目录

一、设计过程

1.1 分组密码快速实现

（1）设计内容

SPN 是一种分组密码的结构，其加密过程可以概括为以下步骤：

1.初始置换（IP）：明文数据首先经过一个初始置换，通过重新排列明文的位，增加密码的随机性。

2.多轮迭代：SPN 加密结构通常包括多轮迭代，每一轮都会对明文进行一系列的操作。通常，每一轮包括以下步骤：a. 轮密钥添加：将轮密钥与明文按位进行异或操作，以引入密钥的影响。b. 替代：通过 S 盒将一组输入位替换为另一组输出位，引入非线性变换。c. 置换：通过 P 盒重新排列位，增加密码的扩散性和混淆性。

3.最终置换（FP）：在最后一轮迭代后，将密文进行最终的置换操作，类似于初始置换。

4.输出密文：得到的最终置换后的数据即为密文，可用于传输或存储。

解密过程类似，先与轮密钥异或，再按照 S 盒代换，与轮密钥异或，P 盒置换的顺序循环进行，最后一轮不需要 P 盒置换操作。

（2）设计过程

SPN 的具体实现步骤如下：1.数据的读写 2.加密过程 3.解密过程 4.输出结果

1.数据的读写：为提高代码的效率，我利用 getchar()自己封装了一个快速读取函数，如下图 1。

```
void quick_read(int *x){
    int c = getchar();
    int j = 0,tmp;
    while(true)
    {
        if(c == ' ' || c == '\n') break;
        if(c <= '9') tmp = c - '0';
        else tmp = c - 87;
        x[j++] = tmp;
        c = getchar();
    }
}
```

图 1

2.加密过程：对每一组数据，首先进行 5 轮的加密操作，每轮加密包括 K 轮密钥添加、S 盒替代、P 盒置换等步骤，每轮加密后输出加密结果。

3.解密过程：对每一组数据，进行 5 轮的解密操作，与加密相反，每轮解密包括 P 盒置换、S 盒替代、K 轮密钥添加等步骤，每轮解密后输出解密结果。

4.输出结果：将加密和解密的结果输出。

（3）小结

第 7 个测试案例时间为 993ms，非常惊险的通过了。我觉得此代码有两个不足：一是代码中没有足够的错误处理机制，在实际应用中，应该处理各种错误情况，以提高系统的稳定性和安全性；二是 S 盒和 P 盒其实可以合并起来用一个盒子，由此来提高效率。

1.2 线性密码分析

（1）设计内容

对于上文所示的 SPN 进行线性攻击。每一组需要破译的密钥都给出了 8000 组明密文对，破译出一组密钥后，即可容易的分析出所有 32bit 的密钥。

（2）设计过程

首先对 S 盒进行理性分析，找到两条线型逼近链（若仅为一条线型逼近链，计算量过大），然后通过分析大量明密文对，找到两个相应的子密钥，最后经过穷举前 16 位密钥，得到最终的正确密钥。两条链如下：x5 ^ x7 ^ x8 ^ u6 ^ u8 ^ u14 ^ u16；x5 ^ x6 ^ x7 ^ x8 ^ u2 ^ u4 ^ u6 ^ u8 ^ u10 ^ u12。第一条链是通过书上给出的，链偏差为 1/32；第二条链则是自己手动尝试得到的。主要设计过程如下：

1.开始处理每个数据集的循环：初始化 count1 数组，用于统计密钥猜测的次数。循环读取一组明文和密文数据，保存到 p 和 Cipher 数组中。

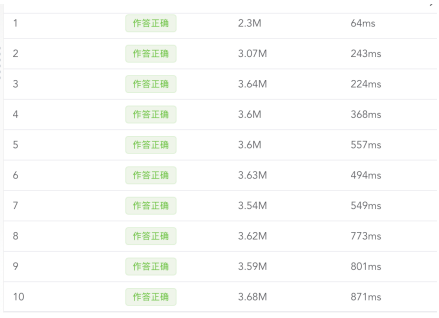
2.密钥猜测阶段：针对每组明文和密文数据，使用双重循环进行密钥猜测。内层循环遍历可能的密钥值，计算中间结果 u[1] 和 u[3]，然后检查是否满足某个条件（在 Encode 函数中判断）。如果满足条件，将密钥猜测结果保存到 count1 数组中，统计次数。

3.密钥猜测结果排序：对 count1 数组中的密钥猜测结果进行排序，将结果保存到 count1 数组中。寻找最大的密钥猜测结果：从排序后的 count1 数组中找到最大的密钥猜测结果，并保存在 max_key[0] 和 max_key[1] 中。

4.进一步密钥猜测：针对最大的密钥猜测结果，继续密钥猜测，以找到完整的密钥。使用三重循环来尝试密钥的不同组合，计算中间结果 x。使用 Encode 函数检查是否满足条件，如果满足，表示找到了密钥。

5.输出找到的密钥：如果成功找到密钥，将其以十六进制格式输出到标准输出。这个程序的主要目的是通过密码分析攻击技术找到一组明文和密文对应的密钥，然后输出这个密钥。

（3）小结



1	作答正确	2.3M	64ms
2	作答正确	3.07M	243ms
3	作答正确	3.64M	224ms
4	作答正确	3.6M	368ms
5	作答正确	3.6M	557ms
6	作答正确	3.63M	494ms
7	作答正确	3.54M	549ms
8	作答正确	3.62M	773ms
9	作答正确	3.59M	801ms
10	作答正确	3.68M	871ms

图 2

运行结果如图 2 所示，成功通过所有测试案例。此实验有不足如下：性能和效率上，代码中使用了大量的循环和嵌套循环，可能导致性能问题，特别是对于大规模的数据集，可以尝试优化算法以提高效率。错误处理上，代码中没有足够的错误处理机制，当输入数据不合法或出现问题时，程序可能崩溃或产生不可预测的结果。应该添加更多的错误检查和处理代码。此外次题效率尤其依赖逼近链的选择，有不确定因素。

1.3 差分密码分析

（1）设计内容

差分密码分析首先选择一对明文 (P1, P2)，其中 P1 和 P2 只在几个比特上有差异，通过加密 P1 和 P2 得到密文 (C1, C2)，计算密文差分 ΔC = C1 ⊕ C2，则可以通过多次观察不同的明文对和它们的密文差分来构建一个统计模型。通过对每一轮的差分进行分析，即能生成一个差分链，通过对最后一轮的差分进行分析，即可找出最后一轮的秘钥。

（2）设计过程

差分密码攻击和线性密码攻击比较相似，首先读入明密文对，然后需要找出差分链，通过教材和网络上的查找，本题有两条差分链如图 3，4 所示

输入异或值：0x0B00，输出异或值：0x0606；

```
0B00 -> 0200 -> 0040 -> 0060 -> 0220 -> 0550 -> 0606
```

图 3

输入异或值：0x0050，输出异或值：0x5050；

```
0050 -> 0010 -> 0002 -> 0005 -> 0101 -> 0A0A -> 5050
```

图 4

其具体设计过程如下：

1. 使用计数器，在枚举密钥时若异或值输出则增加计数器的值，最后找到最有可能的最后一轮轮密钥，然后穷举前 16 位秘钥，依次取偏差率较大的子秘钥，组合成完整的秘钥，并且通过选取若干明密文对进行测试，均正确的即为最终密钥。

（3）小结

序号	结果	内存	时间
1	作答正确	4.35M	2ms
2	作答正确	4.36M	0ms
3	作答正确	4.38M	5ms
4	作答正确	6.07M	22ms
5	作答正确	6.11M	89ms
6	作答正确	6M	193ms
7	作答正确	6.11M	1000ms

图 5

运行结果如图 5 所示，成功通过所有测试案例。此实验延续了以上的一些良好策略，如自己写了快速读取的函数，通过位运算提高速率，但还是通过的比较极限，测试案例 7 正好卡 1000ms，由此我思考应该有以下优化手段：可以在不干扰检验的正确性下一定程度上减少检验密钥时的取样数；goto 语句通常被视为不良实践，因为它们可能使代码更难理解和维护，尝试通过使用循环和条件语句来重构代码，以减少 goto 的使用。

1.4 算法增强

（1）设计内容

主要就是对实验 1 的 SPN 进行了一些优化，一是对秘钥长度、分组长度和 S/P 盒优化：秘钥长度为 128 位，分组长度了 64 位，采用了 16 位的 S 盒和 64 位的 P 盒，这增加了密码算法的非线性性和扩散性。同时，轮数是 4 轮，增加了密码算法的迭代次数，进一步增强了安全性。轮秘钥生成算法优化：通过优化轮秘钥生成算法，可能采用更复杂的方式，如混合置换、置换选择等，以增强密钥的随机性和不可预测性，提高了密码的安全性。
采用分组密码加密模式：引入 CBC（Cipher Block Chaining）模式，使用较大的初始向量（IV），将明文块与前一个密文块进行异或操作，从而增加了密码的随机性和抵御了重放攻击。

（2）设计过程

substitution 数组存储了 16 个替代操作的常数，permutationArr 数组用于执行置换操作。permutationPos 数组存储了置换操作的位置映射。init_key、round_key 和 IV 存储了初始密钥、轮密钥和初始化向量。MingWen 和 MiWen 存储了明文和密文数据。

通过 permutation 函数执行置换操作，接受一个 64 位的明文作为输入，并根据 permutationPos 数组的映射执行置换操作。

SPN 函数是主要的 SPN 加密函数。该函数首先执行与初始化向量 IV 的异或操作。然后，在一个循环中，执行多轮的替代和置换操作。每一轮的轮密钥从 round_key 数组中获取。最后，与最后一个轮密钥进行异或操作，并再次执行替代和置换操作，然后返回结果。

主函数初始化了轮密钥 round_key，将初始密钥 init_key 拆分为多个子密钥。然后，它进入一个循环，每次从标准输入读取一个明文块 MingWen，将其传递给 SPN 函数以进行加密，然后将结果写入标准输出。这个循环将一直执行，直到标准输入中没有更多数据。

总体来说，实现了一个 SPN 加密算法，用于将输入的明文块加密为输出的密文块。

（3）小结

序号	结果	内存	时间
1	作答正确	1.45M	816ms
2	作答正确	1.39M	754ms
3	作答正确	1.35M	736ms
4	作答正确	1.33M	783ms
5	作答正确	1.42M	738ms
6	作答正确	1.32M	743ms
7	作答正确	1.34M	740ms

图 6

运行结果如图 6 所示，成功通过所有测试案例。我结合第一关 SPN 加密的基本算法，改进了 SPN 中的 S 盒代换，利用了 CBC 加密模式，且将密钥长度增大到 128 位，最终顺利通过了本关 。

1.5 RSA 参数计算

（1）设计内容

首先，选择两个大素数，通常称为 p 和 q，然后计算它们的乘积 n（n = p * q）。接下来，计算欧拉函数 φ(n)（φ(n) = (p - 1) * (q - 1)）。选择一个加密指数 e，满足 1 < e < φ(n)且 e 与 φ(n)互质。最后，计算解密指数 d，使得(d * e) % φ(n) = 1。公钥包含{n, e}，私钥包含{n, d}。

加密：将明文消息转换为整数 M，然后使用公钥进行加密，计算密文 C = M^e mod n。密文 C 可以被传输给接收方。

解密：接收方使用私钥进行解密，计算 M = C^d mod n，从而还原出原始的明文消息

（2）设计过程

1.通过 bn_mod_inverse 函数实现了 RSA 解密的关键部分，使用了扩展欧几里得算法来计算模反元素，这是 RSA 解密所必需的，因为只有知道模反元素才能正确解密密文。

2.bn_gcd 函数：这个函数计算两个大整数的最大公约数。通常在 RSA 密钥生成和验证阶段使用。

3.DeCipher 函数：这是主要的解密函数，它读取用户输入的公钥(e, p, q)和一些密文，并尝试对密文进行解密。在解密之前，它还检查了一些条件，例如密文长度是否满足要求，公钥是否合法，最大公约数是否为 1 等。

4.main 函数是程序的入口点，它调用 DeCipher 函数来执行解密操作。

总的来说，整个过程是将 RSA 密文解密为明文，但在解密之前会进行一些条件检查，以确保解密是有效和安全的。如果不满足一些条件，程序将输出"ERROR"。否则，它将输出解密后的明文。

（3）小结

序号	结果	内存	时间
1	作答正确	5.03M	2ms
2	作答正确	5.21M	0ms
3	作答正确	5.26M	21ms
4	作答正确	5.26M	101ms
5	作答正确	4.7M	0ms

图 7

运行结果如图 7 所示，成功通过所有测试案例。代码中使用了 OpenSSL 库中的 BIGNUM（大整数）和相关函数来进行 RSA 解密操作，加强了我对该库的熟悉情况，但要注意代码可以更好地模块化，将一些功能封装成函数，可提高可重用性和可维护性。

1.6 模重复平方计算

（1）设计内容

给定大数 m，e，p，q，用模重复平方的方法计算 expmod(m,e,p*q)，如图 8。

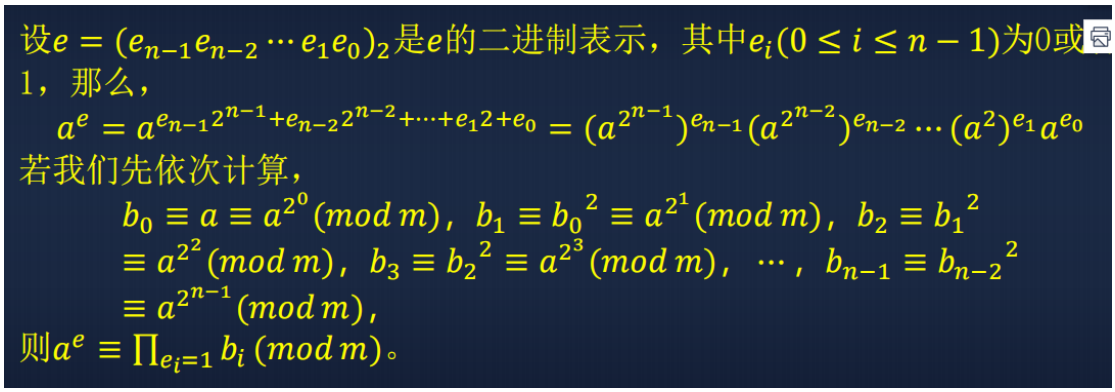


图 8

（2）设计过程

1.performModularEx 函数：执行模平方重复算法，通过将指数表示为二进制形式的位来计算模幂，以提高计算效率。它使用 Montgomery 模乘法来进行模幂运算，以减少大整数操作的复杂性。

2.DeCipher 函数：处理多个测试用例，每个测试用例包括指数、消息、以及两个素数 p 和 q。它将这些值转换为 BIGNUM 类型，并计算出模数 N。然后，它调用 performModularEx 函数来执行模幂运算，并打印结果。

3.main 函数：程序入口点，负责调用 DeCipher 函数来执行模平方重复算法。

（3）小结

序号	结果	内存	时间
1	作答正确	2.43M	145ms
2	作答正确	2.55M	291ms
3	作答正确	2.46M	219ms
4	作答正确	2.58M	336ms
5	作答正确	2.52M	397ms
6	作答正确	2.58M	577ms

图 9

运行结果如图 9 所示，成功通过所有测试案例。这一关比较简单，基本就是使用教材上的模重复平方法，本质是对二进制中不同位的数进行累加得到大数次方的模结果，再利用蒙哥马利算法来优化即可。

1.7 中国剩余定理

（1）设计内容

利用 1.5 中对加密参数 e 求逆得到解密参数 d 的算法以及 1.6 中模重复平方的算法，再结合中国剩余定理的相关知识，设计相应程序。中国剩余定理原理如图 10。

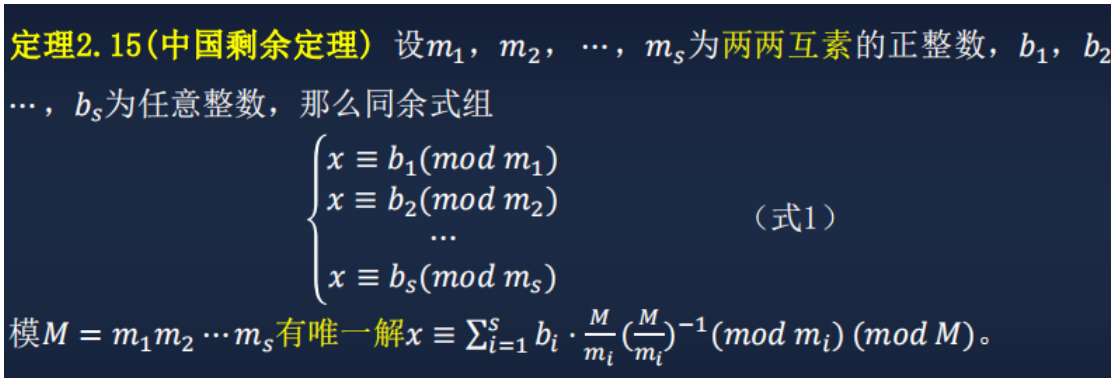


图 10

（2）设计过程

1.定义了 bn_mod_inverse 函数，用于计算大整数的模逆。

2.定义了 exp_mod 函数，执行模幂运算，这里用于计算两个模数 p 和 q 上的解密结果。

3.在 DeCiper 函数中：读取输入的密文数量 n 以及两个素数 p 和 q 以及加密后的消息 e。计算欧拉函数 fai，模数 N，并计算模数 p 和 q 上的模逆 p_ 和 q_。

4.循环处理每个密文：读取密文 c。使用 exp_mod 函数计算 p 和 q 上的解密结果 r1 和 r2。对 r1 和 r2 进行一些操作，然后合并它们以获得最终的解密结果。最后，确保结果不超过模数 N，并将解密后的消息打印出来。

总体而言，这段代码通过使用中国剩余定理，以一种高效的方式对 RSA 密文进行解密，并在每个步骤中使用 OpenSSL 的 BIGNUM 类型来处理大整数。这有助于提高解密效率，尤其是在处理非常大的密文时。

（3）小结

运行结果如图 11 所示，成功通过所有测试案例。CRT 主要用于处理大整数问题，因为它可以将复杂的问题分解为更简单的部分，提高计算效率。在加密算法（如 RSA）和计算机科学领域，CRT 非常有用，可以加速大整数的运算，提高性能。

序号	结果	内存	时间
1	作答正确	4.63M	3ms
2	作答正确	4.91M	15ms
3	作答正确	4.81M	33ms
4	作答正确	5.03M	69ms
5	作答正确	5.18M	116ms
6	作答正确	5.17M	144ms
7	作答正确	5.38M	336ms
8	作答正确	5.5M	530ms
9	作答正确	5.62M	381ms

图 11

1.8 HASH 函数与彩虹表

（1）设计内容

彩虹表（Rainbow Table）是一种预先计算并存储的数据表，用于加速密码破解过程。它是密码学和计算机安全领域的一个重要概念，主要用于破解哈希函数加密的密码。

彩虹表可以加速密码破解过程。它通过预先计算并存储大量可能的明文密码和其对应的哈希值，以便在破解时进行快速查找。这些预先计算的数据表大大减少了密码破解所需的时间和计算资源。彩虹表包含两列数据：原始密码和密码哈希值。它们按照一定的规则生成，通常使用不同的哈希函数和不同的种子值。攻击者在破解密码时，首先将目标哈希值与彩虹表中的哈希值进行比较。如果找到匹配项，攻击者可以查看相应的原始密码。

（2）设计过程

1.利用 unitSHA1 函数计算输入字符串的 SHA-1 哈希值。这个函数将输入字符串转换为 SHA-1 散列，用于后续比较。

2.通过 getString 函数将一个整数转换为 36 进制字符串。

3.使用 R 函数通过对给定的 SHA-1 哈希值进行一系列操作，生成一个新的字符串。

4.DeCipher 函数是解密的核心。它首先读取一些数据，包括一系列头部和尾部字符串以及一个目标 SHA-1 哈希值。然后，它使用一个嵌套的循环来搜索可能的字符串组合，以生成与目标 SHA-1 哈希值匹配的原始字符串。如果找到匹配项，则将原始字符串打印出来。

总体而言，这段代码是一个用于暴力搜索可能字符串组合的解密算法。它搜索原始字符串，计算其 SHA-1 哈希值，然后与目标 SHA-1 哈希值进行比较，以找到匹配项。这个算法的效率可能不高，因为它需要搜索大量的字符串组合。根据不同的输入数据，运行时间可能会很长。

（3）小结

序号	结果	内存	时间
1	作答正确	1.92M	18ms
2	作答正确	1.92M	952ms
3	作答正确	1.94M	582ms
4	作答正确	2M	795ms
5	作答正确	1.96M	73ms
6	作答正确	1.91M	560ms
7	作答正确	1.93M	1067ms
8	作答正确	2M	210ms
9	作答正确	1.93M	66ms

图 12

运行结果如图 12 所示，成功通过所有测试案例。在开始解决这个问题时，我一开始感到非常困惑，几乎没有任何头绪。我尝试了一种暴力方法，即使用所有哈希链中的节点来搜索原始密码，但很快发现这个方法的计算量太大，根本无法运行。最后，我通过查阅资料发现由于我们存储了多条彩虹链，可以利用这些链来大大提高破解的效率。

1.9 数字信封

（1）设计内容

PKCS#7（Public Key Cryptography Standards #7）是一种密码学标准，用于加密通信和数字签名中的数据格式。它定义了一种通用的数据格式，用于在计算机网络中传输和存储加密数据，以及进行数字签名和数字证书操作。

（2）设计过程

1.引入所需的 OpenSSL 头文件和命名空间。

2.定义了两个常量字符串 cacert 和 pkeyB，它们分别包含了 X.509 证书和私钥的 PEM 编码形式。这些证书和私钥用于验证数字签名和解密数字信封。

3.实现了三个主要的函数 getX509、getpkey 和 get_PKCS7，它们分别用于从 PEM 编码字符串中获取 X.509 证书、私钥和 PKCS#7 格式的数字信封。

4.verify_signature 函数用于验证数字签名的有效性。它首先从数字信封中提取签名者的信息，然后使用 X.509 证书构建 X.509 存储和上下文，最后使用 PKCS7_dataVerify 函数验证数字签名。如果所有签名都有效，函数返回 true，否则返回 false。

5.pkcs7_decrypt 函数用于解密数字信封并验证数字签名。它从标准输入中读取数字信封数据，然后将其转换为 PKCS#7 结构。接着，它使用提供的私钥 pkeyB 来解密数字信封中的数据，并获取 X.509 证书 cacert 用于验证数字签名。如果数字签名有效，它将解密后的数据输出到标准输出。

6.main 函数是程序的入口点。它初始化 OpenSSL 库，调用 pkcs7_decrypt 函数来执行解密和验证操作。

总之，通过使用 OpenSSL 库来解密 PKCS#7 数字信封并验证数字签名的有效性。

（3）小结

图 13

运行结果如图 13 所示，成功通过所有测试案例。此题主要是研究库函数，在查找 OpenSSL 中的 PKCS#7 函数时，我成功地找到了一些与数字签名和解密数字信封有关的函数。接下来，我分析了这些函数的功能并结合之前对蒙哥马利算法的经验，逐步拼凑出了验证数字签名的完整流程。通过这一过程，我成功地实现了该算法，完成了对加密通信中数字信封的解密和数字签名的验证，这种自主学习和查找相关资料的能力对于解决实际问题和开发复杂的加密应用非常重要。

二、实验心得

在进行密码学和加密算法的实验中，我深刻体验到了密码学在信息安全领域的关键作用。这些实验不仅让我深入了解了各种常见的加密算法和攻击方法，还提高了我的算法设计和密码学概念的理解。

SPN（Substitution-Permutation Network）：通过 SPN 算法，我学会了如何将简单的替代和置换操作组合成一个强大的块密码算法。了解 SPN 结构以及如何设计和实现它，为我理解对称密钥加密提供了重要基础。

线性密码分析和差分密码分析：这两个实验让我明白了密码分析方法是如何工作的。线性密码分析和差分密码分析揭示了密码算法中的弱点，并让我更深入地了解密码分析的概念和技术。

SPN 算法增强：通过增强 SPN 算法，我了解了如何提高密码算法的安全性。这包括增加轮数、改进替代和置换盒的设计，以及使用更强大的密钥调度算法。

RSA 解密：理解了 RSA 加密和解密的原理以及如何使用 OpenSSL 进行实际操作。这个实验揭示了非对称密钥加密的概念和操作。

模重复平方算法：这个算法展示了如何高效地进行模幂运算，对于密码学中的许多应用都至关重要，如 RSA 加密。

中国剩余定理：理解了中国剩余定理的原理和用途。这个定理在解决模数大的情况下，能够有效地加速计算。

数字信封：数字信封是一种常见的加密和认证方法，允许发送者安全地传输消息给接收者，同时保证消息的机密性和完整性。我学会了如何创建和验证数字信封。

Hash 函数与彩虹表：Hash 函数是密码学中不可或缺的组成部分，用于生成数据的固定长度摘要。彩虹表则是一种高效的密码破解方法，这个实验让我深入了解了它们的工作原理和应用。

总的来说，这些实验为我提供了丰富的密码学知识和实际操作经验。我认识到了密码学在保护信息安全和数据隐私方面的关键作用，也明白了密码学领域仍然存在许多挑战和机会，需要不断学习和探索。这些实验经验将对我未来在信息安全和加密领域的学术研究和职业发展产生积极影响。

三、对课程设计内容和过程的建议

1.有几道题目时间卡的太紧了，例如差分密码分析，可以适当放宽一点要求。

2.TASSL 库函数的安装有些不友好，希望老师提供相关资料。