1. 编程实现 RSA 算法

- Python 实现

```python
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 ϕ => 求 d
int modInverse(int e, int phi) {
    for (int d = 1; d < phi; d++) {
        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 << " ";
    cout << endl;

    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

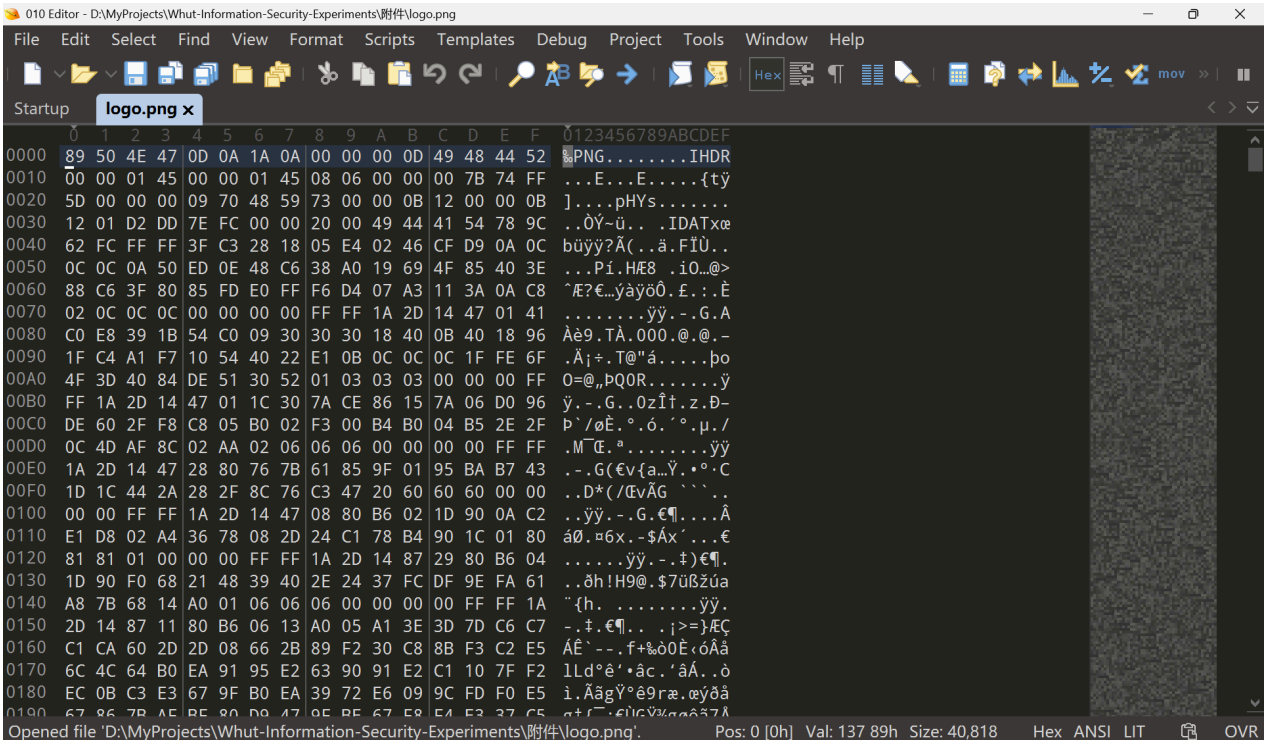
```python
import base64

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

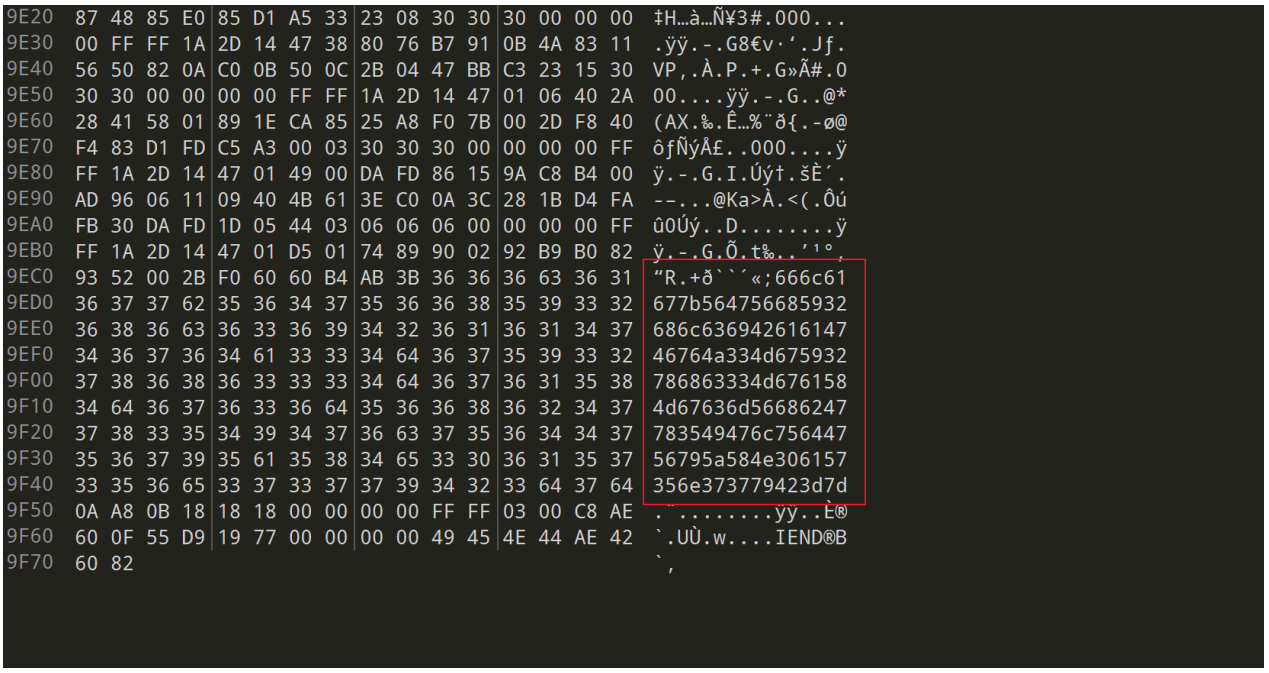
```
原始字符串: Hello, Base64 编码测试!
编码后字符串: SGVsbG8sIEJhc2U2NCDml6DmsqHmnKzoqLk=
解码后字符串: Hello, Base64 编码测试!
```

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



- 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 =
"666c61677b564756685932686c6369426161474676 4a334d675932786863334d6761584d676
36d566862477835494 76c75644756795a584e306157356e373779423d7d"

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

- 得到如下密文：
  flag{VGVhY2hlciBaaGFvJ3MgY2xhc3MgaXMgcmVhbGx5IGludGVyZXN0aW5n77yB

=}

- 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!**