[一、简介](#5430-1541643336531)

[二、数学基础](#3936-1541642057985)

[2.1 互质关系](#6411-1541642082329)

[2.2 欧拉函数](#1995-1541642379056)

[2.3 欧拉定理](#2085-1541642953764)

[2.4 模反元素](#5084-1541643862284)

[三、RSA秘钥的生成](#6027-1541643308481)

[四、RSA 加密和解密](#5076-1541649269413)

[4.1 加密 - 公钥 (n, e)](#6786-1541643623377)

[4.2 解密 - 私钥 (n, d)](#2293-1541656618640)

[五、利用 openssl实现RSA、AES数据加密](#2723-1541656841876)

[5.1 下载和安装](#4615-1541657154740)

[5.2 生成 RSA 公钥和私钥](#5850-1541660640320)

[5.2.1 第一种生成方式](#2587-1541661286555)

[5.2.2 第二种生成方式](#5729-1541660785595)

[5.3 利用 RSA 的公钥和私钥进行加解密](#1720-1541661481300)

[5.4 RSA API 的介绍](#7470-1541661756787)

[5.4.1 The Basic](#7861-1541664674698)

[5.4.2 RSA\_generate\_key](#5326-1541664689903)

[5.4.3 RSA\_public\_encrypt](#3540-1541664861671)

[5.4.4 RSA\_private\_encrypt](#4988-1541665055709)

[5.4.5 RSA\_public\_decrypt](#8032-1541665199953)

[5.4.6 RSA\_private\_decrypt](#6027-1541665200466)

**一、简介**

RSA加密算法是一种非对称加密算法。

(对称加密 DES/3DES/AES/SM1：加解密双方使用同一套密钥，即甲用密钥加密，乙还得用与甲同样的密钥来减密

非对称加密：[公开密钥](https://baike.baidu.com/item/%E5%85%AC%E5%BC%80%E5%AF%86%E9%92%A5/7453570)（publickey）和私有密钥（privatekey）。公开密钥与私有密钥是一对，如果用公开密钥对数据进行加密，只有用对应的私有密钥才能解密；如果用私有密钥对数据进行加密，那么只有用对应的公开密钥才能解密。因为加密和解密使用的是两个不同的密钥，所以这种算法叫作非对称加密算法。)

**二、数学基础**

**2.1 互质关系**

质数=素数=只能被1和它本身整除

互质关系：如果两个正整数，除了1以外，没有其他公因子，那这两个数就是互质关系。

**任意两个质数构成互质关系**，比如13和61。

**一个数是质数，另一个数只要不是前者的倍数，两者就构成互质关系**，比如3和10

**如果两个数之中，较大的那个数是质数，则两者构成互质关系**，比如97和57。

**1和任意一个自然数是都是互质关系**，比如1和99。

**p是大于1的整数，则p和p-1构成互质关系**，比如57和56。

**p是大于1的奇数，则p和p-2构成互质关系**，比如17和15。

**2.2 欧拉函数**

在任意的正整数n，请问在小于等于n的正整数中，有多少个与n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。

比如正整数8，φ(8)=？，很快算出有1、3、5、7，所以φ(8)=4

如果n可以分解成两个互质的整数之积，即**n=p×q**，则有：**φ(n)=φ(pq)=φ(p)φ(q)**;

2. 一个数如果是质数，则小于它的所有正整数与它都是互质数；所以如果一个数p是质数，则有：**φ(p)=p-1**

3. 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)**

4. 如果n是质数的某一个次方，即 **n = p^k** (p为质数，k为大于等于1的整数)，则

**φ(p^k)=p^k-p^(k-1)=p^k(1-1/p).**

比如： φ(8) = φ(2^3) =2^3 - 2^2 = 8 -4 = 4; φ(8) = φ(2^3)=2^3(1-1/2)=4

5. 因为任意一个大于1的正整数，都可以写成一系列质数的积：

n=p1^k1p2^k2…pr^kr，根据第三条定律，则φ(n)=φ(p1^k1)φ(p2^k2)…φ(pr^kr)

再根据第四条定律，则φ(n)=p1^k1p2^k2…pr^kr(1-1/p1)(1-1/p2)…(1-1/pr)，也就等于 φ(n)=n(1-1/p1)(1-1/p2)…(1-1/pr)，

举例：1323的欧拉函数，计算过程如下：

φ(1323)=φ(3^3x7^2)=1323(1-1/3)(1-1/7)=756

**2.3 欧拉定理**

如果两个正整数a和n互质，则n的欧拉函数φ(n)可以让下面的等式成立：

a^φ(n)≡1 mod(n)  读作 a 的 φ(n) 次方 - 1 可以被n整除

比如 a = 3，n = 7；

**2.4 模反元素**

如果两个正整数a和n互质，那么一定可以找到整数b，使得 ab-1 被n整除，或者说ab被n除的余数是1。

根据欧拉定理有：**a^φ(n)=axa^φ(n-1)≡1mod(n)，令b=a^φ(n-1)，ab≡1mod(n)，这里b就是a的模反元素。**

**三、RSA秘钥的生成**

在前面的基础上进行 RSA 密钥的生成。

1. 任取两个不相等的质数 p 和 q

2. 比如 p = 61， q = 53， 则 n = 61\*53 = 3233

3233 写成二进制是 1101 1010 0001 一共12位。**实际应用中,RSA密钥一般是1024位，重要场合则为2048位**

3. 计算n的欧拉函数φ(n) : φ(3233) = 60 \* 52 = 3120

4. 随机取一个整数e，使其满足 1 < e < φ(n)，且 e 与 φ(n) 互质

5. 计算e对于φ(n)的模反元素d

根据前面模反元素公式e^φ(N)=exe^φ(N-1)≡1mod(N)，这个公式中的N等于这里的φ(n), 令d=e^φ(N-1), 则ed≡1mod(N)，即

ed ≡ 1 (mod φ(n))

这个式子等价于

ed - 1 = kφ(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 + φ(n)y = 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，此处省略具体的计算过程，有兴趣可以研究下这个算法，算出一组整数解为 (x,y)=(2753,-15)，即 d=2753。至此所有计算完成。

6. 将 n 和 e 封装成公钥，n 和 d 封装成私钥

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，ASN.1是抽象语法标记(Abstract Syntax Notation One)，它是一个 ISO/ITU-T国际标准，简单说就是加一些头部信息，有兴趣可以熟悉下ASN.1。

**四、RSA 加密和解密**

**4.1 加密 - 公钥 (n, e)**

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 ≡ c (mod n) 公钥是 (3233, 17)，m假设是65，那么可以算出下面的等式：

65^17 ≡ 2790 (mod 3233)

于是，c等于2790，即加密后的密文

**4.2 解密 - 私钥 (n, d)**

用私钥(3233, 2753) 进行解密。可以证明，下面的等式一定成立：

c^d ≡ m (mod 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： 2790^2753 ≡ 65 (mod 3233)

**当然，这么复杂的算法我们不可能自己去写，必须去借用一些依赖库**

**五、利用 openssl实现RSA、AES数据加密**

  openssl是可以很方便加密解密的库，可以使用它来对需要在网络中传输的数据加密。可以使用非对称加密：公钥加密，私钥解密。openssl提供了对RSA的支持，**但RSA存在计算效率低的问题，所以一般的做法是使用对称密钥加密数据，然后再把这个只在当前有效的临时生成的对称密钥用非对称密钥的公钥加密之后传递给目标方**，目标方使用约定好的非对称密钥中的私钥解开，得到数据加密的密钥，再进行数据解密，得到数据，这种使用方式很常见，可以认为是对HTTPS的裁剪。对称密钥加密可以选择AES，比DES更优秀。

**5.1 下载和安装**

|  |
| --- |
| sudo apt-get install openssl |

|  |
| --- |
| git source code |
| https://github.com/openssl/openssl |

**5.2 生成 RSA 公钥和私钥**

**5.2.1 第一种生成方式**

|  |
| --- |
| 生成私钥 |
| openssl genrsa -out test.key 1024 |

|  |
| --- |
| 生成公钥 |
| openssl rsa -in test.key -pubout -out test\_pub.key |

**5.2.2 第二种生成方式**

|  |
| --- |
| 进入 openssl |
| openssl |

|  |
| --- |
| 生成私钥 |
| genrsa -out rsa\_private\_key.pem 1024 |

|  |
| --- |
| 生成公钥 |
| rsa -in rsa\_private\_key.pem -pubout -out rsa\_public\_key.pem |

**5.3 利用 RSA 的公钥和私钥进行加解密**

|  |
| --- |
| 加密 |
| openssl rsautl -encrypt -in hello -inkey test\_pub.key -pubin -out hello.en |

|  |
| --- |
| 解密 |
| openssl rsautl -decrypt -in hello.en -inkey test.key -out hello.de |

**5.4 RSA API 的介绍**

<https://www.cnblogs.com/aixiaoxiaoyu/p/8195626.html>

**5.4.1 The Basic**

|  |
| --- |
| 对照欧拉公式随便看看 |
| typedef struct rsa\_st  {  BIGNUM \*p;  BIGNUM \*q;  BIGNUM \*n;  BIGNUM \*e;(e使用(p-1)(q-1))  BIGNUM \*d; (使用e,p和q计算)  } RSA |

**5.4.2 RSA\_generate\_key**

|  |  |
| --- | --- |
| (RSA \*RSA\_generate\_key(int bits, unsigned long e, void  (\*callback) (int, int, void \*),  void \*cb\_arg)) | |
| 功能 | RSA算法不需要参数来生成密钥，这使得密钥生成变得非常简单 |
| num | 指定公有模数中的位数 |
| e | 用于公有指数的值 |
| callback | 指向在素数生成过程中将被调用的函数的指针，用于报告素数生成的状态 |
| cb\_arg | 指向特定于应用程序的数据的指针。 |

**5.4.3 RSA\_public\_encrypt**

|  |  |
| --- | --- |
| int RSA\_public\_encrypt(int flen, const unsigned char \*from,  unsigned char \*to, RSA \*rsa, int padding); | |
| 功能 | RSA公钥加密数据 |
| flen | 指定要加密的缓冲区中的字节数。 |
| from | 包含要加密的数据的缓冲区。 |
| to | 将用于保存加密数据的缓冲区。 |
| rsa | 包含用于执行加密的公钥的RSA对象。 |
| padding | 指定在需要填充时应使用OpenSSL支持哪些内置填充类型。 |

**5.4.4 RSA\_private\_encrypt**

//参数参照 5.4.3

**5.4.5 RSA\_public\_decrypt**

//参数参照 5.4.3

**5.4.6 RSA\_private\_decrypt**

//参数参照 5.4.3