密码学第八次实验报告

椭圆曲线相关算法

原理

椭圆曲线是定义在射影平面中的一种曲线. 在密码学中能够应用的椭圆曲线的方程形式为

$$y^2 = x^3 + ax + b \quad (4a^3 + 27b^2 \neq 0)$$

这种曲线原来是定义在 $\mathbb{R}$ 上的, 但是也可以扩充到有限域 F 上. 这样更适合密码学应用.

由于椭圆曲线定义在射影平面上, 在普通平面上表示会丢失射影平面上的无穷远点, 所以还要定义一个无穷远点 P_∞.

定义椭圆曲线在同一条直线上的三个点之和为 P_∞. 可以证明出椭圆曲线上所有点关于这种加法形成一个 Abel 群, 加法单位元为 P_∞. 同时, 可以定义 $2P$ 为过点 P 的切线与椭圆曲线的另一个交点. 若无交点, 定义 $2P$ 为 P_∞. 这样, 就可以定义椭圆曲线上点的数乘.

椭圆曲线上点的加法可以推导出公式. 同样的, 求某个点左乘 2 也可以推导出公式. 这样, 椭圆曲线上点的加法和数乘都可以推导出公式, 这就把几何操作转化成了代数运算.

椭圆曲线有以下困难问题: 已知 P 和 $kP(k \in \mathbb{Z})$, 求 k. 该问题被称作椭圆曲线上的离散对数问题, 可以用来构造椭圆曲线上的公钥密码体制.

伪代码

算法 1: 椭圆曲线上点的加法算法

```
输入: 椭圆曲线上的两个点 A 和 B
输出: 椭圆曲线上的点 C
数据: 椭圆曲线参数 a, b
if A = P∞ and B = P∞ then
    return P∞
end
if A = P∞ then
    return B
end
if B = P∞ then
    return A
end
if x_A ≠ x_B then
    δ = (y_B − y_A)/(x_B − x_A)
    x_C = δ^2 − x_A − x_B
    y_C = −y_A + δ(x_A − x_C)
    return C
else
    if y_A = y_B then
        δ = (3x_A^2 + a)/(2y_A)
        x_C = δ^2 − 2x_A
        y_C = −y_A + δ(x_A − x_C)
        return C
    else
        return P∞
    end
end
```

算法 2: 椭圆曲线上点的数乘运算

```
输入: 整数 n，椭圆曲线上的点 P
输出: 结果 R
数据: 椭圆曲线参数 a, b
if n < 0 then
    抛出参数错误
end
if n = 0 then
    return P_∞
end
if n = 1 then
    return P
end
if n = 2 then
    return P + P
end
curr_factor ← null for n 从低到高的每个二进制位 do
    if curr_factor = null then
        curr_factor ← self
    else
        curr_factor ← 2* curr_factor
    end
    if 该位为 1 then
        result ← result + curr_factor
    end
end
```

分析

点的加法

很显然, 对点的加法的各种情况, 时空复杂度为 $\mathrm{O}(1)$.

点的数乘运算

设要乘的数为 n.

由于要遍历 n 的每一位, 所以易知时间复杂度为 $\mathrm{O}(\log n)$. 每步之间没有数据关联, 所以空间复杂度为 $O(1)$.

优化

点的加法

实际上, 可能可以通过对点的加法的代数性质, 对点的加法进行优化. 我在网络上也找到了 NIST 的一种优化方法.

点的数乘运算

对点的数乘运算, 由于前后数据相关性较大, 没有很好的优化方法. 但是可以借用模平方乘算法的优化方式 (如加法链) 来优化.

椭圆曲线上的 Diffie-Hellman 密钥交换协议

原理

Diffie-Hellman 密钥交换协议是基于离散对数问题求解困难性的. 椭圆曲线上的 Diffie-Hellman 密钥交换协议是基于椭圆曲线上离散对数问题求解困难性的.

密钥交换双方首先生成 X_A, X_B 作为私钥, 然后生成公钥 $Y_A = X_AP$, $Y_B = X_BP$, 其中 P 为椭圆曲线上规定好的基点. 收到对方的公钥时, 能够得到最终的密钥 $Y = X_BY_AP = X_AY_BP$, 完成密钥交换.

伪代码

密钥生成算法

算法 3: Diffie-Hallman 密钥交换协议密钥生成算法

输入: 无

输出: 有限域上的数 n，椭圆曲线上的点 p

数据: 椭圆曲线的参数 a, b, 它的一个生成元（同时也是系统公开参数）G

$n \leftarrow (1, n)$ 上的随机整数 $p \leftarrow n * G$ **return** n, p

密钥获取算法

算法 4: Diffie-Hallman 密钥交换协议密钥获取算法

输入: 己方私钥 x，获取到的数据 P

输出: 椭圆曲线上的点 P'

$P' \leftarrow xP$

return P'

分析

由于所有算法中数据的上界已经给定, 且各子算法的时空复杂度已知, 所以两个算法的时空复杂度都是 $\mathrm{O}(1)$.

测试

测试模块为 `diffie_hellman.py`. 主要的测试功能是测试密钥生成和接收算法及其正确性, 能够测试通过.

优化

实际上, 两种算法比较简单, 我看不出明显的优化空间. 但是, 可以通过对子算法的优化, 来间接地优化两种算法.

椭圆曲线上的 ElGamal 公钥密码体制

原理

椭圆曲线上的 ElGamal 公钥加密算法也是基于椭圆曲线上的离散对数问题, 并且与常规的 ElGamal 公钥密码体制类似.

首先约定一条 F_p 上的椭圆曲线 $E_p(a,b)$, 它的一个生成元 G, 以及一个不超过 p 的数 n.

密钥生成算法

Alice 首先选择 $(1,p)$ 上的一个随机数 d, 把 d 作为私钥, $Q=dP$ 作为公钥.

加密算法

Bob 把满足 $1 \leq m \leq n$ 的消息 m 表示成 F_p 上的元素, 使用的字母不变. 然后他在 $[1,n-1]$ 内选择一个随机数 k, 计算 $C_1 = kP$. 然后计算 $(x_2,y_2)=kQ$, 若 $x_2=0$, 则重新生成 k, 重新计算. 然后计算 $C_2=mx_2$, 并传送密文 (C_1,C_2) 给 Alice.

解密算法

Alice 使用私钥 d, 计算 $D = dC_1$, 再计算 F_p 中 D 的横坐标 x_2 的逆元 x_2^{-1}. 然后通过 $m=C_2x_2^{-1}$ 恢复出明文 m.

伪代码

算法 5: ElGamal 公钥密码体制密钥生成算法

输入: 无
输出: 私钥 n，公钥 P
数据: 椭圆曲线的参数 a, b，它的一个生成元 G

```
n ← (1, n) 中的随机数
P ← nG
return n, P
```

算法 6: ElGamal 公钥密码体制加密算法

输入: 整数形式的消息 m，公钥 P
输出: 椭圆曲线上的点 x_1，整数 c
数据: 椭圆曲线的参数 a, b，它的一个生成元 G

```
if p >= n or p < 0 then
    抛出参数错误
end
while true do
    k ← [1, n − 1] 中的随机数
    x1 ← kG
    x2 ← kP
    if x2 的横坐标 = 0 then
        continue
    end
end
c ← p x2 的横坐标
return x1, c
```

算法 7: ElGamal 公钥密码体制解密算法

输入: 密文对 $(x_1, c$，私钥 n

输出: 椭圆曲线上的点 x_1，整数 c

数据: 椭圆曲线的参数 a, b，它的一个生成元 G

if $c \geq n$ *or* $c < 0$ **then**

  抛出解密出错

end

$x_2 = nx_1$

$x_2' \leftarrow x_2$ 的横坐标

$m = cx_2'$

return m

分析

由于数据的上界已知, 各子算法的时空复杂度也已知, 所以时空复杂度为 $\mathrm{O}(1)$.

测试

由于测试要求使用文件, 所以采用了文件加解密方式进行测试. 经过测试, 恢复的明文文件和原明文文件相同. 测试模块为 `elgamal_test.py`.

优化

主要算法

其实对主要算法, 我也没想出很好的方式来优化. 但是实际上同样也可以通过优化子算法来优化算法的效率.

文件加解密算法

对文件加解密算法, 可以利用处理器的并行性来优化, 也可以提前算出需要的中间值.

SM2 公钥密码体制

原理

SM2 公钥密码体制也是基于椭圆曲线上的离散对数问题的。但是，它有两点非常独特，一个是通过一个密钥生成函数（KDF）来生成密钥，另一个是它通过哈希函数实现了消息鉴别功能，提高了敌手伪造消息的成本。

SM2 算法需要使用国家密码管理局批准的哈希算法，这里使用的是 SM3 算法。事实上，SM2 密码体制支持多种有限域及多种点的表示，这里只实现了 F_p 域上点的未压缩表示。

伪代码

算法 8: SM2 公钥密码体制密钥生成算法

输入: 无

输出: 有限域上的点（私钥）d，椭圆曲线上的点（公钥）P

数据: 椭圆曲线的参数 a, b，它的生成元 G

$d \leftarrow [1, n-2]$ 中的随机数

$P \leftarrow dG$

return d, P

算法 9: SM2 公钥密码体制密钥派生函数（KDF）

输入: 字节串 Z，密钥长度 $kLen$

输出: 密钥字节串 K

数据: 哈希函数的输出位长度 v

ct 为一个 32 位的计数器变量

$ct \leftarrow$ 0x00000001

for i = *1* ... $\lceil kLen/v \rceil$ **do**

  $Ha_i \leftarrow Hash(Z \parallel ct$

  $ct \leftarrow ct + 1$

end

if $kLen \mod v = 0$ **then**

  $Ha_{\lceil kLen/v \rceil} \leftarrow Ha_{\lceil kLen/v \rceil}$ 最左边的 $kLen - (v\lfloor kLen/v \rfloor)$ 位

end

$K \leftarrow Ha_1 \parallel Ha_2 \parallel \cdots \parallel Ha_{\lceil kLen/v \rceil}$

return K

算法 10: SM2 公钥密码体制加密算法

输入: 明文 m，公钥 P

输出: 密文 c

数据: 椭圆曲线的生成元 G

while *true* **do**

  $k \leftarrow [1, n-1]$ 中的随机数

  $c_1 \leftarrow kG$

  把 c_1 转换为字节串

  $h \leftarrow 1$

  $s \leftarrow hP$

  if $s = P_\infty$ **then**

    抛出错误

  end

  $x_2, y_2 \leftarrow ks$ 的横坐标和纵坐标

  把 x_2, y_2 转换为字节串

  $kLen \leftarrow m$ 的位长度

  $t \leftarrow KDF(x_2 \parallel y_2, kLen)$

  if t 非全 0 的字节串 **then**

    break

  end

  $c_2 \leftarrow m \oplus t$

  $c_3 \leftarrow Hash(x_2 \parallel m \parallel y_2)$

  return $c_1 \parallel c_2 \parallel c_3$

end

算法 11: SM2 公钥密码体制解密算法

输入: 密文 c，私钥 d

输出: 明文 m

数据: 椭圆曲线参数 a, b，椭圆曲线的有限域 F_p 的模数 p

从 c 中提取 c_1, c_2, c_3

if 按照格式出现索引错误 **then**

  抛出错误

end

if c_1 不在椭圆曲线上 **then**

  抛出错误

end

$x_2, y_2 \leftarrow dc_1$ 的横坐标和纵坐标

把 x_2, y_2 转换成字节串

$kLen \leftarrow c_2$ 的位长度

$t \leftarrow KDF(x_2 \parallel y_2, kLen)$

if t 为全为 0 的字节串 **then**

  抛出错误

end

$m \leftarrow c_2 \oplus t$

$u \leftarrow Hash(x_2 \parallel m \parallel y_2)$

if $u \neq c_3$ **then**

  抛出错误

end

return m

分析

设消息的长度为 m。

密钥生成算法

该算法步骤恒定，因此时空复杂度都是 $\mathrm{O}(1)$。

加密算法

要注意的一点是，实际上 KDF 是需要不停地保存中间值的，这些中间值的长度跟 m 成正比，计算它们的时间也是。同样地，计算哈希值也要考虑输入数据的长度。因此，算法的总时空复杂度分别是 $\mathrm{O}(m)$ 和 $\mathrm{O}(m)$.

解密算法

类似地，算法的总时空复杂度分别是 $\mathrm{O}(m)$ 和 $\mathrm{O}(m)$。

测试

采用自动化脚本进行测试，脚本文件名为 `sm2_test.py`。它可以测试加密和解密算法的正确性，以及自动加解密一个文件作为测试。经过测试，加密和解密算法互为逆，解密的文件也与加密的文件相同，因此算法正确。

优化

密钥派生函数

实际上，哈希函数的值可以并行计算，因为数据以来并不高。其实，结果 K 的值同样可以并行。这种并行对多核 CPU 更为有利，因为不同的 ct 值触发的代码执行路径一般是不相同的，在一个 CPU 上，分支预测器容易迷惑。而且，并行更有利于利用每个 CPU 的局部性。

加密算法和解密算法

这两个算法实际上可以优化数据结构，这样可以简化对点的转换。但是并行性不够好，算法的分支情况也不大好预测。不过局部性还是比较强的。其实可以用上比较新的 x86 指令集，这样能直接支持 256 位整数运算。

SM9 公钥密码体制

原理

SM9 公钥密码体制是基于身份的密码体制，因此能够简化对用户的认证等过程。用户需要信任一个密钥分发中心 KDC，由 KDC 生成密钥。

SM9 系列密码算法的数学基础，是双线性对，这是一种建立在循环群 G_1, G_2 上的映射，并且映射到另一个循环群 G_T。双线性对 e 最基本的性质，是对任意自然数 a, b 和分别在 G_1, G_2 上的元素 m_1, m_2，$e(am_1, bm_2) = ab \cdot e(m_1, m_2)$。这里假定三个群都是加群。

在 SM9 中，双线性对的 G_1 是通常意义上椭圆曲线的一部分点构成的一个群，但 G_2 和 G_T 比较特殊，是首先得到 F_{q^2} 和 $F_{q^{12}}$（其中 q 为大素数）这两个扩域，再基于 F_{q^2} 上定义的椭圆曲线上的点，这样来构成两个循环群。实际上，从 F_q 到 $F_{q^{12}}$ 经历了三次扩张，这种方法也叫塔式扩张方法。为了能保证 G_1, G_2 和 G_T 确实是循环群，需要让这三个群、椭圆曲线的一些参数和扩张方法满足一些性质。

由于 F_{q^2} 和 $F_{q^{12}}$ 是扩域，且扩张用到的本原多项式已知，所以在实现时可以进行优化。大体的思路是借助多项式本身的结构，把这两个域上的元素表示成有序对的形式。然后，就可以在有序对上做计算了。表示成有序对形式，实际上就是表示成多项式形式，有序对的元素按一定顺序组成多项式的系数。这是符合抽象代数原理的。然后，根据标准的规定，让多项式高次项的系数放在有序对的前面。

由于时间不够，所以实际上 SM9 公钥密码体制没写完，只写完了参数定义、F_{q^2} 上元素的计算、KDC 生成两种加密密钥的部分、密码杂凑函数和密钥派生函数这几个部分。

SM9 公钥密码体制利用双线性对，不仅是基于椭圆曲线上求离散对数困难问题，还基于椭圆曲线上双线性对的几个困难问题，进一步增强了安全性，也增强了它的实用性。

伪代码

以下对数的四则运算，都是在 F_q 上进行的。

算法 12: F_{q^2} 上元素的加法

输入: F_{q^2} 上的两个元素 a, b

输出: F_{q^2} 上的元素 c

把 a 和 b 分别表示成 (a_1, a_2) 和 (b_1, b_2) 的形式

$c_1, c_2 \leftarrow a_1 + a_2, b_1 + b_2$

根据 (c_1, c_2) 得到 c

return c

算法 13: F_{q^2} 上元素的减法

输入: F_{q^2} 上的两个元素 a, b

输出: F_{q^2} 上的元素 c

把 a 和 b 分别表示成 (a_1, a_2) 和 (b_1, b_2) 的形式

$c_1, c_2 \leftarrow a_1 - a_2, b_1 - b_2$

根据 (c_1, c_2) 得到 c

return c

算法 14: F_{q^2} 上元素的乘法

输入: F_{q^2} 上的两个元素 a, b

输出: F_{q^2} 上的元素 c

把 a 和 b 分别表示成 (a_1, a_2) 和 (b_1, b_2) 的形式

$c_1, c_2 \leftarrow a_2b_2 - 2a_1b_1, -2(a_1b_2 + a_2b_1)$

根据 (c_1, c_2) 得到 c

return c

算法 15: F_{q^2} 上元素求逆元

输入: F_{q^2} 上的元素 a

输出: F_{q^2} 上的元素 c

把 a 表示成 (a_1, a_2) 的形式

$a_1, a_2 \leftarrow -\frac{a}{b^2+2a^2}, \frac{b}{b^2+2a^2}$

根据 (c_1, c_2) 得到 c

return c

算法 16: F_{q^2} 上元素的除法

输入: F_{q^2} 上的两个元素 a, b

输出: F_{q^2} 上的元素 c

$c \leftarrow a \cdot b$ 的逆元

return c

算法 17: 系统加密主密钥生成算法

输入: 无

输出: 加密主私钥 ke，加密主公钥 P_{pub-e}，加密私钥生成函数识别符 hid

数据: 循环群 G_1, G_2 和 G_T 的阶 N，循环群 G_1 的生成元 P_1

$ke \leftarrow [1, N-1]$ 中的随机数

$P_{pub-e} \leftarrow ke \cdot P_1$

选择并公开 1 个字节长的加密私钥生成函数识别符 hid

return ke, P_{pub-e}, hid

SM9 公钥密码算法中用到的密钥派生函数 KDF 与 SM2 的基本一样，因此省略了伪代码。

算法 18: 用户加密密钥生成算法

输入: 加密私钥生成函数识别符 hid，加密主私钥 ke，用户标识 ID

输出: 用户的加密私钥 de

数据: 循环群 G_1, G_2 和 G_T 的阶 N，循环群 G_2 的生成元 P_2

$t_1 \leftarrow H_1(ID \parallel hid, N) + ke$

if $t_1 = 0$ **then**

　抛出错误

end

$t_2 \leftarrow ke \cdot t_1^{-1}$

$de \leftarrow t_2 P_2$

return de

算法 19: 密码函数 H_1

输入: 字节串 Z，整数 n

输出: 整数 h_1

数据: 哈希函数 $H_v(m)$，哈希函数的输出位长度 v

32 位大端序整数 $ct \leftarrow$ `0x00000001`

$hLen \leftarrow 8\lceil \frac{5\log_2 n}{32} \rceil$

for $i = 1 \ldots \lceil \frac{hLen}{v} \rceil$ **do**

　$Ha_i \leftarrow H_v($`0x01` $\parallel Z \parallel ct)$

　$ct \leftarrow ct + 1$

end

if $\frac{hLen}{v}$ 不是整数 **then**

　$Ha_{\lceil \frac{hLen}{v} \rceil} \leftarrow Ha_{\lceil \frac{hLen}{v} \rceil}$ 最左边的 $\frac{hLen - v\lfloor \frac{hLen}{v} \rfloor}{8}$ 字节

end

$Ha \leftarrow Ha_1 \parallel Ha_2 \parallel \cdots \parallel Ha_{\lceil \frac{hLen}{v} \rceil - 1} \parallel Ha_{\lceil \frac{hLen}{v} \rceil}$

把 Ha 按大端序转换为整数

$h_1 \leftarrow (Ha \mod (n-1)) + 1$

return h_1

密码函数 $H_2(Z, n)$ 除了使用的常量从 `0x01` 变成了 `0x02` 以外，其它地方都一样，因此省略了伪代码。

分析

系统加密主密钥和用户加密密钥生成算法

易知算法的空间需求和计算量都是有上界的。因此算法的时空复杂度都是 $\mathrm{O}(1)$。

密码函数 $H_1(Z,n)$ 和 $H_2(Z,n)$

设 Z 的长度为 l。

它们的结构完全一样，只是参数不同，因此可以一块分析。它们需要计算哈希函数的次数易知量级是 $\mathrm{O}(l)$，需要的空间也是这个量级的，因为块数是这个量级的。因此，该算法的时空复杂度都是 $\mathrm{O}(l)$。

优化

由于我把 SM9 公钥加密体制的文档看了一下，所以会有对我没有实现的算法的优化方案。

系统加密主密钥和用户加密密钥生成算法

该算法可以通过缓存随机数的方式进行优化。KDC 可以缓存几个随机数，然后提前生成密钥，等待新用户来时进行分配。但是，这样做的一个缺点就是提前生成的数据需要放在安全的地方。

还有一种方法是根据数学性质进行优化，提前缓存好 P_1 的 2 的各次幂倍，需要用到的时候再加起来。

密钥派生函数 KDF

基本上与 SM2 公钥加密算法一样，因此优化也类似。

密码函数 $H_1(Z, n)$ 和 $H_2(Z, n)$

该函数的哈希也是有前缀的，而且哈希重复计算。一般用到的哈希函数是分组链接结构的哈希函数，如 SM3，因此可以通过类似于 HMAC 算法的方式进行优化，也就是提前把新的初始向量 iv' 算出来。但是，这种方法需要注意好分块的细节。

另外一种方式就是并行计算，也就是每次计算多个 ct 的值。这样也能提高计算效率。

消息认证码函数

这里消息认证码函数不是标准的 HMAC，是对它的拼接，而且密钥放在后面，因此感觉比较难优化。不过，密钥放在后面可以提高安全性，因为这样攻击者没法通过穷举密钥来穷举相应的初始向量 iv，从而简化穷举攻击。

密钥封装算法

这里随机数 r 和 C 可以提前生成，g 也可以。这样能够简化需要时的计算量，但是也要注意 r 和 C 的保密。

密钥解封装算法

这个算法数据依赖性比较高，我感觉优化比较难，难以想出怎么优化。

加密算法和解密算法

感觉这两种算法和密钥封装和解封装算法的结构比较相似，因此也可以用优化它们的方法进行优化。

测试

因为我写完这一点的时候，提交快要截止了，因此测试缺少了。

总结

椭圆曲线相关算法

这些算法主要是对书上的公式的理解, 以及区分好各种特殊情况, 尤其是关于 P_∞ 的情况. 还有一种特殊情况值得注意: 自己与自己相加 (或者说乘以 2).

椭圆曲线上的 Diffie-Hellman 密钥交换协议

这一部分也主要是对书上公式的理解. 但是这一部分主要是一个协议原语, 所以能实现的主要是一方, 但测试时要测试双方, 这一点要注意.

SM2 公钥密码体制

这个公钥密码体制其实是椭圆曲线密码体制的更复杂的应用，其实更有意思。SM2 公钥密码体制实际上更为实用，因为它加上了消息鉴别的功能。

椭圆曲线上的 ElGamal 公钥密码体制

这部分其实是以 Diffie-Hallman 密钥交换协议作为基础的. 其实该密码体制最重要的是它的随机性和密文扩张, 所以在与文件配合时要注意, 保存的文件会更大, 而且具有随机性.

SM9 公钥密码体制

这部分因为时间和复杂程度关系，没有写完。但是我感觉 SM9 公钥密码体制和国密算法中其它商用密码体制一样，都有很高的实用性，同时也保证了安全性。它的一大特点，是不仅有基于身份的加密，而且有密钥封装和解封算法，简化了密钥的分发，这是十分实用的。同时，我感觉它的数学基础也比较先进。而且，SM9 公钥密码体制不仅基于椭圆曲线上的离散对

数问题这个困难问题，还基于椭圆曲线上双线性对的几个困难问题，进一步增强了安全性。

算法评估与优化

对这次实验里的算法, 我其实没有想到太好的优化方法. 但是, 我感觉真正的优化应该去掉抽象, 把底层的椭圆曲线算法和顶层的算法结合起来, 虽然这只是一个直觉.

系统设计与维护

这次实验的完成, 实际上多少考验着系统设计能力. 椭圆曲线上点的算法, 实际上可以从点坐标所在的数域中抽象出来, 是点坐标的四则运算. 这时, 就可以使用运算符重载的方法, 来把对点坐标的运算抽象地写出来.

同时, 对于下层的 F_p 中的元素, 肯定要抽象成类. 把 p 固定, 会使得该类并不能可移植, 因此, 一种解决办法是给一个函数, 输入 p, 返回表示 F_p 中元素的类. 这样就类似于元类 (实例是类的类), 不过这里是生成类的函数.

对于椭圆曲线的各种参数, 可以把参数都封装到一个单独的模块里. 这里当然也得在那个模块中声明椭圆曲线对应的 F_p. 声明完以后, 关于椭圆曲线的密码学原语只要导入对应的模块, 就能使用相应的椭圆曲线了.

对文件的 ECB 模式加解密, 也可以考虑把负责 ECB 模式的代码抽象出来, 形成一个函数. 输入的时候把输入和输出文件名、分块加密或解密函数、加密或解密时相应分块的长度当做函数参数输入进去, 负责 ECB 模式的函数就可以执行对文件的加解密了. 这样能减小重复代码的量, 也有利于代码的可移植性, 更好地实现和测试其它加密算法.

最终, 负责加解密的上层代码基本上不超过 20 行, 而且跟课件上的说明差不多是一致的. 这样极大地方便了开发, 也明显降低了 bug 发生的概率. 感觉这个实验其实比较需要总体的系统设计, 因为我想了两个小时, 才把大致的系统构架图想明白, 之后才开始编程.

对课程的建议

感觉这次密码学实验课程终于回到正轨了, 不再出难死人的大整数题了……这次我感觉思路比较正, 就是考对课上内容的理解和熟练应用, 以及系统的合理设计, 其实并没有什么太刁钻或太细节的地方. 感觉以后密码学实验其实应该把系统设计的良好程度放入总评, 这样也对得起我们想出良好的系统结构的努力.

总结

这次实验感觉路子比较正, 也很能考验系统设计能力. 当然, 我也感觉很有意思. 但是, 我感觉非常可惜的一点, 是我白白花了两个小时却找不到合适的资料. 所以希望以后老师或者助教学长们能帮我们找一下, 哪怕找最必需的也好.