RSA密码算法的安全及有效实现*

张宝华[1]，殷新春[2]
（1. 复旦大学 计算机科学技术学院，上海 200433；
2. 扬州大学 信息工程学院，江苏 扬州 225009）

摘 要： RSA密码算法的实现电路易受到诸如能量分析、时间分析等旁门攻击。综述了RSA密码算法中模幂运算的主要攻击方法及其防御措施。针对模幂运算的软件实现和硬件实现，提出了基于随机概率的软件实现防御措施和基于模幂指数三进制表示的硬件实现防御措施。两种防御措施较之前的防御措施在安全性和效率方面都有显著的提高。两种防御措施具有通用性，可移植到ECC中的标量乘法运算中去。

关键词： RSA密码算法；旁门攻击；软件实现；硬件实现；模幂
中图分类号： TP309.7 **文献标识码：** A **文章编号：** 0529-6579（2008）06-0022-05

目前为止，RSA密码算法仍然是公钥制密码算法中被广泛应用于各种嵌入式密码系统的算法之一，其核心运算是模幂运算。在评估密码系统的安全性时，传统安全模型中考虑的攻击都是从协议规范的数学基础进行分析的，如对大整数分解难题的分析等。且Kocher等[1]提出针对智能卡微处理器的时间分析及功耗分析以来，研究者越来越注意到可能利用实现和操作环境的特性进行攻击，产生了很多新的攻击旁门攻击方法（side channel attack，SCA）方法，如简单功耗分析（Simple Power Analysis，SPA）、差分功耗分析[2]（Differential Power Analysis，DPA）等。

因此，研究RSA密码算法的安全及有效实现是人们十分关注的问题，产生了很多抵抗功耗分析等SCA攻击的防御措施。如Montgomery模幂算法[3]、随机扫描模幂算法[4]等。Messerges认为随机化模幂算法能够显著抵抗他所列举的所有功耗分析方法的攻击，包括SPA、DPA和相关性攻击（correlation attack）。然而，研究表明随机化模幂算法不能抵抗文［5］提出的功耗轨迹分析方法。

本文针对文［5］的功耗轨迹方法，首先给出了一种新的基于随机概率的软件实现防御措施，对一般模幂运算算法进行了改进，其次，结合整数的三进制表示，给出了模幂运算的硬件实现防御措施；最后，给出了我们提出的新的防御措施和已有防御措施的比较，结果表明，在安全性和效率方面都有显著的提高。

1 RSA密码体制中模幂运算相关算法概述

模幂运算 $x \to x^d \pmod N$ 是RSA密码算法（加密、解密、签名及验证签名过程）的核心运算。给定密文 C、密钥 d 及 N，RSA的解密过程即 $M \equiv C^d \pmod N$。令 $d = \sum_{i=0}^{n-1} d_i 2^i = d_{n-1}d_{n-2}\cdots d_1 d_0$ 为 d 的二进制表示形式，实现RSA解密过程的基本算法是“平方-乘”（square and multiply）算法。下面介绍不同形式的“平方-乘”算法。

算法1：从左到右平方-乘算法
输入：C、N，d 为 n-bit 的整数
1）$R = 1$
2）for（$i = n-1$ down to 0）
3）　{ $R = R^2 \bmod N$
4）　if（ith bit of d is a 1）
5）　$R = R \cdot C \bmod N$ }
6）return R }

算法2：从右到左平方-乘算法
输入：C、N，d 为 n-bit 的整数
1）$R = 1$，$S = C$
2）for（$i = 0$ to $n-1$）
3）　{ if（ith bit of d is a 1）
4）　$R = R \cdot S \bmod N$
5）　$S = S^2 \bmod N$ }

* **收稿日期：** 2008-03-05
基金项目： 国家自然科学基金资助项目（90607005）；国家“863”高科技研究发展计划基金资助项目（2007AA012448）；江苏省“六大人才高峰”资助项目（06-E-025）
作者简介： 张宝华（1979年生），女，博士研究生；E-mail：061021050@fudan.edu.cn

6）return R

算法 2 中，平方、乘法两个运算步骤可用串行、并行两种方式实现，因此，模幂运算的“平方－乘”实际上有四种基本实现形式。从算法 1 和 2 可以看出，模幂运算中的执行的运算依赖于密钥 d 的具体 bit 位。

2　SCA 攻击及其防御措施

2.1　SCA 攻击概述

Kocher 于 1995 年最早提出了针对 RSA 智能卡处理器的时间攻击（timing attack）和 SPA 攻击，通过精确测量密码设备执行密码操作所用的时间或能耗轨迹，从而推断出密钥的信息。1999 年，Kocher 等人又提出了 DPA 攻击。Messerges 等人则最先将该方法应用于公钥密码系统如 RSA 等中，提出了 SEMD、MESD 和 ZEMD 等攻击方法[6]。Walter 对 RSA 进行了分析，指出滑动窗口法也是不安全的[7]。Klima 等人对 RSA 进一步进行了分析，给出了三种攻击方式[8]。此外，文［5］进一步提出了对 RSA 的新的功耗轨迹分析。

2.2　防御措施

对于时间攻击，可采用随机化密钥的方法：

算法 3：RSA（抵抗时间攻击）防御措施

1）随机产生整数 $r \in Z_N^*$

2）计算密文 $C' \equiv Cr^e \pmod N$

3）计算明文 $M' \equiv (C')^d \pmod N$

4）计算明文 $M \equiv Mr^{-1} \pmod N$

对于能耗分析攻击，可采用随机扫描模幂算法和添加伪操作[9]的方法，如下：

算法 4：随机扫描模幂运算算法

输入：C、N，d 为 n－bit 的整数

1）$R = 1$，$S = C$

2）Z = random number between 1 and $n-2$

3）for（$i = Z$ to $n-2$）

4）　｛if（ith bit of d is a 1）

5）　　$R = R \cdot S \bmod N$，$S = S^2 \bmod N$｝

6）for（$i = Z-1$ down to 0）

7）　｛$R = R^2 \bmod N$

8）　if（ith bit of d is a 1）

9）　　$R = R \cdot M \bmod N$｝

10）return R

算法 5：添加伪操作后的从左到右扫描模幂运算算法

输入：C、N，d 为 n－bit 的整数

1）$R = 1$

2）for（$i = n-1$ down to 0）

3）　｛$R = R^2 \bmod N$

4）　if（ith bit of d is a 1）

5）　　$R = R \cdot C \bmod N$

6）　else

7）　　$R = R \cdot C \bmod N$（should not be stored）｝

8）return R

2.3　研究思路

RSA 密码算法的实现，既可以在通用的微处理上用高效的软件实现，也可以用嵌入在智能卡上的专用硬件实现。这两种实现方式都易受到旁门攻击。

探讨各种攻击的防御措施，大体上可从两条线索进行，一条线索是追求设备执行密码操作的过程中其泄露的信息的“平衡”，即所泄露的时间或能耗信息对敌手而言是无效的，比如添加伪操作方法、随机化密钥等方法（如算法 3、4、5）；另一线索是追求设备执行密码操作过程中的每一步运算的“平等”，即密钥为 0 或为 1 时密码设备执行的操作是“平等”的，这里的“平等”有两层含义：一是密码设备执行密码操作过程中的每一个运算是相等的，在 RSA 模幂运算中，由于平方运算和乘法运算已经是有限域的基本运算，因此不可行；另一层含义是，密钥为 0 和为 1 时密码设备执行的操作是相等的，这跟添加伪操作不同，是真正意义上的相等（即相同的运算）。

本文正是从这两条线索出发，给出两种防御措施。一种防御措施是针对 RSA 密码算法的软件实现，通过随机添加伪操作的方法达到设备执行密码操作过程中泄露信息的“平衡”；另一种防御措施则是针对 RSA 密码算法的硬件实现，给出模幂运算的硬件实现，达到设备执行密码操作过程中每一步运算的“平等”。需要指出的是，本文提出的两种防御措施具有一定的通用性，与具体的设备和密码操作无关，因此，可移植到椭圆曲线密码体制[10－13]的标量乘法运算上去。

3　两种防御措施

3.1　模幂运算硬件实现防御措施

本小节中，令 $d = \sum_{i=0}^{m-1} d_i 3^i = d_{m-1} d_{m-2} \cdots d_1 d_0$ 为 d 的三进制表示形式，其中，$d_i \in \{0, 1, 2\}$，模幂运算算法如下：

算法 6：基于三进制的模幂运算算法

输入：C、N 及 d 的三进制表示

1）$R_0 = 1$，$R_1 = C$

2）for（$i = m-2$ down to 0）

3） { if $d_i = 0$ then

4） $R_0 = R_0^3$, $R_1 = R_0^3 - R_1$

5） elseif $d_i = 1$ then

6） $R_0 = R_0^2 \cdot R_1$, $R_1 = R_1^2 - R_0$

7） elseif $d_i = 2$ then

8） $R_0 = R_1^2 \cdot R_0$, $R_1 = R_1^3$ }

9） return R_0

易验证，算法6的输出 R_0 即为 $M \equiv C^d \pmod N$ 的值。算法6优势在于它是基于密钥的三进制表示，所以循环的次数少于一般二进制表示，且易于硬件实现。以下用 d_i^H 和 d_i^L 表示 d_i 的具体bit位，下面是算法6的硬件实现实现：

算法7：基于三进制的模幂运算硬件实现算法

输入：C、N 及 d 的三进制表示

1） $R_0 = 1$, $R_1 = C$

2） for ($i = m-2$ down to 0)

3） { $R_2 = R_{(e_i^L)}$, $R_3 = R_{(\overline{e_i^L})}$, $R_4 = R_i^H$

4） $R_1 = R_{(e_i^L) \vee (e_i^H)}$, $R_0 = R_4$

5） $R_0 = R_0^2 \bmod N$, $R_0 = R_0 R_2 \bmod N$

6） $R_1 = R_1^2 \bmod N$, $R_1 = R_1 \cdot R_3 \bmod N$ }

7） return R_0

定理1. 输入 C、N 及 d 的三进制表示，算法7输出的值 R_0 等于 $M \equiv C^d \pmod N$。

证明：易知算法6与算法7的差别仅在于循环体内部的运算，因此，只需证明循环体内部运算等价即可。算法2中，若 d_i 等于0，首先执行运算 $R_2 = R_0$, $R_3 = R_1$, $R_4 = R_0$, $R_1 = R_0$，然后并行计算 $R_0 = R_0^2 \bmod N$, $R_0 = R_0 \cdot R_2 \bmod N$, $R_1 = R_1^2 \bmod N$, $R_1 = R_1 \cdot R_3 \bmod N$ 即 $R_0 = R_0^3 \bmod N$, $R_1 = R_0^2 \cdot R_1 \bmod N$，与算法9中的条件为0的运算结果相同，同理可验证 d_i 等于1和2的情况，证毕。

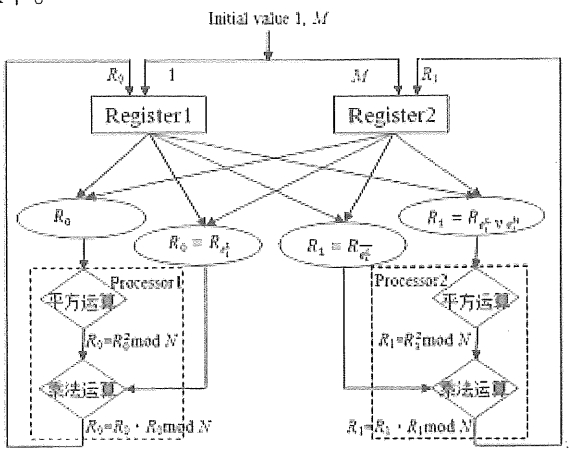

图1 算法7的硬件实现

Fig. 1 Hardware Implementation of Algorithm 7

图1给出了算法7的硬件实现图，由图可知，模幂运算不依赖于密钥 d 的值，都执行相同的运算，且平方运算、乘法运算可并行实现，参见Processor1和Processor2，大大地提高了执行的效率。

3.2 添加随机伪操作的防御措施

算法4是添加伪操作的基本方法，研究表明，该方法并不能完全抵抗SCA攻击。文［5］中对这一方法做了改进，改进算法如下：

算法8：添加随机伪操作后的从左到右扫描模幂运算算法

输入：C、N 及 d 的二进制表示

1） $R = 1$

2） D = random number between 1 and $2K$

3） for ($i = n-1$ down to 0)

4） { $R = R^2 \bmod N$

5） if (ith bit of e is a 1)

6） $R = R \cdot C \bmod N$

7） else if ($D < K$)

8） $R = R \cdot M \bmod N$ (should not be stored) }

9） return R }

算法8中，C、N 及 d 表示如前，K 为任意整数。通过添加随机的伪操作，使得敌手即使在获得精确的功耗轨迹后，难以由功耗轨迹推出相应的秘密信息。但算法8也有缺陷，部分泄露了密钥的信息，假设敌手观测到密码设备执行密码操作的功耗轨迹，经过统计分析后确认某些功耗轨迹是由密钥为0时的运算所产生的，另外一些功耗轨迹则是由密钥为1时的运算产生，由于伪操作是随机的，使得敌手无法辨认功耗轨迹到底是由密钥为1时的运算产生，还是由伪造作产生，但是，敌手却可以确定由密钥为0时的运算所产生的功耗轨迹一定对应于密钥0，因此，算法8泄露了密钥中的部分bit位为0的信息。

本文对算法8进行改进，在随机添加伪操作假装密钥为1的同时，也随机添加假装密钥为0的伪操作，使得敌手即使在通过分析功耗轨迹等之后也无法确认哪些操作是由真正的密钥为0或1产生，还是由伪装为0或1的伪操作产生。改进算法如下：

算法9：添加随机伪操作“1”的从左到右扫描模幂运算算法

输入：C、N 及 d 的二进制表示

1） $R = 1$

2） D = random number between 1 and $2K$

3） for ($i = n-1$ down to 0)

4） { $R = R^2 \bmod N$

5）　if（ ith bit of d is a 1）
6）　$R = R \cdot C \bmod N$
7）　elseif（$D < K$）
8）　$R = R \cdot C \bmod N$（should not be stored）
9）　if（$D < K$）
10）　$R = R^2 \bmod N$（should not be stored）｝
11）return R

算法 10：添加随机伪操作“0”的从左到右扫描模幂运算算法

输入：C、N
1）$R = 1$
2）D = random number between 1 and $2K$
3）for（ $i = n-1$ down to 0 ）
4）　｛ $R = R^2 \bmod N$
5）　if（ ith bit of d is a 1）
6）　$R = R \cdot C \bmod N$
7）　elseif（$D < K$）
8）　$R = R^2 \bmod N$（should not be stored）｝
9）　if（$D < K$）
10）　$R = R \cdot C \bmod N$(should not be stored)
11）return R

4　安全性及有效性分析

4.1　安全性分析

首先分析硬件实现防御措施的安全性。由图 1 易知，模幂运算独立于具体的密钥 d 的 bit 值，且 Processor1 和 Processor2 执行相同的运算，因此，密码设备执行每一次循环的时间消耗和能量消耗等旁门信息是完全相同的，即满足前面我们提出的“平等”的条件，因而诸如时间分析、能耗分析等旁门攻击对它是不可行的。

其次分析算法 9 和 10。针对软件实现的防御措施有很多，诸如随机扫描模幂运算方法、添加随机伪操作方法等，研究表明，随机扫描模幂运算方法并不能抵抗诸如功耗轨迹分析等旁门攻击，文[5]中做了改进，给出添加随机伪操作方法，但是该方法仍然泄露了密钥的部分（bit 位为 0）的信息。算法 9 和 10，由于添加了随机伪装成密钥 bit 为 0 和为 1 的操作，使得敌手即使在分析旁门信息后，无法确认该操作是由真正的密钥产生还是由伪操作产生，因此，它较算法 8 更安全。

4.2　有效性分析

首先分析针对软件实现的防御措施的有效性。设密钥 d 的二进制长度为 n，且每一位为 0 和为 1 的概率是相等的，对于未添加伪操作的从左至右扫描模幂的算法 1，平均需要 n 次平方运算加 $0.5n$ 次乘法运算。添加伪操作的算法 5 则需要 n 次平方运算加 n 次乘法运算。在算法 9 中，设伪操作的执行是随机的，扫描密钥的 bit 位时，由随机数发生器决定是否执行伪操作，假设随机数发生伪操作的概率为 p，那么算法 9 的运算量为（$1+p$）n 次平方运算加（$0.5+0.5p$）n 次乘法运算。算法 10 需要（$1+0.5p$）n 平方运算加（$0.5+p$）n 乘法运算。

其次分析硬件防御措施的有效性。在算法 7 中，无论密钥 d 的值为 0 或为 1，都执行相同的运算。设密钥 d 的三进制表示的长度为 m，采用如图 1 的并行实现后，算法 7 需要 m 次平方运算加 m 次乘法运算。一般而言，我们取 $m=0.6n$，则算法 7 需要 $0.6n$ 次平方运算加 $0.6n$ 次乘法运算。表 1 给出这几种防御措施的效率的比较（假定平方运算与乘法运算的运算量相等）。

表 1　各种防御措施的效率的比较

Tab. 1　Comparison of countermeasures efficiency

	平方次数	乘法次数	总量($n=1024$, $m=0.6n$, $p=0.2$)
算法 5	n	n	$2n$(2048)
算法 9	$(1+p)n$	$(0.5+0.5p)n$	$1.5(1+p)n$(1843)
算法 10	$(1+0.5p)n$	$(0.5+p)n$	$1.5(1+p)n$(1843)
算法 7	m	m	$2m$(1229)

由表 1 可以看出，在安全性相当的情况下，算法 7 较算法 5 效率提高了 40%，在算法 9 和 10 中，若我们控制随机数满足条件的概率在 0.2 时，较算法 5 皆提高了 10%。

5　结　语

RSA 密码算法迄今为止仍然是公钥密码算法中较为常用的算法之一，且经常被应用于智能卡等设备，易受到旁门攻击。本文针对 RSA 密码算法的软件和硬件实现，提出了两种抵抗旁门攻击防御措施，在安全性和效率方面较已有的防御措施有显著的提高。本文提出的抵抗旁门攻击的防御措施具有通用性，也适用于椭圆曲线密码体制的标量乘法运算算法等。

致谢：衷心感谢陈豪教授的宝贵意见。

参考文献：

[1] KOCHER C. Timing attacks on implementations of Diffie-Hellman, RSA , DSS, and other systems[C]//Proceeding of the Advances in Cryptography (CRYPTO ' 96), Springer-Verlag, 1997:104 - 113.

[2] KOCHER C, JAFFE J, JUN B. Differential power analysis [C]//Proceeding of the Advances in Cryptography (CRYPTO'99), Springer- Verlag, 1999:388 - 397.

[3] MONTGOMERY P L. Speeding the Pollard and elliptic curve methods of factorization[J]. Mathematics of Computation, 1987, 48(177): 243 - 264.

[4] MESSERGES T S. DABBISH E A, SLOAN R H., Power analysis atta-cks of modular exponentiation in smartcards [C]//Proceeding of the workshop on Cryptographic Hardware and Embedded Systems (CHES' 99), Springer-Verlag, 2000:144 - 157.

[5] 韩军，曾晓洋，汤庭鳌. RSA 密码算法的功耗轨迹分析及其防御措施[J]. 计算机学报, 2006, 29(4): 590 - 596.
HAN J, ZENG XY, TANG T A. Power trace analysis attack and countermeasures for RSA cryptographic circuits [J]. Chinese Journal of Computers, 2006, 29(4): 590 - 596.

[6] MESSERGES T S. Power analysis attacks and countermeasures for cryptographic algorithms [D]. Chicago: Graduate College of the University of Illionois, 2000.

[7] WALTER C. Sliding windows succumbs to big mac attack [C]//Proceeding of the workshop on Cryptographic Hardware and Embedded Systems(CHES'01), Springer-Verlag, 2001:286 - 299.

[8] VLASTIMILl K, TOMAS R. Futher results and considerations on side channel attacks on RSA[C]//Proceeding of the workshop on Cryptographic Hardware and Embedded Systems(CHES'02), Springer-Verlag, 2002:244 - 259.

[9] CORON J S. Resistance against differential power analysis for elliptic curve cryptosystems [C]//Proceeding of the workshop on Cryptographic Hardware and Embedded Systems(CHES'99). Springer-Verlag, 1999:292 - 302.

[10] KOBLITZ N. Elliptic curve cryptosystems[J]. Mathematics of Computation, 1987, 48(177): 203 - 209.

[11] MILLER K. Uses of elliptic curve in cryptography[C]// Proceeding CRYPTO'85, Springer Verlag, 1986: 417 - 426.

Secure and Efficient Implementation for RSA Cryptographic Algorithm

ZHANG Bao-hua[1], *YIN Xin-chun*[2]

(1. School of Computer Science, Fudan University, Shanghai 200433, China;
2. Information Engineering College, Yangzhou University, Yangzhou 225009, China)

Abstract: The implementation of RSA cryptosystems is vulnerable to SCA attacks such as power analysis and time attack. First countermeasures for the exponentiation computation of RSA cryptographic algorithm were summarized. Then the software countermeasures based on random probability and hardware countermeasure based on the 3-adic representation of exponent were proposed. Analysis shows that the two countermeasures achieved great improvements in both security and efficiency compared to existed countermeasures. Both two generic countermeasures can be transplanted to the scalar multiplication of ECC.

Key words: RSA cryptographic algorithm; side channel attacks; software implementation; hardware implementation; exponentiation