融入中国剩余定理及Montgomery算法的快速RSA算法研究

于丽丽，王丽君
（辽宁科技大学 软件学院，辽宁 鞍山 114051）

摘 要：利用中国剩余定理和Montgomery模乘算法的思想，改进了RSA密码体制。改进后的中国剩余定理算法在时间效率上有较大提高，而且加入Montgomery模乘算法使模乘速度及安全性都有较大的提高，更加适合于高速的RSA密码体制。

关键词：RSA密码体制；中国剩余定理；Montgomery模乘算法

中图分类号：TP309.2 **文献标识码**：B

Integration into the Chinese remainder theorem and Montgomery algorithm for fast RSA algorithm

YU Li Li, WANG Li Jun
(College of Software Engineering, Liaoning University of Science and Technology, Anshan 114051, China)

Abstract: Make use of the idea of Chinese remainder theorem and Montgomery modular multiplication algorithm to improved RSA cryptosystem: on the one hand, improved after the Chinese remainder theorem algorithm for time efficiency has improved greatly, the speed increase is about 4 times more than before; the other hand, by joining the Montgomery modular multiplication algorithm for modular multiplication so that we have had good speed and security improvements, and more suitable for high-speed RSA cryptosystem.

Key words: RSA cryptosystem; Chinese remainder theorem; Montgomery modular multiplication algorithm

RSA算法是一种公开密钥算法，其加密密钥和算法本身都可以公开，解密密钥则归用户私人拥有。从诞生那天起，RSA就因为安全强度高、使用方便等卓越性能而受到关注，并得到广泛应用。但是，近年来由于分解大整数的能力日益增强，为保证RSA密码体制的安全性总是要增加模长，使加密、解密操作需要计算的位数达十进制百位以上的模幂乘函数，使运行时间很长，这一直是制约其应用的瓶颈问题，因此本文在速度方面对RSA算法做了相关改进。

本文主要针对RSA公钥密码体制中中国剩余定理的使用以及大整数模指数算法进行了深入的研究。经证明，在RSA算法中使用中国剩余定理可以提高其加密的速度达到之前的4倍。模指数运算是大部分密码算法实现中最耗时的运算步骤，因此模指数运算正是这些密码体制实现的关键，而大整数模运算又是其核心运算。本文将对密码学中的大整数模运算进行研究。

1 利用中国剩余定理提高RSA算法效率

中国剩余定理(CRT)又称为孙子定理，是数论中最有用的定理之一，并在我国科学技术史上对数学的发展作出了重要贡献。

中国剩余定理可有几种不同的表示形式，这里给出其中最有用的一种表示形式，即：

$$M=\prod_{i=1}^{k} m_i$$

其中m_i是两两互素的，即对$1\leqslant i,j\leqslant k, i\neq j$有$\gcd(m_i, m_j)=1$。可将$Z_M$中的任一整数对应一个$k$元组，该$k$元组的元素均在$Z_{mi}$中，这种对应关系可表示为：

$$A\leftrightarrow(a_1, a_2, \cdots, a_k)$$

其中$A\in Z_M$，对$1\leqslant i\leqslant k$，$a_i\in Z_{mi}$，且$a_i=A \bmod m_i$。中国剩余定理的用途之一是给出了一种方法，使非常大的数对M的模运算转化到更小的数上运算，当M为150位或150位以上时，这种方法非常有效。

1.1 加速算法

在 RSA 加密算法中，$n=pq$（p，q 为大素数），因此，在试图计算 $c\equiv m^d(\mathrm{mod}n)$ 前，可以先计算：

$$c_p\equiv m^d(\mathrm{mod}p)\qquad c_q\equiv m^d(\mathrm{mod}q)$$

而这两步运算，又因为模数的变小，可以简化为：

$$d_1\equiv d(\mathrm{mod}p)\qquad d_2\equiv d(\mathrm{mod}q)$$

$$c_p\equiv m^{d_1}(\mathrm{mod}p)\qquad c_q\equiv m^{d_2}(\mathrm{mod}q)$$

然后，根据中国剩余定理，计算出 c 为：

$$c\equiv c_p(q^{-1}\mathrm{mod}p)q+c_q(p^{-1}\mathrm{mod}q)p(\mathrm{mod}n)$$

在这个算法中，假设 n 的二进制长度为 k，那么 p，q 的长度约为 $k/2$。假设 d_1，d_2，$p^{-1}(\mathrm{mod}q)$，$q^{-1}(\mathrm{mod}p)$ 都已知，则整个计算过程约需要 $3k^3/8$ 次位运算。而不用中国剩余定理，则约需要 $3k^3/2$ 次位运算。其中的假设条件，由于 RSA 系统一经建立，所涉及参数就相对稳定，因而是可以实现的。所以新算法可以切实地将 RSA 系统的运算速度提高约 4 倍。加之计算 c_p、c_q 的过程相对独立，满足并行计算的条件，因此基于中国剩余定理的改进算法在速度上有着明显的优势。

1.2 出错攻击

用中国剩余定理加速计算 RSA 体制时，如果由于某些意外满足以下 3 个条件：

(1)签名(加密)信息已知；

(2)在签名(加密)时出现了一个错误；

(3)错误的签名(密文)被输出。

那么，这个错误就可能导致参数 n 被分解。

假设，设备在计算 c_p 时发生了错误，即获得了 c_p'，$c_p'\neq c_p$，而 c_p 计算正确。则通过 c_p' 和 c_p，可以计算出一个错误的 c'。由此可以得到：

$$q=\gcd((c')^e-m(\mathrm{mod}n),n)$$

证明：

根据中国剩余定理有

$$c'\equiv c_p'(q^{-1}(\mathrm{mod}p))q+c_q(p^{-1}(\mathrm{mod}q)p)(\mathrm{mod}n)$$

又因为 $n=pq$，所以

$$(c')^e\equiv(c_p'(q^{-1}(\mathrm{mod}p))q+c_q(p^{-1}(\mathrm{mod}q)p)(\mathrm{mod}n))^e(\mathrm{mod}q)$$
$$\equiv(c_q)^e(\mathrm{mod}q)\equiv m(\mathrm{mod}q)$$

而 $(c')^e\equiv(c_q)^e(\mathrm{mod}p)$，即 $((c')^e-m(\mathrm{mod}n))(\mathrm{mod}p)\neq 0$。等式成立。

从应用角度来看，出错攻击是一个非常强大的攻击方法。它提醒在设计系统的过程中，必须小心谨慎。因为，只要发生了一个错误，那么整个系统将会面临崩溃的危险。而事实上硬件电子设备出错是时常出现的，所以这种威胁极其现实和紧迫。

对于签名或者认证服务商，还存在一种被称为“否定服务”的攻击。一个攻击者只需要匿名的声称，就能获得服务商的一个错误签名，进而将迫使服务商放弃目前的密钥对。

而对于真正的攻击者，有许多已知的方法可以帮助攻击者实现诱导系统出错，包括重写 ROM、修改 EEPROM、破坏逻辑门电路等。

1.3 改进方法

几种比较实用的，对抗出错攻击的方法有：

(1)冗余计算 c_p，c_q 的值。对 c_p，c_q 的值各做 2 次计算，如果 2 次结果不同，则废止此次运算；如果结果相同，则计算 c 值并输出。

(2)增加对 c 的验证环节。在计算获得密文(签名)c 之后，验证 $m=c^e(\mathrm{mod}n)$ 是否成立。若不成立，则废止；若成立，则输出结果 c。

(3)Shamir 在 1997 年提出了一种验证模幂运算的方法[1]。具体到这个问题中，需要随机选取 r，并计算：

$$c_{pr}\equiv m^{d(\mathrm{mod}\varphi(pr))}(\mathrm{mod}pr)\qquad c_{qr}\equiv m^{d(\mathrm{mod}\varphi(qr))}(\mathrm{mod}qr)$$

如果 $c_{pr}\equiv c_{qr}(\mathrm{mod}r)$，则认为计算过程正确，进而计算：

$$c\equiv c_{pr}(q^{-1}(\mathrm{mod}p)q+c_{qr}(p^{-1}\mathrm{mod}q)p(\mathrm{mod}n)$$

否则，废止此次运算。

事实上，目前已经有很多将此加速算法应用到密码产品中的想法，参考文献[2]就是一个例子。只是它使用了混合半径转换，进一步降低了运算强度，即将

$$c\equiv c_p(q^{-1}\mathrm{mod}p)q+c_q(p^{-1}\mathrm{mod}q)p(\mathrm{mod}n)$$

替换为

$$c\equiv c_p+(c_q-c_p)(p^{-1}\mathrm{mod}q)p(\mathrm{mod}n)$$

这是因为：

$$c\equiv c_p(\mathrm{mod}p),c\equiv c_p+(c_q-c_p)\times 1(\mathrm{mod}q)$$

但是，混合半径 r 的使用，本身并不能回避出错攻击。当 c_p 被错误地计算成 c_p' 时，依旧无法进行有效的验证与更正。因此，这一计算同样会受到出错攻击，需要对其步骤加以调整，增加输出前的验证。由于是硬件实现的问题，因此更适合采用方法(3)。方法(1)意味着运算量加倍，而方法(2)则会需要全程储存被加密信息，方法(3)随机参数的获得可以增加运算过程的随机性，同时验证步骤只是简单的位比对，更加适合硬件实现。

2 Montgomery 模乘算法

在没有出现 Montgomery 模约减算法之前，研究人员用除法求余数的方法进行模约减、模乘等算法，即经典的模约减算法。Montgomery 模约减无需使用经典模约减算法而能有效地实现模乘法的技术。

设 m 是正整数，R 和 T 是整数，满足 $R>m$，$\gcd(m,R)=1$，$0\leqslant T\leqslant mR$。Montgomery[3]于 1985 年提出一种可以不使用经典乘法计算 $TR^{-1}\mathrm{mod}m$ 的方法。$TR^{-1}\mathrm{mod}m$ 称为 T 模 m 关于 R 的 Montgomery 约减。选择适当的 R，Montgomery 约减可以有效地计算。

如果将 m 表示成 n 位的 b 进制数，那么 R 的典型选择是 b^n。条件 $R>m$ 显然满足，但条件 $\gcd(b,m)=1$ 当且仅当 $\gcd(b,m)=1$ 时才能满足。因此这种 R 的选择并非对所有的模数都是可行的。对于 RSA 算法来说，m 是奇

数，因此 b 可以取 2 的幂，且 $R=b^n$。

引理 1 m 和 R 是整数，满足 $\gcd(m,R)=1$。令 $m'=-m^{-1}\bmod R$，T 是满足 $0\leqslant T\leqslant mR$ 的正整数。若 $U=Tm'\bmod R$，则 $(T+Um)/R$ 是整数，且 $(T+Um)/R\equiv TR^{-1}(\bmod m)$。

证明：$T+Um=T(\bmod m)$，因此 $(T+Um)/R^{-1}\equiv TR^{-1}(\bmod m)$。为了说明 $(T+Um)R^{-1}$ 是整数，观察到对于某些整数 k 和 1，有 $U=Tm'+kR$ 和 $mm'=-1+lR$，于是 $(T+Um)/R=(T+(Tm'+kR)m)/R=(T+T(-1+lR)+kRm)/R=lT+km$。证毕。

$(T+Um)R$ 是对 $TR^{-1}\bmod m$ 的估计。因为 $T<mR$ 且 $U<R$，所以 $(T+Um)/R<(mR+mR)/R=2m$。因此有 $(T+Um)/R=TR^{-1}\bmod m$ 或 $(T+Um)/R=(TR^{-1}\bmod m)+m$。即 $TR^{-1}\bmod m$ 或者等于 $(T+Um)/R$，或者等于 $(T+Um)/R-m$。

如果所有的整数都表示成 b 进制，并且 $R=b^n$，那么 $TR^{-1}\bmod m$ 可以采用算法 1，用两个多精度乘法（$U=Tm'$ 和 Um）、一次多精度加法（$T+Um$）和一次右移（即除以 R）计算出来。

算法 1 Montgomery 模约减算法

输入：整数 $m=(m_{n-1}\cdots m_1m_0)_b$，$\gcd(m,b)=1$，$R=b^n$，$m'=-m^{-1}\bmod R$，$T=(t_{2n-1}\cdots t_1t_0)_b<mR$；

输出：$TR^{-1}\bmod m$。

算法步骤为：

(1)$A\leftarrow T$；

(2)$U\leftarrow Tm'\bmod R$；

(3)$A\leftarrow T+Um$；

(4)$A\leftarrow A/b^n$；

(5)**如果** $A\geqslant m$，**则** $A\leftarrow A-m$；

(6)**返回**(A)。

显然，在两次多精度乘法中，算法 1 需要进行 $2n^2$ 次单精度乘法。1990 年，Dusse 等人[4]改进了算法 1，他们将步骤(2)和(3)巧妙地融合到一起，新的算法只需要 $n(n+1)$ 即 n^2+n 次乘法，见算法 2。

算法 2 改进的 Montgomery 模约减算法

输入：整数 $m=(m_{n-1}\cdots m_1m_0)_b$，$\gcd(m,b)=1$，$R=b^n$，$m'=-m^{-1}\bmod b$，$T=(t_{2n-1}\cdots t_1t_0)_b<mR$；

输出：$TR^{-1}\bmod m$。

算法步骤为：

(1)$A\leftarrow T$；

(2)**对于** i **从** 0 **到** $n-1$，**执行**：

①$u_i\leftarrow a_im'\bmod b$；

②$A\leftarrow A+u_imb^i$。

(3)$A\leftarrow A/b^n$；

(4)**如果** $A\geqslant m$，**则** $A\leftarrow A-m$；

(5)**返回**(A)。

算法 2 没有像算法 1 那样要求 $m'=-m^{-1}\bmod R$，而是要求 $m'=-m^{-1}\bmod b$。步骤(1)中，当 $i=l$ 时，A 具有性质 $a_j=0$，$0\leqslant j\leqslant l-1$。步骤(2)并没有修改这些 0 的值，但是将 a_l 替换成了 0。于是在步骤(3)中，A 能被 b^n 整除。

在步骤(3)中，A 的值等于 T 加上 m 的某个倍数（见步骤(2)）；这里 $A=(T+km)/b^n$ 是一个整数，并且 $A\equiv TR^{-1}(\bmod m)$。剩下的就是说明 A 小于 $2m$，因此在步骤(4)中，一次减法就足够了。回到步骤(3)中，$A=T+\sum_{i=0}^{n-1}u_ib^im$。但 $\sum_{i=0}^{n-1}u_ib^im<b^nm=Rm$，$T<Rm$，因此 $A<2Rm$。回到步骤(4)（A 除以 R 以后），$A<2m$ 成立。

算法 2 的步骤(1)和(2)总共需要 $n+1$ 次单精度乘法，由于步骤(2)需要执行 n 次，因此单精度乘法的总数是 $n(n+1)$。此外，还需要少量可以忽略不计的加法和移位等运算，而不需要任何单精度除法。

2.1 SOS 模乘法算法

Separated Operand Scanning 算法简称 SOS 算法，是模乘法运算中最基本的方法。即先用传统多精度乘法或其他方法进行多精度乘法运算，然后用算法 2 进行模约减运算。算法 3 给出了 SOS 算法的描述。

算法 3 SOS 模乘算法

输入：整数 $m=(m_{n-1}\cdots m_1m_0)_b$，$x=(x_{n-1}\cdots x_1x_0)_b$，$y=(y_{n-1}\cdots y_1y_0)_b$，$\gcd(m,b)=1$，$R=b^n$，$m'=-m^{-1}\bmod b$；

输出：$xyR^{-1}\bmod m$。

算法步骤为：

(1)$A\leftarrow xy$；

(2)**对于** i **从** 0 **到** $n-1$，**执行**：

①$u_i\leftarrow a_im'\bmod b$；

②$A\leftarrow A+u_imb^i$。

(3)$A\leftarrow A/b^n$；

(4)**如果** $A\geqslant m$，**则** $A\leftarrow A-m$；

(5)**返回**(A)。

通过算法 3 可以看出，SOS 算法仅仅是将乘法和模约减算法前后罗列，组合到一起，并没有做实质性改进。该算法效率相对不高，其单精度乘法的次数为 $2n^2+n$。但是，由于步骤(1)相对于其他步骤是独立的，因此，在某些特定的环境下，可以采用 Karatsuba 算法、Comba 算法等快速乘法来提高算法 3 的效率。

2.2 CIOS 模乘法算法

Coarsely Integrated Operand Scanning 算法简称 CIOS 算法，是 SOS 算法的一个改进。Koc 等人给出了 CIOS 的算法描述，并认为 CIOS 算法是最理想的模乘法算法。该方法将乘法和模约减算法完美地融合为一体，在读写内存等方面节省了许多的资源。算法 4 给出了 CIOS 算法的描述。

算法 4 CIOS 模乘法算法

输入：整数 $m=(m_{n-1}\cdots m_1m_0)_b$，$x=(x_{n-1}\cdots x_1x_0)_b$，$y=(y_{n-1}\cdots y_1y_0)_b$，$\gcd(m,b)=1$，$R=b^n$，$m'=-m^{-1}\bmod b$；

输出：$xyR^{-1}\bmod m$。

算法步骤为：

(1)$A\leftarrow 0$；

(2)对于 i 从 0 到 $n-1$,执行:

①$A \leftarrow A+x_i y$;

②$u_i \leftarrow a_0 m' \bmod b$;

③$A \leftarrow (A+u_i m)b$。

(3)如果 $A \geqslant m$,则 $A \leftarrow A-m$;

(4)返回(A)。

算法 4 实质是将 $x\times y$ 和 $U\times m$ 两个多精度乘法按照传统乘法的方法结合到了一起。在步骤(2)开始的时候,U 是未知的,需要在进行每个 $u_i\times m$ 之前用 $x_i\times y_0$ 的值计算出 u_i。最终,步骤(2)交替完成了将所有的 $x_i\times y$ 和 $u_i\times m$ 累加到 A 的过程。

CIOS 模乘法中,单精度乘法的次数和 SOS 算法一样,也是 $2n^2+n$。但是与 SOS 算法相比,它将两个乘法结合到了一起,减少了一次循环。步骤(3)在实现的时候,可以不进行除以 b 的移位运算,而是将结果直接保存到 A 的高一位中。CIOS 详细的效率分析,将在下一节中与 FIPS 方法一起给出。

2.3 FIPS 模乘法算法

Finely Integrated Operand Scanning 算法简称 FIPS 算法,在 1993 年由 Kaliski 提出,该方法设计思想来源于 Comba 算法,它采用类似 Comba 算法的从右向左的按列累加的方法设计而成,将乘法和模约减算法完美地融合为一体。Koc[5]认为 FIPS 算法的"乘积累加"结构非常适用于数字信号处理器(DSP)等微处理器,是 RSA 硬件实现中常用的算法,在某些特定的硬件设备上具有很高的执行效率。算法 5 给出了 FIPS 算法的描述。

算法 5 FIPS 模乘法算法

输入:整数 $m=(m_{n-1}\cdots m_1 m_0)_b$,$x=(x_{n-1}\cdots x_1 x_0)_b$,$y=(y_{n-1}\cdots y_1 y_0)_b$,$\gcd(m,b)=1$,$R=b^n$,$m'=-m^{-1}\bmod b$;

输出:$U=xyR^{-1}\bmod m$。

算法步骤为:

(1)$(v_2 v_1 v_0)_b=0$;

(2)对于 i 从 0 到 $n-1$,执行:

①$(v_2 v_1 v_0)_b \leftarrow (v_2 v_1 v_0)_b+\sum_{j=0}^{i-1} x_j y_{i-j}+\sum_{j=0}^{i-1} u_j m_{i-j}+y_i x_0$;

②$u_i \leftarrow v_0 m' \bmod b$;

③$(v_2 v_1 v_0)_b \leftarrow (v_2 v_1 v_0)_b+u_i m_0$;

④$v_0 \leftarrow v_1$,$v_1 \leftarrow v_2$,$v_2 \leftarrow v_0$。

(3)对于 i 从 0 到 $n-1$,执行:

①$(v_2 v_1 v_0)_b=0(v_2 v_1 v_0)_b \leftarrow (v_2 v_1 v_0)_b+\sum_{j=0}^{i-1} x_j y_{i-j}+\sum_{j=0}^{i-1} u_j m_{i-j}$;

②$u_{i-n} \leftarrow v_0$;

③$v_0 \leftarrow v_1$,$v_1 \leftarrow v_2$,$v_2 \leftarrow v_0$。

(4)$u_n \leftarrow v_0$;

(5)如果 $U \geqslant m$,则 $U \leftarrow U-m$;

(6)返回(U)。

算法 5 实质上是将 $x\times y$ 和 $U\times m$ 这两个多精度乘法按照 Comba 算法的方法结合到了一起。在步骤(2)中,分别计算了右边 n 列的乘积之和,并求出所有的 u_i。步骤(3)计算出左边的 $n-2$ 列的乘积之和,同时得到了 $A=xyR^{-1}\bmod m$ 的值。由于每得到一个 a_i 时,相应的 u_i 值不会被再用到,因此可以直接用 u_i 来保存 a_i 的值,在编程序时可节省一个长为 n 的数组。

FIPS 模乘法中,单精度乘法的次数和 CIOS 算法一样,也是 $2n^2+n$。但在具体实现时的效率却大大不同。可以根据算法,从理论上统计这两种算法在乘法、加法、读内存、写内存方面的差异。通过表 1 可以看出,FIPS 算法的读内存、写内存次数都远远高于 CIOS 算法。这是因为由于寄存器个数有限,在计算机程序中,FIPS 算法中的三元组$(v_2 v_1 v_0)$需要用内存变量来存储,使访问内存所消耗的时间大大增加。但是如果在数字信号处理器(DSP)等微处理器上实现,将会使这些访问内存所消耗的时间忽略不计。因此,在这类硬件上实现 FIPS 算法,效率将大大提高。如果不考虑三元组$(v_2 v_1 v_0)$访问内存的次数,表 1 将修改为表 2。表 2 中给出的数据可以明显看出,在 DSP 等硬件上实现模乘法时,FIPS 算法仅仅在加法次数上略多于 CIOS 算法,而读写内存的次数都大大减少,有着很大的优势,非常适合于 RSA 算法的硬件实现。

表 1 CIOS 算法与 FIPS 算法运算次数统计表

操作	CIOS 算法	FIPS 算法	后者较前者
乘法	$2n^2+n$	$2n^2+n$	相等
加法	$4n^2-n-1$	$4n^2-2$	多 $n-1$ 次
读内存	$4n^2+7n$	$10n^2+5n-2$	多 $6n^2-2n-2$ 次
写内存	$2n^2+4n$	$6n^2+8n-3$	多 $4n^2+4n-3$ 次

表 2 特定硬件上 CIOS 算法与 FIPS 算法运算次数统计表

操作	CIOS 算法	FIPS 算法	后者较前者
乘法	$2n^2+n$	$2n^2+n$	相等
加法	$4n^2-n-1$	$4n^2-2$	多 $n-1$ 次
读内存	$4n^2+7n$	$4n^2+5n-2$	少 $2n+2$ 次
写内存	$2n^2+4n$	$8n-3$	少 $2n^2-4n+3$ 次

算法速度测试:

测试平台环境为 CPU:Genuine Intel(R) CPU 2140@ 1.60 GHz 1.60 GHz;内存:1 G;操作系统:Windows XP。

综合运用以上 3 个加速方案,对于一次模幂运算 $M^e \bmod n$ 所消耗的时间(取 50 次的平均值),对比结果如表 3 所示。

对于 RSA 密码体制来说,安全性固然是其特征之一,但是加解密的速度却是衡量其算法好坏的首要标准。经过以上分析表明,中国剩余定理可以使非常大的数对 M 的模运算转化到更小的数上来运算,并且新算法

表 3 对比结果

M，N 长度/bit	e 长度/bit	耗费时间/ms		时间减少量/%
		基于传统 Montgomery 算法	基于快速 Montgomery 算法	
512	256	22.43	20.09	10.43
768	384	51.19	42.83	16.33
1 024	512	102.62	80.12	21.93
2 048	1 024	648.73	461.08	28.93

可以切实地将 RSA 系统的运算速度提高了约 4 倍。Montgomery 模乘算法不仅能够简化 RSA 算法中的模乘步骤，而且能够有效地节省算法复杂度。

参考文献

[1] SHAMIR A.How to check modular exponentiation[C].The rump session of EUROCRYPT，1977.

[2] 涂航，刘玉珍，张焕国，等.智能卡操作系统中 RSA 算法的实现与应用[C].第六届中国密码学学术会议论文集，2000：239-243.

[3] MONTGOMERY P.Modular multiplication without trial division. Mathematics of Computation，1985，(44)：519-521.

[4] DUSSE B，KALISHI J.A cryptographic library for the Motorola DSP56000[C].Advances in Cryptology-EUROCRYPT90，1990.

[5] KOC C K.High-speed RSA implementation[C].RSA Labs Technical Report TR-201，1994.

(收稿日期：2009-10-30)

作者简介：

丁丽丽，女，1984 年生，硕士研究生，主要研究方向：信息安全、密码学。

王丽君，女，1952 年生，教授，主要研究方向：信息安全、密码学。