RSA 算法

一、基础数论

1、**互质关系**

如果两个正整数，除了 1 以外，没有其他公因子，我们就称这两个数是互质关系（coprime）。比如，15 和 32 没有公因子，所以它们是互质关系。这说明，不是质数也可以构成互质关系。

2、**欧拉函数**

定义：任意给定正整数 n，请问在**小于等于 n 的正整数之中**，**有多少个与 n 构成互质关系**？（比如，在 1 到 8 之中，有多少个数与 8 构成互质关系？），计算这个值的方法就叫做**欧拉函数，以 φ(n)**表示。

欧拉函数求法及性质：

1.对于素数 p, φ(p)=p-1，**对于对两个素数 p,q，** **//φ（pq）=pq-1**，
欧拉函数是积性函数,但不是完全积性函数.

2.对于一个正整数 N 的素数幂分解 N=P1^q1*P2^q2*...*Pn^qn，则
φ(N)=N*(1-1/P1)*(1-1/P2)*...*(1-1/Pn).

3.**除了 N=2,φ(N)都是偶数**.

4.如果 n **可以分解成两个互质的整数之积**，n = p1 × p2，
则 φ(n) = φ(p1p2) = φ(p1)φ(p2)

二、RSA 加密

第一步，随机选择两个不相等的质数 p 和 q。

A 选择了 61 和 53。（**实际应用中，这两个质数越大，就越难破解**。）

第二步，计算 p 和 q 的乘积 n。

A 把 61 和 53 相乘。

$n = 61\times53 = 3233$

n 的长度就是密钥长度。3233 **写成二进制**是 110010100001，
一共有 12 位，所以这个密钥就是 12 位。
实际应用中，RSA 密钥一般是 1024 位，重要场合则为 2048 位。

第三步，计算 n 的欧拉函数 $\phi(n)$。

根据公式：

$\phi(n) = (p-1)(q-1)$

A 算出 $\phi(3233)$等于 60×52，即 3120。

第四步，**随机选择一个整数 e，条件是 $1< e < \phi(n)$，且 e 与 $\phi(n)$ 互质**。

A 就在 1 到 3120 之间，随机选择了 17。（//实际应用中，常常选择 65537。）

第五步，计算 e 对于 $\phi(n)$的模反元素 d。

所谓"模反元素"**就是指有一个整数 d，可以使得 ed 被 $\phi(n)$除的余数为 1**。

$ed \equiv 1 \pmod{\phi(n)}$

这个式子**等价于**

$ed - 1 = k\phi(n)$

于是，找到模反元素 d，实质上就是对下面这个**二元一次方程求解**。

$ex + \phi(n)*k = 1$

已知 e=17, ϕ(n)=3120,

$17x + 3120y = 1$

这个方程可以用"**扩展欧几里得算法**"求解，此处省略具体过程。
总之，A 算出一组整数解为 (x,y)=(2753,-15)，即 d=2753。

至此所有计算完成。

第六步，**将 n 和 e 封装成公钥，n 和 d 封装成私钥**。

在 A 的例子中，n=3233，e=17，d=2753，
所以公钥就是 (3233,17)，私钥就是（3233, 2753）。

实际应用中，公钥和私钥的数据都采用 ASN.1 格式表达（实例）。

RSA 算法的可靠性

回顾上面的密钥生成步骤，一共出现六个数字：

p
q
n
ϕ(n)
e
d

这六个数字之中，公钥用到了两个（n 和 e），其余四个数字都是不公开的。其中最关键的是 d，因为 n 和 d 组成了私钥，一旦 d 泄漏，就等于私钥泄漏。

那么，有无可能在已知 n 和 e 的情况下，推导出 d？

（1）ed≡1 (mod φ(n))。只有知道 e 和 φ(n)，才能算出 d。

（2）φ(n)=(p-1)(q-1)。只有知道 p 和 q，才能算出 φ(n)。

（3）n=pq。只有将 n 因数分解，才能算出 p 和 q。

结论：**如果 n 可以被因数分解，d 就可以算出**，也就意味着私钥被破解。

可是，**大整数的因数分解，是一件非常困难的事情**。目前，除了暴力破解，还没有发现别的有效方法。维基百科这样写道：

"对极大整数做因数分解的难度决定了 RSA 算法的可靠性。换言之，**对一极大整数做因数分解愈困难，RSA 算法愈可靠**。

假如有人找到一种快速因数分解的算法，那么 RSA 的可靠性就会极度下降。但找到这样的算法的可能性是非常小的。今天只有短的 RSA 密钥才可能被暴力破解。到 2008 年为止，世界上还没有任何可靠的攻击 RSA 算法的方式。

只要密钥长度足够长，用 RSA 加密的信息实际上是不能被解破的。"

举例来说，你可以对 3233 进行因数分解（61×53），但是你没法对下面这个整数进行因数分解。

12301866845301177551304949
58384962720772853569595334
79219732245215172640050726
36575187452021997864693899
56474942774063845925192557
32630345373154826850791702
61221429134616704292143116
02221240479274737794080665
351419597459856902143413

它等于这样两个质数的乘积：

33478071698956898786044169
84821269081770479498371376
85689124313889828837938780
02287614711652531743087737
814467999489
×
36746043666799590428244633
79962795263227915816434308
76426760322838157396665112
79233373417143396810270092
798736308917

事实上，这大概是人类已经分解的最大整数（232 个十进制位，768 个二进制位）。比它更大的因数分解，还没有被报道过，因此目前被破解的最长 RSA 密钥就是 768 位。

加密和解密

有了公钥和密钥，就能进行加密和解密了。

（1）**加密要用公钥 (n,e)**

假设 B 要向 A 发送加密信息 m，他就要**用 A 的公钥 (n,e) 对 m 进行加密**。这里需要注意，**m 必须是整数（字符串可以取 ascii 值或 unicode 值），且 m 必须小于 n**。

所谓"加密"，就是算出下式的 c：

$$m^e \equiv c \pmod{n}$$

A 的公钥是 (3233, 17)，B 的 m 假设是 65，那么可以算出下面的等式：

$$65^{17} \equiv 2790 \pmod{3233}$$

于是，c 等于 2790，B 就把 2790 发给了 A。

（2）解密要用私钥(n,d)

A 拿到 B 发来的 2790 以后，就用自己的私钥(3233, 2753) 进行解密。可以证明，下面的等式一定成立：

$c^d \equiv m \pmod{n}$

也就是说，c 的 d 次方除以 n 的余数为 m。现在，c 等于 2790，私钥是(3233, 2753)，那么，A 算出

$2790^{2753} \equiv 65 \pmod{3233}$

因此，A 知道了 B 加密前的原文就是 65。

至此，"加密--解密"的整个过程全部完成。

我们可以看到，如果不知道 d，就没有办法从 c 求出 m。而前面已经说过，**要知道 d 就必须分解 n，这是极难做到的**，所以 RSA 算法保证了通信安全。

你可能会问，公钥(n,e) 只能加密小于 n 的整数 m，**那么如果要加密大于 n 的整数，该怎么办**？有两种解决方法：
一种是把长信息分割成若干段短消息，每段分别加密；
另一种是**先选择一种"对称性加密算法"**（比如 DES），
用这种算法的密钥加密信息，**再用 RSA 公钥加密 DES 密钥**。

私钥解密的证明

最后，我们来证明，为什么用私钥解密，一定可以正确地得到 m。也就是证明下面这个式子：

$c^d \equiv m \pmod{n}$

因为，根据加密规则

$m^e \equiv c \pmod{n}$

于是，c 可以写成下面的形式：

$$c = m^e - kn$$

将 c 代入要我们要证明的那个解密规则：

$$(m^e - kn)^d \equiv m \pmod n$$

它等同于求证

$$m^{ed} \equiv m \pmod n$$

由于

$$ed \equiv 1 \pmod{\phi(n)}$$

所以

$$ed = h\phi(n)+1$$

将 ed 代入：

$$m^{h\varphi(n)+1} \equiv m \pmod n$$

接下来，分成两种情况证明上面这个式子。

（1）m 与 n 互质。

根据欧拉定理，此时

$$m^{\varphi(n)} \equiv 1 \pmod n$$

得到

$$(m^{\varphi(n)})^h \times m \equiv m \pmod n$$

原式得到证明。

（2）m 与 n 不是互质关系。

此时，由于 n 等于质数 p 和 q 的乘积，所以 m 必然等于 kp 或 kq。

以 m = kp 为例，考虑到这时 k 与 q 必然互质，则根据欧拉定理，下面的式子成立：

$$(kp)^{q-1} \equiv 1 \pmod q$$

进一步得到

$$[(kp)^{q-1}]^{h(p-1)} \times kp \equiv kp \pmod q$$

即

$$(kp)^{ed} \equiv kp \pmod q$$

将它改写成下面的等式

$$(kp)^{ed} = tq + kp$$

这时 t 必然能被 p 整除，**即 t=t'p**

$$(kp)^{ed} = t'pq + kp$$

因为 m=kp，n=pq，所以

$$m^{ed} \equiv m \pmod n$$

原式得到证明。

RSA 数字签名

一 用 RSA 生成签名

在 RSA 中，被签名的消息、密钥以及最终生成的签名都是以数字形式表示的。

在对文本进行签名时，需要事先对文本编码成数字。

签名 = $\textit{消息}^{D}$ mod N　　　（用 RSA 生成签名）

这里所使用的 D 和 N 就是签名者的私钥。

签名就是对消息的 D 次方求 mod N 的结果，也就是说将消息和自己相乘 D 次，然后再除以 N 求余数，最后求得的余数就是签名。

生成签名后，发送者就可以将消息和签名发送给接收者了。

二 用 RSA 验证签名

由签名求得的消息 = $\textit{签名}^{D}$ mod N　　　（用 RSA 验证签名）

这里所使用的 E 和 N 就是签名者的公钥。
接收者计算签名的 E 次方并求 mod N，
得到“由签名求得的消息”，并将其与发送过来的“消息”内容进行对比，
如果两者一致，则签名验证成功，否则签名验证失败。

数字签名的作用:保证数据完整性,机密性和发送方角色的不可抵赖性
加密与签字结合时，两套公私钥是不同的。

为了方面演示，手动生成一个密钥对

（项目中的秘钥对由开发来生成，会直接给到我们）

生成秘钥对的时候，可以指定生成秘钥的长度，

一般推荐使用 1024bit, 1024bit 的 rsa 公钥，加密数据时，最多只能加密 117byte 的数据），数据量超过这个数，则需要对数据进行分段加密。

但是目前 1024bit 长度的秘钥已经被证明了不够安全，尽量使用 2048bit 长度的秘钥。2048bit 长度的秘钥，最多 245byte 长度的数据。

上面生成秘钥的时候提到过在我们加密的时候，如果数据长度超过了当前秘钥的所能处理最大长度，则需要进行分段加密，

分段加密：通俗易懂的讲就是把原来一长串的数据，分割成多段，每段的大小控制在秘钥的最大加密数量之内，加密完了之后再把数据进行拼接。

分段解密：经过分段加密的数据，在进行解密的时候我们也要将它进行分成多段，然后解密之后再进行拼接就能得到原来的数据内容。