

实验七 DSA的实现

17341125 蒙亚愿 信息安全

一、实验目的

1. 了解数字签名的基本原理，掌握数字签名算法DSA的实现方法
2. 实现DSA算法并验证

二、实验要求

实现数字签名算法DSA，对消息签名并给出验证。

p: 2048比特素数

q: 可被p-1整除的256比特素数

hash函数: SHA-256 (可以调库)

消息: SchoolofDataandComputerScience,Sunyat-senUniversity (无空格)

私钥: 自己选取随机数，生成公钥并给出

三、实验介绍

3.1 实验综述

本次实验按照书中的DSA密码体制的介绍，可以简单的实现：

实验中已知变量 p, q, α ，私钥 a 可以自己确定，且 $1 \leq a \leq q - 1$ ，则可以通过

$$\beta = \alpha^a \pmod{p}$$

计算出公钥之一 β 。于是得到了整个密钥 K ：

$$K = \{ (p, q, \alpha, a, \beta) : \beta = \alpha^a \pmod{p} \}$$

在签名中，选择一个随机数 $k : 1 \leq k \leq q - 1$ ，就可以计算对消息 x 的签名：

$$\begin{aligned} sig_K(x, k) &= (\gamma, \delta) \\ \gamma &= (\alpha^k \pmod{p}) \pmod{q} \\ \delta &= (SHA - 256(x) + a\gamma)k^{-1} \pmod{q} \end{aligned}$$

如果 $\gamma = 0$ 或者 $\delta = 0$ ，则重新选随机数 k 计算（概率很小）。

验证过程：

$$\begin{aligned} e_1 &= SHA - 256(x)\delta^{-1} \pmod{q} \\ e_2 &= \gamma \delta^{-1} \pmod{1} \\ ver_K(x, (\gamma, \delta)) &= true \Leftrightarrow (\alpha^{e_1} \beta^{e_2} \pmod{p}) \pmod{q} \end{aligned}$$

3.2 实验环境

运行环境: Windows

使用语言: C++

编译器: CodeBlocks

库: gmp库

四、算法原理以及代码实现

4.1 SHA256函数

由于openssl、miracl和crypto库一直安装不了，所以采用网上的SHA-256的代码。在与DSA实现的衔接部分，由于SHA-256的输出是8个*unsigned int*类型的数组，所以在计算出SHA-256之后，还需要把这个数组串在一起，转成字符串类型：(123→"123")。

总的来说，就是逐个数的转换成字符串，然后再拼接，最后返回mpz_class类型的结果。

1. 用sprintf函数，逐个把*unsigned int*类型转成char*类型。

每个*unsigned int*的数字长度最多为10（十进制），所以声明一个长度为10的buff (char)，用于存储每个数字的字符串类型。

2. 拼接已经转好类型的字符串。

用Codeblocks似乎不能运行strcat函数，也就是strcat函数在DevC++能够正确运行，但是在Codeblock总是出错，debug之后发现，每次转换成字符串*buff*之后，之前拼接好的长字符串*str*的第一位就会变成'\0'，所以需要记下第一位，在每次转换完类型之后，把长字符串*str*的第一位恢复回来。

并且即使这样，还是不能正确运行strcat函数，只能把转换好的字符串*buff*逐位复制给*str*。

3. 转成mpz_class类型。

转换成长的字符串*str*之后，在用这个字符串初始化mpz_class类型的*ans*之前，需要检查第一位是否是'0'，是'0'的话，需要去掉，否则初始化*ans*时会报错。

```
_sha256 = _hash(x);
```

```
mpz_class _hash(UChar Y[Max]){
    char str[81] ;
    PAD(Y);
    // codeblocks太奇怪了，只能这样子拼接
    char first ;
    int flag = 0 ;
    for(int i=0;i<8;i++){
        char buff[10] = "" ;
        sprintf(buff, "%010lu", M_D.H[i]) ;
        if(i>0) str[0]=first ; // 恢复第一位
        for(int j=0;j<10;j++){
            str[flag+j] = buff[j] ;
        }
        flag+=10 ;
        if(i==0) first = str[0] ;
    }

    while(str[0]=='0'){ //消0
        for(int i=0;i<80;i++) str[i] = str[i+1] ;
    }
}
```

```

    }
    mpz_class ans ;
    ans = str ;
    return ans;
}

```

4.2 DSA实现

4.2.1 初始化

用十六进制的字符串初始化 p, q, α :

```

    mpz_init_set_str(key_p.get_mpz_t(),
        "008746338ba0d0b67ec6dde878f14d624dfe86a4663aaf170208b02b020c09199af1dbaaf0ba5a
        70d52b3eb715f775b0989ff176d1d6c680042dab48d35d802c598036280c559ffb59ea4c82c01fae
        4a227847cb715e03511602f7dedd0d0f1556c5e63c9b181a4a7ffb1416a3d6ef69cb9b413746aab5
        e259c3b12bec3244a7e09f6718ecb721c528020a9ae09568f461512c526593563fad544106fb5fa9
        0922aef456a0f110ccab7628d793ca3c5f6a4c6e22ee05227e1d3c395f99f52418c5dfdee1442d0a
        a3ca1e9154cae569c25497c3445f2bed1e8de3cada7e2f1825d7341e591234df4053ca5b33c92e32
        17967ca00211b68c1b0aafa5152b84557a81",
        16 ) ;

    mpz_init_set_str(key_q.get_mpz_t(),
        "00bdd727428445e6e4ad64978a4184de1f28ab3f298e79f11147bee060150040e1",
        16 ) ;

    mpz_init_set_str(key_alpha.get_mpz_t(),
        "75fd519cb49bcfd504dc7f8bfb100511dee71335ba07d7bf85e0b602441b121fc1b537f29b94125
        4f85703d2aa41e2cf400e3be7d952ad9a32f461174873034631295c2bc86d5134779358e8f4c6d54
        e591edc645dbd389088f9a7dfa3927fbeedce1bc57fa1e6dd3908daac26914ead4f40e8acab8b57a
        303d1970a06393789c56decbb504f7ae3d9f8e2cd1850b9e77d4aa64f49218fd0f6c24ccb5aa6af9
        d97e4d041b0676dda90dfbba0177df89b730ab1b8db5ef19ebd2fed52a790826fc507647576bebed
        69e5a8a11ec25f89d445b56b738d2a7f713d74527a4e6ed99af43ebcdb2cfe2ccaee76fd543ea429
        ec6402c273b030a63e6a686b2d299d8d7",
        16 ) ;

```

规定好私钥 a 的取值，并计算 $\beta = \alpha^a \bmod p$:

```

    key_a =
        "1352189132780742348147690325435974676612066923875027455889316666163952829642123
        " ;
    mpz_powm(key_beta.get_mpz_t(), key_alpha.get_mpz_t(), key_a.get_mpz_t(),
        key_p.get_mpz_t()) ;

```

4.2.2 签名

先选择一个随机数 $k: 1 \leq k \leq q - 1$ ，再通过公式计算签名 γ 和 δ :

$$\begin{aligned}
 sig_K(x, k) &= (\gamma, \delta) \\
 \gamma &= (\alpha^k \bmod p) \bmod q \\
 \delta &= (SHA - 256(x) + a\gamma)k^{-1} \bmod q
 \end{aligned}$$

若 $\gamma = 0$ 或者 $\delta = 0$ ，就重新选择随机数 k 计算。

```

void signature(mpz_class x){
    cout<<"\n----- signature(message,k) -----
-----"<<endl;

    start:
    // 选1~q-1的随机数
    mpz_class k;
    gmp_randclass rr(gmp_randinit_default) ;
    rr.seed(time(NULL)) ;
    k = rr.get_z_bits(160) ;
    k += key_q ;
    k %= key_q ;

    cout<<"\n k is "<<k<<endl;

    mpz_powm(sig_gamma.get_mpz_t(), key_alpha.get_mpz_t(), k.get_mpz_t(),
key_p.get_mpz_t()) ;
    mpz_mod(sig_gamma.get_mpz_t(), sig_gamma.get_mpz_t(), key_q.get_mpz_t()) ;
    cout<<"\n gamma is "<<sig_gamma<<endl;

    mpz_class invert_k ;
    mpz_invert(invert_k.get_mpz_t(), k.get_mpz_t(), key_q.get_mpz_t()) ;
    sig_delta = ( _sha256+key_a*sig_gamma)*invert_k ;
    sig_delta %= key_q ;
    cout<<"\n delta is "<<sig_delta<<endl;

    if(sig_gamma==0 || sig_delta==0) goto start ;
    cout<<"\n Neither gamma nor delta is equal to 0, continue..."<<endl;

}

```

4.2.3 验证

通过公式计算 $e_1, e_2, \alpha^{e_1}, \beta^{e_2}$, 验证 $(\alpha^{e_1} \beta^{e_2} \bmod p) \bmod q$ 是否等于 γ 。

$$e_1 = SHA - 256(x) \delta^{-1} \bmod q$$

$$e_2 = \gamma \delta^{-1} \bmod 1$$

```

bool verify(mpz_class sig_gamma, mpz_class sig_delta){
    cout<<"\n----- verify(message, gamma, delta) -----
-----"<<endl;

    mpz_class e1, e2 ;

    mpz_class invert_delta ;
    mpz_invert(invert_delta.get_mpz_t(), sig_delta.get_mpz_t(),
key_q.get_mpz_t()) ;

    e1 = _sha256*invert_delta ;
    e1 %= key_q ;
    cout<<"\n e1 is "<<e1<<endl;

    e2 = sig_gamma*invert_delta ;
    e2 %= key_q ;
    cout<<"\n e2 is "<<e2<<endl;

    mpz_class test, t1, t2 ;

```

```

        mpz_powm(t1.get_mpz_t(), key_alpha.get_mpz_t(), e1.get_mpz_t(),
key_p.get_mpz_t());
        mpz_powm(t2.get_mpz_t(), key_beta.get_mpz_t(), e2.get_mpz_t(),
key_p.get_mpz_t());
        test = t1*t2 ;
        test %= key_p ;
        test %= key_q ;

        return test==sig_gamma?true:false ;
}

```

```

if(verify(sig_gamma, sig_delta)) {
    cout<<"\n Test is passed! Congratulations!"<<endl;
}
else cout<<"\n Validation failed. Invalid signature!"<<endl;

```

五、实验效果即分析

每次实验都会显示公钥、私钥和消息以及消息的散列值。

规定

$a = 1352189132780742348147690325435974676612066923875027455889316666163952829642123$

在签名过程中，显示随机数 k 以及签名 γ, δ 。

在验证过程中，显示变量 e_1 和 e_2 。

1. 规定消息是“SchoolofDataandComputerScience,Sunyat-senUniversity”。

散列值为1f274b71a5bec2cc57914166d66d000d7b32c163105a626d5de878eba8a8ecd3

验证通过。

```

C:\Users\DELL\Desktop\密码学\实验7 DSA\CODE\DSA\bin\Debug\DSA.exe
The message is "SchoolofDataandComputerScience,Sunyat-senUniversity"

***** public key *****

p is 1707678874597591438281790056162780764022636829658889384457507120989307948980190533507461838999421902303482024972745268450971426943787028521830
242884057779297127024684281186989560758402548961537325231536514165647887944898601944211530238008635549785227448242103611624914245965438296675597253
892346698962539341558984207736649575603690287581963025244894396712853707951820099549239475575843965277441944983011699228188757965126638234967787033
96159023835084793497270962962664692175276964322862898792365394664775663353564080495990729593252516546223182590659842700445136304545766209788465045
134161667824761110805615245953

q is 85867271462558683382392908778501421404821495432764570749371157194193678385377

alpha is 148947978357088403932274382567808579730397872025864507725835283862924623247765469915618585578050942445721284027237277971510346499192925508
3417569033947596080262511359495883458993898718374418861146443293245179540367801696394741385034765398740062868429040839175767996081826767720952740015
3134357104789680912688993778967215948924426385238818727898653357851312140467497526709370020937231633377845527906471224864732671884047236017222906428
1098370118363345608737958610883275984475440284850637215006124957570071405118601646038671827690889256483762505360840516569303908514556665764958000126
58853715478193904171187728662780119

beta is 1637129901424008461656104194022830532800274944661361598666760695174766065216717724392233103742465732606136671768768710280799461989461783806
7283945883099015409326042807199170128643526161154271719734952691405174673519224560056067710443454291811355734142907554546062402123302333954941352508
8728094972106707721179512044582950642538190506146345462238206825267893361899272970565393015039959610578115254533990975968128013610078647604523247332
74704850256313980823041471994755538176909031037767726967313435501799358367379836323852525223044600219586744324907303084098895766500325621232457813
9785740876950421830643117517161515

***** private key *****

a is 1352189132780742348147690325435974676612066923875027455889316666163952829642123

_sha256(message) is 5226689132780742348146913725435974676612066923875027435889315755163952829642963

signature(message,k)
-----
k is 573664435271981311612238096116767356833299612191
gamma is 49112435404219206016358626351324549913927609347497161907348222694677508549831
delta is 35260578374521153724665013814497265250685787655488333079400131679130920747849

Neither gamma nor delta is equal to 0, continue...

----- verify(message, gamma, delta) -----
e1 is 56497357025414178937401106484174001852638289493348357949033003971125379947604
e2 is 2097084265427261310564906684253661060702733933221289908948104128611336757

Test is passed! Congratulations!
请按任意键继续. . .

```

2. 任意消息验证

输入消息为goodbye

散列值为82e35a63ceba37e9646434c5dd412ea577147f1e4a41ccde1614253187e3dbf9

验证通过。

```
"C:\Users\DELL\Desktop\密码学实验7 DSA\CODE\DSA\DSA2\bin\Debug\DSA2.exe"
3417569033947596080262511359495883458993898718374418861146443293245179540367801696394741385034765398740062868429040839175767996081826767720952740015
3134357104789680912688993778967215948924426385238818727898653357851312140467497526709370020937231633377845527906471224864732671884047236017222906428
1098370118363345608737958610883275984475440284850637215006124957570071405118601646038671827690889256483762505360840516569303908514556665764958000126
58853715478193904171187728662780119

beta is 1637129901424008461656104194022830532800274944661361598666760695174766065216717724392233103742465732606136671768768710280799461989461783806
7283945883099015409326042807199170128643526161154271719734952691405174673519224560056067710443454291811355734142907554546062402123302333954941352508
8728094972106707721179512044582950642538190506146345462238206825267893361899272970565393015039959610578115254533990975968128013610078647604523247332
7470485025631398082304147199475553817690903103777677269673134355017993583673798363238525252223044600219586744324907303084098895766500325621232457813
9785740876950421830643117517161515

***** private key *****

a is 1352189132780742348147690325435974676612066923875027455889316666163952829642123

Enter message: goodbye
HASH (HEX) : 82e35a63ceba37e9646434c5dd412ea577147f1e4a41ccde1614253187e3dbf9
_sha256(message) is (Dec) 21959378913468310505168428870937120365171997831966124582627003704189932279857145

----- signature(message, k) -----

k is 3a65c1316ad779437435dba8a86d9ee17847cbbe

gamma is 148f5d3346a8e81a8f4e120276b41ade4fd2b282546f53da748923d8e257ebbf

delta is 5371dc1701ea3b857b4ed704500e385f4a83c0cf52be87ee2856dff0205d468d

Neither gamma nor delta is equal to 0, continue

----- verify(message, gamma, delta) -----

e1 is 41fdc9fcf8b852790b6f4848e78a01b9dc2e7ccbd33c5568b57a33dc51111fa
e2 is 9a66fe2b3876c0af625b2da7fafb925f43748ef2feed745471e5e286180beac

Test is passed! Congratulations!

Process returned 0 (0x0)   execution time : 2.443 s
Press any key to continue.
```

六、实验感想

1. 关于SHA-256。我最开始是想安装一下库，直接调用函数会比较方便，没想到安装了五天都不行，不知道什么原因，每到最后一步就不行，后来就放弃了，既然SHA-256不需要自己实现，我就上网抄了一段运行正确的代码。只是在和DSA的接口那有点费劲，但总比安装库好解决。
2. 关于编译器。我先是在DevC++写的把散列值转成字符串，很快就写出来并且运行正确，但是在Codeblocks同一段代码就怎么都运行不对，百度也不知道咋回事，总是会把长字符串的第一个元素“吃”掉，并且strcat函数也运行不对，所以只能用最笨的方法慢慢实现，代码略长略啰嗦，但总比安装库好解决。
3. 关于DSA。其实DSA直接按照书上说的实现就可以，没有难度，很快就打出来了，并且通过这次实验对于DSA也有了更加深刻的理解和记忆。比安装库好解决。

DSA与之前不一样的，在Alice这里，签名是私有的，所以需要私钥，而验证是谁都可以的，所以不需要私钥。之前是双方都需要私钥进行加解密，所以私钥需要在安全的信道中传输。签名方案中也有很多脑洞很大的算法，总而言之都是共同算出相同的变量的话验证就成功了。

这次实验很简单，希望下次随机数的实验更加简单，辛苦TA和老师啦~