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

点的加法

```
def __add__(self, other):
    if self == $P_\infty$ and other == $P_\infty$:
        return $P_\infty$
    if self == $P_\infty$:
        return other
    if other == $P_\infty$:
        return self
    if $x_{self}$ != $x_{other}$:
        delta = $\frac{y_{other}-y_{self}}{x_{other}-x_{self}}$
        result.x = $delta^2 - x_{self} - x_{other}$
        result.y = $-y_{self} + delta * (x_{self} - x_{result})$
```

```
    else:
        if y_self == y_other:
        intermediate = (3x_self^2 + a) / (2y_self)
        result.x = intermediate^2 − 2x_self
        result.y = intermediate(x_self − x_result) − y_self
    else:
    return P∞
```

点的数乘运算

```
def __rmul__(self, other):
    if other < 0:
        raise 参数错误
    if other == 0:
        return P∞
    if other == 1:
        return self
    if other == 2:
        return self + self
    curr_item = None
    result = P∞
    for other 从低到高的每一位:
        if curr_item is None:
            curr_item = self
        else:
            curr_item = 2 * curr_item
        if 该位为 1:
            result += curr_item
    return result
```

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 O(1).

点的数乘运算

「」　　设要乘的数为 m.

由于要遍历 m 的每一位, 所以易知时间复杂度为 O$(\log m)$. 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 $Y_A = X_A P$, $Y_B = X_B P$, 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 $Y = X_B Y_A P = X_A Y_B P$, 完成密钥交换.

伪代码

密钥生成算法

```
def gen_key() -> Tuple[fp, ECPoint]:
    x = (1, n) 上的随机整数
    return x, xP
```

密钥获取算法

```
def retrieve_key(sk: fp, p_: ECPoint) -> ECPoint:
    return sk · 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 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 $m = C_2 x_2^{-1}$ 恢复出明文 m.

伪代码

密钥生成算法

```
def gen_key() -> Tuple[fp, ECPoint]:
    x = (1, n) 中的一个随机数
    return (x, xG)
```

加密算法

```
def encrypt(p_: int, pk: ECPoint) -> Tuple[ECPoint, int]:
    if p_ >= n or p < 0:
        raise 参数错误
    while True:
        k = [1, n − 1] 中的一个随机数
        x1 = kG
        x2 = k · pk
        if x2 的横坐标 == 0:
            continue
        c = p.x2 的横坐标
      return (x1, c)
```

解密算法

```
def decrypt(c: Tuple[ECPoint, int], sk: fp) -> int:
    x1 = c[0]
    c_int = c[1]
    if c_int >= n or c_int < 0:
        raise 解密出错
    x2 = sk · x1
    m = c_int · x2 的横坐标
    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 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.

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.

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.

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.

但是这次还有一个明显的缺陷, 就是我不知道怎么实现高级要求.我不知道国家密码局批准了什么样的随机数生成器. 我觉得我无法写出合规的SM2 算法, 所以我只好放弃高级要求了. 同样地, 关于 SM9 算法, 我确实在国家密码局网站找到了官方资料, 但是英文文献实在太繁, 再加上我估计自己看不懂且需要一定学术水平, 于是也放弃了. 因此我在找资料上白白浪费了两个多小时, 却没有结果, 我感觉很可惜. 所以我有一个很冒昧的想法: 希望在布置实验任务以前, 能先评估一下可行性.

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.