

经典的非对称加密手段。

This is why RSA is not used to encrypt long messages. RSA can only encrypt input data that is smaller than the modulus value of the RSA key. RSA is most commonly used to encrypt temporary session keys (e.g., AES encryption key) which are then used to encrypt long messages.

4. Let's encrypt a short text. The key size is 512bits (64 bytes) so the maximum message size after taking away padding should be 53 bytes.

```
echo "Ethical hacking is fun" | openssl pkeyutl -encrypt -pubin -inkey public.key  
> message.dat
```

5. Trying to encrypt something too long gives you this error (please try)

```
RSA operation error
3080140544:error:0406D00E:rsa routines:RSA_padding_add_PKCS1_type_2:data too larg
e for key size:../crypto/rsa/rsa_pk1.c:125:
```

RSA 算法

```

1 # function to calculate inverse modular
2 # using the extended Euclidean algorithm
3 def invmod(a,n):
4     i=1
5     while True:
6         c = n * i + 1;
7         if(c%a==0):
8             c = c//a
9             break;
10        i = i+1
11    # note: indent before the following statement
12    return c
13
14 p = int("E017",16) # first prime
15 q = int("D20D",16) # second prime
16 e = int("010001",16) # a number that is co-prime with (p-1)*(q-1)
17
18 # calculate modulus n
19 n = p*q
20 print("n is: " + str(n))
21
22 # calculate inverse modular d of exponent e and (p-1)*(q-1)
23 d = invmod(e, (p-1)*(q-1))
24 print("d is: " + str(d))
25
26 # check that d*e mod (p-1)*(q-1) is indeed 1
27 print("checking d*e mod (p-1)*(q-1): " + str(((d*e) % ((p-1)*(q-1)))))
28
29 # encrypting short message using public exponent e
30 msg = 12345
31 enc = pow(msg,e, n) # supplying the 3rd parameter in pow() efficiently computes the mod
32 print("Message " + str(msg) + " is encrypted to: " + str(enc))
33
34 # decrypt message using private exponent d
35 plain = pow(enc,int(d), n) # supplying the 3rd parameter efficiently computes the mod
36 print("Cipher " + str(enc) + " is decrypted to: " + str(plain))

```

```
n is: 3084795691
d is: 224795969
checking d*e mod (p-1)*(q-1): 1
Message 12345 is encrypted to: 1335268360
Cipher 1335268360 is decrypted to: 12345
```

其中的欧拉函数

<https://peienwu.com/totient/>

欧拉函数，可以求得某正整数，含有多少个质数。

定義與公式

對於一個正整數 n ，歐拉函數 $\phi(n)$ 表示閉區間 $[1, n]$ 中的正整數與 n 互質的個數。定義序列 p 為所有正整數 n 質因數分解之後的數字。在不失一般性的假設序列 p 的長度為 r ，以質因數分解表示 n 即為 $n = p_1^{k_1} \cdot p_2^{k_2} \cdots p_r^{k_r}$ ，則歐拉函數滿足以下關係式：

$$\begin{aligned}\phi(n) &= n \left(1 - \frac{1}{p_1}\right) \left(1 - \frac{1}{p_2}\right) \cdots \left(1 - \frac{1}{p_r}\right) \\ &= n \prod_{p|n} \left(1 - \frac{1}{p}\right)\end{aligned}$$

特別定義 $\phi(1) = 1$

來看看實際的例子，以20為例。以質因數分解來看， $20 = 2^2 \times 5$ ，因此序列 $p = 2, 5$ ，則歐拉函數 $\phi(20) = 20 \times \left(1 - \frac{1}{2}\right) \left(1 - \frac{1}{5}\right) = 8$ 稍微驗證一下，從1到20的正整數中，與20互質的有1, 3, 7, 9, 11, 13, 17, 19 一共8個數字。

扩展欧几里得公式

用于求解两个整数a,b 的最大公因数GCD greatest common divisor是什么。



扩展欧几里得算法是欧几里得算法的扩展，用于求解两个整数 a 和 b 的最大公约数 (GCD)，同时找到满足以下等式的整数 x 和 y ：

$$a \cdot x + b \cdot y = \gcd(a, b)$$

算法步骤

1. 初始化：

- 设 $r_0 = a$, $r_1 = b$ 。
- 设 $s_0 = 1$, $s_1 = 0$ 。
- 设 $t_0 = 0$, $t_1 = 1$ 。

2. 迭代计算：

- 对于 $i \geq 1$, 计算商 $q_i = \left\lfloor \frac{r_{i-1}}{r_i} \right\rfloor$ 。
- 更新余数 $r_{i+1} = r_{i-1} - q_i \cdot r_i$ 。
- 更新系数 $s_{i+1} = s_{i-1} - q_i \cdot s_i$ 。
- 更新系数 $t_{i+1} = t_{i-1} - q_i \cdot t_i$ 。
- 重复上述步骤，直到 $r_{i+1} = 0$ 。

3. 结果：

- 当 $r_{i+1} = 0$ 时, r_i 即为 $\gcd(a, b)$ 。
- 系数 s_i 和 t_i 即为满足 $a \cdot s_i + b \cdot t_i = \gcd(a, b)$ 的整数解。

例子 [\[編輯\]](#)

54和24的最大公因數是多少？

數字54可以表示為幾組不同正整數的乘積：

$$54 = 1 \times 54 = 2 \times 27 = 3 \times 18 = 6 \times 9$$

故54的正因數為1, 2, 3, 6, 9, 18, 27, 54。

同樣地，24可以表示為：

$$24 = 1 \times 24 = 2 \times 12 = 3 \times 8 = 4 \times 6$$

故24的正因數為1, 2, 3, 4, 6, 8, 12, 24。

這兩組數列中的共同元素即為54和24的公因數：

$$1, 2, 3, 6$$

其中的最大數6即為54和24的最大公因數，記為：

$$\gcd(54, 24) = 6$$

— RSA —

RSA 数学原理

有两个很大的质数p和q，共同计算 $n = p * q$

有 $(d * e) \bmod (p-1)(q-1) = 1$. 通常，e private component 是固定65537，如此，只要能够得知p和q，就能够求解d public component。

因此，使用这样的性质，给message做加密和解密。

因为根据n，反过来求解p和q，是非常困难的。



Mathematics of RSA

RSA relies on the mathematical property of primes and factorisation. If you have a very large prime p and another very large prime q , it is trivial to compute $n = p * q$, but it is computationally hard to factorise n into its factors p and q . In RSA, n is called the modulus.

The next step is to calculate the public component d and private component e that satisfy the following:

$$(d * e) \bmod [(p - 1)(q - 1)] = 1$$

Here, $\phi(n) = (p - 1)(q - 1)$ is called the Euler's totient function and represents the number of integers that are "co-prime" with n (i.e., all integers x such that the only common factor between n and x is 1).

There are many combinations of d and e that satisfy the above equation, so usually e is fixed at 65537 (which is a prime) and d is calculated from $(p - 1)(q - 1)$ and e using the extended Euclidean Algorithm. It is important to note that d can be derived from e easily only if you know p and q !

Finally, given plaintext message M , ciphertext C is computed as follows using the public component.

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

In order to get M back from C , we compute C to the power of the private component d :

$$C^d \bmod n = M^{(d * e)} \bmod n = M$$

Without the knowledge of the original prime pair p and q (and hence d), there is no easy way of decrypting C back to M .

加密和解密 python 代码

```
enc = pow(msg,e, n) # supplying the 3rd parameter in pow() efficiently computes the mod
print("Message " + str(msg) + " is encrypted to: " + str(enc))


# decrypt message using private exponent d
plain = pow(enc,int(d), n) # supplying the 3rd parameter efficiently computes the mod
print("Cipher " + str(enc) + " is decrypted to: " + str(plain))

n is: 3084795691
d is: 224795969
checking d*e mod (p-1)*(q-1): 1
Message 12345 is encrypted to: 1335268360
Cipher 1335268360 is decrypted to: 12345
```

```
pow(x, y, z)
```

Parameter Values

Parameter	Description
<i>x</i>	A number, the base
<i>y</i>	A number, the exponent
<i>z</i>	Optional. A number, the <u>modulus</u>



More Examples

Example

Return the value of 4 to the power of 3, modulus 5 (same as $(4 * 4 * 4) \% 5$):

```
x = pow(4, 3, 5)
```

a1-p2-q2 题目例子

解题

```
[12]: enc = 0x50543d1e0fda637c109bb32c706dbaec8d8d20ce001cca02f8576a4852a072c9
      d=0x0D067636BAC6088AD2281E4BFFCACEFEF9BC1A69FB9E701063DFBAAB436E4C1
      n=0x9B51C20306EDE535C8FCAADBC3F3515E52A0D005703DD449BEC66B23E2932313

      enc1=int("50543d1e0fda637c109bb32c706dbaec8d8d20ce001cca02f8576a4852a072c9",16)
      d1=int("0D067636BAC6088AD2281E4BFFCACEFEF9BC1A69FB9E701063DFBAAB436E4C1",16)
      n1=int("9B51C20306EDE535C8FCAADBC3F3515E52A0D005703DD449BEC66B23E2932313",16)
      enc1,d1,n1
```

```
[12]: (36333864857097773831497749468960098445056382106661802261245266130180568543945,
      5891483995546727120328637652438091886928667918827959299068453294388039574721,
      70252945163054194920847511977408156476811805984949774253562479404028773212947)
```

```
[14]: plain = pow(enc1,int(d1), n1)
      plain
```

```
[14]: 9525492127002064617920538183367821855620405883731014525
```

```
[16]: import binascii

      # convert string to integer using
      def string_to_int(string):
          return int.from_bytes(binascii.a2b_qp(string),byteorder='big')

      # convert into back to string
      def int_to_string(number):
          bin = number.to_bytes((number.bit_length() + 7) // 8, byteorder='big')
          return binascii.b2a_qp(bin).decode("utf-8")
      int_to_string(9525492127002064617920538183367821855620405883731014525)
```

```
[16]: 'csf2024s1_{voteptarius}'
```