

第3章 加密：足够吗

不要对私人使用加密手段耿耿于怀！政府关心的是国家安全，加密应当广泛应用于保护电子商务的传输。

电子商务的不断发展将数据加密推到前台。越来越多的公司和网络服务商需要在 Internet、商务、金融业务传输时保护他们的私人秘密。但是这会使政府感到为难，因为加密是一柄双刃剑，使得法律执行机构无法监视私人活动，如果一个有效的加密算法落入别有用心的人手中，将使犯罪分子可以肆无忌惮地犯罪而不被发现。

加密的主要工具——计算机随处可见。二次世界大战后，世界各国政府都在努力控制数据加密的使用（不信去问 Phil Zimmermann）。我们现在不再需要 Colossus，即二战时为了跟踪、破译德军密码而制造的那台机器。我 14 岁的儿子现在已经在家中使用 Pentium 计算机连入 Internet，并用 CodeDrag 加密他的文件了！

注意 关于 Phil Zimmermann

他是 Pretty Good Privacy (PGP) 的设计者。PGP 是他在 Internet 上做的一个加密软件，就是由于这个 PGP 使他受到美国政府的指控。如想要了解更多的细节，请访问以下站点：

<http://web.its.smu.edu/~dmcnickl/miscell/warnzimm.html>。

提示 什么是 CodeDrag？

如图 3-1 所示，CodeDrag 是一种非常快的加密软件。它是一种 C 语言实现的快速 DES 软件。它是由奥地利的 Linz 大学设计的。CodeDrag 完全嵌入 Windows 桌面，被用以证明新的 Windows 95/98 外壳的可行性。为了得到更多的信息，可以与开发小组取得联系：dragon@fim.uni-linz.ac.at 或访问他们的站点（可以从那里下载一份 CodeDrag 的拷贝）：

<http://www.fim.uni-linz.ac.at/codeddrag/codeddrag.htm>。

从 1979 年起，就如同对待喷气式战斗机及核导弹一样，美国国家安全局 (NSA) 将任何形式的密码都视为武器类。但是那些像 Zimmermann 的人考虑到私人和普通用户的利益向政府所垄断的密码发起了挑战。70 年代，来自斯坦福研究院的 Whitfield Diffie 提出闻名世界的公开密钥密码学。我们在本章将给以详细论述。

Diffie 的发明确实是在密码界、特别是在政府中引发了一场革命。当政府的秘密部门仍在使用单密钥体制的时候，他设想了一个通过双密钥体制解决单密钥体制中的发送方、收收方都必须使用同一密钥的问题，这种双密钥体制使得加密数据变得更简单。

后来在 1977 年，来自麻省理工学院的三位科学家成立了一家公司即 RSA Data Security 公司，推出了第一种公开密钥加密软件并在美国获得专利。

1991 年，一名叫做 Zimmermann 的计算机程序员发布了他的 Pretty Good Privacy (PGP) 加密软件，并将其在 Internet 上免费发布，使得它在全世界范围都能够得到。他的这一行为不但引起了政府的注意，受到了政府的指控，甚至 RSA Data Security 公司也抱怨 PGP，因为它威

胁到该公司的商业利益。

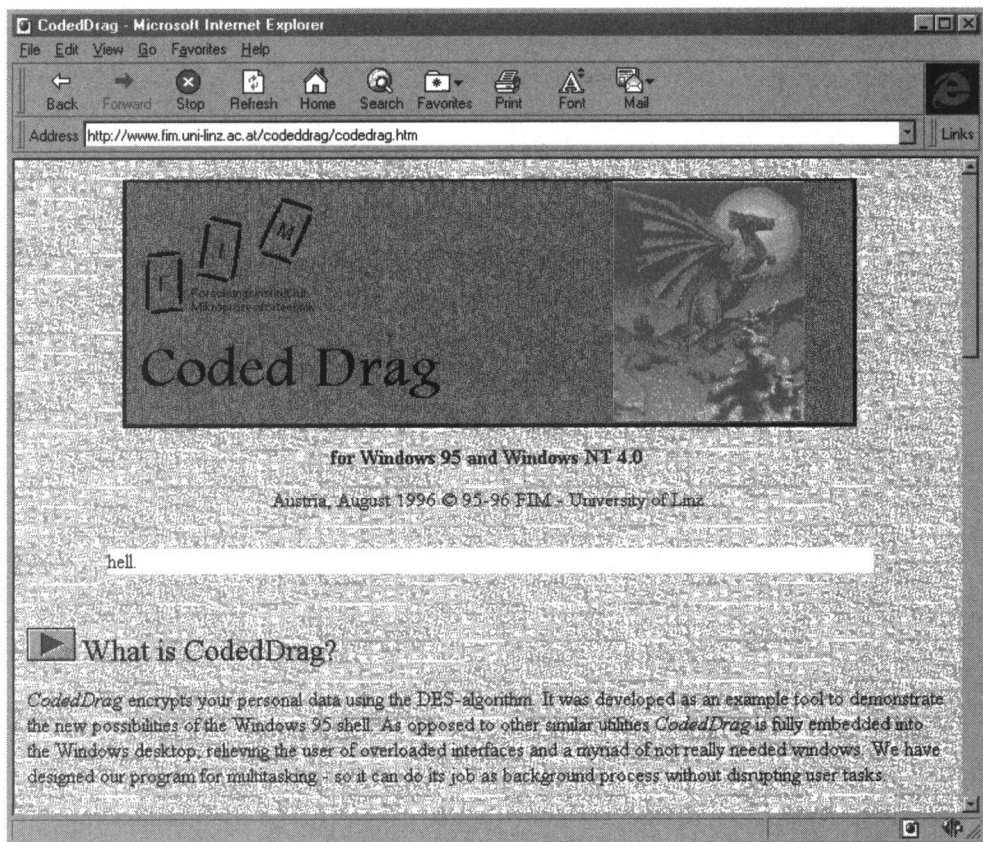


图3-1 奥地利Linz大学设计开发的加密工具CodeDrag

现在，连商业软件公司也开发了它们自己的加密产品。如 Netscape 开发了自己的加密产品，并在因特网上免费发布。Netscape 的 Secure Sockets Layer (SSL) 加密方案，使用 56 位的密钥以增加数据的安全性。Microsoft 公司也提供了它的加密工具，即 Private Communications Technology (PCT) 协议。

在前面讨论过的两章中，涉及到计算机网络的安全问题，因为随着网络数目的增加和网络规模的扩大，它变得越来越重要。因特网已逐渐成为公司所需要保护的网路的外延。不久前，企业内部网还是新事物，但一年多之后，我们已经开始谈论并探索外部网了。随着资源共享及信息世界化(当然也包括电脑空间！)变得越来越容易，保护信息、资源不被非法使用变得尤为重要。

现在你应该意识到构建一个百分之百安全的网络是不可能的。只有能够访问的信息才是有价值的。访问和安全一直是一对矛盾，它应由管理者作出的策略来决定。

一个好的安全体制包括对安全策略的深入计划、分析。安全策略包括访问控制以及认证机制。这些安全策略以及过程可以从最简单的口令策略到复杂的加密方案。这里假设你已经实现了口令策略(不会还没有吧？)，本章将讨论各个级别、各种形式的加密方案，以及它们何时能用或是否能用。

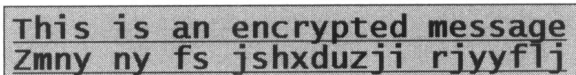
3.1 引言

把公司的信息加密是一种重要的安全手段，并为你提供了网络上最基本的安全服务：认证交换。其他方案，如数字签名 (Digital Signature) 和数据认证都运用了加密手段。

3.2 对称密钥加密 (私有密钥)

现在市场上提供好几种加密技术，使用若干种不同的算法。最基本的有两种：一种是使用密钥，另一种算法本身不依赖于密钥。

不使用密钥的加密技术十分简单，通过替换或编码以达到加密目的。例如，可以通过给每一个字母的 ASCII 值加上一个数来加密一组英文信息，参见图 3-2。看起来它似乎已经很安全了，但实际上这种算法并不是你想像中的那么安全。事实上它们很容易被破译。一旦知道了加密算法，就能够破译加密过的信息。



This is an encrypted message
Zmny ny fs jshxdudzji rjyyflj

图3-2 加密信息示例

一些更安全的加密算法是将数据与一种密钥配合使用。两种主要的加密算法是私有密钥加密和公开密钥加密。在以后的章节中将会具体讨论。私有密钥也被称为单钥，秘密密钥或对称密钥。公开密钥也称为非对称密钥。

就私有密钥加密算法而言，算法中只存在一个密钥。同一个密钥被用于加密、解密过程。为了保证安全，你必须保护好这个密钥而且确保只有你一个人知道。本章中将要讨论的 Kerberos 认证协议就是使用这样的私有密钥算法。

私有密钥加密的另一特点，就是其使用的密钥长度一般都比较短，这使得它的算法实现比非对称加密要快，也要容易一些。

私有密钥加密的一个主要缺陷是当密钥分配给每个需要的人，尤其是密钥分配本身就必须保密。另外，如果暴露或损坏密钥，那么这就等于暴露或损坏了用它加密过的信息。因此，很有必要经常更改密钥。

如果只有私有密钥方案，我建议将其与数字签名一同使用。因为它们更加有效也更加安全。

3.2.1 数据加密标准

数据加密标准 (DES) 是使用最为普遍的私有密钥算法。DES 算法由 IBM 发明，并于 1976 年成为美国政府标准。这一著名算法在金融和政府工程中得到大量运用。上文提到的 Kerberos，就是使用 DES 算法加密消息，并产生用于不同情况下的私有密钥。

DES 算法十分快。根据 RSA 实验室提供的数据，当 DES 完全由软件实现时，它至少比 RSA 算法快 100 倍。如果由硬件实现，DES 比 RSA 快 1000 甚至 10 000 倍。因为 DES 使用 S 盒运算，它使用简单的表查找功能，而 RSA 却是建立在非常大的整数运算上。

DES 使用相同的加密、解密算法。密钥可以是任意一个 64 位的数。算法的工作方式决定

了只有56位有效。NIST授权DES成为美国政府的加密标准，但只适用于加密“绝密级以下信息”。尽管DES被认为十分安全，但确实存在一种方法可以攻破它。

通过穷尽搜索密钥空间，提供总共 2^{56} (大约 7.2×10^{16}) 个可能的密钥。如果每秒能检测一百万个密钥的话，只需2000年就够了。祝你好运！

直到最近，DES还未被攻破，因此也被视为是安全的。但有一组 Internet 用户，用了4个多月时间分工合作解决了 RSA DES 挑战(见图3-3)并最终攻破了这一算法。该小组在检验大约 18×10^{15} 个密钥后找到了正确的密钥，并恢复了明文：

strong cryptography makes the world a safer place.

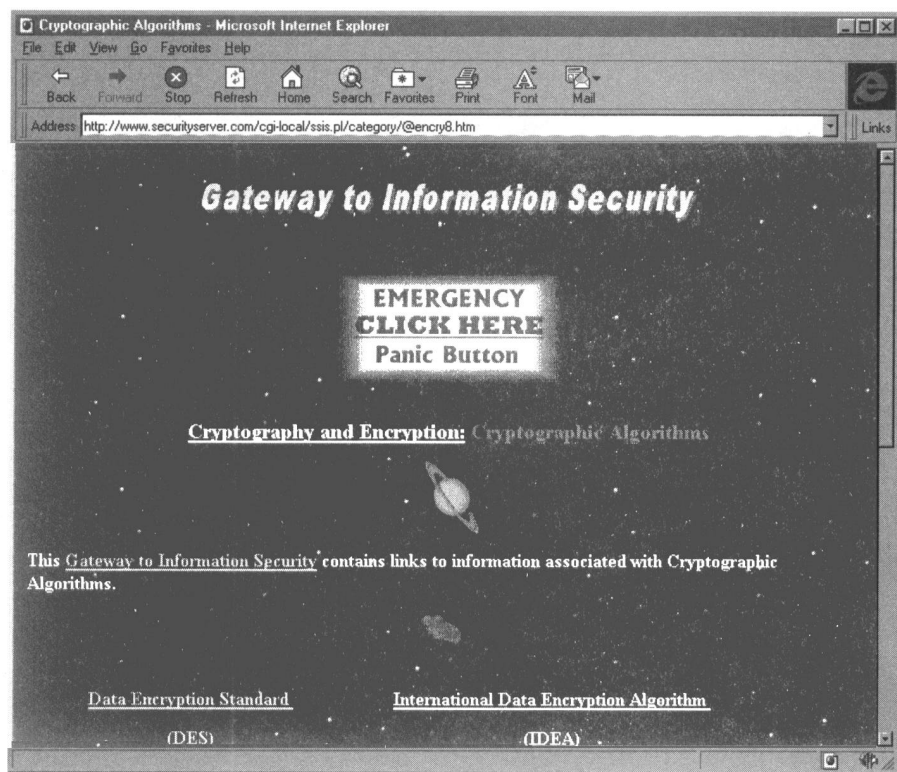


图3-3 悬赏\$10000挑战DES的站点

注意 尽管DES在美国以外已得到普遍实现。美国政府禁止出口任何包含有DES实现的硬件及软件产品。美国出口商必须遵循这一政策。

该小组采用一种称为“强行攻击”(brute-force)的技术，即所有参加这一挑战的计算机搜索所有可能的密钥，一共有超过 72 057 594 037 927 936 个密钥。当1997年6月，把这一正确密钥报告给 RSA Data Security 公司时，该小组已经搜索了大约所有可能密钥的 25%。在该小组工作的巅峰状态时刻，每秒有 70 亿个密钥被检测。图3-4是 DESCHALL 站点的网页界面。网址是：<http://www.frii.com/~rcv/deschall.htm>。

尽管DES被攻破，但是它在20年之内都是一个安全算法。强行攻击算法攻击DES是破译一个密码算法的通用方法。尽管必须搜索所有 256 个密钥以加密一组明文并将结果与相应密文

相比较,但通过不同的加密分析,我们可以将密钥数量降至 2^{47} 个。这仍是一个很大的工程。如果DES使用长度超过56位的密钥,那么破译它的可能性几乎为0。

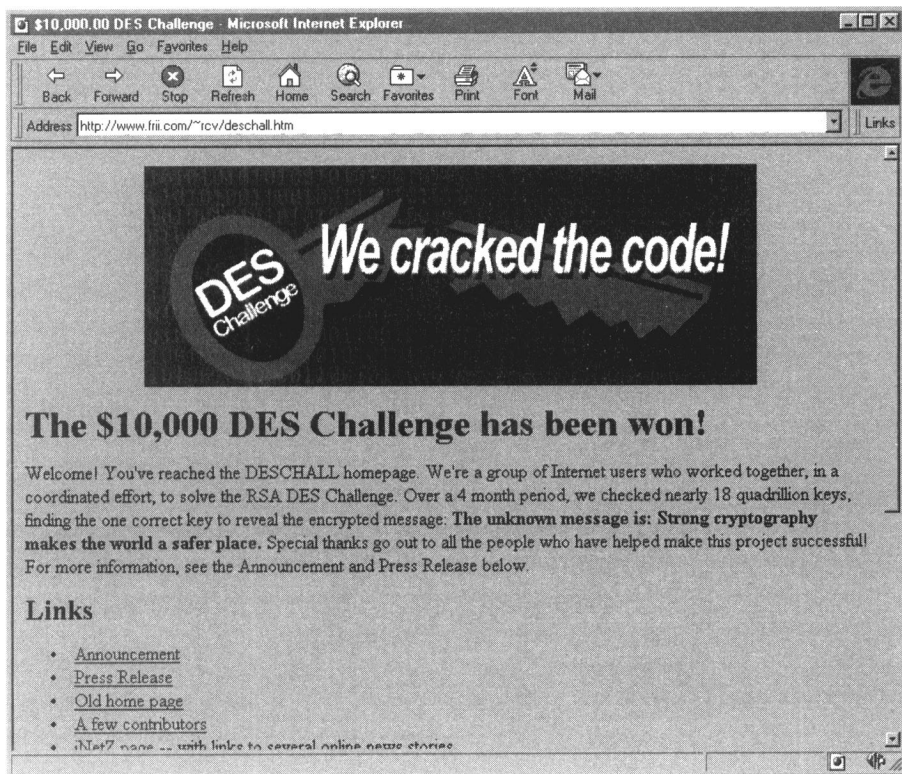


图3-4 破解DES算法的DESHALL站点

3.2.2 国际数据加密算法

国际数据加密算法 (IDEA)是行之有效的加密算法中最好、最安全的一种。由瑞士联邦科学技术学院(SFT)的Xuejia Lai 和James Massey提出。IDEA使用3个64位的块,以进一步防范加密分析过程。IDEA使用了密码反馈操作,使得算法强度更高。在这种模式下,密文输出也被用来作为加密运算的输入。

IDEA的另一个重要特点是它的密钥长度为 128位。正如我们见到的 DES,密钥越长,其保密性越高。当试图破译 IDEA时,它和DES一样没有泄露任何明文组成的信息。IDEA能够将1位的明文扩散到多个位的密文中去。以达到完全隐藏明文的统计结构。

IDEA确实存在最低要求。为得到强有力的密文,它在一个编码块中需要 64位的信息文本。如果加密大量的数据,它不成问题,但对于一个一个键盘字符敲击作为输入的情况它就不适用了。必须明确,IDEA是为有大量数据传输的FTP设计的。你可以想像一下在Telnet中它的效率是多么低。

Fauzan Mirza 编写了一个称为Tiny IDEA的安全文件加密程序。网址是:

<http://www.dcs.rhbnc.ac.uk/~fauzan/tinyidea.html>。图3-5所示为Tiny IDEA 的网站的面,可在那里下载该程序及该程序的简介和其他详细资料。

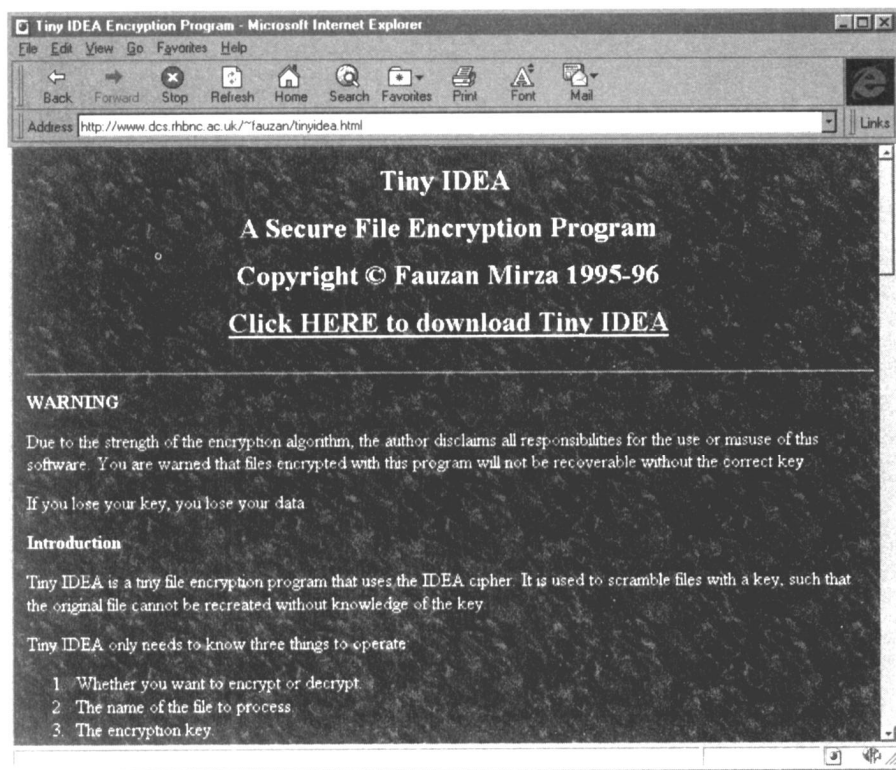


图3-5 Tiny IDEA 加密站点

3.2.3 CAST算法

CAST算法由Carlisle Adams 及Stafford Tavares开发。该算法使用64位的块长及64位的密钥。它使用六个8位输入32位输出的S盒。不要问我这些S-盒的结构，它们实在是太复杂了，已经超出了本书的范围。想得到更多此方面信息，我推荐由 Bruce Schneier编写，John Wiley出版社出版的“Applied Cryptography”(ISBN 0-471-11709-9)。它对想要深入研究密码学的人来说是一部力作。

CAST加密过程是将明文块分为两个子块，左子块和右子块。该算法有8圈，每圈一半明文经过“f”函数运算与某一密钥组合，然后将结果与另半部分异或，左子块形成新的右子块，原来的右子块变为左子块。经过8次这样的运算之后，这两部分的输出就是密文了。参见上文提到的Schneier书中第335的例子，可见这种运算十分简单。表3-1给出了“f”函数：

表3-1 CAST中将明文加密成密文所用的函数

- | |
|--|
| 1) 把32位输入分成4个8位组：a,b,c,d |
| 2) 将16位子密钥分成两个8位子密钥：e, f |
| 3) 将a通过S盒1, b通过S盒2, c通过S盒3, d通过S盒4, e通过S盒5, f通过S盒6进行处理 |
| 4) 将6个S盒的输出异或得到最终的32位输出 |

注意 什么是S盒？

S盒,或称为选择盒，是一组高度非线性函数。在DES中S盒看起来像一组表，它们是DES真正执行加密、解密运算的函数部分。因为DES变换的其他部分都是线性的。

图3-6是William&Mary学院DES S盒网站的界面。它由 Serge Hallyn 免费提供。网址是：<http://www.cs.wm.edu/~hallyn/des/sbox.html>。为了方便大家，我们在图3-7~图3-14一一给出了DES的8个S盒的屏幕显示。

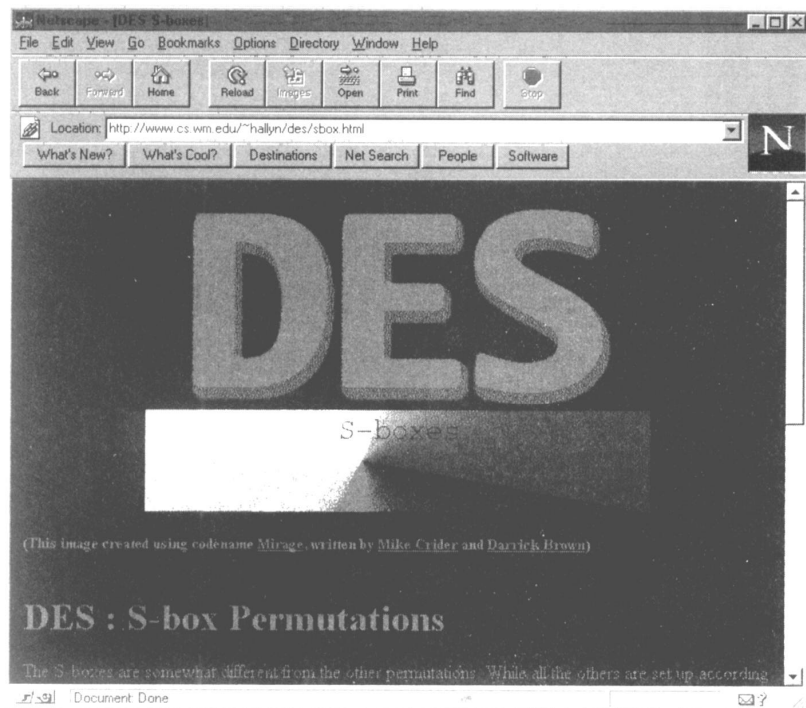
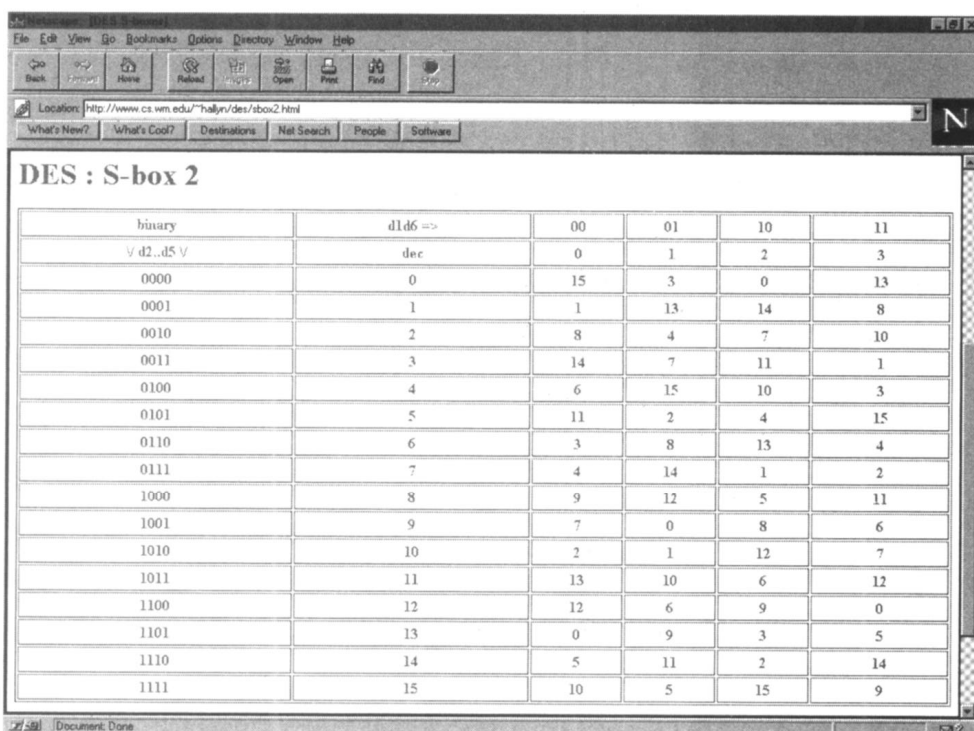


图3-6 William 和Mary学院的DES S盒站点

Binary	d1d6 =>	00	01	10	11
0000	0	14	0	4	15
0001	1	4	15	1	12
0010	2	13	7	14	8
0011	3	1	4	8	2
0100	4	2	14	13	4
0101	5	15	2	6	9
0110	6	11	13	2	1
0111	7	8	1	11	7
1000	8	3	10	15	5
1001	9	10	6	12	11
1010	10	6	12	9	3
1011	11	12	11	7	14
1100	12	5	9	3	10
1101	13	9	5	10	0
1110	14	0	3	5	6
1111	15	7	8	0	13

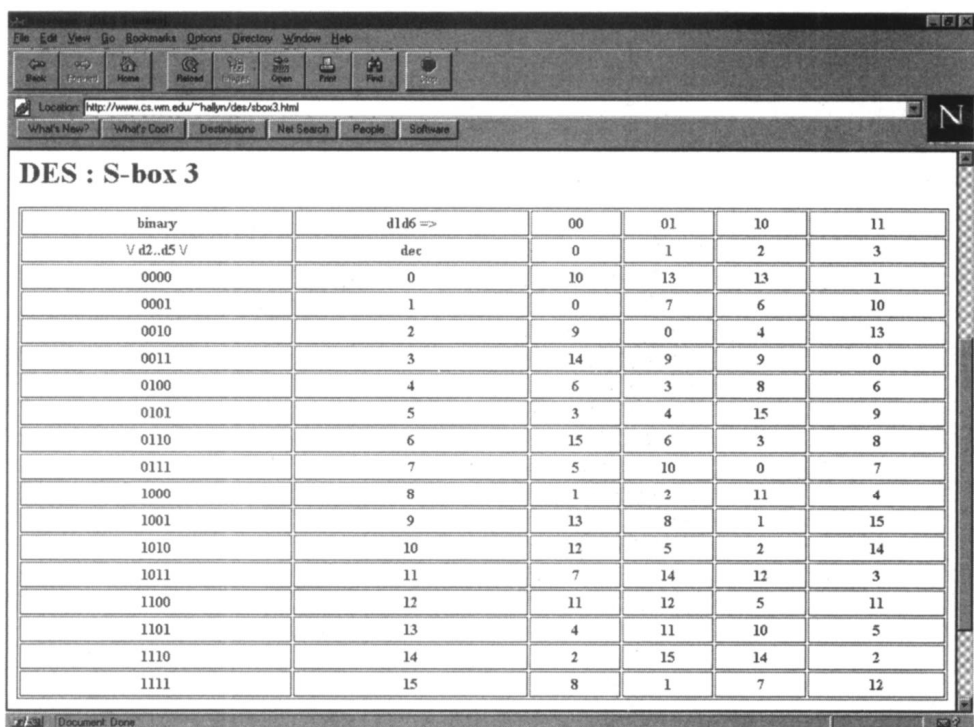
图3-7 DES 的第1个S盒



DES : S-box 2

binary	d1d6 =>	00	01	10	11
∨ d2...d5 ∨	dec	0	1	2	3
0000	0	15	3	0	13
0001	1	1	13	14	8
0010	2	8	4	7	10
0011	3	14	7	11	1
0100	4	6	15	10	3
0101	5	11	2	4	15
0110	6	3	8	13	4
0111	7	4	14	1	2
1000	8	9	12	5	11
1001	9	7	0	8	6
1010	10	2	1	12	7
1011	11	13	10	6	12
1100	12	12	6	9	0
1101	13	0	9	3	5
1110	14	5	11	2	14
1111	15	10	5	15	9

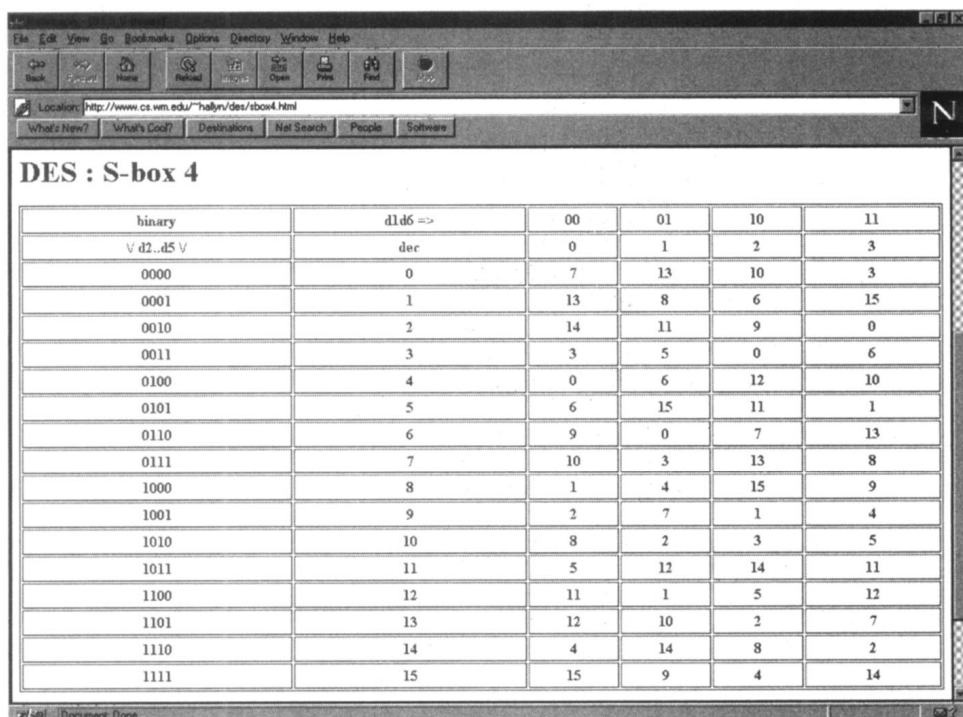
图3-8 DES 的第2个S盒



DES : S-box 3

binary	d1d6 =>	00	01	10	11
∨ d2...d5 ∨	dec	0	1	2	3
0000	0	10	13	13	1
0001	1	0	7	6	10
0010	2	9	0	4	13
0011	3	14	9	9	0
0100	4	6	3	8	6
0101	5	3	4	15	9
0110	6	15	6	3	8
0111	7	5	10	0	7
1000	8	1	2	11	4
1001	9	13	8	1	15
1010	10	12	5	2	14
1011	11	7	14	12	3
1100	12	11	12	5	11
1101	13	4	11	10	5
1110	14	2	15	14	2
1111	15	8	1	7	12

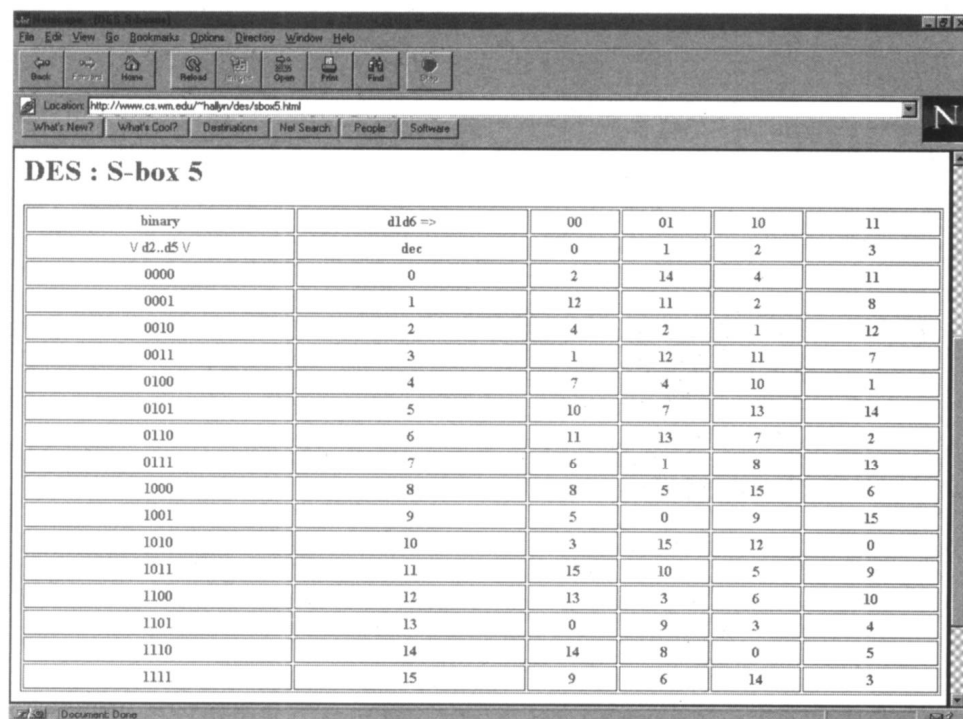
图3-9 DES的第3个S盒



DES : S-box 4

binary	d1d6 =>	00	01	10	11
∨ d2...d5 ∨	dec	0	1	2	3
0000	0	7	13	10	3
0001	1	13	8	6	15
0010	2	14	11	9	0
0011	3	3	5	0	6
0100	4	0	6	12	10
0101	5	6	15	11	1
0110	6	9	0	7	13
0111	7	10	3	13	8
1000	8	1	4	15	9
1001	9	2	7	1	4
1010	10	8	2	3	5
1011	11	5	12	14	11
1100	12	11	1	5	12
1101	13	12	10	2	7
1110	14	4	14	8	2
1111	15	15	9	4	14

图3-10 DES 的第4个S盒



DES : S-box 5

binary	d1d6 =>	00	01	10	11
∨ d2...d5 ∨	dec	0	1	2	3
0000	0	2	14	4	11
0001	1	12	11	2	8
0010	2	4	2	1	12
0011	3	1	12	11	7
0100	4	7	4	10	1
0101	5	10	7	13	14
0110	6	11	13	7	2
0111	7	6	1	8	13
1000	8	8	5	15	6
1001	9	5	0	9	15
1010	10	3	15	12	0
1011	11	15	10	5	9
1100	12	13	3	6	10
1101	13	0	9	3	4
1110	14	14	8	0	5
1111	15	9	6	14	3

图3-11 DES 的第5个S盒

DES : S-box 6

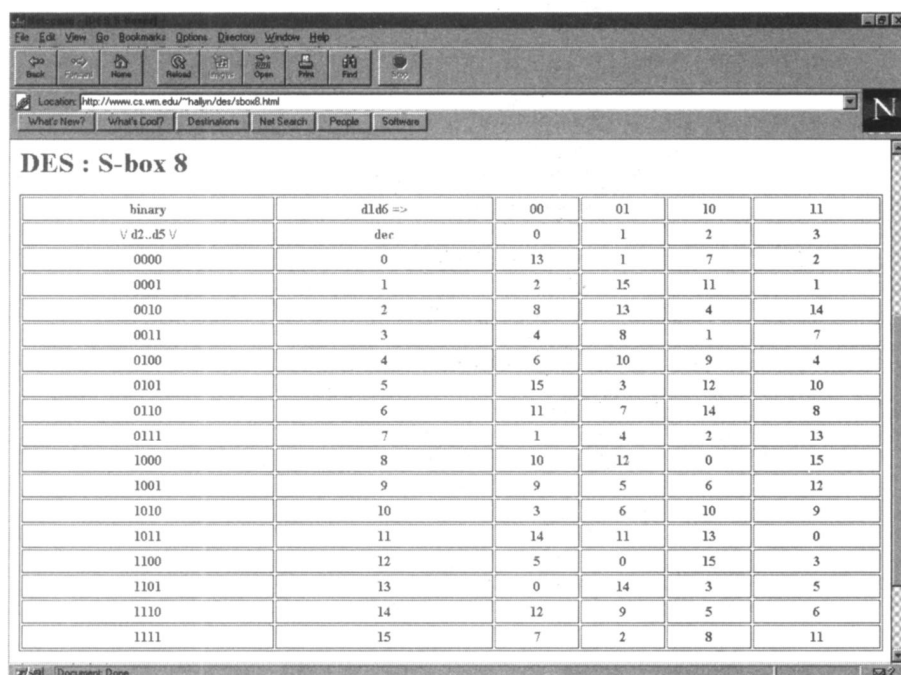
binary	d1d6 =>	00	01	10	11
√ d2...d5 √	dec	0	1	2	3
0000	0	12	10	9	4
0001	1	1	15	14	3
0010	2	10	4	15	2
0011	3	15	2	5	12
0100	4	9	7	2	9
0101	5	2	12	8	5
0110	6	6	9	12	15
0111	7	8	5	3	10
1000	8	0	6	7	11
1001	9	13	1	0	14
1010	10	3	13	4	1
1011	11	4	14	10	7
1100	12	14	0	1	6
1101	13	7	11	13	0
1110	14	5	3	11	8
1111	15	11	8	6	13

图3-12 DES 的第6个S盒

DES : S-box 7

binary	d1d6 =>	00	01	10	11
√ d2...d5 √	dec	0	1	2	3
0000	0	4	13	1	6
0001	1	11	0	4	11
0010	2	2	11	11	13
0011	3	14	7	13	8
0100	4	15	4	12	1
0101	5	0	9	3	4
0110	6	8	1	7	10
0111	7	13	10	14	7
1000	8	3	14	10	9
1001	9	12	3	15	5
1010	10	9	5	6	0
1011	11	7	12	8	15
1100	12	5	2	0	14
1101	13	10	15	5	2
1110	14	6	8	9	3
1111	15	1	6	2	12

图3-13 DES 的第7个S盒



The screenshot shows a web browser window with the address bar displaying "http://www.cs.wm.edu/~hahny/des/sbox8.html". The page title is "DES : S-box 8". The table below is the DES S-box 8, which maps 6-bit binary inputs to 4-bit binary outputs.

binary	d1d6 =>	00	01	10	11
0000	0	13	1	7	2
0001	1	2	15	11	1
0010	2	8	13	4	14
0011	3	4	8	1	7
0100	4	6	10	9	4
0101	5	15	3	12	10
0110	6	11	7	14	8
0111	7	1	4	2	13
1000	8	10	12	0	15
1001	9	9	5	6	12
1010	10	3	6	10	9
1011	11	14	11	13	0
1100	12	5	0	15	3
1101	13	0	14	3	5
1110	14	12	9	5	6
1111	15	7	2	8	11

图3-14 DES 的第8个S盒

3.2.4 Skipjack 算法

Skipjack 算法是NSA为Clipper芯片开发的加密算法。然而，它被划为美国政府机密，没有太多关于该算法的描述。但有一点已经清楚：它是一个对称密码算法。使用 80位的密钥，且加、解密需要进行32次圈运算。

Clipper芯片是一种使用Skipjack 算法的加密芯片。它是一种由NSA设计的商用加密芯片。AT&T计划用它去加密语音电话线路。

但是Skipjack 算法安全吗？

就我本人所知，NSA已经采用Skipjack 算法加密它自己的信息系统。因此，我们可以认为算法本身是安全的。Skipjack 算法使用长为80位的密钥。也就是说有 2^{80} (大约 10^{24}) 或多于1GG的可能的密钥供你使用！这意味着你将花费4000亿年才能穷尽该算法的密钥空间 (你做好思想准备了吗？)。

为了使你有一个比较形象的理解，我们假设使用100 000台RISC计算机，每台计算机每秒能够进行100 000次加密运算，那么为了破译一组代码将需要4百万年。

Skipjack 算法的设计者估计，每18个月，破译这个算法的代价将减半。根据这种假设，不到36年破译Skipjack算法的代价就会和今天通过“强行攻击”手段破译DES的代价相同。据此，他们认为，在以后的30~40年内Skipjack 算法是可靠的。另外，我们已经知道Skipjack 算法能够抵抗密码分析攻击，它不依赖于算法本身，因此即使是算法被公开，Skipjack 仍然会十分可靠。

提示 要获取更多Skipjack算法的详细资料请访问：

http://www.cpsr.org/cpsr/privacy/crypto/clipper/skipjack_interim_review.txt ,
那里提供了该算法的完整介绍。

Clipper芯片使用带2个密钥的Skipjack算法。无论是谁，只要知道了“主密钥”就可以解密所有用该芯片加密的信息。因此，在必要情况下 NSA至少在理论上可以利用它的“主密钥”破译一切用Clipper芯片加密的信息。这种在算法中留有后门的方法被称为第三方密钥 (key escrow)。

许多对安全考虑比较多的用户和商业部门都抵制 Clipper芯片，因为他们感到这种芯片会侵犯他们的隐私权。从 <http://www.austinlinks.com/Crypto/non-tech.html> 可以获得更多的关于这种芯片窃取信息的详细报导。

3.2.5 RC2/RC4 算法

直到源代码被公开在 Usenet上之前，RC4算法一直是商业机密。它是由 RSA Data Security 公司设计的一种非常快的加密算法。RC4被认为是一种强度很高的加密算法。但它提供给 Netscape的SSL使用RC4-40的出口版本最近至少被2个独立的小组破解，他们花费了大约8天的时间。

表3-2给出了各对称密码算法的比较。

表3-2 对称密码体制比较表

算 法	安 全 性	速度(486 PC)	密钥长度
DES	低	400kbps	56 位
3DES	好	150kbps	112 位
IDEA	好*	200kbps	128 位
3IDEA	很好*	~ 100kbps	256 位
Skjjack	好*	~ 400kbps	80 位
CLIPPER chip	好**	-	80 位

注 “*”表示该算法被认为是很强的。

“**”表示该算法本身很好，但实现有漏洞。

3.3 非对称密钥加密/公开密钥加密

在这种密码系统中，需要有两个密钥一起使用。其中一个总是保密而另一个可以公开。可以使用它们中的任意一个用于加密或解密。公开密钥密码体制可以用来解决向用户进行密钥分配的问题。

下面是一些公开密钥加密体制运用的例子：

证书 可以用来确认传输中使用的是正确的公开密钥和秘密密钥。

数字签名 是接收者确认消息发送者身份的一个手段。在这种情况下，只有使用者知道秘密密钥，而且他必须保证这个秘密密钥不被泄露出去。该使用者的公开密钥可以公开，任何想与此用户通信的人都可以使用这个密钥。

明文 用秘密密钥加密后的明文，可以使用相应的公开密钥甚至是同一秘密密钥来解密。

一种最主要的公开密钥加密算法是 RSA，它是根据其三位发明者 Rivest，Shamir及

Adleman的名字命名的。公开密钥密码算法总是优点与缺点并存的。通常该算法的加解密使用相当大的密钥，一般都是100位以上的数。这就是为什么行业中趋向于使用灵巧卡（smart card）如安全身份证等来解决密钥管理和计算日常开支的问题。

Zimmermann的PGP就是公开密钥系统的一个例子。它被越来越广泛地用于在Internet上传输信息。它使用的密钥比较简单，但能提供较高的保密等级。PGP唯一不便之处，在于如何去获取接收方的公开密钥。随着使用的增加，有许多公开密钥将无处存贮。针对这一现象，作为新的LDAP技术承诺的一种“全球公开密钥注册”已经开始运作。

注意 什么是LDAP?

LDAP是Lightweight Directory Access Protocol (轻型目录访问协议)的缩写，它是一系列关于访问信息目录的协议。LDAP基于X.500协议，但比X.500更容易使用，而且支持TCP/IP(X.500不支持)，这对任何类型的因特网访问都是必要的。

使用LDAP的用户可以最终获取任一接入因特网的计算机上的目录信息，而且与该计算机的硬件及软件平台无关。因此它允许在不需清除主站点的情况下查找某一特定地址或公开密钥。

3.3.1 RSA

RSA于1977年由Ron Rivest, Adi Shamir及Leonard Adleman发明,是一种既可以用于加密又可以用于认证的公开密钥密码算法。由于它是目前最为广泛使用的公开密钥密码系统，RSA已成为一种标准。

RSA的工作原理如下：寻找两个大素数 p 、 q ，计算它们的积 $n=p*q$ ，计算一个小于 n 且与 $(p-1)*(q-1)$ 互素的数 e 。使得：

$$ed=1 \bmod (p-1)(q-1);$$

e 和 d 分别被称为公开指数和秘密指数。 (n,e) 是公开密钥， d 是秘密密钥，因子 p 和 q 必须保密或被销毁。

很难(据推测)由公开密钥 (n,e) 推导出秘密密钥 d ，如果能够将 n 分解为 p 和 q ，那么就能得到秘密密钥 d 。因此整个RSA的安全性建立在大数分解很难这一假设基础之上。一个简便而又行之有效的大数分解算法就可以攻破RSA。

RSA比较快，但它没有DES快。目前最快的RSA芯片对于模512位大数的计算能力超过600Kbps，这意味着每秒能进行1000次RSA私有密钥操作。

RSA算法安全吗？

RSA的安全性取决于密钥长度。攻破一个384位的密钥比攻破512位的密钥要简单得多。前者是不安全的，可破解的。如果使用的是768位的密钥，那么可能的密钥组合数量将急剧增长。根据RSA的FAQ(见图3-15)，一个1024位的密钥在将来几十年内都是安全的。RSA的FAQ网址是：<http://www.rsa.com/rsalabs/newfaq/secprserv.htm>。

但这并不意味着RSA是不可破解的。如果能计算 n 的 e 次方根，那么就可以破解这个密码。由于 $C=m^e$ ， C 的 e 次方根就是明文 m 。这种攻击意味着第三方即使在不知道秘密密钥的情况下也能伪造签名或恢复明文。

同样，根据上文提到的RSA的FAQ，该密码系统对选择明文攻击十分脆弱。一个有效的猜测将会破解出其所使用的密钥。因此，加密明文时最好加入一些随机数（至少64位）。

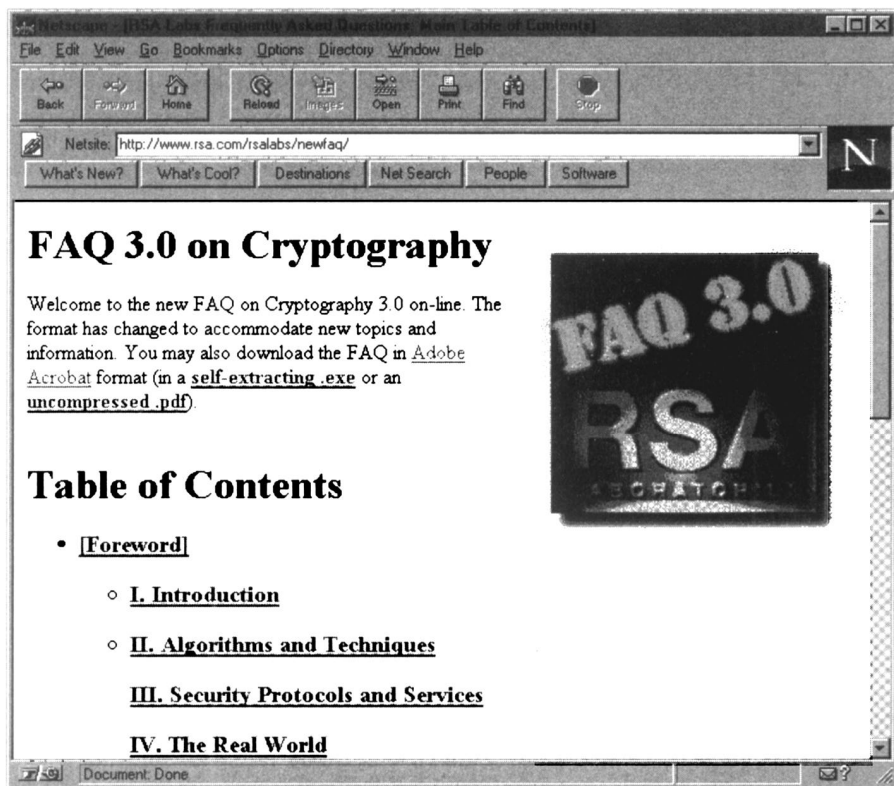


图3-15 RSA 的FAQ Web站点

3.3.2 数字签名标准

数字签名标准是美国政府数字签名标准。DDS存在一些问题，其中之一就是秘密数据的泄露。并且，如果你两次使用相同的随机数用以签名，那么秘密密钥将会被破解，而且随着 Diffie-Hellman 及RSA密码系统的实现(这两者都比DSS好)，我认为没有必要使用DSS。

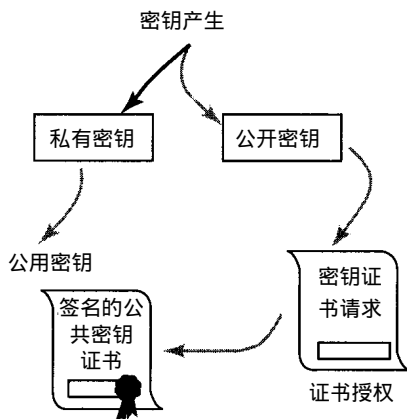


图3-16 公开/私有密钥的产生

表3-3给出了目前可行的非对称密码系统的比较。

图3-16概述了公开密钥和私有密钥产生的过程。

表3-3 非对称密码体制对照表

算 法	安 全 性	速 度	密 钥 长 度
RSA	好	快	可变(1024位以上算法安全)
Diffie-Hellman	好	< RSA	可变(1028位以上算法安全)
DSS	低	—	512位

3.4 消息摘要算法

消息摘要(Message Digest, MD)算法(通常称之为“哈希”算法——译者注),能对任何输入的消息进行处理,生成128位长的“消息摘要”输出,也叫做“指纹采集”。两个输入的消息不可能有相同的消息摘要。下面将详细讨论三个版本的消息摘要算法: MD2, MD4和MD5。

3.4.1 MD2、MD4和MD5

MD5是MD算法的最新版本,是一种安全的哈希算法,是由RSA Data Security公司提出的。MD5可以将任意长度的字符串散列成128位的值,这与它以前的版本一样。但MD5采用了更安全的哈希算法,因此它的应用更广泛。

MD5将输入文本放入512位的块中,然后分成16个32位的子块。输出是4个32位的子块,它们组成了一个128位的哈希值。

尽管它十分安全,但最近有报导说,MD5有潜在的安全漏洞,在某些情况下是可攻破的。也有报导说,花费几百万美元就可以建造一台特殊用途的计算机,只需几个星期的时间就可以找到一个明文匹配给定的哈希值。也有可能更容易做到这一点。

例如Microsoft Windows NT就使用了MD4(MD4将在下一节中讨论),它将加密后的口令都存放在其安全帐户管理(SAM)数据库中。1997年春季早些时候,Windows NT的一个安全漏洞被批露出来,这当然也就涉及到了MD4。

因特网上有许多实用程序,如:PWDUMP(可以从<http://www.masteringcomputers.com/util/nt/pwdump.htm>下载)和NTCRACK(可以从<http://www.masteringcomputers.com/util/nt/ntcrack.htm>下载)都被用来攻击NT用户口令。用以存放NT口令的SAM数据库就是PWDUMP的攻击对象。但SAM不是将口令以明文形式存放,而是用哈希值的形式存放的。如图3-17所示。

如果仔细研究图3-17,就能得到我的计算机口令的哈希值,但口令本身你是无法得到的。当一个口令第一次输入给NT时,系统用MD4为此口令产生一个哈希值,如图3-17第四行所示,它在域“NTHASH”之前。尽管这个哈希值在存放至SAM数据库之前被加密,但可以用PWDUMP攻破这个哈希值。

把MD4产生的哈希值进行加密的函数可由PWDUMP找到。因为MD4的加密过程是已知的(还记得在本章前几节中,我们所提到的Usenet上提供的MD4源代码的事吗?),于是口令可由一个逆过程找到。可以使用NTCRACK或其他派生工具为MD4加密系统输入一组单词(例如,可

以从一本字典中去找)，然后比较每个字符串的哈希值，直到找到一个与口令的哈希值匹配的单词。这对攻击NT很有效，因为它在加密过程中不使用随机数 (SALT)。当然，这并不意味着使用了SALT的UNIX操作系统就会更安全些，它只是稍稍延长了破译时间！

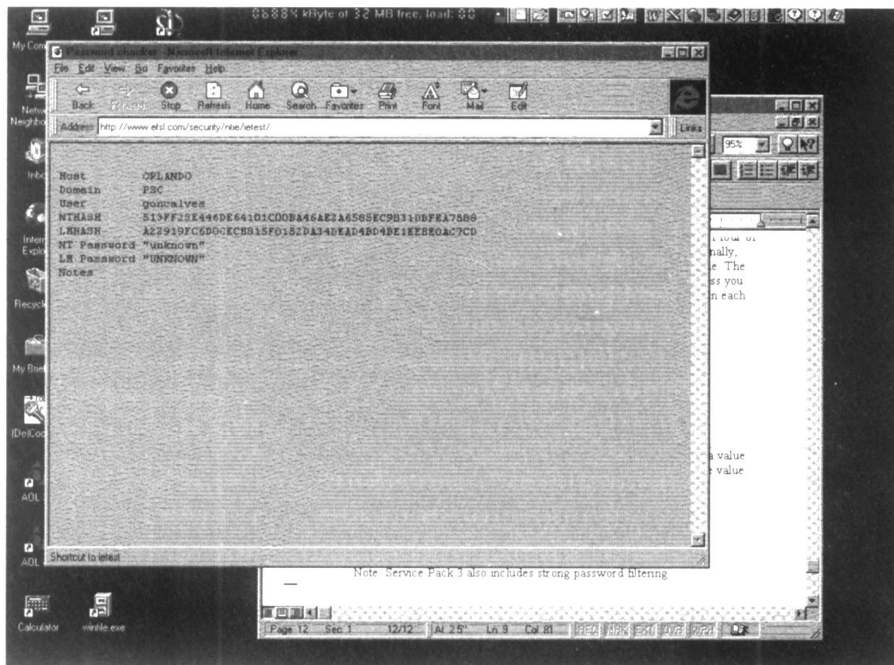


图3-17 存在NT SAM 数据库的口令的哈希值

破译NT的口令加密系统和MD4，就现在而言已不再是一项大问题了。最大的挑战是你需要以管理员的身份连接到想要浏览的机器上。如果做到了这些，下面将是需要继续做的：

- 1) 建立一个临时目录用以运行上述工具，确认 PWDUMP及NTCRACK在这一目录中。
- 2) 输入 PWDUMP>LIST.TXT(或其他任一你想存放的文件名)，这个文件将存放所有 PWDUMP所能找到的口令哈希值。

3) 现在是使用NTCRACK的时候了！输入 NTCRACK PASSWORDS LIST.TXT> CRACKED.TXT。(PASSWORDS是包含单词文件的文件名。最好是一本ASCII格式的字典。) NTCRACK提供了基本的字典文件，但应该再添加一些字词。甚至可输入一整本字典！一旦这一操作过程结束，只需使用任何文本编辑器打开CRACKED.TXT文件即可查看被破译的口令。

前文给出的 NTCRACK版本是本章编写时的最新版本。该版本不仅可攻击在其字典库中的口令，而且也能用于确认用户名。我用它作为破译口令的一个例子，如图 3-18所示。注意只有在字典库中的口令才是可破译的。这就是为什么建议使用 8个以上的字符，而且要在任何字典中都找不到的长口令。

如果想试试这个工具，可以在 Web上实践一下。所有你需要进行的操作只是运行 Internet Explorer并访问<http://www.efsl.com/security/ntie/>。

然后点击超链接“TRY IT”。Internet Explorer也同样存在安全漏洞。如果你的口令是它字典文件中的一部分，该系统就能给出被解密的口令：

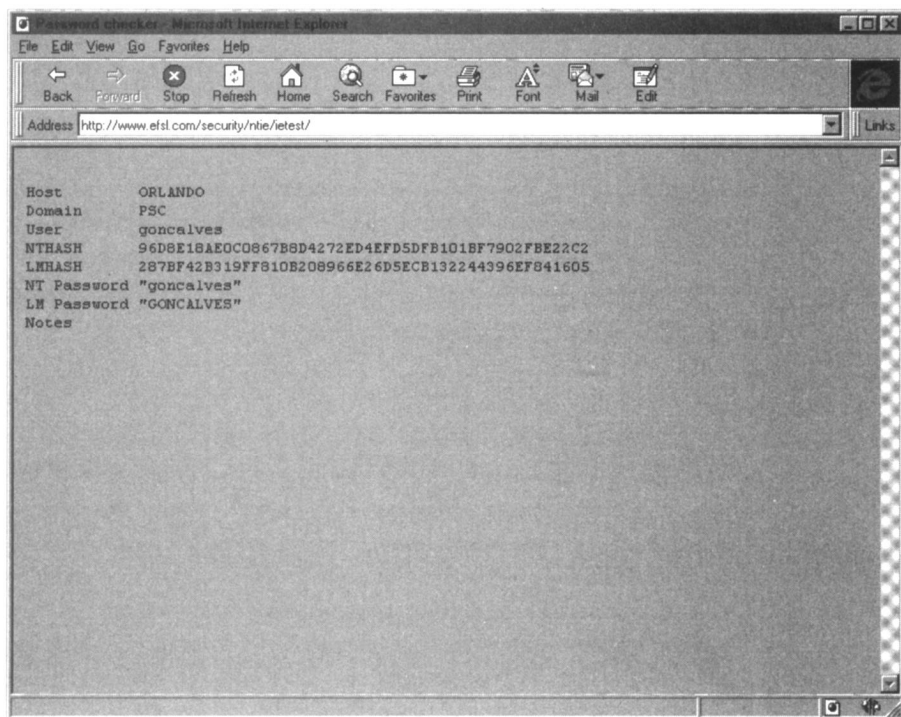


图3-18 被NTCRACK侦测出的口令，因为它与用户帐户名一致

正如图3-18所示，在前一幅图中，口令是不可知的，而现在它给出了我的姓氏：GONCALVES。说明我的口令与用户名是相同的。

MD5被认为比MD4更安全，可以在大多数情况下使用。

3.4.2 安全哈希标准/安全哈希算法

安全哈希算法(SHA)也被称为安全哈希标准(SHS)，是由美国政府提出的，它能为任意长度的字符串生成160位的哈希值。

SHS在结构上与MD4、MD5相似。它只比MD5慢25%。但在另一方面，它更安全。因为它的消息摘要要比由MD函数产生的要长25%。这使得它比MD5能更加有效地抵抗强行攻击。

3.5 证书

为了提供用户身份及他们的密钥认证服务，公开密钥系统要求一个可信任的、独立于通信双方的第三方。

由于它的工作是确认公开密钥持有人的真实身份，因此这个第三方机构被称为“认证中心(Certification Authority, CA)”。CA(如VeriSign)用它自己的秘密密钥签名一个证书，使之成为公开密钥证书(Public Key Certificate)，它包括一些用户的身份细节和用户的公开密钥。

用户通过使用CA的公开密钥验证CA的签名，可用来验证其他用户的公开密钥证书，从而证明该用户及其公开密钥的可靠性。这项技术已获得广泛运用。

解密消息之后，接收方可以验证发送方的数字签名。为了做到这一点，可用相同的产生

原始签名的哈希算法生成一个文档摘要。同时将附加在该文档的数字签名利用发送方的公开密钥进行解密运算，这就生成了数字签名的摘要。

然后，比较数字签名摘要和利用哈希算法生成的文档摘要是否一致。如果摘要和数字签名匹配，接收方就可以确认文档在传输过程中未被改变，并可以确认发送方的真实身份。即使有细微的差别，签名也会被拒绝。

由于发送方是唯一知道用于签名的秘密密钥的人，所以发送方不能抵赖他曾发送过该消息。图 3-19 给出了一个验证数字签名的过程。

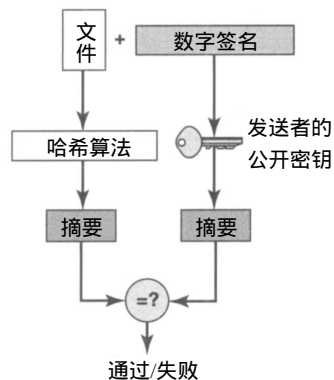


图3-19 验证一个数字签名

证书服务器用于生成、签署和管理基于标准的公开密钥

证书。使用证书服务器的机构，如 Netscape 的证书服务器 (http://home.netscape.com/comprod/server_central/support/faq/certificate_faq.html#1) 能够在不依赖于诸如前文提到的 VeriSign 外部 CA 服务的情况下，由这个基础机构管理它们自己的公开密钥证书。

另一个供应商 OpenSoft (<http://www.opensoft.com/products/expressmail/overview/certserver/>)，也提供了基于 Windows NT 及 Windows 95 平台的证书服务器技术。OpenSoft 使用一种基于新的分布式证书系统 (Distributed Certificate System, DCS) 的技术，这种新技术使得它能提供一个可靠的公开密钥分配系统，图 3-20 给出了 OpenSoft 的认证服务器网页。



图3-20 OpenSoft 的认证服务器网页

注意 什么是DCS?

DCS 服务器是速度优化的证书服务器，基于 DNS 模型。该服务器最初只支持4种资源记录类型：证书记录（CRT），证书撤销清单，由可区别名指定的证书服务器记录（CS）和邮件域上证书服务器记录（CSM）。

分布式证书

DCS面纱下到底是什么？

DCS是一种基于DNS模式的速度优化证书服务器，这种服务器最初仅支持四种格式的资源记录：

证书记录(certificate record,CRT)

证书撤销表单(certificate revocation list,CRL)

使用可区别名称的证书服务器记录 (certificate server records by distinguished name,CS)

邮件域上的证书服务器记录 (certificate server records by mail domain,CSM)

由于分布式证书系统趋于继续扩展，正式协议中未定义的新的数据类型以及实验性操作总有可能出现在系统中。在 DNS中，DCS服务器使用一种定界的、基于名为 DCS主文件 (master file)的文件格式。DCS服务器允许多个主文件被连接使用，同样存在根文件（root file）存储根服务器信息。

证书服务器 证书服务器允许一个用户代理或其他证书服务器查询证书信息。下面是证书服务器的特性：

1) 一个证书服务器拥有以下记录：

a. 一个有三个域的记录：

可区别名

记录类型(CRT)

证书

b. 一个有三个域的记录：

CA的可区别名

记录类型(CRL)

签名的CRL

c. 一个有三个域的CS记录：

可区别名段

记录类型 (CS)

服务器地址

d. 一个有三个域的CSM记录：

域名

记录类型(CSM)

服务器地址

2) 在CRT或CRL查询中，用户代理向证书服务器发送一个证书或 CRL请求，并给出可区别名。

a. 如果一个CRT或CRL记录不存在，服务器搜索 CS记录以确认在什么地方能找到这个证书；否则，该服务器向根 DCS服务器询问到哪里去查找这个证书或 CRL。

b. 如果CRT或CRL记录存在，就返回这个证书或 CRL。

a. 根据RFC 1779(“可区别名的字符串表示”)获取CS(M)查询所需的可区别名的必要格式。

c. 只有标记过的属性或属性组才能用于 CS 查询；此标识是位于该服务器上有正确密钥的可区别名的证书的公共成员。但不是所有此服务器上的证书都有此标识。

e. 在默认情况下,如果没有其他属性或属性组被标识,用户代理人使用 e-mail属性作为它的标识属性。即从e-mail 地址抽取出域名并将其作为标识属性。

4) CRT, CRL和CS(M)记录存储在一个DCS主文件, 该文件格式与DNS主文件格式相似。

在图3-21中，请注意以下几点：

- 1) 编辑DCS主文件，使用记录：CRT，CRL，CS，CSM。
- 2) 向证书认证机构申请CRL，使用记录：CRL。
- 3) 向证书服务器申请证书及CRL，使用记录：CRT，CRL。
- 4) DCS内部服务器通信，使用记录：CS，CSM。

DCS的拓扑结构说明了该系统具有高速的特性。用户代理可以查询本地证书服务器，并能在毫秒时间内接收到从本地证书服务器或是另一个位于因特网任一地点的服务器发回的目标证书或CRL。

DCS协议 参考RFC 1032~1035 中有关DCS精确查询语法的DNS协议。DCS查询协议将与DNS查询协议有相同的格式。DCS查询中有关可区别名的语法将遵循 RFC 1779(“可区别名的字符串表示”)。

所有DCS协议内部通信都由一个被称为DCS消息(DCS message,DCSM)的唯一形式所携带。该消息的高层格式与DNS一样被分为5个部分。在某些情况下某些部分可能为空。如图3-22所示。

观察图3-22，请注意消息头部分必须非空。消息头部分标明了其他的哪几个部分不空，

以及这个消息是一个查询或一个响应、一个标准查询还是其他操作码等等。

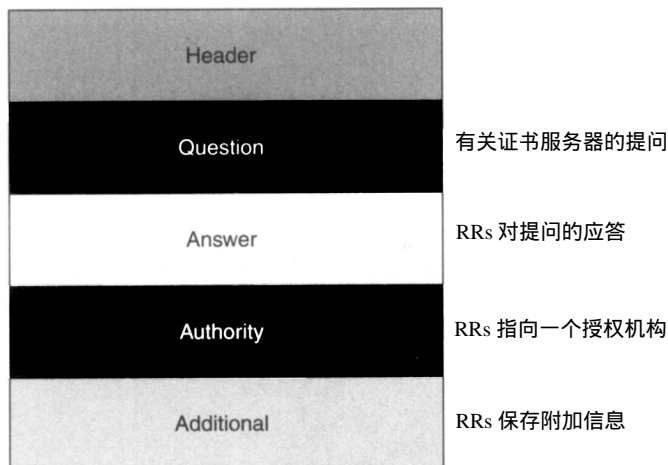


图3-22 DCS 消息 (DCSM) 的高层格式

消息头后其他部分的名称，是根据它们在标准查询中所起的作用命名的。消息提问部分包含了一些向某一名服务器提问的域。这些域的类型为查询型（同DNS中的QTYPE）或查询类（同DNS中的QCLASS）。最后三部分有相同的格式：一个可能为空的连接 DCS记录的列表。应答部分包含RRs应答提问；权限部分包含指向授权的名服务器的RRs；附加部分在DCS中没有使用。

消息头部分的格式 消息头包括以下几个域。如图3-23所示：

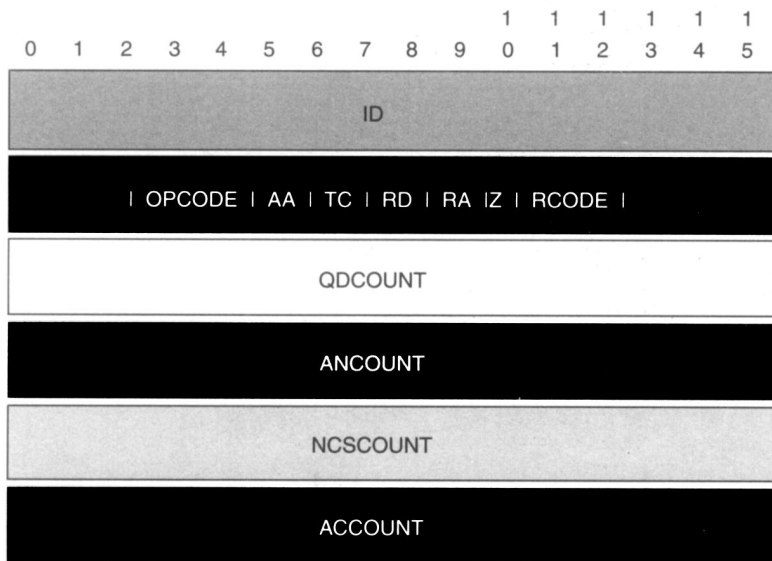


图3-23 DCS 消息的头部格式

ID——一个由产生任何形式查询的程序分配的 16位长的标识。该标识被拷贝到相应的应答，并被请求方用以匹配查询应答。

QR——一个用来指明该消息是查询(0)还是一个响应(1)，长度为1位的域。

OPCODE——一个用以标明此消息的查询类型，长度为4位的域。这个值由查询的始发站给出并被拷贝到响应中。其值为：

0——一个标准查询(QUERY)。

1——一个反向查询(IQUERY)(DCS不支持)。

2——一个服务器状态请求(STATUS)。

3——一个简单查询。证书服务器搜索一个信息直到它找到第一个被请求的 DCS记录(SMQUERY)。

4——一个更新查询。当传送给证书服务器新的证书或 CRL时，认证中心(CA)设置该值。

5~15——保留给将来使用(DCS中)。

AA——授权应答(Authoritative Answer) 在应答及确定请求部分中唯一的、拥有授权的名服务器时该位有效。注意，由于使用别名，应答部分的内容可能会有多个拥有者。

AA位与所查询或应答部分中的第一个拥有者的名字相匹配时做出响应。

TC——截断(Truncation)，标明由于消息长度超过信道允许的最大长度而被截断。

RD——递归请求(Recursion Desired)，该位在查询时设置并被写入响应。若 RD置位，它表明进行递归查询的名服务器。递归查询支持是可选的。

RA——递归有效(Recursion Available)，该位在响应中被置位或被清除，给出该名服务器是否支持递归查询。

Z——保留将来使用。在所有的查询和响应中必须为0。

RCODE——响应码(Response code)，该4位的域是响应的一部分。其值的说明如下：

0——无错状态。

1——格式错，证书服务器无法解释该查询。

2——服务器失败，由于证书服务器出现问题，DCS服务器无法处理被申请的查询。

3——名错误，只对由一个授权名服务器的响应有意义。该值指出在查询中所给出的可区别名不存在。

4——未实现，证书服务器不支持该类型的查询请求。

5——拒绝，证书服务器由于决策原因而拒绝执行某一操作。例如，一台证书服务器可能不愿为某些请求者提供信息或不愿为某些数据执行某些操作，如零传输。

6~15——保留给将来使用。

QDCOUNT——一个标明提问部分的条目数，16位无符号整数。

ANCOUNT——一个标明应答部分中RR数，16位无符号整数。

NSCOUNT——一个标明权限记录部分RR数，16位无符号整数。

ARCOUNT——一个标明附加记录部分的资源数，16位无符号整数。在DCS协议中，该值必须为0。

提问部分格式 在大多数查询中，提问部分被用来携带“问题”，如定义目前正在被请求的参数。该部分包括QDCOUNT(通常为1)条目，条目格式如图3-24所示。

QNAME——一个DER编码的可区别名。

QTYPE——一个标明查询类型的双字节代码。该域的值包括所有TYPE域可能的代码。

QCLASS——一个标明查询类的双字节代码。该域只为了与DNS兼容。对于DCS，该域

一定等于IN(Internet)。

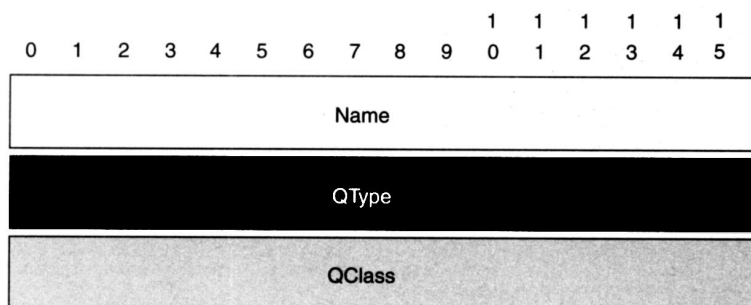


图3-24 DCS 消息提问部分的格式

DCS记录 应答与认证都使用同一格式：一个数目可变的资源记录数，记录的个数在消息头部分相应的计数域中给出。

如图3-25所示，每一个资源记录有以下格式：

NAME——一个DER编码的可区别名。如果它的第一个属性为e-mail地址，服务器通过e-mail地址查询信息。否则，服务器根据所有可区别名进行查找。在一个查询中，一个可区别名的属性可以包含一个星号(*)作为通配符，也就是说，所有该属性的值都适合这一模板。实际上，对于一个等于星号的值，服务器只检测该属性的存在而忽略它的值。

TYPE——一个包含DCS记录类型之一的双字节代码。该域指定RDATA域中数据的含义。

对于CS记录 Type值为1001
 对于CSM记录 Type值为1002
 对于SOC记录 Type值为1003
 对于SOCM记录 Type值为1004
 对于CRT记录 Type值为1005
 对于CRL记录 Type值为1006

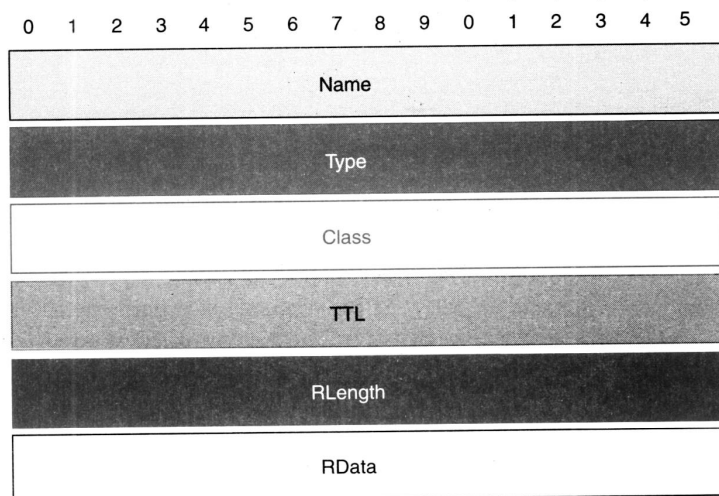


图3-25 DCS 记录格式

AXFR——252——请求传输条目(与DNS查询类似)，该值与DNS中的AFXR的值一致。

CLASS——一个用以标明RDATA域数据类型的双字节代码。对于DCS，该值一定IN。

TTL——一个用以标明资源记录在删除之前能被缓存的时间长度(以秒为单位)，32位无符号整数。该值为0表示RR只能用于程序中的事件处理而不会被缓存；每一个DCS记录都有一个时间值。这个域不是必须的。

RDLENGTH——一个用以标明RDATA域长度(用八位组形式)的16位无符号整数。在DCS中，DATA是DER编码过的值。RDATA包含了它的长度；因而该域没有用。

RDATA——一个DER编码的ASN.1变量类型。该信息的格式随RR的TYPE的改变而改变。

如果你想了解更多关于DCS的消息压缩、传输、服务器算法，请查询OpenSoft的网页：<http://www.opensoft.com/dcs/>。这部分内容已超出了本书所讨论的范围。

3.6 密钥管理

保护信息的完整性与安全性唯一有效的方法，是按本章前半部分所提到的方法，即使用秘密密钥将信息进行加密或签名等处理，使之成为秘密信息。管理和处理秘密信息通常称为密钥管理。它包括选择、交换、存储、注销、更改、传输密钥及签署证书。因此大部分信息安全的工作都被放在密钥管理上。

正如在前几节所提到的公开密钥密码的密钥管理，它很受欢迎的原因就是它把密钥传输的问题简单化了。某人发送一个消息，只有接收方能够读取，而接收方无需知道发送方使用的密钥。由于加密密钥与解密密钥不同，接收方也无需与发送方共用同一密钥。

密钥管理不仅为加密信息的交换提供了便利，而且提供了数字签名的实现方案。公开密钥与秘密密钥的分离，允许用户对其数据进行签名，允许他人能通过公开密钥验证其签名而又不必将他们的秘密密钥公开，它为这一系列活动提供了必要的保障。

3.6.1 Kerberos 协议

Kerberos协议通过规范用户访问网络服务来提供网络安全性。在Kerberos环境下，至少有一个运行的Kerberos服务器。系统必须保证其安全性。Kerberos服务器被认为是一个“可信赖的”服务器，它为发送请求的用户提供身份认证服务。Kerberos服务器也被称为密钥分配中心(Key Distribution Center, KDS)。

其他网络上的服务器及所有的客户，均被认为是不可信赖的。为了能使Kerberos协议正常工作，所有依赖该协议的系统都必须信任Kerberos服务器。

除了能够提供认证服务之外，Kerberos服务器还能提供诸如数据完整性、可靠性等安全服务。

Kerberos使用基于DES的私有密钥加密，每一个客户及服务器都有自己的DES私有密钥。Kerberos协议将客户及服务器作为主体。客户口令被映射到它的私有密钥上。

提示 想要了解关于Kerberos及其在网络安全环境中运用的更多信息，请访问Process Software网站<http://www.process.com>。它不仅是TCP/IP解决方案公司中比较出色的一个，而且它拥有大量的IPv6，Kerberos，以及TCP/IP的资料。

Kerberos服务器有一张存放所有被允许使用Kerberos服务器服务的客户及服务器名字，及其私有密钥的数据库列表。Kerberos假设它的所有用户(客户及服务器)都很好地保存着它们的口令。

Kerberos协议通过客户和服务器都信任的第三方，即Kerberos服务器验证客户的身份，解决了服务器确认客户身份的问题。

1. 了解Kerberos 术语

下面是一些与Kerberos相关的术语：

主体(principal)——Kerberos将客户及服务器视为主体，并分配一个名字。一个常见名字格式示例为：`name.instance@realm`。

名字(name)——对于客户来说是它的记录名，对于服务器来说则是其所提供的服务名。通常是`rcmd`。

实例(instance)——对于客户来说，它因为并非必要通常被忽略。对于Kerberos管理员，这个值是`admin`；对于服务器，它指出拥有Kerberos认证支持的应用服务器的机器名。如`hostX`的远程登录服务器，如果拥有Kerberos认证支持，主体将会获取如下格式：
`rcmd.hostX@your_realm`。

领域(realm)——它关联所有Kerberos数据库中的主体。它是一组诸如在LAN的机器名字。它标识了Kerberos的域名。

在某些主体中，`instance`及`realm`部分可以被省略。如，`joshua`(本地客户)的主体对于`xuxu.com`域的客户Jones来说，可以是`joshua@xuxu.com`。一个可能的主体同样可能是`rcmd.hostX`(对于登录本地域的远程登录服务器而言)或`rcmd.hostX@xuxu.com`(对于域为`xuxu.com`的`hostX`上的远程登录服务器而言)。

凭证-授权凭证(ticket-granting ticket) 它包含客户Kerberos口令的加密形式。利用它可以从Kerberos服务器上获取应用服务凭证(server ticket)。在没有获取这一凭证之前，你不会获得Kerberos授权。凭证-授权凭证有一个由Kerberos服务器标识的有效期，此有效期通常为8小时。你可以在不需要这个凭证，或它的有效期结束之前反复使用它。

服务凭证(service ticket)——Kerberos使用服务凭证向应用服务器证明客户的身份。Kerberos服务器使用应用服务器的私有密钥加密服务凭证。这使得只有相应的应用服务器才能解读这一服务凭证。

认证者(authenticator)——Kerberos协议使用“认证者”，防止偷听者窃取凭证。客户为每一个服务请求发送一个新的“认证者”。“认证者”包括客户的名字、IP地址，以及用以标明当前时间的时间标识。

服务器根据“认证者”中的信息以确认出示该凭证拥有者的真实身份。为了实现这一点，客户与服务器必须同步它们的时钟。通过网络时间协议(Network Time Protocol, NTP)可以得到该问题的解决方案。

2. Kerberos会话中有些什么

Kerberos协议是一个用于开放系统和网络的认证系统。Kerberos使用一系列的加密过的密钥及凭证完成认证，以实现两个系统安全认证。

另一方面，由于标准认证方法通常将客户名、口令在网络上以明文形式传输，因而不是安全的。

一个典型的Kerberos会话 下文描述了一个Kerberos会话的基本过程(如图3-26所示)：

1) 客户向 Kerberos 服务器提交“凭证—授权凭证 (TGT)”的申请。Kerberos 服务器查询 Kerberos 数据库 (KDB) 以得到客户的 Kerberos 口令并将其加密。

2) Kerberos 服务器将加密了的口令写入 TGT 中，并传送给客户，当客户收到这个 TGT 后，它再申请客户的 Kerberos 口令，然后加密该口令并与 TGT 中的口令进行比较。Kerberos 服务器用这种方式认证一个客户。

3) 客户使用 TGT 申请应用服务凭证，使得它能够访问特定的应用程序。每一张服务凭证都向应用服务器提供了客户的身份。

4) 客户向应用服务器出示服务凭证以获取认证。应用服务器解密凭证以检验其可靠性。

5) 如果应用服务器验证该服务凭证有效，它向客户提供最初拒绝的访问服务。如果应用服务器无法解密该服务凭证、该服务凭证过期或未得到授权，客户将得不到认证。

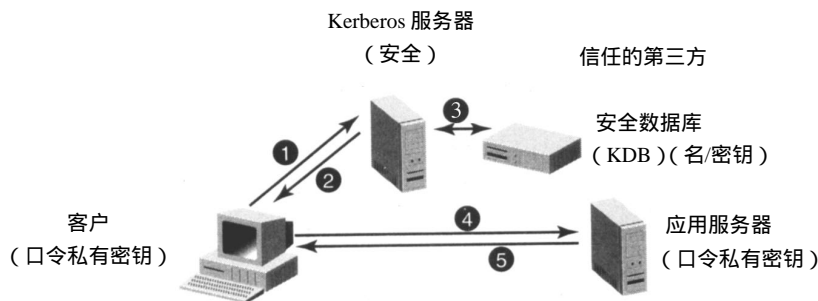


图3-26 典型的Kerberos会话顺序

下面几部分将更详细地描述一个 Kerberos 会话。

从 Kerberos 服务器获取一张凭证—授权凭证 (TGT) Kerberos 服务器在其机器上有一个安全数据库。客户必须从 Kerberos 服务器获取一张 TGT，而客户自身无法解读这张 TGT。

TGT 允许客户向 Kerberos 服务器申请获取访问应用服务器的应用服务凭证。客户在请求应用服务器服务时必须出示此服务凭证。如图 3-27 所示，下述过程描述了获得一张 TGT 的过程。

1) 客户用户向 Kerberos 服务器发出一个请求，该请求包中有客户的用户名。

2) Kerberos 服务器在它的安全数据库中查找该用户名并提取该用户的私有密钥。

3) Kerberos 服务器：

a. 产生一个用于客户与 Kerberos 服务器间的随机生成的密钥。这就是被称为凭证—授权凭证的会话密钥。

b. 产生一个 TGT，使得客户能够从 Kerberos 服务器那里获取应用服务凭证。Kerberos 服务器使用从 Kerberos 数据库中获得的客户的私有密钥加密这个 TGT。

凭证 (Ticket)：(客户名，Kerberos 服务器名，客户 Internet 地址，会话密钥) 私有密钥。

Kerberos 在 TGT 中也包含一个时间戳。

c. 产生一个包含有会话密钥及加密过的 TGT 的包，并用从安全数据库中得到的客户私有密钥加密这一消息。

包 (Packet)：(会话密钥，加密后的凭证—授权凭证) 客户私有密钥。

d. 向客户发送包含有加密后的 Kerberos 口令包。

4) 客户使用它的私有密钥解密这个包。当收到这个包，解密过程就能使客户得到它的口

令。客户使用这一私有密钥加密它的口令并将它与传送来的 TGT 中加密过的口令比较。若口令匹配，客户得到一有效的 TGT；否则，包将被丢失，客户无法得到 Kerberos 认证，从而无法访问任何应用服务器。

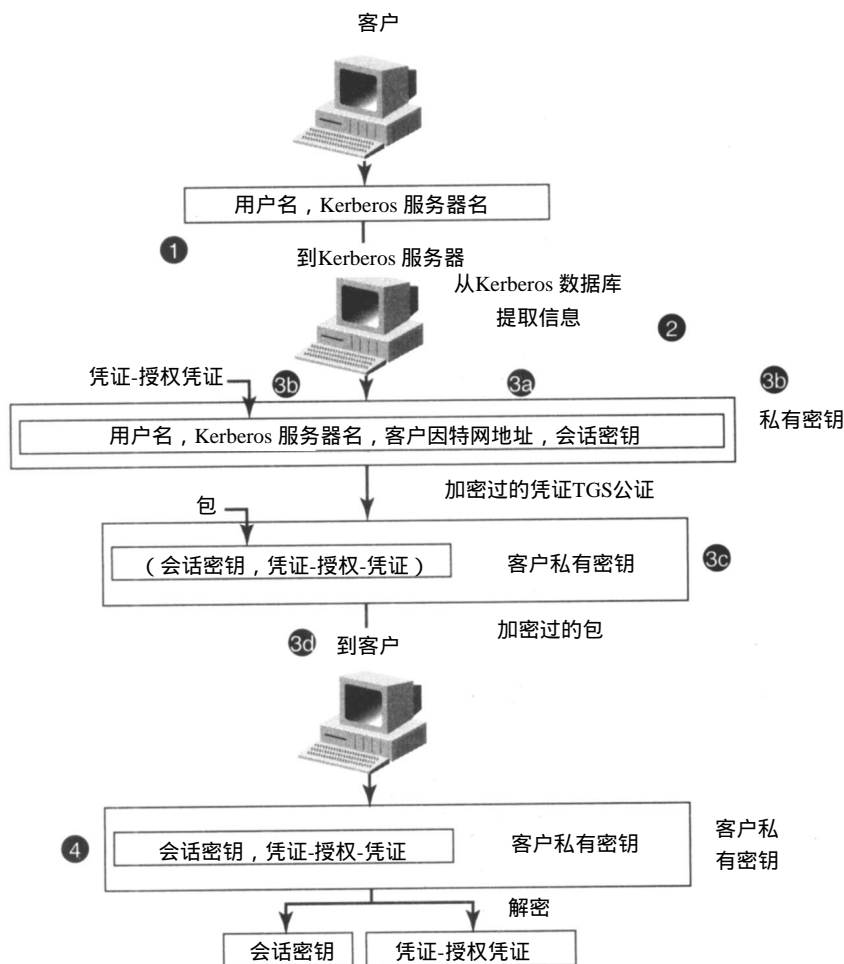


图3-27 获得一个凭证-授权凭证

注意 名字只存在于Kerberos 服务器和客户工作站之间以明文交换。

从Kerberos服务器获取网络服务的应用服务凭证 一旦客户获取了“凭证-授权凭证(TGT)”，它就能向应用服务器申请访问网络应用程序。

每一个这样的请求，首先从凭证-授权服务(Ticket-Granting Service, TGS)处获取一张访问特定应用服务器的应用服务凭证。

图3-28及图3-29描述了获取用于访问应用服务器的应用服务凭证的过程。

客户

产生一个客户和Kerberos服务器共享的“认证者(authenticator)”。客户使用在早些时候收到的会话密钥加密这个“认证者”。“认证者”包含三个部分：

用户名。

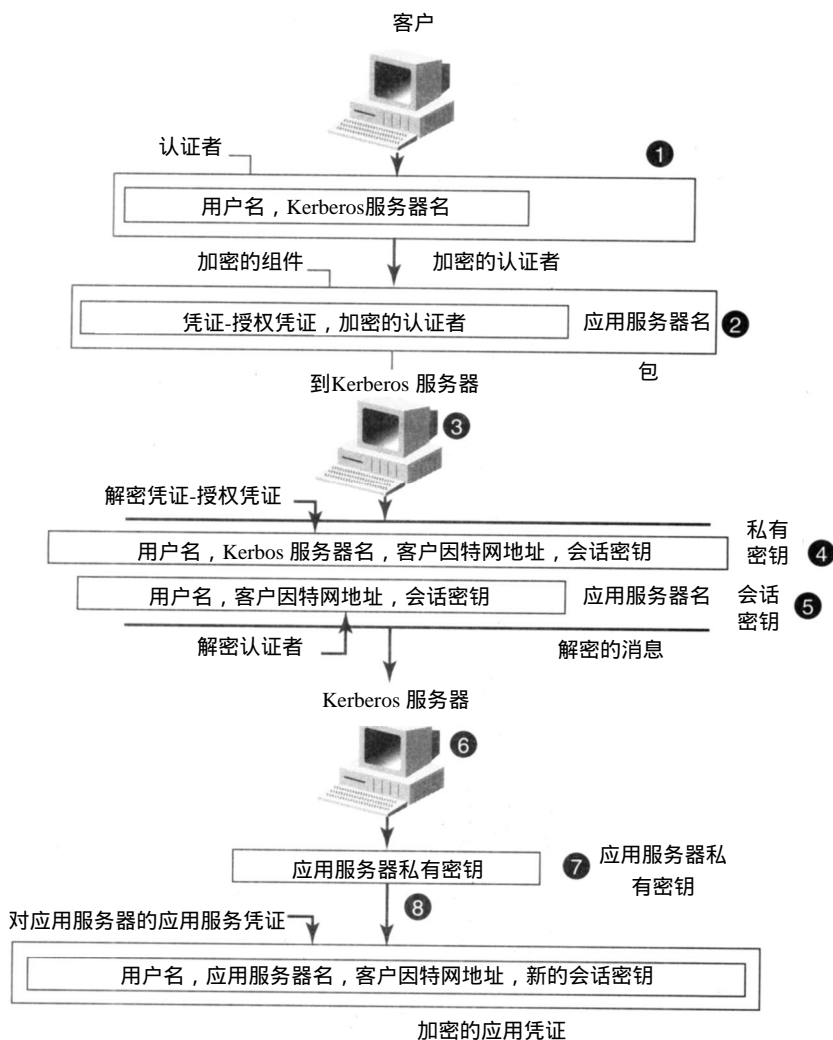


图3-28 获得用于访问应用服务器应用服务凭证

客户的因特网地址。

当前时间。

产生发送给 Kerberos 服务器的消息。这个消息包含三个部分：

凭证-授权凭证。

加密后的“认证者”。

应用服务器名。

Kerberos 服务器

用它的私有密钥解密凭证-授权凭证，以获取会话密钥。（凭证-授权凭证最初由同一密钥加密。）

使用会话密钥解密“认证者”，并比较：

凭证中的用户名与“认证者”中用户名。

凭证中的 Kerberos 服务器名与它自己的名字。

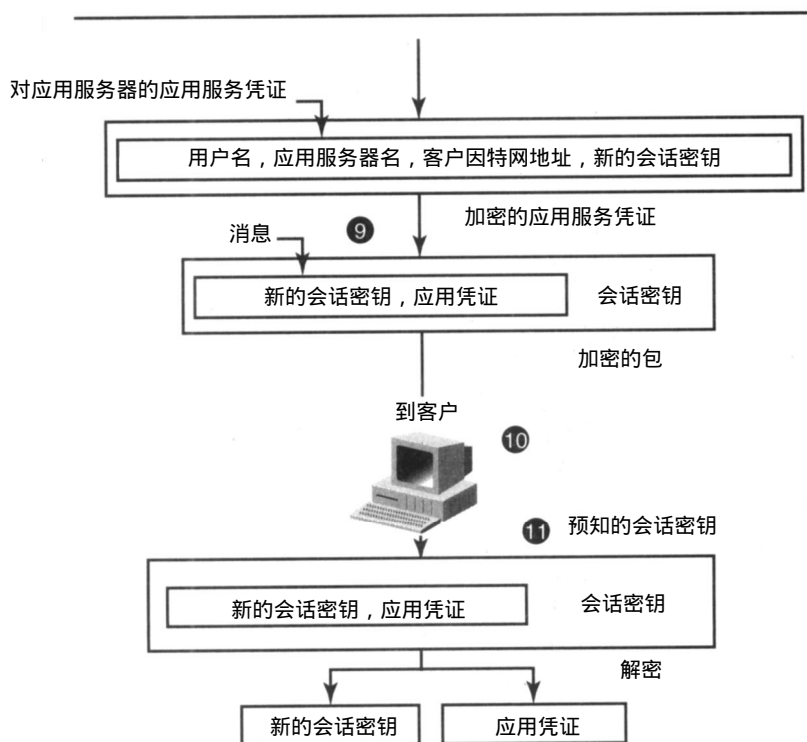


图3-29 获得用于访问一个应用服务器的凭证

凭证中的因特网地址, “认证者”的因特网地址及接收到的包中的相应地址。

“认证者”中的当前时间与服务器的当前时间, 以确认消息是授权的、未过期的 Kerberos 服务器验证凭证中的信息之后, 服务器为客户生成一个应用服务凭证包。

Kerberos 服务器:

根据消息中的应用服务器名从 Kerberos 数据库中获取应用服务器的私有密钥。

产生一个新的会话密钥, 然后再产生一个基于应用服务器名和这个新生成的会话密钥的应用服务凭证。Kerberos 服务器用应用服务器的私有密码加密该凭证。这张凭证也被称为应用凭证。该凭证与凭证-授权凭证有相同的域:

用户名。

应用服务器名。

客户的因特网地址。

新的会话密钥。

应用服务器的私有密钥。

产生一个包含有新生成的会话密钥及加密后的应用服务凭证的包; 并用客户已经接收到的会话密钥加密该包。客户已知:

新的会话密钥。

应用凭证。

将包发送给客户。

客户使用它以前收到的会话密钥解密此包。从包中的消息获取它不能解密的应用服务凭

证，以及一个新的用来与应用服务器通信的会话密钥。

一旦客户接收到这张应用服务凭证，它就可以请求服务。客户将应用服务凭证写入发送给应用服务器的认证请求。图 3-30 给出了向应用服务器请求服务的基本过程。

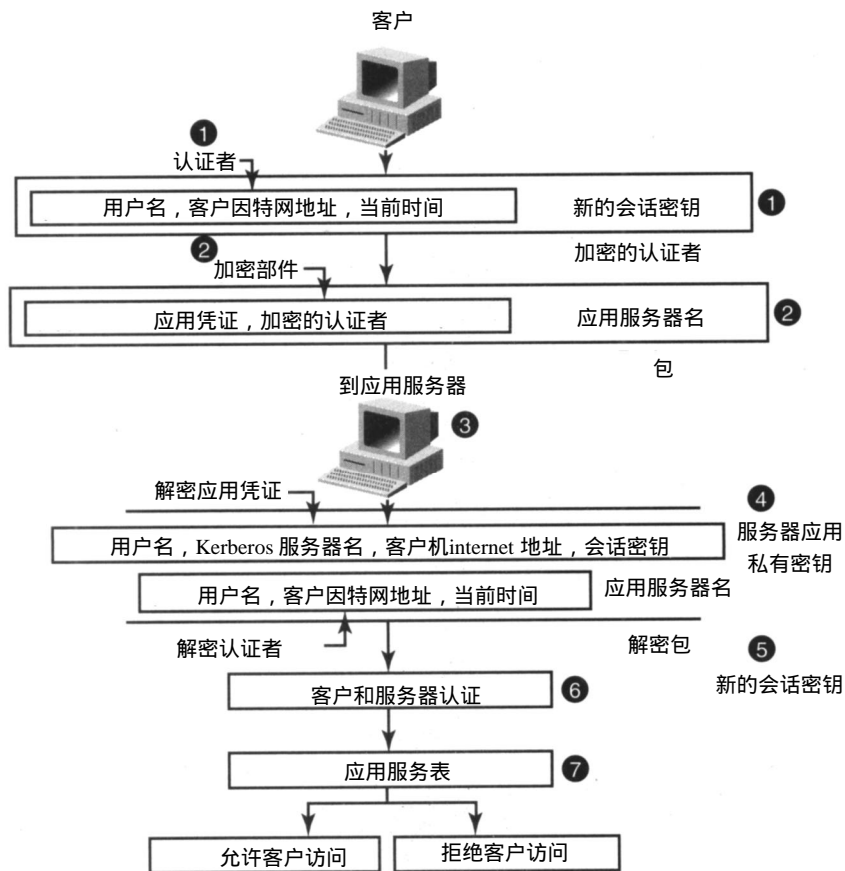


图3-30 向应用服务器请求服务

Kerberos认证摘要 在Kerberos工作过程主要有以下三步。客户：

- 1) 请求一张TGT。
- 2) 请求访问某一应用服务器时向 Kerberos服务器提交 TGT 和一个“认证者”。Kerberos 服务器为它授权一张访问该应用服务器的应用服务凭证。
- 3) 当它请求访问该应用服务器时，它向应用服务器出示应用凭证和“认证者”。服务器访问控制协议授权或拒绝客户的访问请求

Kerberos工作过程中使用凭证、“认证者”以及消息。这些元素提供了有关客户及服务器加密过的信息。在加解密凭证、“认证者”以及消息中都用到了密钥。

下文是一些关于凭证和“认证者”所需要了解的知识：

客户在访问任何应用服务器的时候，都必须出示凭证 - 授权凭证(TGT)和服务凭证。客户的所有凭证均来自于 Kerberos 服务器。

由于Kerberos服务器将这些凭证用相应的应用服务器的私有密钥加密，客户无法阅读这些凭证。每一张凭证都与一个会话密钥对应。

每一张TGT都有它的有效期(通常为8小时),而且在有效期内可以反复使用。

每当客户开始连接一个新的应用程序时, Kerberos服务器向其索要一个新的“认证者”。“认证者”有一个短的有效期(通常5分钟)。

加密凭证和“认证者”包含有客户的网络地址。另一个客户在不将其网络地址改为该客户的网址的情况下,其盗取的凭证和“认证者”拷贝是没有任何用处的。

攻击Kerberos十分困难。因为在其“认证者”失效之前,黑客想要攻击 Kerberos就必须作到:

- 截获原始凭证。

- 截获“认证者”。

- 阻止凭证及“认证者”的原始拷贝到达目标服务器。

- 将其网址改为其想侵入客户的网址。

3. Cygnus的KerbNet

KerbNet安全软件是Cygnus基于MIT的Kerberos第五版的商用软件。

由于该产品以其唯一可信赖认证服务器结构保证了 Kerberos的安全。因此,它对保护网络安全来说是一个出色的产品。唯一可信赖认证服务器结构为用户提供了单一的双方认证接口技术基础。一旦安装并配制了 KerbNet认证服务器,客户及服务器应用程序都能被“Kerberos化”,以使其能在KerbNet中正常工作。这在多用户应用环境中,能够十分方便地实现这一点。一般情况下你所要做的就是根据 Cygnus即将出台的Kerberos化新版本重置你的e-mail,FTP或Telnet。

KerbNet允许公司的家庭办公开发人员将 KerbNet认证及密码技术直接加入他们现有的客户-服务器应用程序中。KerbNet认证服务器是第一种能同时提供 UNIX, Windows NT请求服务加密凭证的Kerberos服务器。这种请求服务加密凭证技术使口令不再出现在网络上,防止了口令的电子欺骗攻击,并允许客户与服务器之间建立加密通信。

提示 想要了解更多有关KerbNet信息或下载免费拷贝,请访问Cygnus站点:

<http://www.cygnus.com/product/kerbnet-index.html>

3.6.2 密钥交换算法

Diffie-Hellman是第一位提出公开密钥算法的人士,该算法是在1976年发明。其安全性源于在有限域上计算离散对数比计算指数更困难。Diffie-Hellman算法可以用作密钥分配。让我们来看看该算法是如何工作的。也可以将该算法用以产生秘密密钥。但必须清楚,该算法不能用于加密或解密消息!

Diffie-Hellman公开密钥算法

Diffie-Hellman系统要求动态地为每一对收发方交换密钥。这种双向密钥会话对提高消息的安全性很有好处。当加密一个消息后,可以使用该方案进一步将消息解密变得更复杂。这是因为黑客必须先解密密钥然后才能解密这个消息。但必须清楚这意味着加大通信开销。

在RSA系统中,附加的通信开销被减少了。这是由于在分层模式下,正式的“可信的权力机构”“宣布”接收者的密钥固定不变,或在非正式的可信赖网络中,每个接收者被分配固定不变的密钥的缘故。

Diffie-Hellman算法可用于密钥认证,它使用的是非常简单的数学方法。其根本目的是为

了允许两个主机产生和共享一个秘密密钥。假设你想与你的合作者 (SO) 产生密钥，下面就是工作的过程：

1) 首先，你和你的合作者必须遵循“Diffie-Hellman参数”，即你们需要得到一个大于2和“基准”的素数“ p ”，一个小于“ p ”的整数“ g ”，你们可以自己生成，也可以从你们的服务器上获取。

2) 你们必须各自生成一个小于 $p-1$ 的私有数据，我们称之为“ x ”，“ x' ”。

3) 完成上述步骤后，就可以得到公开密钥“ y ”、“ y' ”

$$y = g^x \% p, y' = g^{x'} \% p$$

4) 你们互换公开密钥“ y ”，然后将交换后的数转换为私有密钥“ z ”、“ z' ”

$$z = y^{x'} \% p, z' = y'^x \% p$$

“ z ”就能作为任意加密方法的密钥，在你们二者之间进行信息交换。用数学的语言来说，通过计算：

$$z = (g^x \% p)^{x'} \% p = (g^{x'} \% p)^x \% p$$

就可以获得相同的值。上述所有的数均为正整数。

x^y 表示 x 的 y 次方

$x \% y$ 表示 x 模 y 后所得的余数

3.7 密码分析与攻击

密码分析是一种在不知道密钥的情况下破译加密通信的技术。当然如果你愿意的话，密码分析也可以是一种破译密码的技术。就如同存在大量的密码分析者一样，密码分析技术也有很多种。尽管任何想要破译密码的黑客行为都可被称为是密码分析，但是认为密码分析就是对安全的威胁是不正确的。通过密码分析可能会找到危及信息安全交换的漏洞。因此不管是什么形式的密码实际上都是一柄双刃剑。

如果一个密钥暴露在某个未加密的环境中，比如，你为了记住密钥，把它写在了某个地方，那么这个密钥就有可能泄露。如果有人发现了它，密钥就泄露了。同样，如果有人进行密码分析，密钥也有可能受到攻击。

下一节我们会讨论一些最典型的密码分析攻击方法。

3.7.1 唯密文攻击

黑客或密码分析者不知道任何消息的内容，这种情况下他只能从密文着手。

事实上，由于许多类型的消息都有固定的头格式，所以猜测出明文是很有可能。通常情况下的信件、文档都有很强的可预测性，也就是说有可能会猜测到某些密文块中包含有相同的字词。

在这里，密码分析者的目的是推导出加密该消息的密钥，这也就使得他或她能够破译同一密钥加密的其他消息。

3.7.2 已知明文攻击

在此情况下，黑客知道或能够猜测出密文某些部分所对应的明文。他们的目的是破译消

息的其余密文块。黑客有可能采取的一种手段是找出用于加密的密钥。

3.7.3 选择明文攻击

在此情况下，黑客能够得到用未知密钥加密过任一段密文；他可以选择被加密的明文，这使得他能够得到关于密钥的更多信息。因此它的目的是确定用于加密的密钥。一些加密算法特别是RSA，对选择明文攻击很脆弱。

3.7.4 自适应选择明文攻击

这是选择明文攻击的特殊情况。在这种情况下，黑客基于以前加密的结果修正它对明文的选择；这使得他能够选择较小的明文块。

3.7.5 中间人攻击

这是对加密通信和密钥交换协议进行的一种相关攻击。它是一种密钥电子欺骗。黑客能够介入诸如Diffie-Hellman系统进行密钥交换。在安全通信的两部分之间通过与每一部分进行单独的密钥交换，并强迫它们使用不同的密钥，从而达到破坏通信双方密钥的目的。该过程使得黑客掌握双方的密钥。在此基础上黑客可以解密任一个通信信息，并能用另一个密钥加密该消息，然后再将其转发给通信的另一方。糟糕的是，通信的双方仍然认为它们在进行安全通信。由于整个过程完全被分成两个部分，直到一切都太晚了之前，它们永远也不会察觉到！

一个防止中间人攻击的方法是通信双方计算密钥的某一哈希函数值，进行数字签名并将签名发送给对方。接收方验证由发送方发送的数字签名并将计算出的哈希值与签名中的哈希值进行匹配。

3.7.6 选择密文攻击

在此情况下，黑客或密码分析者不仅可以选择他想破译的密文而且还能将明文加密。一般情况下这种攻击适用于公开密钥算法，对于对称密码体制同样有效。

3.7.7 选择密钥攻击

这并不表明攻击者可以选择密钥。攻击者只有一些不同密钥之间的有关知识。这种方法有点奇特和晦涩。Bruce Schneier在其“Applied Cryptography”中的“差分与线性分析”一节中做了精辟地论述。

3.7.8 软磨硬泡密码分析

黑客使用恐吓、威胁、勒索或折磨某人以得到密钥。

提示 想要了解更多的关于密码分析、攻击方面的信息请查阅：

Bruce Schneier：“Applied Cryptography”，John Wiley & Sons出版社1994出版。

Jennifer Seberry and Josed Pieprzyk：“Cryptography:An Introduction to Computer Security”，Prentice-Hall出版社1989出版。

Man Young Rhee：“Cryptography and Secure Data Communications”，McGraw-Hill 出版社，1994出版。

M.E.Hellman and R.C.Merkle：“Public Key Cryptographic Apparatus and Method”。

1995年RSA Data Security 公司编写的 RSA FAQ (<http://www.rsa.com/faq.htm>)。

3.7.9 时间攻击

这是由Paul Kocher发明的一种新的攻击方法。该攻击方法建立在不同的 RSA取模幂方运算所花费的处理时间不同这一基础之上。在此过程中，密码分析者循环测试取模幂方运算。这与RSA、Diffie-Hellman及椭圆曲线算法有很大的相关性。

通常RSA幂方运算是根据中国剩余定理 (Chinese Remainder Theorem)来完成的。但如果不是这样的话，黑客就可以利用RSA运算的细微时间差别来恢复进行运算的数。

如图3-31所示，Southwest Texas州立大学网站给出了中国剩余定理的描述。

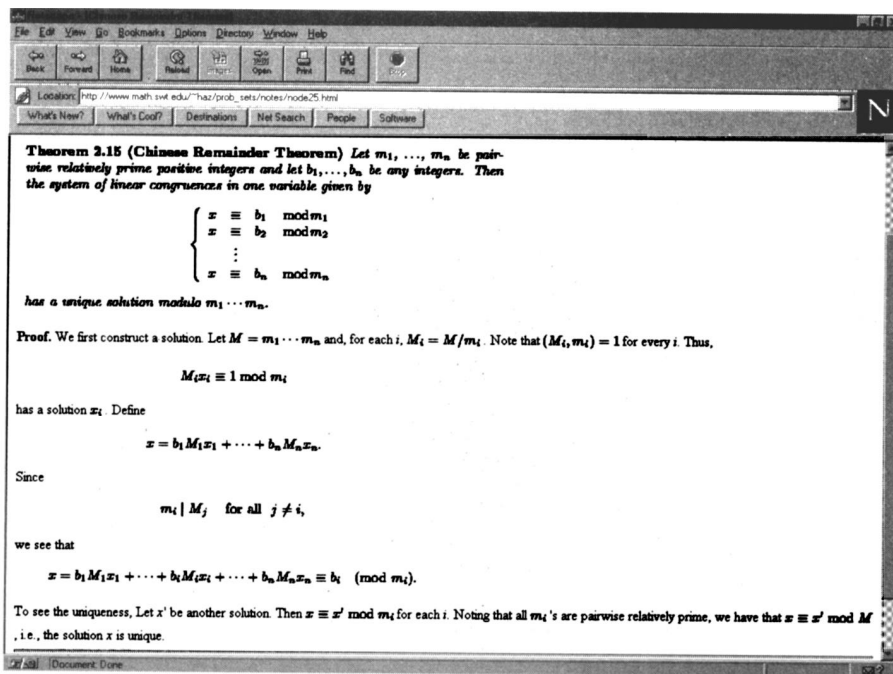


图3-31 中国剩余定理

提示 如果你想要得到更多关于中国剩余定理的信息，请访问Southwest Texas 州立大学主页：

http://www.math.swt.edu/~haz/prob_sets/notes/node25.html。

攻击者仔细观察用于分析时间“t”的“k”运算。“t”是计算取模幂方运算： $m=c^d \bmod n$ 的时间开销。攻击者已知“c”和“n”，该攻击的伪代码如下：

```
Algorithm to compute  $m=c^d \bmod n$ 
  Let  $m0=1$ .
  Let  $c0=x$ .
```

```

For i=0 upto (bits in d-1):
    If (bit i of d ) is 1 then
        Let mi+1=(mi*ci) mod n
    Else
        Let mi+1=mi.
    Let di+1=di^2 mod n.
End.

```

麻省理工学院的 Ron Rivest (rivest@theory.lcs.mit.edu) 指出，对付时间攻击最简单的方法是“确保加密运算所花销的时间不依赖于操作数。如对于 RSA，应该做到取模幂方运算的时间花销总量固定，而不是由操作数决定”。

他还提出了另一种使用“盲技术 (blinding techniques)”的方法。根据他的理论，你可以先屏蔽数据进行加密运算，然后去除屏蔽。对于 RSA 来说，这一点很容易做到（但屏蔽与去除屏蔽也需要一个固定量的时间）。这并未给出一个总的运算时间，但运算时间成了不依赖于操作数而随机改变的值。

注意 屏蔽过程将一个随机值引入了解密过程。如下：

$$m = c^d \bmod n$$

变成：

$$m = r^{-1} (cr^e)^d \bmod n$$

r 是一个随机数， r^{-1} 是它的逆

British Columbia 大学 (<http://www.ubc.ca>) 的一个网站：<http://axion.physics.ubc.ca/pgp-attack.html> 提供了对非对称密码和对称密码加密算法攻击的大量文档，这个站点很值得去浏览一下。

3.8 加密应用程序及应用程序编程接口

如果真的想要了解密码世界的发展情况，就必须了解飞速发展的信息高速公路上新技术的发展与动态，以及安全、私人通信、访问控制的需求。

当然，随着电子货币及商业信息传输变得越来越普遍，美国政府也存在同样的忧虑。尽管美国政府也确实担忧，但她的那个代表 Internet 上电子事务处理的“Clipper 芯片计划”实在令人不满。可以说这个计划是一柄双刃剑。因为它确实提供了安全的事务处理，但这一切均是在政府的控制之下。

这种情况所导致的后果是大量加密应用程序的出现、应用程序编程接口的加速发展、数据保密以及安全传输信道开始出现并得以推广。而数据保密以及安全传输信道包括了但并不局限于认证机制和安全准则。

乔治华盛顿大学 (<http://www.seas.gwu.edu/>) 的 Cyberspace 决策学院 (<http://www.cpi.seas.gwu.edu/>)，提供了大量的决策书目，这些资料为数据保护、安全通信以及它们在整个信息处理过程的实现提供了一个坚实的基础。本节将讨论它们的某些成果及它们对 Internet 安全的冲击，还有密码算法以及防火墙漏洞方面的问题。

3.8.1 数据保密及安全通信信道

因特网用户以及受保护网络的用户，都应时时保持本机构和用于交换的数据的安全性。

给出一个因特网的安全策略，使得用户必须遵守该安全策略，以及给出信息保密准则的概要，这是因特网管理员的责任。

安全策略中清楚地标明，系统管理员有责任保护好客户的个人秘密。安全策略应该界定公司数据的基本结构，例如，数据的机密程度（某些数据需要加密，另一些需要访问控制）及应用。

不论数据的原始来源，存储形式或存储地址，数据安全策略应该适用于整个公司。客户必须明白，只有所有的用户都清楚且遵循安全策略，私人秘密和信息的安全保密才有可能得以实现。因此，接收机密数据或直接下载文件都应维护数据的保密性。

同样，应用程序开发小组必须处处考虑到安全策略。即使眼前没有这种策略，不管是对 intranet 还是因特网级开发的应用程序都必须将安全方面的因素考虑进去。

在第9章“建立防火墙安全策略”中将深入讨论安全与防火墙策略。现在我们还是接着讨论认证过程，安全 API 以及它们对安全通信信道的影响。

一些数据安全策略及工具

目前市场上已经有若干种能单独使用，或与其他应用程序或设备配合使用的应用程序。如防火墙、代理服务器、路由器，它们能有效地保护你个人及金融秘密数据的安全。Pretty Good Privacy (PGP) 及 Privacy Enhancement Mail (PEM) 就是它们中的一些例子。但重要的是必须先加以说明。

你必须了解你的系统。逻辑映射还是物理映射对于发现系统内数据安全的脆弱之处是很重要的。映射对于计划将来修改系统也是很有帮助的。对将来可能的修改加以计划，可以使你确信牢牢地控制住了系统安全并能使你对所需回答的问题有更深入的理解。当为一个系统实现一项新功能，比如说一个匿名 FTP 服务（假设该站点以前不提供这项服务），那么这些控制就变得尤为重要了。

口令策略 不管你信不信，保护用户数据安全的最简单、最有效的方法是使用口令策略。它迫使 Internet/intranet 环境下的每一个人都必须得到认证。如果好的口令对数据安全是至关重要的，那么第一个受到攻击的必然是口令！原因在于：许多用户和网络系统使得攻击口令比攻击网络的其他可能的安全漏洞更容易。

由于现在存在大量的工具，如 Crack，NTCrack 以及其他更先进的用于破解口令的工具，每个人都有可能获取你的口令。很多时候，由于太大意，口令被泄露。比如，让朋友使用他们的帐号，在输入口令时被别人偷看，这些都很容易造成口令泄露。避免这一情况发生的一个最好方法，是使用包含字母和数字的独特的口令。最好这个口令在字典中找不到且别人也猜不到。

下文提供了产生一个独特的安全口令的技巧：

在口令中使用数字、但最好不要在开始处或结尾处。

在口令中使用非基本单词。外语中的基本单词也同样容易被猜测到。

使用一些击键技巧，例如当输入口令时，手指向左或向右移动一个键位。

当别人站在你背后时，请他转过身去。

使用非字母字符。诸如 \$，%，^，& 之类的符号，应该经常被用在口令中。

在某一系统中，管理员可以在口令中使用控制字符。你可以决定哪些控制字符便于跟踪和查错。

大小写字母混用。

3.8.2 认证

正如前一节所提到的，一个好的口令策略对于保证用户数据的完整性、可靠性以及安全性十分重要。特别是当涉及目前越来越成为一种时尚需求的电子商务时，它显得尤为重要。因此用户在登录其计算机、Internet或企业网时都必须进行认证。而且这一行为必须成为用户每天所必须进行的程式而不是一种特殊的、非必要的过程。

当运用认证手段时，必须考虑到电子欺骗的威胁。前面所提到的各种加密手段，有助你制订出一个不易受黑客欺骗的安全策略。但对于保护公共资源或其他非私有的相关数据、资源它就可能不够用了。

因此，必须将它与其他策略和技术一起使用，以提高公用网络的安全级。防火墙无疑就成了这样的一种需求。关于防火墙的讨论将贯穿本书的始终。

3.8.3 认证码

认证码最早出现于 Microsoft代码签名的商业软件中。认证码签名基于目前流行的工业标准：x 509v3证书标准及 PKCS # 7签名标准。

提示 关于认证码及其相关文档可以查询：<http://www.microsoft.com/intdev/security>

在 Usenet和商业杂志上有许多关于 ActiveX及认证码的相关文章。但其中大多数是关于 ActiveX控件如何操作及 Microsoft应该或是不应该推出这个工具的讨论。因此，在这里对同样的问题我不想再做更多的评论；如果要了解更多有关认证码如何、应该还是不应该之类的问题，请查阅 Alta Vista，我相信在那里你会得到满意的答案。这里，我更想讨论的是 Microsoft对认证码底层基础的构想以及认证码对基于密码学应用的数据安全的冲击。

Brent Laminack在 Usenet上提出了一些对认证码的考虑。这些考虑阐述了认证码底层的一些问题。如果你打算将数据安全建立于认证码的基础之上，请仔细权衡一下。

Laminack设想，如果有两个 ActiveX控件，一个控件提供类似于 Win98“开始”菜单（它将用户计算机所有命令以一张列表形式给出）的控制服务。假设，这一控件将所有这些命令保存在一个执行文件如 C:\window\mycommand中，该文件包含类似 Word，Excel，format c:，IE3等命令。

Laminack又设想，另一个提供“计时”功能的 ActiveX控件。该控件能在某时刻自动唤醒如备份、碎片整理等命令进行内部操作。假设它将命令列表存放到 C:\windows\mycommands。用他自己的话来说，“你就等着看它自己干吧”，确实如此。当第二个控件找到由第一个控件编写的那个文件并忠实地运行 Word，Excel，然后是……，这个命令之后将会发生什么情况？是毁灭性的后果。

现在情况如何？你完了！根据 Laminack所假设的，你没有认证码指纹标识，你已经丢失了硬盘上所有的数据。甚至，即使你有认证码指纹标识，你又能去怪谁？每一个控件都准确地完成了它所被要求完成的任务。你能去告谁？

更糟的是，任何一个控件都“没有发生错误操作”。发生在硬盘上的只是两个控件之间进行了未能预测到数据交换。Laminack认为经过仔细地分析研究，有可能设计出一个 ActiveX控

件合作组，通过合作完成预谋的非法活动。其中的每一个程序只做它“该做的事情”。但它们的整体活动就超过了它们部分活动的简单和。当然，这个 ActiveX 控件合作组的各个组成部分之间都没有明显的直接联系。

想要避免上述情况的发生，只有严格地将它们去耦。比如，可以通过为它们中的每一个都分配单独的文件存取空间，以达到相互间不允许共享任何信息，进而达到去耦的目的。但事实并非如此。

根据 Microsoft 认证码执行方案，不管从法律义务还是从技术保障上来说，当你对一个代码签字负责时，作为代码发布者的你，要对代码承担直接责任。从法律角度来说这是一个不能轻视的问题。

但是，认为对代码进行签名，就说你必须对出现的问题负责也未免太早了。毕竟，你还有可能提供审计和跟踪证明此事与你无关。就软件行业而言，历史证明，对于可能由软件造成的系统破坏，软件往往是不承担这个责任的。

3.8.4 NT 安全支持服务接口

Microsoft 的安全支持服务接口 (SSPI)，是任意分布式应用程序协议获取完整安全认证服务、消息完整性、消息保密以及优质安全服务的一个通用应用程序编程接口。应用程序协议设计者可以利用这个接口，在不改变协议本身的情况下获取不同的安全服务。

图3-32指出了 SSPI 安全服务应该在什么地方，才能与全面的分布式应用体系结构相适应。

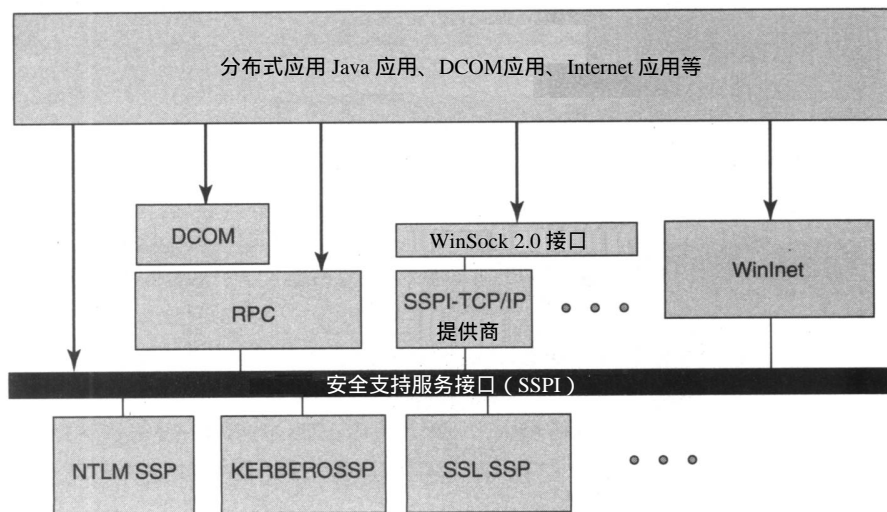


图3-32 Microsoft 的 SSPI 安全服务在分布式应用体系结构中的位置

SSPI 为传输层应用程序如 Microsoft RPC，或文件转向器以及安全提供程序如 Microsoft NT Distributed Security，提供了一个通用接口。SSPI 提供了一个机制，在此机制下，分布式应用程序在不知道安全协议细节的情况下就能从几个安全提供者那里获取认证了的连接。

SSPI 包含以下 API:

证书管理 API (Credential Management API) 提供对主体证书的访问 (如口令数据，凭证等) 或释放该访问。这些 API 包括：

AcquireCredentialsHandle 获取相关证书的句柄。

FreeCredentialsHandle 释放证书句柄及相关的资源。

QueryCredentialAttributes 允许对证书相应属性如：名称、域名等进行查询。

上下文管理API(Context Management API) 提供产生及使用安全上下文的方法。这些上下文由建立通信连接的客户端与服务器双方产生。这些上下文可以在将来与API一起使用。这些API是：

InitializeSecurityContext 通过产生一个能被传送到服务器的安全令牌初始化安全上下文。

AcceptSecurityContext 利用从客户端那里接收到的非透明消息生成一个安全上下文。

DeleteSecurityContext 释放一个安全上下文及与之相关的资源。

QueryContextAttributes 允许查询不同的上下文属性。

ApplyControlToken 将一个安全消息补充到一个已存在的安全上下文中。

CompleteAuthToken 完成一个认证令牌。因为在某些协议中，如DEC RPC，一旦传输过程中更新了消息的某些域，那么就on须修改安全信息。

ImpersonateSecurityContext 服务提供者以模拟令牌的形式将客户端上下文与呼叫线程连接。

RevertSecurityContext 服务提供者终止模拟令牌，并将呼叫线程恢复为其默认安全上下文。

消息支持API(Message Support API) 提供基于安全上下文的通信完整性以及保密服务。这些API是：

MakeSignature 产生一个基于消息和安全上下文的安全签名。

VerifySignature 验证签名是否与接收到的消息匹配。

包管理API 提供安全提供商所支持的不同的安全包服务。这些API是：

EnumerateSecurityPackages 列出可用的安全包及其容量。

QuerySecurityPacketInfo 查询单个安全包的容量。

SSPI目前不为加解密提供任何通用接口。安全提供商是指实现安全支持服务接口的一个动态链接库。安全提供商为应用程序提供一个或多个安全包。安全包将SSPI函数映射到该包所对应的具体执行协议上，如NTLM，Kerberos或SSL协议。安全包有时指的是SSP，例如NTLM SSP。安全包的名字被用于确定某一特定包的初始化步骤中。

SSPI允许应用程序在不改变接口就使用安全服务的情况下，使用系统中的任一可用的安全包。SSPI不建立登录证书，因为登录证书通常是应该由操作系统完成的授权操作。

应用程序可以使用包管理函数列出可用的安全包，然后选择一个来满足它的需求。应用程序利用证书管理函数以获取这些证书管理函数所代表的用户证书的句柄。有了这一句柄，应用程序就可以使用上下文管理函数为服务产生一个安全上下文。安全上下文是一个不透明的数据结构。它包括与连结有关的安全数据，如会话密钥、会话持续时间等等。最后应用程序将安全上下文与消息支持函数配合使用以保证在连结过程中消息的完整性与安全性。

3.8.5 Microsoft加密API

Microsoft加密API使应用程序开发人员能够向他们的Win32应用程序中添加加密功能。就像应用程序在不知道任何关于图形硬件配制的情况下就可以使用图形库一样，应用程序在不

了解CryptoAPI中的函数内部实现机理的情况下就能够使用它们。

CryptoAPI是一系列允许应用程序灵活地进行加密或数字签名的函数，同时也提供对用户敏感的秘密密钥数据的保护。

所有的加密操作都由相互独立的模块完成。这些模块被称为加密服务提供商 (Cryptographic Service Provider, CSP)。Microsoft RSA Base Provider就是一个捆绑在其操作系统中的CSP。

每一个CSP提供CryptoAPI的不同实现。其中一些还提供更强的加密算法，而另一些则是使用了一些硬件，如智能卡。另外，某些 CSP可以在某个时刻与用户建立直接通信，如使用用户的私有密钥进行数字签名时。

注意 想要了解有关CryptoAPI更多、更具体的信息，请访问

<http://www.graphcomp.com/info/specs/ms/capi.html>。

加密与防火墙：一对功能强大的好伙伴

毫无疑问，有些公司想要也确实需要因特网。对于某些因特网管理员，也许会涉及到企业内部网——一个由 Web服务器及运行在保密网上的浏览器所构成的私有 IP网络的实现。但大多数情况下，它将涉及到在因特网上寻求传输包括敏感信息在内的数据的方法。

防火墙在保护因特网上的公司站点起了很大作用。但基于路由器和极少量的拒绝 /许可状态的防火墙概念，已经不再足以将黑客与破译者拒之门外。形势不容乐观，根据 Computer Security Institute的报告，因特网上五分之一的公司已经或即将被侵入。更糟的是它们中至少有三分之一是安装了防火墙的。

第7章，“究竟什么是因特网 /内部网防火墙”详细阐述了目前市场上提供的各种防火墙产品以及它们的功能，最重要的一点是给出了它们所使用的技术。但是尽管安装了防火墙，由于数据会在防火墙之外被截获，因此数据仍有可能在因特网上被盗取，甚至还有可能被篡改。任何人都可以在截获的 e-mail电子邮件中插入恶意的 applet。一旦这个 applet被激活，它就有可能使防火墙失效甚至危及到保密网的安全。

请考虑这样一种情况：你请我开发一些使用 ActiveX的应用程序。我为你的 Internet Explorer 编写了应用程序插件，这些插件增强了你的 Internet Explorer的功能，这使得你很高兴。但是，一旦你的用户使用这些插件，我就可以以一个可信赖的开发者的身份注册到他们的 Explorer上。这也就意味着从现在开始，任何下载由我开发的插件的请求都不会触发权限对话框！它是一个漏洞还是一个功能？！还记得前些时候关于 ActiveX的讨论吗？

不幸的是这并不是一个神话，这是事实！如果你在 1997年2月左右访问 C|net的站点 <http://www.news.com/News/Item/0,4,3707,00.html>，那么同样的事情将发生在 InfoSpace身上。幸运的是，InfoSpace发言人解释这些“资源”是一个小失误，并升级了他们的插件。但存在一个问题，我们是否可以相信所有 Internet Explorer插件编写者都会像InfoSpace那样负责呢？

当一个可执行组件下载完毕，特别是在防火墙 (如果存在的话)不能阻止恶意程序访问保密网的情况下，这个组件不应该能够秘密地控制系统的安全策略。然而当面对 Microsoft的活动内容模式时，我们几乎不可能防止这种情况的发生。

Java模式比ActiveX在处理这一问题上显得更健壮，这并不是什么新闻。但是，有得必有失，如果你认为它是一个功能的话，那么 Java就失去了这一功能。

不管它是一个功能还是一个小错误，我对这一实际情况考虑更多的是：一个使用 ActiveX控件的精明的开发商，除了会为他开发的其他程序在不经认证码认证的情况下进入系统而

大开方便之门之外，我认为不会有其他太多作为。ActiveX控件甚至可以让他以自己的方式即根据标记但不引入恶意代码来访问系统。这种方式可以掩盖它在该系统中的所有行踪。

然而，由于有了ActiveX，当用户允许一个代码在其系统中运行时，将会有许多“危险”的事情会发生。在某种程度上，这不是一个只影响到ActiveX的问题。它广泛存在于各种平台与各种类型的代码之中。如果Web能使编程人员发布他们的代码变得简单，那么Web同样也能使确定一个恶意程序和警告受威胁部分以及与之进行通信变得简单。

毫无疑问，认证码为质量控制、代码认证提供了大量的帮助。我们能够迅速地确定一个代码的编写者并要求他修复程序中的小错误就是一个很好的例子。如果该编程人员拒绝修复他的代码，有许多方法可以迫使他去修复。从商业角度而言，你可以拒绝使用这一代码或从法律角度而言，你可以送他去法庭。使用上述方法中的一种就能实现认证码的一些价值。

Java的健壮性和其他Java安全程序，如Java Blocking的存在已经足以满足开发ActiveX或Java的要求。

另一个防止这一缺陷发生的可行性办法，是配合防火墙使用一个过滤器，使得那些applet(Java,JavaScript或是ActiveX对象)能被过滤掉。这中间具有代表性的工具是Java Blocking。但根据许多意见表明，它在如何使之最有效地工作的同时却又造成了大量的混乱。

我的建议是在防火墙中将Java Blocking作为一种服务运行。通过这种方式，它能够提高整个网络抵抗Java applet的攻击能力。一些浏览器如Netscape Navigator提供了客户级的防止Java applet攻击的安全性。允许用户在浏览器上使Java applet失效。但它集中管理所有客户却很难。

Hitachi Data Systems的Carl V Claunch为TIS防火墙工具包开发了补丁程序，该补丁能将TIS http-gw代理网关转换为代理过滤器。这个过滤器能在IP/域地址级上以统一或不同的安全策略加以实现。该过滤器可以阻塞、允许或根据浏览器的版本综合处理这两种情况。这个安全策略分别在Java, JavaScript, VBScript, ActiveX, SSL以及SHTTP得以实现。

就JavaScript而言，Claunch指出阻塞过程包含扫描不同的结构：

- 1 -<SCRIPT language=javascript>...</SCRIPT>
- 2 -<SCRIPT language=livescript>...</SCRIPT>
- 3 -以onXXXX形式给出的标记中的属性。这里XXXX表示浏览器的行为如：点击、鼠标移动等。

4 -根据javascript：协议扫描HREF和SRC的URL

5 -根据livescript：协议扫描HREF和SRC的URL

Java Blocking使<APPLET...>和<APPLET>标识的语句失效，但允许字符通过，这些字符通常是HTML。

对于阻塞VBScript，它包括：

- 1 -扫描及过滤 <SCRIPT language=VBScript>...</SCRIPT>之间的语句
 - 2 -扫描及过滤 <SCRIPT language=vbs>...</SCRIPT>之间的语句
 - 3 -同处理JavaScript一样清除onXXXX标记以及其他标记间的属性组
- 阻塞ActiveX包括清除<OBJECT...>...</OBJECT>之间的语句。

但是，SSL及SHTTP对话将HTML交给代理服务器，其结果是这些SHTTP及HTTPS HTML页面不能被过滤。

不要认为我会改造 ActiveX或是升级 Java！任何人只要愿意的话都可以为 Netscape 编写出恶意的插件。事实上，其带来的破坏超过了 ActiveX 对象给过滤器带来的破坏。毕竟，插件与 ActiveX 对象对 Windows 有同样大的控制能力。

不要告诉我，必须安装插件而不是自动接收 ActiveX 对象就使得插件比 ActiveX 更安全。Netscape 上众多的应用程序表明，安装了具有恶意破坏功能的插件的用户数目，与在其网页上面对恶意 ActiveX 对象攻击的用户数目不相上下。

由于专业人员涉及了网络和网站安全，还是让我们现实一些吧。许多专家指出了 Java 执行程序中的安全漏洞以及 Java 安全模型的基本问题。例如，由于运行了某些小程序，它们以最高权限执行任意代码这使得 Java 系统出现混乱。

注意 Dean, Felten 及 Wallach 编写了一篇名为 “Java Security: From HotJava to Netscape and Beyond” 的白皮书。其中讨论了许多关于 Java 的问题及安全漏洞。你可以从 Princeton 大学的站点：<http://www.cs.princeton.edu/sip> 下载这篇论文。

到目前为至，一些用户与系统开发人员仍然认为这些有关 Java 的问题只是“暂时的”，他们认为这些小错误很快会被修复，危险性会被降低。毕竟 Netscape 已经以令人不可思议的速度修复了它所存在的严重问题。

但是，这些数目巨大的可能运行 Java 的浏览器，如果它们中的每一台都被恶意 applet 所控制，那么就值得仔细考虑一下在 Java 结构的执行级上存在的安全漏洞了。

另一篇文章描述了以合法 Java applet 身份对防火墙进行攻击。该文章叙述了在某些防火墙环境下，运行于防火墙浏览器上的 Java applet 可以迫使防火墙接受诸如将 Telnet 或其他形式的 TCP 登录直接连接到主机的请求！在另一些情况下，这些源程序甚至可以利用防火墙去任意访问其它被认为是受防火墙防护的主机。（该文章可以从波士顿大学：

<http://www.cs.bu.edu/techreports/96-026-java-firewalls.ps> 那里下载。）

让我解释一下，那些暴露在这些攻击下的漏洞不是由 Java applet 本身或防火墙自身所引起的，而是两者结合的产物；对于安全模型来说，这是由于浏览器访问被认为是受到了保护的主机而引起的。

对于那些怀疑在 Web 上、尤其是在浏览器上运行的 Java applet 的安全性的人，你们可以做一个小试验（适用于 Netscape 3.0）。请访问：<http://www.geocities.com/CapeCanaveral/4016>。一旦登录到那里，请点击 Java-Jive 网页并仔细观察 “little Java devils” 的动作。

如果你没有意识到发生了什么事情，请再试一次。请注意，尽管没有得到你的许可，每次你进入这个网页，一条消息就被发送给了该网页的制作者 Francesco Iannuzzelli (ianosh@mv.itline.it)。该消息包括你的地址（用户的以及服务器的！）根据 Francesco Iannuzzelli 所说的，你没有任何办法得知这个小错误的存在。

该网页上的按钮能够被隐藏，用以显示连接你的邮件服务器的状态条也能被隐藏，这是你唯一能发现的不寻常之处。

那么该怎么办呢？加密显然就成了最佳选择。有一个好消息是，防火墙产品供应商认识到，为他们的防火墙产品提供加密支持是一个好主意。许多防火墙甚至包含了 applet 过滤器。Border Network Technologies 公司，Check Point 公司，Trusted Information System 公司就是其中的一些供应商。

路由器供应商也同样努力使其产品能够提供更高的整体网络安全保护级。Border Network

Technologies公司，Cisco Systems公司就是其中的几个供应商。

Lee Bruno于1996年4月在关于Web上数据通信的一篇文章中，提到有少量的公司已经开始提供了可独立工作的加密装置。Bruno认为：选择了正确的工具就意味着解决了一个复杂的问题。从最基本的入手：数据应该在何处被加密？一些供应商将其在应用层实现；另一些将其在IP层实现。前者使得网络管理员选择他们想要加密的东西，后者却是强迫他们加密给定连接上的所有内容。

这种事实使得加密与防火墙成了一对强有力的伙伴。请回顾一下本书第二部分“防火墙的实现与局限性”中所提到的内容，并考虑一下本章我们所讨论的有关加密及其应用的内容，然后建立你自己的安全策略。记住你应该同时使用加密和防火墙两种手段以可靠地保护网络整体安全。请阅读一下本书第三部分“防火墙资源向导”，它将帮助你使用“防火墙攻击终结者”并为你提供丰富的有关防火墙供应商、防火墙效能的资料及补充信息。