密码学实验报告 10

张天辰 17377321

2019 年 6 月 5 日

1 SM2 数字签名算法

1.1 算法原理

SM2 算法利用椭圆曲线构造数字签名。以下流程不考虑边界情况。首先根据椭圆曲线参数、用户的公钥 P_A 和用户的个人身份构造身份信息 Z_A。将 Z_A 和消息 M 组合到一起，再经过 Hash 函数得到 e。此后产生随机数 k，并计算 $(x_1, y_1) = kG$。令 $r = (e + x_1) \mod n$，$s = \frac{k - rd_A}{1 + d_A} \mod n$。签名即为 (r, s)。

签名的验证方法如下。由生成过程可知，重要的在于恢复 k。首先用与生成签名相同的方法可以得到 e，此后令 $t = r + s$，则有：

$$t = r + s = \frac{k - rd_A}{1 + d_A} + r = \frac{k + r}{1 + d_A}$$

$$s + td_A = \frac{kd_A + rd_A + k - rd_A}{1 + d_A} = k$$

因此可以如上恢复出 k。从而可以恢复 x_1，并计算 $R = (e + x_1) \mod n$ 与收到的 r 进行验证。如果相等则验证成功。

1.2 算法实现

关于椭圆曲线上的运算以及类型转换等算法可以直接复用先前实现 SM2 公钥算法时的代码。

1. 签名生成算法

```
def sign(message):
    za = sm3(entl || id || a || b || g.x || g.y || public.x || public.y)
    m1 = za || message
    e = sm3(m1)
    while True:
        k = randint(1, n - 1)
        temp = g.multi(k)
        r = (e + temp.x) mod n
        if r == 0 or r + k == n:
            continue
        s = (reverse(1 + private, n) * (k - r * private)) mod n
        if s != 0:
```

```
            break
        return (r, s)
```

2. 签名验证算法

```
    def authenticate(message, signature):
        r = signature[0]
        s = signature[1]
        if r < 1 or r >= n or s < 1 or s >= n:
            return False
        za = sm3(entl || id || a || b || g.x || g.y || public.x || public.y)
        m1 = za || message
        e = sm3(m1)
        t = (r + s) mod n
        if t == 0:
            return False
        temp = g.multi(s) + public.multi(t)
        R = (e + temp.x) mod n
        if R == r:
            return True
        else:
            return False
```

1.3 算法测试

这里采用 SM2 标准中的测试数据进行检验。用户 ID 为 ALICE123@YAHOO.COM，发送的消息为 message digest，椭圆曲线参数、公私钥等参数比较复杂，略去。对这组数据，标准的结果为：

r = 0x40F1EC59F793D9F49E09DCEF49130D4194F79FB1EED2CAA55BACDB49C4E755D1

s = 0x6FC6DAC32C5D5CF10C77DFB20F7C2EB667A457872FB09EC56327A67EC7DEEBE7

如图 1所示，使用上述内容进行数字签名，得到的结果与标准相同。标准的结果按照十六进制数的形式给出，但根据标准的算法流程，这里的签名应该是字节串，因此本实现按照字节串输出，不难验证两者的相同性。此后，使用上述签名进行验证（图 1代码第 88 行），得到了认证通过的结果；使用另外的签名进行认证（图 1中代码第 89 行），则认证不通过。

```
81 if __name__ == '__main__':
82
83     sign = SM2Sign("ALICE123@YAHOO.COM")
84     message = b'message digest'
85     sign.generateKey()
86     signature = sign.sign(message)
87     print(signature)
88     sign.authenticate(message, signature)
89     sign.authenticate(message, (b'test', b'test'))
90

(b'@\xf1\xecY\xf7\x93\xd9\xf4\x9e\t\xdc\xefI\x13\rA\x94\xf7\x9f\xb1\xee\xd2\xca\xa5[\xac\xdbI\xc4\xe7U\xd1',
b"o\xc6\xda\xc3,]\\\xf1\x0cw\xdf\xb2\x0f|.\xb6g\xa4W\x87/\xb0\x9e\xc5c'\xa6~\xc7\xde\xeb\xe7")
Authentication passed.
Authentication failed.
[Finished in 0.3s]
```

图 1: SM2 数字签名算法测试

2 感想

这次实验相对而言比较容易。我认为在初级、中级、高级要求中选择一个进行实现是很合理的制度，因为对于本次实验而言，不同的要求只是实现的数字签名算法不太一样而已，而不管实现哪种，都是按照标准一步一步实现。只要实现了一种，就能加深对数字签名的理解。