

实验五 数字签名算法DSA

一、实验目的

通过对数字签字算法DSA的实际操作，理解DSS的基本工作原理。

二、实验原理

以往的文件或书信可以通过手写亲笔签名来证明其真实性，而通过计算机网络传输的信息则通过数字签字技术实现其真实性的验证。

数字签字目前采用较多的是非对称加密技术，其实现原理简单的说，就是由发送方利用杂凑函数对要传送的信息进行计算得到一个固定位数的消息摘要值，用发送者的私钥加密此消息的杂凑值所产生的密文即数字签字。然后将数字签字和消息一同发送给接收方。接收方收到消息和数字签字后，用同样的杂凑函数对消息进行计算得到新的杂凑值，然后用发送者的公开密钥对数字签字解密，将解密后的结果与自己计算得到的杂凑值相比较，如相等则说明消息确实来自发送方。

三、DSA数字签名算法描述

1. 全局公开钥

p : 满足 $2^{L-1} < p < 2^L$ 的大素数，其中 $512 \leq L \leq 1024$ 且 L 是64的倍数。

q : $p-1$ 的素因子，长为160比特。

g : $g \equiv h^{(p-1)/q} \pmod p$, 其中 h 是满足 $1 < h < p-1$ 且使得 $h^{(p-1)/q} \pmod p > 1$ 的任一整数

2. 私钥

私钥 x 是随机数或伪随机数，其中 $0 < x < q$

3. 公钥

$y = g^x \pmod p$, (p, q, g, y)为公钥。

4. 秘密数 k

k 为随机数或伪随机数，其中 $0 < k < q$

5. 签名过程

(γ, δ) 为对消息 m 的签字，签字长度为320比特

$$\gamma = (g^k \pmod p) \pmod q$$

$$\delta = (k^{-1}(H(m) + xr)) \pmod q$$

其中， H 是一个安全杂凑函数，在DSS标准中，采用SHA-1算法作为安全的杂凑函数。签字完成后，把对消息 m 的数字签字 (γ, δ) 和消息 m 一同发送给接收方。

6. 验证过程

接收方接收到消息 m 和数字签字接收方接收到消息 m 和数字签字 (γ, δ) 后, 对数字签字的验证过程如下后, 对数字签字的验证过程如下

$$w = \delta^{-1} \bmod q \leftarrow$$

$$u1 = H(m)w \bmod q \leftarrow$$

$$u2 = \gamma w \bmod q \leftarrow$$

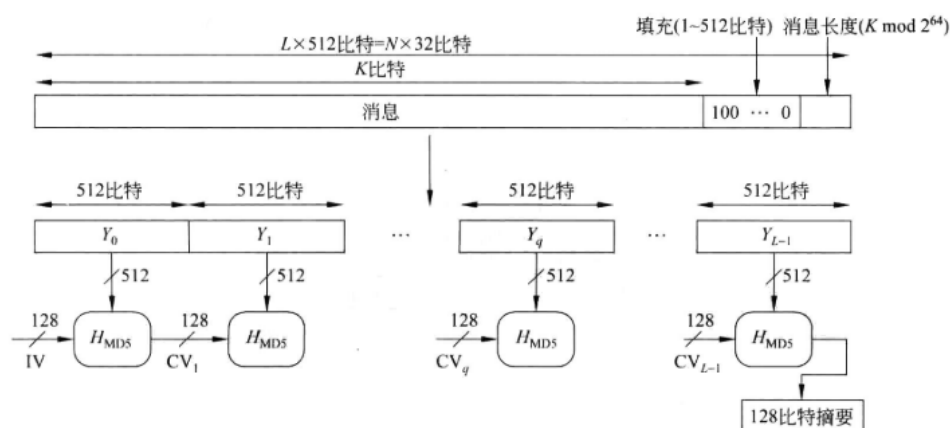
若有 $((g^{u1} y^{u2}) \bmod p) \bmod q = \gamma$, 则说明信息确实来自发送方。否则, 签字是无效的。

四、SHA杂凑函数算法描述

算法的输入为小于 2^{64} 比特长的任意消息, 分为512比特长的分组, 输出为160比特长的消息摘要。算法的框图与MD5的算法框图一样, 如下图, 但哈希值的长度和链接变量的长度为160比特。

(1)消息填充

与MD5的步骤(1)完全相同, 如下图



(2)附加消息长度

与MD5的步骤(2)类似, 不同之处在于以大端方式表示填充前消息的长度。即步骤(1)留出的64比特当作64比特长的无符号整数。

(3)缓冲区初始化

算法使用160比特长的缓冲区存储中间结果和最终哈希值, 缓冲区可表示为5个32比特长的寄存器(A, B, C, D, E), 每个寄存器都以大端方式存储数据, 其初始值分别为 $A=67452301$, $B=EFCDAB89$, $C=98BADCFB$, $D=10325476$, $E=C3D2E1F0$ 。

(4)分组处理

每一分组 Y_q 都经一压缩函数处理, 压缩函数由4轮处理过程(如图6-9所示)构成, 每一轮又由20步迭代组成。4轮处理过程结构一样, 但所用的基本逻辑函数不同, 分别表示为 f_1, f_2, f_3, f_4 。每轮的输入为当前处理的消息分组 Y 和缓冲区的当前值A、B、C、D、E, 输出仍放在缓冲区以替代A、B、C、D、E的旧值, 每轮处理过程还需加上一个加法常量 K , 其中 $0 \leq t \leq 79$ 表示迭代的步数。80个常量中实际上只有4个不同取值, 如表6-7所示, 其中 $[x]$ 为 x 的整数部分。第4轮的输出(即第80步迭代的输出)再与第1轮的输入 CV_q 相加, 以产生 CV_{q+1} 其中加法是缓冲区中5个字中的每一个字与 CV_q 中相应的字模2相加。

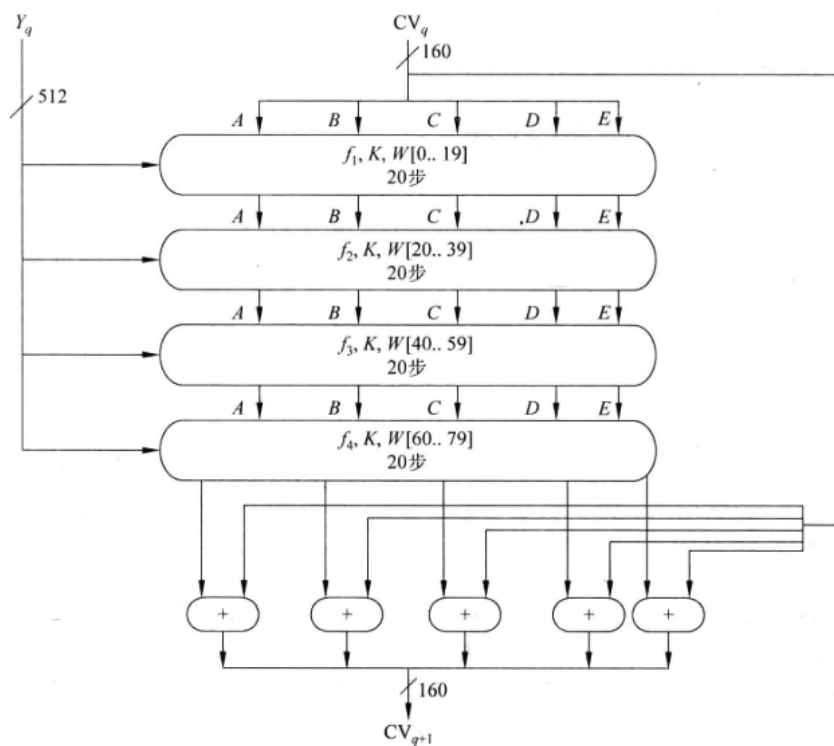


图 6-9 SHA 的分组处理框图

表 6-7 SHA 的加法常量

迭代步数 t	常量 K_t (十六进制)	K_t (十进制)	迭代步数 t	常量 K_t (十六进制)	K_t (十进制)
$0 \leq t \leq 19$	5A827999	$\lfloor 2^{30} \times \sqrt{2} \rfloor$	$40 \leq t \leq 59$	8F1BBCDC	$\lfloor 2^{30} \times \sqrt{5} \rfloor$
$20 \leq t \leq 39$	6ED9EBA1	$\lfloor 2^{30} \times \sqrt{3} \rfloor$	$60 \leq t \leq 79$	CA62C1D6	$\lfloor 2^{30} \times \sqrt{10} \rfloor$

(5)输出

消息的L个分组都被处理完后,最后一个分组的输出即为160比特的消息摘要。

步骤(3)到(5)可总结如下

$$CV_0 = IV;$$

$$CV_{q+1} = \text{SUM}_{32}(CV_q, ABCDE_q);$$

$$MD = CV_L$$

其中四轮压缩函数的过程如下图。

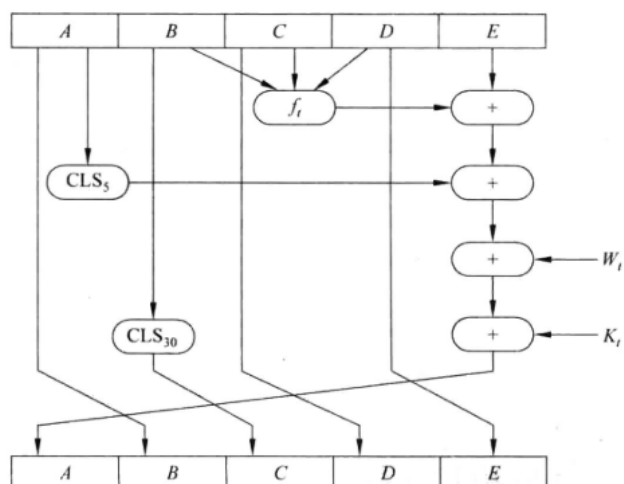



图 6-10 SHA 的压缩函数中一步迭代示意图

五、演示

下面是使用演示程序生成并验证的整个过程。

(1) 生成密钥过程



v1.3

Size of P (Bits)

512

Number Base

16

Random seed generation

Start

572a7472

100%

P (public) = Prime in range 512-1024 Bits - 64 Bit

ABBBFEDDF209B37331966E0E8F766742A91A6D47AB8FD9066BA7234ECBAED0C6CCBB9D14A73222D0EABCB78CB40FD1D598093AA597AB07C7EA94077007F9F07F

Q (public) = 160 Bit prime factor of P-1

A5F0222C58C4F102AC1DEFD158453A690FFE931B

G (public) = $H^{(P-1)/Q}$ with $H < (P-1)$ and $H^{(p-1)/Q} \text{ MOD } P >$

23BDD52AD4EEB59FC3F13816D91E93BAAD366C47A95A8F0797A534B05F12CC38B7A59930696525D7D5BF7EC8626239D621D629C7EC0DF356D629563957D14815

Y (public) = $G^X \text{ MOD}$

792FCB5099EEB6D42E3657172BA793805B731A050E10615A220B1F1054DAF84FBB9A8BA537E3FA3ACA66B2C0AF5D08AA334642B6C9CE971CE777C4311C7BD35F

X (private) = 160 Bits and $< Q$

91EB1BAEAB99D666783D4019B74B0D5BFCF12851

Generate

Test

Info

Exit

Done. You can test your keys right now.

参数x

X (private) = 160 Bits and $< Q$
91EB1BAEAB99D666783D4019B74B0D5BFCF12851

全局公开钥 (P, Q, G, Y)

P

P (public) = Prime in range 512-1024 Bits - 64 Bit
ABBBFEDDF209B37331966E0E8F766742A91A6D47AB8FD9066BA7234ECBAED0C6CCBB9D14A73222D0EABCB78CB40FD1D598093AA597AB07C7EA94077007F9F07F

Q

Q (public) = 160 Bit prime factor of P-1
A5F0222C58C4F102AC1DEFD158453A690FFE931B

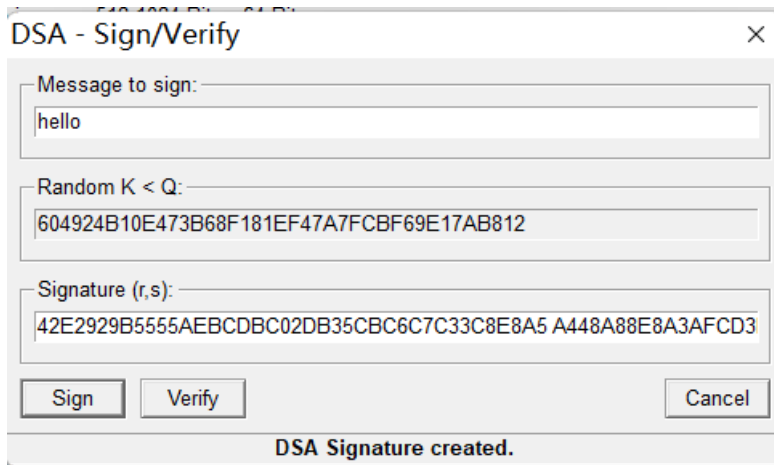
G

G (public) = $H^{(P-1)/Q}$ with $H < (P-1)$ and $H^{(p-1)/Q} \text{ MOD } P >$
23BDD52AD4EEB59FC3F13816D91E93BAAD366C47A95A8F0797A534B05F12CC38B7A59930696525D7D5BF7EC8626239D621D629C7EC0DF356D629563957D14815

计算的Y

Y (public) = $G^X \text{ MOD}$
792FCB5099EEB6D42E3657172BA793805B731A050E10615A220B1F1054DAF84FBB9A8BA537E3FA3ACA66B2C0AF5D08AA334642B6C9CE971CE777C4311C7BD35F

(2) 对消息签名



DSA - Sign/Verify

Message to sign: hello

Random K < Q: 604924B10E473B68F181EF47A7FCBF69E17AB812

Signature (r,s): 42E2929B5555AEBBCDBC02DB35CBC6C7C33C8E8A5 A448A88E8A3AFCD3

Sign Verify Cancel

DSA Signature created.

消息: hello

随机选取的秘密数K

Random K < Q: 604924B10E473B68F181EF47A7FCBF69E17AB812

签名

$(\gamma, \delta) = (42E2929B5555AEBBCDBC02DB35CBC6C7C33C8E8A5, A448A88E8A3AFCD3D61CDB4B1F40B4E57D63AEAD)$

Signature (r,s): 42E2929B5555AEBBCDBC02DB35CBC6C7C33C8E8A5 A448A88E8A3AFCD3

(3) 验证

不修改签名内容，直接验证得到结果**Success, Signature verified**



DSA - Sign/Verify

Message to sign: hello

Random K < Q: 604924B10E473B68F181EF47A7FCBF69E17AB812

Signature (r,s): 42E2929B5555AEBBCDBC02DB35CBC6C7C33C8E8A5 A448A88E8A3AFCD3

Sign Verify Cancel

--- Success. Signature verified ---

修改签名，将r的前两个数42改为53后得到结果**ERROR. SIGNATURE DOES NOT MATCH !!!**，如下图所示。

DSA - Sign/Verify

×

Message to sign:

hello

Random K < Q:

604924B10E473B68F181EF47A7FCBF69E17AB812

Signature (r,s):

53E2929B5555AEBBCDBC02DB35CBC6C7C33C8E8A5 A448A88E8A3AFCD3

Sign

Verify

Cancel

--- ERROR. SIGNATURE DOES NOT MATCH !!! ---