華中科技大學

课 程 设 计 报 告

题目： 密码学课程设计

课程名称： 密码学课程设计

专业班级： 网安 2104 班

学　　号： U202112131

姓　　名： 邬雪菲

指导教师： 汤学明

报告日期： 2023 年 9 月 7 日

网络空间安全学院

目 录

一、设计过程

1.1 分组密码快速实现

（1）设计内容

按照题目要求实现教材例 3.1 的 SPN 加密解密程序，对于平台测试输入的若干密钥和明文，能正确进行五轮加密并输出密文，还能将密文最后一位取反再进行解密输出。并对程序进行优化，使得代码运行时间在限制范围内。

（2）设计过程

按照如图 1 的教材上的伪码，设计加密流程并根据 SPN 加解密结构相同的特性构建解密结构，编写主函数，部分代码思路如图 2：

图 1 教材上的 SPN 算法

```
scanf("%d", &n); // 读入数据组数
PCul();          // 计算P盒置换表
for (int i = 0; i < n; i++)
{
    // 读入密钥和明文
    read();
    // SPN加密
    for (int j = 0; j < 3; ++j)
    {
        XOR(w, keys + j, u); // 白化
        S(u, v);             // S盒
        P(v, w);             // P盒
    }
    XOR(w, keys + 3, u);
    S(u, v);
    XOR(v, keys + 4, y);
    // 输出密文
    Output(y);
}
```

图 2 SPN 部分主要代码思路

接下来描述具体步骤的实现：

1. 数据读写：输入输出都以十六进制表示，且 SPN 盒代换过程是四位二进制操作，故将十六进制输入存为整数，将 32 位密钥存为长度为 8 的整型数组并逐个操作。为了优化读写提高效率，使用快读读取字符，putchar 输出结果。

2. 加密解密函数：每轮取出密钥中对应位置的轮密钥。异或函数直接采用异或运算。使用 S 盒数组实现四位代换。考虑到 P 盒置换操作耗时多，在程序开始运行时提前计算 P 盒置换表，之后的 P 盒操作直接查询数组。

（3）小结

该题核心部分不难理解和实现，难点是如何优化加速，这也是本人花费最多时间的部分。一开始没考虑运行时间的问题，最后一个数据集直逼 2000ms。搜索后尝试以下操作：①函数、循环展开对该题效果不明显；②优化读写效果显著，时间减半到 1000ms；③P 盒打表又加速到 700ms；④inline 写的快读改为 define 很有效；⑤开 O3 优化效果明显。最后一版仅用时三百多毫秒。但为保证速度造成了一定的代码冗余。

1.2 线性密码分析

（1）设计内容

对于上题实现的 SPN 加密，通过分析 S 盒的非线性特性，发现明密文比特和密钥比特之间可能存在的线性关系，由此实现线性攻击破解出密钥。对每组密钥，需要选取合适的线性链统计分析 8000 组明密文对，得出部分密钥，再通过穷举 s 剩余密钥完成解密。

（2）设计过程

线性攻击是一种已知明文分析方法。本题通过对大量明密文对的分析，实现 S 盒的线性逼近，分析出部分代码以缩小密钥穷举范围。具体实现步骤如下：

1. 读入数据：沿用上的方式快读，为每组密钥读入 8000 对明密文对。

2. 线性分析链的选择：一号链从教材中得到，偏差为 1/32，可分析出最后一轮轮密钥的 5-8 位、13-16 位。第二条链一开始是手动选择的，但是实际分析的效果不好。搜索得到思路：枚举所有输入输出线性掩码的组合，并用测试集数据统计偏差量，可统计出偏差大的关联 3 或 4 个子密钥的链。最后参考了同学的思路选出二号链，可分析出最后一轮密钥的 1-4 位、9-12 位。

一号链：x5 ^ x7 ^ x8 ^ u6 ^ u8 ^ u14 ^ u16。

二号链：x5 ^ x6 ^ x7 ^ x8 ^ u2 ^ u4 ^ u6 ^ u8 ^ u10 ^ u12。

3. 偏差量的统计：穷举候选子密钥，统计各自偏差量。

4. 确定子密钥：根据偏差统计量的大小可确定子密钥，但要注意偏差量最大的不一定是正确的密钥，这里取第一条链偏差量前 16 大的子密钥继续分析。又因第二条链是在第一条链的基础上分析的，其偏差量更能反映候选密钥正确性，故认为偏差量最大的是正确密钥。

5. 剩余密钥的穷举和验证：枚举空间已减少到 65536 个。由于错误密钥对不同明文加密均得到正确密文的可能极小，故认为验证成功两对即为正确密钥，减少计算量。

```
PCul();                         // 提前计算P盒表
scanf("%d", &n);                // 读入数据组数
for (int i = 0; i < n; i++) // 主循环分析每一组密钥
{
    read(); // 读入8000组明密文对
    // 穷举256个候选子密钥，统计偏差
    Culchain1();
    // 从偏差最大的子密钥开始计算,只计算前16个
    for (int j = 0; j < 16; ++j)
    {
        // 统计第二条链
        Culchain2();
        // 认为偏差最大的子密钥为最终密钥，穷举前16位密钥
        for (int key = 0; key < 65536; ++key)
        {
            // 验证密钥，成功两对则认为是正确的密钥
            for (times = 0; times < 2; ++times)
                if (!SPN(keys))
                    break;
            if (times == 2)
                Output(); // 输出结果
        }
        if (times == 2)
            break;
    }
}
```

图 3 线性分析代码思路展示

（3）小结

本题难点在于线性链的选择，还有一点优化问题。枚举剩下密钥过程中单层循环比四层循环快一些，说明在循环较大时展很有效。还尝试在读入明密文的同时统计第一条链，速度变慢，猜想是访存顺序改变导致命中率降低。还有针对测试数据减少了对候选子密钥的验证，可能通不过随机数检测。

1.3 差分密码分析

（1）设计内容

差分密码分析与线性密码分析类似，是一种对明文差异扩散的研究，通过研究 S 盒对输入异或产生的输出异或的偏向性，由此分析出部分子密钥。仅选择输入异或为特定值的明文对分析其最后一轮的差分，是一种选择明文攻击。

（2）设计过程

本题通过 S 盒差分扩散的分析完成对密钥的破译。具体过程如下：

1. 读入数据：快读读入入所有密文

2. 差分链的选择：差分分析可以两条链分别单独进行，再直接将两组子密钥进行组合。选取的一号链来自教材，可分析最后一轮轮密钥的 5-8、13-16 位，二号链可分析最后一轮密钥的 1-4、9-12 位。

一号链：输入异或 0x0B00，输出异或 0x0606

二号链：输入异或 0x0050，输出异或 0x5050

3. 差分扩散的统计：穷举候选子密钥，对符合输入异或的明密文，若为正确对则计数。

4. 子密钥的确定和组合：由于链 1 和链 2 的计算是分别单独进行的，且扩散率最高的不一定是正确密钥，故对两条链扩散率较大的候选密钥进行组合验证。

5. 剩余密钥的穷举和验证：和线性密码分析类似，穷举空间已经减少到 65536 个。但沿用线性分析的验证方式不可取。阅读了该题讨论区中鹿修齐同学的提示后注意到，本题是按顺序给出 0x0000 到 0xffff 的密文，这意味着相邻明文的差别相当小，可能只差 1 位，这导致在验证密钥匙时，由于该 SPN 有着比较强的线性特征，错误密钥得到正确密文的概率大大上升，所以应提高密钥验证通过的门槛，并选择较为离散的明文进行检验。

```
PCul(); // P盒打表
scanf("%d\n", &n);
for (int i = 0; i < n; i++)
{
    read();      // 读密文
    Culchain1(); // 分析第一条链
    Culchain2(); // 分析第二条链
    // 找两条链统计量最高的64*64种候选子密钥组合
    // 逐一穷举剩下的密钥并进行验证
    for (int j = 255; j >= 192; --j)
    {
        for (int j = 255; j >= 192; --j)
        {
            // 穷举前16位密钥
            for (keys[0] = 0; keys[0] < 65536; ++keys[0])
            {
                // 验证成功三次就认为正确
                for (times = 0; times < 3; times++)
                    if (!SPN(key))
                        break;
                if (times == 3)
                    goto Output;// 跳出循环输出密钥
            }
        }
    }
}
```

图 4 差分分析代码思路展示

（3）小结

理解了线性分析之后再做差分分析，过程顺利不少，差分链的选择也比线性链要简单。差分攻击采取了过滤噪声等操作，且可以独立分析差分链，效率较高。不过在验证密钥的时候需要注意一下相邻明文差异小的问题。根据测试减少取样数和候选密钥数，最后一个数据集仅用时不到 500ms。

1.4 算法增强

（1）设计内容

对第一题中的 SPN 加解密进行优化，以适应本题的二进制流输入。取输入的前 128 位采用合适方法生成轮密钥，修改密钥长度、分组长度、S 盒、P 盒和加密轮数，并选择采用合适的工作模式，使最后的输出能通过随机数检测。

（2）设计过程

相较于之前的简单 SPN，增加分组长度至 64 位，并采用 CFB 模式输入。具体步骤如下：

1. 读入数据：使用了题目提示的读写方式，输入是二进制流的形式，先读取 128 位密钥，再循环读入分组长度的字符流，存为 64 位无符号长整型待处理。

2. 密钥编排：轮密钥为 64 位，设计密钥编排算法从 128 位密钥中从左往右每隔 16 比特移动一次，抽出 64 位作为轮密钥。

3. 白化：待加密文本直接和轮密钥异或。

4. S 盒代换：S 盒不变，代换操作扩展到 64 位。

5. P 盒置换：从网上借鉴了一个扩展的 64 位 P 盒。由于内存的限制，不支持 P 盒打表故每次加密都直接逐位进行 64 位置换操作。

6. 分组密码模式：CFB 密文反馈模式。将前一个分组的密文加密后与当前分组明文异或得到当前分组密文。对于第一个分组，设置初始向量作为前一个密文。

```
// 主函数代码
// 读入密钥 初始化向量
unsigned long long keys[5] = {0}, plaintext = 0, ciphertext = 202112131;
fread(keys, SPNBYTES, 1, stdin);
fread(keys + 4, SPNBYTES, 1, stdin);
// 轮密钥生成
keys[1] = ((keys[0] << 16) ^ (keys[4] >> 48));
keys[2] = ((keys[0] << 32) ^ (keys[4] >> 32));
keys[3] = ((keys[0] << 48) ^ (keys[4] >> 16));
// CFB模式
for (int i = 0; i < (INPUTBYTES / SPNBYTES); i++)
{
    fread(&plaintext, SPNBYTES, 1, stdin);
    ciphertext = SPN(keys, ciphertext);
    ciphertext ^= plaintext;
    fwrite(&ciphertext, SPNBYTES, 1, stdout);
}
```

图 5 SPN 增强代码思路展示

（3）小结

这道题是一个加解密结构设计的题目，沿用之前的SPN结构，扩展分组长度、更换P盒、使用CFB模式即可完成设计，结构清晰简单，加密轮数无需增加却能满足随机数检测标准。但有个疑惑的点是P盒的选择问题：一开始选择了不合适的P盒，需要设置特殊的初始向量才能通过随机数检测。以为是加密结构太简单的问题，增加了加密轮数还是有部分数据集未通过检测，而且耗时增加了不少，如果设计更复杂的加密结构想必会超时，故猜想这并非是加密结构简单的问题，而是有别的原因。更换P盒之后，问题解决，随机取若干初始向量值都能达到随机性要求，查找了很久也没找到对这个现象的合理解释。最后测试数据集平均耗时700ms左右。

1.5 RSA 参数计算

（1）设计内容

完成非对称加密 RSA 的参数合法性检查和计算。RSA 是一种非对称加密体制，n 是两个不同的奇素数 p 和 q 的乘积，ϕ(n) = (p-1)(q-1)。取随机数 e 使得 gcd(e, ϕ(n)) = 1，d =e ^ (-1) mod ϕ(n)。其中 e 用于加密，(n,e)作为公钥 ,d 用于解密,(p, q, d)作为私钥。

（2）设计过程

首先是环境的配置，因为 RSA 加解密涉及大整数，故可使用 OPENSSL 库或 GMP 库，我使用的是 OPENSSL 库。然后再根据思路编写伪码，检查 RSA 各参数是否合理，合理则求出私钥 d，最后调用 OPENSSL 库实现各种计算。具体步骤如下：

1. OPENSSL 环境配置：按照老师的推荐在 win11 下尝试配置 TASSL 失败，遂配置 OEPNSSL，并将其缺少的国密算法文件复制对应位置，配合 Visual Studio 可以实现 OPENSSL 库函数的调用并能本地调试代码。

2. 数据的读入：调用 OPENSSL 库函数将读入的字符串转换为大整数结构。

3. 参数检查：在本题讨论区参考了平仔骏同学的提示，对 e 的大小、n 的大小 pq 的素性、pq 的差值、p-1 和 q-1 的公因数大小、ϕ(n)和 e 是否互质进行了合理性判断。其中，在判断 ϕ(n)和 e 是否互质的过程中，是通过调用扩展欧拉定理计算两者公因数函数实现的，同时也可求出两者对应的模逆，这样避免了之后对参数 d 的重复计算。最后实现如图 6：

4. 扩展欧拉定理的实现：已知 m,n 实现对 xm+yn=gcd(m,n)式中其它数的求解。其实就是辗转相除法的升级版。递归版函数很容易理解，但是大整数递归操作函数开销大，故参考了网上的迭代版算法，转为 OPENSSL 库实现，缺点是迭代算法不易阅读理解，花了一些时间才弄清楚它的实现过程，但效率更高。最终实现如图 7：

```
// 判断RSA参数合法性，合法则算出结果d
bool ifRSA(BIGNUM *e, BIGNUM *p, BIGNUM *q, BIGNUM *phi, BIGNUM *result)
{
    if (BN_num_bits(e) < 4)
        // printf("e太小\n");
        return true;
    if (BN_num_bits(n) < 1000)
        // printf("n太小\n");
        return true;
    if ((BN_is_prime(p, BN_prime_checks, NULL, NULL, NULL) != 1) ||
(BN_is_prime(q, BN_prime_checks, NULL, NULL, NULL) != 1))
        // printf("pq不全为质数\n");
        return true;
    if (BN_cmp(pqdiff, pchuyi10) == -1)
        // printf("pq差值太小\n");
        // p - q > (1/10) * p
        return true;
    gcd = exgcd(x, y, pminus1, qminus1);
    if (BN_cmp(gcd, num16) == 1)
        // printf("公因数太大gcd(p - 1, q - 1) > 16\n");
        return true;
    gcd = exgcd(x, y, e, phi); // 扩展欧拉定理计算gcd，若互质可得模逆
    if (BN_cmp(gcd, BN_value_one()) != 0)
        // printf("e phi不互质\n");
        return true;
    BN_copy(result, x); // 判断e phi互质的时候已经求出了逆元d存入x
    return false;
}
```

图 6 RSA 参数检查代码思路

```
// 扩展欧拉定理实现已知m、n，求xm+yn=gcd(m,n)中各数的操作
// 这样在判断互质性的同时就能计算出模逆，避免了重复运算
BIGNUM *exgcd(BIGNUM *x, BIGNUM *y, BIGNUM *m, BIGNUM *n)
{
    if (n == 0)
    {
        x = 1, y = 0;
        return m;
    }
    a1 = 1, a2 = 0;
    b1 = 0, b2 = 1;
    c = m, d = n;
    q = c / d, r = c % d;
    while (r != 0)
    {
        c = d, d = r;
        t = a1, a1 = a2;
        v = q * a2;
        a2 = t - v;
        t = b1, b1 = b2;
        v = q * b2;
        b2 = t - v;
        q = c / d, r = c % d;
    }
    x = a2;
    b = b2;
    return d;
}
```

图 7 扩展欧拉定理迭代算法

（3）小结

RSA 核心思想的理解和学习不难，时间主要花在环境配置上。Windows 配置 TASSL 失败，可参考资料少，尝试了很多办法都没有用，才转向配置 OPENSSL。VSCode 配置也失败了，换成 VS 才成功实现本地调试。从一开始的不经思考就按教程操作，到失败后一步步查找问题所在并试图理解每一步操作的意义，最后成功完成配置，配环境的过程着实让人获益匪浅。

1.6 模重复平方计算

（1）设计内容

实现模重复平方算法快速计算模幂。并考虑实现其它的快速模幂算法，比较各种方法的性能。

（2）设计过程

按照教材上的算法，以二进制的方式逐位读取指数，每一轮循环中判断二进制位是否不为 0，是则作模平方运算，不是则继续循环，时间复杂度 *O*(logn)，可以通过所有测试集。但在做下一题的时候发现，如果使用普通的快速幂算法是无法通过的，对时间限制较高。

考虑到对于大整数来说，加减运算都是位数线性的，而乘法运算是位数平方级别的。所以首要目的是减少乘法的使用。首先用引入逆即-1 次方的方法优化，把一连串的 1 编程一个更高位上的 1 和最低位上的一个-1。通过这种方法，整体 1 的个数减少。但是实际运行中的优化效果有限。后来又看到了蒙哥马利算法。蒙哥马利算法使用约简，通过变换将需要取模的数控制到很小的范围，这样只需要最多一次减法即可完成取模运算，从而加快整体算法速度。 故本题尝试使用蒙哥马利模乘替换原来的乘法，最终实现的函数和 OPENSSL 自带的模幂函数速度持平。实现代码如图 8：

```
// 蒙哥马利乘替换普通模乘，时间基本减半
void mgml_expmod(BIGNUM *result, BIGNUM *m, BIGNUM *e, BIGNUM *n)
{
    BN_MONT_CTX *mont_ctx = BN_MONT_CTX_new();
    BN_CTX *ctx = BN_CTX_new();
    BN_MONT_CTX_set(mont_ctx, n, ctx);
    BIGNUM *zero = BN_new();
    BN_zero(zero);
    BN_one(result);
    BN_to_montgomery(result, result, mont_ctx, ctx);
    BN_to_montgomery(m, m, mont_ctx, ctx);
    while (BN_cmp(e, zero) != 0)
    {
        if (BN_is_odd(e))
        {
            BN_mod_mul_montgomery(result, result, m, mont_ctx, ctx);
        }
        BN_mod_mul_montgomery(m, m, m, mont_ctx, ctx);
        BN_rshift1(e, e);
    }
    BN_from_montgomery(result, result, mont_ctx, ctx);
}
```

图 8 蒙哥马利快速幂算法代码

（3）小结

本题较为简单，基本是教材算法的简单复现，不过需要考虑常数优化问题，而这恰恰是容易被忽略的问题，以便在下一关中国剩余定理的测试中达到时间要求。采用蒙哥马利乘替换模重复平方中的模乘，实现的快速模幂和原生模幂速度已经很相近，数据集最长耗时 400ms 出头。之后了解到还有更快的滑动窗口算法，但算法的实现较为复杂，需要额外进行窗口预处理和滑动过程，使用 OPENSSL 库实现稍微麻烦一点。考虑到已实现函数速度达到要求，就没有做进一步探究。

1.7 中国剩余定理

（1）设计内容

在前两题的基础上，沿用设计的求参数 d 函数和模乘函数，实现题目要求，运用中国剩余定理求解 c^d(modpq)。

（2）设计过程

中国剩余定理的内容如图 9 所示：

定理 2.15(中国剩余定理) 设m_1，m_2，…，m_s为两两互素的正整数，b_1，b_2，…，b_s为任意整数，那么同余式组

$$\begin{cases} x \equiv b_1 (mod\ m_1) \\ x \equiv b_2 (mod\ m_2) \\ \cdots \\ x \equiv b_s (mod\ m_s) \end{cases}$$

模$M = m_1 m_2 \cdots m_s$有唯一解$x \equiv \sum_{i=1}^{s} b_i \cdot \frac{M}{m_i} (\frac{M}{m_i})^{-1} (mod\ m_i)\ (mod\ M)$。

图 9 中国剩余定理

故本题的计算可分为这几个部分：①计算参数 d 的值；②计算 c ^ d mod p 的值；③ 计算 q ^(-1) mod p；④计算 c ^ d mod p *q *q ^(-1) mod p 的值；⑤计算 c^d mod q 的值；⑥计算 p ^ (-1) mod q；⑦计算 c^d *p * p^(-1) mod q；⑧将 4、7 步中的结果相加 mod pq 即得到结果。

代码思路展示如图 10：

```
// 中国剩余定理
void CRT(BIGNUM *result, BIGNUM *c, BIGNUM *p, BIGNUM *q, BIGNUM *d)
{
    BN_mul(n, p, q, ctx);       // n = p*q;
    BN_sub(pmi, p, BN_value_one()); // pmi = d % p-1
    BN_mod(pmi, d, pmi, ctx);
    BN_sub(qmi, q, BN_value_one()); // qmi = d % q-1
    BN_mod(qmi, d, qmi, ctx);
    mgmlexpmod(c1, c, pmi, p); // c1 = c ^ pmi % p
    BN_mod_inverse(qinverse, q, p, ctx);
    mgmlexpmod(c2, c, qmi, q); // c2 = c ^ qmi % q
    BN_mod_inverse(pinverse, p, q, ctx);
    BN_mul(c1, c1, q, ctx); // c1 = c1 * q * q ^ -1
    BN_mod(c1, c1, n, ctx);
    BN_mul(c1, c1, qinverse, ctx);
    BN_mod(c1, c1, n, ctx);
    BN_mul(c2, c2, p, ctx); // c2 = c2 * p * p ^ -1
    BN_mod(c2, c2, n, ctx);
    BN_mul(c2, c2, pinverse, ctx);
    BN_mod(c2, c2, n, ctx);
    BN_add(result, c1, c2); // result = (c1+c2) % pq
    BN_mod(result, result, n, ctx);
}
```

图 10 中国剩余定理代码思路展示

（3）小结

本题是前两题的组合。乍一看题目好像不用中国剩余定理也能直接用模幂算，尝试了一下果然通过不了，还是得用 CRT 组合加速求解，使得运行时间几乎减半。总体做得还算顺利，最后一个数据集耗时四百多毫秒。

1.8 HASH 函数与彩虹表

（1）设计内容

彩虹表的构造是一个提前计算和存储若干哈希链头和链尾生成查找表的过程，折衷了暴力法和查找法。根据已有的彩虹表，从哈希值 h 开始，不断施加 R 和 H 生成哈希链，如果该链的任何节点与查找表的某个终点发生碰撞，就可用对应的起点重建可能包含 h 的哈希链。

（2）设计过程

首先为了验证理解题意是否正确，编写了一个函数测试每条链是否能从链头走到链尾，结果确实如此。接着根据设计内容完成各部分的编写，具体过程如下：

1. 读入数据：读入 m 条链头和链尾并存储，读入待破解的散列值。

2. 链尾碰撞：遍历起始 R 函数，每条链最多计算 10000 次判断是否碰撞成功，若成功则进下一步的碰撞验证。

3. 碰撞验证：从碰撞成功的链头开始，重建哈希链，看哈希值是否存在于链中，存在则说明是真碰撞，并得到破译的口令值。

```
read(); // 数据读入和预处理
for (int i = 0; i < 100; ++i) // 查找链，每次换一个R函数开始，一共100个
{
    sha1temp = sha1;
    for (int j = 0; j < 9999; j++) // 一条链上要测试9999次
    {
        k = (i + j) % 100 + 1; // 计算所用R函数序号
        R(sha1temp, str, k);
        // 比对当前文本和所有链尾，看是否碰撞
        if (crush(k))
        {
            IfFind = FindStr(str, sha1); // 验证是否是真碰撞
            if (IfFind)
                break;
        }
        UnitSHA1(str, sha1temp); // 哈希，继续前进
    }
    if (IfFind)
        break;
}
if (IfFind)
    printf("%s\n", str);
else
    printf("None\n");
return 0;
```

图 11 彩虹表代码思路展示

（3）小结

该实验的难点在于理解彩虹表的原理并实现破译操作，主要是题意的理解和注意细节。本题对时间的限制较为宽松，因此不需要很多优化操作就能通过。

1.9 数字信封

（1）设计内容

实现 PDCS#7 数字信封解密和验证。对于题目给出的可信根 CA 和用户 B 私钥，用户 B 是 PKCS#7 格式加密消息的接收者，输入是发送者 A 发送的 PKCS#7 格式的信息。需要完成 PKCS#7 包装解开，验证 A 身份合法性、信息完整性，解密提取 A 发送消息的任务

（2）设计过程

参考资料中的代码，PKCS#7 解密分为以下几个部分:

1. 初始化：调用 ERR_load_crypto_strings 和 OpenSSL_add_all_algorithms 函数初始化
2. 预处理：将私钥和证书转为对应的结构体
3. 密文处理：使用 BIO_new_fd 从 stdin 中读入密文，使用 B 的私钥初始化，使用 PKCS7 解密密文并存储结果。
4. 验证签名：从 pkcs7 中获取签名者信息，依次遍历签名者信息，进行验证，若与证书匹配则输出解密密文，否则输出 ERROR。

```
/void gen_pkcs7()
{
    bool flag = true;
    BIO *input = BIO_new_fd(fileno(stdin), BIO_NOCLOSE); // 读数据
    PKCS7 *p7 = PEM_read_bio_PKCS7(input, NULL, NULL, NULL); // 读密文
    EVP_PKEY *pkey = getpkey(pkeyB);                          // 获得私钥B
    BIO *p7out = PKCS7_dataDecode(p7, pkey, NULL, NULL);      // 解密密文
    if (!p7out)
        flag = false;
    int len = BIO_read(p7out, message, 10000); // 读取原始数据，返回数据长度
    if (len <= 0 || len > 200)                 // 判断长度是否合法
        flag = false;
    message[len] = 0;
    STACK_OF(PKCS7_SIGNER_INFO) *sk = PKCS7_get_signer_info(p7); // 获取签名者信息
    int signnum = sk_PKCS7_SIGNER_INFO_num(sk); // 获取签名者数量
    X509_STORE *ca = X509_STORE_new();          // 创建证书ca
    X509_STORE_CTX *ctx = X509_STORE_CTX_new(); // 创建证书存储区上下文环境函数
    X509_STORE_add_cert(ca, getX509(cacert));   // 添加证书
    for (int i = 0; i < signnum; ++i){
        PKCS7_SIGNER_INFO *signInfo = sk_PKCS7_SIGNER_INFO_value(sk, i); // 获得签名者信息
        if (!PKCS7_dataVerify(ca, ctx, p7out, p7, signInfo))             // 验证签名
            flag = false;
    }
    for (int i = 0; i < len; ++i) // 判断是否可打印
        if (isprint(message[i]) == 0)
            flag = false;
    if (flag)
        printf("%s\n", message);
    else
      printf("ERROR\n");
}
```

图 12 PKCS#7 解密验证函数代码

（3）小结

这个题目是最让人困惑的实验，难点就是对题目和示例代码的理解。一开始完全看不懂题目在干什么，示例函数的理解就花了好长时间。而且由于不清楚调用函数的具体实现和用法，最后的程序编写也是一知半解，OPENSSL 官方文档提供的信息少之又少，查阅库函数功能拼凑出代码的过程是痛苦的，借助了同学的帮助才完成了程序的理解和编写，结果还是部分正确，又艰难地进行了调试和修改才通过。

二、实验心得

这次密码学课设是一次全新的体验，与其说是课程设计，不如说是分级通关实验，是对上学期学习的密码学理论课的巩固和拓展。每个关卡都让我对相关的知识点有了更清晰的认知，强化了对密码学的理解，深刻感受到密码学原理与实践的实践部分。

一开始以为只是简单的算法实现，但我在第一关就花费了许多时间。不过也学习到了很多，在提高代码的运行速度方面，学会了快读、循环展开等优化方式，对密码学也有了更深层的理解。对于密码体制，除了加解密结构的设计和安全性等等，要考虑的问题还有很多，效率就是一个。不断修改调试优化代码的过程让我明白程序设计是一门实践的学问，密码学是应用性很强的学科，不单单只有抽象的理论学习。

每尝试解决一个问题我都有很大的收获。初期的 SPN 实现和分析，让我对代换置换网络的核心思想有着更深的体会。而之后 RSA 相关的题目，也是对信数和密码学的回顾与拓展。配置 OPENSSL 环境的时候，虽然遇到很大困难，但不断寻找解决方案甚至自己独立分析问题所在的过程无疑很好地锻炼了我的能力，成功实现本地调试时那股巨大的成就感让我觉得之前花费再多精力都是值得的。还有对彩虹表乃至数字信封的理解与实现，让我深刻明白一个道理，面对毫无头绪的问题不要抱有过多畏难情绪，网络时代信息发达，我们总能搜索到有帮助的内容，浏览到许多优秀前辈留下的博客。即使没有，和同学互相讨论交流的过程也会有很大帮助！总之办法总比困难多，如果一直选择逃避，那么问题会一直横亘在身前阻碍前路，不如大胆尝试，这也是提升自己的好机会。

总之，这次课设让我深刻体会到实践需与理论结合、知识需要不断复习拓展的道理，我的各方面能力又一次得到了提升。在不断探索未知的过程中，不能只拘泥于书上的理论，实践才是一个华中大学生该有的作为，应在实践中不断摸索，践行“明德厚学、求是创新”的校训，努力成为一名优秀的网安学子。

三、对课程设计内容和过程的建议

首先是课程设计的时间安排问题，个人认为将本课设改为与密码学理论同期进行的实验课比较好，这次课设的形式也更像是实验。每个实验都有助于我们更好地理解对应密码体制的加解密原理和过程，如果能和理论课一起进行，会让同学们对密码学抽象理论有着更实践性的理解和体会。

此外，部分题目给的信息过少，缺少足够的描述信息，特别是数字信封。希望能对其实现过程有着更清晰的描述，在示例代码部分添加足够的说明和注释以便理解使用。题目简洁虽然在一定程度上锻炼了学生的信息提取和检索能力，但如果全靠自己入门还是比较考验人的心态的。还有代码优化这一块，虽然是作为信息类专业学生必须了解的技能，但大部分同学之前都没有接触过，只能通过网络搜索得出一些零零碎碎的解决方案，老师们如果能多指点一二会更好。

还有就是环境配置问题，为了实现本地调试，配置 OPENSSL 花了我很长时间。网络上主流操作都是在 linux 平台上配置，但跑虚拟机有点麻烦，而且在 win 上的配置其实是可行的，只不过在缺少参考资料的条件下要走些弯路。因此我建议征集各位同学配环境遇到的问题和解决方案，整理出一个配置文档，方便之后做课设的同学参考，将精力更多地花在代码的设计和编写上。

最后，在完成这个课设的过程中的收获还是非常之大的，感谢老师们提供了这样一个优质的实践课，也希望它能在不断的完善变得更好。

四、参考文献

[1]密码学原理与实践（第三版）. DouglasR.Stinson 著，冯登国译，电子工业出版社，2009
[2]应用密码学：协议算法与 C 源程序（第二版）. Bruce Schneier 著，吴世忠等译，机械工业出版社，2014
[3]visual studio 2019 内调用 OpenSSL3.0_openssl visual studio_禾烟雨的博客-CSDN 博客
[4]中国剩余定理加速实现 RSA 解密_expmod_岁余十二.的博客-CSDN 博客
[5]彩虹表 rainbow-tables - 知乎 (zhihu.com)
[6]数字信封原理_pengshengli 的博客-CSDN 博客