密码学实验报告 实验十

2019 年 6 月 6 日

1 DSA 数字签名算法

1.1 算法原理

NIST 发布的 FIPS 186 标准中规定了数字签名标准（DSS），包含基于离散对数、RSA 和椭圆曲线的数字签名算法，使用了安全 Hash 算法（SHA），本节我们介绍和实现基于离散对数的数字签名算法（DSA）。

DSA 建立在离散对数问题的困难性以及 ElGamal 和 Schnorr 最初提出的数字签名方案之上，算法包含三个公开参数，这些参数被一组用户所共有。选择一个N位的素数q，然后选择一个长度在$512 \sim 1024$之间且满足q能整除$p-1$的素数p，最后选择形为$h^{(p-1)/q} \mod p$的g，其中h是1到$p-1$之间的整数使得$g>1$。DSA 的公开参数的选择与 Schnorr 签名方案完全一样。

选定公开参数后，每个用户选择私钥并产生公钥。私钥x必须是随机或伪随机选择的、位于1到$q-1$之间的数，由$y=g^x \mod p$计算出公钥。由给定的x计算y比较简单，而由给定的y确定x则在计算上不可行。

对消息进行签名时，用户需要计算两个量r和s。r和s是参数p,q,g、发送方私钥x、消息的 Hash 码$H(M)$和附加整数k的函数，其中k是随机或伪随机产生的，且k对每次签名是唯一的。这两个量作为消息的签名与消息一同发送。

令M',r',s'是接收方收到的M,r,s，接收方计算值v，它是参数p,q,g、发送方公钥y、接收到的消息的 Hash 码的函数。若v与签名中的r相同，则签名是有效的。由于求离散对数的困难性，攻击者要想从r恢复出k或从s中恢复出x都是不可行的。

根据 FIPS 186，我们使用一种基于 Hash 函数的方法生成素数p,q用于提供 DSA 的合法参数，该算法可以产生任意长的素数对以满足算法的安全需求。

1.2 算法实现的伪代码

参数生成算法 生成 DSA 的合法参数

输入：素数p的期望长度L，素数q的期望长度N，种子长度$seedlen$
输出：长度为L的素数p和长度为N的素数q

function $genParam(L, N, seedlen)$
  检查(L, N)是否符合标准
  $n \leftarrow \lceil L/outlen \rceil - 1$
  $b \leftarrow L - 1 - (n * outlen)$
  生成一个长度为$seedlen$的随机比特串$seed$
  $U \leftarrow Hash(seed) \bmod 2^{N-1}$
  $q \leftarrow 2^{N-1} + U + 1 - (U \bmod 2)$
  检验q是否为素数，否则重新生成$seed$
  $offset \leftarrow 1$
  for $counter \leftarrow 0$ to $(4L - 1)$ **do**
    for $j \leftarrow 0$ to n **do**
      $V_j = Hash\left((seed + offset + j) \bmod 2^{seedlen}\right)$
    end for
    $W \leftarrow V_0 + (V_1 * 2^{outlen}) + \cdots + \left(V_{n-1} * 2^{(n-1)*outlen}\right) + \left((V_n \ mod\ 2^b) * 2^{n*outlen}\right)$
    $X \leftarrow W + 2^{L-1}$
    $c \leftarrow X \bmod 2q$
    $p \leftarrow X - (c - 1)$
    if $p \geq 2^{L-1}$ **and** p是素数 **then**
      return p, q
    end if
    $offset \leftarrow offset + n + 1$
  end for
  重新生成$seed$
end function

密钥生成算法 生成 DSA 的公私钥对

输入：无
输出：私钥x和公钥y

function $genKey()$
  产生随机数x使$0 < x < q$
  计算$y \leftarrow g^x \bmod p$
  return x, y
end function

签名算法 计算消息的签名

输入：消息M，私钥x

输出：消息签名(r, s)

function $sign(M, x)$

  生成1到$q-1$的随机数k

  $r \leftarrow (g^k \mod p) \mod q$

  $s \leftarrow [k^{-1}(H(M) + xr)] \mod q$

  return (r, s)

end function

验证算法 验证消息的签名

输入：消息M'，签名(r', s')，公钥y

输出：签名是否合法

function $verify(M', (r', s'), y)$

  $w \leftarrow (s')^{-1} \mod q$

  $u_1 \leftarrow [H(M') * w] \mod q$

  $u_2 \leftarrow (r' * w) \mod q$

  $v \leftarrow [(g^{u_1} * y^{u_2}) \mod p] \mod q$

  if $v = r'$ **then**

    return 签名合法

  else

    return 签名不合法

  end if

end function

1.3 测试样例及运行结果

首先使用参数生成算法得到 DSA 的三个参数如表 1 所示。

表 1 DSA 全局参数

变量	十六进制值	位数
p	80000000000000006BE46285E19FEA4642DD7E308912A 091CC043CBB95F448A51839D9FED70C611FE0E8B756 467971ADE7B2BB177994A48A74A432B5E5C5BA95A7 407FEF2C07E283E19E3E2D64742F9CD91381A225DA5 88B46FFDAB7CB246A55205A04AAFE3B7F1FE80D70E 8D56CC3785F2FE168A45FF6E452881F1CE72D26F7	1024
q	95B69E0A96A4FA811FC6681360F653A8B135646D	160
g	63D75D5062F30A1E0AF58B01C3E7AEA8293560E7FA2 7F9DEB506F4E08468450D7EC1EDF2091CD9E0C83675 F453642AC402E6E5DAD026AF21FBB461AAD1765D18 57CF30CE92A0F54FAAF16EFE35AD64518E549359F3A A2E46722BA2E5C4340C6AF30E2F73A4F91EB8A50B9 B87D2D1D47B8F3690121A4B37077410C1311A7B7594	1024

接着计算消息“message”的签名，字符串使用 ASCII 编码，得到的签名为

$$\begin{cases} r = (59BA5C7A2EF1E562D158610E28DB4017800EB5CF)_{16} \\ s = (8634A499475C889CB2482CC81F098206CF50C4C1)_{16} \end{cases}$$

将消息和签名一起输入验证算法，输出结果为签名合法，证明了 DSA 算法是正确的。

2 ElGamal 数字签名方案

2.1 算法原理

ElGamal 数字签名方案与 ElGamal 公钥加密方案类似，只不过是用私钥进行加密，公钥进行解密的，该方案利用了数论中原根的性质：对于素数q，如果α是q的原根，则有$\alpha, \alpha^2, \cdots, \alpha^{q-1}$取模后各不相同。ElGamal 数字签名方案的参数的素数q和α，其中α是q的原根。方案在计算签名和验证签名时需要用到安全 Hash 算法（SHA）计算消息的 Hash 值$m = H(M)$，其中$0 \le m \le q-1$。

2.2 算法实现的伪代码

密钥生成算法 生成 ElGamal 数字签名方案的公私钥对

输入： 无

输出： 私钥X_A和公钥Y_A

function $genKey()$
  产生随机整数X_A，使得$1 < X_A < q-1$
  计算$Y_A \leftarrow \alpha^{X_A} \mod q$
  return X_A, Y_A
end function

签名算法 计算消息的签名

输入： 消息的 Hash 值m，私钥X_A

输出： 消息签名(S_1, S_2)

function $sign(m, X_A)$
  选择随机整数K，使得满足$1 \le K \le q-1$以及$\gcd(K, q-1) = 1$
  $S_1 \leftarrow \alpha^K \mod q$
  $S_2 \leftarrow [K^{-1}(m - X_A S_1)] \mod (q-1)$
  return (S_1, S_2)
end function

验证算法 验证消息的签名

输入：消息的 Hash 值m，签名(S_1, S_2)，公钥Y_A
输出：签名是否合法

function $verify(m, (S_1, S_2), Y_A)$
    $V_1 \leftarrow \alpha^m \bmod q$
    $V_2 \leftarrow \left(Y_A^{S_1} * S_1^{S_2}\right) \bmod q$
    if $V_1 = V_2$ **then**
        return 签名合法
    else
        return 签名不合法
    end if
end function

2.3 测试样例与运行结果

我们选择素数域$GF(19)$，即令$q = 19$，其原根为$\{2,3,10,13,14,15\}$，选择其中一个原根$\alpha = 10$，计算消息“message”的签名，字符串使用 ASCII 编码，得到的签名为$(13, 5)$，将消息和签名一起输入验证算法，输出结果为签名合法，证明算法正确。

3 Schnorr 数字签名方案

3.1 算法原理

Schnorr 数字签名方案也是基于离散对数的，其主要优势是可以将生成签名所需的消息计算量最小化，生成签名的主要步骤不依赖于消息，可以在处理器空闲时执行，生成与消息有关的部分只需要进行一个$2n$位的整数与n位的整数相乘。

该方案基于素数模p，且$p-1$包含大素数因子q，即$p-1 \equiv 0(mod\ \ q)$。一般取$p \approx 2^{1024}$和$q \approx 2^{160}$，即p为1024位整数，q为160位整数，也正好等于 SHA-1 中 Hash 值的长度。通常选择 SHA-1 位内部 Hash 算法。由于 DSA 公开参数的选择与 Schnorr 签名方案完全一样，我们可以直接使用 FIPS 186 标准中的参数生成算法。

3.2 算法实现的伪代码

密钥生成算法 生成 Schnorr 数字签名方案的公私钥对

输入：无

输出：私钥s和公钥v
function $genKey()$
产生随机整数s，使得$0 < s < q$
计算$v \leftarrow \alpha^{-s} \bmod p$
return s, v
end function

签名算法 计算消息的签名
输入：消息M，私钥s
输出：消息签名(e, y)
function $sign(M, s)$
选择随机整数r，且$0 < r < q$
$x \leftarrow \alpha^r \bmod p$
$e \leftarrow H(M||x)$
$y \leftarrow (r + se) \bmod q$
return (e, y)
end function

验证算法 验证消息的签名
输入：消息M，签名(e, y)，公钥v
输出：签名是否合法
function $verify(M, (e, y), v)$
$x' \leftarrow \alpha^y v^e \bmod p$
if $e = H(M||x')$ **then**
return 签名合法
else
return 签名不合法
end if
end function

3.3 测试样例及运行结果

选择 Schnorr 数字签名方案的公开参数如表 1 所示，计算消息“message”的签名，字符串使用 ASCII 编码，得到的签名为

$$\begin{cases} e = (FC1B4373C2FDC976998CE66322F477ADE0CFBAF5)_{16} \\ y = (92437A5474C58E2127B3F4D00EE383938014A185)_{16} \end{cases}$$

将消息和签名一起输入验证算法，输出结果为签名合法，证明了算法的正确性。

4 SM2 数字签名算法

4.1 算法原理

利用素域$GF(p)$或$GF(2^m)$域上的椭圆曲线都可以构成椭圆曲线数字签名方案，基于椭圆曲线的数字签名方案有着安全、密钥短、软硬件实现更优等特点，NIST 发布的数字签名标准（DSS）的较新版 FIPS 186-3 中已引入基于椭圆曲线的数字签名算法，我国也于 2012 年颁布了椭圆曲线数字签名标准 SM2。SM2 标准中规定了唯一的椭圆曲线，并给出了参数，在计算签名和验证签名时候需要用到安全 Hash 算法，本节中我们针对签名算法和验证算法进行了实现。

除了椭圆曲线的系统参数，我们还需要给出用户 A 的密钥对，包括私钥d_A和公钥$P_A = d_A * G = (x_A, y_A)$。以及签名者的一个长度为$entlen_A$比特的可辨别标识$\mathrm{ID_A}$，将整数$entlen_A$用两个字节表示，记作$ENTL_A$，首先要计算用户 A 的 Hash 值$Z_A$，其值为

$$Z_A = H_{256}(ENTL_A||ID_A||a||b||x_G||y_G||x_A||y_A)$$

4.2 算法实现的伪代码

签名算法 计算消息的签名

输入： 消息M，私钥d_A
输出： 消息签名(r, s)
function $sign(M, d_A)$
　　$\bar{M} \leftarrow Z_A||M$
　　$e \leftarrow H_v(\bar{M})$
　　产生随机数k，使$0 < k < n$
　　$(x_1, y_1) \leftarrow k * G$
　　$r \leftarrow (e + x_1) \bmod n$，若$r = 0$或$r + k = n$则重新产生$k$
　　$s \leftarrow [(1 + d_A)^{-1} * (k - r * d_A)] \bmod n$，若$s = 0$则重新产生$k$
　　return (r, s)
end function

验证算法 验证消息的签名

输入： 消息M'，签名(r', s')，公钥P_A
输出： 签名是否合法
function $verify(M', (r', s'), P_A)$
　　检验r', s'是否符合$0 < r', s' < n$
　　$\bar{M}' \leftarrow Z_A||M'$

$e' \leftarrow H_v(\bar{M}')$
$t \leftarrow (r'+s') \mod n$，检验$t \neq 0$
$(x_1', y_1') \leftarrow s' * G + t * P_A$
$R \leftarrow (e' + x_1') \mod n$
if $R = r'$ **then**
    return 签名合法
else
    return 签名不合法
end if
end function

4.3 测试样例及运行结果

令签名者的用户标识为“ID:A”，对消息“message”进行签名，字符串均采用 ASCII 编码，得到的签名为

$$\begin{cases} r = (F0FA8C0D00161592C1F87A5D8DAB5328FC69D2F9FD460A64FC48E3064B34FF75)_{16} \\ s = (131C2970518A5E79DE361D666EB5796377F1AD4D479B6E5540C9F506577724FE)_{16} \end{cases}$$

将消息和签名一起输入验证算法，输出结果为签名合法，证明了 SM2 数字签名算法是正确的。

同时我们还实现了对于文件的签名和验证算法，我们选择两个不同大小的图片文件计算签名，将签名结果保存为 sign 类型的文件，结果如图 1 所示。

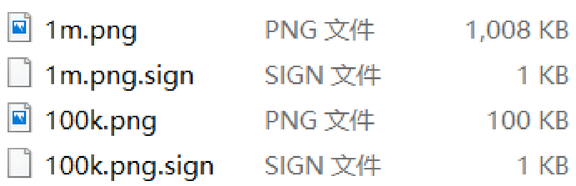

图 1 对文件的数字签名测试

可以看到两个图片的签名文件均为 64 字节，对存储空间以及传输的耗费极小，然后将两个图片以及对应的签名文件输入验证算法，得到的结果是签名合法，证明对文件的数字签名算法也是正确的。

5 总结

本次实验中我们实现了四种不同的数字签名方案，包括基于离散对数以及基于椭圆曲线密码体制的数字签名算法。其中 DSA 作为数字签名标准中给出的算法，内容十分完整，包括了参数的生成以及验证算法，其生成算法巧妙地利用了 Hash 函数来生成给定长度且符合要求的素数。DSA 的签名和验证算法与 Schnorr

数字签名方案类似，所以在之后实现 Schnorr 数字签名方案中的算法时，也可以借助之前实现 DSA 时的经验，轻松地将函数编写出来。

对于 ElGamal 数字签名方案，由于之前我们也实现过 ElGamal 公钥加密方案，所以这里我们只要将其公钥和密钥反用即可，原理基本类似，但是 ElGamal 数字签名方案需要用到原根，而我们目前的原根生成算法效率并不高，无法生成长度达到安全长度的数字的原根，所以我们只好选取了较小的数字作为测试。

SM2 数字签名方案是不同于之前三种的，基于椭圆曲线密码学的数字签名。SM2 的签名计算过程中消息通过一个变量参与到了计算中，只需要经过一次椭圆曲线上点的运算，速度较快，且有更好的安全性。由于 SM2 有着完整的标准，所以我们也能很方便地实现对文件的签名和验证算法，并且也利用文件测试证明了我们算法的正确性。