华中科技大学

硕士学位论文

OpenSSL分析与测试

姓名：雷长青

申请学位级别：硕士

专业：软件工程

指导教师：覃中平

20061023

摘 要

随着 Internet 的迅速发展和广泛应用，网络与信息安全的重要性和紧迫性日益突出。Netscape 公司提出了安全套接层协议 SSL （Secure Socket Layer），该协议基于公开密钥技术，可保证两个实体间通信的保密性和可靠性，是目前 Internet 上保密通信的工业标准。

OpenSSL 是开源产品，实现了 SSL 协议并支持多种加密算法，其官方的文档是关于 OpenSSL 的 API 使用说明和指令的用法。无从了解作者的设计思想与开发思路。更无法了解其实现 SSL 的详细过程。

我们从 OpeSSL 源代码客户端出发，跟踪 SSL 客户端与服务器连接，分析了作者的关于 OpenSSL 的开发设计思路及软件实现部分技巧；分析了 OpenSSL 利用面向对象的设计方法中的接口与多态来支持多种加密算法；分析了 OpenSSL 初始化时加载加密算法的过程；分析了 SSL 客户端筛选加密套件过程；分析了 OpenSSL 实现 SSL 握手的消息的格式与 SSL 协议中存在差异部分，并指出了 SSL 协议的 Client hello, Server hello, Client Key Exchange, Server certificate 等消息存在不足之处。

利用嵌入 OpenSSL 库的第三方收发邮件软件 iScribe，与 Gmail 邮件服务器连接测试，测试结果验证了本文的分析过程。另外，用 iScribe 比对 IE 连接 Gmail 邮件服务器，测试结果表明 IE 缺省的加密套件的加密算法及强度的安全性不高。

通过 OpenSSL 分析与测试，了解原作者当初的设计思想与开发思路，对今后深入地分析与研究 OpenSSL 和 SSL 协议及详细实现提供了参考基础。对我国信息与安全的设计与实现有借鉴意义。

关键词：安全套接层　开源代码　信息安全　密码学

Abstract

With the rapid development and the extensive application of Internet, the importance and the urgency of the network and information security have become increasingly prominent. Netscape Company has put forward the Secure Socket Layer protocol SSL （Secure Socket Layer） based on public-key technology, therefore it can guarantee the confidentiality and the reliability of the communications between two entities. The SSL protocol is the Industry Standard for secure communications via internet.

The OpenSSL is an open source product implementing SSL protocol. Its documents are published on the official website of OpenSSL. The documents of OpenSSL are limited in command line tool and API Specifications,

This paper is based on the source codes of OpenSSL, tracks communication between client and server, The author's ideas about the development of software and realization of the technology and skills shall be sorted out. To analyze how to use inteface and polymorphism of object-oriented to dynamically load multi encryption algorithm. To analyze how to select process of cipher suites. To analyze handshake message in OpenSSL implemented is different with SSL protocol. And point out deficiency of SSL message, like Client hello, Server hello, Client Key Exchange, Server certificate etc. but but the OpenSSL covers this gap during its implementation.

Using third-party mail software with OpenSSL library embedded, Connect with the Gmail server to testing, Test results showed that the analysis process above. By the way, compare to IE connection with Gmail server, Test results show that default encryption algorithm and strength of IE is not high.

Analysis and testing through OpenSSL, it is a reference for studying OpenSSL and SSL protocol and implementation. It is reference for our Design and Implementation of SSL in the future.

Key Words: SSL/TLS OpenSource Information Secure Cryptography

独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：潘春青

日期：2006年 10月 26日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权华中科技大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本论文属于 保密□，　在_____年解密后适用本授权书。

不保密☑。

（请在以上方框内打“√”）

学位论文作者签名：潘春青　　指导教师签名：

日期：2006年10月26日　　日期：06年10月26日

1 绪论

1.1 研究背景

信息安全是自古以来就存在的概念，以前为了保证传递书信的保密性，使用腊封或其它方式将书信封装在信封内；还有使用暗号口令确认接受信息的人的身份等等方法。信息安全技术是跟信息的载体形式和传送媒介密切相关的，信息载体的变化和信息传送媒介的变化必然会导致信息安全技术的变化发展。

在过去的二十多年中，信息技术取得令人惊异的发展，越来越多的有价值的信息和资料以数字信息存放在计算机等数字信息存储设备中。与此同时，信息共享技术也获得了巨大的突破，以 Internet 的发展为代表，短短的时间内，从美国军方的一个专用网络发展到联系着全世界千千万万人的庞大信息网络。这些客观的变化导致对信息安全的要求发生了重大的变化。

随着信息数字化以及计算机应用的发展，对存储在计算机中的文件和其它数字信息的保护需求成为了一种必然，尤其对一个能够通过公共网络进入的共享系统来说，这种需求显得尤为迫切。针对这种需求目前发展起来的技术有防病毒技术和防火墙技术等等。文献将这些保护数据、阻挡非法数据访问的技术统称为计算机安全或系统安全技术。

信息安全技术的另外一个重要变化是由于网络和通信设施的产生和应用引起的。这些网络和通信设施用来在用户各种终端以及计算机之间传输数据信息，这个传输过程很容易受到非法窃听等攻击，这就需要对在网络中传输的数据采取安全的保护措施。针对这种需求发展起来的技术有 VPN、SSL 等。文献将这种类型的技术统称为网络安全技术。

随着 Internet 的迅速发展和广泛应用，网络与信息安全的重要性和紧迫性日益突出。Netscape 公司提出了安全套接层协议 SSL （Secure Socket Layer） [1]，该协议基于公开密钥术[2]，可保证两个实体间通信的保密性和可靠性[3,4]，目前是 Internet 上保密通信的工业标准。

Eric A. Young 和 Tim J. Hudson，自 1995 年开始编写后来具有巨大影响的 OpenSSL 软件包[5]，这是一个没有太多限制的开放源代码的软件包，可以利用这个软件包做很多事情。1998 年，OpenSSL 项目组接管了 OpenSSL 的开发工作，并推出了 OpenSSL 的 0.9.1 版，到目前为止，OpenSSL 的算法已经非常完善，对 SSL2.0、SSL3.0 以及 TLS1.0 都支持。OpenSSL 目前最新的版本是 0.9.8 版。

OpenSSL 采用 C 语言作为开发语言，使得 OpenSSL 具有优秀的跨平台性能，可以在不同的平台使用。OpenSSL 支持 Linux、Windows、BSD、Mac、VMS 等平台，OpenSSL 具有广泛的适用性。OpenSSL 实现 8 种对称加密算法如 AES、DES、Blowfish、CAST、IDEA、RC2、RC4、RC5，实现了 4 种非对称加密算法如 DH 算法、RSA 算法、DSA 算法和椭圆曲线算法（EC），实现了 5 种信息摘要算法如是 MD2、MD5、MDC2、SHA（SHA1）和 RIPEMD，密钥及证书管理[6,7]。

OpenSSL 的 license 是 Ssleay license 和 OpenSSL license 的结合，这两种 license 实际上都是 BSD 类型的 license，依照 license 里面的说明，OpenSSL 可以被用做各种商业、非商业的用途，但是需要相应遵守一些协定，其实这都是为了保护自由软件作者及其作品的权利。

1.2 论文主要分析工作

对 OpenSSL 从整体进行分析。包括 OpenSSL 源代码的模块结构，它支持的对称加密算法，非对称加密算法，信息摘要算法，密钥和证书管理等。

以 OpeSSL 源代码为基础，利用 VC6 开发工具，跟踪 Client/Server 范例代码执行 SSL 的完整过程，分析出作者的开发设计思路及软件实现部分技巧；使用面向过程 C 语言去实现面向对象的设计与实现；将网络及文件操作封装成 BIO 接口；将各种加密算法封装成 EVP 接口；设计与实现接口的技巧；利用宏定义来初始化数据结构体等。

通过跟踪源代码运行过程，分析了 SSL 的握手过程和应用数据加密的完整过程。SSL 握手过程的加密套件的筛选过程；客户端与服务端通讯过程；证书链及私钥文件如何加载；握手过程的部分消息等。分析后发现 SSL 的握手协议的 Client hello,

Server hello, Client Key Exchange, Server certificate 等消息有一点不足。但在 OpenSSL 代码实现中弥补了此不足。

利用 ssldump 分文分析工具软件，嵌入 SSL 库的 iscribe 邮件收发软件，IE 等，测试了 OpenSSL 源代码在执行中，其接口的调用与实现的函数之间是运用了面向对象的设计思路；验证了 OpenSSL 加密套件筛选符合其定义规则；跟踪并分析了 SSL 握手过程。

通过 iscribe 和 IE 两种不同的客户端与同一服务端通讯来比对，结果表明 IE 缺省的加密套件中的加密算法和强度的安全性并不高。

2 SSL 概述

2.1 SSL 概述

安全套接层协议（Security Socket Layer，SSL）是由网景（Netscape）公司提出的基于公钥密码体制的网络安全协议，用于在浏览器软件（例如 Internet Explorer、Netscape Navigator）和 Web 服务器之间建立一条安全通道，实现 Internet 上信息传送的保密性。它包括服务器认证、客户认证（可选）、SSL 链路上的数据完整性和 SSL 链路上数据保密性。现在一些对保密性要求较高的电子商务、电子事务等系统大多数是以 SSL 协议为基础建立的，SSL 协议已成为 Web 安全方面的工业标准。目前广泛采用的是 SSLv3。

SSL 提供的面向连接的安全性具有以下在三个基本性质[8,9]：

（1）连接是秘密的[10]。在初始握手定义会话密钥后，用对称密码（例如用 DES）加密数据。

（2）连接是可认证的[10]。实体的身份能够用公钥密码（例如 RSA、DSS 等）进行认证。

（3）连接是可靠的[10]。消息传输包括利用安全 Hash 函数产生的带密钥的 MAC（Message Authentication Code，报文鉴别码）。

尽管 SSL 协议本身在设计时考虑其安全性，但在实际应用中 SSL 安全协议并不完备，缺陷是只能保证传输过程的安全，无法知道在传输过程中是否受到窃听，黑客可破译 SSL 的加密数据，破坏和盗窃 Web 信息[11]。目前新的 SSL 协议被命名为 TLS（Transport Layer Security），安全可靠性进一步提高，但仍不能消除原有技术的基本缺陷。另外， SSL 握手协议有一个小的疏漏，即在 Finished 消息中没有对 Change Cipher Spec 消息的认证保护。这将导致“修改密码规约”失效[12]。

为了克服 SSL 协议的缺点，Master-Card 和 Visa 合作开发了 SET 协议[13]。

SSL 协议综合了多种加密算法，其设计协议时，考虑了实际执行过程的性能问题。应用系统使用了 SSL 协议比不使用 SSL 协议，不会对系统统性能造成影响[14,15]。

2.2 SSL/TLS 的设计目标

SSL 原先是为 Web 设计的。网景公司的意图是针对其所有的通信安全问题。当时面临的问题是 Web 通信，所以 SSLv2 就是为了满足 Web 通信的需要而设计的。SSLv2 必须能够与 HTTP 协议一起很好的工作。在 1994 年设计 SSLv2 的时候，人们关心的问题主要有两点。其一，如何不泄露给实施攻击的第三方条件下，将信息从客户端传到服务端。即客户端与服务器之间的保密性；其二，客户端（Web 浏览器）能够确信其传输数据的服务端是可靠的，即服务器认证[16]。

SSLv2 的流行大体保证了 SSLv2 的设计目标在 SSLv3 中保留下来。SSLv3 设计者首要工作是修正 SSLv2 中的多处安全问题。SSLv3 明显的变化是设计一种安全磋商多种加密算法的机制。结果 SSLv3 比 SSLv2 支持的算法数量更多。从而 SSLv3 与 SSLv2 完全不同。

SSL 发展历史如下图[17]：

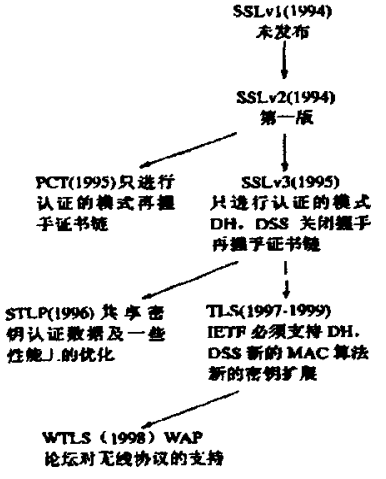

图 2-1 SSL 历史发展过程

2.3 SSL 协议结构

SSL 协议基于 TCP/IP 的客户/服务器应用程序提供了客户端和服务器的鉴别、数据完整性及信息机密性等安全措施。SSL 可运行在任何可靠的通信协议之上、并

可运行在 HTTP、FTP、TELNLET 等应用层协议之下。SSL 协议与 OSI 模型对应关系如下图[18]：

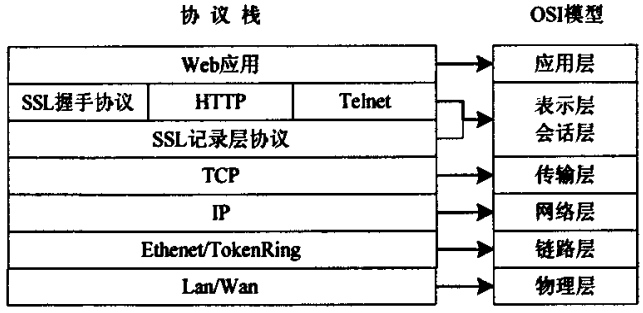

图 2-2 SSL 协议与 OSI 模型对应关系

SSL 记录层协议与 SSL 握手协议相当于 OSI 模型中的会话层与表示层。

SSL 是一种分层协议[19]。它由 SSL 记录层协议以及记录层上承载不同的消息类型组成。如下图：

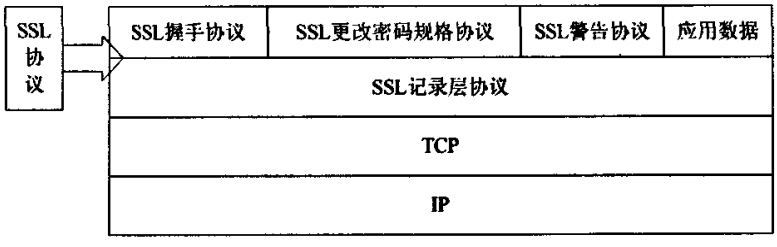

图 2-3 SSL 协议结构

2.3.1 SSL 记录层

SSL 记录层协议是建立在可靠的传输协议（如 TCP）基础上，它使用对称加密来保证连接安全性，使用 HMAC 算法来保证完整性，实现压缩/解压缩数据包；用来封装高层的协议。SSL 记录层的主要作用是数据包分片，数据包压缩，MAC 计算，数据加密，封装[20]。

分片：SSL 的应用数据将分割成一系列的片段。每个片段的最大长度为 2^{14} 字节。

压缩：数据经过分片后，就要经过压缩。

MAC 计算：将经过压缩的数据包。计算 MAC 值。

加密：如果是分组密码，输入的长度为分组长度的倍数。有必要增加一定数量的填充字节。

封装：对加密的数据和 MAC，加上记录头进行封装。

SSL 记录层协议的操作过程如下示意图：

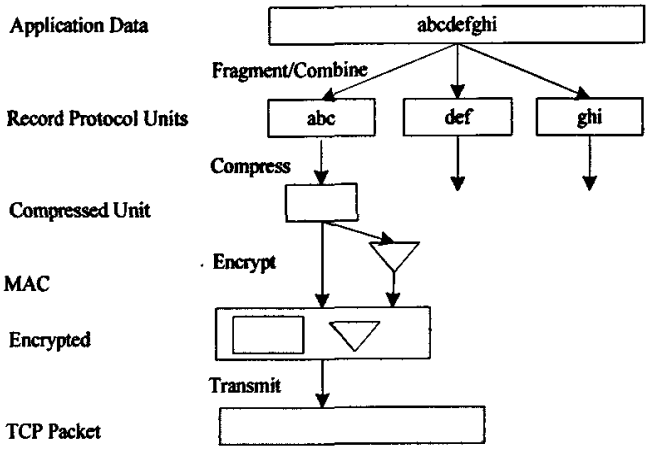

图 2-4 SSL 记录层操作过程

2.3.2 SSL 握手协议

SSL 握手协议是用来在客户端和服务器端传输应用数据之前建立安全通信机制。首次通信时，双方通过握手协议协商密钥加密算法、数据加密算法和摘要算法。然后互相验证对方身份，最后使用协商好的密钥交换算法产生一个只有双方知道的秘密信息，客户端和服务器各自根据这个秘密信息确定数据加密算法的参数（一般是密钥）。SSL 协议在网络协议族中处于 HTTP、FTP、TELNET 等应用层协议和 TCP、UDP 等传输层协议之间，其对应用层透明。SSL 握手过程如图 2-5 所示[21]：

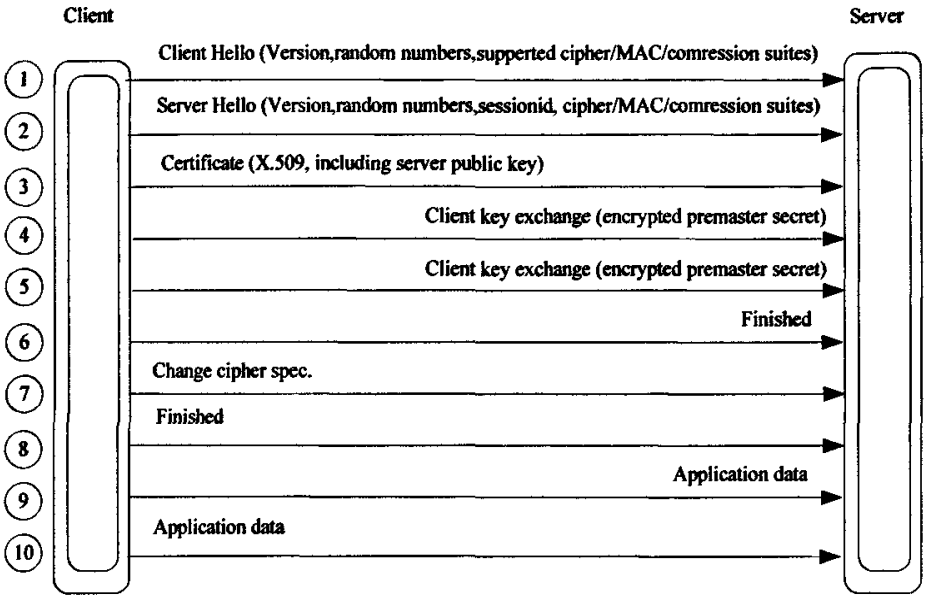

图 2-5 SSL 握手过程

（1）客户端向 Server 端发送客户端 SSL 版本号、加密算法设置、随机数和其它服务器需要用于和客户端通讯的数据。

（2）服务端向客户端发送服务器的 SSL 版本号、加密算法设置、随机数。另外，服务器还可以发送自己的数字证书，密钥交换和请求证书。

（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，用户将得到一个警告，然后加密数据连接无法建立。如果成功，则继续下一步。

（4）客户端用从握手建立开始至现在产生的所有数据，创建连接所用的 Pre_master_secret，用服务器的公钥加密（在第 2 步中传送的服务器证书中得。或者由服务器生成临时密码，并使用 server_key_exchange 消息发送公钥），传送给服务端。

（5）如果服务器也请求客户端验证，客户端将对另外一份不同于上次用于建立加密连接使用的数据进行签名。在这种情况下，客户端会把本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 Pre_master_secret。

（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，连蕤将被中止。如果被成功认证，服务器用自己的私钥加密

Pre_master_secret，然后执行一系列步骤产生 Master_Secret。

（7）服务器和客户端同时产生 Master key，之后将所有的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 Master key 加密。至此，它会传送一个单独的信息标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 Master key 加密。至此，它会传送一个单独的信息标示服务器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。客户端和服务端开始用 Master key 加密，解密双方交互的所有数据。

SSL 协议状态变化图如下[22]：

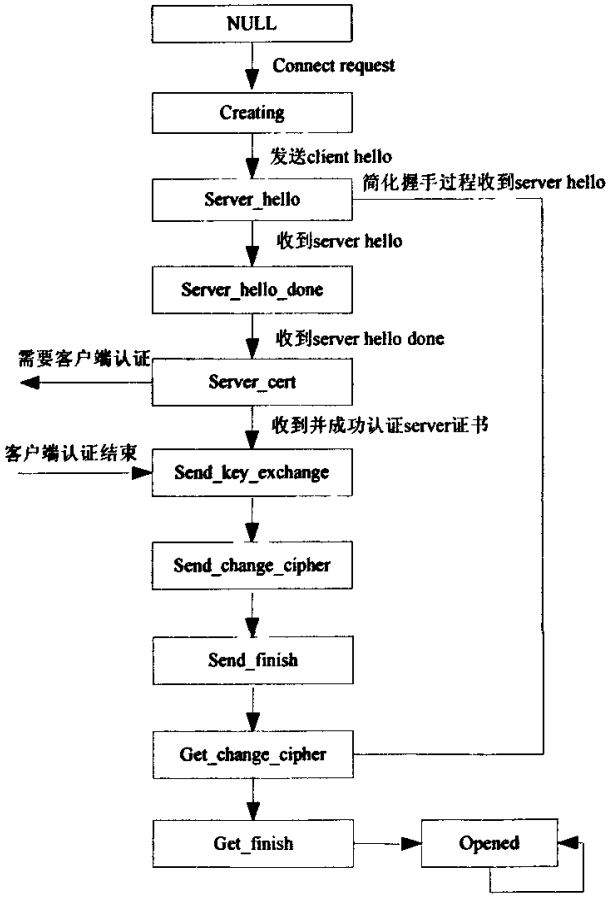

图 2-6　SSL 状态变化

2.4 更改密码说明协议

此协议由一条消息组成，可由客户端或服务器发送，通知接收方后面的记录将被新协商的密码说明和密钥保护。接收方得此消息后，立即指示记录层把即将读状态变成当前读状态，发送方发送此消息后，应立即指示记录层把即将写状态变成当前写状态。

2.5 警告协议

警告消息传达消息的严重性并描述警告[23]。一个致命的警告将立即终止连接。与其他消息一样，警告消息在当前状态下被加密和压缩。

警告消息有以下几种：

关闭通知消息、意外消息、错误记录 MAC 消消息、解压失败消息、握手失败消息、无证书消息、错误证书消息、不支持的证书消息、证书撤回消息、证书期满消息、证书未知消息、非法参数消息等。

2.6 本章小结

本章介绍了 SSL 协议和 SSL 设计思路上安全性的特点。

着重介绍了 SSL 的协议结构，SSL 记录层协议、握手协议、密码更改协议、警告协议等。其中 SSL 记录层协议是握手协议、密码更改协议、警告协议的基础。

3 OpenSSL 方法与技术分析

3.1 OpenSSL 模块分析

Eric A. Young 和 Tim J. Hudson，自 1995 年开始编写后来具有巨大影响的 OpenSSL 软件包，OpenSSL 是开放源代码的软件包， 1998 年，OpenSSL 项目组接管了 OpenSSL 的开发工作，并推出了 OpenSSL 的 0.9.1 版，到目前为止，OpenSSL 的算法已经非常完善，对 SSL2.0、SSL3.0 以及 TLS1.0 都支持。OpenSSL 目前最新的版本是 0.9.8 版。

OpenSSL 采用 C 语言作为开发语言，使得 OpenSSL 具有优秀的跨平台性能。OpenSSL 支持 Linux、Windows、BSD、Mac、VMS 等平台，使得 OpenSSL 具有广泛的适用性。

3.1.1 OpenSSL 源代码模块结构

OpenSSL 整个软件包大概可以分成三个主要的功能部分[7]：密码算法库、SSL 协议库以及应用程序。OpenSSL 的目录结构也是围绕这三个功能部分进行规划的。

表 3-1 OpenSSL 部分模块的功能说明

目录名	功能描述
Crypto	OpenSSL 所有加密算法源码文件和相关标准如 X.509 源码文件，是 OpenSSL 中最重要的目录，包含了 OpenSSL 密码算法库的所有内容。
SSL	存放 OpenSSL 中 SSL 协议各个版本和 TLS 1.0 协议源码文件，包含了 OpenSSL 协议库的所有内容。
Apps	存放 OpenSSL 中所有应用程序源码文件，如 CA、X509 等应用程序的源文件就存放在这里。
Doc	存放了 OpenSSL 中所有的使用说明文档，包含三个部分：应用程序说明文档、加密算法库 API 说明文档以及 SSL 协议 API 说明文档。
Demos	存放了一些基于 OpenSSL 的应用程序例子，这些例子一般都很简单，演示怎么使用 OpenSSL 其中的一个功能。
Include	存放了使用 OpenSSL 的库时需要的头文件。
Test	存放了 OpenSSL 自身功能测试程序的源码文件。

OpenSSL 的算法目录 Crypto 目录包含了 OpenSSL 密码算法库的所有源代码文件，是 OpenSSL 中最重要的目录之一。OpenSSL 的密码算法库包含了 OpenSSL 中所有密码算法、密钥管理和证书管理相关标准的实现。

3.1.2 OpenSSL 加密库调用方式

OpenSSL 是全开放的和开放源代码的工具包,，实现安全套接层协议（SSL v2/v3） 和传输层安全协议（TLS v1） 以及形成一个功能完整的通用目的的加密库 SSLeay ，OpenSSL Engine 的加密库结构简图[24]。如图 3-1 所示，从图 3-1 可以看出，应用程序可通过三种方式调用 SSLeay。一是直接调用，二是通过 OpenSSL 加密库接口调用，第三还可以通过 Engine 平台和 OpenSSL 对象调用。除了 SSLeay 外，用户可通过 Engine 安全平台访问 CSP 。

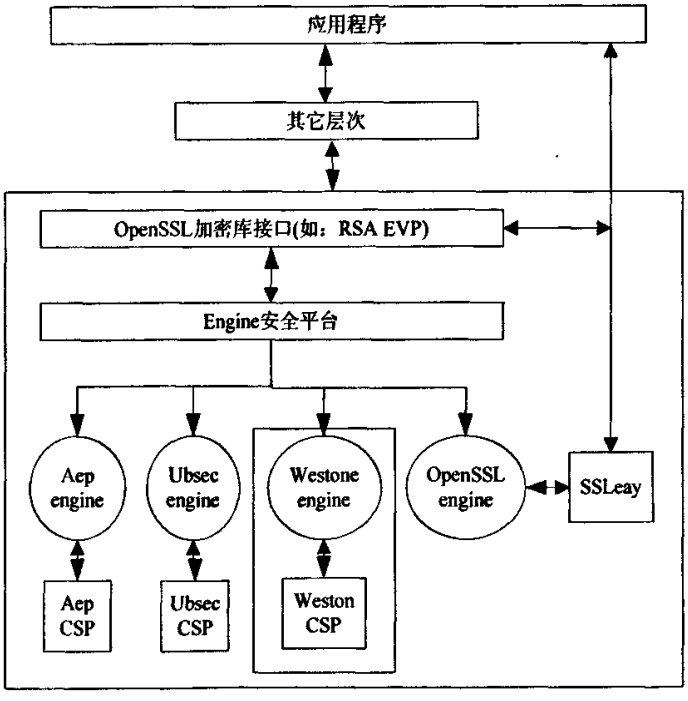

图 3-1 OpenSSL 加密 Engineer

使用 Engine 技术的 OpenSSL 已经不仅仅是一个密码算法库，而是一个提供通用加解密接口的安全框架[25]，在使用时，只要加载了用户的 Engine 模块，应用程序中所调用的 OpenSSL 加解密函数就会自动调用用户自己开发的加解密函数来完

成实际的加解密工作。这种方法将底层硬件的复杂多样性与上层应用分隔开，大大降低了应用开发的难度。

3.1.3 OpenSSL 支持的对称加密算法

OpenSSL 一共提供了 8 种对称加密算法，其中 7 种是分组加密算法，仅有的一种流加密算法是 RC4。这 7 种分组加密算法分别是 AES、DES、Blowfish、CAST、IDEA、RC2、RC5，都支持电子密码本模式（ECB）、加密分组链接模式（CBC）、加密反馈模式（CFB）和输出反馈模式（OFB）四种常用的分组密码加密模式。其中，AES 使用的加密反馈模式（CFB）和输出反馈模式（OFB）分组长度是 128 位，其它算法使用的则是 64 位。事实上，DES 算法里面不仅仅是常用的 DES 算法，还支持三个密钥和两个密钥 3DES 算法[26]。OpenSSL 还使用 EVP 封装了所有的对称加密算法，使得各种对成加密算法能够使用统一的 API 接口 EVP_Encrypt 和 EVP_Decrypt 进行数据的加密和解密[27]，大大提供了代码的可重用性能。

3.1.4 OpenSSL 支持的非对称加密算法

OpenSSL 一共实现了 4 种非对称加密算法，包括 DH 算法、RSA 算法、DSA 算法和椭圆曲线算法（EC）。DH 算法一般用户密钥交换。RSA 算法既可以用于密钥交换，也可以用于数字签名[28]，当然，如果你能够忍受其缓慢的速度，那么也可以用于数据加密。DSA 算法则一般只用于数字签名。

跟对称加密算法相似，OpenSSL 也使用 EVP 技术对不同功能的非对称加密算法进行封装，提供了统一的 API 接口。如果使用非对称加密算法进行密钥交换或者密钥加密，则使用 EVP_Seal 和 EVP_Open 进行加密和解密；如果使用非对称加密算法进行数字签名，则使用 EVP_Sign 和 EVP_Verify 进行签名和验证。

3.1.5 OpenSSL 支持的信息摘要算法

OpenSSL 实现了 5 种信息摘要算法[18]，分别是 MD2、MD5、MDC2、SHA（SHA1）和 RIPEMD。SHA 算法事实上包括了 SHA 和 SHA1 两种信息摘要算法，此外，OpenSSL 还实现了 DSS 标准中规定的两种信息摘要算法 DSS 和 DSS1。

OpenSSL 采用 EVP_Digest 接口作为信息摘要算法统一的 EVP 接口，对所有信

息摘要算法进行了封装，提供了代码的重用性。当然，跟对称加密算法和非对称加密算法不一样，信息摘要算法是不可逆的，不需要一个解密的逆函数。

3.1.6 OpenSSL 密钥和证书管理

OpenSSL 实现了 ASN.1 的证书和密钥相关标准，提供了对证书、公钥、私钥、证书请求以及 CRL 等数据对象的 DER、PEM 和 BASE64 的编解码功能[18]。OpenSSL 提供了产生各种公开密钥对和对称密钥的方法、函数和应用程序，同时提供了对公钥和私钥的 DER 编解码功能。并实现了私钥的 PKCS#12 和 PKCS#8 的编解码功能。OpenSSL 在标准中提供了对私钥的加密保护功能，使得密钥可以安全地进行存储和分发。

在此基础上，OpenSSL 实现了对证书的 X.509 标准编解码、PKCS#12 格式的编解码以及 PKCS#7 的编解码功能。并提供了一种文本数据库，支持证书的管理功能，包括证书密钥产生、请求产生、证书签发、吊销和验证等功能。

事实上，OpenSSL 提供的 CA 应用程序就是一个小型的证书管理中心（CA），实现了证书签发的整个流程和证书管理的大部分机制。

3.2 面向对象与 OpenSSL

OpenSSL 支持常见的密码算法。OpenSSL 成功的运用了面向对象的方法与技术，才使得它能支持众多算法并能实现 SSL 协议。OpenSSL 可贵之处在于它利用面向过程的 C 语言去实现了面向对象的思想。

面向对象方法是一种运用对象、类、继承、封装、聚合、消息传递、多态性等概念来构造系统的软件开发方法[29]。

面向对象方法与技术起源于面向对象的编程语言（OOPL）。但是，面向对象不仅是一些具体的软件开发技术与策略，而且是一整套关于如何看待软件系统与现实世界的关系、以什么观点来研究问题并进行求解、以及如何进行系统构造的软件方法学。概括地说，面向对象方法的基本思想是，从现实世界中客观存在的事物（即对象）出发来构造软件系统，并在系统构造中尽可能运用人类的自然思维方式。

面向对象方法强调直接以问题域（现实世界）中的事物为中心来思考问题、认

识问题，并根据这些事物的本质特征，把它们抽象地表视为系统中的对象，作为系统的基本构成单位。这可以使系统直接地映射问题域，保持问题域中事物及其互相关系的本来面貌。

结构化方法采用了许多符合人类思维习惯的原则与策略（如，自顶向下、逐步求精）。面向对象方法则更加强调运用人类在日常的逻辑思维中经常采用的思想方法与原则，例如，抽象、分类、继承、聚合、封装等等。这使得软件开发者能更有效地思考问题，并以其他人也能看得懂的方式把自己的认识表达出来。

具体地讲，面向对象方法有如下一些主要特点：

（1） 从问题域中客观存在的事物出发来构造软件系统，用对象作为这些事物的抽象表示，并以此作为系统的基本构成单位。

（2） 事物的静态特征（即可以用一些数据来表达的特征）用对象的属性表示，事物的动态特征（即事物的行为）用对象的服务表示。

（3） 对象的属性与服务结合成一体，成为一个独立的实体，对外屏蔽其内部细节（称作封装）。

（4） 对事物进行分类。把具有相同属性和相同服务的对象归为一类，类是这些对象的抽象描述，每个对象是它的类的一个实例。

（5） 通过在不同程度上运用抽象的原则（较多或较少地忽略事物之间的差异），可以得到较一般的类和较特殊的类。子类继承超类的属性与服务，面向对象方法支持对这种继承关系的描述与实现，从而简化系统的构造过程及其文档。

（6） 复杂的对象可以用简单的对象作为其构成部分（称作聚合）。

（7） 对象之间通过消息进行通信，以实现对象之间的动态联系。

（8） 通过关联表达对象之间的静态关系。

概括以上几点可以看到，在用面向对象方法开发的系统中，以类的形式进行描述并通过对类的引用而创建的对象是系统的基本构成单位。这些对象对应着问题域中的各个事物，他们内部的属性与服务刻画了事物的静态特征和动态特征。对象类之间的继承关系、聚合关系、消息和关联如实地表达了问题域中事物之间实际存在的各种关系。因此，无论系统的构成成分，还是通过这些成分之间的关系而体现的系统结构，都可以直接地映射问题域。

面向对象方法代表了一种贴近自然的思维方式，它强调运用人类在日常的逻辑思维中经常采用的思想方法与原则。面向对象方法中的抽象、分类、继承、聚合、封装等等思维方法和分析手段，能有效地反映客观世界中事物的特点和相互的关系。而面向对象方法中的继承、多态等特点，可以提高过程模型的灵活性、可重用性。因此，应用面向对象的方法将降低工作流分析和建模的复杂性，并使工作流模型具有较好的灵活性，可以较好地反映客观事物。

图 3-3 所示在磋商后确认某一种对称加密算法后，OpenSSL 调用该算法的 UML 图。

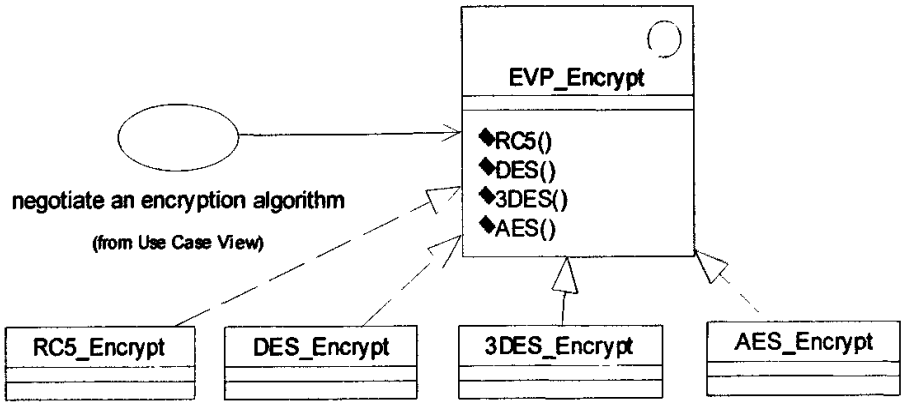

图 3-3 OpenSSL 由接口到多态实现的具体加密算法

在 OpenSSL 源代码对文件及网络操作封装成 BIO。BIO 基本几乎封装了除了证书处理外的 OpenSSL 所有的功能，包括加密库以及 SSL/TLS 协议。当然，它们都只是在 OpenSSL 其它功能之上封装搭建起来的，但却方便了不少。OpenSSL 对各种加密算法封装。就可以使用相同的代码但采用不同的加密算法进行数据的加密和解密。

3.2.1 BIO 接口

在 OpenSSL 源代码中，I/O 操作主要有网络操作，磁盘操作。为了方便调用者实现其 I/O 操作，OpenSSL 源代码中将所有的与 I/O 操作有关的函数进行统一封装。即无论是网络还是磁盘操作，其接口是一样的。对于函数调用者来说，以统一的接

口函数去实现其真正的I/O操作。为了达到此目的，OpenSSL采用BIO抽象接口。BIO是在底层覆盖了许多类型I/O接口细节的一种应用接口，如果在程序中使用BIO，那么就可以和SSL连接、非加密的网络连接以及文件I/O进行透明的连接。

BIO接口的定义如下：

```
struct bio_st
        {
.....
        BIO_METHOD *method;
.....
};
```

其中BIO_METHOD结构体是各种函数的接口定义。如果是文件操作时，此结构体如下：

```
 static BIO_METHOD methods_filep=
        {
        BIO_TYPE_FILE,
        "FILE pointer",
        file_write,
        file_read,
        file_puts,
        file_gets,
        file_ctrl,
        file_new,
        file_free,
        NULL,
        };
```

以上定义7个文件操作的接口函数的入口。这7个文件操作函数的具体实体与操作系统提供的API有关。

BIO_METHOD结构体如果用于网络操作，其结构体如下：

```
static BIO_METHOD methods_sockp=
    {
    BIO_TYPE_SOCKET,
    "socket",
    sock_write,
    sock_read,
    sock_puts,
    sock_ctrl,
    sock_new,
    sock_free,
    NULL,
    };
```

它跟文件类型 BIO 在实现的动作上基本上是一样的。只不过是前缀名和类型字段的名称不一样。其实在象 Linux 这样的系统里，Socket 类型跟 fd 类型是一样，他们是可以通用的，但是，为什么要分开来实现呢，那是因为有些系统如 windows 系统，socket 跟文件描述符是不一样的，所以，为了平台的兼容性，openssl 就将这两类分开来了。

3.2.2 EVP 接口

EVP 系列的函数定义包含在"evp.h"里面，这是一系列封装了 openssl 加密库里面所有算法的函数。通过这样的统一的封装，使得只需要在初始化参数的时候做很少的改变，就可以使用相同的代码但采用不同的加密算法进行数据的加密和解密。

EVP 系列函数主要封装了三大类型的算法，要支持全部这些算法，请调用 OpenSSL_add_all_algorithms 函数。

【公开密钥算法】

函数名称：EVP_Seal*...*，EVP_Open*...*

功能描述：该系列函数封装提供了公开密钥算法的加密和解密功能，实现了电子信封的功能。

相关文件：p_seal.c,p_open.c

【数字签名算法】

函数名称：EVP_Sign*...*，EVP_Verify*...*

功能描述：该系列函数封装提供了数字签名算法和功能。

相关文件：p_sign.c,p_verify.c

【对称加密算法】

函数名称：EVP_Encrypt*...*

功能描述：该系列函数封装提供了对称加密算法的功能。

相关文件：evp_enc.c,p_enc.c,p_dec.c,e_*.c

【信息摘要算法】

函数名称：EVP_Digest*...*

功能描述：该系列函数封装实现了多种信息摘要算法。

相关文件：digest.c,m_*.c

【信息编码算法】

函数名称：EVP_Encode*...*

功能描述：该系列函数封装实现了 ASCII 码与二进制码之间的转换函数和功能。下面将简单分析对称密算法 API。其它的算法与此类似。对称加密算法公开的 API 如下：

int EVP_CIPHER_CTX_init（EVP_CIPHER_CTX *a）；

该函数初始化一个 EVP_CIPHER_CTX 结构体，只有初始化后该结构体才能在下面提到的函数中使用。操作成功返回 1，否则返回 0。

int EVP_EncryptInit_ex（EVP_CIPHER_CTX *ctx, const EVP_CIPHER *type, ENGINE *impl, unsigned char *key, unsigned char *iv）；

该函数采用 ENGINE 参数 impl 的算法来设置并初始化加密结构体。其中，参数 ctx 必须在调用本函数之前已经进行了初始化。参数 type 通常通过函数类型来提供参数，如 EVP_des_cbc 函数的形式，对称加密算法的类型。如果参数 impl 为 NULL，那么就会使用缺省的实现算法。参数 key 是用来加密的对称密钥，iv 参数是初始化向量（如需要）。在算法中真正使用的密钥长度和初始化密钥长度是根据算法来决

定的。在调用该函数进行初始化的时候，除了参数 type 之外，所有其它参数可以设置为 NULL，留到以后调用其它函数的时候再提供，这时候参数 type 就设置为 NULL 就可以了。在缺省的加密参数不合适的时候，可以这样处理。操作成功返回 1，否则返回 0。

int EVP_EncryptUpdate（EVP_CIPHER_CTX *ctx, unsigned char *out, int *outl, unsigned char *in, int inl）；

该函数执行对数据的加密。该函数加密从参数 in 输入的长度为 inl 的数据，并将加密好的数据写入到参数 out 里面去。可以通过反复调用该函数来处理一个连续的数据块。写入到 out 的数据数量是由已经加密的数据的对齐关系决定的，理论上来说，从 0 到（inl+cipher_block_size-1）的任何一个数字都有可能（单位是字节），所以输出的参数 out 要有足够的空间存储数据。写入到 out 中的实际数据长度保存在 outl 参数中。操作成功返回 1，否则返回 0。

int EVP_EncryptFinal_ex（EVP_CIPHER_CTX *ctx, unsigned char *out,int *outl）；

该函数处理最后（Final）的一段数据。在函数在 padding 功能打开的时候（缺省）才有效，这时候，它将剩余的最后的所有数据进行加密处理。该算法使用标志的块 padding 方式（AKA PKCS padding）。加密后的数据写入到参数 out 里面，参数 out 的长度至少应该能够一个加密块。写入的数据长度信息输入到 outl 参数里面。该函数调用后，表示所有数据都加密完了，不应该再调用 EVP_EncryptUpdate 函数。如果没有设置 padding 功能，那么本函数不会加密任何数据，如果还有剩余的数据，那么就会返回错误信息，也就是说，这时候数据总长度不是块长度的整数倍。操作成功返回 1，否则返回 0。

PKCS padding 标准是这样定义的，在被加密的数据后面加上 n 个值为 n 的字节，使得加密后的数据长度为加密块长度的整数倍。无论在什么情况下，都是要加上 padding 的，也就是说，如果被加密的数据已经是块长度的整数倍，那么这时候 n 就应该等于块长度。比如，如果块长度是 9，要加密的数据长度是 11，那么 5 个值为 5 的字节就应该增加在数据的后面。

3.3 OpenSSL client/server 主要流程

OpenSSL 采用了接口、封装的技术，对外调用了提供了简单易用的 API 接口。在此给出 OpenSSL client/server 主要的流程[30]。此流程框架能基本实现 SSL 过程。

上下文初始化

```
/* OpenSSL headers */
#include "openssl/bio.h"
#include "openssl/ssl.h"
#include "openssl/err.h"
/* Initializing OpenSSL */
SSL_load_error_strings（）;
ERR_load_BIO_strings（）;
OpenSSL_add_all_algorithms（）;
```

建立非安全连接

```
bio = BIO_new_connect（"hostname:port"）;
if（bio == NULL）
{
    /* Handle the failure */
}

if（BIO_do_connect（bio） <= 0）
{
    /* Handle failed connection */
}
```

与服务器进行通信

```
int x = BIO_read（bio, buf, len）;
```

```
if（x == 0）
{
    /* Handle closed connection */
}
else if（x < 0）
{
   if（! BIO_should_retry（bio））
    {
        /* Handle failed read here */
    }
    /* Do something to handle the retry */
}

if（BIO_write（bio, buf, len） <= 0）
{
    if（! BIO_should_retry（bio））
    {
        /* Handle failed write here */
    }
    /* Do something to handle the retry */
}
```

建立安全连接

```
SSL_CTX * ctx = SSL_CTX_new（SSLv23_client_method（））;
```

加载可信任证书库

```
if（! SSL_CTX_load_verify_locations（ctx, "/path/to/TrustStore.pem", NULL））
{
    /* Handle failed load here */
```

```
}
if (! SSL_CTX_load_verify_locations (ctx, NULL, "/path/to/certfolder"))
{
    /* Handle error here */
}
bio = BIO_new_ssl_connect (ctx) ;
BIO_get_ssl (bio, & ssl) ;
SSL_set_mode (ssl, SSL_MODE_AUTO_RETRY) ;
/* Attempt to connect */
BIO_set_conn_hostname (bio, "hostname:port") ;
/* Verify the connection opened and perform the handshake */
if (BIO_do_connect (bio) <= 0)
{
    /* Handle failed connection */
}
if (SSL_get_verify_result (ssl) != X509_V_OK)
{
    /* Handle the failed verification */
}
SSL_CTX_free (ctx) ;
```

3.4 OpenSSL 源代码中 C 程序设计技巧

本节着重探讨 OpenSSL 源代码中用面向过程的 C 语言去实现面向对象的思想。在源代码中大量运用了面象对象的继承，多态，接口（纯虚函数）等。

3.4.1 接口（纯虚函数）

框架化编程的一个核心问题是抽象[31]，用抽象的对象构建程序的主体框架，这是面向对象编程的普遍思想。用抽象构建框架，再加上多态就形成了一个完整的程

序。

以 ssl 公开的调用函数为例，

```
typedef struct ssl_method_st
{
...
    int  (*ssl_new)(SSL *s);
    int  (*ssl_accept)(SSL *s);
    int  (*ssl_connect)(SSL *s);
    int  (*ssl_read)(SSL *s,void *buf,int len);
    int  (*ssl_peek)(SSL *s,void *buf,int len);
    int  (*ssl_write)(SSL *s,const void *buf,int len);
    int  (*ssl_shutdown)(SSL *s);
......
};
```

OpenSSL 对于不同版本的 SSL 的调用函数都是一样的。但每个版本的 SSL 对于以上每个 API 的实现都不同。以 int (*ssl_connect)(SSL *s) 为例，在 SSLv2 中，此函数实现的实际函数是 ssl2_connect ()；在 SSLv3 中，此函数实现的实际函数是 ssl3_connect ()。在 OpenSSL 源代码中，大量的公开接口采用以上的方式。将接口与具体的实现分离。以上的编程思想与目前流行的面向对象的继承与多态是完全吻合的。但作者在上世纪 90 年初就能灵活运用此方法与技巧。

3.4.2 宏

在面向对象语言中，象 Java 或 C++，提供了 wrap 类。这些类能将不同类型的对象包含其中。Wrap 类有点象我们生活中的容器。可以装入不同的东西。但面向过程的 C 语言没有 wrap 类或结构。但作者巧妙地运用宏及数据结构，实现了 wrap 类的思想。

在 C 语言的宏定义中，通常见到别名的方式，例如：

```
#define PORT        "16001"。
```

还有宏连接符"##"，它表示将##的变量替换。例如：

#define STACK_OF（T） StackOf##T

如果类型 T 为 int，以上等价于

#define STACK_OF（int） StackOfint

在分析了##宏连字符后，下面用简单的例子来说明 C 语言用宏来实现 warp 类的思想。

```
#define STACK_OF（T） StackOf##T
#define STACK_DECLARE（Type） \
class STACK_OF（Type） {  \
public:          \
STACK_OF（Type）（void）: sp（0） {} \
void push（Type e） { buff[sp++] = e;} \
……}

STACK_DECLARE（int）;
STACK_DECLARE（double）;
```

当编译器碰到 STACK_DECLEARE 两行时，它自动解析成如下代码。如果在 Linux 环境，可以用 gcc –E 查看编译器执行的结果。

```
class StackOfint {
        public : StackOfint（void） : sp（0） {}
        void push（int e） {
            buff[sp++] = e;
        }
    };
class StackOfdouble {
        public : StackOfdouble（void） : sp（0） {}
        void push（double e） {
            buff[sp++] = e;
```

```
    }
};
```

在OpenSSL源代码中，经常将需要用到的数据以STACK的方式保存在内存中。在不同的情形下装入STACK的数据结构不同，但其实现方式与以上内容相同。

3.5 本章小结

在本章中对OpenSSL整体分析。OpenSSL源程序模块结构，OpenSSL支持的各种类型的加密算法。面向对象的思想在OpenSSL源代码中的实现方法与技巧。分析了两个重要的接口，网络及I/O操作接口BIO，各种中密算法接口EVP。分析了接口与宏的实现技术的特点。分析了OpenSSL客户端，服务端通讯的程度基本框架与流程。

4 OpenSSL 过程分析

SSL 握手有三个目的。第一，客户端与服务器需要就一组用于保护数据的算法达成一致。第二，它们需要确定一组由那些算法所使用的加密密钥。第三，可选择对客户端认证。

本章将利用 Visual C++6 为工具，以 OpenSSL0.97f 版本为基础，从 OpenSSL client 入手，详细分析在 OpenSSL 如何实现 SSL 协议的握手过程。

因为 OpenSSL 代码可以从 www.openssl.org 下载，本章将只给出代码的程序文件的名称及相对路径，函数 API 的名称。重要时给出代码片段。

4.1 SSL 加密算法库初始化

OpenSSL 支持各种对称加密，非对称加密，摘要算法等。在 OpenSSL 初始运行时，OpenSSL 初始化时加载了对称加密算法的 DES，IDEA，RC2，RC4，AES。信息摘要算法的 MD2，MD5，SHA。非对称算法的 DSA，RSA 等。

在 OpenSSL 加密库初始化过程中，它用了许多的数据结构及复杂函数调用，最终的目的是构建全局变量的数据结构 LHASH，此数组结构存在于内存中。直到 OpenSSL 结束运行。

按照与 API 调用相反的方向来分析。即由最底层的 API 或数据结构入手。最后按 API 调用方向综合之前的分析过程，

重点分析 DES 算法加载过程，其它算法与它类似。

4.1.1 EVP_des_cbc（）定义加密结构体

EVP_des_cbc（）利用了宏来定义此 DES 的 CBC 的结构体。宏代码可读性差，但可重用性好，它可以定义不同方式的 DES。如 cfb，cbc 等。此宏将关联到

【路径】：E:\openssl-0.9.7f\crypto\evp\e_des.c

【API】：BLOCK_CIPHER_defs（des, DES_key_schedule, NID_des, 8, 8, 8, 64,
EVP_CIPH_FLAG_FIPS, des_init_key, NULL,

EVP_CIPHER_set_asn1_iv,

EVP_CIPHER_get_asn1_iv,

NULL）

而宏 BLOCK_CIPHER_defs（）的定义在

【路径】：E:\openssl-0.9.7f\crypto\evp\evp_locl.h

【API】：#define BLOCK_CIPHER_defs（cname, kstruct, \

nid, block_size, key_len, iv_len, cbits, flags, \

init_key, cleanup, set_asn1, get_asn1, ctrl） \

将此宏展开后，代码如下：

#define BLOCK_CIPHER_defs（cname, kstruct, \

nid, block_size, key_len, iv_len, flags,\

init_key, cleanup, set_asn1, get_asn1, ctrl）\

static const EVP_CIPHER cname##_cbc = {\

nid##_cbc, block_size, key_len, iv_len, \

flags | EVP_CIPH_CBC_MODE,\

init_key,\

cname##_cbc_cipher,\

cleanup,\

sizeof(EVP_CIPHER_CTX)-sizeof((((EVP_CIPHER_CTX *)NULL)->c))

+\

sizeof ((((EVP_CIPHER_CTX *) NULL) ->c.kstruct)) ,\

set_asn1, get_asn1,\

ctrl, \

NULL \

};\

const EVP_CIPHER *EVP_##cname##_cbc（void） { return &cname##_cbc; }\

宏在 VC 中不能跟踪。通过复杂的宏替换 EVP_des_cbc（）宏的作用是初始化 DES 的 CBC 方式的 evp_cipher_st 结构体。

4.1.2 ASN1_OBJECT 类型常数数组

【路径】：E:\openssl-0.9.7f\crypto\objects\obj_dat.h

【API】：static ASN1_OBJECT nid_objs[]

此文件定义了常数数组，例如找出此数组下标 31 的元素如下：

{"DES-ECB","des-ecb",29,5,&（lvalues[187]）,0},

lvalues[187]在数组 lvalues 的值为

0x2B,0x0E,0x03,0x02,0x06,

OBJ_nid2sn（）返回此元素的 DES-ECB 或 des-ecb。

4.1.3 全局 LHASH 结构体及其操作

【路径】：E:\openssl-0.9.7f\inc32\openssl\lhash.h

在此文件中定义了 LHASH 和 LHASH_NODE 结构体。由此结构体定义可知，LHASH_NODE 是一个线性链表数据结构。LHASH_NODE 的 hash 是某个 hash 值，LHASH_NODE 的 data 是指向 evp_cipher_st 结构体的指针。LHASH 结构体是 LHASH_NODE 线性链表的头结点。

【路径】：E:\openssl-0.9.7f\crypto\lhash\lhash.c

【API】：void *lh_insert（LHASH *lh, const void *data）

此 API 是将 evp_cipher_st 结构体存入 LHASH 线性链表。在存入 LHASH 线性链表之前，要调用 LHASH_NODE **getrn（LHASH *lh, const void *data, unsigned long *rhash）函数得到 hash 值，在 LHASH 链表中找到空的 LHASH_NODE 结点。每当有 LHASH_NODE 结点存入时，会更新 LHASH 头结点的相关数据。如 num_inser 等。

OpenSSL 加密算法初始过程主要如下：

采用宏的方式初始化 evp_cipher_st 结构体。初始化全局线性链表 LHASH，并将 evp_cipher_st 结构化存入 LHASH_NODE 的结点。

4.2 磋商前加密套件准备

在 SSL 协议中，规定了若干个加密套件。这些加密套件用固定的编号来标识。

每个加密套件由认证算法、密钥交换算法、加密算法、摘要算法组成。例如加密套件 DHE_RSA_WITH_DES_CBC_SHA，其算法分别是认证算法为 RSA，密钥交换算法为 DHE，加密算法为 DES，摘要算法为 SHA。其在 SSL 协议中的编号为 0x0015。

在 SSL 握手的 Client Hello 报文中，客户端需要将其所支持的加密套件放在 Client Hello 报文中，服务器端也需要将其所支持的加密套件在初始化时载入。客户端与服务器端就利用了加密套件中的编号。如何筛选加密套件[32]，SSL 协议并无具体要求。

4.2.1 加密套件及加密算法定义及表示

在 OpenSSL 源代码中，它用 24 个 bit 来的掩码来表示认证算法掩码、密钥交换算法掩码、加密算法掩码、摘要算法掩码。24 个 bit 的位次由右到左的次序。

SSL_MKEY_MASK 0x0000003FL 密钥交换 111111 1-6 bit

SSL_AUTH_MASK 0x00000FC0L 认证 111111000000 7-12 bit

SSL_ENC_MASK 0x0087F000L 加密 100001111111000000000000 13-19,24bit

SSL_MAC_MASK 0x00180000L 摘要 110000000000000000000 21-21 bit

SSL_SSL_MASK 0x00600000L SSL 1100000000000000000000 22-23 bit

从右向左的 23 个 bit 中，每个 bit 代表了一个加密算法。以 SSL_MAC_MASK 为例：

```
#define SSL_SSL_MASK    0x00600000L
#define SSL_SSLV2       0x00200000L
#define SSL_SSLV3       0x00400000L
```

掩码的作用是与加密套件的算法的二进制表示进行 and 操作。

在源代码中，事先将 OpenSSL 支持的加密套件定义在全局数组变量中，如下。

【路径】：E:\openssl-0.9.7f\ssl\s3_lib.c

【API】：OPENSSL_GLOBAL SSL_CIPHER ssl3_ciphers[]

将 OpenSSL 支持的加密算法的结构体放入数组中。以其中某个元素为例：

```
/* Cipher 3A */
    {
```

```
1,
TLS1_TXT_ADH_WITH_AES_256_SHA,    --加密套件的名称
TLS1_CK_ADH_WITH_AES_256_SHA, --SSL 协议的编号 0x003a 宏
SSL_kEDH|SSL_aNULL|SSL_AES|SSL_SHA|SSL_TLSV1, --支持的算
法
SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
0,
256,              --算法的位数，32 字节
256,              --算法的位数  32 字节
SSL_ALL_CIPHERS,
SSL_ALL_STRENGTHS,
},
```

【路径】: E:\openssl-0.9.7f\ssl\ssl_ciph.c

【API】: SSL_CIPHER cipher_aliases[]

将 OpenSSL 加密套件中支持的每一个算法定义在此数组中。

OpenSSL 以 24 bit 来表示支持的算法。再以掩码将算法分为认证算法，加密算法，密钥交换算法，摘要算法等。同时将 SSL 协议中的加密套件，由多个算法组合表示。

4.2.2 加密套件筛选过程

在加密套件筛选过程中[33]，用到了两个数据结构，struct cipher_order_st，struct ssl_cipher_st。cipher_order_st 是线性链表，每个链表的节点包括了 struct ssl_cipher_st。

【路径】: E:\openssl-0.9.7f\ssl\ssl_ciph.c

【API】: static void ssl_cipher_collect_ciphers（const SSL_METHOD *ssl_method,
int num_of_ciphers, unsigned long mask, CIPHER_ORDER *co_list,
CIPHER_ORDER **head_p, CIPHER_ORDER **tail_p）

此 API 将 ssl3_ciphers 数组中初始定义的 SSL 协议中的 48 个加密套件，保存于

cipher_order_st 链表中。并设置了链表的首尾指针节点。

【路径】：E:\openssl-0.9.7f\ssl\ssl_ciph.c

【API】：static void ssl_cipher_collect_aliases（SSL_CIPHER **ca_list,
int num_of_group_aliases, unsigned long mask,
CIPHER_ORDER *head）

此 API 首先将 cipher_order_st 中 48 个加密套件保存在 ca_list 中。ca_list 双重指针相当于一个数组，数组中元素类型是 SSL_CIPHER，即 struct ssl_cipher_st。接着将 cipher_aliases 数组中的元素追加至 ca_list 中。

OpenSSL 中缺省的加密套件规则字符串如下：

ALL:!ADH:RC4+RSA:+SSLv2:@STRENGTH

加密套件规则以":"分隔。ALL 表示除 eNULL（不加密）外的所有加密算法。

！表示从加密套件中删除此加密算法。

+表示将加密算法移到加密算法的尾位。并不是新增加密算法。

@STRENGTH 表示加密算法按加密强度降排序。

在测试中，将加密套件规则设为 DES+RSA:SSLv3:@STRENGTH。经过筛选后的加密套件是以下三个。

<EXP1024-DES-CBC-SHA>

<DES-CBC-SHA>

<EXP-DES-CBC-SHA>

筛选过程如下：

解析加密套件规则，以":"为单元，将此单元的每个加密算法的字符串在之前保存的 ca_list 数组中去寻找，将找到的加密算法的 algorithms，mask，algo_strength，mask_strength 分别用二进制的或操作。

对 cipher_order_st 线性链表中的每一个加密算法用 cipher 表示。进行如下运算：

ma = mask&cipher->algorithms

ma_s = mask_strength & cipher ->algo_strength

运算结果符合下列条件之一就是符合加密套件规则的。

ma == 0 & ma_s == 0;

ma & algorithms == ma;

ma_s & algo_strength == ms。

当找到符合加密套件规则的加密算法时，在 cipher_order_st 链表中，将此加密算法节点移到链表的末尾。修改此加密算法状态为 active。

在对每个":"分隔符内的加密算法处理后，对 cipher_order_st 链表按每个加密算法的强度降排序。

之后将筛选过的加密套件保存在 stack 中。供握手时使用。

【路径】: E:\openssl-0.9.7f\ssl\ssl_ciph.c

【API】: STACK_OF（SSL_CIPHER） *ssl_create_cipher_list（const SSL_METHOD *ssl_method, STACK_OF（SSL_CIPHER） **cipher_list, STACK_OF（SSL_CIPHER） **cipher_list_by_id,const char *rule_str）

此 API 实现了如上所述的加密套件筛选全过程。

4.3 客户端与服务端通讯连接

不管连接是安全的还是不安全的，OpenSSL 都使用了一个名为 BIO 的抽象库来处理包括文件和套接字在内的各种类型的通信。这里对 BIO 库 API 非常之多，进行全面说明有点困难。将根据需要逐步分析。在建立连接（无论安全与否）之前，要创建一个指向 BIO 对象的指针。这类似于在标准 C 中为文件流创建 FILE 指针。

打开连接[34]

创建新的连接需要调用 BIO_new_connect 。可以在同一个调用中同时指定主机名和端口号。也可以将其拆分为两个单独的调用：一个是创建连接并设置主机名的 BIO_new_connect 调用，另一个是设置端口号的 BIO_set_conn_port （或者 BIO_set_conn_int_port ）调用。

一旦 BIO 的主机名和端口号都已指定，该函数会尝试打开连接。如果创建 BIO 对象时遇到问题，指针将会是 NULL。为了确保连接成功，必须执行 BIO_do_connect 调用。

与服务器进行通信

不管 BIO 对象是套接字还是文件，对其进行的读和写操作都是通过以下两个函数来完成的： BIO_read 和 BIO_write 。BIO_read 将尝试从服务器读取一定数目的字节。它返回读取的字节数、 0 或者 -1。在受阻塞的连接中，该函数返回 0，表示连接已经关闭，而 -1 则表示连接出现错误。在非阻塞连接的情况下，返回 0 表示没有可以获得的数据，返回 -1 表示连接出错。可以调用 BIO_should_retry 来确定是否可能重复出现该错误。

BIO_write 会试着将字节写入套接字。它将返回实际写入的字节数、0 或者 -1。同 BIO_read ，0 或 -1 不一定表示错误。 BIO_should_retry 是找出问题的途径。如果需要重试写操作，它必须使用和前一次完全相同的参数。

4.4 证书链及私钥文件加载

SSL 协议通常要求服务端提供证书[35]，客户端检测服务端的证书来决定服务端是被认证及可信任的。对等方需要验证签发的证书链。所以 SSL 服务端需要提供它的证书链给客户端。但客户端不一定需要提供证书。如果服务端需要客户端提供证书但客户端不能提供时，就不能建立安全连接。

OpenSSL 在握手时服务端提供客户端的证书，在创建 SSL 上下文结构之后，必须加载一个可信任证书链。这是成功验证每个证书所必需的。如果不能确认证书是可信任的，那么 OpenSSL 会将证书标记为无效（但连接仍可以继续）。

OpenSSL 附带了一组可信任证书。它们位于源文件树的 certs 目录中。不过，每个证书都是一个独立的文件 —— 也就是说，需要单独加载每一个证书。在 certs 目录下，还有一个存放过期证书的子目录。试图加载这些证书将会出错。

可以分别加载每一个文件，但为了简便起见，最新的 OpenSSL 发行版本的可信任证书通常存放在源代码档案文件中，这些档案文件位于名为“TrustStore.pem”的单个文件中。如果已经有了一个可信任证书库，并打算将它用于特定的项目中，那么只需使用您的文件（或者使用单独的函数调用将它们全部加载）即可。

可以调用 SSL_CTX_use_certificate_chain_file 来加载可信任证书链文件。证书

链文件格式为 PEM。这里要用到两个参数：上下文指针、可信任链文件的路径及文件名。如果指定成功，则返回 1，如果遇到问题，则返回 0。

SSL_CTX_use_certificate_chain_file（ctx, CERTFILE）

在客户端加载它自己证书链之时，SSL_CTX 上下文结构体需要该证书公钥的私钥[36]。通过 SSL_CTX_use_PrivateKey_file 来加载。这里要用三个参数：上下文指针，可信任证书链文件，文件的格式 PEM。

SSL_CTX_use_PrivateKey_file（ctx, CERTFILE, SSL_FILETYPE_PEM）

4.5 OpenSSL 握手过程分析

4.5.1 client hello

在握手之前，初始化 BUF_MEM 结构体，SSL3_BUFFER 读写数据结构体。MD5 和 SHA 算法的初始常数值。MD5 和 SHA 算法在加密库初化时已经指定了。

struct {

协议版本：2 byte;

随机数[37]：32 byte;

会话 ID：在没有建立安全连接前，值是 0;

加密套件：前 2 byte 是长度，后面是加密套件内容。

压缩算法：暂时未启用，固定 2 byte;

} ClientHello;

根据前面的自定义的加密套件规则条件，有 27 组加密套件符合要求。则 Client Hello 总长度是 2+32+1+2+54+2 = 93 byte。在记录层协议在握手报文前面有消息类别 1byte，握手报文的长度 3byte。ClientHello 被记录层封装后总长度是 93+4=97 byte。

其中加密套件内容是从加密磋商准备时，从 STACK 中读出来的。

在 SSLv3 规范中，加密套件的资料格式为 CipherSuite cipher_suites<0..2^16-1>。在规范中实现草案为 CipherSuite cipher_suites<2..2^16-1>，在实现中加密套件至少有 2byte。但此 2byte 是什么，规范中并不说明。从 OpenSSL 源代码中，可以看到，

此 2byte 就是加密套件的长度。

当客户端第一次与服务端连接时，并没有会话 ID。有的书上说此时会话 ID 的长度是 0。其实这个说法与源代码有出入。在源代码中，此会话 ID 的内容是 0，会话 ID 长度为 1。

加密套件的长度占 2byte，clienthello 报文长度用 3byte。其扩展方式是 c[0]=((p) >>) 16&0xff, c[1]= ((p) >>8) &0xff,c[2]= (p) &0xff。其中 p 是长度值。

此 97 个长度数据是放在 SSL->init_buf->data 中。

4.5.2 Server hello

客户端从服务端接收到 Server Hello 消息。客户端从网络读数据的过程如下：

因为记录层协议头是 5byte 长。客户端先读出 5byte 内容。分别读到了记录层的内容类型值为 22，表明是 handshake（22）。协议的版本是 3.0，握手报文消息长度为 74byte。

接着从网络读出 74byte 握手消息内容。此 74byte 中前 4byte 是握手消息头。后 70byte 是 server Hello 消息体内容。其内容分别是 2byte 协议版本，32byte 随机数，1byte sessionid 长度，32byte 的 sessionid 内容，2byte 的加密套件，1byte 的压缩算法，其值为 0，表示未采用压缩算法。70=2+32+1+32+2+1。

在 SSL 协议中，Server hello 规格如下：

```
struct {
    ProtocolVersion server_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suite;
    CompressionMethod compression_method;
} ServerHello;
```

在 OpenSSL 源程式中，增加了 1 个 byte 的 sessionID 长度。在 Client Hello 中有加密套件长度，但在 Server hello 报文中没有，是因为客户端需要将它支持的加密套件筛选出来，所以需要加密套件的长度。SSL 服务端会根据其支持的加密套件，

从客户端送上来的多个加密套件中选择其一。所以不需要在 server hello 报文中不需要加密套件长度。

4.5.3 Server certificate

由 SSLv3 协议规格，服务端证书格式如下：

```
opaque ASN.1Cert<2^24-1>;
struct {
    ASN.1Cert certificate_list<1..2^24-1>;
} Certificate;
```

由协议规格可知证书的格式是 ASN.1 规范，有可能是证书链[38]。从 OpenSSL 源代码中可知，SSL 记录层协议中的服务端证书的报文结构如下：前 3byte 是证书链报文长度。在每一份证书的报文中，前 3 个 byte 是此份证书的报文长度，其后是证书以 ASN.1 格式的表示的证书内容，长度是该证书前 3byte 长度值。

表 4-1 SSL 证书链结构

证书链长度	第 i 个证书长度	第 i 个证书内容
3 byte	3 byte	第 i 个证书的长度

在客户端接收到服务端的证书时，它解析每一个证书到 X509 结构体，并放在 stack 中。接着验证证书，在握手时所提供的服务器的证书应该有一个名字与该服务器的主机名匹配。如果没有，那么这个证书就应该标记为值得怀疑的。内部验证过程已经对证书进行信任和有效期的验证；如果这个证书已经超期，或者包含一个不可信的签名，那么这个证书就会被标记为无效的。由于这不是 SSL 标准的一部分，因此 OpenSSL 并不需要根据主机名对该证书的名字进行检查。OpenSSL 在对证书进行验证时，有一些安全性检查并没有执行，包括证书的失效检查和对证书中通用名的有效性验证。

之后将根证书存放在上下文的 s->session->sess_cert 结构体中。

4.5.4 Server Hello Done

此消息是一条空消息。它表示服务器已经发送了在此阶段所要发送的全部信息。这条消息是必须的。

4.5.5 Client Key Exchange

Client Key Exchange 消息提供创建 pre_master_secret 时客户端所提供的资料。当使用 RSA 钥交换时，这就是指客户端产生一个 PreMasterSecret 结构并用服务器的 RSA 密钥对其进行加密。Pre_master_secret 是一个 48 byte 的值。其中由两字节版本号以及跟在后面的 46 字节随机产生。

在之前的 Server Hello 消息中包括了磋商的加密套件，当客户端得到此加密套件后，它将此加密套件的对应的相关算法存放在 SSL 上下文的 s->s3->tmp.new_cipher 结构中。由 s->s3->tmp.new_cipher->algorithms 的 24byte 的信息可得知加密套件的 4 个加密算法。由加密规则 RSA+3DES，选出的加密套件是 DES-CBC3-SHA，它的 algorithms 的二进制值是 10100000010000001000001，其认证算法是 RSA。

在 Server certificate 消息中，客户端将根证书的 X509 结构体存放在 SSL 上下文的 s->session->sess_cert->peer_pkeys 中。此时从此结构体中读出 RSA 结构体的内容。在 RSA 结构体中包括了服务端的 RSA 公钥。客户端生成 46byte 的随机数，再加上 SSL 版本号 2byte 放在 46byte 随机数组成 48byte 随机数。本次跟踪服务端的证书的公钥是 512bit。在 RSA 公钥加密时，明文长度需要与公钥长度保持一致。OpenSSL 客户端在 48byte 随机数后加了 PADDING 报文。使明文长度长度到 64 byte。这一点在 SSL 协议中并没有说明。接着用 RSA 公钥对随机数加密。得到 64byte 的密文。客户端将 64byte 的密文送服务端。客户端将 48byte 的随机数[39]，Client Hello 中的随机数和 Server Hello 中的随机数，按 SSL 协议中的主密钥生成规则生成主密钥。

```
master_secret =
  MD5（pre_master_secret + SHA（'A' + pre_master_secret +
      ClientHello.random + ServerHello.random））  +
  MD5（pre_master_secret + SHA（'BB' + pre_master_secret +
```

ClientHello.random + ServerHello.random)) +

MD5 (pre_master_secret + SHA ('CCC' + pre_master_secret + ClientHello.random + ServerHello.random)) ;

如图 4-1 为 master secret 生成示意图。

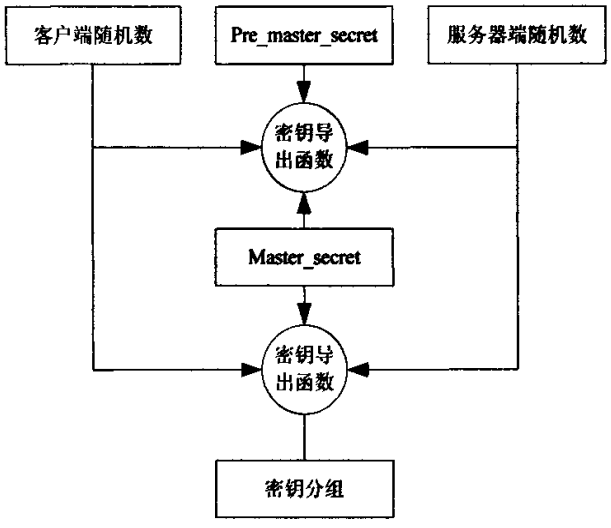

图 4-1 master_secret 的生成

4.5.6 Change Cipher Spec

此消息报文长度仅为 1byte。它表示发送实现已经切换至新磋商好的算法和密钥资料，而未来发送的消息将使用这些磋商后的算法加以保护。此消息并不算是握手过程中的一部分，相反而是有自己的内容类型。

4.5.7 Finished

Finished 消息是第一条使用磋商后的新的加密参数进行加密的消息。它使实现能够确认没有任何消息被篡改过。MD5 和 SHA 的计算规则如下：

md5_hash MD5（master_secret + pad2 +
MD5（handshake_messages + Sender +
master_secret + pad1)) ;

sha_hash SHA（master_secret + pad2 +

SHA（handshake_messages + Sender +

master_secret + pad1））；

其中 pad1 是 0x36byte 重复 36 次，pad2 是 0x5cbyte 重复 48 次。

Sender 是常数值 0x434C4E54。

handshake_messages 是所有中客户端发送的，接收服务端的握手消息。但不包括此 Finished 消息。

4.6 本章小结

本章从 OpenSSL 客户端为切入点，分析了 OpenSSL 加载加密算法及数据结构。双方进行加密套件的筛选规则及实现。客户端与服务端的通讯连接。在握手过程中的证书链及证书的私钥加载过程。分析了握手过程中的消息。并指出了 SSL 协议的 Client hello, Server hello, Client Key Exchange, Server certificate 等消息存在不足之处。

5 OpenSSL 测试

5.1 测试环境

硬件：CPU AMD3000+，硬盘 120G，内存 DDR 512M

操作系统：Windows 2000 SP4

SSL 客户端：iscribe-win32 及 SSL plugin，IE6.0（SSL3.0）

SSL 服务端：gmail.google.com

协议分析工具：ssldump-0.9b3

5.2 测试目的

SSL 协议的握手过程和应用数据加密的安全性与 SSL 客户端与服务端双方磋商的加密套件有直接的影响。

本测试的目的有四点：

第一，测试源代码中实现了 EVP, BIO 接口的多态性；

第二，测试了 SSL 客户端服务端双方磋商过程的细节；

第三，用 OpenSSL 客户端与第三方 Gmail 服务端通讯，分析握手过程；

第四，用 IE 客户端与第三方 Gmail 服务端通讯，将结果与第三项比对，IE 缺省的加密套件的安全性。

为了使测试更公正及科学。本次测试选择全球知名的公司 Goole 的 Gmail 邮件服务器为服务端进行测试。

Windows 平台的 SSL 客户端都集成于 IE,Netscape,及 Outlook 等。它没有提供第三方的 SSL 库接口。本测试的客户端选择 iscribe windows 版本。此 iscribe 可以收发邮件，它提供了嵌入第三方的 SSL 库接口。本测试在 iscribe 软件加载 OpenSSL 源代码编译之后的 ssleay32.dll 和 libeay32.dll 动态链接库。此时 iscribe 收发邮件就可以支持第三方的 SSL 客户端。

由于美国密码安全出口的限制，同样的 SSL 协议，但 IE 的加密套件磋商结果

不同，直接影响安全级性。本测试采用嵌入自己指定的加密套件的客户端，与 IE 客户端与同一服务端通讯来比对。

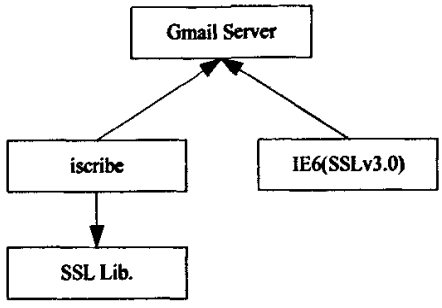

图 5-1 Gmail 服务器与 iscribe 和 IE 客户端测试过程

5.4 测试过程与结果

5.4.1 BIO 及 EVP 接口的实现测试

BIO 及 EVP 接口只能从跟踪源代码的调用方式看出来。

以跟踪 BIO_read（）接口为例，对外提供的 BIO_read（）操作。最终调用 socket 的 read（）操作来实现此接口的。调用关系如下：

BIO_read（s->rbio, &（s->s3->rbuf.buf[off+newb]）, max-newb）；

b->method->bread（b,out,outl）;

conn_read（BIO *b, char *out, int outl）;

read（b->num,out,outl）;

以 EVP_DigestInit_ex（）接口为例，对外提供的 EVP_DigestInit_ex（）操作。最终调用 MD5_Init（）的操作来实现此接口。调用关系如下：

EVP_DigestInit_ex（&（s->s3->finish_dgst1）,s->ctx->md5, NULL）;

MD5_Init（ctx->md_data）。

5.4.2 加密套件筛选结果

测试中将 OpenSSL 的加密套件规则设为：

#define SSL_DEFAULT_CIPHER_LIST　DES+RSA:SSLv3:@STRENGTH

测试结果中客户端将符合此规则的加密套件筛选如下：

<EXP1024-DES-CBC-SHA>

<DES-CBC-SHA>

<EXP-DES-CBC-SHA>

5.4.3 IE6 连接 Gmail 服务器

ssldump 指令格式如下[40]：

ssldump –Ad –i 网卡名

本测试的 ssldump 指令如下：

ssldump -Ad -i "\Device\Packet_{33D14A43-8238-4BEA-A993-AA8EF164E392}"

用 IE6 中的 SSLv3.0 连接 Gmail 服务端之后的跟踪结果如下：

New TCP connection #1: leicq（1274） <-> 216.239.63.19（443）

1 1 0.4547 （0.4547） C>SV3.0（97） Handshake

ClientHello

Version 3.0

random[32]=

45 31 97 fe df 36 da 9f 91 b2 6d 20 5c 8c 38 47

a9 3c 45 cb 67 5f 3a 59 5a ec 3f 92 0a 25 f7 0f

resume [32]=

85 f3 c0 52 bb aa d0 a9 a9 8f c8 77 15 15 38 d6

ef 98 fd 35 d6 9a 2b 18 43 eb 4f dd 12 eb a3 56

cipher suites

SSL_RSA_WITH_RC4_128_MD5

SSL_RSA_WITH_RC4_128_SHA

SSL_RSA_WITH_3DES_EDE_CBC_SHA

SSL_RSA_WITH_DES_CBC_SHA

SSL_RSA_EXPORT1024_WITH_RC4_56_SHA

SSL_RSA_EXPORT1024_WITH_DES_CBC_SHA

SSL_RSA_EXPORT_WITH_RC4_40_MD5

SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5

SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA

SSL_DHE_DSS_WITH_DES_CBC_SHA

SSL_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA

compression methods

NULL

12 0.9663 (0.5115) S>CV3.0 (74) Handshake

ServerHello

Version 3.0

random[32]=

45 31 97 f6 ad e2 f6 86 ef 2f fc 1c 3e ba 53 10

4e cc ac bc 54 79 c8 b3 13 92 0d 80 e1 50 43 ba

session_id[32]=

85 f3 c0 52 bb aa d0 a9 a9 8f c8 77 15 15 38 d6

ef 98 fd 35 d6 9a 2b 18 43 eb 4f dd 12 eb a3 56

cipherSuite SSL_RSA_WITH_RC4_128_MD5

compressionMethod NULL

13 0.9663 (0.0000) S>CV3.0 (1) ChangeCipherSpec

14 0.9663 (0.0000) S>CV3.0 (56) Handshake

15 0.9666 (0.0003) C>SV3.0 (1) ChangeCipherSpec

16 0.9666 (0.0000) C>SV3.0 (56) Handshake

17 0.9669 (0.0003) C>SV3.0 (634) application_data

Gmail 回复的加密套件是

SSL_RSA_WITH_RC4_128_MD5。

5.4.4 嵌入 SSL 的 iscribe 连接 Gmail 服务器

用嵌入 OpenSSL 库的 iscribe 连接 Gmail 服务端之后的跟踪结果如下：

New TCP connection #1: leicq（1408） <-> 64.233.167.109（995）

1 1 0.4732 （0.4732） C>S SSLv2 compatible client hello

Version 3.1

cipher suites

TLS_RSA_WITH_3DES_EDE_CBC_SHA

SSL2_CK_3DES

1 2 0.9513 （0.4780） S>CV3.1（74） Handshake

ServerHello

Version 3.1

random[32]=

45 1b c8 32 b9 39 e8 ad 46 4e d1 67 5c 65 d6 d4

ce 55 91 ed 60 b9 63 ec 15 d9 fe a8 6e 1c a2 f8

session_id[32]=

6f 18 e6 9e a3 9e 35 a8 5f d1 c7 40 6e 56 51 56

73 d0 ae b7 fd ba e9 1a 55 88 34 4b e8 bd cd 55

cipherSuite TLS_RSA_WITH_3DES_EDE_CBC_SHA

compressionMethod NULL

1 3 0.9513 （0.0000） S>CV3.1（747） Handshake

Certificate

certificate[737]=

30 82 02 dd 30 82 02 46 a0 03 02 01 02 02 03 05

（具体数据略）

1 4 0.9513 （0.0000） S>CV3.1（4） Handshake

ServerHelloDone

1 5 0.9700 （0.0187） C>SV3.1（134） Handshake

ClientKeyExchange

EncryptedPreMasterSecret[128]=

35 1d c2 48 b8 82 ff 26 cd 3e 92 a7 46 d9 68 e5

84 89 93 f9 2f aa c9 fe 3b 64 ee af f9 37 bf 13

4c b8 5d e9 1f 3e fa 49 dc fc 90 15 13 f3 71 8f

76 37 8f b7 2e 0c 8b 44 76 71 b8 18 a7 96 58 73

6f 86 ce c5 76 05 7a d9 a3 9f 6c ba 35 5a f8 5b

f6 95 7c 15 47 56 51 6e 91 73 23 e1 33 0c 77 99

9e 91 47 28 7f 0a 89 59 a1 88 8e f9 b9 9b e2 74

10 3a af 91 7c 8e 72 6a 57 f4 7d 8f 5d 09 8d 95

16 0.9700 (0.0000) C>SV3.1 (1) ChangeCipherSpec

17 0.9700 (0.0000) C>SV3.1 (40) Handshake

18 1.4472 (0.4771) S>CV3.1 (1) ChangeCipherSpec

19 1.4472 (0.0000) S>CV3.1 (40) Handshake

1 70 22.3994 (5.3502) C>SV3.1 (24) application_data

5.5 测试结果分析

OpenSSL 源代码的加密采用接口的方法，对外提供简单的 API，但其实现 API 的加密算法由双方磋商后的加密套件中的具体算法来执行。

SSL 握手双方如何筛选加密套件，在 SSL 协议中没有规定。OpenSSL 有其自身的加密套件定义规则。本测试验证了其输出结果符合其定义规则。

SSL 客户端与服务端磋商的加密套件不同，其握手过程状态可能不同。用 IE 连接 Gmail 服务器和有嵌入 OpenSSL 库的 iscribe 两种客户端连同一服务器端可以看出。用 IE 客户端连接 Gmail 时，没有 Client Key Exchange 报文。

通过比对 Gmail 服务器与 iscrbe 和 IE 连接测试，表明 IE 缺省的加密套件算法及强度的安全性不高。

5.6 本章小结

本章通过用嵌入 OpenSSL 库的 iscribe 邮件软件和 IE 客户端，用这两种不同的客户端分别连接同一服务端的测试的方式，验证了 OpenSSL 的接口实现方式；

OpenSSL 客户端加密套件的筛选结果是符合 OpenSSL 的加密套件筛选规则。加密套件不同，其握手过程也可能不相同。比对这两种客户端的加密套件，结果表明了 IE 缺省的加密套件的加密算法及强度的安全级别不高。

结束语

本人在工作期间，参与过某地区的证券交易所的一些系统开发。因证券交易安全要求极高。在系统开发中运用了数字证书和 SSL 技术。结合研究生期间开设的网络与信息安全的相关课程，考虑到 OpenSSL 源代码的开放性和它在许多工程中的实际成熟应用的特性，鉴于目前国内对于 OpenSSL 源代码的分析与研究的相关文献极少。本人从 OpenSSL 源代码作为切入点，来深入分析 OpenSSL 实现 SSL 协议方法与技术、设计思想等。分析并研究 OpenSSL 源代码，目的不是抄袭别人的技术。重要的是学习别人的设计理念，实现方式与技巧，为今后信息安全工作奠定基础。

OpenSSL 源代码内容非常丰富。因时间及精力有限。因人只涉及了 SSL 的握手部分的源代码。分析了 OpenSSL 的代码模块结构，支持的加密算法。OpenSSL 在实现方面用到的方法与技巧。包括采用了面向对象的思想；将网络及 IO 操作封装成 BIO；将加密算法及证书操作算法等封装成 EVP。用实际的例子描述了 OpenSSL 的客户端与服务端通讯的完整过程。

通过跟踪源代码运行过程，分析了 SSL 的握手过程和应用数据加密的完整过程。SSL 握手过程的加密套件的筛选过程；客户端与服务端通讯过程；证书链及私钥文件如何加载；握手过程的部分消息等。分析后发现 SSL 的握手协议的 Client hello、Server hello、Client Key Exchange、Server certificate 消息有一点不足。并指出了不足的原因。

用 OpenSSL 源代码编译之后的 Dll 库，嵌入到 iscribe 软件，连接到 Gmail 邮件服务器，来验证 OpenSSL 的 API 接口的实现符合分析结果；验证其握手时的加密套件的磋商结果符合自定义的加密套件规则；验证了加密套件不同，SSL 握手过程可能不同。通过比对 iscribe 与 IE 两种不同的客户端都使用 SSL 方式连接同一 Gmail 服务端，从握手的加密套件磋商结果中的加密算法，结果表明 IE 虽然利用了 SSL，但其缺省的加密套件的加密算法及强度安全级别并不高。

致 谢

三年的研究生生活是我一生中永远难忘的经历，在此期间，在敬爱的导师覃中平教授的精心指导、亲爱的同学们给予的帮助和带来的快乐，都在我心里留下了深深的印记。

这三年的学习、科研、工作中，覃老师严谨的治学态度，务实、勤勉、奋进的工作作风都给了我许多有益的影响。我不仅学到了知识、技术，更重要的是领悟了许多做人的道理，这是覃老师尽心指点的功劳。在论文撰写的紧张过程中，是覃导师的严格要求与精心指导使得本文得以顺利完成。在此，谨向敬爱的导师表示衷心的感谢。

感谢我的父母和家人，他们殷切的期望将永远激励我不断向前，他们无尽的关怀和深沉的爱是我今生最大的财富。

最后感谢各位专家评委对我的论文的悉心批评和指正！

参考文献

[1] A. O. Freier, P. Karlton, P. C. Kocher, The SSL Protocol Version 3. 0, USA: Network Working Group, 1996: 1-43

[2] 谭晓青. 利用 OpenSSL 建立 PKI 数字证书系统. 科学技术与工程, 2005, 5(20): 1553-1557

[3] D. Wagner, B. Schneier. Analysis of the ssl 3. 0 protocol. The Second USENIX Workshop on Electronic Commerce Proceedings, USENIX Press, 1996, 11: 29-40.

[4] D. C. BAKER, S. LAMBERT. Multiparabolic ionospheric model for SSL application. ELECTRONICS LETTERS, 1998, 24(7): 425-426

[5] K. Murphy. Traveling Through the OpenSSL Door. SANS Institute 2003; 1-34

[6] 李建设, 吴庆波. 基于 OpenSSL 的 VNC 安全性研究及实现. 微计算机信息, 2005, 21(3): 3-4

[7] 霍英. OpenSSL 在网络安全传输中的应用研究. 企业技术开发, 2006, 25(8): 2-3

[8] W. Chou. Accelerating secure transactions. IT Professional, 2002, 2: 37-43

[9] Nachiketh, Potlapally, S Ravi. A Study of the Energy Consumption Characteristics of Cryptographic Algorithms and Security Protocols. IEEE TRANSACTIONS ON MOBILE COMPUTING, 2006, 5(2): 128-132

[10] 关振胜. 公钥基础设施 PKI 与认证机构 CA. 北京: 电子工业出版社, 2002;21-24

[11] 徐静, 常朝稳. SSL 协议的安全性分析. 微计算机信息, 2006, 22(3): 3-4

[12] 苏成, 殷兆麟. SSL 协议的安全性分析与应用. 现代计算机, 2002, 6: 29-31

[13] 郭中华, 徐飞. 基于 SSL 协议的 SET 协议模拟实现. 计算机工程与设计, 2003, 24(9);80-81

[14] 马英杰, 肖丽萍, 何文才等. SSL 协议分析及其在 Web 服务应用中的改进. 微处理机, 2005, 6: 31-33

[15] G. Apostolopoulos, V. Peris, D. Saha. Transport Layer Security: How much does it really cost. 1999 IEEE, 0-7803-541 7-6/99: 718-721

[16] William, Stallings 著, 刘玉珍译. 密码编码学与网络安全-原理与实践(第三版). 北京: 电子工业出版社, 2004; 115-141

[17] Eric, Rescoria 著, 崔凯译. SSL 与 TLS Designing and Building Secure Systems. 中国电力出版社, 2002 : 43-47

[18] 张峰, 王小妮, 杨根兴. 采用 SSL 保障系统安全的一种方法. 北京机械工业学院学报, 2001, 16(3): 38-39

[19] 赵环宇. 用 OpenSSL 实现企业内部网安全通信模型. 网络安全技术与应用, 2006, 2: 77-78

[20] K. Kant, Ravishankar. Architectural Impact of Secure Socket Layer on Internet Servers. 2000 IEEE, 0-7695-0801-4: 7-8

[21] 张世永. 网络安全原理与应用[M]. 北京: 科学出版社, 2003:78-80

[22] 罗建超, 俸皓, 罗蕾. 一种嵌入式 SSL 协议栈的设计. 计算机工程, 2006, 32(16): 283-284

[23] 卿斯汉. 安全协议. 北京: 清华大学出版社, 2005:72-75

[24] 谭良. OpenSSL Engine 安全平台下的 Engine 对象分析. 四川师范大学学报(自然科学版), 2004, 27(4): 1-2

[25] 何志平, 康荣保, 罗慧. 基于 CSP 的与硬件无关的 OpenSSL Engine 分析与实现. 信息安全与通信保密, 2006, 7: 90-91

[26] R. Stinson 著, 冯登国译. 密码学原理与实践. 北京: 电子工业出版社, 2003:47-52

[27] 钟源, 崔树鹏, 容晓峰. 基于 OpenSSL 的密码支撑平台的研究与开发. 计算机与现代化, 2004, 8: 2-5

[28] 卢开澄. 计算机密码学-计算机网络中的数据保密与安全(第 3 版). 北京: 清华大学出版社, 2003: 61-73

[29] 邵维忠, 杨芙清. 面向对象的系统分析. 北京: 清华大学出版社. 1998:81-99

[30] 陈国辉, 施伟. OpenSSL 在电子商务安全中的应用. 微计算机信息, 2004, 20(5): 2-3

[31] Eric, S. Roberts. Programming Abstractions in C: A Second Course in computer Science. Pearson Education Asia Ltd, 2004 :151-172

[32] 邓晓军, 陈怀义. 基于 SSLTLS 的安全应用中独立端口和磋商升级的研究与实现. 计算机工程与科学, 2006, 28(4): 22-23

[33] 虞俊杰, 陈启祥. 利用 SSL 对 IP 数据包进行加密. 微处理机, 2006, 1: 32

[34] 袁素春, 王育民, 李纲. OpenSSL 程序设计. 计算机安全, 2005, 6: 17-18

[35] 王娟, 邱宏茂, 盖磊等. SSL 及使用 OpenSSL 实现证书的签发和管理. 微机发展, 2004, 14(10): 2-3

[36] 孟彦, 侯整风. 基于 SSLTLS 的安全文件传输系统. 计算机技术与发展, 2006, 16(5): 2-3

[37] 陈敬佳. SSLTLS 的安全性分析与实现. 武汉理工大学学报信息与管理工程版, 2005, 27(5): 2-3

[38] 章晓明, 王则林, 陆建德. 基于 OpenSSL 的嵌入式网络安全通信设计与实现. 计算机技术与发展, 2006, 16(1): 219-220

[39] P. Chandra, M. Messier, J. Viega. Network Security with OpenSSL. California: O'Reilly. 2002: 93-121

[40] Todd, Heberlein. A Network Securit Monitor. Proceedings of the IEEE Computer Society Symposium, 1990:293-303

OpenSSL分析与测试

作者： 雷长青
学位授予单位： 华中科技大学

相似文献(10条)

1.学位论文 王智超 基于数据加密的网络通信系统的研究 2006

随着网络通信技术和Internet 的联系日益增强，与网络安全相关的问题：如网络上传输的信息被截取、篡改、重发等对网络应用的进一步推广构成巨大威胁，基于数据加密的网络通信系统便是在这种背景下应运而生的。

课题首先分析了网络安全的现状，并指出了其面临的主要问题，然后提出了综合利用数据加密技术、认证技术和远程动态口令鉴别技术的解决方案。课题所设计的网络通信系统分为：网络通信模块、加密通信模块和远程动态口令认证，从而很好的解决了信息的安全通信问题。

在网络通信模块的设计中，分析了网络通信协议TCP/IP，研究了Windows Socket 程序的设计与实现方法，采用了面向连接协议TCP 的WinSock 设计，最终实现网络中两台主机的通信。

在加密通信模块的设计中，提出了一种新型的混合加密机制，即明文用加密强度大且速度快的排列码算法加密，密钥用RSA算法进行加密，用SHA算法产生数字摘要来实现认证，这样既解决了密钥管理的难题，又保证了数据的完整性。

在远程动态口令认证的设计中，分析了传统口令认证的不足，采用了一种新的方案，实现了用户登陆的口令是动态变化的，以抵御重传攻击。

最后对整个系统进行了相关总结和评价，同时指出了课题中的不足和下一步工作。

2.期刊论文 黄志清.Huang Zhiqing 网络安全中的数据加密技术研究 -微型电脑应用2000,16(5)

数据加密技术是实现网络安全的关键技术之一.本文系统地讨论了对称式加密、公开密钥加密以及混合式加密三种数据加密技术以及链路加密和端对端加密两种网络数据加密方式.

3.学位论文 程时宏 基于网络安全的闸门命令单系统的设计与实现 2007

网络安全是信息安全中的重要研究内容之一，也是当前信息安全领域中的研究热点。保护网络传输过程中的数据不受偶然或恶意原因而遭到破坏、更改、泄露，是网络安全的主要内容之一。目前比较实用的方法是对网络中传输的数据进行加密，而数据加密要依赖于成熟的数据加密算法。

本文基于二滩公司已有的广域网平台，结合网络闸门命令单操作系统的要求，通过对RSA进行身份认证和DES加密技术、B/S和C/S体系结构、网络操作系统和网络数据库系统的分析对比，确定了适合实际情况的数据加密方式、系统体系结构、网络操作系统和网络数据库系统，在Delphi平台上设计实现了基于网络安全的闸门命令单系统。使用Delphi语言开发核心内容，即数据的加密和解密模块，与后台SOL Server进行数据交换模块等，HTML语言调用该核心部分来向浏览信息的客户提交网页界面。

本文还详细阐述了系统总体设计和详细设计，对系统数据库设计、身份认证、加密方式的实现进行了理论探讨。这些都为系统开发奠定了理论和实际的基础。整个系统也就具有良好的可操作性，能够适用于不同的应用环境。

系统运行结果证明，本文所设计的网络闸门命令单操作系统可以满足命令单发出、命令单反馈和命令单管理等方面的需要。制定的实施方案符合二滩公司流域梯极调度的现实情况，并通过实践证明是切实可行的，为流域梯极调度中信息化系统的继续建设打下了坚实的基础。

4.期刊论文 李红军.缪旭东 数据加密在网络安全中的应用 -微型机与应用2002,21(10)

数据加密的特点及网络加密的策略,讨论了网络加密技术存在的问题.

5.学位论文 张颖哲 基于Web Services技术的网络安全管理平台的设计与实现 2007

在当今这个信息化社会中，在网络应用的深入发展和技术进步的同时，非法访问、恶意攻击、病毒传播等网络安全威胁也越来越严重。为了保护网络系统安全，防火墙、IDS、防病毒、身份鉴别、数据加密、安全审计等安全设备在网络系统中得到了广泛应用。

但是，这些安全设备只能在特定方面发挥一定的作用，大部分功能单一，彼此之间没有有效的统一管理协调机制，不能互相支持、协同工作，从而使得各个安全设备的应用效能无法得到充分的发挥。为了对这些异构性的安全设备实施有效的统一管理，网络安全管理平台应运而生。

本论文主要对网络安全管理平台的架构进行研究，对实验室已有网络安全管理平台存在的不足进行改进，具体研究目标是构建一个通用的、简单的、安全的网络安全管理平台。文中提出了以面向服务的方式来管理各安全设备，通过面向服务屏蔽了各系统的异构性，实现了管理平台对被管安全设备跨平台的管理。各安全设备通过Web Services技术来提供服务，SOAP消息传送中绑定HTTP协议使用80端口，这使得设备与网络安全管理平台能穿越防火墙进行通信。为了确保SOAP消息传输的安全，通过应用WS-Security框架，在SOAP消息中插入XML签名、加密信息来实现，通过使用这些现有的安全技术，能够实现SOAP消息安全、可靠的传输。面向服务的架构(Service-Oriented Architecture，SOA)是当前技术的热点，它极具扩展性的特点为网络安全管理平台对各类安全设备进行管理提供了可能。

6.期刊论文 何明.HE Ming 论数据加密在网络安全中的应用方案 -光盘技术2008,""(12)

本文在明确数据加密技术在网络中作用的基础上,首先简要阐述了相关数据加密技术,接着重点研究了认证方式的具体应用方案,有效实现了用户和服务器间的相互认证,能显著增强网络系统的安全性.

7.学位论文 朱作付 基于DES和RSA算法的数据加密传输系统的设计和实现 2005

网络安全是信息安全中的重要研究内容之一，也是当前信息安全领域中的研究热点。DES和RSA是两种应用非常广泛和成熟的数据加密算法。本文通过对DES和RSA加密技术进行分析，在VC++平台上设计实现了网络数据加解密系统。即建立一个安全的数据传输系统，包括信息交换格式、交换协议和加解密算法，从而保证网络数据的安全传输。系统的整体设计建立于现在流行的分布式系统的基础上，即保证各个Agent与中央控制器进行安全数据传输，具有良好的可操作性，能够适用于不同的应用环境。

和传统的一些平台相比，VC++平台能够使系统具有较高的效率和可用性。在系统的具体实现上，充分利用了VC++平台的特点与优势，大大提高了系统的的可靠性。

8.学位论文 钟颖 无线网络安全层数据加密技术的研究与应用 2002

该文从Internet的安全层协议出发,引出了在此基础上进行了简化的无线网络安全层,详细介绍了无线网络安全层的结构模型、功能.无线网络安全层的数据加密技术是其它安全技术的基础,为了便于读者理解,作者介绍了加密技术的基本概念.分别介绍了目前网络安全层协议中使用到的几种数据加密技术,其中详细介绍了数据加密标准DES算法,分析其安全性,指出其存在的不安全因素,提出应当用更新更安全的算法来代替的思想.针对DES算法的问题所在,选择高级加密标准AES这一被称为21世纪的数据加密标准来代替DES.作者在接下来便对AES评选中获胜的来自比利时的Rijndael算法进行了深入的探讨,介绍了算法的数学基础、设计原理及结构,分析了算法的优缺点,并将该算法与DES算法进行了比较.在进行了理论讨论后,作者给出了AES算法在Linux平台PC机上的标准实现和优化实现,并将AES算法、优化AES算法与DES算法进行比较,得出优化AES算法综合性能优于其它两种算法的结论.根据这一结论,将优化AES算法应用在"无线智能终端"无线上网模型的数据加密中,结果显示,该算法的应用是有效的.最后对全文成果进行了总结,并给出了进一步研究的讨论.

9.期刊论文 黄世权 网络安全及其基本解决方案 -科技情报开发与经济2004,14(12)

简述了网络安全的重要性,分析了威胁网络安全的各种因素,提出了保障网络安全的相关技术对策,包括数据加密、数字签名、访问控制、防火墙、身份鉴别等.

10.学位论文 徐棣 网络安全分析与安全策略研究 2006

近十多年来，以Intemet为代表的计算机网络应用在我国得到了飞速发展。作为Intemet的协议基础的TCP/IP协议族，具有开放性和方便性等优点。同时TCP/IP在设计上的安全性缺陷，在互联网的高速发展的今天，对网络安全与网络管理提出了新的要求。

网络安全是为了保障网络服务的可用性和网络信息的完整性及可靠性不被蓄意或偶然地破坏而采用的一切措施。网络安全研究是当今网络研究的一个热点，也是当今网络应用中最敏感的问题之一。本文主要内容围绕网络安全分析与防范策略的研究展开。论文介绍了当前互联网的发展状况，以及当前互联网对网络安全与管理提出的要求，并概括了当前网络安全分析与防范策略研究的主要内容。TCP/IP协议族作为网络安全的理论基础，是研究网络安全必须掌握的内容之一。论文阐述了TCP/IP协议的层次划分，以及该协议族中用于网络管理的简单网络管理协议SNMP。论文围绕网络安全的主要技术进行了深入探讨，内容包括数据加密、网络侦听与反侦听等技术，并结合实例进行了详细的分析。网络安全管理技术也是网络安全研究的重要内容。本文提出了法律手段、技术防范和安全教育相结合的网络安全管理的基本思想，然后对企业网络防治病毒进行了研究，并给出了三套实用的集成化防病毒方案。

随着网络技术的发展和网络应用的更加普及，新的网络安全问题又会出现。目前尚未出现任何一劳永逸的网络安全解决方案。研究网络安全分析和网络安全策略管理和网络安全技术，就是用尽可能少的代价，把风险降到最低。