DALISTUDIO.NET

ECDSA 签名与验证

基于 OpenSSL 库

Dali Wang <wangdali@qq.com>

2014/1/1

目录

一、密钥对的生成

1、生成密钥对

①首先声明 EC_KEY *ec_key; 结构，椭圆曲线的参数；私钥和公钥都保存在这个结构中。

②声明 EC_GROUP *ec_group; 结构，这个结构保存着椭圆曲线的参数。

③使用 ec_key = EC_KEY_new(); 生成一个新的 EC_KEY 结构。

④使用下面代码选择一条曲线参数，填充 EC_GROUP 结构：

ec_group = EC_GROUP_new_by_curve_name(NID_secp256k1);

NID_secp256k1 为椭圆曲线，在 obj.mac.h 中定义。

⑤使用 int ret = EC_KEY_set_group(ec_key,ec_group); 将 EC_GROUP 结构的内容填充到 EC_KEY 结构中。

⑥使用 int ret = EC_KEY_generate_key(ec_key); 生成私钥和公钥对，并填充到 EC_KEY 结构中。

通过上面六个步骤，EC_KEY 结构已经填充完成，可以用于签名和验证了。

2、导出私钥

使用 int len = i2d_ECPrivateKey(ec_key,&PrivateKey); 将 ec_key 结构中的私钥数据导出到 PrivateKey 指向的空间。

len 返回私钥数据的长度。导出的私钥数据包含有曲线参数，公

钥和私钥的所有数据。

在导出私钥的时候可以使用 AES256 或 RC4 等对称算法加密，以保证私钥的安全，每次使用私钥时，必须要输入密码解密后才可以使用。

3、导出公钥

使用 int len = i2o_ECPublicKey(ec_key,&PublicKey); 将 ec_key 结构中的公钥数据导出到 PublicKey 指向的空间。

len 返回公钥数据的长度。

4、生成密钥源代码

```
#include <stdio.h>
#include <openssl/crypto.h>
#include <openssl/evp.h>
#include <openssl/ecdsa.h>

#define PRIVKEY   "static unsigned char privkey[%d] = {"
#define PUBKEY    "static const unsigned char pubkey[%d] = {"
#define ENDKEY    "\n};\n"

int main(int argc, char* argv[])
{
```

```
    EC_KEY *ec_key;
    EC_GROUP *ec_group;
    unsigned char buf[1024];
    unsigned char *pp;
    int i,len;

    if ((ec_key = EC_KEY_new()) == NULL)
    {
        printf("Error: EC_KEY_new()\n");
        return 0;
    }

    /* 选择一条椭圆曲线 */
   if ((ec_group = EC_GROUP_new_by_curve_name(NID_secp256k1))
== NULL)
    {
        printf("Error: EC_GROUP_new_by_curve_name()\n");
        EC_KEY_free(ec_key);
        return -1;
    }

    /* 设置密钥参数 */
```

```
int ret;
ret = EC_KEY_set_group(ec_key,ec_group);
if(ret!=1)
{
   printf("Error: EC_KEY_set_group()\n");
   return -1;
}

/* 生成密钥对 */
if (!EC_KEY_generate_key(ec_key))
{
   printf("Error: EC_KEY_generate_key()\n");
   EC_KEY_free(ec_key);
   return -1;
}

/* 导出私钥 */
pp = buf;
len = i2d_ECPrivateKey(ec_key,&pp);
if (!len)
{
   printf("Error: i2d_ECPrivateKey()\n");
```

```
    EC_KEY_free(ec_key);
    return -1;
}

printf(PRIVKEY,len);
for (i=0; i<len; i++)
{
    if ( !(i % 8) )
        printf("\n");
    if(i==len-1)
        printf("0x%02X ",buf[i]);
    else
        printf("0x%02X , ",buf[i]);
}
printf(ENDKEY);


/* 导出公钥 */
pp = buf;
len = i2o_ECPublicKey(ec_key,&pp);
if (!len)
{
```

```
        printf("Error: i2o_ECPublicKey()\n");
        EC_KEY_free(ec_key);
        return -1;
    }

    printf(PUBKEY,len);
    for (i=0; i<len; i++)
    {
        if ( !(i % 8) )
            printf("\n");
        if(i==len-1)
            printf("0x%02X ",buf[i]);
        else
            printf("0x%02X , ",buf[i]);
    }
    printf(ENDKEY);

    EC_KEY_free(ec_key);
    return 0;
}
```

二、对数据的签名

1、导入私钥

使用 EC_KEY *ec_key = NULL; 定义一个空结构，然后使用下面代码导入私钥：

```
ec_key = d2i_ECPrivateKey(&ec_key, & PrivateKey, sizeof(PrivateKey));
```

2、数据签名

使用 ECDSA_sign(0, data, data_len, sign, &sign_len, ec_key); 对数据 data 进行签名，并将签名返回到 sign 中。通常这里的 data 是要签名数据的散列值，可以使用 SHA1 、 SHA256 等散列算法。

3、数据签名源代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <openssl/crypto.h>
#include <openssl/evp.h>
#include <openssl/ecdsa.h>
#include <openssl/sha.h>
```

```
#define SIGN   "static unsigned char signature[%d] = {"
#define ENDKEY   "\n};\n"

#define MAXSIGLEN  128


/* 私钥数据 */
static unsigned char privkey[279] = {
0x30 , 0x82 , 0x01 , 0x13 , 0x02 , 0x01 , 0x01 , 0x04 ,
0x20 , 0xE8 , 0x01 , 0x44 , 0xD9 , 0x98 , 0x71 , 0x41 ,
0x54 , 0x2B , 0x4D , 0xDF , 0x50 , 0x7E , 0x4D , 0xB3 ,
0xCA , 0x5F , 0x30 , 0x39 , 0xA0 , 0x51 , 0x82 , 0x76 ,
0x39 , 0xFF , 0xC4 , 0x63 , 0x38 , 0x0E , 0xDB , 0x2A ,
0xB9 , 0xA0 , 0x81 , 0xA5 , 0x30 , 0x81 , 0xA2 , 0x02 ,
0x01 , 0x01 , 0x30 , 0x2C , 0x06 , 0x07 , 0x2A , 0x86 ,
0x48 , 0xCE , 0x3D , 0x01 , 0x01 , 0x02 , 0x21 , 0x00 ,
0xFF , 0xFF , 0xFF , 0xFF , 0xFF , 0xFF , 0xFF , 0xFF ,
0xFF , 0xFF , 0xFF , 0xFF , 0xFF , 0xFF , 0xFF , 0xFF ,
0xFF , 0xFF , 0xFF , 0xFF , 0xFF , 0xFF , 0xFF , 0xFF ,
0xFF , 0xFF , 0xFF , 0xFE , 0xFF , 0xFF , 0xFC , 0x2F ,
0x30 , 0x06 , 0x04 , 0x01 , 0x00 , 0x04 , 0x01 , 0x07 ,
0x04 , 0x41 , 0x04 , 0x79 , 0xBE , 0x66 , 0x7E , 0xF9 ,
```

```
0xDC , 0xBB , 0xAC , 0x55 , 0xA0 , 0x62 , 0x95 , 0xCE ,
0x87 , 0x0B , 0x07 , 0x02 , 0x9B , 0xFC , 0xDB , 0x2D ,
0xCE , 0x28 , 0xD9 , 0x59 , 0xF2 , 0x81 , 0x5B , 0x16 ,
0xF8 , 0x17 , 0x98 , 0x48 , 0x3A , 0xDA , 0x77 , 0x26 ,
0xA3 , 0xC4 , 0x65 , 0x5D , 0xA4 , 0xFB , 0xFC , 0x0E ,
0x11 , 0x08 , 0xA8 , 0xFD , 0x17 , 0xB4 , 0x48 , 0xA6 ,
0x85 , 0x54 , 0x19 , 0x9C , 0x47 , 0xD0 , 0x8F , 0xFB ,
0x10 , 0xD4 , 0xB8 , 0x02 , 0x21 , 0x00 , 0xFF , 0xFF ,
0xFF , 0xFF , 0xFF , 0xFF , 0xFF , 0xFF , 0xFF , 0xFF ,
0xFF , 0xFF , 0xFF , 0xFF , 0xFF , 0xFE , 0xBA , 0xAE ,
0xDC , 0xE6 , 0xAF , 0x48 , 0xA0 , 0x3B , 0xBF , 0xD2 ,
0x5E , 0x8C , 0xD0 , 0x36 , 0x41 , 0x41 , 0x02 , 0x01 ,
0x01 , 0xA1 , 0x44 , 0x03 , 0x42 , 0x00 , 0x04 , 0x68 ,
0x45 , 0x28 , 0xB0 , 0x36 , 0xEA , 0x02 , 0x46 , 0x0A ,
0x4E , 0x24 , 0x43 , 0x28 , 0x97 , 0x61 , 0xC5 , 0xE0 ,
0x30 , 0xE7 , 0xAE , 0x79 , 0x5A , 0xFE , 0x27 , 0x9D ,
0xF2 , 0x76 , 0xD8 , 0xAC , 0x97 , 0x6E , 0x1C , 0x73 ,
0x6E , 0x42 , 0x40 , 0x36 , 0x00 , 0x8A , 0x01 , 0x2D ,
0xBD , 0xF2 , 0x9A , 0x68 , 0x83 , 0x24 , 0xA1 , 0x6F ,
0x77 , 0xD0 , 0x4E , 0xE1 , 0x2D , 0x07 , 0x33 , 0xE6 ,
0xCE , 0x24 , 0x61 , 0xAE , 0xF3 , 0xCB , 0x38
};
```

```
/* 签名函数 */
static unsigned int sign(unsigned char *sig,const unsigned
char *buf,int len)
{
   unsigned int sign_len = MAXSIGLEN;
   EC_KEY *ec_key = NULL;
   unsigned char *pp = (unsigned char*)privkey;

   /* 导入私钥 */
   ec_key = d2i_ECPrivateKey(&ec_key, (const unsigned
char**)&pp, sizeof(privkey));
   memset(privkey,0,sizeof(privkey));
   if ( ec_key == NULL)
   {
      printf("Error: d2i_ECPrivateKey()\n");
      return -1;
   }

   /* 数据签名 */
   if (!ECDSA_sign(0,buf, len, sig,&sign_len,ec_key))
   {
```

```
        printf("Error: ECDSA_sign()\n");
        EC_KEY_free(ec_key);
        return -1;
    }
    EC_KEY_free(ec_key);
    return sign_len;
}

/* 主函数 */
int main(int argc, char* argv[]) {
    const char message[] = "wangdali";
    unsigned char *signature,digest[32]={};
    unsigned int  dgst_len = 0;

    EVP_MD_CTX md_ctx;
    EVP_MD_CTX_init(&md_ctx);
    EVP_DigestInit(&md_ctx, EVP_sha256()); // 散列算法
    EVP_DigestUpdate(&md_ctx, (const
void*)message,sizeof(message));
    EVP_DigestFinal(&md_ctx, digest, &dgst_len);

    signature=(unsigned char *)malloc(MAXSIGLEN);
```

```
    int len = sign(signature,(unsigned
char*)&digest,dgst_len);

    int i=0;
    printf(SIGN,len);
    for (i=0; i<len; i++)
    {
       if ( !(i % 8) )
          printf("\n");
       if(i==len-1)
          printf("0x%02X ",signature[i]);
       else
          printf("0x%02X , ",signature[i]);
    }
    printf(ENDKEY);
    return 0;
}
```

三、对签名的验证

1、导入公钥

首先声明 EC_KEY *ec_key = NULL; 和 EC_GROUP *ec_group; 生成两个空结构。

通过下面代码填充 EC_GROUP 结构：

```
ec_group = EC_GROUP_new_by_curve_name(NID_secp256k1);
```

使用 int ret=EC_KEY_set_group(ec_key, ec_group); 将 ec_group 的内容填充到 ec_key 结构中。

使用下面代码将公钥数据导入 ec_key 结构中：

```
ec_key = o2i_ECPublicKey(&ec_key, & publickey,sizeof(publickey));
```

2、验证签名

使用下面代码对签名进行验证：

```
int ret = ECDSA_verify(0, data, data_len, sign, sign_len, ec_key);
```

验证 data 数据，这个数据和签名时使用的数据要一样，可以将数据进行散列，sign 为签名的数据，导入这两个数据即可进行签名的验证。ret 返回 1 为验证通过，其他为验证失败。

3、验证签名源代码

```
#include <openssl/crypto.h>
#include <openssl/evp.h>
```

```
#include <openssl/ecdsa.h>
#include <openssl/sha.h>

#define MAXSIGLEN  128

/* 公钥 */
static const unsigned char pubkey[65] = {
0x04 , 0x68 , 0x45 , 0x28 , 0xB0 , 0x36 , 0xEA , 0x02 ,
0x46 , 0x0A , 0x4E , 0x24 , 0x43 , 0x28 , 0x97 , 0x61 ,
0xC5 , 0xE0 , 0x30 , 0xE7 , 0xAE , 0x79 , 0x5A , 0xFE ,
0x27 , 0x9D , 0xF2 , 0x76 , 0xD8 , 0xAC , 0x97 , 0x6E ,
0x1C , 0x73 , 0x6E , 0x42 , 0x40 , 0x36 , 0x00 , 0x8A ,
0x01 , 0x2D , 0xBD , 0xF2 , 0x9A , 0x68 , 0x83 , 0x24 ,
0xA1 , 0x6F , 0x77 , 0xD0 , 0x4E , 0xE1 , 0x2D , 0x07 ,
0x33 , 0xE6 , 0xCE , 0x24 , 0x61 , 0xAE , 0xF3 , 0xCB ,
0x38
};

 /* 验证函数 */
static int verify(const unsigned char *sig,int siglen,const
unsigned char *buf,int buflen)
{
```

```
    int ret;
    EC_KEY *ec_key = NULL;
    EC_GROUP *ec_group;
    unsigned char *pp = (unsigned char*)pubkey;
    if ((ec_key = EC_KEY_new()) == NULL)
    {
        printf("Error: EC_KEY_new()\n");
        return -1;
    }
   if ((ec_group = EC_GROUP_new_by_curve_name(NID_secp256k1))
== NULL)
    {
        printf("Error: EC_GROUP_new_by_curve_name()\n");
        EC_KEY_free(ec_key);
        return -1;
    }

    /* 设置密钥参数 */
    ret=EC_KEY_set_group(ec_key,ec_group);
    if(ret!=1)
    {
        printf("Error: EC_KEY_set_group\n");
```

```
        EC_KEY_free(ec_key);
        return -1;
    }

    /* 导入公钥 */
    ec_key = o2i_ECPublicKey(&ec_key,(const unsigned
char**)&pp,sizeof(pubkey));
    if (ec_key == NULL)
    {
        printf("Error: o2i_ECPublicKey\n");
        EC_KEY_free(ec_key);
        return 0;
    }

    /* 验证签名 */
  ret = ECDSA_verify(0,(const unsigned char*)buf, buflen, sig,
siglen,ec_key);
    EC_KEY_free(ec_key);
    return ret == 1 ? 1 : 0;
}
```

```
/* 主函数 */
int main(int argc, char* argv[])
{
    const char message[] = "wangdali";
    unsigned char digest[32]={};
    unsigned int  dgst_len = 0;

    EVP_MD_CTX md_ctx;
    EVP_MD_CTX_init(&md_ctx);
    EVP_DigestInit(&md_ctx, EVP_sha256()); // 散列算法
    EVP_DigestUpdate(&md_ctx, (const
void*)message,sizeof(message));
    EVP_DigestFinal(&md_ctx, digest, &dgst_len);

static unsigned char signature[72] = {
0x30 , 0x46 , 0x02 , 0x21 , 0x00 , 0xC4 , 0x2D , 0xE1 ,
0x99 , 0xC4 , 0xF4 , 0xA5 , 0x91 , 0x14 , 0x63 , 0x06 ,
0x75 , 0xCC , 0x72 , 0xBC , 0x1F , 0x8B , 0xA4 , 0x4B ,
0x68 , 0x78 , 0xCF , 0xBB , 0xE2 , 0xCD , 0x39 , 0xE0 ,
0xA9 , 0xE2 , 0xC8 , 0xBA , 0xB7 , 0x02 , 0x21 , 0x00 ,
0xE7 , 0x6E , 0x45 , 0x2C , 0x1B , 0x71 , 0x8F , 0xE5 ,
0x9E , 0xA3 , 0x65 , 0xF8 , 0x22 , 0xD8 , 0x1F , 0xA7 ,
```

```
0x3C , 0x08 , 0x62 , 0x57 , 0x33 , 0xD5 , 0xE8 , 0x08 ,
0xD0 , 0xC2 , 0x85 , 0x50 , 0xEA , 0x48 , 0x7A , 0xD7
};
    int ret = verify((const unsigned char
*)&signature,sizeof(signature),(const unsigned char
*)&digest,dgst_len);
    if(ret==1)
    {
        printf("Verify: OK\n");
    }
    else
    {
        printf("Verify: Error\n");
    }

    return 0;
}
```