密码学实验报告 8

张天辰 17377321

2019 年 5 月 23 日

1 ECC 实现 Diffie-Hellman 密钥交换

1.1 ECC 上相关计算及其实现

完成所有 ECC 上密码算法的前提是实现 ECC 上的基本运算，包括点的加减法以及点与数的数乘运算。

椭圆曲线 $E(F_q)$ 上点的加法有如下规则：

1 $P + O = P$。这条基本不会用到。

2 $P + -P = O$，其中若 $P = (x, y)$，则 $-P = (x, -y)$。这条规则使得 $P - Q = P + (-Q)$，因此可以通过加法直接实现减法。

3 一般地，设 $P + Q = R$，则

$$R_x = \lambda^2 - P_x - Q_x \mod q$$

$$R_y = \lambda(P_x - R_x) - P_y \mod q$$

其中

$$\lambda = \begin{cases} \dfrac{3P_x^2 + a}{2P_y} \mod q \\ \dfrac{Q_y - P_y}{Q_x - P_x} \mod q \end{cases}$$

在 Python 语言中，这一步操作可以重载运算符。算法实现如下：

Algorithm 1 ECC 加减法

```
1: function ADD(P, Q)
2:     if P == Q then
3:         λ ← ((3 * P_x^2 + a)*REVERSE(2 * P_y))%q
4:     else
5:         λ ← ((Q_y − P_y)*REVERSE(Q_x − P_x))%q
6:     end if
7:     x = (λ * λ − P_x − Q_x)%q
8:     y = (λ * (P_x − x) − P_y)%q
```

    return (x, y)
end function
function SUB(P, Q)
    return ADD$(P, (Q_x, -Q_y))$
end function

而在椭圆曲线上的数乘运算 $Q = nP$ 实际上就是 n 个 P 相加，但是如果单纯地用循环的方法相加，显然效率太低。从上述的 P, Q 中恢复 n 的问题称为椭圆曲线上的“离散对数问题”，这不由得让人想到有一些“幂”运算的特征。事实上，这里的算法确实可以采用类似快速幂算法的方式，即将 n 写成二进制形式，然后根据其每一位为 1 或 0 进行操作。如果是 1 就将当前结果数乘 2（自己加自己），然后再和 P 相加；如果是 0 就仅仅数乘 2。这样就大量简化了运算。

算法实现如下：

Algorithm 2 ECC 数乘

function MULTI(P, n)
    $b \leftarrow n$ 转化为二进制后从第二位开始的比特串
    $result \leftarrow P$
    for each $i \in b$ **do**
        if $i == 1$ **then**
            $result = result + result$
            $result = result + P$
        else
            $result = result + result$
        end if
    end for
    return $result$
end function

1.2 ECCDH 密钥交换协议及其实现

ECCDH 交换协议流程如下：A 和 B 共享一条椭圆曲线及其生成元 G 以及 G 的阶 n。A 和 B 各自从 $[1, n-1]$ 中选取随机数 n_A, n_B 作为私钥，然后 A 发送给 B$P_A = n_A \times G$，B 发送给 A$P_B = n_B \times G$。基于等式 $n_A \times P_B = n_B \times P_A = K$，双方可以共享密钥 K。想破解这个协议就要解决椭圆曲线上的离散对数问题，这被认为是困难的。

算法实现如下：

Algorithm 3 ECCDH

function GENERATEKEY
    $private \leftarrow rand(1, n-1)$
    $public \leftarrow$MULTI$(G, private)$
end function
function CALCKEY(P_B)

$sharedKey \leftarrow$ MULTI$(P_B, private)$
end function

2 ECC 实现 ElGamal 密码体制

2.1 算法原理

变量名延续上节的定义。用户私钥 n_A 为 $[1, n-1]$ 中的一个数，公钥 P_A 为 $n_A \times G$。设需要加密的信息可以写为椭圆曲线上的点 P，则加密时首先生成随机数 $k \in [1, n-1]$，然后令 $C_1 = k \times G$，$C_2 = P + k \times P_A$。C_1, C_2 为密文。其原理就是先利用 k 和 P_A 掩盖 M，再利用 C_1 掩盖 k。想要去除掩盖，除非拥有私钥，否则必须解椭圆曲线上离散对数问题。解密时，根据等式 $k \times P_A = n_A \times C_1$，就可以利用 $P = C_2 - n_A \times C_1$ 得到明文。

2.2 算法实现

1. 密钥生成参考 DH 协议里的密钥生成方法，即算法 3中的函数 GENERATEKEY。

2. ElGamal 加解密

Algorithm 4 ElGamal

function ENCRYPT(P)
  $k \leftarrow rand(1, n-1)$
  return (MULTI(G, k), $P+$MULTI(P_A, k))
end function
function DECRYPT(C_1, C_2)
  return C_2-MULTI(C_1, n_A)
end function

2.3 算法测试

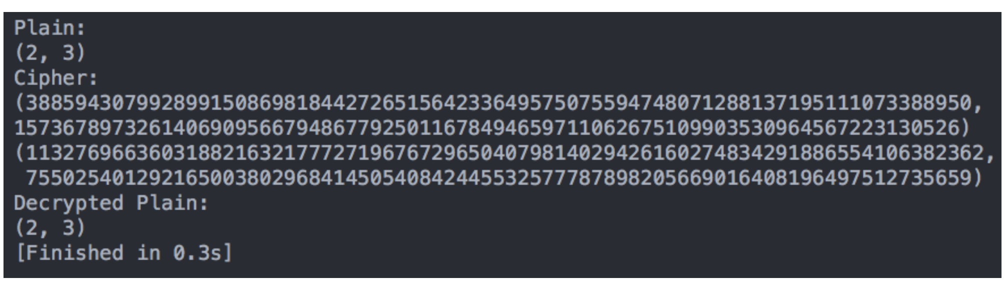

图 1: ElGamal 测试

3 国密 SM2 算法

3.1 SM2 算法原理

SM2 是对简单椭圆曲线密码体制的改进。其密钥生成和前述的 ElGamal 体制相同，但加解密过程就相对复杂一些。**尽管算法标准中将很多操作规定为在比特串上进行，但是考虑到计算机读写文件都以字节为单位，而且使用字节串并不影响其本身的逻辑，还更加节省空间。因此这里都采用字节串进行实现。比特串的应用场景应该是硬件实现，而不是软件。**

沿用之前的符号定义，如果要加密字节串 M，长度为 $klen$，则要先生成随机数 $k \in [1, n-1]$，并且令 $C_1 = k \times G$ 转换为字节串后的结果。此后令 $(x2, y2) = k \times P_A$，并将 $x2, y2$ 转化为字节串后拼接在一起为 $x2||y2$。令 $T = KDF(x2||y2, 8 * klen)$，其中 KDF 为密钥派生函数，会在稍后介绍。如果 T 全为 0，就重新选 k。此后令 $C_2 = M \oplus T$。再令 $C_3 = Hash(x2||M||y2)$，这里的 Hash 函数论理应当是 SM3 等国密算法，但这里用 sha-256 代替。

解密时，根据等式 $k \times P_A = n_A \times C_1$，就可以利用私钥恢复 $(x2, y2)$，从而恢复 T，则有 $M = T \oplus C_2$。C_3 的 Hash 函数值用于验证完整性。

接下来介绍 KDF 函数，其输入本为一个整数，但在函数里也要转成比特串，于是本方案考虑直接输入字节串，减少无谓的类型转换。此外还要输入一个预定的输出位数。其函数逻辑为，定义一个 32 位计数器 ct，每次将其拼接在输入字节串 Z 的后面，然后送入 Hash 函数，并拼接在以前的输出之后，此后 ct 自增 1。这样的操作持续到位数足够，并截断最后的位数以凑整输入的比特长度。

3.2 SM2 算法实现

密钥生成参考 DH 协议里的密钥生成方法，即算法 3中的函数 GENERATEKEY。

Algorithm 5 SM2

```
function KDF(Z, klen)
    因为采用 sha-256，因此 v ← 256
    ct ← 1
    for each i ∈ [1, klen/v] do
        hashList ← hashList||Hash(Z||ct)
        ct ← ct + 1
    end for
    从右边截断 hashList 若干位，直到它长度为 klen 位。
    return hashList
end function
function ENCRYPT(plainList)
    klen ←LEN(plainList)
    k ←RANDINT([1, n − 1])
    c1 ←MULTI(G, k)
    (x2, y2) ←MULTI(public, k)
    t ←KDF(x2||y2, 8 ∗ klen)
```

    如果 $t == 0$，则返回 12
    $c2 \leftarrow plainList \oplus t$
    $c3 \leftarrow$ HASH$(x2||plainLlist||y2)$
    return $c1||c3||c2$
end function
function DECRYPT$(cipherList)$
    $cipherList$ 前 65 字节为 $c1$，中间 32 字节为 $c3$，其余为 $c2$
    $klen \leftarrow$ LEN$(c2)$
    $(x2, y2) \leftarrow$ MULTI$(private, c1)$
    $t \leftarrow$ KDF$(x2||y2, 8 * klen)$
    $m \leftarrow c2 \oplus t$
    $u \leftarrow$ HASH$(x2||m||y2)$
    if $u \neq c3$ **then**
        报错
    end if
    return m
end function

3.3 算法测试

算法明文文件默认为主程序同一文件夹下的 Plain.txt，密文文件默认为同一文件夹下的 Cipher.txt。在程序开始时会询问是否生成密钥，如果是则生成一对公私钥，否则沿用已有的（如果有）。如图 2，尽管我们无法从程序输出看到具体运行信息，但我们还是可以看到，至少解密得到的文件通过了 Hash 检验。[htbp]

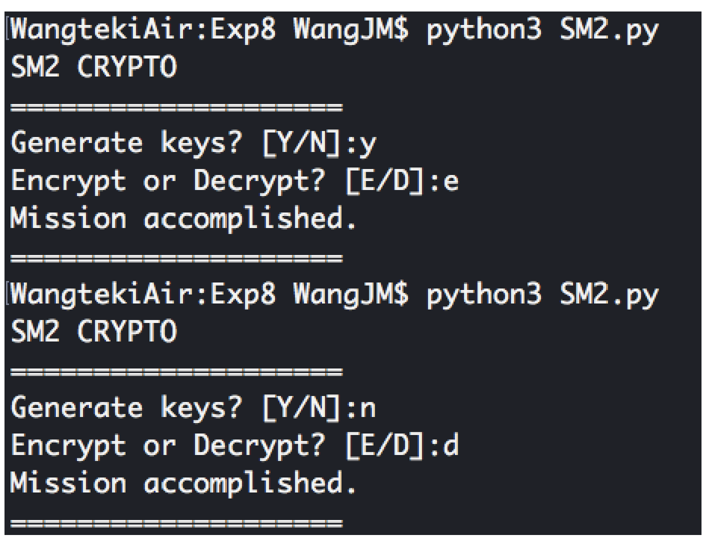

图 2: SM2 程序运行图

然后我们再更改 Plain.txt 的后缀为.jpg，可以看到这实际上是一张图片，解密成功。

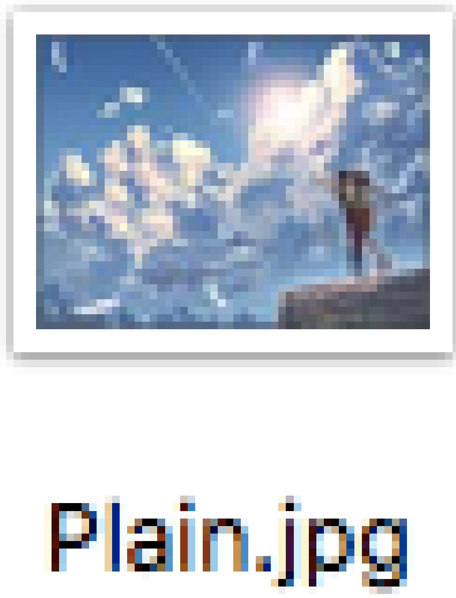

图 3: 解密后明文

4 感想

本次实验我对椭圆曲线上的加解密算法进行了实现。实际上，非对称加密可能比起对称算法要好实现一些，毕竟加解密方式都比较干脆，不像对称密码那样操作很多。在实现 SM2 的过程中我也尝试进行了一些优化，比如通过更改数据类型提高速度、更换更快的函数等等。

在实现了 SM3 后，我会将这里的 Hash 函数换成 SM3，实现技术独立（笑）。