密码学实验报告 实验八

2019 年 5 月 22 日

1 椭圆曲线相关运算

1.1 算法原理

椭圆曲线密码学（ECC）是一种比 RSA 安全性更高的密码学算法，其安全性建立在由kP和P确定k的困难问题上，称为椭圆曲线离散对数问题（ECDLP）。目前已知求解该困难问题的最快算法的时间复杂度是指数级，而非一般数域上的离散对数问题（以及大数分解问题）的亚指数级，所以在同等安全性的条件下，ECC 的密钥更短，计算量也较小，适合于嵌入式设备中。

通常将F_p上的一条椭圆曲线描述为

$$T = \{p, a, b, G, n, h\}$$

其中p, a, b确定数域F_p上的一条椭圆曲线，G为基点，n为点G的阶，h为余因子，即椭圆曲线上所有点的个数m与n相除的商的整数部分。

椭圆曲线方程通常表述为

$$y^2 \mod p = (x^3 + ax + b) \mod p$$

若$(x^3 + ax + b) \mod p$ 无重复因子，则基于集合$E_p(a, b)$可定义一个有限 Abel 群，该条件等价于$(4a^3 + 27b^2) \mod p \neq 0$。ECC 的加法运算对应于 RSA 中的模乘运算，ECC 的乘法运算对应于 RSA 中的模幂运算，定义 ECC 中的加法运算和乘法运算如下。

对任何点$P, Q \in E_p(a, b)$：

(1) $P + O = P$

(2) 若$P = (x_P, y_P)$，则点$(x_P, -y_P)$是P的负元，记为$-P$。

(3) 若$P = (x_P, y_P)$，$Q = (x_Q, y_Q)$，且$P \neq -Q$则$R = P + Q = (x_R, y_R)$由下式得到

$$x_R = (\lambda^2 - x_P - x_Q) \mod p$$

$$y_R = (\lambda(x_P - x_R) - y_P) \mod p$$

其中

$$\lambda = \begin{cases} \left(\dfrac{y_Q - y_P}{x_Q - x_P}\right) \mod p, & P \neq Q \\ \left(\dfrac{3x_P^2 + a}{2y_P}\right) \mod p, & P = Q \end{cases}$$

(4) 乘法定义为重复相加，$kP = \sum_{i=1}^{k} P$。

除此之外，在椭圆曲线密码体制中，我们常常需要将一个点与较大的数做乘法，传统的乘法算法会存在效率过低的问题，我们延伸了模幂算法中的“平方-乘”算法，将乘法的时间复杂度由$O(n)$降到了$O(\log n)$，提高了算法的效率。

1.2 算法实现的伪代码

椭圆曲线加法 计算椭圆曲线上两点相加的和

输入： 椭圆曲线点p, q，椭圆曲线参数a, b，素数域的阶m

输出： 椭圆曲线点$p + q$

function $add(p, q)$
  if $p = O$ **then**
    return q
  end if
  if $q = O$ **then**
    return p
  end if
  if $p = -q$ **then**
    return O
  end if
  if $p = q$ **then**
    $\lambda \leftarrow \frac{3x_p^2 + a}{2y_p} \mod m$
  else
    $\lambda \leftarrow \frac{y_q - y_p}{x_q - x_p} \mod m$
  end if
  $x_r \leftarrow \lambda^2 - x_p - x_q \mod m$
  $y_r \leftarrow \lambda(x_p - x_r) - y_p \mod m$
  return r
end function

椭圆曲线减法 计算椭圆曲线上两点相减的差

输入： 椭圆曲线点p,q，椭圆曲线参数a,b，素数域的阶m
输出： 椭圆曲线点$p-q$

```
function sub(p, q)
    r ← add(p, −q)
    return r
end function
```

椭圆曲线数乘 计算椭圆曲线上点的数乘

输入： 椭圆曲线点p，整数k
输出： 椭圆曲线点kp

```
function mul(p, k)
    r ← O
    for i ← 1 to k do
        r ← add(r, p)
    end for
    return r
end function
```

椭圆曲线快速数乘 快速计算椭圆曲线上点的数乘

输入： 椭圆曲线点p，整数k
输出： 椭圆曲线点kp

```
function faster_mul(p, k)
    r ← O
    while k ≠ 0 do
        if k mod 2 ≠ 0 then
            r ← add(r, p)
        end if
        p ← add(p, p)
        k ← ⌊k/2⌋
    end while
    return r
end function
```

1.3 测试样例及运行结果

我们选择的椭圆群是$E_{23}(1,1)$，即曲线$y^2=x^3+x+1$，对应参数$a=1,b=1,p=23$，以及曲线上两个点$p=(3,10),q=(9,7)$，对椭圆曲线上的加法、减法、数乘进行了验证，结果如表 1 所示。

表 1 椭圆曲线运算的实验验证

数据	运算结果

p	(3, 10)
q	(9, 7)
$-p$	(3, -10)
$p+q$	(17, 20)
$p-p$	Infty
$233*p$	(0, 1)

2 Diffie-Hellman 密钥交换协议算法实现

2.1 算法原理

利用椭圆曲线可以实现 Diffie-Hellman 密钥交换协议。首先，挑选一个大素数q及椭圆曲线参数a,b，由此定义点的椭圆群$E_q(a,b)$。其次，在$E_q(a,b)$中挑选基点$G=(x_1,y_1)$，G的阶为一个非常大的数n。n为使得$nG=O$成立的最小正整数。$E_q(a,b)$和G是该密码体制中通信各方均已知的参数。

用户 A 和用户 B 之间完成密钥交换过程如下：

(1) A 选择一个小于n的整数n_A作为其私钥，然后产生其公钥$P_A=n_A\times G$，该公钥是$E_q(a,b)$中的一个点。

(2) B 可类似地选择私钥n_B并计算对应公钥P_B。

(3) A 产生秘密钥$K=n_A\times P_B$，B 产生秘密钥$K=n_B\times P_A$。

要破译这种体制，攻击者必须由G和kG计算k，这个问题是椭圆曲线上的离散对数问题，被认为是计算上不可行的。

2.2 算法实现的伪代码

DH 密钥生成算法 产生 DH 算法的公私钥对

输入：椭圆曲线生成元G，生成元的阶n
输出：公钥PU，私钥PR

```
function genKey(G, n)
    随机选择1到n − 1之间的整数d
    PU ← {d * G}
    PR ← {d}
    return PU, PR
end function
```

DH 密文生成算法 使用对方公钥和私钥产生目标秘密钥

输入：公钥PU_b，私钥PR_a
输出：秘密钥K
function $genMsg(PU_b, PR_a)$
$K \leftarrow PU_b * PR_a$
return K
end function

2.3 测试样例与运行结果

我们选择的椭圆群是$E_{211}(0,-4)$，生成元为$G=(2,2)$，阶$n=240$。利用密钥生成算法得到 A 的公私钥对$\{PU_A, PR_A\}=\{(115,48),121\}$，B 的公私钥对$\{PU_B, PR_B\}=\{(130,203),203\}$，A 和 B 各自使用对方的公钥和自己的私钥产生的秘密钥为$K=(161,69)$。

3 ElGamal 公钥密码算法

3.1 算法原理

ElGamal 是一种基于离散对数的公钥密码体制，全局参数为素数q及椭圆曲线参数a,b，以及$E_q(a,b)$中的基点$G=(x_1,y_1)$。用户 A 生成的密钥对如下：

(1) 随机选择整数X_A，使得$1<X_A<n-1$。

(2) 计算$Y_A=X_A\times G$

(3) A 的私钥为X_A，公钥为Y_A。

其他用户 B 通过 A 的公开密钥可以加密信息：

(1) 将信息表示为椭圆曲线上一个点M。

(2) 选择任意整数k，使得$1<k<n-1$。

(3) 计算一次密钥$K=k\times Y_A$，验证$K\neq O$，否则重新挑选k

(4) 将M加密为明文对(C_1,C_2)，其中

$$C_1=k\times G \quad C_2=M+K$$

用户 A 恢复明文：

(1) 通过计算$K=C_1\times X_A$恢复密钥

(2) 计算$M=C_2-K$

ElGamal 的安全性基于计算离散对数的困难性之上，而椭圆曲线离散对数问

题已被证明是困难问题，所以攻击是计算上不可行的。

3.2 算法实现的伪代码

ElGamal 密钥生成算法 产生 ElGamal 算法的公私钥对

输入： 椭圆曲线生成元G，生成元的阶n

输出： 公钥PU，私钥PR

function $genKey(G, n)$

随机选择1到$n-1$之间的整数d

$PU \leftarrow \{d * G\}$

$PR \leftarrow \{d\}$

return PU, PR

end function

ElGamal 加密算法 使用公钥产生密文

输入： 明文M，公钥PU，椭圆曲线生成元G，生成元的阶n

输出： 密文C

function $encrypt(M, PU, G, n)$

随机选择1到$n-1$之间的整数k

$X \leftarrow k * PU$

$C \leftarrow \{k * G, M + X\}$

return C

end function

ElGamal 解密算法 使用私钥产生明文

输入： 密文C，私钥PR

输出： 明文M

function $decrypt(C, PR)$

$X \leftarrow PR * C_1$

$M \leftarrow C_2 - X$

return M

end function

3.3 测试样例及运行结果

我们选择的椭圆群是$E_{211}(0,-4)$，生成元为$G = (2,2)$，阶$n = 240$。利用密钥生成算法得到 A 的公私钥对$\{PU, PR\} = \{(111,66), 232\}$，B 利用 A 的公钥对消息$M = (129,56)$进行加密得到密文$C = \{(58,12), (146,84)\}$，A 利用自己的私钥对密文解密得到明文$M = (129,56)$。

4 SM2 公钥密码算法

4.1 算法原理

SM2 公钥密码算法是我国国家密码管理局提出的一种基于椭圆曲线离散对数问题的公钥密码体制，相比 RSA 算法更有优势。SM2 算法标准制定了椭圆曲线的参数，确定了唯一的标准曲线。其加密算法支持 128GB 的数据长度，加密强度为 256 位。通过引入随机预言机，使得相同明文产生的密文是不唯一的，从而可以抵抗选择明文攻击。该椭圆曲线公钥加密算法还涉及到三类辅助函数：密码杂凑算法，密钥派生函数和随机数发生器，这三类辅助函数的强弱会直接影响加密算法的安全性。在本次实验中选择 SHA-512 作为密码杂凑算法，以及 Python 内置的伪随机数发生器，并且实现了基于该杂凑算法的密钥派生函数。

4.2 算法实现的伪代码

KDF 密钥派生函数 产生与输入长度相同的散列值

输入： 比特串Z，输出长度$klen$，密码杂凑算法H_v，杂凑长度v
输出： 密钥数据比特串K

function $KDF(Z, klen, H_v, v)$
    令T为空比特串
    初始化一个 32 比特计数器$ct \leftarrow 0x00000001$
    for $i \leftarrow 1$ to $\lceil klen/v \rceil$ **do**
        $K \leftarrow K||H_v(Z||ct)$
        $ct \leftarrow ct + 1$
    end for
    return $K[0:klen]$
end function

SM2 密钥生成算法 产生 SM2 算法的公私钥对

输入： 椭圆曲线生成元g，生成元的阶n
输出： 公钥PU，私钥PR

function $genKey(g, n)$
    随机选择1到$n-2$之间的整数d
    $PU \leftarrow \{d * g\}$
    $PR \leftarrow \{d\}$
    return PU, PR
end function

SM2 加密算法 利用 SM2 算法进行加密

输入： 明文M，公钥PU，椭圆曲线生成元G，生成元的阶n

输出： 密文C

function $sm2\_encrypt(M, PU, G, n)$

令$klen$为M的比特长度

用随机数发生器产生随机数$k \in [1, n-1]$

$C_1 \leftarrow k * G$，并将C_1转换为比特串

$(x_2, y_2) \leftarrow k * PU$，并将$x_2, y_2$转换为比特串

$t \leftarrow KDF(x_2 || y_2, klen)$

$C_2 \leftarrow M \oplus t$

$C_3 \leftarrow Hash(x_2 || M || y_2)$

$C \leftarrow C_1 || C_3 || C_2$

return C

end function

SM2 解密算法 利用 SM2 算法进行解密

输入： 密文C，私钥PR

输出： 明文M

function $sm2\_decrypt(C, PR)$

$C_1, C_3, C_2 \leftarrow C$

将C_1转换为椭圆曲线上的点，并验证C_1是否满足椭圆曲线方程

$(x_2, y_2) \leftarrow PR * C_1$，并将$x_2, y_2$转换为比特串

$t \leftarrow KDF(x_2 || y_2, klen)$

$M' \leftarrow C_2 \oplus t$

$u \leftarrow Hash(x_2 || M' || y_2)$，验证$u = C_3$

return M'

end function

4.3 测试样例及运行结果

首先我们通过尝试对文本文件的加密和解密，验证了算法的正确性，程序输入结果如图 1 所示。

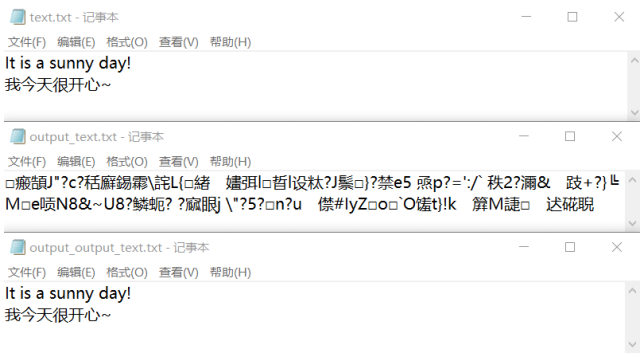

图 1 SM2 文件加解密验证

接着，为了分析加解密速度，我们通过分别对不同大小的文件进行加解密，观察运行时间的变化趋势来分析算法的效率，结果如表 2 所示。

表 2 算法执行时间（单位：秒）

文件大小	加密	解密
100k	0.131	0.104
200k	0.238	0.238
300k	0.393	0.426
400k	0.612	0.655
500k	0.889	0.942
600k	1.204	1.304
初始化时间		0.044

利用 Python 的 matplotlib 模块绘制数据曲线，结果如图 2 所示。

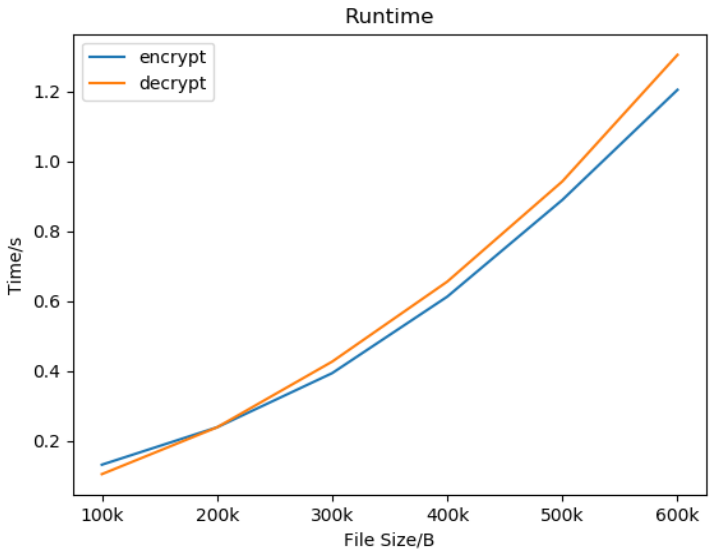

图 2 算法运行时间对比

可以看出随着文件大小的增加，算法运行时间的增长速度逐渐加快，并且解密时间平均要比加密时间更长，这也是 SM2 算法的一个性质。

5 总结

ECC 作为即将取代 RSA 的下一代公钥加密体制，其算法安全性和效率都比 RSA 体制更优，在通过对 ECC 的算法实现之后，我们更进一步地了解了椭圆曲线密码的原理，以及椭圆曲线上元素进行运算的方法，并且通过构造椭圆曲线上的加法循环群实现了 Diffie-Hellman 密钥交换协议和 ElGamal 公钥密码体制，验证了算法的正确性。然而这两种算法的通用性不是很强，只能将椭圆曲线上的点作为输入或输出。

在实现国密 SM2 算法的时候，接触了一些将椭圆曲线上的点转化为字节串的方法，这种方法使得该算法能够接受任意的字节串作为输入并输出字节串，适用于各种类型的文件，通用性较强。SM2 规定了唯一的标准椭圆曲线，保证了加密体制的安全性，在我们的测试中，发现 SM2 在同等安全性下比 RSA 快了近百倍，而且支持大文件的加密，着实是一个先进的加密算法标准，并且 SM2 利用随机预言机的思想也是十分值得学习的。

原本我还打算对 SM9 算法做相应的实现，但是在编程中遇到了许多困难，由于 Python 没有密码学有限域数学运算的相关库，只能从头开始编写程序，目前只做到了双线性对部分，在之后可能会继续尝试之后的部分，本次的代码中暂时不包含此算法的实现。