《现代密码学》实验报告

实验名称：DSA 签名的实现与验证	**实验时间**：2025 年 12 月 28 日
学生姓名：谢润文	**学号**：23336262
学生班级：23 计科	**成绩评定**：

一、实验目的

通过前置大数运算的模乘、求逆、模幂等操作，进而实现(L,N)=(2048,256)的 DSA 数字签名算法的签名与验证，认识并理解“数字签名是非对称密钥加密技术与数字摘要的应用”，理解签名中防伪造、防重放、防篡改设计的思想，提高大数运算、非对称加密及 hash 函数的应用能力。

二、实验内容

用 C/C++实现(L,N)=(2048,256)的 DSA 数字签名算法的签名与验证。

参数要求：

1. 签名，输入由 uint8_t[256]以大端序表示的 2048 bit 的素数 p，uint8_t[32]以大端序表示的 256 bit 的被 p-1 整除的素数 q，uint8_t[256]以大端序表示的 2048 bit 的 g（在 mod p 下生成 q 个元素的子群），uint8_t[32]以大端序表示的 256 bit 的 x（私钥，并生成 $g^x=y$ 作为公钥），uint16_t 签名的消息长度，uint8_t[]签名的消息内容构成。输出签名大端序表示的 32bit 的（r，s）。

2. 验证，输入由 p、q、g、y、r、s 构成，定义与签名相同。如果签名有效，输出 PASS 即十六进制数据 50 41 53 53；否则输出 FAIL 也即 46 41 49 4C。

三、实验原理

1. 数字签名的定义及功能

数字签名（又称公钥数字签名）是只有信息的发送者才能产生的别人无法伪造的一段数字串，这段数字串同时也是对信息的发送者发送信息真实性的一个有效证明。一套数字签名通常定义两种互补的运算，一个用于签名，另一个用于验证。数字签名是非对称密钥加密技术与数字摘要技术的应用。[1]

数字签名机制作为保障网络信息安全的手段之一，可以解决伪造、抵赖、冒充和篡改问题。数字签名的目的之一就是在网络环境中代替传统的手工签字与印章。

2. DSA 数字签名算法

DSA 是由 Schnorr 和 ElGamal 签名算法演变而来的，被 NIST 作为数字签名标准。除了定义的参数外，还需要一个单向哈希函数 H(x)，DSS（**D**igital **S**ignature **S**tandard）中选用 SHA-1 或 SHA-2，此实验为方便起见，仍然利用之前的 SHA-256。

签名算法如下：

1）选择随机数 k，满足 0<k<q。

2）计算 $r=(g^k \bmod p) \bmod q$。

3）计算 $s=(H(m)+xr)k^{-1} \bmod q$。

4）如果 r=0 或 s=0，重新回到第一步，选另一个 k。否则输出（r，s）。

验证算法如下：

1）计算 $e1=H(m)s^{-1} \bmod q$

2）计算 $e2=rs^{-1} \bmod q$

3）计算 $r'=(g^{e1}y^{e2} \bmod p) \bmod q$，如果 r’=r，验证通过，否则失败。

不难证明这个验证是有效的。这是因为 $y^{e2} \bmod p=g^{xe2} \bmod p$，从而 $r'=(g^{e1+xe2} \bmod p) \bmod q$。注意到 $e1=H(m)s^{-1} \bmod q$，$e2=rs^{-1} \bmod q$，从而 $e1+xe2=s^{-1}(H(m)+xr) \bmod q$。再因为 $s=(H(m)+xr)k^{-1} \bmod q$，两边求逆得 $s^{-1}=(H(m)+xr)^{-1}k \bmod q$，代入得 $e1+xe2=k \bmod q$。注意到 0<k<q，则 $k \bmod q=k$，从而 $r'=(g^k \bmod p) \bmod q=r$。

3. 对 DSA 的攻击

对 DSA 的攻击主要包括共享 k 攻击和线性 k 攻击，这些攻击利用了 DSA 在随机数 k 选择上的弱点，可能导致私钥泄露。上述攻击会在思考题中介绍。

此外，利用格规约算法（如 BKZ）进行复杂的数学攻击可以解决当 k 的部分位泄露时的离散对数问题，即使是 384bit 的 k，在 5bit 泄露的情况下仍然会有 81%的概率成功恢复[2]，足以破坏安全性。

其他攻击包括常见的侧信道攻击，比如时机/缓存攻击（例如，FLUSH+RELOAD），通过分析内存访问模式或执行时间可以揭示密钥或 k 的部分，从而针对 DSA 攻击（参见 RFC 6979）。

四、实验步骤（源代码）

这里先展示 DSA 的签名算法。

```
#include<iostream>
#include<vector>
#include<string>
#include<algorithm>
#include<cstring>
#include<cstdio>
#include<fstream>
#include<cstdint>
#include<fcntl.h>
#include<wmmintrin.h>
#include<immintrin.h>
#include<emmintrin.h>
using namespace std;
/*
```

```
    FILE *fp=fopen("dump (12).bin","rb");
    FILE *wp=fopen("output.bin","wb");
    */

    #define fp stdin
    #define wp stdout

    const uint32_t h[9]={0,0x6a09e667,0xbb67ae85,0x3c6ef372,0xa54ff53a,
                         0x510e527f,0x9b05688c,0x1f83d9ab,0x5be0cd19};
    uint32_t _hash[10]={0};
    __m128i
state0,state1,msg,msgtmp0,msgtmp1,msgtmp2,msgtmp3,msgtmp4,abef,cdgh;
    __m128i shuf_mask=_mm_set_epi32(
        0x0c0d0e0f,
        0x08090a0b,
        0x04050607,
        0x00010203
    );
    void sha_256(const uint8_t* data,int start){
        if(!start){
            state0=_mm_set_epi64x(0x6a09e667bb67ae85,0x510e527f9b05688c);
            state1=_mm_set_epi64x(0x3c6ef372a54ff53a,0x1f83d9ab5be0cd19);
        }
        abef=state0;
        cdgh=state1;
        msg=_mm_loadu_si128((__m128i*)(data+0));
        msgtmp0=_mm_shuffle_epi8(msg,shuf_mask);
        msg=_mm_add_epi32(msgtmp0,_mm_set_epi64x(0xe9b5dba5b5c0fbcf,0x71
374491428a2f98));
        state1=_mm_sha256rnds2_epu32(state1,state0,msg);
        msg=_mm_shuffle_epi32(msg,0x0e);
        state0=_mm_sha256rnds2_epu32(state0,state1,msg);
        msg=_mm_loadu_si128((__m128i*)(data+16));
        msgtmp1=_mm_shuffle_epi8(msg,shuf_mask);
        msg=_mm_add_epi32(msgtmp1,_mm_set_epi64x(0xab1c5ed5923f82a4,0x59
f111f13956c25b));
        state1=_mm_sha256rnds2_epu32(state1,state0,msg);
        msg=_mm_shuffle_epi32(msg,0x0e);
        state0=_mm_sha256rnds2_epu32(state0,state1,msg);
        msgtmp0=_mm_sha256msg1_epu32(msgtmp0,msgtmp1);
        msg=_mm_loadu_si128((__m128i*)(data+32));
        msgtmp2=_mm_shuffle_epi8(msg,shuf_mask);
        msg=_mm_add_epi32(msgtmp2,_mm_set_epi64x(0x550c7dc3243185be,0x12
835b01d807aa98));
```

```
        state1=_mm_sha256rnds2_epu32(state1,state0,msg);
        msg=_mm_shuffle_epi32(msg,0x0e);
        state0=_mm_sha256rnds2_epu32(state0,state1,msg);
        msgtmp1=_mm_sha256msg1_epu32(msgtmp1,msgtmp2);
        msg=_mm_loadu_si128((__m128i*)(data+48));
        msgtmp3=_mm_shuffle_epi8(msg,shuf_mask);
        msg=_mm_add_epi32(msgtmp3,_mm_set_epi64x(0xc19bf1749bdc06a7,0x80
deb1fe72be5d74));
        state1=_mm_sha256rnds2_epu32(state1,state0,msg);
        msgtmp4=_mm_alignr_epi8(msgtmp3,msgtmp2,4);
        msgtmp0=_mm_add_epi32(msgtmp0,msgtmp4);
        msgtmp0=_mm_sha256msg2_epu32(msgtmp0,msgtmp3);
        msg=_mm_shuffle_epi32(msg,0x0e);
        state0=_mm_sha256rnds2_epu32(state0,state1,msg);
        msgtmp2=_mm_sha256msg1_epu32(msgtmp2,msgtmp3);

msg=_mm_add_epi32(msgtmp0,_mm_set_epi64x(0x240ca1cc0fc19dc6,0xefbe4786e
49b69c1));
        state1=_mm_sha256rnds2_epu32(state1,state0,msg);
        msgtmp4=_mm_alignr_epi8(msgtmp0,msgtmp3,4);
        msgtmp1=_mm_add_epi32(msgtmp1,msgtmp4);
        msgtmp1=_mm_sha256msg2_epu32(msgtmp1,msgtmp0);
        msg=_mm_shuffle_epi32(msg,0x0e);
        state0=_mm_sha256rnds2_epu32(state0,state1,msg);
        msgtmp3=_mm_sha256msg1_epu32(msgtmp3,msgtmp0);
        msg=_mm_add_epi32(msgtmp1,_mm_set_epi64x(0x76f988da5cb0a9dc,0x4a
7484aa2de92c6f));
        state1=_mm_sha256rnds2_epu32(state1,state0,msg);
        msgtmp4=_mm_alignr_epi8(msgtmp1,msgtmp0,4);
        msgtmp2=_mm_add_epi32(msgtmp2,msgtmp4);
        msgtmp2=_mm_sha256msg2_epu32(msgtmp2,msgtmp1);
        msg=_mm_shuffle_epi32(msg,0x0e);
        state0=_mm_sha256rnds2_epu32(state0,state1,msg);
        msgtmp0=_mm_sha256msg1_epu32(msgtmp0,msgtmp1);
        msg=_mm_add_epi32(msgtmp2,_mm_set_epi64x(0xbf597fc7b00327c8,0xa8
31c66d983e5152));
        state1=_mm_sha256rnds2_epu32(state1,state0,msg);
        msgtmp4=_mm_alignr_epi8(msgtmp2,msgtmp1,4);
        msgtmp3=_mm_add_epi32(msgtmp3,msgtmp4);
        msgtmp3=_mm_sha256msg2_epu32(msgtmp3,msgtmp2);
        msg=_mm_shuffle_epi32(msg,0x0e);
        state0=_mm_sha256rnds2_epu32(state0,state1,msg);
        msgtmp1=_mm_sha256msg1_epu32(msgtmp1,msgtmp2);
        msg=_mm_add_epi32(msgtmp3,_mm_set_epi64x(0x1429296706ca6351,0xd5
```

```
a79147c6e00bf3));
        state1=_mm_sha256rnds2_epu32(state1,state0,msg);
        msgtmp4=_mm_alignr_epi8(msgtmp3,msgtmp2,4);
        msgtmp0=_mm_add_epi32(msgtmp0,msgtmp4);
        msgtmp0=_mm_sha256msg2_epu32(msgtmp0,msgtmp3);
        msg=_mm_shuffle_epi32(msg,0x0e);
        state0=_mm_sha256rnds2_epu32(state0,state1,msg);
        msgtmp2=_mm_sha256msg1_epu32(msgtmp2,msgtmp3);
        msg=_mm_add_epi32(msgtmp0,_mm_set_epi64x(0x53380d134d2c6dfc,0x2e
1b213827b70a85));
        state1=_mm_sha256rnds2_epu32(state1,state0,msg);
        msgtmp4=_mm_alignr_epi8(msgtmp0,msgtmp3,4);
        msgtmp1=_mm_add_epi32(msgtmp1,msgtmp4);
        msgtmp1=_mm_sha256msg2_epu32(msgtmp1,msgtmp0);
        msg=_mm_shuffle_epi32(msg,0x0e);
        state0=_mm_sha256rnds2_epu32(state0,state1,msg);
        msgtmp3=_mm_sha256msg1_epu32(msgtmp3,msgtmp0);
        msg=_mm_add_epi32(msgtmp1,_mm_set_epi64x(0x92722c8581c2c92e,0x76
6a0abb650a7354));
        state1=_mm_sha256rnds2_epu32(state1,state0,msg);
        msgtmp4=_mm_alignr_epi8(msgtmp1,msgtmp0,4);
        msgtmp2=_mm_add_epi32(msgtmp2,msgtmp4);
        msgtmp2=_mm_sha256msg2_epu32(msgtmp2,msgtmp1);
        msg=_mm_shuffle_epi32(msg,0x0e);
        state0=_mm_sha256rnds2_epu32(state0,state1,msg);
        msgtmp0=_mm_sha256msg1_epu32(msgtmp0,msgtmp1);
        msg=_mm_add_epi32(msgtmp2,_mm_set_epi64x(0xc76c51a3c24b8b70,0xa8
1a664ba2bfe8a1));
        state1=_mm_sha256rnds2_epu32(state1,state0,msg);
        msgtmp4=_mm_alignr_epi8(msgtmp2,msgtmp1,4);
        msgtmp3=_mm_add_epi32(msgtmp3,msgtmp4);
        msgtmp3=_mm_sha256msg2_epu32(msgtmp3,msgtmp2);
        msg=_mm_shuffle_epi32(msg,0x0e);
        state0=_mm_sha256rnds2_epu32(state0,state1,msg);
        msgtmp1=_mm_sha256msg1_epu32(msgtmp1,msgtmp2);
        msg=_mm_add_epi32(msgtmp3,_mm_set_epi64x(0x106aa070f40e3585,0xd6
990624d192e819));
        state1=_mm_sha256rnds2_epu32(state1,state0,msg);
        msgtmp4=_mm_alignr_epi8(msgtmp3,msgtmp2,4);
        msgtmp0=_mm_add_epi32(msgtmp0,msgtmp4);
        msgtmp0=_mm_sha256msg2_epu32(msgtmp0,msgtmp3);
        msg=_mm_shuffle_epi32(msg,0x0e);
        state0=_mm_sha256rnds2_epu32(state0,state1,msg);
        msgtmp2=_mm_sha256msg1_epu32(msgtmp2,msgtmp3);
```

```
msg=_mm_add_epi32(msgtmp0,_mm_set_epi64x(0x34b0bcb52748774c,0x1e376c081
9a4c116));
        state1=_mm_sha256rnds2_epu32(state1,state0,msg);
        msgtmp4=_mm_alignr_epi8(msgtmp0,msgtmp3,4);
        msgtmp1=_mm_add_epi32(msgtmp1,msgtmp4);
        msgtmp1=_mm_sha256msg2_epu32(msgtmp1,msgtmp0);
        msg=_mm_shuffle_epi32(msg,0x0e);
        state0=_mm_sha256rnds2_epu32(state0,state1,msg);
        msgtmp3=_mm_sha256msg1_epu32(msgtmp3,msgtmp0);
        msg=_mm_add_epi32(msgtmp1,_mm_set_epi64x(0x682e6ff35b9cca4f,0x4e
d8aa4a391c0cb3));
        state1=_mm_sha256rnds2_epu32(state1,state0,msg);
        msgtmp4=_mm_alignr_epi8(msgtmp1,msgtmp0,4);
        msgtmp2=_mm_add_epi32(msgtmp2,msgtmp4);
        msgtmp2=_mm_sha256msg2_epu32(msgtmp2,msgtmp1);
        msg=_mm_shuffle_epi32(msg,0x0e);
        state0=_mm_sha256rnds2_epu32(state0,state1,msg);
        msg=_mm_add_epi32(msgtmp2,_mm_set_epi64x(0x8cc7020884c87814,0x78
a5636f748f82ee));
        state1=_mm_sha256rnds2_epu32(state1,state0,msg);
        msgtmp4=_mm_alignr_epi8(msgtmp2,msgtmp1,4);
        msgtmp3=_mm_add_epi32(msgtmp3,msgtmp4);
        msgtmp3=_mm_sha256msg2_epu32(msgtmp3,msgtmp2);
        msg=_mm_shuffle_epi32(msg,0x0e);
        state0=_mm_sha256rnds2_epu32(state0,state1,msg);
        msg=_mm_add_epi32(msgtmp3,_mm_set_epi64x(0xc67178f2bef9a3f7,0xa4
506ceb90befffa));
        state1=_mm_sha256rnds2_epu32(state1,state0,msg);
        msg=_mm_shuffle_epi32(msg,0x0e);
        state0=_mm_sha256rnds2_epu32(state0,state1,msg);
        state0=_mm_add_epi32(state0,abef);
        state1=_mm_add_epi32(state1,cdgh);
    }
    void H(uint8_t data[],uint16_t len){
        uint8_t tmp[64];
        int start=0;
        uint16_t rem=len;
        int offset=0;
        while(rem>=64){
            sha_256(data+offset,start);
            start=1;
            rem-=64;
            offset+=64;
```

```
        }
        memcpy(tmp,data+offset,rem);
        tmp[rem]=0x80;
        if(rem<=55) memset(tmp+rem+1,0,55-rem);
        else{
            memset(tmp+rem+1,0,63-rem);
            sha_256(tmp,start);
            start=1;
            memset(tmp,0,56);
        }
        uint64_t tot_bits=(uint64_t)len*8;
        for(int i=0;i<8;++i) tmp[63-i]=(tot_bits>>(i<<3))&0xFF;
        sha_256(tmp,start);
        _hash[0]=_mm_extract_epi32(state0,3);
        _hash[1]=_mm_extract_epi32(state0,2);
        _hash[4]=_mm_extract_epi32(state0,1);
        _hash[5]=_mm_extract_epi32(state0,0);
        _hash[2]=_mm_extract_epi32(state1,3);
        _hash[3]=_mm_extract_epi32(state1,2);
        _hash[6]=_mm_extract_epi32(state1,1);
        _hash[7]=_mm_extract_epi32(state1,0);
    }
    const int MAXN=130;
    const uint64_t BASE10_19=10000000000000000000ull;
    inline uint8_t addcarry_u64(uint8_t c_in,uint64_t a,uint64_t b,uint64_t
*out){
        unsigned __int128 res=(unsigned __int128)a+b+c_in;
        *out=(uint64_t)res;
        return (uint8_t)(res>>64);
    }
    inline   uint8_t   subborrow_u64(uint8_t   b_in,uint64_t   a,uint64_t
b,uint64_t *out) {
        unsigned __int128 res=(unsigned __int128)a-b-b_in;
        *out=(uint64_t)res;
        return (uint8_t)(res>>64)&1;
    }

    inline uint64_t mulx_u64(uint64_t a,uint64_t b,uint64_t *hi){
        unsigned __int128 res=(unsigned __int128)a*b;
        *hi=(uint64_t)(res>>64);
        return (uint64_t)res;
    }

    struct bign{
```

```
    uint64_t d[MAXN];
    int len;
    bool sign;
    bign(){
        d[0]=0;
        len=1;
        sign=false;
    }
    bign(int v){
        if(v<0){
            sign=true;
            v=-v;
        }
        else sign=false;
        d[0]=(uint64_t)v;
        len=1;
    }
    bign(long long v){
        if(v<0){
            sign=true;
            v=-v;
        }
        else sign=false;
        d[0]=(uint64_t)v;
        len=1;
    }
    bign(unsigned long long v){
        d[0]=v;
        len=1;
        sign=false;
    }
    void clean(){
        while(len>1 && d[len-1]==0) len--;
        if(len==1 && d[0]==0) sign=false;
    }
    int abs_cmp(const bign& b) const {
        if(len!=b.len) return len<b.len?-1:1;
        for(int i=len-1;i>= 0;--i){
            if(d[i]!=b.d[i]) return d[i]<b.d[i]?-1:1;
        }
        return 0;
    }
    bool operator<(const bign& b) const {
        if(sign!=b.sign) return sign;
```

```
            if(sign) return abs_cmp(b)>0;
            return abs_cmp(b)<0;
        }
        bool operator>(const bign& b) const { return b < *this; }
        bool operator<=(const bign& b) const { return !(*this > b); }
        bool operator>=(const bign& b) const { return !(*this < b); }
        bool operator==(const bign& b) const { return sign==b.sign &&
abs_cmp(b)==0; }
        bool operator!=(const bign& b) const { return !(*this == b); }

        static bign abs_add(const bign& a,const bign& b){
            bign res;
            res.len=max(a.len,b.len);
            uint8_t carry=0;
            for(int i=0; i<res.len;++i){
                uint64_t ai=(i<a.len)?a.d[i]:0ull;
                uint64_t bi=(i<b.len)?b.d[i]:0ull;
                carry=addcarry_u64(carry,ai,bi,&res.d[i]);
            }
            if(carry){
                if(res.len<MAXN) res.d[res.len++]=carry;
            }
            return res;
        }

        static bign abs_sub(const bign& a,const bign& b){
            bign res;
            res.len=a.len;
            uint8_t borrow=0;
            for (int i=0;i<res.len;++i){
                uint64_t ai=(i<a.len)?a.d[i]:0ull;
                uint64_t bi=(i<b.len)?b.d[i]:0ull;
                borrow=subborrow_u64(borrow,ai,bi,&res.d[i]);
            }
            res.clean();
            return res;
        }

        void shift_words(int k){
            if(len==1 && d[0]==0) return;
            if(k<=0) return;
            for(int i=len-1;i>=0;--i) if(i+k<MAXN) d[i+k]=d[i];
            for(int i=0;i<k && i<MAXN;++i) d[i]=0;
            len+=k;
```

```
            if(len>MAXN) len=MAXN;
            while(len>1 && d[len-1]==0) len--;
        }
        void div_2(){
            uint64_t rem=0;
            for(int i=len-1;i>=0;--i){
                uint64_t cur=d[i];
                d[i]=(cur>>1) | (rem<<63);
                rem=cur&1;
            }
            clean();
        }
        void div_3(){
            uint64_t rem=0;
            for(int i=len-1;i>=0;--i){
                unsigned __int128 cur=((unsigned __int128)rem<<64)|d[i];
                d[i]=(uint64_t)(cur/3);
                rem=(uint64_t)(cur%3);
            }
            clean();
        }

        static bign mul(const bign& a,const bign& b){
            if((a.len==1 && a.d[0]==0) || (b.len==1 && b.d[0]==0)) return
bign(0);
            bign res;
            res.len=a.len+b.len;
            if(res.len>MAXN) res.len=MAXN;
            memset(res.d,0,sizeof(uint64_t)*res.len);
            for(int i=0;i<a.len;++i){
                uint64_t r_c=0;
                for(int j=0;j<b.len;++j){
                    if(i+j>=MAXN) break;
                    unsigned            __int128            prod=(unsigned
__int128)a.d[i]*b.d[j]+res.d[i+j]+r_c;
                    res.d[i+j]=(uint64_t)prod;
                    r_c=(uint64_t)(prod>>64);
                }
                if(i+b.len<MAXN) res.d[i+b.len]=r_c;
            }
            res.clean();
            res.sign=a.sign^b.sign;
            return res;
        }
```

```
       static pair<bign, bign> div_mod(const bign& u, const bign& v){
           if(v.len==1 && v.d[0]==0) return {bign(0),bign(0)};
           if(u.abs_cmp(v)<0) return {bign(0),u};
           if(v.len==1){
               bign q;
               q.len=u.len;
               uint64_t rem=0;
               for(int i=u.len-1;i>=0;--i){
                   unsigned            __int128            cur=((unsigned
__int128)rem<<64)|u.d[i];
                   q.d[i]=(uint64_t)(cur/v.d[0]);
                   rem=(uint64_t)(cur%v.d[0]);
               }
               q.clean();
               q.sign=u.sign^v.sign;
               bign r((unsigned long long)rem);
               r.sign=u.sign;
               return {q,r};
           }
           int s=__builtin_clzll(v.d[v.len-1]);
           bign vn=v,un=u;
           if(s>0){
               for(int                                  i=vn.len-1;i>0;--i)
vn.d[i]=(vn.d[i]<<s)|(vn.d[i-1]>>(64-s));
               vn.d[0]<<=s;
               if(un.len<MAXN) un.d[un.len]=0;
               for(int            i=un.len;i>0;--i)            if(i<MAXN)
un.d[i]=(un.d[i]<<s)|(un.d[i-1]>>(64-s));
               un.d[0]<<=s;
               if(un.len<MAXN && un.d[un.len]) un.len++;
           }
           if(un.len==u.len && un.len<MAXN) un.d[un.len++]=0;
           bign q;
           q.len=un.len-vn.len;
           for(int j=un.len-vn.len-1;j>=0;--j){
               unsigned              __int128              num=((unsigned
__int128)un.d[j+vn.len]<<64)|un.d[j+vn.len-1];
               unsigned __int128 q_=num/vn.d[vn.len-1];
               unsigned __int128 r_=num%vn.d[vn.len-1];
               while(1){
                   if(q_>=((unsigned
__int128)1<<64)||(q_*vn.d[vn.len-2]>((r_<<64)|un.d[j+vn.len-2]))){
                       q_--;
```

```
                    r_+=vn.d[vn.len-1];
                    if(r_>=((unsigned __int128)1<<64)) break;
                }
                else break;
            }
            unsigned __int128 k=0;
            for(int i=0;i<vn.len;++i){
                unsigned __int128 p=(unsigned __int128)q_*vn.d[i]+k;
                uint64_t sub=(uint64_t)p;
                k=p>>64;
                if(un.d[i+j]<sub) k++;
                un.d[i+j]-=sub;
            }
            if(un.d[j+vn.len]<k){
                un.d[j+vn.len]-=k;
                q_--;
                uint8_t c=0;
                for(int                                i=0;i<vn.len;++i)
c=addcarry_u64(c,un.d[i+j],vn.d[i],&un.d[i+j]);
                un.d[j+vn.len]+=c;
            }
            else un.d[j+vn.len]-=k;
            q.d[j]=(uint64_t)q_;
        }
        q.clean();
        q.sign=u.sign^v.sign;
        if(s>0)               for(int               i=0;i<vn.len;++i)
un.d[i]=(un.d[i]>>s)|(un.d[i+1]<<(64-s));
        un.len=vn.len;
        un.clean();
        un.sign=u.sign;
        return {q,un};
    }

    bign operator+(const bign& b) const {
        if(sign==b.sign) {
            bign res=abs_add(*this,b);
            res.sign=sign;
            return res;
        }
        else{
            if(abs_cmp(b)>=0){
                bign res=abs_sub(*this,b);
                res.sign=sign;
```

```
                return res;
            }
            else{
                bign res=abs_sub(b,*this);
                res.sign=b.sign;
                return res;
            }
        }
    }
    bign operator-(const bign& b) const{
        bign t=b;
        t.sign=!t.sign;
        return *this+t;
    }
    bign operator*(const bign& b) const{ return mul(*this, b); }
    bign operator/(const bign& b) const{ return div_mod(*this,
b).first; }
    bign operator%(const bign& b) const{
        bign res=div_mod(*this,b).second;
        if(res.sign) res=res+b;
        return res;
    }
    bign operator+=(const bign& b) { *this = *this + b; return *this; }
    bign operator-=(const bign& b) { *this = *this - b; return *this; }
    bign operator*=(const bign& b) { *this = *this * b; return *this; }
    bign operator/=(const bign& b) { *this = *this / b; return *this; }
    bign operator%=(const bign& b) { *this = *this % b; return *this; }

    static bign from_string(const uint8_t* str){
        bign res(0);
        int i=0;
        bool neg=false;
        if(str[0]=='-'){ neg = true; i++; }
        for(;str[i]; ++i){
            bign ten(10);
            bign digit(str[i]-'0');
            res=res*ten+digit;
        }
        res.sign=neg;
        return res;
    }
    void print(){
        if(len==1 && d[0]==0){
            putchar('0');
```

```
            putchar('\n');
            return;
        }
        if(sign) putchar('-');
        bign tmp=*this;
        tmp.sign=false;
        bign base((unsigned long long)BASE10_19);
        vector<uint64_t> c;
        while(tmp.len>1 || tmp.d[0]>=BASE10_19){
            pair<bign,bign> qr=div_mod(tmp,base);
            c.push_back(qr.second.d[0]);
            tmp=qr.first;
        }
        c.push_back(tmp.d[0]);

        uint64_t val = c.back();
        if (val == 0) putchar('0');
        else {
            char buf[32];
            int idx = 0;
            while(val) {
                buf[idx++] = val % 10 + '0';
                val /= 10;
            }
            while(idx--) putchar(buf[idx]);
        }

        for(int i=c.size()-2;i>=0;--i) {
            val = c[i];
            char buf[32];
            for(int j=0; j<19; ++j) {
                buf[j] = val % 10 + '0';
                val /= 10;
            }
            for(int j=18; j>=0; --j) putchar(buf[j]);
        }
        putchar('\n');
    }
    static bign from_bytes(const uint8_t* buf,int len){
        bign res=0;
        for(int i=0;i<len;++i) res=res*256+buf[i];
        return res;
    }
    void to_bytes(uint8_t* buf,int len)const{
```

```
            bign temp=*this;
            for(int i=len-1;i>=0;--i){
                buf[i]=(uint8_t)((temp%256).d[0]);
                temp=temp/256;
            }
        }
    };
    void mont_mul(const bign& a, const bign& b, const bign& n, uint64_t
n_prime, bign& res){
        uint64_t t[2*MAXN+2]={0};
        int k=n.len;
        for(int i=0;i<a.len;++i){
            uint64_t ai=a.d[i];
            if(ai==0) continue;
            uint64_t carry=0;
            uint64_t *tp=t+i;
            for(int j=0;j<b.len;++j){
                unsigned              __int128              prod=(unsigned
__int128)ai*b.d[j]+tp[j]+carry;
                tp[j]=(uint64_t)prod;
                carry=(uint64_t)(prod>>64);
            }
            tp[b.len]+=carry;
        }
        for(int i=0;i<k;++i){
            uint64_t m=t[i]*n_prime;
            uint64_t carry=0;
            uint64_t *tp=t+i;
            for(int j=0;j<k;++j){
                unsigned              __int128              prod=(unsigned
__int128)m*n.d[j]+tp[j]+carry;
                tp[j]=(uint64_t)prod;
                carry=(uint64_t)(prod>>64);
            }
            int current=i+k;
            while(carry){
                unsigned __int128 sum=(unsigned __int128)t[current]+carry;
                t[current]=(uint64_t)sum;
                carry=(uint64_t)(sum>>64);
                current++;
            }
        }
        res.len=k;
        for(int i=0;i<k;++i) res.d[i]=t[i+k];
```

```
        bool fl=(t[2*k]!=0);
        if(!fl){
            for(int i=k-1;i>=0;--i){
                if(res.d[i]>n.d[i]) { fl = true; break; }
                if(res.d[i]<n.d[i]) { break; }
                if(i==0) fl=true;
            }
        }
        if(fl){
            uint8_t borrow=0;
            for(int i=0;i<k;++i){
                uint64_t ni=n.d[i];
                borrow=subborrow_u64(borrow,res.d[i],ni,&res.d[i]);
            }
        }
        res.clean();
    }

    bign mont_mul(const bign& a, const bign& b, const bign& n, uint64_t
n_prime){
        bign res;
        mont_mul(a,b,n,n_prime,res);
        return res;
    }
    bign zero(0),one(1);
    uint64_t get_n_prime(const bign& n){
        uint64_t x=1;
        uint64_t n0=n.d[0];
        for(int i=0;i<6;++i) x=x*(2-n0*x);
        return -x;
    }
    bign get_R2_mod_n(const bign& n){
        bign R;
        R.len=n.len+1;
        R.d[n.len]=1;
        bign R_mod_n=R%n;
        return (R_mod_n*R_mod_n)%n;
    }

    bign mont_pow(bign a,bign b,bign n, uint64_t n_prime, const bign&
R2_mod_n){
        if(n.len==1 && n.d[0]==1) return zero;
        if(b.len==1 && b.d[0]==0) return one;
        bign a_mont=mont_mul(a,R2_mod_n,n,n_prime);
```

```
        const int w=5;
        bign precomp[1<<w];
        precomp[0]=mont_mul(one,R2_mod_n,n,n_prime);
        precomp[1]=a_mont;
        for(int i=2;i<(1<<w);++i){
            precomp[i]=mont_mul(precomp[i-1],a_mont,n,n_prime);
        }
        bign ans=precomp[0];
        int bit_len=b.len*64-__builtin_clzll(b.d[b.len-1]);
        int i=bit_len-1;
        while(i>=0){
            if(((b.d[i/64]>>(i%64))&1)==0){
                ans=mont_mul(ans,ans,n,n_prime);
                i--;
            }
            else{
                int s=max(0,i-w+1);
                while(s<i && (((b.d[s/64]>>(s%64))&1)==0)) s++;
                int value=0;
                for(int j=i;j>=s;--j){
                    value=(value<<1) | ((b.d[j/64]>>(j%64)) & 1);
                }
                for(int                                    j=0;j<(i-s+1);++j)
ans=mont_mul(ans,ans,n,n_prime);
                ans=mont_mul(ans,precomp[value],n,n_prime);
                i=s-1;
            }
        }
        bign result=mont_mul(ans,one,n,n_prime);
        return result;
    }
    void ex_gcd(bign a,bign b,bign& x,bign& y){
        bign r0=a,r1=b;
        bign s0(1),s1(0),t0(0),t1(1);
        while(r1.len>1 || r1.d[0]!=0){
            pair<bign,bign> qr=bign::div_mod(r0, r1);
            bign q=qr.first,r2=qr.second;
            bign s2=s0-q*s1;
            bign t2=t0-q*t1;
            r0=r1;
            r1=r2;
            s0=s1;
            s1=s2;
            t0=t1;
```

```
            t1=t2;
        }
        x=s0;
        y=t0;
    }
    uint8_t buf[70000],bbuf[64];
    uint8_t seed[32];
    /*
    uint8_t
seed[32]={0x6E,0xFB,0x67,0x55,0xC5,0xF6,0xC5,0x95,0xC6,0x18,0xF8,0xC9,0
x3A,0x86,0xB4,0x71,

0x2A,0x64,0x60,0x9E,0x8F,0x22,0x08,0x2A,0x9C,0xA2,0xBC,0xAE,0x7E,0x77,0
x9B,0xBB};
    */
    int main(){
        #ifdef _WIN32
        setmode(fileno(stdin), O_BINARY);
        setmode(fileno(stdout), O_BINARY);
        #endif
        fread(buf,1,576,fp);
        bign p=bign::from_bytes(buf,256);
        bign q=bign::from_bytes(buf+256,32);
        bign g=bign::from_bytes(buf+288,256);
        bign x=bign::from_bytes(buf+544,32);
        uint64_t n_prime=get_n_prime(p);
        bign R2_mod_p=get_R2_mod_n(p);
        uint16_t len;
        fread(&len,sizeof(uint16_t),1,fp);
        fread(buf,1,len,fp);
        H(buf,len);
        uint8_t tttmp[32];
        for(int i=0;i<=7;++i){
            uint32_t ttmp=_hash[i];
            for(int j=0;j<=3;++j){
                tttmp[(i<<2)+3-j]=ttmp;
                ttmp>>=8;
            }
        }
        bign Hm=bign::from_bytes(tttmp,32);
        bign r(0),s(0);
        do{
            for(int i=0;i<32;i+=8){
                unsigned long long t;
```

```
            while(_rdrand64_step(&t)==0) ;//wait for random
            memcpy(seed+i,&t,8);
        }
        bign k=bign::from_bytes(seed,32);
        bign k_inv,y;
        ex_gcd(k,q,k_inv,y);
        r=mont_pow(g,k,p,n_prime,R2_mod_p);
        r%=q;
        s=Hm+x*r;
        s*=k_inv;
        s%=q;
        r.to_bytes(bbuf,32);
        s.to_bytes(bbuf+32,32);
    }while(r==0 || s==0);
    fwrite(bbuf,1,64,wp);
    return 0;
}
```

对于验证部分，前面相同，main 函数如下：

```
int main(){
    #ifdef _WIN32
    setmode(fileno(stdin), O_BINARY);
    setmode(fileno(stdout), O_BINARY);
    #endif
    fread(buf,1,864,fp);
    bign p=bign::from_bytes(buf,256);
    bign q=bign::from_bytes(buf+256,32);
    bign g=bign::from_bytes(buf+288,256);
    bign y=bign::from_bytes(buf+544,256);
    bign r=bign::from_bytes(buf+800,32);
    bign s=bign::from_bytes(buf+832,32);
    uint64_t n_prime=get_n_prime(p);
    bign R2_mod_p=get_R2_mod_n(p);
    uint16_t len;
    fread(&len,sizeof(uint16_t),1,fp);
    fread(buf,1,len,fp);
    H(buf,len);
    uint8_t tttmp[32];
    for(int i=0;i<=7;++i){
        uint32_t ttmp=_hash[i];
        for(int j=0;j<=3;++j){
            tttmp[(i<<2)+3-j]=ttmp;
            ttmp>>=8;
        }
    }
```

```
        //fwrite(tttmp,1,32,wp);
        bign Hm=bign::from_bytes(tttmp,32);
        bign s_inv,tmp0,tmp1;
        ex_gcd(s,q,s_inv,tmp0);
        bign e1=Hm*s_inv%q;
        bign e2=r*s_inv%q;
        tmp0=mont_pow(g,e1,p,n_prime,R2_mod_p);
        tmp1=mont_pow(y,e2,p,n_prime,R2_mod_p);
        if((tmp0*tmp1%p)%q==r){
            bbuf[0]=0x50;
            bbuf[1]=0x41;
            bbuf[2]=0x53;
            bbuf[3]=0x53;
        }
        else{
            bbuf[0]=0x46;
            bbuf[1]=0x41;
            bbuf[2]=0x49;
            bbuf[3]=0x4C;
        }
        fwrite(bbuf,1,4,wp);
        return 0;
    }
```

五、实验结果

对于签名部分，如下图所示，选用实验 7-1 中样例#1 作为输入二进制数据，去掉随机化 seed（k）的过程，而是用 k=6E FB 67 55 C5 F6 C5 95 C6 18 F8 C9 3A 86 B4 71 2A 64 60 9E 8F 22 08 2A 9C A2 BC AE 7E 77 9B BB 代替，运行得到结果（保存在 outbut.bin 中）如下：

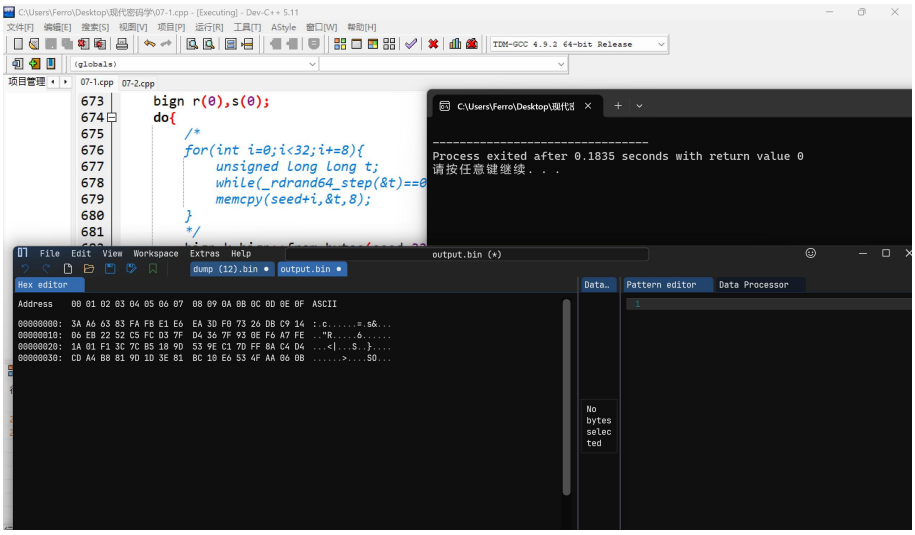

图 1 DSA 签名结果（例）

结果与输出#1 进行对比，发现完全正确，这可以验证签名算法的正确性。在 OJ 上提交记录如下：

提交 ID	用户名	提交时间	语言	代码长度	评测结果
3535	23336262	2025-12-25 15:37:13	C++	23404	7.07 ms / 512 KB / Accept 8/8
3534	23336262	2025-12-25 15:35:33	C++	23398	10.213 ms / 668 KB / Accept 8/8
3532	23336262	2025-12-25 11:39:10	C++	23664	8.128 ms / 512 KB / Accept 8/8
3531	23336262	2025-12-25 11:34:17	C++	23665	7.586 ms / 512 KB / Accept 8/8
3341	23336262	2025-12-18 09:27:53	C++	24320	12.132 ms / 512 KB / Accept 8/8
3268	23336262	2025-12-17 15:41:05	C++	24317	7.929 ms / 596 KB / Accept 8/8

图 2 DSA 签名评测结果

对于验证部分，采用实验 7-2 中#1 样例作为检验，结果如下：

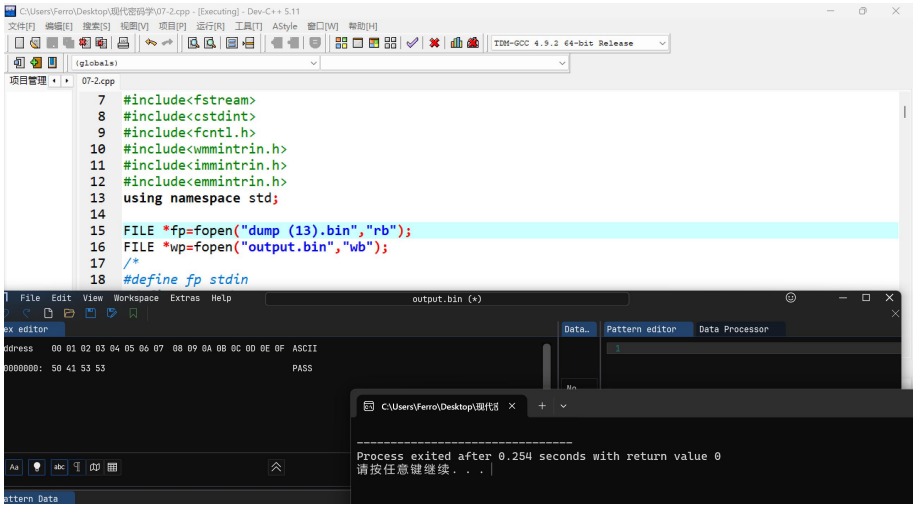

图 3 验证 DSA 签名的有效性

可以看到二进制数据对应的 ASCII 码是“PASS”，验证通过。

在 OJ 上提交结果如下：

提交 ID	用户名	提交时间	语言	代码长度	评测结果
3536	23336262	2025-12-25 15:42:48	C++	23441	14.627 ms / 768 KB / Accept 16/16
3530	23336262	2025-12-25 11:29:49	C++	23706	13.874 ms / 644 KB / Accept 16/16
3405	23336262	2025-12-20 09:31:06	C++	24341	16.141 ms / 512 KB / Accept 16/16

图 4 DSA 验证评测结果

六、思考题

1. 针对随机数 k 的攻击

针对随机数 k 的攻击主要包括共享 k 攻击和线性 k 攻击，其中共享 k 攻击是利用同一个随机数 k 对不同的消息进行签名的漏洞恢复出 k 从而进一步得到私钥（网站上的思考题就是共享 k 攻击的例子），线性 k 攻击则是利用不同消息中 k 存在线性关系这一特征进行密钥恢复。

当使用同一组公开参数(p,q,g)和私钥 x 对消息 m1 和 m2 产生两个签名(r1,s1)和(r2,s2)的输入和输出时，假如 r1==r2（记为 r），说明随机数 k 是共享的（这是因为 r 和 k 是模 q 意义下的一一映射，网站上的思考题就是这种情况）。利用 $s1=(H(m1)+xr)k^{-1} \bmod q$ 得到 $ks1=H(m1)+xr \bmod q$，同理 $ks2=H(m2)+xr \bmod q$。两式相减得到 $k(s2-s1)=H(m2)-H(m1)$，从而 $k=(H(m2)-H(m1))*(s2-s1)^{-1}$。恢复出 k 之后，任意代入 s1 或 s2，可以得到 $x=(ks1-H(m1))r^{-1} \bmod q$，从而获取出私钥。所以用同一个随机数对不同消息进行签名是非常危险的。事实上，索尼曾经就犯过这样的错误，并且遭到了攻击[3]。

回到网站思考题的例子上，把每组对应的输入输出都用 bin 文件读入，执行上述恢

复密钥算法即可。具体代码如下（main 函数）：

```
int main(){
    #ifdef _WIN32
    setmode(fileno(stdin), O_BINARY);
    setmode(fileno(stdout), O_BINARY);
    #endif
    fread(buf,1,576,fp);
    bign p=bign::from_bytes(buf,256);
    bign q=bign::from_bytes(buf+256,32);
    bign g=bign::from_bytes(buf+288,256);
    bign x_false=bign::from_bytes(buf+544,32);
    uint16_t len;
    fread(&len,sizeof(uint16_t),1,fp);
    fread(buf,1,len,fp);
    H(buf,len);
    uint8_t tttmp[32];
    for(int i=0;i<=7;++i){
        uint32_t ttmp=_hash[i];
        for(int j=0;j<=3;++j){
            tttmp[(i<<2)+3-j]=ttmp;
            ttmp>>=8;
        }
    }
    //fwrite(tttmp,1,32,wp);
    bign Hm1=bign::from_bytes(tttmp,32);
    fread(buf,1,576,fp2);
    fread(&len,sizeof(uint16_t),1,fp2);
    fread(buf,1,len,fp2);
    H(buf,len);
    for(int i=0;i<=7;++i){
        uint32_t ttmp=_hash[i];
        for(int j=0;j<=3;++j){
            tttmp[(i<<2)+3-j]=ttmp;
            ttmp>>=8;
        }
    }
    bign Hm2=bign::from_bytes(tttmp,32);
    fread(buf,1,64,fp3);
    bign r1=bign::from_bytes(buf,32);
    bign s1=bign::from_bytes(buf+32,32);
    fread(buf,1,64,fp4);
    bign r2=bign::from_bytes(buf,32);
    bign s2=bign::from_bytes(buf+32,32);
```

```
        if(r1==r2){
            bign delta_s=s2-s1,inv_delta_s,inv_r,k_inv;
            bign y;
            ex_gcd(delta_s,q,inv_delta_s,y);
            bign k=(Hm2-Hm1)*inv_delta_s%q;
            cout<<"k: ";
            k.print();
            k.to_bytes(buf,32);
            ex_gcd(r1,q,inv_r,y);
            bign x=(s1*k-Hm1)*inv_r%q;
            cout<<"x: ";
            x.print();
            x.to_bytes(buf+32,32);
            fwrite(buf,1,64,wp);
            //validate x
            ex_gcd(k,q,k_inv,y);
            if(s2==(Hm2+x*r2)*k_inv%q) cout<<"pass";
        }
        return 0;
    }
```

这里为了验证得到私钥的正确性，加入了检验过程，即检验恢复出的 k 和利用第一组（r,s）恢复的 x 是否可以正确的产生第二组的 s。运行，得到结果如下：

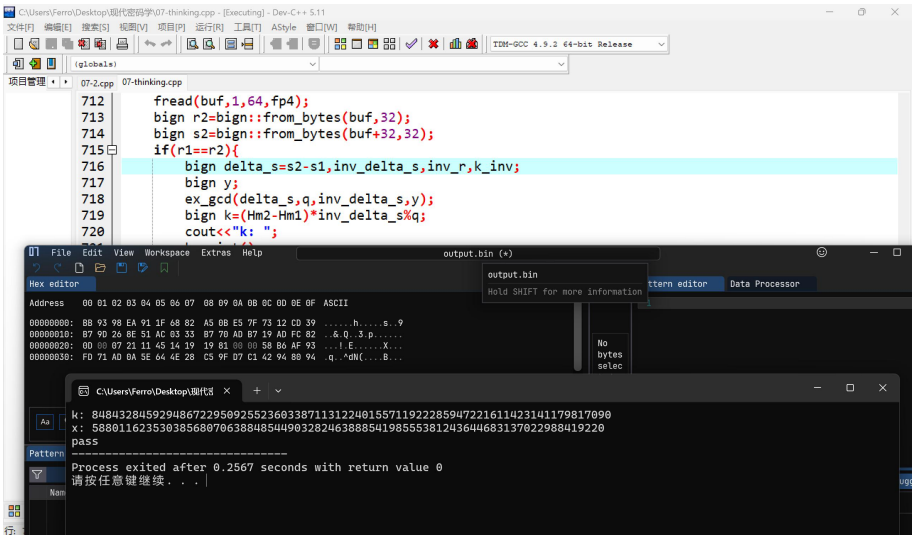

图 5 思考题-私钥 x 的恢复

可以看到检验通过，恢复出来的 x 的十六进制大端表示为 0D 00 **07 21 11 45 14 19 19 81 0**0 00 58 B6 AF 93 FD 71 AD 0A 5E 64 4E 28 C5 9F D7 C1 42 94 80 94。（~~从加粗的字体也可以知道大概率恢复正确了~~），十进制下表示为：5880116235303856807063884854490328246388854198555381243644683137022988419220。

至此，共享密钥 k 攻击的部分宣告完成。

线性 k 攻击的思想如下：若使用不同的消息 k1、k2 对消息 m1、m2 签名，但是 k1、k2 存在线性关系：k2=a*k1+b（其中 a、b 已知），则 k 也会被泄露。共享密钥 k 相当于是 a=1，b=0 的特例。和共享密钥 k 攻击的推导类似，可以得到 $k=(H(m2)r1-H(m1)r2-bs2r1)*(as2r1-s1r2)^{-1} \bmod q$，再代入 s 同样可以恢复私钥 x。

七、实验总结

本次实验（DSA 签名与验证）是现代密码学实验平时实验的最后一个实验。DSA 算法的安全性基于离散对数问题，使得攻击者难以伪造数字签名。实际应用中，DSA 算法广泛应用于电子签名、身份验证、数据完整性校验等领域。例如，在电子商务中，商家可以使用 DSA 算法为电子合同或订单生成数字签名，以确保合同的有效性和数据的完整性。在网络安全领域，DSA 算法可以用于验证软件更新、操作系统启动等关键操作的真实性和完整性。本次实验通过 DSA 基本的签名、验证算法实现，加深了对公钥密码学中数字签名的理解，也巩固了大数运算和哈希函数。在思考题中展示了关于随机数的两种攻击，事实上在随机数的选取中，对于不同消息的随机数不能存在线性关系，才能保证签名的安全性。

八、参考文献

【1】刘建华主编；孙韩林副主编. 物联网安全. 中国铁道出版社. 2013.09. 第 68 页

【2】Xu, L., Dai, Z., Wu, B., & Lin, D. (2023). Improved Attacks on (EC)DSA with Nonce Leakage by Lattice Sieving with Predicate. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2023(2), 568-586. https://doi.org/10.46586/tches.v2023.i2.568-586

【3】Sony PS3 attack，https://www.theguardian.com/technology/gamesblog/2011/jan/07/playstation-3-hack-ps3