RSA 系列算法在工程中的应用研究*)

李荣森[1] 秦 杰[2] 窦文华[1]

(国防科技大学 计算机学院 长沙 410073)[1] (河南工业大学信息科学与技术学院 郑州 450052)[2]

摘 要 网络安全产品中,大都需要使用密码算法。公开密钥算法主要有 RSA 和 ECC 等。本文根据工程应用的实际情况,对 RSA 系列算法进行了深入研究,分析了不同子算法的优劣,从中选出了适合工程应用的子算法,并结合我们的项目需求提出了一些对算法的改进。

关键词 RSA,雅可比算法,蒙哥马利算法,加法-减法链,米勒罗宾测试,Agrawal -Kayal -Saxena 测试

Study on Implementing RSA Algorithm in Actual Project

LI Rong-Sen[1] QIN Jie[2] DOU Wen-Hua[1]

(College of Computer Science, National University of Defense Technicology, Changsha, 410073)[1]

(School of Information Science and Engineering, Henan University of Technology, Zhengzhou 450052)[2]

Abstract Most network-security product needs some cipher algorithm. RSA and ECC perhaps are the main public-key algorithms. This paper deeply investigates the RSA algorithm according to actual project. We discuss almost all the sub-algorithms that contained in RSA, select the most proper sub-algorithm for our project. Then we put forward some improve to these sub-algorithms according to our project.

Keywords RSA, Yacobi algorithm, Montgomery algorithm, Addition-Subtraction chains, Miller-Rabin test, Agrawal -Kayal -Saxena test

1 引言

随着网络的普及,网络安全正变得越来越重要,各种网络安全产品应运而生。这些产品涉及到的有信息保密、漏洞防护等若干个方面。其中综合使用密码算法来保证数据机密性的产品相对较多。这些产品不可避免地要实现某些密码算法,而且往往是多种算法。作为现代密码体制不可或缺的一部分,公钥密码算法在其中扮演着重要的角色[1~3]。

自从 D-H 公钥算法[4]出现后,目前已相继出现了多个公钥算法,较为著名的有 ECC[5]和 RSA[6,7]等。由于种种原因,目前的产品大多倾向于选择 RSA 算法。本文结合我们的工程应用对 RSA 系列算法进行了较为全面的分析,指出每一种子算法的优劣之处,之后选择组合最合适的子算法,并结合应用需求进行了改进。

本文后续内容安排如下:第 2 部分是 RSA 算法简介;第 3 部分对各种典型的模幂运算进行分析,并结合实际应用提出改进的蒙哥马利算法;第 4 部分首先介绍各种素数产生方法,之后给出我们的解决方案。最后是我们工作的总结。

2 RSA 算法简介

2.1 算法原理

首先选择两个大的素数 p,q,令 $n=p*q$,则 n 的欧拉函数为 $\Phi(n)=(p-1)*(q-1)$,选择一个随机数 e,使得 $(e,\Phi(n))=1$,求解 d,使得 d 满足 $e*d\equiv 1 \pmod{(p-1)*(q-1)}$,故有 $A^{e*d}\equiv A^{\Phi(n)+1} \bmod n$。根据欧拉定理,有 $A^{\Phi(n)+1}\equiv A \bmod n$,故 $A^{e*d}\equiv A \bmod n$,其中 e 和 n 就构成了公开密钥,而 d 和 n 则构成了私有密钥。(这两对数也可以反过来用,即用 e 和 n 构成私有密钥,而用 d 和 n 构成公开密钥)。

加密时计算 $c=m^e \bmod n$

解密时计算 $m=c^d \bmod n$

其中 m 是加密前的明文,c 是加密后的密文,e、d、n 就是上面说的密钥对。

2.2 构成 RSA 的各个子算法

应用 RSA 时需要解决的子问题,主要有:模幂运算、素数产生等。这些问题的困难之处在于要在有限时间内快速进行很大的数的计算,否则 RSA 将只能停留于理论。其中模幂运算是最基础的,素数产生等后继的问题都依赖于它。下面分别讨论这两个主要的子问题。

3 模幂运算

3.1 概述

模幂运算有很多种算法,从最简单的循环相加,到雅可比算法[8]、蒙哥马利算法[9],每种算法都各有千秋,要根据具体的应用环境来选择合适的算法。

普通的循环相加算法实现简单,但速度很慢,适合讲解原理演示用;Barrett 归约算法速度较快,但只对模同一个模数时有效,如果有很多个模数,则预处理的时间将会显得很长;加法链算法速度最快,但求最短的加法链本身就是 NP 问题,所以计算之前求加法链的过程很难实施;简单预处理建表法速度仅次于加法链算法,主要归功于计算时可以查表,节约了大量的计算,但同 Barrett 归约算法类似,对于每一个模数,都要预先建表,当系统中有大量不同模数的计算时,这个表将会

*)国防预先研究课题(No. 413040402)资助。李荣森 硕士生,研究方向为计算机网络、信息安全;秦 杰 博士,讲师;窦文华 教授,博士生导师。

变得很大，而且预处理的过程也会显得很长；综合加法链和预处理，滑动窗口算法和雅可比算法被提了出来(后者是前者的改进)，该种算法可有效减少模幂运算中乘法的数量，从而大幅提高运算速度；加法-减法链算法是加法链算法的另外一种引申，它以 NAF(Nonadjacent Form)为基础，实践性更强，但以逆元的快速求解为前提，故多用于 ECC 等算法中；蒙哥马利算法则从另外一方面考虑，巧妙地将一次代价很大的模乘运算转化为两次代价较小的乘法运算，所以能够对速度有很大提高，而且该算法不需预处理，不局限于同一个模数。还有其他一些算法，由于代表性不强，不再一一列举。

所以，我们采用了雅可比算法、蒙哥马利算法。

3.2 Yacobi 算法介绍

3.2.1 简单二进制算法

该算法有时也称为“乘法-平方”算法，是计算模幂运算的最基本的算法，其他的一些算法都以这个算法为基础。该算法有两个大同小异的具体版本，分别是自左向右的版本和自右向左的版本。一般情况下两个版本的效率是一样的，只在加法-减法链算法中，自左向右的版本效率会更高一些。以下只介绍自左向右的版本。

算法描述

输入：a,x,L(L 是 x 的二进制形式的长度)

输出：$c=a^x$

```
c=1;
for (i=L-1;i>=0;i--)
{
    c=c^2;
    if(x_i ==1)
        c=c * a;
}
```

3.2.2 m 进制算法

该算法是在二进制算法的基础上改进提出的，当 m 等于 2 的时候，它就又回到了二进制算法。为了加快计算速度，该算法需要预计算一些结果，从而减少乘法的数量。当计算的幂次较高时，该算法可以表现出比简单二进制算法明显的优越性。

算法描述

输入：a,x,L(L 是 x 的 m 进制形式的长度)

输出：$c=a^x$

预计算并存储 $a,a^2,a^3,\cdots,a^{m-1}$

```
c=1;
for (i=L-1;i>=0;i--)
{
    c=c^m;
    if(x_i ! =0)
        c=c * a^{x_i};
}
```

预计算过程处理：

预计算主要是构造一颗二叉树，左边的子节点代表 1，右边的子节点代表 0。构造的时候同样有自左向右和自右向左两种方案，由于大部分情况下都使用自右向左的构造方法，所以这里将采用自右向左的构造法对输入串进行扫描。

对于简单的 m 进制算法，构造的预处理二叉树将是满的，第 i 层将有 2^i 个节点(根节点是第 0 层)。取 $m=8$，则构造出来的预计算二叉树如图 1 所示。

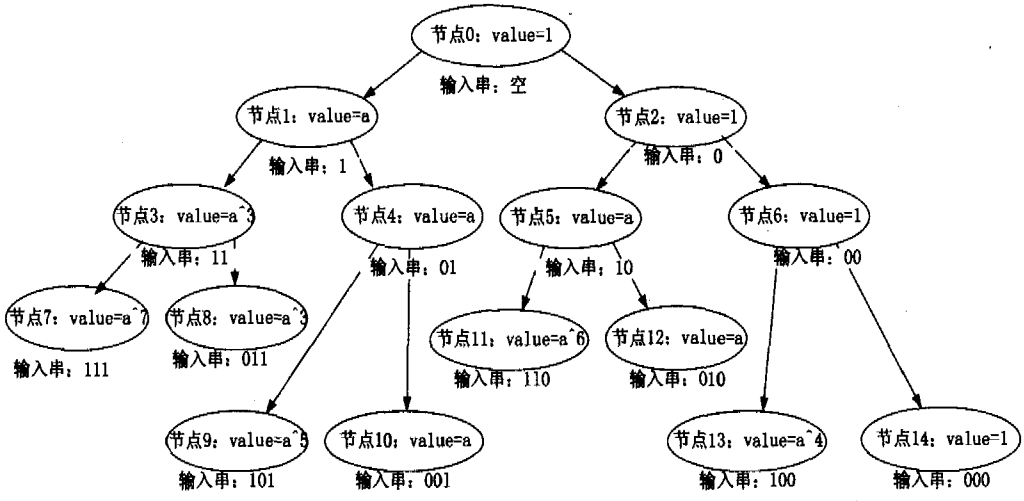

图 1 $m=8$ 的 m 进制预计算树图

可以看出，8 个叶结点分别保存了预计算结果 $a^0,a^1,a^2,a^3,\cdots,a^7$，从而在以后的计算中就可以利用这些结果，节省计算量。

3.2.2 滑动窗口算法

在上面的算法中，可以看出，计算了好多以后用不到的数据，还有一些是可以用更简单的方法计算出来的。而这些无用的计算，都是可以通过其他方法来避免的。为此，引入窗口的概念，即每次考察时处理的二进制比特的位数。如果在 m 进制算法中，$m=2^d$，则我们可以认为是窗口大小为 d 的改进二进制算法。如果我们移动窗口时，不是固定的只移动 d 个比特，而是可以移动任意多位，并且预计算时只计算奇数次幂，不计算偶数次幂，则我们就得到了滑动窗口 m 进制算法。这里移动窗口的原则是保证大小为 d 的窗口内(输入串末端的情况可以小于 d)出现的是一个奇数，并且输入串中的 1 都被计算了；对于输入串中的 0，并不进行计算，简单地把窗口滑过去。

以 $m=8(d=3)$ 为例，计算 $c=a^{5287}$

$5287=1010010100111_2$，采用从左至右的方法，每次计算的窗口及计算过程如下：

$\underset{1}{101}\underset{2}{001}\underset{3}{0}\underset{3}{111}$，下标 1、2、3 分别代表计算乘法的次序，中

间的0简单滑动过去，不在窗口里出现。实际的计算步骤为：

预计算：

$a \xrightarrow{\text{平方}} a^2 \xrightarrow{\text{乘法}} a^3 \xrightarrow{\text{乘法}} a^5 \xrightarrow{\text{乘法}} a^7$。

$c=1\to 1*a^5=a^5\to(a^5)^{2^2}=a^{20}\to(a^{20})^{2^3}*a^5=a^{165}\to$

$(a^{165})^{2^2}=a^{660}\to(a^{660})^{2^3}*a^7=a^{5287}$

中间共计算了2次乘法，10次平方，加上预计算时候的开销，总共5次乘法，11次平方。如果采用简单的m进制算法，则计算步骤划分为：$\underset{1}{101}\underset{2}{00}\underset{3}{101}\underset{4}{00}\underset{5}{111}$，共需要4次乘法，10次平方。根据3.2.1小节的预计算二叉树，可知预计算还需要3次平方、3次乘法，共7次乘法，13次平方。相比之下，滑动窗口算法具有很大的优越性。如果计算的幂次更高，则优越性还会表现得更明显一些。

3.2.3 雅可比算法

受LZ(Lempel-Ziv)压缩算法的启发，Yacobi提出了一种新的算法，即这里所说的雅可比算法。基本原理和上面的算法是相同的，主要是预计算以及窗口的处理上有较大改进。其核心思想是：如果某些形式的二进制串重复出现在了扫描序列中，则没必要再次计算它们，利用以前存储的结果就可以了。

基于上述原理，该算法和以上算法的差别主要是两点：(1)以上的算法中计算了一些以后并没有用到的中间结果；在新的算法中，这种情况将被改善，算法只计算以后肯定会用到的中间结果。(2)并不是将窗口大小固定为预先设置的一个初始值，而是根据具体情况动态调整窗口的大小和位置。

为了计算$c=a^n$，将n写成如下的形式：$n=e_k0^{z_k}e_{k-1}0^{z_{k-1}}\cdots e_10^{z_1}$，这里，$e_k$代表连续的非0序列，$0^{z_k}$代表连续$z_k$个0。此处只是写成这个形式，并不知道$e_k$或$z_k$具体的数值。然后，对于这个序列，执行如下的算法构造预计算树。(这里我们不妨设输入序列为seq)

构造预计算树函数

输入：a，n，seq

输出：构造好的预计算树

$e[0]=1$；$Z[0]=0$；$L[0]=0$；$L[-1]=0$；(因为根节点没有父节点，所以$L[-1]$不需要在根节点存储，这里写出来只是为了保持统一性)；标记根节点为#0；←(初始化根节点；)

```
在seq的最左边添加一个0;
while(seq不空)
{
    L[i]=0;Z[i]=0;
    从右向左扫描seq,将扫描进来的一位存储为ns;
    while(ns==0)
{
    Z[i]=Z[i]+1;
    从右向左扫描seq,将扫描进来的一位存储为ns;
}
while(没有到达叶节点)
{
    L[i]=L[i]+1;
    沿着构造的预计算树向下推进一步;
    从右向左扫描seq,将扫描进来的一位存储为ns;
    if(seq结束了)
    {
        记目前推进到的节点为# j号节点,
        加入一个新节点,标记为# i;
        从刚才的# j号节点引一条弧线到这个节点,弧线上标记为ns;
        e[i]=e[j];
        计算a^e[i](可直接复制# j号节点的);
        将a^e[i]和L[i-1]、Z[i]一起存储在节点# i中;
    }
}
设刚才的叶节点为# k号节点。
新加入一个新节点,标记为# i;
从刚才的# k号叶节点引一条弧线到这个节点,弧线上标记为ns;
if(ns==0)
    e[i]=e[k];
else
    e[i]=e[k]+(1≪L[i]);
L[i]=L[i]+1;
计算a^e[i];
将a^e[i]和L[i-1]、Z[i]一起存储在节点# i中;
i=i+1;
}
```

则计算$c=a^n$的程序为：

①调用构造预计算树函数

②计算

$c=((((a^{e_k})^{2^{Z_k+L_{k-1}}}*a^{e_{k-1}})^{2^{Z_{k-1}+L_{k-2}}}*\cdots)^{2^{Z_2+L_1}}*a^{e_1})^{2^{Z_1}}$

③输出c

以下是用该算法进行计算的一个具体例子。

例：计算a^{660731}

$660731_{10}=01010000101001111101l_2$

扫描划分步骤为：

$\underset{6}{01010000}\ \underset{5}{10100}\ \underset{4\ 3\ 2\ 1}{11111011}$，其中的下标1、2、…、6即为扫描时依次构造的节点。

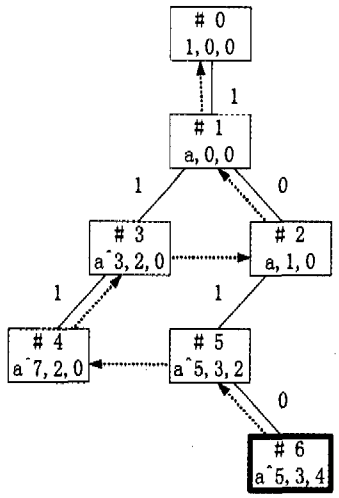

图2 雅可比算法示例

$a^{660731}=(((((a^5)^{2^{4+3}}*a^5)^{2^{2+3}}*a^7)^{2^{0+2}}*a^3)^{2^{0+2}}*a^1)^{2^{0+1}}*a^1$

使用雅可比算法，其计算量已经接近最短加法链的计算量。当然，如果仔细推敲，还是可以在此基础上提出一些小的改进的，但提高不会太大，部分研究者已经在这方面取得了一些进展[12,13]。

3.3 Montgomery算法

3.3.1 蒙哥马利算法

令m是一正整数，R和T是整数，$R>m$，$(m,R)=1$，即m和R互素，$0\leqslant T<mR$，则计算$TR^{-1} \bmod m$是有一种简便的方法。

如果m是基b长度为n的数，令$R=b^n$。则必有$R>m$，且要满足$\gcd(R,m)=1$只须满足$\gcd(b,m)=1$。如果满足$\gcd(R,m)=1$，令

$m'=-m^{-1} \bmod R$

令$U=Tm' \bmod R$，则$T+Um\equiv T \bmod m$，且$U=Tm'+kR$，$m'm=-1+lR$，k和l是整数

则

$(T+Um)/R=[T+(Tm'+kR)m]/R$

$=[T+T(-1+lR)+kRm]/R$

$=1T+km$

所以 $(T+Um)/R$ 是整数。

又$[((T+Um)/R-TR^{-1})*R] \bmod m=[(T+Um)/R*R-TR^{-1}*R] \bmod m$

$=[(T+Um)-T(1+jm)] \bmod m=[(Um-jm)] \bmod m=0 \bmod m$ （j 是一个整数）

所以 $(T+Um)/R \equiv TR^{-1} \bmod m$

又 $T<mR, U<R$,

故 $(T+Um)/R<(mR+mR)/R=2m$

则有

$(T+Um)/R=TR^{-1} \bmod m$ 或

$(T+Um)/R=(TR^{-1} \bmod m)+m$

若 $R=b^n$，则 $TR^{-1} \bmod m$ 可以通过计算两个多位数的乘法$(U=Tm', U^* m)$来实现，因为除以 R 可以通过简单的向右移位操作来完成。

3.3.2 改进的蒙哥马利算法

在上述原始蒙哥马利算法中，计算 m' 时使用了负数，但最后的计算结果却还是正整数。初看起来似乎无妨，但作为工程实现，应该尽量追求优化。所以我们考虑去掉这个负号。

其实很简单，只需将 m' 的计算公式改为

$m'=(R-m^{-1}) \bmod R$

问题即得到解决。不难证明，使用改动之后的 m'，完全不影响后续过程的计算，但却把对负号的需求解决掉了。那么这么做有什么好处呢？下面将给出分析。

首先，RSA算法的输入和输出都应该是无符号的二进制，在算法中引入一个负数符号，无疑很不方便。并且这个正负符号也只是在中间过程中使用，对最后的结果并没有太大用处。

其次，如果要引入一个负号，则所有的数据表示必然都要包含一个符号位，只有个别数据包含符号位是不可能的。这样，程序中每进行一次计算都要处理这个符号，否则就有可能出错。而实际上大部分的运算都是两个正整数之间的，根本不用考虑符号。现在为了保证正确性，却不得不考虑符号，毫无疑问会带来性能上的损失。当然，改进算法后会多出一次减法运算，但对于整个的计算量来讲，比重非常小，不会带来附加的性能损失。

再次，这个符号位必然需要用存储空间来表示。传统的补码表示法显然不适合于表示这个特殊的符号，所以我们不得不单独为这个符号位分配一些存储空间。按照常规，应该是一个字节，这就势必带来存储空间上的浪费。

所以，改进算法并去掉符号，对于工程实现是具有很大意义的。下面以列表的形式给出改进前后，算法在同一台电脑上执行时的性能差异：

	改进前	改进后	性能提升百分比%
进行一次512位模幂运算的平均时间	17.1 ms	16.8 ms	1.79
存储单位数据需要消耗的存储空间	520 bit	512 bit	1.56

从上表可以看出，改进后的算法比改进前是有较大性能提升的。将这个改进应用到实际工程中，将可以收到事半功倍的效果。

4 素数产生

4.1 算法比较

产生素数的核心问题是检验一个给定的数是不是素数。但要判定一个很大的数是不是素数，采用确定性的算法基本是不可能的，因为时间上的开销太大了。所以一般采用求取一个伪素数的办法。同样地，也存在着许多种方法，有：费马伪素数、欧拉伪素数、索洛维-斯特拉森测试、米勒罗宾测试等多种方法。前两种伪素性检验效果较差，对很多合数都会误判为素数，接下来两种方法的效果都好得多，但米勒罗宾测试相比之下速度更快，所以也有更多的系统采用这种方法来产生素数，包括我们的系统。

2002年，印度的3位科学家提出了 Agrawal-Kayal-Saxena 测试[14]，这是一个多项式时间内的测试算法，而且是确定性的，而前面几个算法都不是确定性的。但是，这个算法虽然证明了素性判定是一个 P 问题而不再是 NP 问题，具有重大的理论价值，但它却不适合于工程上应用。该算法的空间开销是巨大的，时间开销也不小，所以我们没有采用此算法。下面以 C 伪代码的形式给出算法轮廓：

```
(1)  Input: Integer n > 1
(2)  if (n has the form a^b with b > 1) then output
     COMPOSITE;
(3)  r := 2;
(4)  while (r < n){
(5)  if (gcd(n,r)is not 1)then output COMPOSITE;
(6)  if (r is prime greater than 2)then {
(7)  let q be the largest factor of r-1;
(8)  if (q > 4sqrt(r) log n)and (n^((r-1)/q) is not 1 (mod
     r))then
(9)  break;
(10) }
(11) r := r+1;
(12) }
(13) for a := 1 to 2sqrt(r)log n {
(14) if ( (x-a)n is not (x^n-a)(mod x^r-1,n))then
     output COMPOSITE;
(15) }
(16) output PRIME。
```

很容易看出，这是一个多项式时间内的算法，但其时空开销却非常巨大，以至于在目前的任何一台电脑上实现基本上都是不可能的。该算法的价值主要在于证明了素性判定是一个 P 问题。但立足于工程实现，我们还是能从中吸取到一些有益的思想的，下面会提到这一点。

4.2 Miller-Rabin 测试算法

Miller-Rabin 算法是一个容易且广泛使用的简单算法。它基于 Gary Miller 的部分想法，由 Michael Rabin 发展。事实上，这是在 NIST（美国国家标准和技术研究所）的 DSS 建议中推荐的算法的一个简化版。算法的描述如下。

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

(1)选择一个小于 p 的随机数 a。

(2)设 $j=0$ 且 $z=am \bmod p$。

(3)如果 $z=1$ 或 $z=p-1$，那么 p 通过测试，可能是素数。

(4)如果 $j>0$ 且 $z=1$,那么 p 不是素数。

(5)设 $j=j+1$。如果 $j<b$ 且 $z\neq p-1$,设 $z=z2 \bmod p$,然后回到第(4)步。如果 $z=p-1$,那么 p 通过测试,可能是素数。

(6)如果 $j=b$ 且 $z\neq p-1$,那么 p 不是素数。

据测试[16,17],该算法一次测试错误的可能性小于 $4n2^{(k/2^{(1/2)})}$ 分之一。如果 n 为一个256位的数,则六次测试的错误可能性小于 2^{51} 分之一。我们系统中的长度是512,则其错误的可能性就更小了,会提高不止一个数量级。这个数据要远好于最初理论推导的结果,也许这也是该算法更实用的原因。

4.3 综合的考虑

米勒罗宾测试准确性很高,但代价比较大。如果对每一个数都直接进行这种测试,则产生一个随机素数的耗费将会非常的大。但我们可以借鉴 Agrawal -Kayal -Saxena 测试中第二步的做法,即并不是对每一个数都直接用米勒罗宾测试,而是先确保这个数不能被一些小的素数(如2、3、5、7等)整除,然后再进行米勒罗宾测试,就可以既保证准确性,又提高素数求解的速度。具体操作时还可以采用"筛"法[18],以进一步提高速度。

我们的实践表明,预先进行一些测试,可以排除80%以上的合数。从而可以将素数求解的速度提高至少5倍。

总结 密码运算被越来越多的产品所采用,RSA算法在其中占有重要的地位。构成RSA的子算法种类非常多,这些算法各自都有自己的优缺点,分别适用于不同的场合。我们从软件实现以及工程应用的角度提出了最为合适的算法进行组合,希望对从事类似工作的人有一些帮助。如果应用于其它环境,比如嵌入式(CPU能力低,内存空间小,功耗有限制)等,则应选择其他一些算法进行组合,以保证系统的最优化。

参 考 文 献

1 CCITT, Recommendation X. 509. The Directory-Authentication Framework. Consultation Committee, International Telephone and Telegraph, International Telecommunications Union, Geneva,1989

2 Diffie W. The First Ten Years of Public-Key Cryptography. Proceedings of the IEEE, 1988,76(5):560~577

3 Jin T. Care and Feedingof Your Three-Headed Dog. Document Number TAG-90-012, Hewlett-Packard, May 1990

4 Diffie W, Hellman M E. New directions in cryptography. IEEE Transactions on Information Theory, 1976,22:644~654

5 Menezes A. Elliptic Curve Public Key Cryptosystems, Kluwer Academic Publishers, 1993

6 Rivest R L, Shamir A, Adleman L M. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM, 1978, 21(2):120~126

7 Rivest R L, Shamir A, Adleman L M. On Digital Signatures and Public-Key Cryptosystems: [Technical Report]. MIT/LCS/TR-212. MIT Laboratory for Computer Science, Jan. 1979

8 Yacobi Y. Exponentiating Faster With Addition Chains. In: Damgard I B. ed. Advances in Cryptology-Proceedings of Eurocrypt' 90 LNCS, Vol 473:222~229

9 Montgomery P L. Modular multiplication without trial division, Mathmetics of computations, 1985,44(70):519~522

10 Brickell E F, Gordon D M, McCurley K S, et al. Fast exponentiation with precomputation. In: Advances in Cryptology - Proceedings of Eurocrypto' 92, Vol 658. Springer-Verlag, 1992. 200~207

11 Lim C H, Lee P J. More flexible exponentiation with precomputation. In: Advances in Cryptology - Proceedings of Crypto' 94, Vol 839, 1994. 95~107

12 Yang W C, Hsieh P Y, Laih C S, Efficient Squaring of Large Integers. The Institute of Electronics Information and Communication Engineers (IEICE) Transactions on Fundamentals, 2004, E87-A(5)

13 Joye M, Yen S M. Optimal left-to-right binary signed-digit recoding. IEEE Transactions on Computers, 2000,(7):740~748

14 Agarwal M, Kayal N, Saxena N. PRIMES is in P. Annals of Mathematics, 2004,160:781~793

15 Goldwasser S, Kilian J. Almost all primes can be quickly certified. In: Proc. Annual ACM Symposium on the Theory of Computing, 1986. 316~329

16 Damgard I B, Landrock P. Improved Bounds for the Rabin Primality Test. In: Ganley M J, ed. Cryptography and Coding III, Oxford: Clarendon Press, 1993. 117~128

17 Damgard I B, Landrock P, Pomerance C. Average Case Error Estimates for the Strong Probable Prime Test. Mathematics of Computation, 1993, 61(203):177~194

18 Lenstra A K, Lenstra H W Jr. eds. Lecture Notes in Mathematics 1554: The Development of the Number Field Sieve, Springer-Verlag, 1993

19 Selby A, Mitchell C. Algorithms for software implementations of RSA. IEEE Proceedings, 1989,136(3): 166~170

20 Schneier B. Applied Cryptography. Second Edition. Protocols, Algorithms, and Source Code in C (cloth), John Wiley & Sons, Inc, 01/01/96

(上接第67页)

成具有某些特征变化的攻击数据包,这些数据包对于测试评估IDS、Firewall等网络安全设施的性能具有重要意义。因此,本文提出的网络攻击流量生成器的设计思想在理论和实践上都具有可行性和实际意义。

结束语 网络攻击流量的仿真生成对于网络安全防护设施的性能测试以及网络攻击行为攻击效果的评估具有重要意义。网络攻击流量的仿真生成最重要的是要保持原始网络攻击的行为特征。本文研究了一种网络攻击流量生成器的设计思想与实现方法。提出了利用有限状态自动机对网络攻击过程进行建模描述的方法。实验结果表明本文所设计的网络攻击流量生成器可以忠实地再现原始网络攻击的行为特征,证明本文提出的设计思想是切实可行的。当前的网络攻击流量生成器还是试验性的,已建模的网络攻击行为还不丰富,网络攻击模型的构建需要人工参与,还不能自动完成。这些都需要我们在以后的工作中进一步完善。

参 考 文 献

1 Templeton S, Levitt K. A Requires Provides Model for Computer Attacks. In: Proceedings ACM Workshop on New Security Paradigms, 2001

2 Tao Wan, Xue Dong Yang. IntruDetector: A Software Platform for Testing Network Intrusion Detection Algorithms. In: Proceedings, IEEE 17th Computer Security Applications Conference, 2001

3 Jonckheere E, Shah K, Bohacek S. Dynamic Modeling of Internet Traffic for Intrusion Detection. In: Proceedings American Control Conference, 2002

4 Sommers J, Barford P. Self-configuring network traffic generation. In: Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement, 2004

5 Sommers J, Yegneswaran V, Barford P. A framework for malicious workload generation. In: Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement, 2004

6 Nessus. http://www.nessus.org, 2005

7 Barford P, Crovella M. Generating Representative Web Workloads for Network and Server Perfromance Evaluation. In: Proceedings of ACM SIGMETRICS, 1998

8 Mutz D, Vigna G, Kemmerer R. An Experience Developing an IDS Simulator for the Black-Box Testing of Network Intrusion Detection Systems. In: Proceedings of ACSAC, 2003

9 Schiffman M. libnet. http://packetfactory.net/libnet/, 2005