# 1. 编程实现 RSA 算法

• Python 实现

```
import random
from math import gcd
# 求模逆
def modinv(a, m):
   for x in range(1, m):
        if (a * x) % m == 1:
           return x
   return None
# 生成两个随机素数(简化实验,用小素数)
def generate_keypair(p, q):
   n = p * q
   phi = (p - 1) * (q - 1)
   e = random.randrange(2, phi)
   while gcd(e, phi) != 1:
        e = random.randrange(2, phi)
   d = modinv(e, phi)
   return ((e, n), (d, n))
# 加密
def encrypt(pk, plaintext):
   key, n = pk
   cipher = [pow(ord(char), key, n) for char in plaintext]
   return cipher
#解密
def decrypt(pk, ciphertext):
   key, n = pk
   plain = [chr(pow(char, key, n)) for char in ciphertext]
   return ''.join(plain)
# 示例使用
p = 61
q = 53
public, private = generate_keypair(p, q)
message = "HelloRSA"
encrypted_msg = encrypt(public, message)
decrypted_msg = decrypt(private, encrypted_msg)
```

```
print("原始消息:", message)
print("加密后:", encrypted_msg)
print("解密后:", decrypted_msg)
```

#### • C ++ 实现

```
#include <iostream>
#include <cmath>
#include <vector>
#include <string>
using namespace std;
// 计算最大公约数 GCD
int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a % b);
}
// 求模逆: 计算 (e*d) ≡ 1 mod ф => 求 d
int modInverse(int e, int phi) {
    for (int d = 1; d < phi; d++) {</pre>
        if ((e * d) % phi == 1) return d;
    }
    return -1;
}
// 快速幂(加快加密/解密效率)
long long modPow(long long base, long long exp, long long mod) {
    long long result = 1;
    base %= mod;
    while (exp > 0) {
        if (exp & 1) result = (result * base) % mod;
        base = (base * base) % mod;
        exp >>= 1;
    }
    return result;
}
// RSA 加密
vector<long long> encrypt(string msg, int e, int n) {
    vector<long long> cipher;
   for (char ch : msg) {
        long long m = static_cast<int>(ch);
        cipher.push_back(modPow(m, e, n));
    }
    return cipher;
}
// RSA 解密
```

```
string decrypt(const vector<long long>& cipher, int d, int n) {
    string result;
   for (long long c : cipher) {
        char ch = static_cast<char>(modPow(c, d, n));
       result += ch;
   }
   return result;
}
int main() {
   // 选择两个小素数
   int p = 61;
   int q = 53;
   int n = p * q;
                               // 计算 n
   int phi = (p - 1) * (q - 1); // 欧拉函数
   // 选择 e, 满足 1 < e < phi 且 gcd(e, phi) = 1
   int e = 17;
   int d = modInverse(e, phi); // 计算 d
   cout << "公钥 (e, n): (" << e << ", " << n << ")\n";
    cout << "私钥 (d, n): (" << d << ", " << n << ")\n";
   string message = "HelloRSA";
   cout << "原文: " << message << endl;
   auto encrypted = encrypt(message, e, n);
   cout << "加密后: ";
   for (auto c : encrypted) cout << c << " ";</pre>
   cout << endl;</pre>
    string decrypted = decrypt(encrypted, d, n);
    cout << "解密后: " << decrypted << endl;
   return 0;
}
```

#### Java 实现

```
import java.math.BigInteger;
import java.security.SecureRandom;
import java.util.Scanner;

public class RSAExample {
    private BigInteger p, q, n, phi, e, d;
    private int bitlen = 512;

    public RSAExample() {
        SecureRandom r = new SecureRandom();
    }
}
```

```
p = BigInteger.probablePrime(bitlen, r);
       q = BigInteger.probablePrime(bitlen, r);
       n = p.multiply(q);
       phi =
p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
       // 选择公钥 e (常用值为 65537)
       e = BigInteger.valueOf(65537);
       while (phi.gcd(e).intValue() > 1) {
           e = e.add(BigInteger.TWO);
       }
       // 计算私钥 d (e 的模反元素)
       d = e.modInverse(phi);
   }
   // 加密函数
   public BigInteger encrypt(BigInteger message) {
       return message.modPow(e, n);
   }
   // 解密函数
   public BigInteger decrypt(BigInteger encrypted) {
       return encrypted.modPow(d, n);
   }
    public static void main(String[] args) {
       RSAExample rsa = new RSAExample();
       Scanner scanner = new Scanner(System.in);
       System.out.print("请输入原文(英文): ");
       String plaintext = scanner.nextLine();
       System.out.println("原始文本: " + plaintext);
       byte[] bytes = plaintext.getBytes();
       // 将明文转为 BigInteger (只演示字符串整体加密,不是逐字符)
       BigInteger message = new BigInteger(bytes);
       BigInteger encrypted = rsa.encrypt(message);
       BigInteger decrypted = rsa.decrypt(encrypted);
       String result = new String(decrypted.toByteArray());
       System.out.println("加密后的密文: " + encrypted);
       System.out.println("解密后的明文: " + result);
   }
}
```

### 2. 编写 Base64 加解密函数

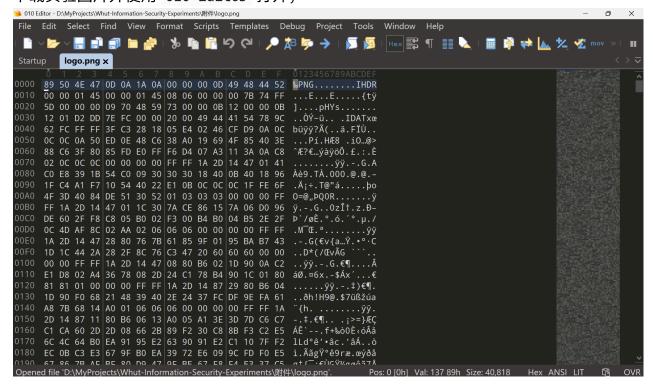
```
import base64
def base64_encode(input_str):
   """将字符串进行 Base64 编码"""
   bytes_data = input_str.encode('utf-8') # 字符串转字节
   encoded_bytes = base64.b64encode(bytes_data) # 编码
   encoded_str = encoded_bytes.decode('utf-8') # 字节转字符串
   return encoded_str
def base64_decode(encoded_str):
   """将 Base64 字符串解码为原始字符串"""
   decoded_bytes = base64.b64decode(encoded_str) # 解码
   decoded_str = decoded_bytes.decode('utf-8') # 字节转字符串
   return decoded_str
# 示例测试
if __name__ == "__main__":
   original = "Hello, Base64 编码测试!"
   encoded = base64_encode(original)
   decoded = base64_decode(encoded)
   print("原始字符串:", original)
   print("编码后字符串:", encoded)
   print("解码后字符串:", decoded)
```

### 运行实例

```
原始字符串: Hello, Base64 编码测试!
编码后字符串: SGVsbG8sIEJhc2U2NCDml6DmsqHmnKzoqLk=
解码后字符串: Hello, Base64 编码测试!
```

## 3. 提取图片中隐写的密文

• 下载实验图片并使用 010 Editor 打开;



• 查看文件头部和尾部是否存在不同,发现图片尾部有一段特殊字符

```
87 48 85 E0 85 D1 A5 33 23 08 30 30 30 00 00 00
9E30  00 FF FF 1A|2D 14 47 38|80 76 B7 91|0B 4A 83 11  .ÿÿ.-.G8€v·'.Jf.
9E40 56 50 82 0A CO 0B 50 0C 2B 04 47 BB C3 23 15 30 VP, .À.P.+.G»Ã#.0
9E50 30 30 00 00 00 FF FF 1A 2D 14 47 01 06 40 2A 00....ÿÿ.-.G..@*
9E60 28 41 58 01 89 1E CA 85 25 A8 F0 7B 00 2D F8 40 (AX.‰.Ê..%"ð{.-ø@
9E70  F4 83 D1 FD|C5 A3 00 03|30 30 30 00|00 00 00 FF  ôfÑýÅ£..000....ÿ
     FF 1A 2D 14 47 01 49 00 DA FD 86 15 9A C8 B4 00 ÿ.-.G.I.Úý†.šÈ
9E90 AD 96 06 11 09 40 4B 61 3E C0 0A 3C 28 1B D4 FA
                                                           --...@Ka>À.<(.Ôú
9EA0 FB 30 DA FD 1D 05 44 03 06 06 06 00 00 00 FF û0Úý..D.....ÿ
9EB0  FF 1A 2D 14 47 01 D5 01 74 89 90 02 92 B9 B0 82 _ÿ.-.G.Õ.t‱..′¹°
9EC0 93 52 00 2B F0 60 60 B4 AB 3B 36 36 36 63 36 31
                                                           "R.+ð``′«;666c61
9ED0 36 37 37 62 35 36 34 37 35 36 36 38 35 39 33 32 677b564756685932
9EE0 36 38 36 63 36 33 36 39 34 32 36 31 36 31 34 37 686c636942616147
9EF0 34 36 37 36 34 61 33 33 34 64 36 37 35 39 33 32 46764a334d675932
9F00 37 38 36 38 36 33 33 33 34 64 36 37 36 31 35 38 786863334d676158
9F10 34 64 36 37 36 33 36 64 35 36 36 38 36 32 34 37 4d67636d56686247
9F20   37  38  33  35 |34  39  34  37 |36  63  37  35 |36  34  34  37 | 783549476c756447
     35 36 37 39 35 61 35 38 34 65 33 30 36 31 35 37
                                                           56795a584e306157
9F40 33 35 36 65 33 37 33 37 39 34 32 33 64 37 64
                                                           356e373779423d7d
9F50 OA A8 OB 18 18 18 00 00 00 00 FF FF 03 00 C8 AE
                                                              ......ÿÿ..È®
9F60 60 0F 55 D9 19 77 00 00 00 00 49 45 4E 44 AE 42
                                                             .UÙ.w....IEND®B
9F70 60 82
```

- 通过观察,文本中存在"d"、"e"等字符,为16进制表示的 ASCII 码
- 利用 python 脚本将其转为字符

```
ascii =
"666c61677b564756685932686c63694261614746764a334d675932786863334d6761584d676
36d56686247783549476c75644756795a584e306157356e373779423d7d"

for i in range(0, len(ascii), 2):
    print(chr(int(ascii[i:i + 2], 16)), end='')
```

得到如下密文: flag{VGVhY2hlciBaaGFvJ3MgY2xhc3MgaXMgcmVhbGx5lGludGVyZXN0aW5n77yB

### =}

- flag{} 中大括号包裹的是前文所学的 Base64 加密的密文。
- 利用 Base64 解密工具对其进行解密。



• 得到最终的明文: Teacher Zhao's class is really interesting!