RSA 公钥密码体制的实现研究

刘宏伟[1,2]　王昭顺[1,2]　班晓娟[1]

[1]（北京科技大学计算机系，北京 100083）

[2]（北京兆日科技有限责任公司集成电路部，北京 100089）

摘　要　RSA 公钥算法是目前被广泛采用的公钥体制密码算法，但 RSA 算法的运算复杂度极大，如果能针对个人用户，用专用集成电路快速而又低成本地实现 RSA 算法，将有助于 RSA 算法的推广。该文对目前应用较好的 RSA 公钥算法的软、硬件算法进行了详细的实现分析，指出采用合理的软硬件算法并增加部分硬件，可以用专用集成电路较快地实现 RSA 算法。

关键词　RSA 公钥算法　公钥密码体制　时间复杂度　专用集成电路

文章编号 1002-8331-（2002）17-0052-03　**文献标识码** A　**中图分类号** TP309

Implementing Research of RSA Cryptosystem

Liu Hongwei[1,2]　Wang Zhaoshun[1,2]　Ban Xiaojuan[1]

[1]（Dept.of Computer Science，Beijing University of Science and Technology，Beijing 100083）

[2]（Dept.of ASIC，SINOSUN Technology Co.Ltd.，Beijing 100089）

Abstract：As a public-key cryptosystem，the RSA cryptosystem is currently gaining broad acceptance，but the complexity of its calculation is rather large.If you can aim at the personal user，use ASIC，implement RSA cryptosystem faster with lower cost，it will greatly promote the application of RSA cryptosystem.This paper analyses at length the better software and hardware implementations of RSA cryptosystem，and points out that proper use of software and hardware arithmetic accompanied with the addition of some appropriate hardware can quicken the implementation of RSA cryptosystem.

Keywords：RSA cryptosystem，public-key，time complexity，ASIC

1　简介

1977 年，Rivest，Shamir 和 Adleman 联合提出一种基于数论中欧拉定理的公钥密码系统，简称 RSA 公钥系统，它的安全性是基于大数因子分解，后者在数学上是个困难问题。RSA 是目前较为完善的公钥算法。在信息安全领域有着广泛的应用，主要包括电子商务中交易双方的身份认证、网上电子证书的发放和管理系统（CA 认证中心）、银行的网上银行业务等等。

RSA 算法采用 e、n 为公开密钥，d 为私钥。加密时，完成如下计算：

$C=M^e \pmod n$

M 是明文，C 是密文。解密时，完成如下计算：

$M=C^d \pmod n$

其中：

（1）公钥模数 n 是 k 位的正数，为安全性，k 应为 512~2048 位。

（2）公钥指数 e 是 h 位的正数，e 通常较小，一般不超过 32 位，最小值为 3。

（3）密钥指数 d 是较大的正数，一般假设 d 是 k 位的正数。

（4）M 应小于 n。如果明文的长度大于 k，通常有应将明文分组，以保证 M 应小于 n。

可见，RSA 算法主要是完成整数的乘幂运算。由于整数和乘幂的位数较多（至少为 1024 比特），造成 RSA 算法的运算复杂度极大，因此，如要快速地实现 RSA 算法，软、硬件的成本较高，这在一定程度上影响了 RSA 算法的推广，尤其是针对个人用户硬件的推广。为了较快地实现 RSA 算法，研究人员提出了很多软硬件算法，下面就目前较好的软硬件算法进行针对个人用户应用的实现分析。

2　软件算法

2.1　Montgomery 算法

在软件算法中，Montgomery 算法是比较优秀的算法，该算法特别适用于通用计算机、DSP 或微处理器芯片，因为它用对 2^i 的取余运算取代了对 n 的取余运算，而对 2^i 的取余运算特别适于二进制的运算。

Montgomery 算法的原理如下所述，假定模数 n 是 k-bit 的数，r 为 2^k，Montgomery 算法要求 r、n 互素，然而当 n 为奇数时，这个要求即可满足。此时，给定一个整数 $a<n$，定义 a 相对于 r 的 n-余数为

$\bar{a}=a\cdot r \bmod n$

给定两个 n-余数 $\bar{a}$、$\bar{b}$，它们的 Montgomery 乘积为

$\bar{R}=\bar{a}\cdot\bar{b}\cdot r^{-1} \bmod n$

基金项目：国家自然科学基金（编号：60075012）；清华大学智能技术与系统国家重点实验室开放课题（编号：0109）

作者简介：刘宏伟，博士，副教授，主要研究方向为信息安全领域的集成电路设计及计算机体系结构。王昭顺，博士研究生，副教授，主要研究方向为计算机体系结构及信息安全领域的集成电路设计。班晓娟，博士研究生，讲师，主要研究方向为人工智能。

(C)1994-2019 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

其中，r^{-1} 是 r 相对模数 n 的倒数，即：

$r^{-1}\cdot r=1 \bmod n$

基于上述公式，Montgomery 算法如下：

function ModExp (M,e,n) {n 是一个奇数}

Step1 调用 ModInverse 计算 n_0'；

Step2 $\overline{M}=M\cdot r \bmod n$；

Step3 $\overline{x}=1\cdot r \bmod n$；

Step4 for $i=k-1$ down to 0

Step5 $\overline{x}$=MonPro ($\overline{x}$,$\overline{x}$)

Step6 if $e_i=1$ then $\overline{x}$=MonPro ($\overline{M}$,$\overline{x}$)

Step7 x= MonPro ($\overline{x}$,1)

Step8 return x.

函数 MonPro 和 ModInverse 参见文献[1,2]。

2.2 Montgomery 算法的时间复杂度分析

在这一分析中，假定机器的字长为 w 位，公钥 e、n 和私钥 d 的长度均为 k 位　实际上，e 通常较短，如 32 位，d 为 512~2048 位)，即 s 个机器字，明文和密文的长度为 k 位。同时，两个 w 位的操作数的加法和乘法时间分别为 A、P 时钟周期。

在函数 ModInverse 中，计算 n_0'的时间为：

$$\sum_{j=2}^{w}(P+A)=(w-1)(P+A)$$

计算 $\overline{M}=M\cdot r \bmod n$ 需要 sw 次 s 长度的减法，同时计算 $\overline{x}=1\cdot r \bmod n$ 需要 w 次 s 长度的减法，因此，这两个操作共需

$$sw(A)+w(A)=(s^2+s)wA$$

在进行乘幂运算中，需要调用 $(k-1)$次 MonPro 完成平方和乘法运算 (平方运算的时间约为乘法运算的一半)。在 MonPro 中，第一步计算$\bar{a}\cdot\bar{b}$需要

$$\sum_{i=0}^{s-1}\sum_{j=0}^{s-1}(P+2A)=s^2(P+2A)$$

周期，如果$\bar{a}=\bar{b}$，则根据优化的算法，只需：

$$\frac{s(s-1)}{2}(P+2A)$$

周期，然后在第 7 步到第 15 步需要：

$$\sum_{i=0}^{s-1}\left[P+\sum_{j=0}^{s-1}(P+2A)+\sum_{j=i+s}^{2s-1}A\right]=sP+s^2(P+2A)+\frac{s^2+s}{2}A$$

$$=(s^2+s)P+\frac{5s^2+s}{2}A$$

周期。然后在第 18 步到第 21 步需要：

$$(s^2+s)P+\frac{5s^2+s}{2}A+sA=(s^2+s)P+\frac{5s^2+3s}{2}A$$

周期。因此，在 MonExp 中，每执行一次 Step6 共需：

$$s^2(P+2A)+(s^2+s)P+\frac{5s^2+3s}{2}A=(2s^2+s)P+\frac{9s^2+3s}{2}A$$

周期，而每执行一次 Step5 共需：

$$\frac{s(s-1)}{2}(P+2A)+(s^2+s)P+\frac{5s^2+3s}{2}A=\frac{3s^2+s}{2}P+\frac{7s^2+s}{2}A$$

因此，完成 RSA 的时间复杂度为：

$$T=(w-1)(P+A)+(s^2+s)wa+(s-1)+\left[\frac{3s^2+s}{2}P+\frac{7s^2+s}{2}A\right]+$$

$$(s-1)\left[\frac{3s^2+s}{2}P+\frac{7s^2+s}{2}A\right]$$

$$=\left[\frac{5Pw}{2}+\frac{23Aw}{4}\right]s^3+\left[Pw+\frac{9Aw}{4}-\frac{7P}{2}-8A\right]s^2+$$

$$\left[Aw-\frac{3P}{2}-2A\right]s+(w-1)(P+A)$$

根据上式，选择不同的 A、P、s、w 及 10Mhz 的机器主频，得出对 RSA 的时间复杂度的测试，如表 1 所示：

表 1

	情况 1			情况 2			情况 3		
A (加法时间)	6	6	6	6	6	6	6	6	6
P (乘法时间)	6	18	60	6	18	60	6	18	60
m (机器周期 /ns)	100	100	100	100	100	100	100	100	100
w (机器字长 /位)	16	16	16	16	16	16	32	32	32
s (/w 的字长)	64	64	64	32	32	32	32	32	32
RSA 时间分析 /秒)	20.9	33.5	77.8	2.6	4.2	9.8	5.3	8.4	19.6

其中：乘法时间 P 分别为 6、18、60，分别对应具有较快的硬件乘法部件、一般的硬件乘法部件和无乘法部件 (软件实现乘法)的情况。

情况 1 为 1024 位的 RSA 计算时间，机器字长为 16 位，指令周期为 6 个时钟周期。可见，当计算 1024 位的 RSA 时，如果用软件实现，即便软件算法较好，也至少需要 21 秒的时间。

情况 2 为 512 位的 RSA 计算时间，机器字长为 16 位，指令周期为 6 个时钟周期。可见，当用软件计算 512 位的 RSA 时，如果有较快的硬件乘法器，可以在 2.6 秒计算完成，否则也需 10 秒的时间。

情况 3 也为 1024 位的 RSA 计算时间，机器字长为 32 位，指令周期为 6 个时钟周期。可见，当机器的字长增加时，RSA 的计算时间大幅度减少，当有较快的硬件乘法器时，RSA 的计算时间为 5 秒。

综上所述，如果要用软件在秒量级实现 1024 位或者更多位数的 RSA 算法，则必须有较快的处理器，如机器的字长至少为 32 位，主频不低于 10MHz，并且有较快的乘法部件。

3 硬件实现方法分析

用硬件实现 RSA 是指完全用集成电路来实现 $C=M^e \bmod n$ 或 $M=C^d \bmod n$ 的运算，即实现乘幂运算的加速器。由于 M、d、e、n 均为 k 位，k 位 512~2048，因此在硬件实现 RSA 算法时，必须有足够的寄存器保存 n、M、e 或 d)。

对于硬件实现乘幂运算，比较好的算法是 LR 算法或 RL 算法，两个算法都是通过对指数 e 或 d 的按位查询，完成相应的乘幂运算，只是 LR 是从指数的高位 (left)到低位 (right)，而 RL 是从低位到高位。以 LR 为例：

LR Binary Method

Input :M,e,n

Output :$C=M^e \bmod n$

(1)if $e_{h-1}=1$ then $C=M$ else $C=1$

(2)for $i=h-2$ downto 0

(3)　$C=C*C \bmod n$

(4)　if $e_i=1$ then $C=C*M \bmod n$

(5)return C

因此，根据 LR 算法，需要找到 k 位整数和 k 位整数相乘然后对 k 位整数取余的硬件算法。显然，乘法和取余交替进行的方法是较好的算法，因为

$$P=A*B=A*\sum_{i=0}^{k-1}B_i2^i=\sum_{i=0}^{k-1}(A\cdot B_i)2^i=2((\cdots 2(0+A\cdot B_{k-1})+A\cdot B_{k-2})+\cdots)+A\cdot B_0$$

(C)1994-2019 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

硬件算法描述为：

(1) $P:=0$；

(2) for $i=0$ to $k-1$

(3) $P:=2P+A\cdot B_{k-1-i}$

(4) $P:=P \bmod n$

假定 A，B，$P<n$，则

$P':=2P+A\cdot B_j\leqslant 2(n-1)+(n-1)=3n-3$

因此，P' 对 n 的取余运算实际上用两次减 $P'-n$ 操作即可实现，因为：

$0\leqslant P'-2n<n$

此时，算法描述为：

(1) $P:=0$；

(2) for $i=k-1$ downto 0

(3) $P:=2P$

(4) if Carry-out then $P:=P+m$

(5) $P:=P+A\cdot B_i$

(6) if Carry-out then $P:=P+m$

其中 $m=2^k-n$，实际上，当 carry-out=1 时，$P=2P-2^k$ 或 $P=P+A\cdot B_i-2^k$，此时，$P=P+m=2P-2^k+2^k-n=2P-n$ 或 $P=P+m=P+A\cdot B_i-2^k+2^k-n=P+A\cdot B_i-n$。

在上述算法中，$P:=2P$ 操作用移位方法易于实现，但 $P+m$ 和 $P+A$ 操作是 512~2048 位的加法，如果采用进位传递的加法器，需要 512~2048 级的进位链，这是不可以忍受的，如果采用先行进位的加法器，则硬件的代价太大 通常认为，当加法的位数大于 256 位，采用先行进位的加法器的性能价格比会很低)。

文献[2]中介绍了使用进位保存加法 (Carry Save Adder)，可以大大减少进位传递的时间，同时又避免硬件开销的过度增加。下面先介绍进位保存加法 CSA，然后介绍使用进位保存加法完成乘法操作中的多次加法操作。

3.1 进位保存加法 (Carry Save Adder)

进位保存加法的主要功能是将 3 个 k 位整数相加，得到两个整数 C' 和 S：

$C'+S=A+B+C$

上式中的和的第 i 位 S_i 和进位的第 $i+1$ 位 C'_{i+1} 的计算公式为：

$S_i=A_i\oplus B_i\oplus C_i$

$C'_{i+1}=A_iB_i+B_i\oplus C_i+A_i\oplus C_i$

实际上，一个进位保存加法的单元就是一个全加器单元。一个 6 位的 CSA 的示意图如图 1 所示。

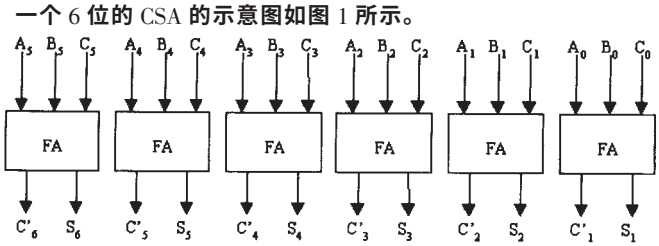

图 1

可见，CSA 对任意位数的加法操作都只需一拍即可完成，但 CSA 并没有完成一次完整的加法操作，因此，如果只进行常规的加法操作，CSA 并不适合，而对于乘法操作，CSA 却可以较为高效地完成。采用 CSA 实现的乘法算法如下所示：

(1) $(C,S):=(0,0)$；

(2) for $i=0$ to $k-1$

(3) $(C,S):=2C+2S+A\cdot B_{k-1-i}$

(4) $(C',S'):=C+S-n$

(5) if SIGN$\geqslant 0$ then $(C,S):=(C',S')$

(6) return (C,S)

其中，SIGN 是判断 (C',S') 的符号的函数。由于 CSA 没有完成完整的加法，因此，对符号位的判定是比较困难的，如果要得到正确的符号判断，就必须完成整字长的加法。一个比较好的方法是对符号位进行预测，到乘法的最后一步进行结果的修正。由此得出的硬件算法如下：

(1) $(C^{(0)},S^{(0)}):=(0,0)$；

(2) for $i=0$ to k

(3) $(C^{(i)},S^{(i)}):=2C^{(i-1)}+2S^{(i-1)}+A_{n-i}\cdot B$

(4) $(C'^{(i)},S'^{(i)}):=C^{(i-1)}+S^{(i-1)}-2N$

(5) If T $(C'^{(i)})+T(S'^{(i)})\geqslant 0$ then $(C^{(i)},S^{(i)}):=(C'^{(i)},S'^{(i)})$

(6) $(C'^{(i)},S'^{(i)}):=C^{(i-1)}+S^{(i-1)}-N$

(7) If $T(C'^{(i)})+T(S'^{(i)})\geqslant 0$ then $(C^{(i)},S^{(i)}):=(C'^{(i)},S'^{(i)})$

(8) End

其中 $T(X)=X-(X \bmod 2^t)$，$0\leqslant t\leqslant n-1$，$T(X)$ 将 X 的低 t 位置为 0，$T(C'^{(i)})+T(S'^{(i)})$ 就是对 $C'^{(i)}$、$S'^{(i)}$ 的高 $n-t$ 位相加的和的符号位的预测。为保证加法的正确性，使用 $(k+3)$ 位的进位保存加法。当 $t=n-1$，$T(X)$ 检查 $C'^{(i)}$，$S'^{(i)}$ 的高 $n-2$ 位到 $n+3$ 位，算法最后产生结果 $(C,S)=(C^{(k)},S^{(k)})$，而 $P=C+S$ 是在$[0,2N)$中。可以在最后用进位传递加法完成 $P=C+S$ 和 $P'=C+S-N$，如果 $P'<0$，最后的结果为 P，否则为 P'。

3.2 性能分析

3.2.1 时间分析

用硬件实现 k-位的 RSA 算法，在最坏情况下，需要完成 $k-1$ 次的平方操作和 $k-1$ 次的乘法运算。在乘法运算中，每次需要进行 $3k$ 次的 CSA 加法和 2 次 k 位的进位传递加法，通常一次 CSA 加法的完成时间为 1 时钟周期，一次 k 位的进位传递加法需要 k/w 个时钟周期 (w 为机器字长)，则完成一次乘法所需时间为 $3k+2k/w$，因此，硬件实现 k-位的 RSA 算法共需：

$T=2(k-1)(3k+2k/w)$

3.2.2 硬件开销分析

用硬件实现 k-位的 RSA 算法，至少需要 $5k$ 的寄存器存放 A、B、C、S 和 n，同时需要 k 个全加器，另外还需要 k 个 2 选 1 的选通器，假定一个寄存器需 10 个 MOS 管，一个全加器需 30 个 MOS 管，一个 2 选 1 的选通器需 8 个 MOS 管，则硬件至少需增加：

$H=5k*10+k*20+k*8=78k$

表 2 在不同的 k 和不同的机器主频下的 RSA 运算时间和硬件。

表 2

主频	k	RSA 运算时间 (秒)	硬件需增加的 MOS 管
5MHz	512	0.33	39936
	1024	1.26	79872
	2048	5.05	159744
10MHz	512	0.16	39936
	1024	0.63	79872
	2048	2.53	159744

由表可见，在增加一定的硬件开销的前提下，RSA 的运算速度可以有明显提高，对 1024 位的 RSA 计算，在 10MHz 的主频下可以在 1 秒以内完成，但增加了 8 万 MOS 管。

(收稿日期：2002 年 6 月)

(下转 78 页)

(C)1994-2019 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

背景区域误认为文字区域。(c)为改进后的 DCT 纹理分析所得结果,即通过改变原输入图像大小,然后将其作为新的输入图像完成纹理分析并将原图像和各新图像所得结果合并。如图 4(c)所示,改进后的方法虽然克服了原方法对文字尺寸的敏感性,可以提取出大尺寸的文字,但其分割结果也变得更粗糙。(d)为单独使用连接组件法的结果,即对整幅图像进行连接组件分析,该方法在准确定位文字的同时,由于整幅图像背景较复杂,将一些背景区域也误认为文字区域提取出来。(e)为使用新算法所得结果,该方法在纹理预分割结果的基础上进行连接组件分析,实现了图中文字的准确定位。

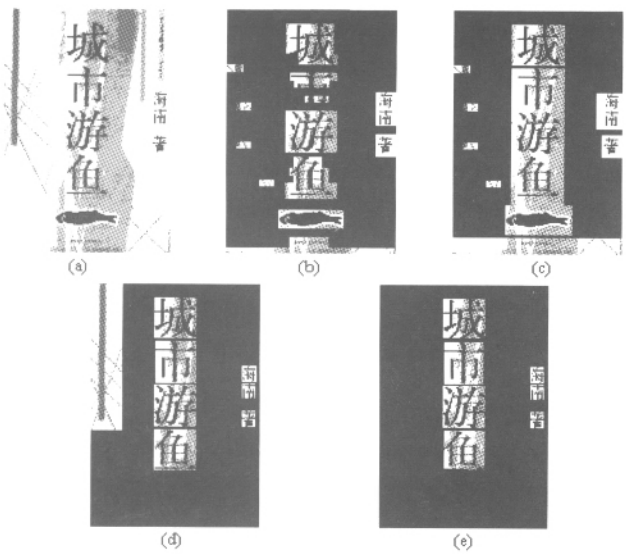

图 4 样本图像及应用不同方法得到的检测结果

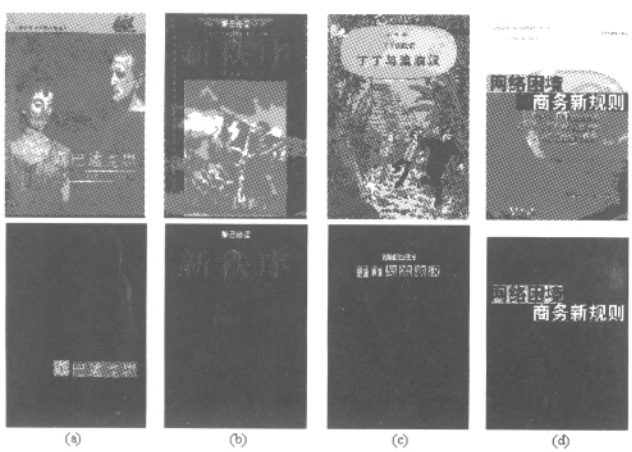

图 5 书籍封面图像及检测结果

应用该文提出的新算法,分别对书籍封面和照片图像进行了测试。图 5 为一组书籍封面图像和相应的检测结果,其中(a)~(c)的大小为 350×510,(d)的大小为 350×481。图 6 为一组照片样本图像和相应检测结果,图像大小均为 410×307。由图示结果可以看出,对于图中清晰且具有一定尺寸的文字,该法均能正确地提取,而对图中一些小的文字(图 5a~5d)和比较模糊、笔画粘连较多的文字(图 5b),该法不能将其取出。然而,如果考虑到定位图像中文字的目的只是为了提取图中的文字区域,并利用 OCR 技术进行识别,而现有 OCR 技术所能识别的单字图像不能小于 16×16,并要求文字的笔画清晰!因此没有必提取尺寸太小或模糊的文字。

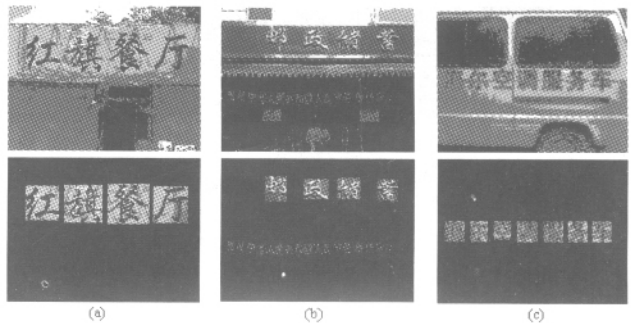

图 6 照片图像及检测结果

4 结论

该文提出了对彩色图像中的汉字进行自动定位和提取的一种新算法。该法克服了 DCT 纹理分析法对文字尺寸的敏感性,也减少了图像背景复杂度对连接组件法的影响,从而使文字定位更准确。应用这一算法对书籍封面图像和包含文字的照片图像进行的测试均取得了比 DCT 纹理分析法或连接组件法更好的效果。(收稿日期:2001 年 10 月)

参考文献

1.A K Jain,S Bhattacharjee.Text Segmentation Using Gabor Filters for Automatic Document Processing[J].Machine Vision and Applications,1992,5(3):169~184

2.Victor Wu,Raghavan Manmatha,Edward M Risemen.TextFinder:An Automatic System to Detect and Recognize Text in Images[J].IEEE Trans Pattern Analysis and Machine Intelligence,1999,21(11):1224~1229

3.Yu Zhong,Hongjiang Zhang,Anil K Jain.Automatic Caption Localization in Compressed Video[J].IEEE Trans Pattern Analysis and Machine Intelligence,2000,22(4):385~392

4.A K Jain,Bin Yu.Automatic Text Location in Images and Video Frames[J].Pattern Recognition,1998,31(12):2055~2076

5.A Jain,B Yu.Document Representation and Its Application to Page Decomposition[J].IEEE Trans Pattern Recognition and Machine Intelligence,1998,20(3):294~308

6.B Yu,A Jain.A Generic System for Form Dropout[J].IEEE Trans Pattern Recognition and Machine Intelligence,1996,18:1127~1134

7.Kenneth R Castleman.Digital Image Processing(影印版)[M].清华大学出版社,1998

(上接 54 页)

参考文献

1.Cetin Kaya Koc.High-Speed RSA Implementation[R].RSA Laboratories Technique Report,1994:www.rsa.com/rsalabs

2.Cetin Kaya Koc.RSA Hardware Implementation[R].RSA Laboratories Technique Report,1995:www.rsa.com/rsalabs

3.B Schneier.Applied Cryptography[M].Second Edition,John Wiley & Sons,Inc,1996

4.R H Cooper,W Patterson.RSA as a Benchmark for Multiprocessor Machines[C].In:Advance in Cryptography-CRYPTO'90 Proceedings,Springer-Verlag,1990:356~359

(C)1994-2019 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net