后，得到一个数据摘要 MD，MD＝hash（信息）。

（2） 发送者 A 用自己的私钥 PV_A 对数据摘要进行加密，得 A 的数字签名。

（3） 发送者 A 将信息明文、数字签名和他的证书上的公钥三项信息，通过对称算法，用对称密钥 SK 进行加密，得密文 E。

（4） 发送者 A 在发送信息之前，必须事先得到接收方 B 的证书公钥 PB_B，用 PB_B 加密 SK，形成一个数字信封 DE。

（5） E+DE 就是将密文与数字信封连接起来，即 A 所发送的内容。

（6） 接收者 B 用自己的私钥 PV_B 解开所收到的数字信封 DE，并从中取出 A 所用过的对称密钥 SK。

（7） 接收者 B 用 SK 将密文 E 解密还原成信息明文、数字签名和 A 的证书公钥。

（8） B 将数字签名用 A 证书中的公钥 PB_A 进行解密，将数字签名还原成信息摘要 MD。

（9） B 再对已收到的信息明文，用同样的 hash 函数算法进行杂凑运算，得到一个新的信息摘要 MD'。

（10） 对数字签名进行校验，比较收到已还原的 MD 和新产生的 MD' 是否相等，二者必须相等无误，否则不接收。

数字信封是 PKI 中使用对称密钥密码算法与非对称密钥密码算法的巧妙结合，是上述加密、数字签名和数据摘要技术的综合应用。数字信封是 PKI 应用系统中常用的一种密码技术。

第三章 加解密算法的实现

3.1 数据加密标准 DES

3.1.1 DES 算法描述

DES 是一个分组加密算法，他以 64 位为分组对数据加密。同时 DES 也是一个对称算法：加密和解密用的是同一个算法。它的密钥长度是 64 位，但有效长度是 56 位（因为每个第 8 位都用作奇偶校验），保密性依赖于密匙。

DES 对 64 位的明文分组进行操作，通过一个初始置换，将明文份组成左半部分和右半部分，各 32 位长。然后进行 16 轮完全相同的运算，这些运算被称为函数 f，在运算过程中数据于密匙结合。经过 16 轮后，左、右半部分合在一起经过一个末置换，这样就完成了。在每一轮中，密钥位移位，然后再从密钥的 56 位中选出 48 位。通过一个扩展置换将数据的右半部分扩展成 48 位，并通过一个异或操作替代成新的 32 位数据，再将其置换一次。这四步运算构成了函数 f。然后，通过另一个异或运算，函数 f 的输出与左半部分结合，其结果成为新的右半部分，原来的右半部分成为新的左半部分。将该操作重复 16 次，就实现了 DES 的 16 轮运算。

3.1.2 DES 加密算法的实现

下图所示为 DES 的加密运算框图。

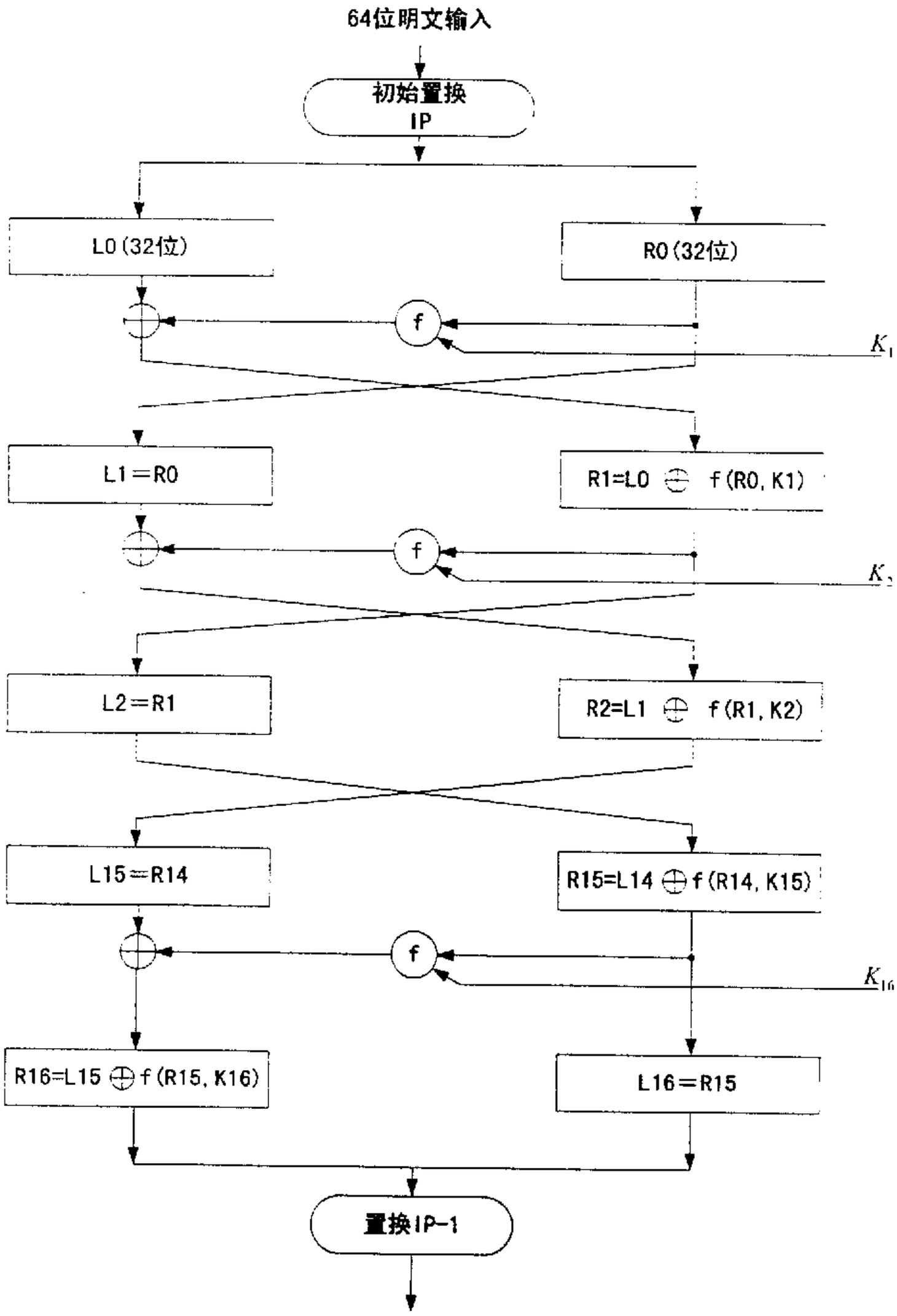


图 4-1 DES 加密运算框图

3.1.2.1 获取密钥

取得 64 位的密钥，每个第 8 位作为奇偶校验位。

3.1.2.2 变换密钥

1、舍弃 64 位密钥中的奇偶校验位，根据下表（PC-1）进行密钥变换得到 56 位的密钥，在变换中，奇偶校验位已被舍弃。

置换选择 1 (PC-1)

57 49 41 33 25 17 9
1 58 50 42 34 26 18
10 2 59 51 43 35 27
19 11 3 60 52 44 36
63 55 47 39 31 23 15
7 62 54 46 38 30 22
14 6 61 53 45 37 29
21 13 5 28 20 12 4

2、将变换后的密钥分为两个部分，开始的 28 位称为 C[0]，最后的 28 位称为 D[0]。

3、生成 16 个子密钥，初始 I=1。

（1）同时将 C[I]、D[I]左移 1 位或 2 位，根据 I 值决定左移的位数。见下表

I： 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
左移位数： 1 1 2 2 2 2 2 2 1 2 2 2 2 2 2 1

（2）将 C[I]D[I]作为一个整体按下表（PC-2）变换，得到 48 位的 K[I]

置换选择 2 (PC-2)

14 17 11 24 1 5
3 28 15 6 21 10
23 19 12 4 26 8
16 7 27 20 13 2
41 52 31 37 47 55
30 40 51 45 33 48
44 49 39 56 34 53
46 42 50 36 29 32

（4） 从（1）处循环执行，直到 K[16]被计算完成。

（5） 整个子密钥的生成过程如图 4-2 所示。

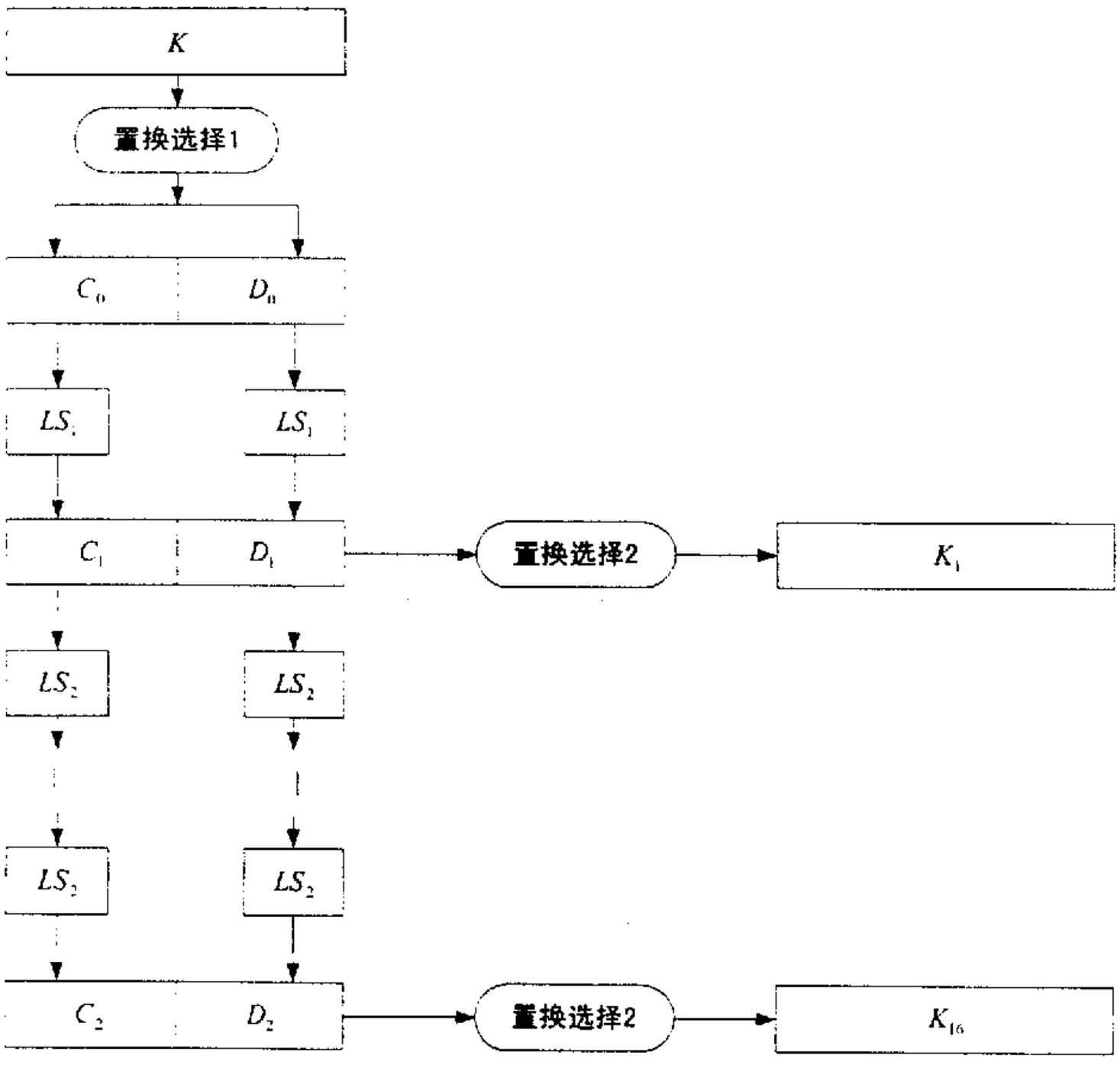


图 4-2 DES 子密钥产生算法

3.1.2.3 明文数据处理

1、取得 64 位数据，如果数据长度不足 64 位，应该将其扩展为 64 位（例如补零）

2、将 64 位数据按下表变换（IP）

初始置换（IP）

58 50 42 34 26 18 10 2
60 52 44 36 28 20 12 4
62 54 46 38 30 22 14 6
64 56 48 40 32 24 16 8
57 49 41 33 25 17 9 1
59 51 43 35 27 19 11 3
61 53 45 37 29 21 13 5
63 55 47 39 31 23 15 7

3、将变换后的数据分为两部分，开始的 32 位称为 L[0]，最后的 32 位称为 R[0]。

3.1.2.4 用 16 个子密钥加密数据

初始 I＝1

1、将 32 位的 R[I-1]按下表（E）扩展为 48 位的 E[I-1]

扩展置换（E）

32 1 2 3 4 5
4 5 6 7 8 9
8 9 10 11 12 13
12 13 14 15 16 17
16 17 18 19 20 21
20 21 22 23 24 25
24 25 26 27 28 29
28 29 30 31 32 1

2、异或 E[I-1]和 K[I]，即 E[I-1] XOR K[I]

3、将异或后的结果分为 8 个 6 位长的部分，第 1 位到第 6 位称为 B[1]，第 7 位到第 12 位称为 B[2]，依此类推，第 43 位到第 48 位称为 B[8]。

4、按 S 表变换所有的 B[J]，初始 J=1。所有在 S 表的值都被当作 4 位长度处理。

（1）、将 B[J]的第 1 位和第 6 位组合为一个 2 位长度的变量 M，M 作为在 S[J]中的行号。

（2）、将 B[J]的第 2 位到第 5 位组合，作为一个 4 位长度的变量 N，N 作为在 S[J]中的列号。

（3）、用 S[J][M][N]来取代 B[J]。

S[1]

14 4 13 1 2 15 11 8 3 10 6 12 5 9 0 7
0 15 7 4 14 2 13 1 10 6 12 11 9 5 3 8
4 1 14 8 13 6 2 11 15 12 9 7 3 10 5 0
15 12 8 2 4 9 1 7 5 11 3 14 10 0 6 13

S[2]

15 1 8 14 6 11 3 4 9 7 2 13 12 0 5 10
3 13 4 7 15 2 8 14 12 0 1 10 6 9 11 5
0 14 7 11 10 4 13 1 5 8 12 6 9 3 2 15
13 8 10 1 3 15 4 2 11 6 7 12 0 5 14 9

S[3]

10 0 9 14 6 3 15 5 1 13 12 7 11 4 2 8
13 7 0 9 3 4 6 10 2 8 5 14 12 11 15 1
13 6 4 9 8 15 3 0 11 1 2 12 5 10 14 7
1 10 13 0 6 9 8 7 4 15 14 3 11 5 2 12

S[4]

7 13 14 3 0 6 9 10 1 2 8 5 11 12 4 15
13 8 11 5 6 15 0 3 4 7 2 12 1 10 14 9
10 6 9 0 12 11 7 13 15 1 3 14 5 2 8 4
3 15 0 6 10 1 13 8 9 4 5 11 12 7 2 14

S[5]

2 12 4 1 7 10 11 6 8 5 3 15 13 0 14 9
14 11 2 12 4 7 13 1 5 0 15 10 3 9 8 6
4 2 1 11 10 13 7 8 15 9 12 5 6 3 0 14
11 8 12 7 1 14 2 13 6 15 0 9 10 4 5 3

S[6]

12 1 10 15 9 2 6 8 0 13 3 4 14 7 5 11
10 15 4 2 7 12 9 5 6 1 13 14 0 11 3 8
9 14 15 5 2 8 12 3 7 0 4 10 1 13 11 6
4 3 2 12 9 5 15 10 11 14 1 7 6 0 8 13

S[7]

4 11 2 14 15 0 8 13 3 12 9 7 5 10 6 1
13 0 11 7 4 9 1 10 14 3 5 12 2 15 8 6
1 4 11 13 12 3 7 14 10 15 6 8 0 5 9 2
6 11 13 8 1 4 10 7 9 5 0 15 14 2 3 12

S[8]

13 2 8 4 6 15 11 1 10 9 3 14 5 0 12 7
1 15 13 8 10 3 7 4 12 5 6 11 0 14 9 2
7 11 4 1 9 12 14 2 0 6 10 13 15 3 5 8
2 1 14 7 4 10 8 13 15 12 9 0 3 5 6 11

(4)、从（1）处循环执行，直到B[8]被替代完成。

(5)、将B[1]到B[8]组合，按下表（P）变换，得到P。

P盒置换

16 7 20 21
29 12 28 17
1 15 23 26
5 18 31 10

2 8 24 14
32 27 3 9
19 13 30 6
22 11 4 25

5、异或 P 和 L[I-1]结果放在 R[I]，即 R[I]=P XOR L[I-1]。
6、L[I]=R[I-1]
7、从 2-4-1 处开始循环执行，直到 K[16]被变换完成。
8、组合变换后的 R[16]L[16]（注意：R 作为开始的 32 位），按下表（IP-1）变换得到最后的结果。

IP^{-1} 置换

40 8 48 16 56 24 64 32
39 7 47 15 55 23 63 31
38 6 46 14 54 22 62 30
37 5 45 13 53 21 61 29
36 4 44 12 52 20 60 28
35 3 43 11 51 19 59 27
34 2 42 10 50 18 58 26
33 1 41 9 49 17 57 25

DES 算法的一次迭代运算过程如下图所示。

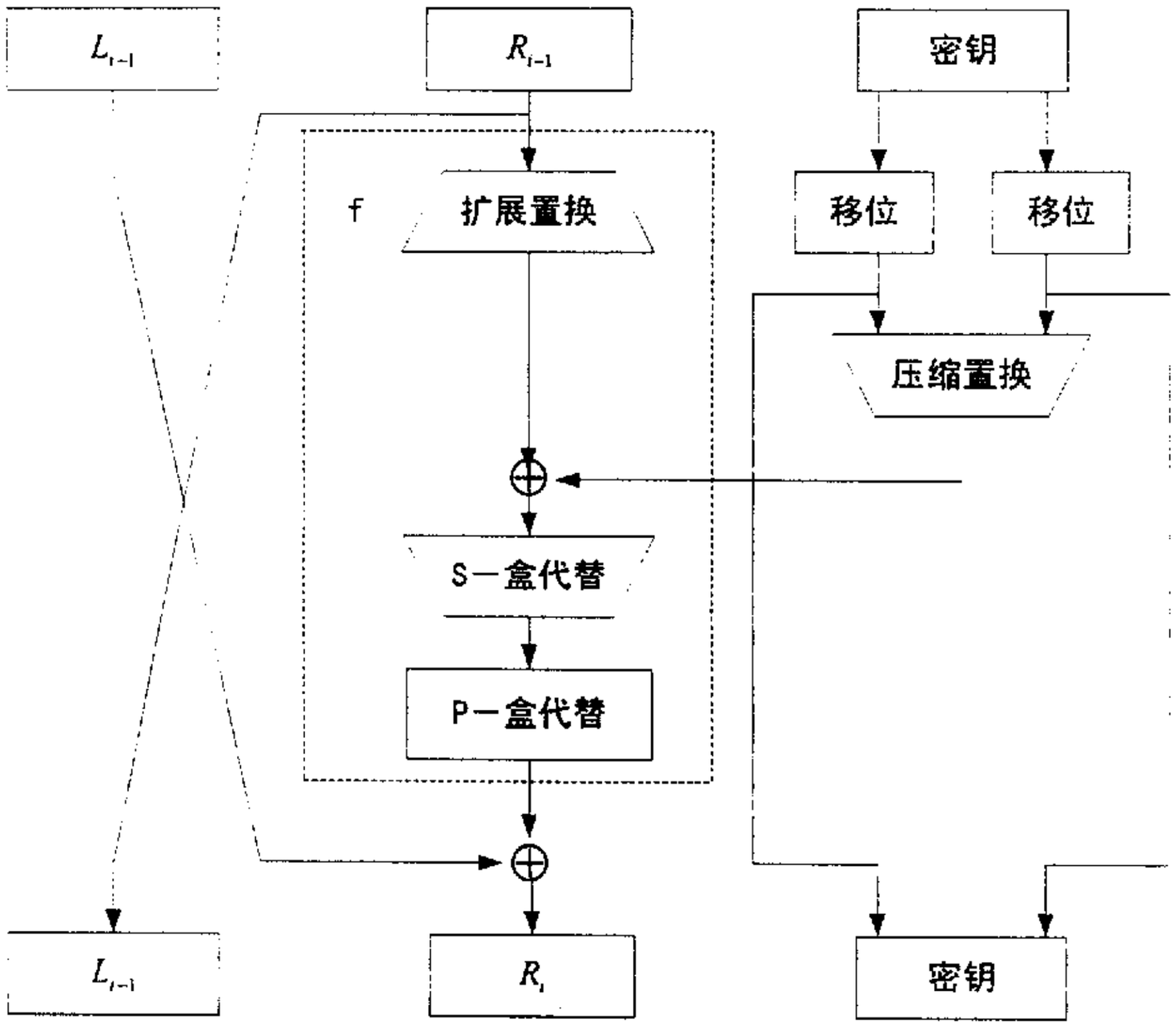


图 4-3 DES 一次迭代过程

其中，函数 f 的计算过程如下图所示。

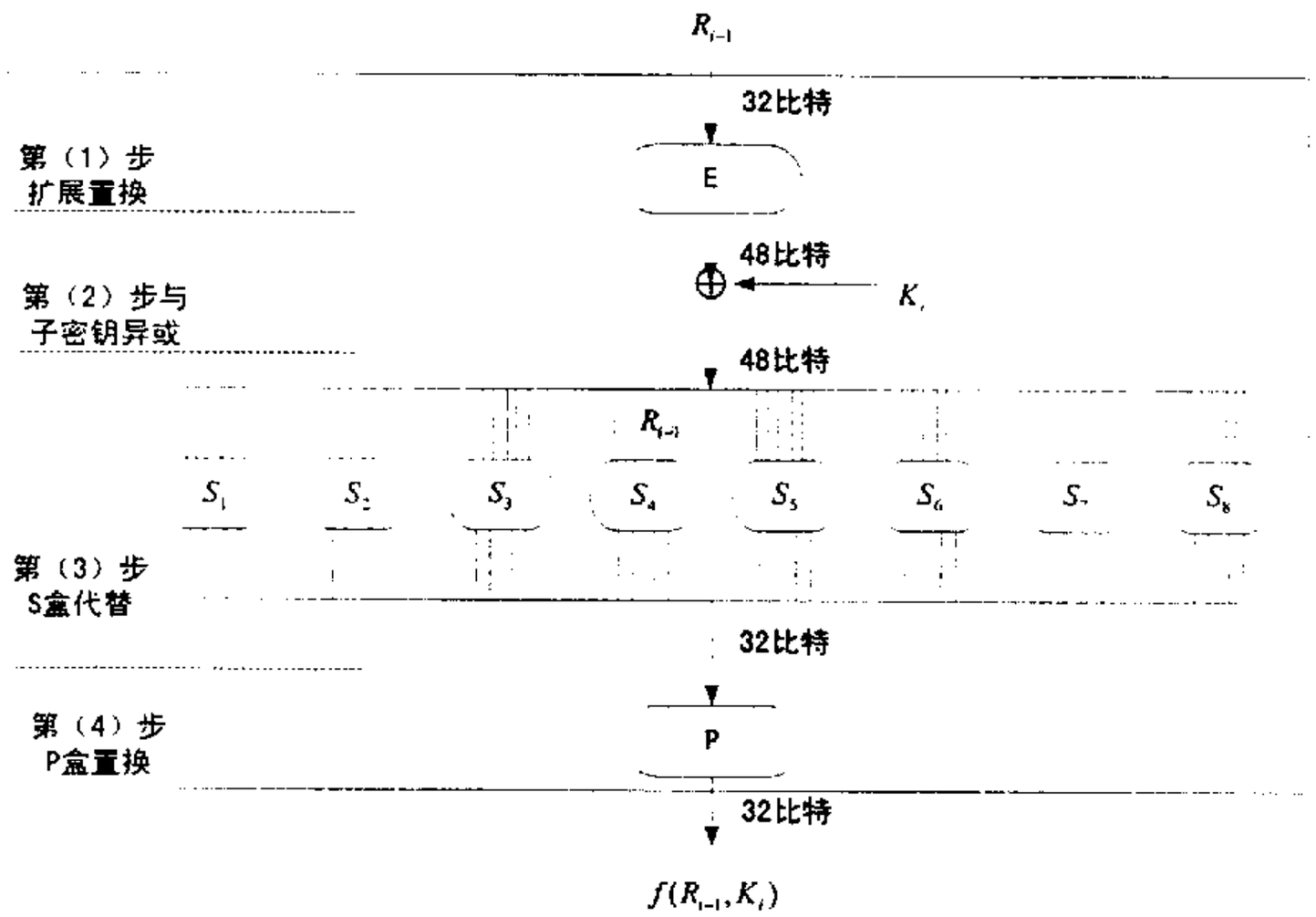


图 4-4 函数 f 的计算过程

3.1.3 DES 解密算法的实现

DES 解密算法与加密算法共用了相同的算法过程，两者的不同之处仅在于解密时子密钥的使用顺序与加密时相反。如果加密的子密钥为 $K_1,K_2,...,K_{16}$，那么解密时子密钥的使用顺序应该为 $K_{16},K_{15},...,K_1$，即：使用 DES 解密算法进行解密时，将以 64 位密文作为输入，第一轮迭代运算使用子密钥 K_{16}；第二轮迭代运算使用子密钥 K_{15}，……，第 16 次迭代运算使用子密钥 K_1，其他的运算过程与加密算法相同，这样，最后输出的便是 64 位明文。

3.1.4 DES 加解密算法接口描述

该接口实现了对输入数据的 DES 加密/解密功能。接口定义了下面一个标准 C 函数：

类型：标准 C 函数

函数名：DES
功能：　实现了对输入数据的 DES 加密/解密功能
声明：　viod DES(mark,key,input,output);
　　unsigned long int mark/*加密/解密选择标志：加密为 0，解密为 1*/
　　unsigned long int key[2] /*DES 加密/解密密钥*/

　　unsigned long int input[2] /*需加密/解密的数据*/
　　unsigned long int len/*加密/解密结果*/
返回值：加密/解密的输出结果
　　当参数 mark 为 0 时，以下列形式输出：
　　加密后的密文是：********
　　当参数 mark 为 1 时，以下列形式输出：
　　解密后的原文是：********

调用 DES 函数接口的一个实例：
　　请输入加密解密选择：加密为 0，解密为 1
　　0
　　请输入 64 位加密密钥：
　　15432ad2
　　ea5c370a
　　请输入需加密的数据：
　　11111111
　　22222222
　　加密后的密文是：
　　c01d7421
　　8c50043

　　请输入加密解密选择：加密为 0，解密为 1
　　1
　　请输入 64 位解密密钥
　　15432ad2
　　ea5c370a
　　请输入需解密的数据
　　c01d7421
　　8c50043
　　解密后的原文是：

11111111
22222222

3.2 单向散列函数 MD5 算法

3.2.1 MD5 算法描述

MD5 的全称是 Message-Digest Algorithm 5，在 90 年代初由 MIT 的计算机科学实验室和 RSA Data Security Inc 发明，经 MD2、MD3 和 MD4 发展而来。

Message-Digest 泛指字节串(Message)的 Hash 变换，就是把一个任意长度的字节串变换成一定长的大整数。

MD5 将任意长度的“字节串”变换成一个 128bit 的大整数，并且它是一个不可逆的字符串变换算法，换句话说就是，即使你看到源程序和算法描述，也无法将一个 MD5 的值变换回原始的字符串。

MD5 的典型应用是对一段字节串产生指纹，以防止被“篡改”。举个例子，你将一段话写在一个叫 readme.txt 文件中，并对这个 readme.txt 产生一个 MD5 的值并记录在案，然后你可以传播这个文件给别人，别人如果修改了文件中的任何内容，你对这个文件重新计算 MD5 时就会发现。如果再有一个第三方的认证机构，用 MD5 还可以防止文件作者的“抵赖”，这就是所谓的数字签名应用。

MD5 还广泛用于加密和解密技术上，在很多操作系统中，用户的密码是以 MD5 值（或类似的其它算法）的方式保存的， 用户登陆的时候，系统是把用户输入的密码计算成 MD5 值，然后再去和系统中保存的 MD5 值进行比较，而系统并不“知道”用户的密码是什么。

3.2.2 MD5 算法的实现

1、附加填充比特

对消息进行填充，使消息的长度（比特数）与 448 模 512 同余，即恰好为一个比 512 比特的倍数仅小 64 位的数。

填充方法是在消息的比特流后面附加一个 1，后接所要求的多个 0。即使消息的长度在填充之前已经满足条件，附加填充比特总是需要进行的。例如，如果消息的长度为 448 比特长，那么将填充 512 比特形成 960 比特的报文。

2、附加消息长度值

将用 64 比特表示的初始消息（填充前）的长度（比特数）附加在上一步的结果后。如果初始消息的长度大于2^{64}，那么仅仅使用长度的低 64 比特进行填充。填充方法是把 64 比特的长度看成是两个 32 比特的字，低 32 比特字先填充，高 32 比特字后填充。

我们可以把前面两步看作是算法的预处理过程，它们的作用是使消息长度恰好是 512 比特的整数倍。在后面的算法步骤中，算法的处理正是按照 512 比特的分组来进行的。

3、 初始化 MD 缓存

MD5 算法使用了一个 4 个字（128 比特，MD4 中每个字 32 比特）的缓存来计算消息摘要，它们主要用来存放 MD5 的中间及最终结果。缓存可以看成是 4 个 32 比特的寄存器（A、B、C、D）。这些寄存器被初始化为如下的整数值（以十六进制表示）：

A＝0x01234567

B＝0x89ABCDEF

C＝0xFEDCBA98

D＝0x76543210

4、以 512 比特（16 个字）分组处理消息

这是 MD5 算法的主循环，它以 512 比特作为分组，从消息开头循环地处理消息序列分组，直至消息的结尾，然后输出散列值。因此，主循环的循环次数将是消息中 512 比特消息分组的数目（即预处理后消息的长度/512）。

每一次循环都以当前处理的 512 比特分组和 MD 缓存 A、B、C、D 作为输入，直接在 MD 缓存 A、B、C、D 上进行运算，并在本次循环结束时利用所保存的循环前 MD 缓存的值来更新缓存内容，然后进行下一个分组的处理或者输出。

主循环中包含了共 64 步操作，所有 64 步操作中分别使用了 4 个非线性函数。按照他们所使用的非线性函数的不同，这些操作可以明显地分为四轮：每一轮 16 步操作（第一轮包括操作 0 至操作 15，第二轮包括操作 16 至操作 31，第三轮包括操作 32 至操作 47，第四轮包括操作 48 至操作 63），每一轮的所有操作都使用了相同的非线性函数；而轮与轮之间则使用了不同的非线性函数。

首先定义 4 个辅助函数，每个函数的输入是三个 32 位的字，输出是一个 32 位的字。X，Y，Z 为 32 位整数。

F(X,Y,Z)=(X∧Y)v(not(X)∧Z)　　G(X,Y,Z)=(X∧Z)v(Y∧not(Z))

H(X,Y,Z)=XxorYxorZ　　I(X,Y,Z)=Yxor(Xvnot(Z))

具体过程如下：

```
    每一次，把512位分组数据原文存放在16个32位元素的数组X中.
        X[0],X[1],….,X[16]
    然后：AA = A    BB = B    CC = C    DD = D
/* 第1轮*/
/* 以 [abcd k s i]表示如下操作
注：T[i]是2^32×abs（sin（i）的整数部分，i的单位是弧度。
a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */
[ABCD 0 7 1] [DABC 1 12 2] [CDAB 2 17 3] [BCDA 3 22 4]
[ABCD 4 7 5] [DABC 5 12 6] [CDAB 6 17 7] [BCDA 7 22 8]
[ABCD 8 7 9] [DABC 9 12 10] [CDAB 10 17 11] [BCDA 11 22 12]
[ABCD 12 7 13] [DABC 13 12 14] [CDAB 14 17 15] [BCDA 15 22 16]
/* 第2轮* */
/* 以 [abcd k s i]表示如下操作
a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */
[ABCD 1 5 17] [DABC 6 9 18] [CDAB 11 14 19] [BCDA 0 20 20]
[ABCD 5 5 21] [DABC 10 9 22] [CDAB 15 14 23] [BCDA 4 20 24]
[ABCD 9 5 25] [DABC 14 9 26] [CDAB 3 14 27] [BCDA 8 20 28]
[ABCD 13 5 29] [DABC 2 9 30] [CDAB 7 14 31] [BCDA 12 20 32]
/* 第3轮*/
/* 以 [abcd k s i]表示如下操作
a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */
[ABCD 5 4 33] [DABC 8 11 34] [CDAB 11 16 35] [BCDA 14 23 36]
[ABCD 1 4 37] [DABC 4 11 38] [CDAB 7 16 39] [BCDA 10 23 40]
[ABCD 13 4 41] [DABC 0 11 42] [CDAB 3 16 43] [BCDA 6 23 44]
[ABCD 9 4 45] [DABC 12 11 46] [CDAB 15 16 47] [BCDA 2 23 48]
/* 第4轮*/
/* 以 [abcd k s i]表示如下操作
a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */
[ABCD 0 6 49] [DABC 7 10 50] [CDAB 14 15 51] [BCDA 5 21 52]
[ABCD 12 6 53] [DABC 3 10 54] [CDAB 10 15 55] [BCDA 1 21 56]
[ABCD 8 6 57] [DABC 15 10 58] [CDAB 6 15 59] [BCDA 13 21 60]
[ABCD 4 6 61] [DABC 11 10 62] [CDAB 2 15 63] [BCDA 9 21 64]
/* 然后进行如下操作 */
A = A + AA
B = B + BB
C = C + CC
D = D + DD
```

5、输出结果

报文摘要的产生后的形式为：A，B，C，D。也就是低位字节 A 开始，高位字节 D 结束。

3.2.3 MD5 加密算法接口描述

该接口实现了对输入数据的 MD5 算法摘要功能。接口定义了下面一个标准 C 函数：

类型：标准 C 函数

函数名：MD5

功能：实现了对输入数据的 MD5 算法摘要

声明：
```
void MD5 (state[4],digest[16],input,inputLen);
 unsigned long int state[4]  /*MD5 需要的初始化值*/
unsigned char digest[16]/*抽取后得到的 MD5 摘要*/
unsigned char *input/*需要抽取摘要的数据*/
unsigned int inputLen/*需要抽取摘要的数据的长度*/
```

输出结果：加密的输出结果即生成的数据摘要

调用 MD5 函数接口的一个实例：

输入的要进行加密的数据为:123456789

数据长度为:9

MD5 摘要：25f9e794323b453885f5181f1b624d0b

3.3 公开密钥算法 RSA

3.3.1 RSA 算法描述

3.3.1.1 RSA 算法

RSA 是第一个既能用于数据加密也能用于数字签名的算法。它易于理解和操作，也很流行。RSA 的安全性依赖于大数分解。公钥和私钥都是两个大素数（ 大于 100 个十进制位）的函数。据猜测，从一个密钥和密文推断出明文的难度等同于分解两个大素数的积。

1、密钥对的产生：

选择两个大素数，p 和 q 。计算：

n = p * q

然后随机选择加密密钥 e（在 PKSC#1 中推荐使用 3 和 65537，经过理论证明，这两个数都不影响安全级别），要求 e 和(p-1)*(q-1)互质。最后，利用欧几里德算法计算解密密钥 d，满足

e * d = 1 (mod (p - 1) * (q - 1))

其中 n 和 d 也互质。数 e 和 n 是公钥，d 是私钥。两个素数 p 和 q 不再需要，应该丢弃，不要让任何人知道。

2、加密过程

加密信息 m（二进制表示）时，首先把 m 分成等长数据块 ml ，m2,...， mi，块长 s，其中 s 是小于 n 的 2 的最大次幂。对应的密文是：

ci = mi^e (mod n) (a)

解密时作如下计算：

mi = ci^d (mod n) (b)

RSA 可用于数字签名，用(b)式签名，(a)式验证。具体操作时考虑到安全性和 m 信息量较大等因素，一般是先作 HASH 运算。

3.3.1.2 RSA 中有关的基本数学理论

1、欧几里德算法(寻找两个数的最大公约数)

现在被称作欧几里德算法的找两个数最大公约数的步骤是在古希腊（公元前 300 年左右）即有记载的一个最熟知的例子。让我们看看这是如何进行的。随意取两个特定的数，譬如讲 1365 和 3654。所谓的最大公约数是可以同时整除这两个数的最大的整数。在应用欧几里德算法时，我们让这两数中的一个被另一个除并取余数，在 3654 中取出 1365 的两倍，其余数为 924（=3654-2730）。我们现在用此余数即 924 以及我们刚用的除数即 1365 去取代原先的两个数。我们用这一对新的数重复上述步骤，用 924 去除 1365，余数为 441。这又得到新的一对 441 和 924，我们用 441 除 924，得到余数 42（=924-882），等等，直到能够被整除为止。我们把这一切如下列出：

3654÷1365	给出余数 924
1365÷924	给出余数 441
924÷441	给出余数 42
441÷42	给出余数 21
42÷21	给出余数 0。

我们最后用于做除数的 21 即是所需要的最大公约数。

欧几里德算法本身是我们寻找这一因子的系统步骤。

2、Rabin-Miller 算法

首先选择一个待测的随机数 p，计算 b，b 是 2 整除 p-1 的次数（即，2^b 是能整除 p-1 的 2 的最大幂数）。然后计算 m，使得 $n=1+2^b m$。

1、选择一个小于 p 的随机数 a;

2、设 j=0 且 $z=a^m \bmod p$；

3、如果 z=1 或 z=p-1，那么 p 通过测试，可能是素数；

4、如果 j>0 且 z=1，那么 p 不是素数；

5、设 j=j+1，如果 j<b 且 $z \neq p-1$,设$z=z^2 \bmod p$，然后回到第四步。如果 z=p-1，那么 p 通过测试，可能是素数。

6、如果 j=b 且 $z \neq p-1$，那么 p 不是素数。

3.3.2 RSA 算法的实现

3.3.2.1 大素数的产生

1、生成一个足够长的随机数 p;

2、将最高位和最低位都置为 1（设高位是为了保证位数，设低位是为了保证位奇数）;

3、确保 p 不能被任何小于 2000 的素数整除；

4、对某随机数 a 运行 Rabin-Miller 测试。如果 p 通过测试，则另外产生一个随机数 a 再重新进行测试。选取较小的 a 值，以保证较快的计算速度。做五次测试，若 p 在其中的一次失败，则重新产生一个随机数 p 进行测试。

3.3.2.1 产生其他 RSA 参数

产生大素数是最关键的，在产生大素数之后根据前面描述的各参数之间的关系产生出所需的参数 n,e,d，其中 n,e 是公钥，d 是私钥。

3.3.2.1 利用已产生的参数进行 RSA 计算

根据RSA计算原理完成公钥加密私钥解密和私钥加密公钥解密的功能。

3.3.3 RSA 加解密函数接口描述

3.3.3.1 RSA 秘/公钥数据结构

1) RSA 公钥数据结构

```
R_RSA_PUBLIC_KEY
typedef struct {
unsigned int bits;
unsigned char modulus[MAX_RSA_MODULUS_LEN];
unsigned char exponent[MAX_RSA_MODULUS_LEN];
} R_RSA_PUBLIC_KEY;
```

R_RSA_PUBLIC_KEY 类型数据是用来存放一个 RSA 公钥。

参数说明:

bits: modulus 的长度（按 bits 计算）

(MIN_RSA_MODULUS_BITS<bits<MAX_RSA_MODULUS_BITS).

modulus 是模数 n;

publicExponent 是公开指数 e。

2) RSA 私钥数据结构

```
R_RSA_PRIVATE_KEY
typedef struct {
unsigned int  Version;
unsigned int  bits;
unsigned char modulus[MAX_RSA_MODULUS_LEN];
unsigned char publicExponent[MAX_RSA_MODULUS_LEN];
unsigned char exponent[MAX_RSA_MODULUS_LEN];
unsigned char prime[2][MAX_RSA_PRIME_LEN];
unsigned char primeExponent[2][MAX_RSA_PRIME_LEN];
unsigned char coefficient[MAX_RSA_PRIME_LEN];
} R_RSA_PRIVATE_KEY;
```

R_RSA_PRIVATE_KEY 类型的数据是用来存放一个 RSA 私钥。

bits: modulus 的长度（按 bits 计算）

(MIN_RSA_MODULUS_BITS<bits<MAX_RSA_MODULUS_BITS).

R_RSA_PRIVATE_KE 类型的字段有下列含义：

- version 是一个为兼容将来此协议的修改的版本号。为了适应此协议的版本它应该是 0；
- modulus 是模数 n；
- publicExponent 是公开指数 e；
- prime[1]是组成模数 n 的一个素数 p；
- prime[2]是组成模数 n 的一个素数 q；
- primeExponentd[1] mod (p-1)；
- primeExponent 是[2]d mod (q-1)；
- coefficient 是中国剩余理论中的系数 q-1 mod p。

3.3.3.2 私钥加密接口描述

该接口实现了对输入数据的 RSA 私钥加密功能。接口定义了下面一个标准 C 函数：

类型：标准 C 函数

函数名：RSAPrivateEncrypt

功能：实现了对输入数据的 RSA 私钥加密功能

声明：
```
int RSAPrivateEncrypt(output, outputLen, input,
inputLen, privateKey)
unsigned char *output;/* 加密后的数据*/
unsigned int *outputLen; /*加密后的数据长度*/
unsigned char *input;    /*被加密的数据区  */
unsigned int inputLen; /*被加密的数据长度*/
R_RSA_PRIVATE_KEY *privateKey; /*加密私钥*/
```

出口参数：0 加密成功

1 加密失败

3.3.3.3 私钥解密接口描述

该接口实现了对输入数据的 RSA 私钥解密功能。接口定义了下面一个标准 C 函数：

类型：标准 C 函数

函数名：RSAPrivateDecrypt

功能：实现了对输入数据的 RSA 私钥解密功能

```
声明：  int RSAPrivateDecrypt (output, outputLen, input,
        inputLen, privateKey)
       unsigned char *output;/* 解密后的数据*/
       unsigned int *outputLen; /*解密后的数据长度*/
       unsigned char *input;   /*被解密的数据区  */
       unsigned int inputLen; /*被解密的数据长度*/
       R_RSA_PRIVATE_KEY *privateKey; /*解密私钥*/
出口参数：0 解密成功
          1 解密失败
```

3.3.3.4 公钥加密接口描述

该接口实现了对输入数据的 RSA 公钥加密功能。接口定义了下面一个标准 C 函数：

```
类型：标准 C 函数
函数名：RSAPublicEncrypt
功能：  实现了对输入数据的 RSA 公钥加密功能
声明：  int RSAPublicEncrypt (output, outputLen, input,
        inputLen, publicKey, randomStruct)
       unsigned char *output;/* 加密后的数据*/
       unsigned int *outputLen; /*加密后的数据长度*/
       unsigned char *input;   /*被加密的数据区  */
       unsigned int inputLen; /*被加密的数据长度*/
       R_RSA_PUBLIC_KEY *publicKey; /*加密公钥*/
       R_RANDOM_STRUCT *randomStruct;/* 随机数据结构体*/
出口参数：0 加密成功
          1 加密失败
```

3.3.3.5 公钥解密接口描述

该接口实现了对输入数据的 RSA 公钥解密功能。接口定义了下面一个标准 C 函数：

```
类型：标准 C 函数
函数名：RSAPublicDecrypt
功能：  实现了对输入数据的 RSA 公钥解密功能
声明：  int RSAPublicDecrypt (output, outputLen, input,
        inputLen, publicKey)
```

```
        unsigned char *output;/* 解密后的数据*/
        unsigned int *outputLen; /*解密后的数据长度*/
        unsigned char *input;    /*被解密的数据区   */
        unsigned int inputLen; /*被解密的数据长度*/
        R_RSA_PUBLIC_KEY *publicKey; /*解密公钥*/
出口参数: 0 解密成功
          1 解密失败
```

3.4 密码技术在 PKI 中的应用和实现

密码技术在 PKI 中的应用作者在第二章已经进行了详细的介绍，包括加密/解密、数字签名、信息摘要、数字信封等。应用基本的加解密算法 DES、MD5、RSA 或者他们的算法组合可以实现上述加密算法在 PKI 中的各种应用。

3.4.1 加解/解密的实现

应用 DES 算法实现对数据的对称加密/解密，应用 RSA 算法实现非对称算法的公钥加密/解密和私钥加密/解密。

接口描述：

函数名：CryptionProc

功　能：实现了对输入数据的加密/解密

声　明：

```
int    CryptionProc(attributes,  keyDERString,
keyStringLen, bEncryption, inData, inLen, outData,
outLen)
ATTRIB_TYPE attributes;/* 加密/解密密钥的操作类
型*/
UCHAR * keyDERString; /*加密/解密中所用的密钥(为
DER 编码串)*/
UINT4 keyStringLen; /*加密/解密中所用密钥的字节
数，即 keyDERString 的字节数*/
UINT4 bEncryption; /*当值为 ENCRYPT 时，表示加密
```

过程；当值为 DECRYPT 时，表示解密过程*/
UCHAR * inData; /*解密/加密输入数据*/
UINT4 inLen; /*解密/加密输入数据的字节数*/
UCHAR * outData; /*加密/解密输出结果*/
UINT4 * outLen;/* 是一个输入/返回类型，其输入值指定 outData 缓冲的大小,输出加密/解密结果的字节数*/

出口参数：0 加密/解密成功
1 加密/解密失败

3.4.2 消息摘要的实现

应用 MD5 算法可对原始数据制作消息摘要
接口描述：
函数名：DigestProc
功　能：　实现了对输入数据的消息摘要
声明：　int DigestProc (attributes,indata, inlen, mddata, mdlen)
ATTRIB_TYPE attributes; /* 消息摘要的操作类型*/
UCHAR * indata; /*需进行消息摘要的数据*/
UINT4 inlen; /*需进行消息摘要的数据长度*/
UCHAR * md; /*消息摘要的输出结果*/
UINT4 * mdlen;/*输出消息摘要的长度*/
出口参数：0 制作消息摘要成功
1 制作消息摘要失败

3.4.3 数字签名的实现

应用 MD5 和 RSA 算法，实现对输入数据的数字签名
接口描述：
函数名：SignProc
功　能：　实现了对输入数据的数字签名

声明： int SignProc(attributes, keyDERString, keyStringLen, indata, inlen, outdata, outlen)

ATTRIB_TYPE attributes; /* 数字签名的操作类型*/

UCHAR * keyDERString; /*签名所用的密钥(为 DER 编码串)*/

UINT4 keyStringLen; /*签名中所用密钥的字节数，即 keyDERString 的字节数*/

UCHAR * indata; /*需进行签名的数据*/

UINT4 inlen; /*需进行签名的数据长度*/

UCHAR * outdata; /*签名后的输出结果*/

UINT4 * outlen;/*输出签名结果的长度*/

出口参数：0 签名成功

1 签名失败

3.4.4 数字信封的实现

综合应用 DES、MD5、RSA 算法实现对数据的数字信封。

函数名：EnvelopProc

功　能：　实现了对数据的数字信封

接口描述：

声　明：　int EnvelopProc (attributes,keyDERString, keyStringLen, openOrSeal,indata, inlen, outdata, outlen)

ATTRIB_TYPE attributes; /* 数字信封的操作类型*/

UCHAR * keyDERString; /*制作数字信封中用到的密钥*/

UINT4 keyStringLen; /*制作数字信封中用到的密钥长度*/

UINT4 openOrSeal;/* 值为 OPEN 表示打开信封，为 SEAL 表示封装信封*/

UCHAR * indata; /*输入数据*/

UINT4 inlen; /*输入数据长度*/

```
        UCHAR * outdata; /*输出结果*/
        UINT4 * outlen;/*输出结果的长度*/
出口参数：0 操作成功
          1 操作失败
```

第四章 base64编码及其实现

4.1 base64编码简介

Base 64编码常用在限制使用US-ASCII数据格式的情况下，用于数据的存储和传输。在PKI系统数据传输的过程中使用了Base 64编码规则。

Base 64 Encode Class将二进位Binary档案转换成ASCII格式，Base 64 Decode Class将编码过的ASCII格式内容转回二进位档。

Base 64编码设计用来表示任意八位的数据序列，它的表现形式不需要人能够读懂。

Base 64编码需要用到由65个元素组成的US-ASCII字符集，每个字符用6bit表示。(第65个字符“=”是用来标志特殊处理功能的)。

编码过程将24—bit数据编码为4个字符输出。24—bit数据是由3个8bit的输入数据组成的。这24个bit就可以被当作4组6-bit数据，每一组都可以被编码为一个base 64字符集中独立的字符。

如果在输入数据的末尾，需要被编码的数据少于24 bits就需要进行特殊的处理，即在输出端数据的末尾以“=”进行填充。由于base 64的输入是8的整数倍，因此只有以下三种情况：

1、 末尾编码输入的位数是24 bits的整数倍。此时编码输出是4个字符的整数倍，没有“=”填充。

2、 末尾编码输入的位数正好是8 bits。此时编码输出的末尾是两个字符后面再填充两个“=”。

3、 末尾编码输入的位数正好是16 bits。此时编码输出的末尾是三个字符后面再填充一个“=”。

Base 64编码字母表如下表所示。

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w	(pad)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

表 5-1 Base 64 编码字母表

下面是一个 base 64 编码的例子。

```
Input data:  0x14fb9c03d97e
Hex:     1   4    f   b    9   c    | 0   3    d   9    7   e
8-bit:   00010100 11111011 10011100  | 00000011 11011001
11111110
6-bit:   000101 001111 101110 011100 | 000000 111101 100111
111110
Decimal: 5      15     46     28     0      61     37     62
Output:  F      P      u      c      A      9      l      +

Input data:  0x14fb9c03d9
Hex:     1   4    f   b    9   c    | 0   3    d   9
8-bit:   00010100 11111011 10011100  | 00000011 11011001
                                       pad with 00
6-bit:   000101 001111 101110 011100 | 000000 111101 100100
Decimal: 5      15     46     28     0      61     36
                                                  pad with =
Output:  F      P      u      c      A      9      k      =

Input data:  0x14fb9c03
Hex:     1   4    f   b    9   c    | 0   3
8-bit:   00010100 11111011 10011100  | 00000011
                                       pad with 0000
6-bit:   000101 001111 101110 011100 | 000000 110000
Decimal: 5      15     46     28     0      48
                                           pad with =      =
Output:  F      P      u      c      A      w      =      =
```

4.2 base64 编码的实现

base64 编码实现流程图如下图所示：

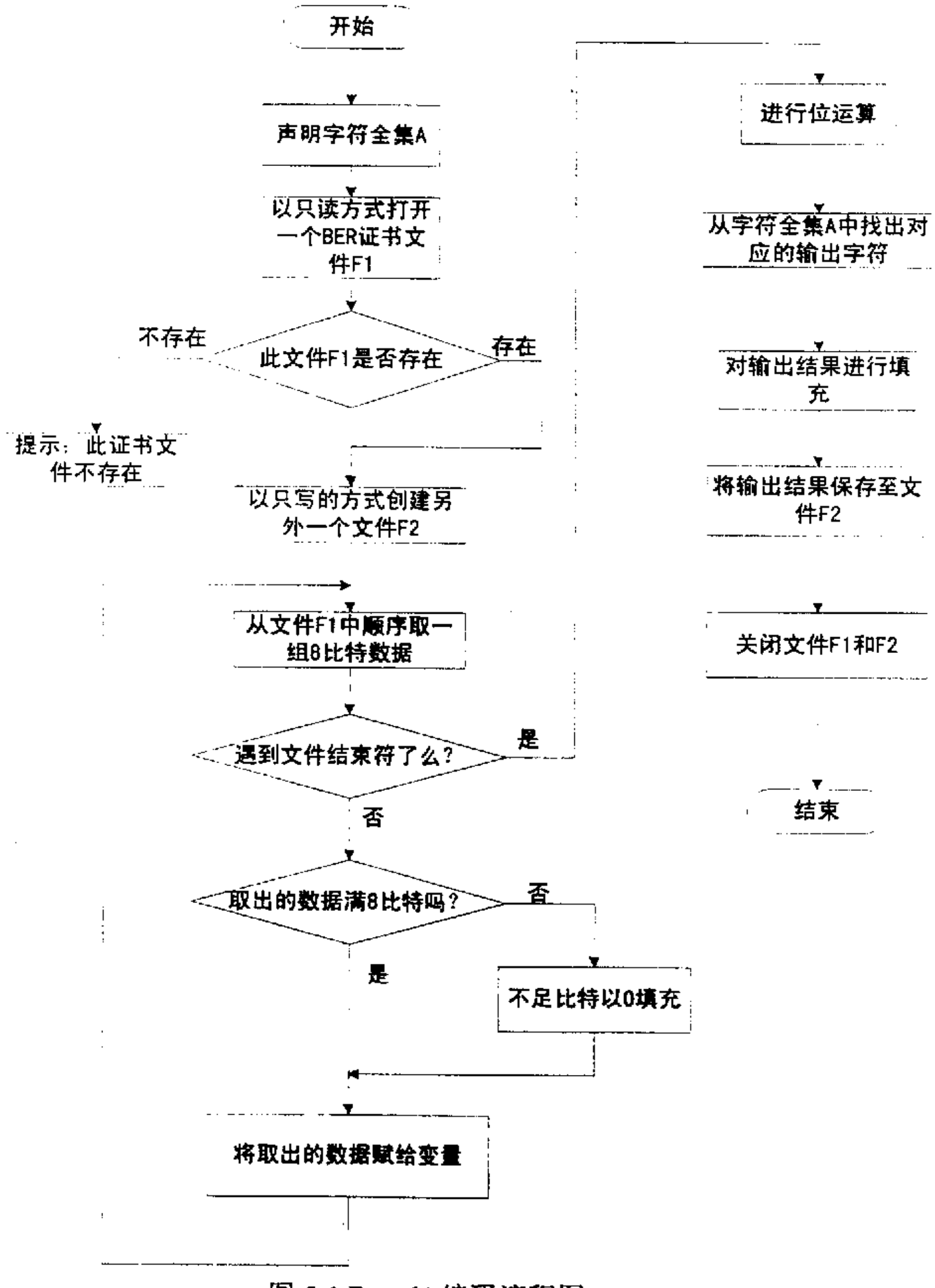


图 5-1 Base64 编码流程图

4.3 base64 解码的实现

base64 解码实现流程图如下图所示：

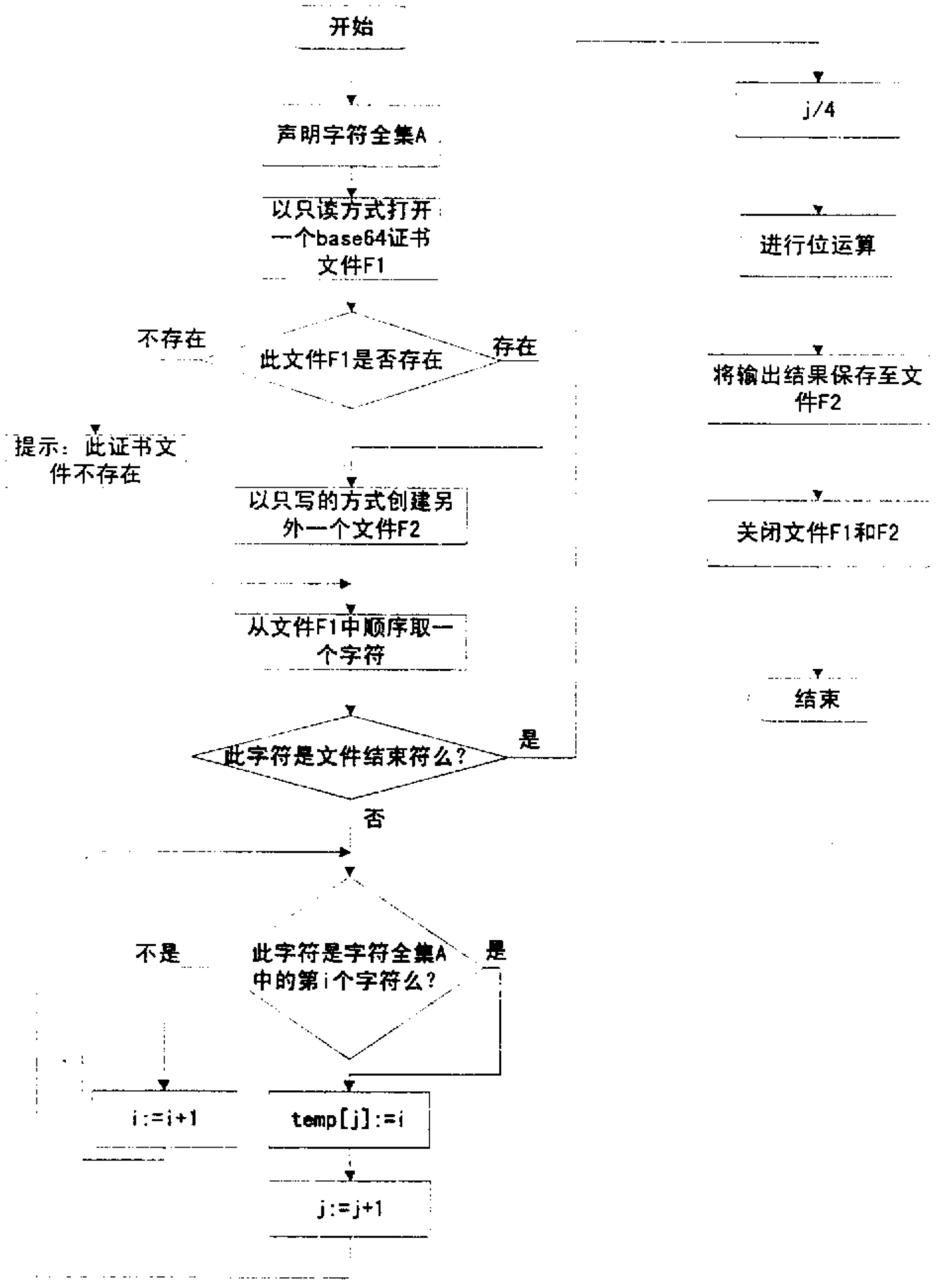


图 5-2 Base64 解码流程图

4.4 接口描述

4.4.1 base64 编码函数接口描述

该接口实现了对输入文件的 base 64 编码。接口定义了下面一个标准 C 函数：

类型：标准 C 函数

函数名：base64encrypt

功能：实现了对输入文件的 base64 编码

声明：void base64encrypt (fp1, fp2);

FILE *fp1/*需进行 base64 编码的文件*/

FILE *fp2/*进行完 base64 编码后生成的 base64 格式文件*/

输出结果：生成的 base64 格式文件

4.4.2 base64 解码函数接口描述

该接口实现了对输入文件的 base64 解码。接口定义了下面一个标准 C 函数：

类型：标准 C 函数

函数名：base64decrypt

功能：实现了对输入文件的 base 64 解码

声明：void base64decrypt (fp1, fp2);

FILE *fp1/*需进行 base64 解码的文件*/

FILE *fp2/*进行完 base64 解码后生成的二进制格式文件*/

输出结果：生成的 BER 格式文件

4.5 BER 格式和 base 64 格式证书对照实例

这里看一下我所做的 base 64 编码的一个实例，以便比较。下图所示为一个 BER 格式证书。

4.4 接口描述

4.4.1 base64 编码函数接口描述

该接口实现了对输入文件的 base 64 编码。接口定义了下面一个标准 C 函数：

类型：标准 C 函数

函数名：base64encrypt

功能： 实现了对输入文件的 base64 编码

声明： void base64encrypt (fp1, fp2);
FILE *fp1/*需进行 base64 编码的文件*/
FILE *fp2/*进行完 base64 编码后生成的 base64 格式文件*/

输出结果：生成的 base64 格式文件

4.4.2 base64 解码函数接口描述

该接口实现了对输入文件的 base64 解码。接口定义了下面一个标准 C 函数：

类型：标准 C 函数

函数名：base64decrypt

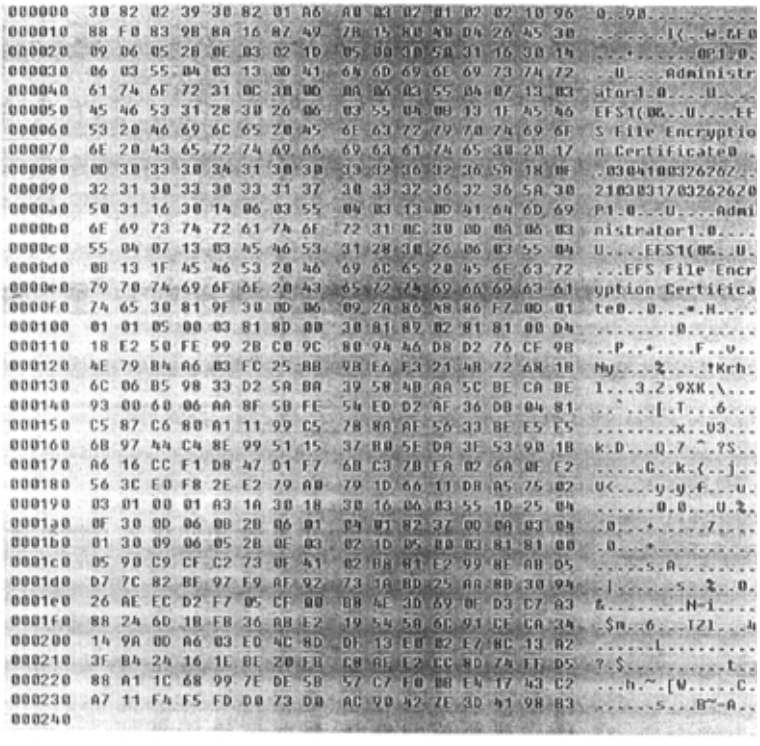
功能： 实现了对输入文件的 base 64 解码

声明： void base64decrypt (fp1, fp2);
FILE *fp1/*需进行 base64 解码的文件*/
FILE *fp2/*进行完 base64 解码后生成的二进制格式文件*/

输出结果：生成的 BER 格式文件

4.5 BER 格式和 base 64 格式证书对照实例

这里看一下我所做的 base 64 编码的一个实例，以便比较。下图所示为一个 BER 格式证书。



经过 base64 编码后的形式为下图所示：

第五章 SSL 协议分析及其实现

5.1 引言

安全套接字层（SSL）协议是 Netscape 公司于 1993 年提出的一个网络安全通信协议，在 Internet 上为通信双方提供可靠连接方式下的防窃听、防篡改、防信息伪造的秘密通信，SSL 最初是通过加密 http 连接为 web 浏览器提供安全而引入的，现在已成为通用 Internet 服务的安全工具，目前已被工业界认可，在电子邮件、Netscape Navigator 和 IE 等网络浏览器、Oracle Applocation Server 等服务器上已广泛应用。

5.2 SSL 协议分析

SSL 协议运行在传输层 TCP 协议和应用层协议（HTTP、LDAP、LMAP）之间。用 TCP/IP 代表高级协议在数据通信过程中允许一个支持 SSL 的 WWW 服务器在支持 SSL 的客户端使协议本身获得信任、使客户端得到服务器的信任，从而在两台机器间建立一个可靠的加密传输连接。要求 SSL 连接的网页名称以“https://”开头，而不是以“http://”开头。SSL 协议的位置如图 6-1 所示：

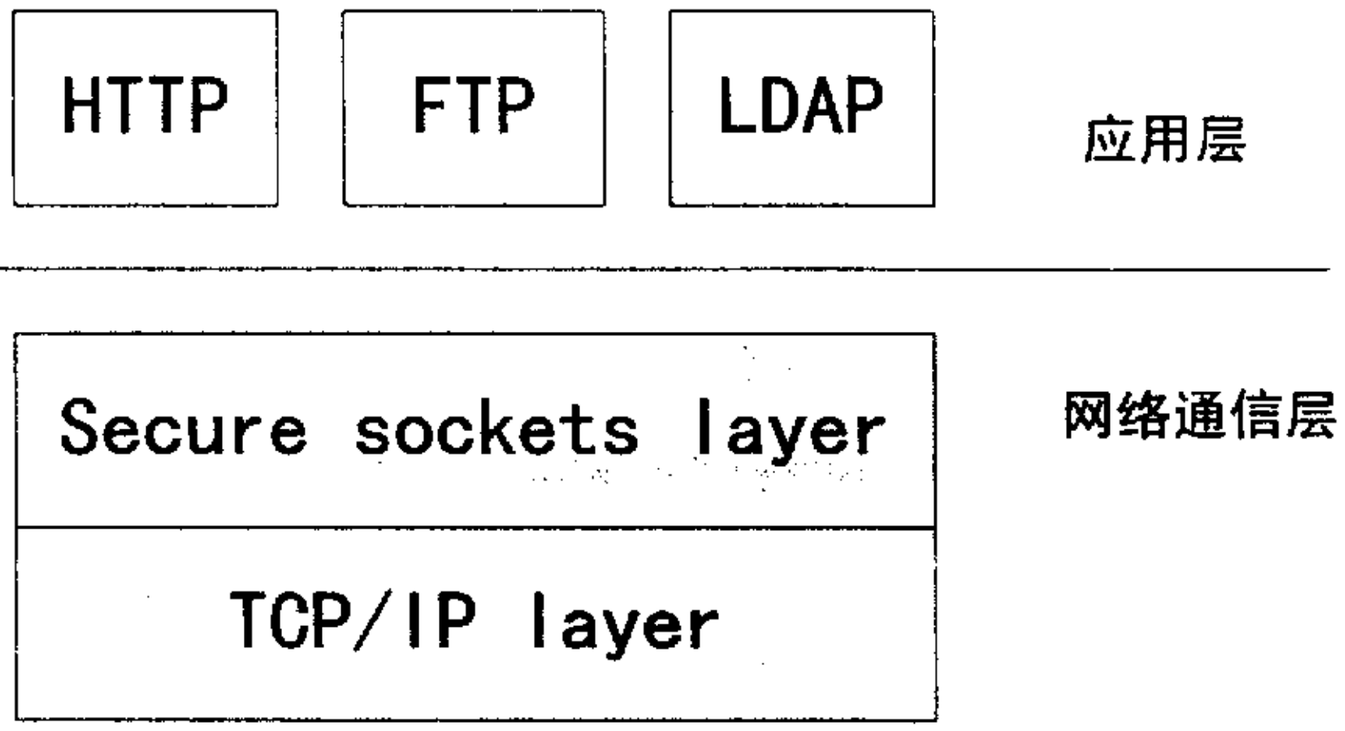


图 6-1 SSL 协议位置示意图

SSL 协议是一个分层协议，由上层的握手协议和下层的记录协议组成。其中握手协议负责在交换应用层协议数据之前协商加密算法与密钥，记录层协议负责封装高层协议（包括握手协议）的数据，保证 SSL 连接的数据保密性和完整性。

5.2.1 SSL 握手层协议

握手层协议实际上包括了三个子协议：加密参数变更协议、报警协议、握手协议。通过握手层的协议协商一个完整的会话：会话标识，双方证书，压缩方法，加密参数，主密钥，恢复标志。记录层协议利用握手层协议协商的结果进行下一步的计算从而保护应用数据。

5.2.1.1 报警协议(Alert Protocol)

当通信过程中出错或发生异常情况时就要用到报警协议，给出警告或终止连接。根据警告内容的严重性级别产生不同的报警信息。报警信息包括警告内容和严重性级别。如果级别是 Fatal（致命错），则会导致当前的连接立即中断，并清除包括会话标识、密钥等在内的所有状态参数。在这种情况下，与该会话相连的其他连接可以继续进行，但不能再重用该会话建立新的连接。同其他的消息一样，报警信息也要经过压缩和加密后进行传输。

报警消息的类型可分为关闭报警（Closure Alert）和错误报警（Error Alert）。

关闭报警信息所包含的警告消息只有一个，即 close_notify 消息。该消息通知对方发送者不再在该连接上发送任何数据，从而防止出现半关闭连接，给攻击者造成机会。

错误报警包含多个警告消息，其中错误级别为 Fatal 的有：unexpected_message、bad_record_mac、decompression_failure、handshake_failure、illegal_parameter，其他错误级别不为 Fatal 的警告消息都是与数字证书有关的错误，如：no_certificate、bad_certificate、unsupported_certificate、certificate_revoked、certificate_expired、certificate_unknown 等。

消息的具体数据结构定义如下：

```
enum { warning(1), fatal(2), (255) } AlertLevel;
enum {
        close_notify(0),
```

```
        unexpected_message(10),
        bad_record_mac(20),
        decryption_failed(21),
        record_overflow(22),
        decompression_failure(30),
        handshake_failure(40),
        bad_certificate(42),
        unsupported_certificate(43),
        certificate_revoked(44),
        certificate_expired(45),
        certificate_unknown(46),
        illegal_parameter(47),
        unknown_ca(48),
        access_denied(49),
        decode_error(50),
        decrypt_error(51),
        export_restriction(60),
        protocol_version(70),
        insufficient_security(71),
        internal_error(80),
        user_canceled(90),
        no_renegotiation(100),
        (255)
    } AlertDescription;
struct {
    AlertLevel level;
    AlertDescription description;
    } Alert;
```

5.2.1.2 修改加密参数协议（Change Cipher Spec Protocol）

修改加密参数协议用来标识信号的转换，它只包含一条一个字节的信息，其值为 1。修改密码参数消息由客户方或服务器方发出，用以通知对方随后的记录将受到刚协商的密码参数和密钥的保护。该消息发出后，发送端立即由待写状态转为当前写状态；接收端收到该消息后，立即由待读状态转为当前读状态。

其说明如下：

```
struct {
        enum { change_cipher_spec(1), (255) } type;
        } ChangeCipherSpec;
```

5.2.1.3 握手协议

一个 SSL 传输过程需要先握手。握手协议主要功能是协商产生记录层协议需要的加密参数，当客户机和服务器开始通信时，双方要完成协商协议版本、选择加密算法、指定彼此的认证方式、利用公钥加密技术产生共享密钥等工作。

过程描述如下：

1、 客户端向 Sever 端发送客户端 SSL 版本号、加密算法设置、随机产生的数据和其他服务器需要用于跟客户端通讯的数据。

2、 服务器向客户端发送服务器的 SSL 版本号、加密算法设置、随机产生的数据和其他客户端需要用于跟服务器进行通讯的数据。另外，服务器还要发送自己的证书，如果客户端正在请求需要认证的信息，那么服务器同时也要请求获得客户端的证书。

3、 客户端用服务器发送的信息验证服务器身份。如果认证不成功，用户就将得到一个警告，然后加密数据连接将无法建立。如果成功，则继续下一步。

4、 用户用握手过程至今产生的所有数据，创建连接所用的 Premastersecret，用服务器的公钥加密（在第二步传送的服务器证书中得到），传送给服务器。

5、 如果服务器也请求客户端验证，那么客户端将对另外一份不同于上次用于加密连接使用的数据进行签名。在这种情况下，客户端会把这次产生的加密数据和自己的证书同时传送给服务器用来产生 PremasterSecret。

6、 如果服务器也请求客户端验证，服务器将试图验证客户端身份。如果客户端不能获得认证，连接将被中止。如果被成功认证，服务器用自己的私钥加密 PremasterSecret，然后执行一系列步骤产生 MasterSecret。

7、 服务器和客户端同时产生 SessionKey，之后的所有数据传输都用对称密钥算法来交流数据。

8、 客户端向服务器发送信息说明以后的所有信息都将用 SessionKey 加密。至此，它会传送一个单独的信息表示客户端的握手部分已经宣告结束。

9、 服务器也向客户端发送信息说明以后的所有信息都将用 SessionKey 加密。至此，它会传送一个单独的信息表示服务器端的握手部分已经宣告结束。

10、 SSL 握手过程就成功结束，一个 SSL 数据传送过程建立。客户端和服务器开始用 SessionKey 加密、解密双方交互的所有数据。

一个 SSL 传输过程大致就是这样，但是很重要的一点不要忽略：利用证书在客户端和服务器端进行的身份验证过程。

一个支持 SSL 的客户端软件通过这些步骤认证服务器的身份：

1、服务器端传送的证书中获得相关信息；

2、当天的时间是否在证书的合法期限内；

3、签发证书的机关是否客户端信任的；

4、签发证书的公钥是否符合签发者的数字签名；

5、证书中的服务器域名是否符合服务器自己真正的域名；

6、服务器被验证成功，客户继续进行握手过程。

一个支持 SSL 的服务器通过下列步骤认证客户端的身份：

1、客户端传送的证书中获得相关信息；

2、用户的公钥是否符合用户的数字签名；

3、当天的时间是否在证书的合法期限内；

4、签发证书的机关是否服务器信任的；

5、签发证书的机关是否客户端信任的；

6、用户的证书是否被列在服务器的 LDAP 用户信息中；

得到验证的用户是否仍然有权限访问请求的服务器资源。

具体的消息交互过程如下图所示：

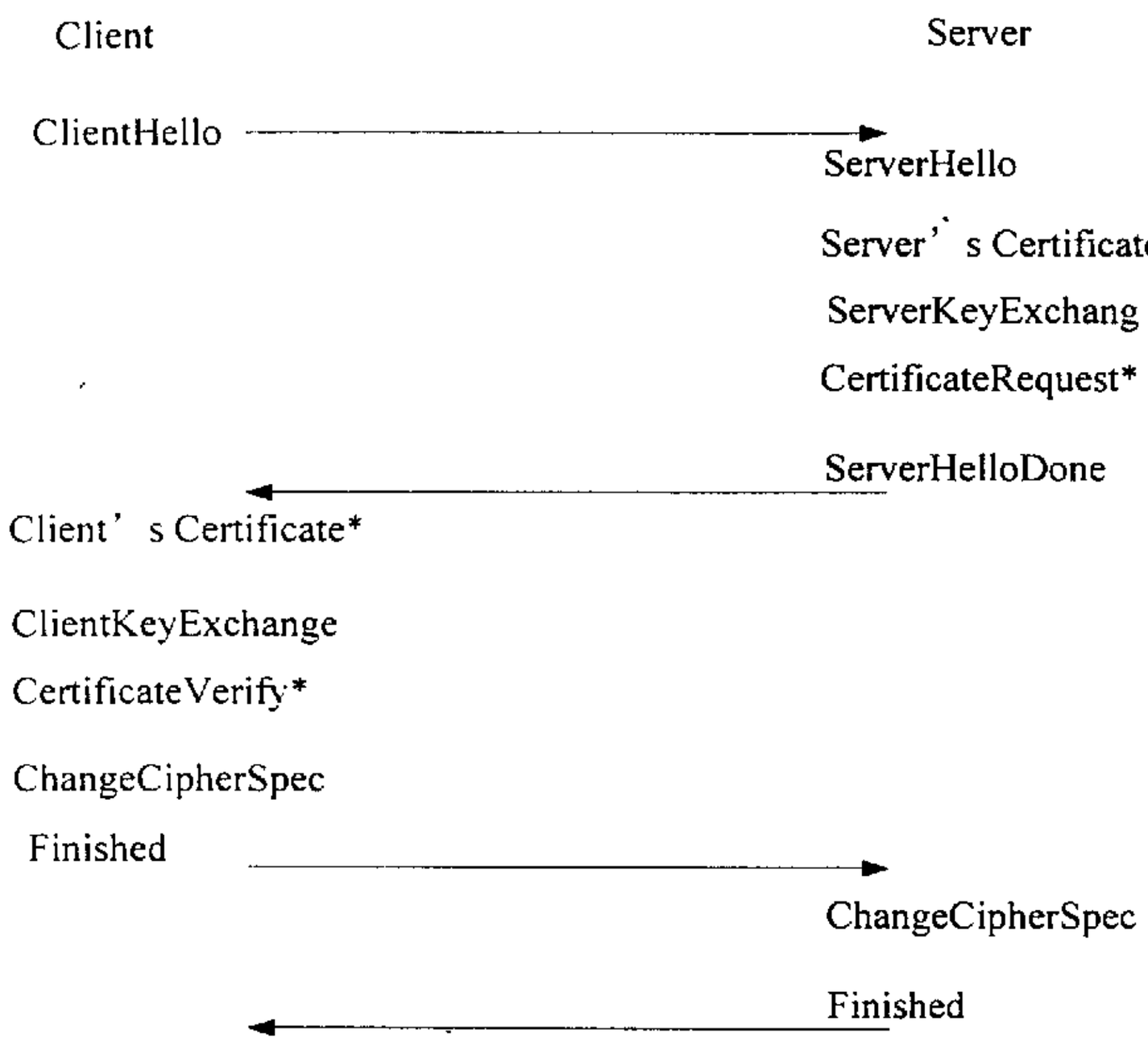


图 6-2 SSL 握手协议消息交互过程

具体的消息交换过程描述如下：

1、Client 向 Server 发送一个 ClientHello 信息，其中包括一个随机数（random1）、SSL 版本号、标识本次连接的唯一标识（ID）用于记录层进行数据压缩的算法，以及一个密码套件列表。

2、Server 收到 Client 发送的 ClientHello 信息之后，发送一个 ServerHello 信息作为应答，其中包括 S 产生的另一个随机数（random2）、同意使用的 SSL 版本号、以及一个密码套件。这个密码套件是 Server 根据自己所能支持的密码套件情况，在 Client 发送的密码套件表中选择的最靠前的一个。

3、如果 Server 有自己的数字证书，则将其发送给 Client（可选）。

4、Server 向 Client 发送其密钥交换信息。

5、如果 Server 是非匿名服务器，可能会对 Client 提出进

行身份认证的请求（可选）。

6、Server 向 Client 发送 HelloDone 信息结束 ServerHello，等待 Client 的响应。

7、如果 Server 要求 Client 发送数字证书并且 Client 持有数字证书，则将它发送给 Server；如果 Client 没有数字证书，则发送 No_certificate 告警。（可选）

8、Client 发送其密钥交换信息。

9、Client 发送一个 ChangeCipherSpec 信息，通知 Server 以下通信将使用协商好的新的密码套件及压缩算法。

10、Client 向 Server 发送一个 Finish 信息，表示与 Server 的握手完成。

11、Server 向 Client 发送 ChangeCipherSpec 信息。

12、Server 向 Client 发送 Finish 信息，结束握手过程。

5.2.2 SSL 记录层协议（Record Protocol）

记录层协议位于传送层之上，握手层协议及其他应用层之下，主要功能是完成上层协议数据的装拆和收发：在发送端，记录层将数据分块，压缩（可选），附加消息认证码 MAC，加密，并附上具有唯一性的序列号后发送给对方；在接收端的记录层中则将数据解密，验证 MAC，解压；然后将数据还原成原始数据，再交由上层处理。这个过程如下图所示：

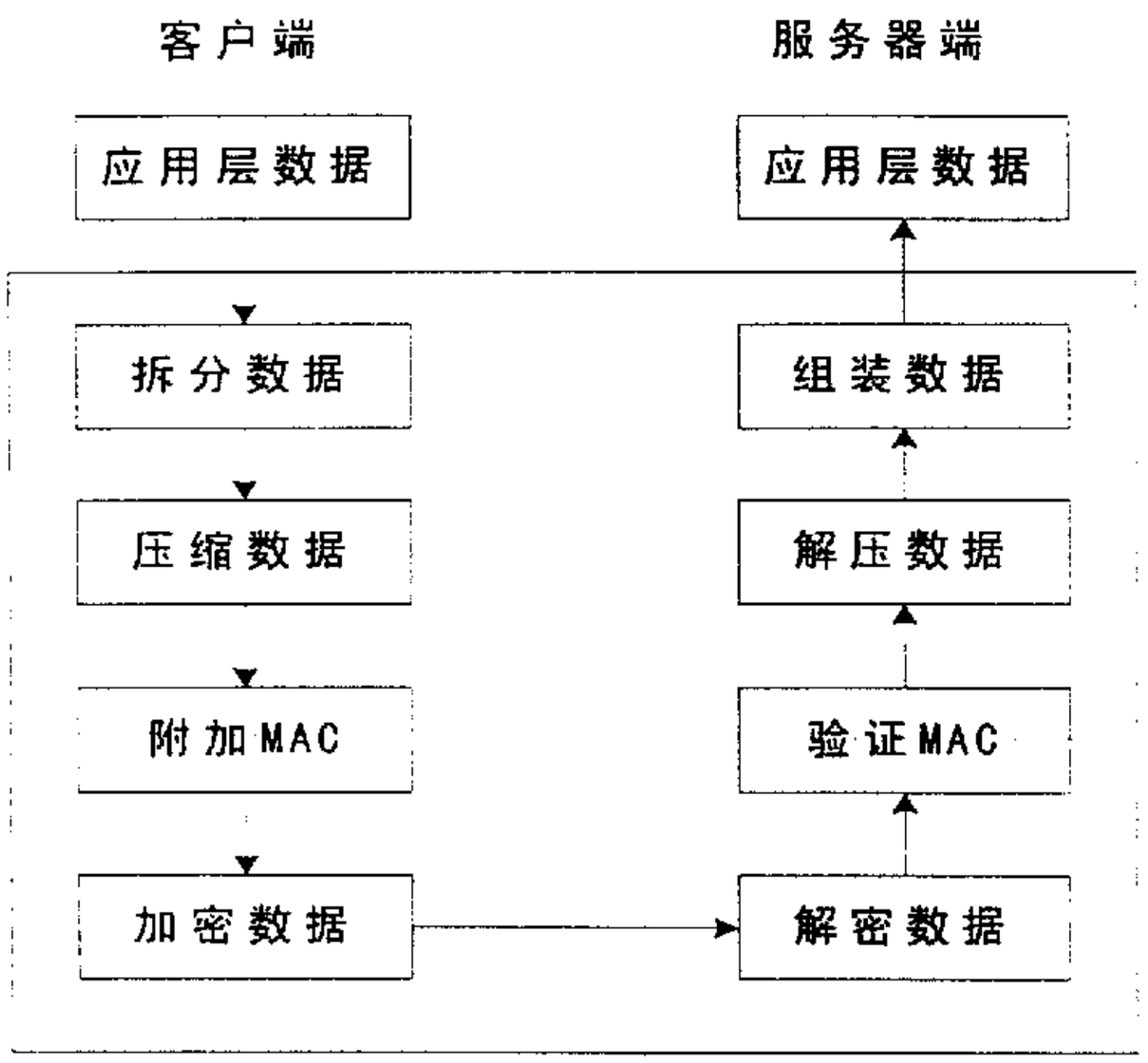


图 6-3 记录层协议处理流程

在 SSL 协议中，所有的传输数据都被封装在记录中。记录是由记录头和长度不为 0 的记录数据组成的。所有的 SSL 通信包括握手消息、安全空白记录和应用数据都使用 SSL 记录层。SSL 记录协议包括了记录头和记录数据格式的规定。

5.2.2.1 SSL 记录头格式

SSL 的记录头可以是两个或者三个字节长的编码。SSL 记录头所包含的信息有：记录头长度、记录数据的长度、记录数据中是否有粘贴数据。其中粘贴数据是在使用块加密算法时，填充实际数据，使其长度恰好是块的整数倍。最高位为 1 时，不含有粘贴数据，记录头的长度为两个字节，记录数据的最大长度为 32767 个字节；最高位为 0 时，含有粘贴数据，记录头的长度为三个字节，记录数据的最大长度为 16383 个字节。

详细的数据结构如下所示：

```
struct {
        uint8 major, minor;
    } ProtocolVersion; /*协议版本号*/
```

```
enum {
    change_cipher_spec(20), alert(21),  handshake(22),
    application_data(23), (255)
} ContentType; /*内容类型*/
struct {
    ContentType type;
    ProtocolVersion version;
    uint16 length;
    opaque fragment[TLSPlaintext.length];
} TLSPlaintext; /*SSL 明文，长度不超过 2^14 */
```

当数据头长度是三字节，次高位有特殊含义：次高位为 1 时，表示所传输的记录是普通的数据记录；次高位是 0 时，表示所传输的记录是安全空白记录（被保留将来用于协议的扩展）。

记录头中数据长度编码不包括数据头所占用的字节长度。记录头长度是两个字节的记录长度的计算公式：

记录长度＝（（byte[0]&0x7f）<<8））| byte[1]

其中 byte[0]、byte[1]分别表示传输的第一个、第二个字节。记录头长度为三个字节的记录长度的计算公式：

记录长度＝（（byte[0]&0x3f）<<8））| byte[1]

byte[0]、byte[1]的含义同上。判断是否是安全空白记录的计算公式：

(byte[0]&0x40)!=0

粘贴数据的长度为传输的第三个字节。

5.2.2.2 SSL 记录数据的格式

SSL 的记录数据包含三个部分：MAC 数据、实际数据以及粘贴数据。

MAC 数据用于数据完整性检查。计算 MAC 所用的散列函数由握手协议中的 CIPHER－CHOICE 消息确定。若使用 MD2 和 MD5 算法，则 MAC 数据长度是 16 个字节。MAC 的计算公式：

MAC 数据＝HASH【密钥，实际数据，粘贴数据，序号】。

当会话的客户端发送数据时，密钥是客户的写密钥（服务器用读密钥来验证 MAC 数据）；当会话的客户端接收数据时，密钥是客户的读密钥（服务器用写密钥来产生 MAC 数据）。序号是一个可以被发送和接收双方递增的计数器。每个通信方向都会建立

一对计数器，分别被发送者和接收者拥有。计数器有 32 位，计数值循环使用，每发送一个记录计数值递增一次，序号的初始值为 0。

在记录层中，所有数据都要用当前会话状态所定义的压缩算法压缩。压缩数据的格式和传输密文的格式如下所示：

```
struct {
        ContentType type;
        ProtocolVersion version;
        uint16 length;
      op&que    fragment[SSLCompressed.length];    /*   由
       SSLPlaintext.fragment 压缩而来*/
       }SSLCompressed;    /*SSL 的压缩格式*/
struct{
       ContentType type; /*同 SSLCompressed.type*/
       ProtocolVersion versio;/*同 SSLCompressed.version*/
       Uint16 length; /*不大于 SSLCiphertext.fragment*/
          select (CipherSpec.cipher_type) {
              case stream:GenericStreamCipher;
              case block: GenericBlockCipher;
            } fragment; /*SSLCompressed fragment 的加密格
                        式*/
      } SSLCiphertext; /*SSL 密文格式*/
stream-ciphered struct {
       opaque content[TLSCompressed.length];
       opaque MAC[CipherSpec.hash_size];
         } GenericStreamCipher;
block-ciphered struct {
       opaque content[TLSCompressed.length];
       opaque MAC[CipherSpec.hash_size];
       uint8 padding[GenericBlockCipher.padding_length];
       uint8 padding_length;
         } GenericBlockCipher
```

5.2.2.3 记录的压缩与解压缩

所有的记录都要使用当前会话状态指定的压缩算法进行压缩，这种压缩应该是无损压缩。在初始状态下，压缩算法缺省为

空。压缩算法将上面的 TLSPlaintext 数据结构转换为 TLSCompressed 结构。TLSCompressed 结构定义如下：

```
struct {
        ContentType type;       /* same as TLSPlaintext.type */
        ProtocolVersion version;
        uint16 length;
        opaque fragment[TLSCompressed.length];
        } TLSCompressed;
```

5.2.2.4 MAC 计算和加密处理

MAC 计算和加密是记录协议处理过程的核心，通过加密和 MAC 计算，达到保护数据的目的。其处理过程是先进行 MAC 计算，得到的结果放在压缩数据后面，与原来的数据一起进行加密处理。根据加密算法的不同，分为流加密和分组加密两种不同的处理流程和数据结构。流密码可以直接进行加密；分组密码要求明文长度为分组长度的整数倍，因此需要先进行填充，然后加密。填充原则是不论原来的数据长度是否为分组长度的整数倍，都在最后增加一个字节用来记录填充数据的长度，然后在原来的数据之后，长度字节之前，添加一定数量的填充，保证得到的数据为分组长度的整数倍。在 MAC 的计算结果中包括一个序列号，从而防止数据的丢失、增加、重复等错误的发生。

主要的数据结构定义如下：

```
struct {
        ContentType type;
        ProtocolVersion version;
        uint16 length;
        select (CipherSpec.cipher_type) {
                case stream:    GenericStreamCipher;
                case block:    GenericBlockCipher;
                 } fragment;
        } TLSCiphertext;
stream-ciphered struct {
        opaque content[TLSCompressed.length];
        opaque MAC[CipherSpec.hash_size];
          } GenericStreamCipher;
block-ciphered struct {
```

```
opaque content[TLSCompressed.length];
opaque MAC[CipherSpec.hash_size];
uint8 padding[GenericBlockCipher.padding_length];
uint8 padding_length;
} GenericBlockCipher
```

5.2.2.5 密钥的计算

记录层要求使用双方协商一致的算法通过握手协议的密码参数计算密钥、初始向量、MAC 密钥等。在 SSL/TLS 中规定了标准的接口函数和相关的数据结构来计算密钥。

相关接口函数定义如下：

```
key_block = PRF(SecurityParameters.master_secret,
                                    "key expansion",
                SecurityParameters.server_random
                +SecurityParameters.client_random);
client_write_MAC_secret[SecurityParameters.hash_size]
server_write_MAC_secret[SecurityParameters.hash_size]
client_write_key[SecurityParameters.key_material_length]
server_write_key[SecurityParameters.key_material_length]
client_write_IV[SecurityParameters.IV_size]
server_write_IV[SecurityParameters.IV_size]
final_client_write_key =
    PRF(SecurityParameters.client_write_key,
                                    "client write key",
                SecurityParameters.client_random +
                SecurityParameters.server_random);
final_server_write_key =
    PRF(SecurityParameters.server_write_key,
                                    "server write key",
                SecurityParameters.client_random +
                SecurityParameters.server_random);
iv_block = PRF("", "IV block",
                SecurityParameters.client_random +
                SecurityParameters.server_random);
client_write_IV[SecurityParameters.IV_size]
server_write_IV[SecurityParameters.IV_size]
```

5.3 SSL 协议的实现

这里我们介绍利用开放的源代码 OPENSSL 提供的应用程序接口来实现 SSL 协议的方法。

5.3.1 握手协议的实现

5.3.1.1 客户端 SSL 握手协商过程的实现

1、初始化状态

设置 SSL 的连接类型为 SSL_ST_CONNECT，分配一个临时缓冲区，调用函数

```
static int ssl3_client_hello(SSL *s);
```

发送 ClientHello 消息报文，进入等待读取 ServerHello 消息报文的状态。

2、读取 ServerHello 消息报文状态下

调用函数

```
static int ssl3_get_server_hello(SSL *s);
```

读取和处理 ServerHello 消息报文，得到服务器端的会话标识符、随机数、服务器指定的协议版本号、加密算法、压缩算法、摘要算法。根据会话标识符是否与 ClientHello 中指定的相同决定是否会话重用。如果重用则转入读取 Finished 消息报文状态，否则转入读取服务器证书消息状态。

3、读取服务器证书消息报文状态下

调用函数

```
static int ssl3_get_server_certificate(SSL *s);
```

读取和处理服务器证书消息报文，进入读取密钥交换消息报文状态。

4、读取密钥交换消息报文状态（ServerKeyExchange）下

调用函数

```
static int ssl3_get_key_exchange(SSL *s);
```

读取和处理服务器 ServerKeyExchange 消息报文，进入读取证书请求消息报文状态。

5、读取证书请求消息报文状态下

调用函数

static int ssl3_get_certificate_request(SSL *s);

读取和处理服务器证书请求消息报文，进入读取ServerHelloDone消息报文状态。

6、读取ServerHelloDone消息报文状态下

调用函数

static int ssl3_get_server_done(SSL *s);

读取和处理服务器ServerHelloDone消息报文。若要对客户端进行身份认证，则进入发送客户端证书消息报文状态，否则进入发送更换加密算法规范消息报文状态。这里按照要求客户端身份验证进行。

7、发送客户端证书消息报文状态下

调用函数

static int ssl3_send_client_certificate(SSL *s);

组装和发送客户端证书消息报文，进入发送密钥交换消息报文状态。

8、发送密钥交换消息报文状态下

调用函数

static int ssl3_send_client_key_exchange(SSL *s);

组装和发送客户端ClientKeyExchange消息报文，发送客户端产生的预主密钥给服务器端（用服务器端公钥加密，用于生成共享密钥）。若客户端证书有签名功能则进入发送证书验证消息报文状态，否则进入发送更换加密算法规范消息报文状态。现在按照客户端证书有签名功能进行。

9、发送证书验证消息报文状态下

调用函数

static int ssl3_send_client_verify(SSL *s);

组装和发送客户端证书验证消息报文，进入发送更换加密算法规范消息报文状态。

10、发送更换加密算法规范消息报文状态下

调用函数

static int ssl3_send_change_cipher_spec(SSL *s, int a, int b);

组装和发送更换加密算法规范消息报文，指定会话协商加密算法、摘要算法、密钥生效。进入发送完成消息报文状态。调用函数

int ssl3_send_finished(SSL *s, int a, int b, const char *sender, int slen);

组装和发送完成消息报文，进入刷新缓存区状态。

11、刷新缓存区状态下

调用函数

int BIO_ctrl(b, BIO_CTRL_FLUSH, 0, NULL);

清楚或者刷新缓冲区。若会话重用则进入握手成功状态，否则进入读取完成消息报文状态。

12、完成消息报文状态下

调用函数

int ssl3_get_finished(SSL *s);

读取和处理完成消息报文。根据会话标识符是否与 ClientHello 中指定的相同决定是否会话重用。如果重用则转入发送更换加密算法规范消息报文状态，否则转入握手成功状态。现在按不重用进行。

13、握手成功状态下

释放临时缓冲区等。

5.3.1.2 服务器端 SSL 握手协商过程的实现

1、初始化状态

设置 SSL 的连接类型为 SSL_ST_ACCEPT，分配临时缓冲区，进入读取 ClientHello 消息报文的状态。

2、读取 ClientHello 消息报文状态下

调用函数

static int ssl3_get_client_hello(SSL *s);

读取和处理 ClientHello 消息报文，进入发送 ServerHello 消息状态。

3、发送 ServerHello 消息状态下

调用函数

static int ssl3_send_server_hello(SSL *s);

发送 ServerHello 消息报文，根据会话标识符决定是否会话重用。重用则进入发送更换加密算法规范消息报文状态，否则进入发送证书消息状态。

4、发送证书消息状态下

调用函数

int ssl3_send_server_certificate(SSL *s);

发送证书消息报文，进入发送密钥交换消息报文状态。

5、发送密钥交换消息报文状态下

调用函数

static int ssl3_send_server_key_exchange(SSL *s);

发送密钥交换消息报文，若要求对客户端进行身份验证，则进入发送证书请求消息报文状态。

6、发送证书请求消息报文状态下

调用函数

static int ssl3_send_certificate_request(SSL *s);

发送证书请求消息报文;进入发送 ServerHelloDone 消息报文状态。

7、发送 ServerHelloDone 消息报文状态下

调用函数

static int ssl3_send_server_done(SSL *s);

组装和发送 ServerHelloDone 消息报文，进入刷新缓存区状态。

8、刷新缓存区状态下

调用函数

int BIO_ctrl(b, BIO_CTRL_FLUSH,0,NULL);

刷新缓存区（输出），进入读取证书消息报文状态。

9、读取证书消息报文状态下

调用函数

static int ssl3_get_client_certificate(SSL *s);

读取和处理客户端证书消息报文,进入读取密钥交换消息报文状态。

10、读取密钥交换消息报文状态下

调用函数

static int ssl3_get_client_key_exchange(SSL *s, int a, int b);

读取和处理密钥交换（ClientKeyExchange）消息报文。若客户端证书具有签名功能，则进入读取证书验证消息报文状态，否则进入读取完成消息报文状态。现按照客户端证书具有签名功能进行。

11、读取证书验证（CertificteVerify）消息报文状态下

调用函数

static int ssl3_get_cert_verify(SSL *s);

读取和处理证书验证消息报文，进入读取完成消息报文状态

12、读取完成（Finished）消息报文状态下

调用函数

int ssl3_get_finished(SSL *s, int a, int b);

读取和处理完成消息报文。根据会话标识符是否决定是否会话重用。如果重用则转入握手协商成功状态，否则转入服务器端发送更换加密算法规范消息报文状态。现在按不重用进行。

13、 发送更换加密算法规范（ChangeCipherSpec）消息报文状态下

调用函数

int ssl3_send_change_cipher_spec(SSL *s, int a, int b);

发送更换加密算法规范消息报文。进入发送完成消息报文状态。

14、发送完成（Finished）消息报文状态下

调用函数

int ssl3_send_finished (SSL *s, int a, int b, const char *sender, int slen);

发送完成消息报文。进入服务器刷新输出缓存区状态。

15、刷新输出缓存区状态下

调用

BIO_flush(BIO *b);

刷新缓冲区。若会话重用，进入读取完成消息报文状态；若会话不重用则进入握手成功状态。现按照会话不重用进行。

16、握手成功状态下

释放临时缓冲区等。

5.3.2 记录层协议的实现

5.3.2.1 报文拆分的实现

记录层协议将上层传来的报文信息拆分为小于或等于2^{14}字节的明文结构，客户消息报文的边界在记录层不予保留（即多个同一内容类型的客户消息可以存放在一个单一的明文结构中）。

发送数据的时候，在函数

int ssl3_write_bytes(SSL *s, int type, const void *buf_, int len)

中比较将要发送的报文长度 len 和 MAX_PLAIN_LENGTH(2^{14}=16384)；如果大于2^{14}，则将报文的

前 2^{14} 个字节，通过调用函数

int do_ssl3_write(SSL *s, int type, const unsigned char* buf, unsigned int len)

发送出去；然后继续发送剩下的报文，直到所有报文的长度都小于等于 2^{14} 为止。

5.3.2.2 记录压缩与解压缩的实现

记录通过当前会话状态所指定的压缩算法进行压缩和解压处理。这个压缩算法是在握手协商阶段确定的，而且通信的双方总是存在一个激活的压缩算法。若是指定的压缩算法是空的，那么只需要将数据简单地拷贝到指定的结构中。压缩算法将记录明文报文转换为压缩结构的报文。要更改加密算法时，压缩函数就将它们的状态信息删除。压缩算法做到不丢失信息，增加内容的长度不能大于 1024 字节。当解压一个压缩的报文结构数据段时，若解压后的报文长度超过 2^{14} 字节，则发出一个解压失败的致命报警消息。

1、进行握手协商时，收到 Client 发来的 ClientHello 消息后，从报文中指定的压缩算法列表里选择要采用的压缩算法，将其保存在 SSL 结构中的 s->s3->tmp.new_compression 和 s->session->compression_meth 中。

2、更换加密算法时，发送更换加密算法的消息报文后，调用函数

s->method->ssl3_enc->change_cipher_state();

实际上就是调用函数

int sl3_change_cipher_state(SSL *s, int which);

期间还通过调用函数

COMP_CTX* COMP_CTX_new(COMP_METHOD *meth);

创建压缩算法上下文控制结构，并初始化结构选项

s->expand->meths 和 s->compress->meth

为握手协商过程中指定的压缩算法。

3、在发送数据时，通过调用函数

static int do_compress(SSL *ssl);

进行记录数据的压缩处理。实际调用函数

int COMP_compress_block(COMP_CTX *ctx. unsigned char *out, int olen, unsigned char *in, int ilen);

其中参数 COMP_CTX *ctx 指定为 ssl->compress，进行记录数据的压缩处理。

4、在读取数据时，调用函数

static int do_uncompress(SSL *ssl);

进行记录数据的解压处理。实际调用函数

int COMP_expand_block(COMP_CTX *ctx, unsigned char *out, int olen, unsigned char *in, int ilen);

其中参数 COMP_CTX *ctx 指定为 ssl->expand，进行数据解压处理。

5.3.2.3 加密实现

1、握手协商过程中

客户端的 ClientHello 消息报文中指定了客户端支持的一系列加密算法，并发送给服务器。服务器在这一系列加密算法中选出服务器也支持的一个加密算法，这个加密算法就成为双方共同的加密算法，将其赋值给 s->s3->tmp.new_cipher，将包含这个加密算法的 ServerHello 报文发送给客户端。客户端收到这个报文后将指定的加密算法赋值给 s->s3->tmp.new_cipher。

2、更换加密算法时

在更换加密算法规范消息报文状态，将 s->s3->tmp.new_cipher 赋值给 s->session->cipher，调用

s->method->ssl3_enc->setup_key_block(s);

建立共享密钥块，其实是调用函数

int ssl3_setup_key_block(SSL *s);

3、发送更换加密规范消息报文后

要使协商的加密算法生效是通过调用函数

s->method->ssl3_enc->change_cipher_state();

其实是调用函数

int ssl3_ change_cipher_state (SSL *s, int which);

实现。再通过调用

viod EVP_CipherInit(EVP_CIPHER_CTX *ctx, const EVP_CIPHER data, unsigned char *key, unsigned char *iv, int enc);

将协商好的加密算法赋给 s->enc_write_ctx 中的结构

EVP_CIPHER

4、发送数据时实现加密

加密过程在函数

static int do_ssl3_write(SSL *s, int type, const unsigned chat *buf, unsigned int len);

中，调用函数

s->method->ssl3_enc->enc->enc(s,1)

实现对数据的加密，实际上是调用

int ssl3_enc(SSL *s, int send);

具体过程如下：

```
{
 EVP_CIPHER_CTX ds; /*定义变量*/
   ds=s->enc_write_ctx;
   rec=&(s->s3->wrec);
  EVP_Cipher(ds, rec->data, rec->input,l);
  /*进行加密，其中 ds 为加密算法，rec->input 是输入
  需要加密的数据，rec->data 是加密后的结果，l 是数
  据长度*/
}
```

5、解密过程实现

解密过程也定义在

static int do_ssl3_write(SSL *s, int type, const unsigned chat *buf, unsigned int len);

调用

s->method->ssl3_enc->enc(s,0)

实现对数据的解密，实际上调用

int ssl3_enc(SSL *s, int send);

具体过程如下：

```
{
EVP_CIPHER_CTX *ds; /*定义变量*/
   ds=s->enc_write_ctx;
   rec=&(s->s3->rrec);
  EVP_Cipher(ds, rec->data, rec->input,l);
  /*进行加密，其中 ds 为解密算法，rec->input 是输入
  需要解密的数据，rec->data 是解密后的结果，l 是数
  据长度*/
```

}

5.3.2.4 负载保护的实现

1、握手协商过程中
客户端和服务器端交换 Hello 报文消息时，将协商好的密码组合算法赋值给 s->s3->tmp.new_cipher。

2、更换加密算法时
将更换加密算法时赋值给赋值给 s->session->cipher，调用
s->method->ssl3_enc->setup_key_block(s);
建立共享密钥块，其实是调用函数
int ssl3_setup_key_block(SSL *s);

3、发送更换加密规范消息报文后
要使协商的加密算法生效是通过调用函数
s->method->ssl3_enc->change_cipher_state();
其实是调用函数
int ssl3_ change_cipher_state (SSL *s, int which);
实现。
指定 s->read_hash 或 s->write_hash 的值为 s->s3->tmp.new_hash（根据 which 的值而定）。

4、发送数据时
在函数
static int do_ssl3_write(SSL *s, int type, const unsigned chat *buf, unsigned int len);
中，调用函数
s->method->ssl3_enc->mac(s, &(p[wr->length]),l);
实现消息摘要的计算，实际上是调用
int ssl3_mac(SSL *ssl,unsigned char *md, nt send);
int send 为 1。

5、接收数据时
计算读取记录数据的信息摘要是通过在
static int ssl3_get_record(SSL *s);
中调用
s->method->ssl3_enc->mac(s,md,0);
实际上是调用
int ssl3_mac(SSL *ssl,unsigned char *md, int send);

int send 为 0。

实现计算消息摘要的具体过程如下：

```
if(send)
{
   rec=&(ssl->s3->wrec);
   mac_sec=&(ssl->s3->write_mac_secret[0]);
   seq=&(ssl->s3->write_sequencep[0]);
   hash=ssl->write_hash;
}
else
{
   rec=&(ssl->s3->rrec);
   mac_sec=&(ssl->s3->read_mac_secret[0]);
   seq=&(ssl->s3->read_sequencep[0]);
   hash=ssl->read_hash;
}
```

总　结

在信息安全技术领域里，公开密钥加密技术近年来发展很快，在这项技术的基础之上形成和发展起来的“公开密钥基础设施”PKI，很好地适应了互联网的特点，为互联网以及相类似网络的应用提供了全面的服务，如网络应用中的认证、加解密的密钥观里，数据的完整性，不可否认性保证等。可以说，今天互联网的安全应用，已经离不开 PKI 技术的支持了。

密码技术是在 PKI 体系中实现安全通信的核心和基础，密码技术的成熟和发展对 PKI 的应用和发展有着重要的决定性意义。

我国作为一个潜在的网络大过，发展自己的 PKI 技术是很有必要的。目前已有一些省市和系统推出了 PKI 实验性系统，但这些系统有不少是出国外公司搭建的。然而，由于国外对中国的出口限制，出口到中国的数据安全产品保密强度是很低的，由于密码分析技术和计算机运算速度的不断提高，这些产品已无法保证所处理和传输的信息的安全性。不仅如此，出于政治和技术上的原因，也必须开发自主版权的高安全性的数据安全产品。从战略意义出发，我国的安全认证平台必须自主开发，研究和开发我们自己实用 PKI 技术显得极为重要，这一课题的研究有着重要的现实意义。

作者对 PKI 系统中常用的密码技术进行了研究，用标准 C 语言实现了 DES、MD5 和 RSA 这三种最通用的加解密算法，并综合利用这三种算法实现了 PKI 中的数据加密解密、数字签名、数字信封、信息摘要等常用技术。作者还实现了 base64 的编解码，

对 SSL 协议进行了详细的分析，并利用 OPENSSL 开放源代码实现了 SSL 协议。这些工作都是 PKI 安全性的重要保障。

但是，安全都是相对的，而且网络安全往往不是单一方面安全就可以完全解决问题的。网络中存在各种各样的服务，某一协议或系统的安全漏洞被攻击者利用，可能导致该系统上运行的其他服务也受到威胁，如重要数据被窃取或恶意删改等。所以安全在某种意义上讲只能是相对的，人们需要做的是尽可能防范各种安全隐患。

致　　谢

首先，衷心感谢我的导师刘云教授。在硕士研究生的学习期间，刘老师在很多方面给予我很大的帮助。刘老师严谨的治学态度、渊博的专业知识使我在学习上受益匪浅；刘老师母亲般的关怀使我在生活中时时感受到春天般的温暖。

感谢计算机学院张振江老师和电信学院计算机通信教研室孟嗣仪、穆海冰、毕红军、周春月老师，张芸秘书对我的指导和帮助。

感谢实验室梁栋、宁红宙同学，在项目研究和论文写作期间给我很多帮助和启发，帮我答疑解惑，耐心而细致。

感谢实验室杨君、吴海旺、郭海燕、孙一峰、郝俊、丁亦志同学在三年共同的学习生活中对我的帮助，从他们身上我学到了很多东西。

感谢同宿舍的姐妹赵桂燕、雷连虹、刑新宇、石金给予我太多的快乐，太多的鼓励，她们的欢笑带给我无尽的喜悦。

最后，特别感谢我的家人，是他们给予我无微不至的关怀，克服困难的信心，他们无私的爱支持我顺利完成学业。

向所有关心、帮助、支持、鼓励过我的人表示深深的谢意！

参考文献

1. ITU-T Recommendation X.509. "Information Technology - Open Systems Interconnection- The Directory: Authentication Framework", June 1997.
2. 【PKCS#1】RSA Encryption Standard. Version 1.5, RSA Laboratories, November 1993.
3. 【PKCS#6】Extended-Certificate Syntax Standard. Version 1.5, RSA Laboratories, November 1993.
4. 【PKCS#7】 Cryptographic Message Syntax Standard. Version 1.5, RSA Laboratories, November 1993.
5. 【PKCS#8】 Private-Key Information Syntax Standard. Version 1.5, RSA Laboratories, November 1993.
6. 【PKCS#9】 Selected Attribute Types. Version 1.1, RSA Laboratories, November 1993.
7. 【PKCS#10】Certification Request Syntax Specification. RSA Laboratories, November 1993.
8. 【PKCS#11】 Cryptographic token interface standard. RSA Laboratories, November 1993.
9. 【SSL3】A. Frier, P. Karlton, P. Kocher, "The SSL 3.0 Protocol", Netscape Communications Corp, November 1996.
10. 【RFC2459】Internet X.509 Public Key Infrastructure Certificate and CRL Profile.

11. 【RFC2510】Internet X.509 Public Key Infrastructure Certificate Management Protocols.
12. 【RFC2511】Internet X.509 Certificate Request Message Format.
13. 【RFC2527】 "Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework", March, 1999
14. 【RFC2528】 "Internet X.509 Public Key Infrastructure Representation of Key Exchange Algorithm(KEA) Keys in Internet X.509 Public Key Infrastructure Certificates", March, 1999
15. 【RFC2560】 "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol-OCSP", June, 1999
16. 【RFC3548】 "The Base16, Base32, and Base64 Data Encodings "
17. http://www.rsa.com
18. http://www.openssl.org
19. http://www.openssl.cn
20. 《应用密码学 协议、算法与 C 源程序》,(美)Bruce Schneier, 机械工业出版社
21. 《公钥基础设施 PKI 与认证机构 CA》, 关振胜, 电子工业出版社
22. 《公开密钥基础设施——概念、标准和实施》,Carlisle Adams Steve Lloyd, 人民邮电出版社
23. 《PKI 技术研究及其客户端实现》, 王念, 西南交通大学

24. 《SSL 安全代理服务器的设计与实现》，郑广春，西南交通大学

25. 《基于 SSL 的数据安全传输系统的研究与实现》，杨亚平，北京航空航天大学

26. 《基于 PKI 的 CA 安全认证服务的研究与实现》，唐国岚，电子科技大学

27. 《公开密钥基础设施(PKI)及电子证书系统的设计与实现》，成孝禹，中国科学院软件研究所

28. 《安全 WWW 服务器的设计与实现及 PKI 体系的设计》，熊雁凌，中国科学院软件研究所

29. 《基于 ECC 和 SSL 的网络安全通信系统研究》，郝家胜，哈尔滨工业大学

30. 《企业级 PKI 设计与研究》，夏锦春，西南交通大学

31. 《PKI 系统设计与实现》，刘明桥，武汉理工大学