RSA 非对称加密算法的一个实现

作者：胡辉 17727536501

一 引言

RSA 非对称加密算法应用广泛，实用性强。据我所知，它在数据通信和数据存储领域是普遍使用的基础技术。

科技时代，数学就是魔法，不懂数学者是麻瓜。或用先贤的话，有“不懂几何者不得入此们”的道理。本文重点描述编程方面的数据结构和算法，数学性质和证明是算法成立的依据，在这方面的描述我采取简介形式，力求百分百正确，但不一定够详细。

二 数学性质与证明

大（整）数，二进制 1024 位或 2048 位的大数字因数分解在数学上很困难，当前的计算机硬件和算法据说最多能分解 150 位（比较接近 1024 二进制位，远不到 2048 二进制位）十进制位的数。将一个大数的高次幂对另一个大数模运算取剩余，确是可行并且容易用计算机实现的，大数的这个性质被用来设计一种非对称的加密/解密算法，即 RSA（三位发明者的名字缩写）算法。

本文主题涉及到的数学知识具体是指数论，其他分支不涉及（如果学过抽象代数群论知识，会更容易理解）。我把他们分成三部分（可对应到算法实现的三个部分）来介绍：**算数基本定理之求最大公约数，费马小定理和欧拉公式，拉宾米勒素性测试公式**。另外再单独讲解述 RSA 算法的数学证明。

基本符号和术语：

1.a｜b　a 整除 b，a 是分，即 b 除以 a 没有剩余（剩余为 0）

2.模运算　整数相除，得商与剩余

3.同余式　a≡b mod m，这个是模运算的数学符号描述形式，即数 a 与 b 模 m

同余

4.gcd 最大公约数的数学简写形式，gcd (a,b)即数 a 与 b 的最大公约数

5.a 的 b 次幂记做 a^b

2.1 算数基本定理：

任何整数 n>=2 可唯一分解成素数的乘积形如 n=p1*p2*p3*...*pn

欧几里得带余除法（或称辗转相除法）是求 gcd 的数学算法，要求数 A 与 B 的最大公约数，写成 A=Q*B+R（Q 即商 Quotient 的缩写 R 即余 Remainder 的缩写）。然后将 B 放到公式中 A 的位置，R 放到公式中 B 的位置得到新的 Q 与 R，按此步骤一直进行下去直到得到 R 等于 0，则倒数第二步的 R 值即为数 A 与 B 的最大公约数。

欧几里得算法不仅可用于计算出最大公约数，还可以用于解形如 ax+by=gcd(a,b)的线性方程。即求数 a 与 b 的最大公约数的欧几里得算法过程的中间结果中含有以上方程的根。

2.2 费马小定理和欧拉公式：

p 是任意素数，a 是任意整数且 a 与 p 互素，则有

$$a^{p-1} \equiv 1 \mod p$$

欧拉Φ函数：在 1 与 m 之间且与 m 互素的整数的个数(计数函数),记做Φ(m)。

对于素数 p 有，Φ(p)=p-1

对任意数 m 和 n，如果 gcd(m,n)=1 有Φ(mn)=Φ(m)Φ(n)

对任意素数 p 有　Φ(p^k)=p^k-p^(k-1)

欧拉公式：如果数 a 与 m 互素，记 gcd(a,m)=1，则有

$$a^{\Phi(m)} \equiv 1 \mod m$$

2.3 拉宾米勒素性测试公式：

设 n 是奇数，记 n-1=(2^k)*q，q 是奇数，对不被 n 整除的某个 a，如果下述

两个条件都成立，则 n 是合数。

(a) a^q !≡1 mod n

(b) 对所有 i=0,1,2,...,k-1, a^(2iq) !≡-1 mod n

由于素数分解的困难性，要判断一个数是否是素数没有简单的方法，只有找出它的因数。拉宾米勒测试公式并不能证明一个数是否是素数，它能证明一个数“很可能是素数”。选 a 的 100 个不同值，其中没有一个能通过测试，则 n 是合数的概率近似等于 6*10^(-61)。（非常小，所以可以确定 n 是素数）

2.4 RSA 算法的数学证明:

算法描述：有数字（自然数）a,k,m,b,u，有以下关系，

1.m=p1p2p3...pn，p1 与 p2...pn 两两互不相等，即数 **m 没有相同的因数**。

2.**k 与Φ(m)互素**，即 gcd(k,Φ(m))=1

3.u 是方程 ku≡1 mod(Φ(m))的（同余）解

4. a<m，b 是 a 的 k 次幂模 m 剩余，即有同余方程 a^k≡b mod m

若数字 a 是已知的数，则对 a 求 k 次幂然后模 m 可得数 b

若将以上同余方程中的 **a 看做未知数**，其他三个数 k,b,m 已知数，需要证明方程 x^k≡b mod m 的解是 a。已知 k 与 m 可通过解方程 ku≡1 mod(Φ(m)求出 u，若证明 b^u≡a mod m 即证明了 **b^u（即 a）是方程 x^k≡b mod m 的解**。

以上，数字 a 是原文，b 是密文，**(k,m)构成公钥，(u,m)构成私钥**。加密过程的计算是将原文 a 做 k 次幂后模 m 得剩余 b，将 b 当做密文。解密的过程是将密文 b 做 u 次幂后模 m 得剩余，这个剩余就是原文 a。

以下是证明过程：

由 3 可将同余方程转为等式方程 ku-1=Φ(m)*v, 变形为 ku=Φ(m)v+1

由 4 即 a^k≡b mod m 两边自乘 u 次得 a^ku≡b^u mod m

把上一步中的等式代换进同余式，得 a^(Φ(m)v+1) ≡b^u mod m

左边展开并减 a 得 a(a^Φ(m)v - 1)

另外，m 是不同素数的乘积，a<m，所以 a 不整除 m，假设 gcd(a,m)=p1 可得 gcd(a,p2p3...pn)=1，所以 a^Φ(p2p3...pn)≡1 mod p2p3…pn, 两边自乘Φ(p1)v 次可得 a^Φ(m)v ≡ 1 mod p2p3…pn, 所以 a^Φ(m)v-1 整除 p2p3…pn。

根据假设 gcd(a,m)=p1,所以 a(a^Φ(m)v-1)整除 p1*(p2p3…pn)=m 即

a(a^Φ(m)v-1) ≡ 0 mod m 所以 a^(Φ(m)v+1) ≡a≡ b^u mod m

证明完毕 QED

三 设计目标

根据算法的数学描述，分解问题并制定设计目标。

1.m 是没有相同因数的数，这个条件在当 m 数字不大的时候没有难度，例如在 C 的最大内置整形类型 64 位范围内的数很容易分解出 p1,p2,...,pn.当 m 数字很大，达到常用的 1024 位或 2048 位的时候，直接进行因数分解不可行。但可以反向操作,找到两个 512 位的素数相乘即可得到一个 1024 位的满足需求的大数。(找两个 1024 位的素数相乘得到一个 2048 位的大数，以此类推)

拉宾米勒公式就是用于解决寻找大素数的问题。但是素数的分布特性决定了当数字越大的时候，越稀疏。所以，实际的挑战不仅是拉宾米勒算法的实现效率，而是随机数生成函数的效率。据我的经验，C 的标准库中的伪随机函数 rand()不能满足要求，因为生成的伪随机数函数多次运行值都不会改变，不具有随机性。

这个是我在分解目标上遇到的第一个问题，可能也是最难解的问题。这里的设计目标是快速分解内置类型对象，对于大数字对象，目标是对大数字对象快速完成 100 次拉宾米勒测试，**快速做出素性判断**。

2.k 与Φ(m)互素，即 gcd(k,Φ(m))=1，这个思路很简单，根据辗转相除法容易实现 gcd 函数。在实际使用过程中，k 取一个小的素数就行，可省去求 gcd 的操作。并且 k 一般作为公钥，在运算能力较低的终端设备商使用，k 值小运算量就小一点。u 相应会是一个很大的值，但是一般作为私钥使用，后端服务器运算能力较强。这里的设计目标实现欧几里得除法，不管小数字对象还是大数字对象，都能**快速求出 gcd**。

3.u 是方程 ku≡1 mod(Φ(m))的（同余）解，这个问题有两个解决算法。作为同余方程来解，可套用欧拉公式，此时 k 与 u 互为乘法逆元，知道其中一个值就可求出另一个。也可以把同余方程转为等式方程 ku-Φ(m)v=1 来解，算法就是欧几里得带余除法。两个算法比较，第一个直观，但是要对Φ(m)再求一次（即Φ(Φ

(m))，当 m 是大数的时候，这个算法需要对Φ(m)分解，所以不可行。所以做大数对象计算的时候，选择第二个算法。这里的设计目标是实现两个算法，第一个算法最多处理内置类型的小数字对象，第二个算法实际上就是 gcd 函数（目标同第 2 条，**快速计算小数字和大数字的 gcd 并得到方程的解**）

4.a<m，b 是 a 的 k 次幂模 m 剩余，即有同余方程 a^k≡b mod m，这个直接解不可行。因为对大数字求大数字次幂，不论是存储空间还是 cpu 计算能力都很难满足。所以用逐次平方求剩余算法，一次只计算对象的最多 2 次幂，不会产生超大数字的中间结果。“逐次平方”算法就是实际加密和解密运算直接使用的算法。这里的设计目标是**快速计算出大数字的大数字次幂模大数字 m 剩余**。

四 数据结构

我们要计算的对象是自然数，用计算机程序来描述自然就是整形类型，而且不需要扩张到负数，使用**补码无符号类型**就足够满足计算需要。

C 的内置整形类型对象最多 8 个字节，我们进行大数字计算用比较大的字节数组来定义，要一定程度上兼顾计算效率，所以数组采用 **4 字节对齐**的策略。做加法和乘法运算时，可以一次取 4 字节进行 0 扩展为一个 unsigned long int 型（8 字节）对象，保证容纳下加法和乘法运算结果的溢出部分。

在当前普及使用的 64 位硬件设备商，4 字节（和 8 字节）对象计算效率比单字节对象快 4 倍(例如，单字节，双字或四字对象的加法都是一条指令，将双字对象拆分成 4 个单字节对象，就需要至少 4 条指令完成计算)，副作用是无法兼顾小端与大端机器。我这里选择**基于小端硬件**来设计数据结构。但是在最直观上要符合数学书写习惯，在数组的左端（低地址）部分的 4 字节对象是整个数字的最高位，在右端（高地址）部分的 4 字节对象是整个数字的最低位。

基本约定:

一个二进制位对象用字母 b 表示，下标表示地址顺序。位对象没有简称，就是 bit

单字节对象用字母 B 表示，下标表示地址顺序。单字节对象没有简称，就是 Byte

四字节对象用字母 D 表示，下标表示地址顺序。四字节对象简称双字 Double Word

八字节对象用字母 Q 表示，下表表示地址顺序。八字节对象简称四字 Quad Word

大数字对象数据结构（数组）的实现：

数字对象 a 定义为 a[BLOCKSIZE * 2]，被设计为双字对齐。

a 展开为：a = {D[0],D[1],D[2],…,D[BLOCKSIZE*2/4 – 1]}

D 展开为：D = {B[0],B[1],B[2],B[3]}

需要注意 a 和 D 的展开都是直观的按地址从低到高进行，由于按照小端法解释，B[0]是 D 的最低位 B[3]是 D 的最高位。而 D[0]是 a 的最高位 D[BOCKSIZE*2/4 -1]是 a 的最低位。

在 x86 64 位机器上，加法有 addb addw addl addq 四种指令，最大只能一次计算 8 字节对象。我的设计思路是用应用层的软件模拟去掉这个限制，用类似 add1024 这样的**函数模拟**大数字加法指令。函数内部把数字拆开逐个计算，在函数外部看就是一条加法指令。以此类推，减法，乘法甚至除法都用这个思路模拟。

因为整形对象的计算实际上是模运算（w 位对象的加法和乘法计算就是模 2^w 运算），容易产生溢出并且直接丢弃掉计算结果中的溢出部分。我们的加解密计算结果要求数学上是准确的，任何溢出都必须得到考虑，默认保留，特定情况下才丢掉溢出。溢出的处理是设计和实现细节上需要特别注意的地方之一，所以我用一个宏 BLOCKSIZE 约定数组的大小。实际定义大数字的时候，需要定义双倍大小的数组。数组的前半段 0 到 BLOCKSIZE-1 置零，后半段 BLOCSIZE 到 BLOCKSIZE-1 放有效内容。除存储大数乘法计算的结果之外，其他情况下数组前半段都被视为无效内容。

若读者想要理解与检验这个数组的结构，可阅读 NumberToString 和 StringToNumber 函数源码。

五 算法

具体的算法根据计算对象的大小，从两个角度考虑，小数字（就是内置类型

对象）算法和大数字（就是大数组结构对象）算法。小数字的因数分解，解同余方程，求幂模剩余等的实现函数有时是大数字对象同样功能函数的底层函数。

在实际的应用中，一个加密系统的实现，通常不会仅仅只追求安全性，把所有数据都是用最高等级 2048 位加密，要兼顾效率去适应不同运算能力的硬件。此时，常见的做法是用最高等级加密来做用户身份认证，核心数据保护这类的场景。即时通信内容，不是很重要的数据文件等，使用小数字位加密就可以满足需求。此时，实现小数字的算法的函数甚至可以直接派上用场。

5.1 生成小素数表

使用最直接的方式即尝试将一个数分解来判断它是否是素数，分解一个数 a 需要尝试用 a 去逐个除以不大于根号 a（用了标准库里面的浮点数运算函数 sqrt）的素数。如果分解成功，则 a 不是素数，否则 a 是素数，此算法能够从最小素数 2 开始给出连续的素数列表。

对应实现为函数 createPtable，生成的素数我把它们放在全局数组 pTable 中。

5.2 内置整形对象的因数分解

算法同生成小素数算法，利用 createPtable 生成的素数表来测试和分解给定的数，对应实现为函数 primeFact

5.3 求最大公约数和解等式方程

将欧几里得除法用 C 语言代码逻辑实现，不管小数字还是大数字，数学算法上没有任何区别，代码逻辑上的区别是由于大数字与小数字的数据结构差别导致。

题目描述：求解等式方程 ax+by=gcd(a,b)=1（用字母 g 代换 gcd(x,y)，等式方程等价于同余式方程 ax≡g mod b 在等式方程的意义下有无穷多解，我们只需要求出值在范围(0,b)之间的一个特解，特解（或称同余方程的解）有且仅有一个。

算法步骤如下：

1. 置 x=1，g=a，v=0 与 w=b

2. 如果 w=0，则置 y=（g-ax）/b 返回值(g,x,y)
3. g 除以 w 得余 t，g=qw+t (0<=t<w)
4. 置 s=x-qv
5. 置(x,y)=(v,w)
6. 置(v,w)=(s,t)
7. 转到第 2 步

对小数字计算的算法实现函数为 gcd 和 linearEquition，对大数字对象计算的算法实现为函数 gcdBigNum 和 ModMROneEquition

5.4 欧拉 fai 函数

将一个整数完全分解，把因数分成重复部分（幂大于 1）和不重复部分（幂等于 1），使用Φ函数的两个计算规则即可求出函数值。

这里只开发小数字的 fai 函数实现，不做大数字计算。正是大数字因数分解的困难性导致 fai 函数计算工作量太大，这是 RSA 密码体制成立的依据。假如能实现大数字的 fai 函数求值，就可以通过公钥计算得出私钥，破解掉密码。

假设计算对象为 x，算法步骤如下：

1. 分解 x 得到完全分解的因数列表
2. 将因数列表中的每一个因数与它的指数对应
3. 应用规则Φ(mn)=Φ(m)Φ(n), Φ(p^k)=p^k-p(k-1)

fai 函数的实现函数为 funcFai

5.5 大数字对象加法

加法运算的溢出最多只有一个二进制位，溢出值是 0 或者 1。所以用一个变量做进位标记，在低位的双字对象做加法过程中置标记并累加到高一位的双字对象加法中。

加法实现函数为 addBigNum，计算对象的位数由 BLOCKSIZE 宏控制，常用值是 128 或 256，对应 1024 位和 2048 位大数字。

5.6 大数字对象减法（和取加法逆）

大数字对象加法运算也是模上的运算，1024 位大数字加法即是模 2^1024 上的加法，减法 a-b 就是 a+(-b)，-b 是 a 的加法逆元。由于我们的数字采用补码编码，将 a，b 以及加法或减法计算结果解释为有符号数，在不发生算数溢出的情况下，跟通常的自然数数学计算结果是一致的。

大数字对象的减法实现函数为 subBigNum，取加法逆实现函数为 addReverse

5.7 大数字对象乘法

乘法运算有比较大的计算量，两个双字对象 a 和 b 相乘结果最多能溢出 32 位，所以不能像加法那样简单的使用一个标记变量。这里把 a 和 b 进行 0 扩展到 64 位的临时变量，做乘法，再通过移位得到结果中溢出部分和非溢出部分。

A 和 B 是两个大数字对象，a 和 b 分别是他们展开出来的双字对象，算法步骤就是常用竖式乘法：

1. A 乘 B 的最低位双字对象
2. A 乘 B 的第二低位双字对象
3. A 乘 B 的第三低位双字对象，一直到 B 的最高位双字对象。
4. 将以上左右的计算结果累加起来

大数字对象乘法实现函数为 multBigNum

5.8 大数字对象的除法（和对模求剩余）

数学意义上的整数除法和对模求剩余运算，在计算机上通常用减法实现，x86 硬件上有 ISA 层的除法指令，ARM 硬件上没有。

要做大数字对象的除法，不管用哪种方式都是个复杂的运算。所以我设计了两种除法实现，一种针对两个宽度相近的大数字对象，用减法实现，另一中针对大数字除以小数字，用除法。

在我们的 RSA 加密解密运算中，使用除法的绝大部分场景都是对模取剩余计算，并不需要商，这种情况下用减法实现的想法非常直观。少数场景下需要除法

的商（具体就是解等式方程）。之所以分两种除法实现，是基于运算效率的平衡，当一个大数字对象除以一个小数字对象，用减法实现的运算量太大而不实用（*可优化到实用等级）。

我的算法设计足够满足常用两个场景下的除法运算需求，求幂模剩余场景下使用大数字除以大数字的减法实现，取小数字 k 值求 u 值的解等式方程场景下，用大数字除以小数字的除法实现。

大数字除法实现函数为 modBigNum，modBigNumWithQuo，sDivideBigNum，sDivideBigNumWithQuo 四个前缀 s 表示 special，表示做大数字除小数字除法的意思。

5.9 幂模求剩余算法（或称逐次平方求剩余算法）

逐次平方计算 a^k mod m 的数学描述：

1. 将数字 k 进行二进制展开，形如

k=(2^0)*u[0]+(2^1)u[1]+(2^2)*u[2]+...+(2^r)*u[r] 其中 u 是二进制位的值，2 的幂即是权的值

2. 逐次平方制作模 m 的幂次表，利用性质 a^2 mod m 等于 a mod m 的平方，a^4 mod m 等于 a^2 mod m 的平方。所以次数再高也只需要对两个小于 m 的数做乘法然后模 m 求剩余。用代码实现的时候当然不需要存储一张这样的“幂次表”，只需要把它们作为中间结果存在同一个对象中。
3. 乘积（累乘）

a^k=a^((2^0)*u[0]+(2^1)u[1]+(2^2)*u[2]+...+(2^r)*u[r])

=a^((2^0)*u[0]))*a^((2^1)*u[1]))*...

代码实现逻辑描述：

1. 置 b=1
2. 当 k>=1 时，循环

2.1 如果 k 是奇数，则置 b=a*b(mod m)

2.2 置 a=a^2 mod m

2.3 置 k=k/2

3. 结束循环，b 的值就是最终计算结果

幂模求剩余算法的实现函数为 powerMode 和 powerModeBigNum，分别计算小数字和大数字对象。

5.10 拉宾米勒测试

拉宾米勒测试算法数学语言描述：

设 n 是奇数，记 n-1=(2^k)*q，q 是奇数，对不被 n 整除的某个 a，如果下述两个条件都成立，则 n 是合数。

(a) a^q !≡1 mod n

(b) 对所有 i=0,1,2,...,k-1, a^((2^i)q) !≡−1 mod n

有了前面的基础算法，拉宾米勒测试函数的设计变得很清晰了。需要用到运算只有两个，幂模求剩余以及 2^k 乘法和除法。幂模求剩余算法已经实现，2^k 乘法和除法我这里用逻辑移位运算实现（也可以用前面的通用乘法除法，用移位更直观，且运算效率高很多很多）。

逻辑移位运算的实现函数为 sllBigNum, slrBigNum，拉宾米勒测试实现函数为 rabinMiller

输入被测试的奇数 x 和任意一个素数 a，rabinMiller 函数执行一次，如果返回 1 即证明了被测试对象为合数，如果返回 0 证明被测试对象是合数的概率小于 0.25。用 100 个不同的素数 a 来测试同一个奇数 x，当函数返回值都是 0 的情况下，x 是合数的概率小于 0-.25 的 100 次幂。如果有一次返回 1，则证明 x 是合数。

六 代码实现（C 语言）

https://github.com/adam-hh/lean01.git