RSA Challenge解题实验报告

姓名：郭子涵 学号:2312145 班级：信息安全、法学双学位班

1 Challenge 1

题目给出了以下信息：

- n1: RSA模数
- e1 = 65537: 公钥指数
- c1: 密文
- hint = (p1 - temp) * d1
- $s = 8 \times temp^3 + 27 \times temp^2 + 2004 \times temp$

我们的目标是恢复明文flag1。

1.1 解题步骤

1.1.1 步骤1：恢复temp值

通过枚举temp求解

```python
s_val = gmpy2.mpz(s_val)
temp_approx = gmpy2.iroot(s_val // 8, 3)[0]
temp = None
for t in range(int(temp_approx) - 1000, int(temp_approx) + 1000):
    t_val = gmpy2.mpz(t)
    f_t = 8*t_val**3 + 27*t_val**2 + 2004*t_val
    if f_t == s_val:
        temp = t_val
        break
if temp is None:
    raise ValueError("失败")
print(f"[+] Found temp: {temp}")
```

得到结果：temp=14006982024182322024418825387284695215340139911

1.1.2 步骤2：分解n1

推导过程：

由于 $e_1 \times d_1 \equiv 1 \pmod{\varphi(n_1)} \Rightarrow e_1 \times d_1 = k\varphi(n_1)+1$

∵ $hint = (p_1 - temp) \times d_1 \Rightarrow hint \times e = p_1 - temp \pmod{\varphi(n_1)}$

∴ $hint \times e + temp = p_1 + k\varphi(n_1)$

由 $\varphi(n_1) = (p_1-1)(q_1-1) = p_1q_1 - p_1 - q_1 + 1 = n_1 - p_1 - q_1 + 1$

∴ $hint \times e_1 + temp = p_1 + k(n_1 - p_1 - q_1 + 1)$

$hint \times e_1 + temp = p_1 + k(n_1 - p_1 - \frac{n_1}{p_1} + 1)$

$p_1(hint \times e_1 + temp) = p_1^2 + k(p_1 n_1 - p_1^2 - n_1 + p_1)$

$p_1(hint \times e_1 + temp) = (1-k)p_1^2 + k n_1 p_1 - k n_1 + k p_1$

$(k-1)p^2 + (hint \times e_1 + temp - k n_1 - k)p_1 + k n_1 = 0$

构建一元二次方程

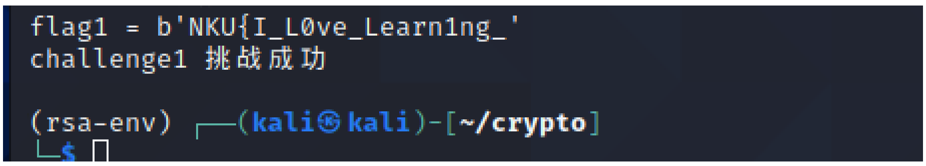
通过这个关系建立方程，尝试找出合理的k值来分解n1:

```python
from Crypto.Util.number import *
import gmpy2
# 给定参数
n1 = ……
hint = ……
c1 =……
e1 = 65537
s_val = ……
#第一步：求解temp
def find_temp():
    """通过近似求解和小范围遍历找到满足方程的temp值"""
    temp_approx = gmpy2.iroot(s_val // 8, 3)[0]  # 近似解
    # 在近似解附近小范围遍历
    for t in range(int(temp_approx) - 1000, int(temp_approx) + 1000):
        t_val = gmpy2.mpz(t)
        # 计算方程左边值
        equation_val = 8*t_val**3 + 27*t_val**2 + 2004*t_val
        if equation_val == s_val:
            return t_val
```

```python
    return None

temp = find_temp()
if temp is None:
    raise ValueError("无法找到满足方程的temp值")

print(f"[+] 找到temp值: {temp}")

#第二步: 分解n1
def factorize_n1():
    """利用已知关系分解n1"""
    # 计算k的估计值
    k_approx = (e1 * hint) // n1
    print(f"[*] k的估计值: {k_approx}")

    # 在估计值附近搜索
    for k in range(k_approx - 100000, k_approx + 100000):
        if k <= 0:
            continue
        # 建立二次方程: x^2 + b*x + c = 0其中x = p1
        a = k-1
        b = hint*e1+temp-k*n1-k
        c = k*n1
        # 计算判别式
        discriminant = b**2 - 4*a*c
        if discriminant < 0:
            continue

        # 检查是否为完全平方数
        sqrt_disc = gmpy2.isqrt(discriminant)
        if sqrt_disc * sqrt_disc != discriminant:
            continue

        # 计算可能的p1值
        for sign in [1, -1]:
            p1_candidate = (-b + sign * sqrt_disc) // 2

            # 检查候选p1是否有效
            if p1_candidate <= 1:
                continue
            if n1 % p1_candidate != 0:
                continue
            q1_candidate = n1 // p1_candidate

            # 简单检查是否为素数
            if gmpy2.is_prime(p1_candidate) and gmpy2.is_prime(q1_candidate):
                print(f"[+] 找到k值: {k}")
```

```python
                return p1_candidate, q1_candidate
    # 如果没找到，尝试另一种方法,从n1的平方根附近开始找因数
    sqrt_n = gmpy2.isqrt(n1)
    for i in range(1, 100000000):
        for direction in [1, -1]:
            p1_candidate = sqrt_n + direction * i
            if p1_candidate <= 1:
                continue
            if n1 % p1_candidate == 0:
                q1_candidate = n1 // p1_candidate
                if gmpy2.is_prime(p1_candidate) and gmpy2.is_prime(q1_candidate):
                    return p1_candidate, q1_candidate
    return None, None

p1, q1 = factorize_n1()
if p1 is None:
    raise ValueError("无法分解n1")

print(f"[+] 找到p1: {p1}")
print(f"[+] 找到q1: {q1}")

# 计算私钥d1
phi1 = (p1 - 1) * (q1 - 1)
d1 = gmpy2.invert(e1, phi1)

# 解密密文
flag1_int = pow(c1, d1, n1)
flag1 = long_to_bytes(flag1_int)

print(f"[+] 解密得到flag: {flag1}")请详细的给出本代码的解题思路
```

最后我们得到的flag是p1 ＝ b'NKU{I_L0ve_Learn1ng_'



2 challenge2

1. 这题中模数n2 是用两数的幂次构成：$n_2 = p_2^5 \times q_2^3$,其中 p2,q2 满足：$p_2^2 + q_2^2 = N$

   给定一个大数N，用 **two_squares** 函数找到a,b 使得$a^2 + b^2 = N$，然后设 $p_2 = a, q_2 = b$

   two_squares 是 SageMath 提供的函数，利用数论方法（如高斯整数理论）分解N为两个平方数之和。

2. **计算欧拉函数**$\varphi(n_2)$,$n_2 = p_2^5 \times q_2^3$,当p,q 互质时，

   $\varphi(n_2) = \varphi(p_2^5) \times \varphi(q_2^3) = p_2^4(p_2 - 1) \times q_2^2(q_2 - 1)$这就是代码中计算欧拉函数的公式。

3. **求私钥：** 私钥指数d 满足$d \times e \equiv 1 \pmod{\varphi(n_2)}$,即d是e 在模$\varphi(n_2)$ 下的乘法逆元，代码中通过**inverse_mod(e, phi)**函数求出。
4. **用私钥d 对密文 c2进行解密：** $m = c_2^d \mod n_2$,**m.digits(256)** 将大整数m拆成256进制的“数字”列表（即每个元素是 0~255 的一个字节），但低位在前（小端序），因此用 [::-1] 翻转顺序，最终获得正确顺序的字节序列。

```
N = 1413143110830814345443500757771600055941920506269861870813395945701197252935449368609310943118429112625519257309092511938909464890191839350386522571064865 8
e = 65537
c = 36973693092976029706514891158491994546679427653385808154970075312189360401669992749686493268520066236617925065280352836335393237668205401310594966311441385258638991920292413280948344149523816142823354787566810494569832312138506519778260047592104155545277214468145293959069867825392508028086999488564631380279216174005129782605797862013882371204185088678659083760136834440952449010128209178209176573492076059788421354711727527118293730794341441114086279741121181284064642496870553384608872297638685395296672347102520892525776868504855632792797615338932673209152948099521121607887918275338569383538790219015769270727 4

a, b = two_squares(N)
p2, q2 = a, b
n2 = p2^5 * q2^3
phi = (p2^4)*(p2 - 1)*(q2^2)*(q2 - 1)
d = inverse_mod(e, phi)
m = power_mod(c, d, n2)

# 将整数 m 转成 bytes
flag_bytes = bytes(m.digits(256)[::-1])
print(flag_bytes)
```

```
flag2: b'Crypt0_So_Much!}'
```

得到flag2: b'Crypt0_So_Much!}'

最终的flag: NKU{I_L0ve_Learn1ng_Crypt0_So_Much!}