第3章 DES和RSA加密算法的程序实现

3.1 实验目的

- 掌握密码学的基本概念、对称密钥加密和公钥加密技术。
- 掌握DES算法的原理，用程序实现。
- 掌握RSA算法的原理，用程序实现。
- 比较DES和RSA算法，理解这两种加密方式。

3.2 实验环境

实验主机操作系统为Windows 7，安装VS编译器。

3.3 实验工具

- Visual Studio：Microsoft Visual Studio是VS的全称。VS是美国微软公司的开发工具包系列产品。VS是一个基本完整的开发工具集，它包括了整个软件生命周期中所需要的大部分工具，如UML工具、代码管控工具、集成开发环境(IDE)等。所写的目标代码适用于微软支持的所有平台。
- 也可以使用其他工具。

3.4 实验内容

3.4.1 实验原理

1. 密码学概述

密码是实现秘密通信的主要手段，是隐蔽语言、文字、图像的特种符号。用特种符号

按照通信双方约定的方法把电文的原形隐蔽起来，不为第三者所识别的通信方式称为密码通信。在计算机通信中，采用密码技术将信息隐蔽起来，再将隐蔽后的信息传输出去，信息在传输过程中即使被窃取或截获，窃取者也不能了解信息的内容，从而保证信息传输的安全。

任何一个加密系统至少包括下面 4 个组成部分。

① 未加密的报文，也称明文。

② 加密后的报文，也称密文。

③ 加密解密设备或算法。

④ 加密解密的密钥。

发送方用加密密钥，通过加密设备或算法，将信息加密后发送出去。接收方在收到密文后，用解密密钥将密文解密，恢复成明文。如果传输中有人窃取，他只能得到无法理解的密文，从而对信息起到保密作用。

2. DES 对称加密技术

基于密钥的算法通常有两类：对称算法和公开密钥算法。在大多数对称算法中，加解密的密钥是相同的。对称算法要求发送者和接收者在安全通信之前协商一个密钥。对称算法的安全性依赖于密钥，泄露密钥就意味着任何人都能对消息进行加密。

DES 是 Data Encryption Standard(数据加密标准)的缩写。它是由 IBM 公司研制的一种对称密码算法，美国国家标准局于 1977 年公布把它作为非机要部门使用的数据加密标准。DES 一直活跃在国际保密通信的舞台上，扮演了十分重要的角色。

DES 是一个分组加密算法，典型的 DES 以 64 位为分组对数据加密，加密和解密用的是同一个算法。它的密钥长度是 56 位(因为每个第 8 位都用作奇偶校验)，密钥可以是任意的 56 位的数，而且可以随时改变。其中有极少数被认为是易破解的弱密钥，但是很容易避开它们不用，所以保密性依赖于密钥。

(1) DES 加密算法的框架

首先要生成一套加密密钥，从用户处取得一个 64 位长的密码口令，然后通过等分、移位、选取和迭代形成一套 16 个加密密钥，分别供每一轮运算中使用。

DES 对 64 位(bit)的明文分组 M 进行操作，M 经过一个初始置换 IP，置换成 m_0。将 m_0 明文分成左半部分和右半部分 $m_0 = (L_0, R_0)$，各 32 位长。然后进行 16 轮完全相同的运算(迭代)，这些运算被称为函数 f，在每一轮运算过程中数据与相应的密钥结合。

在每一轮中，密钥位移位，然后再从密钥的 56 位中选出 48 位。通过一个扩展置换将数据的右半部分扩展成 48 位，并通过一个异或操作替代成新的 48 位数据，再将其压缩置换成 32 位。这 4 步运算构成了函数 f。然后，通过另一个异或运算，函数 f 的输出与左半部分结合，其结果成为新的右半部分，原来的右半部分成为新的左半部分。将该操作重复 16 次。

经过 16 轮迭代后，左、右两半部分合在一起经过一个末置换(数据整理)，这样就完成了加密过程。

加密流程如图 3-1 所示。

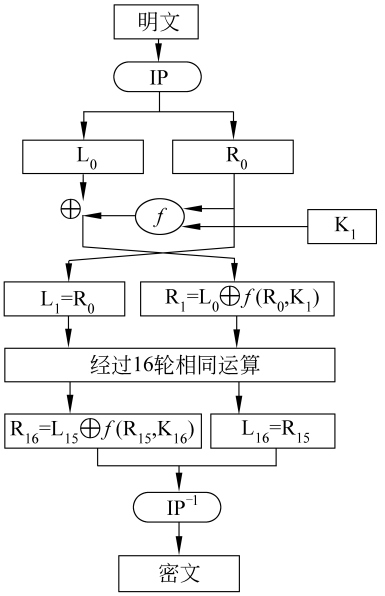

图 3-1　加密流程

（2）DES 解密过程

了解加密过程中所有的代替、置换、异或和循环迭代之后，读者也许会认为，解密算法应该是加密的逆运算，与加密算法完全不同。恰恰相反，经过密码学家精心设计选择的各种操作，DES 获得了一个非常有用的性质：加密和解密使用相同的算法。

DES 加密和解密唯一的不同是密钥的次序相反。如果各轮加密密钥分别是 $K_1, K_2, K_3, \cdots, K_{16}$，那么解密密钥就是 $K_{16}, K_{15}, K_{14}, \cdots, K_1$。

3. RSA 公钥加密技术

公开密钥密码体制就是使用不同的加密密钥与解密密钥，RSA 公开加密密钥体制是一种基于大数不可能质因数分解假设的公钥体系，在公开密钥密码体制中，加密密钥 PK 是公开信息，而解密密钥 SK 是需要保密的。加密算法 E 和解密算法 D 也是公开的。虽然秘密密钥 SK 是由公开密钥 PK 决定的，但却不能根据 PK 计算出 SK。

RSA 体制可以简单描述如下。

① 生成两个大素数 p 和 q。

② 计算这两个素数的乘积 $n=pq$。

③ 计算小于 n 并且与 n 互质的整数的个数，即 $\varphi(n)=(p-1)(q-1)$。

④ 选择一个随机数 b 满足 $1<b<\varphi(n)$，并且 b 和 $\varphi(n)$ 互质，即 $\gcd(b,\varphi(n))=1$。

⑤ 计算 $ab=1 \bmod \varphi(n)$。

⑥ 保密 a、p 和 q，公开 n 和 b。

利用 RSA 加密时，明文以分组的方式加密：每一组的比特数应该小于 $\log_2 n$ 比特。加密明文 x 时，利用公钥 (b,n) 来计算 $c=xb \bmod n$ 就可以得到相应的密文 c。解密时，通过计算 $ca \bmod n$ 就可以恢复出明文 x。

3.4.2　实验步骤

1. 实验 1——DES 算法的程序实现

```
#include"memory.h"
#include"stdio.h"

enum{ENCRYPT,DECRYPT};   //ENCRYPT:加密,DECRYPT:解密
void Des_Run(char Out[8], char In[8], bool Type =ENCRYPT);
//设置密钥
void Des_SetKey(const char Key[8]);
static void F_func(bool In[32], const bool Ki[48]);              //f 函数
```

```
static void S_func(bool Out[32], const bool In[48]);              //S 盒代替
//变换
static void Transform(bool *Out, bool *In, const char *Table, int len);
static void Xor(bool *InA, const bool *InB, int len);             //异或
static void RotateL(bool *In, int len, int lool);                 //循环左移
//字节组转换为位组
static void ByteToBit(bool *Out, const char * In, int bits);
//位组转换成字节组
static void BitToByte(char*Out, const bool *In, int bits);
//置换 IP 表
const static char IP_Table[64] ={
    58,50,42,34,26,18,10,2,60,52,44,36,28,20,12,4,
    62,54,46,38,30,22,14,6,64,56,48,40,32,24,16,8,
    57,49,41,33,25,17,9,1,59,51,43,35,27,19,11,3,
    61,53,45,37,29,21,13,5,63,55,47,39,31,23,15,7
};
//逆置换 IP 表
const static char IPR_Table[64] ={
    40,8,48,16,5624,64,32,39,7,47,15,55,23,63,31,
    38,6,46,14,54,22,62,30,37,5,45,13,53,21,61,29,
    36,4,44,12,52,20,60,28,35,3,43,11,51,19,59,27,
    34,2,42,10,50,18,58,26,33,1,41,9,49,17,57,25
};
//E 位选择表
const static char E_Table[48] ={
    32,1,2,3,4,5,4,5,6,7,8,9,
    8,9,10,11,12,13,12,13,14,15,16,17,
    16,17,18,19,20,21,20,21,22,23,24,25,
    24,25,26,27,28,29,28,29,30,31,32,1
};
//P 换位表
const static char P_Table[32] ={
    16,7,20,21,29,12,28,17,1,15,23,26,5,18,31,10,
    2,8,24,14,32,27,3,9,19,13,30,6,22,11,4,25
};
//PC1 选位表
const static char PC1_Table[56] ={
    57,49,41,33,25,17,9,1,58,50,42,34,26,18,
    10,2,59,51,43,35,27,19,11,3,60,52,44,36,
    63,55,47,39,31,23,15,7,62,54,46,38,30,22,
    14,6,61,53,45,37,29,21,13,5,28,20,12,4
};
//PC2 选位表
const static char PC2_Table[48] ={
```

```
    14,17,11,24,1,5,3,28,15,6,21,10,
    23,19,12,4,26,8,16,7,27,20,13,2,
    41,52,31,37,47,55,30,40,51,45,33,48,
    44,49,39,56,34,53,46,42,50,36,29,32
};
//左移位数表
const static char LOOP_Table[16] ={
    1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1
};
//S 盒
const static char S_Box[8][4][16] ={
    //S1
    14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7,
    0,15,7,4,14,2,13,1,10,6,12,11,9,5,3,8,
    4,1,14,8,13,6,2,11,15,12,9,7,3,10,5,0,
    15,12,8,2,4,9,1,7,5,11,3,14,10,0,6,13,
    //S2
    15,1,8,14,6,11,3,4,9,7,2,13,12,0,5,10,
    3,13,4,7,15,2,8,14,12,0,1,10,6,9,11,5,
    0,14,7,11,10,4,13,1,5,8,12,6,9,3,2,15,
    13,8,10,1,3,15,4,2,11,6,7,12,0,5,14,9,
    //S3
    10,0,9,14,6,3,15,5,1,13,12,7,11,4,2,8,
    13,7,0,9,3,4,6,10,2,8,5,14,12,11,15,1,
    13,6,4,9,8,15,3,0,11,1,2,12,5,10,14,7,
    1,10,13,0,6,9,8,7,4,15,14,3,11,5,2,12,
    //S4
    7,13,14,3,0,6,9,10,1,2,8,5,11,12,4,15,
    13,8,11,5,6,15,0,3,4,7,2,12,1,10,14,9,
    10,6,9,0,12,11,7,13,15,1,3,14,5,2,8,4,
    3,15,0,6,10,1,13,8,9,4,5,11,12,7,2,14,
    //S5
    2,12,4,1,7,10,11,6,8,5,3,15,13,0,14,9,
    14,11,2,12,4,7,13,1,5,0,15,10,3,9,8,6,
    4,2,1,11,10,13,7,8,15,9,12,5,6,3,0,14,
    11,8,12,7,1,14,2,13,6,15,0,9,10,4,5,3,
    //S6
    12,1,10,15,9,2,6,8,0,13,3,4,14,7,5,11,
    10,15,4,2,7,12,9,5,6,1,13,14,0,11,3,8,
    9,14,15,5,2,8,12,3,7,0,4,10,1,13,11,6,
    4,3,2,12,9,5,15,10,11,14,1,7,6,0,8,13,
    //S7
    4,11,2,14,15,0,8,13,3,12,9,7,5,10,6,1,
    13,0,11,7,4,9,1,10,14,3,5,12,2,15,8,6,
```

```
    1,4,11,13,12,3,7,14,10,15,6,8,0,5,9,2,
    6,11,13,8,1,4,10,7,9,5,0,15,14,2,3,12,
    //S8
    13,2,8,4,6,15,11,1,10,9,3,14,5,0,12,7,
    1,15,13,8,10,3,7,4,12,5,6,11,0,14,9,2,
    7,11,4,1,9,12,14,2,0,6,10,13,15,3,5,8,
    2,1,14,7,4,10,8,13,15,12,9,0,3,5,6,11
};
static bool SubKey[16][48];
void Des_Run(char Out[8], char In[8], bool Type) {
    static bool M[64], Tmp[32], *Li=&M[0], *Ri=&M[32];
    ByteToBit(M, In, 64);
    Transform(M, M, IP_Table, 64);
    if (Type==ENCRYPT) {
        for (int i=0; i<16; i++) {
            memcpy(Tmp, Ri, 32);
            F_func(Ri, SubKey[i]);
            Xor(Ri, Li, 32);
            memcpy(Li, Tmp, 32);
        }
    }
    else {
        for (int i=15; i>=0; i--) {
            memcpy(Tmp, Li, 32);
            F_func(Li, SubKey[i]);
            Xor(Li, Ri, 32);
            memcpy(Ri, Tmp, 32);
        }
    }
    Transform(M, M, IPR_Table, 64);
    BitToByte(Out, M, 64);
}
void Des_SetKey(const char Key[8]) {
    static bool K[64], *KL=&K[0], *KR=&K[28];
    ByteToBit(K, Key, 64);
    Transform(K, K, PC1_Table, 56);
    for (int i=0; i<16; i++) {
        RotateL(KL, 28, LOOP_Table[i]);
        RotateL(KR, 28, LOOP_Table[i]);
    }
}
void F_func(bool In[32], const bool Ki[48]) {
    static bool MR[48];
    Transform(MR, In, E_Table, 48);
```

```
    Xor(MR, Ki, 48);
    S_func(In, MR);
    Transform(In, In, P_Table, 32);
}
void S_func(bool Out[32], const bool In[48]) {
    for (char i = 0, j, k; i < 8; i++, In += 6, Out += 4) {
        j = (In[0] << 1) + In[5];
        k = (In[1] << 3) + (In[2] << 2) + (In[3] << 1) + In[4];
        ByteToBit(Out, &S_Box[i][j][k], 4);
    }
}
void Transform(bool * Out, bool * In, const char * Table, int len) {
    static bool Tmp[256];
    for (int i = 0; i < len; i++)
        Tmp[i] = In[Table[i] - 1];
    memcpy(Out, Tmp, len);
}
void Xor(bool * InA, const bool * InB, int len) {
    for (int i = 0; i < len; i++) {
        InA[i] ^= InB[i];
    }
}
void RotateL(bool * In, int len, int loop)
{
    static bool Tmp[256];
    memcpy(Tmp, In, loop);
    memcpy(In, In + loop, len - loop);
    memcpy(In + len - loop, Tmp, loop);
}
void ByteToBit(bool * Out, const char * In, int bits) {
    for (int i = 0; i <bits; i++)
        Out[i] = (In[i / 8] >> (i % 8)) & 1;
}
void  BitToByte(char * Out, const bool * In, int bits) {
    memset(Out, 0, (bits + 7) / 8);
    for (int i = 0; i <bits; i++)
        Out[i / 8] |= In[i] << (i % 8);
}

int main()
{
    char key[8] = { 1,9,8,0,9,1,7,2 }, str[] = "test";
    puts("Before encrypting");
    puts(str);
```

```
    Des_SetKey(key);
    Des_Run(str, str, ENCRYPT);
    puts("After encrypting");
    puts(str);
    puts("After decrypting");
    Des_Run(str, str, DECRYPT);
    puts(str);
    return 0;
}
```

运行结果如图 3-2 所示。

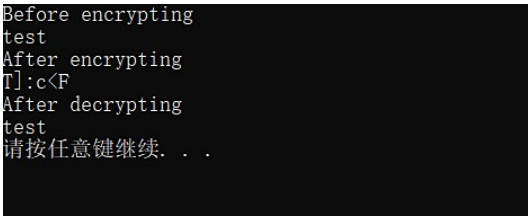

图 3-2　第 3 章实验 1 运行结果

2. 实验 2——RSA 算法的程序实现

```
#include<iostream>
#include<cmath>
#include<cstring>
#include<ctime>
#include<cstdlib>
using namespace std;

int Plaintext[100];                    //明文
long long Ciphertext[100];             //密文
int n, e =0, d;

//二进制转换
int BianaryTransform(int num, int bin_num[])
{
    int i =0,  mod =0;
    //转换为二进制,逆向暂存 temp[]数组中
    while(num !=0)
    {
        mod =num%2;
        bin_num[i] =mod;
        num =num/2;
        i++;
    }
```

```
    //返回二进制数的位数
    return i;
}
//反复平方求幂
long long Modular_Exonentiation(long long a, int b, int n)
{
    int c =0, bin_num[1000];
    long long d =1;
    int k =BianaryTransform(b, bin_num)-1;
    for(int i =k; i >=0; i--)
    {
        c =2 * c;
        d =(d * d)%n;
        if(bin_num[i] ==1)
        {
            c =c +1;
            d =(d * a)%n;
        }
    }
    return d;
}

//生成 1000 以内的素数
int ProducePrimeNumber(int prime[])
{
    int c =0, vis[1001];
    memset(vis, 0, sizeof(vis));
    for(int i =2; i <=1000; i++)if(!vis[i])
    {
        prime[c++] =i;
        for(int j =i * i; j <=1000; j+=i)
            vis[j] =1;
    }
    return c;
}

//欧几里得扩展算法
int Exgcd(int m,int n,int &x)
{
    int x1,y1,x0,y0, y;
    x0=1; y0=0;
    x1=0; y1=1;
    x=0; y=1;
    int r=m%n;
```

```
    int q=(m-r)/n;
    while(r)
    {
        x=x0-q*x1; y=y0-q*y1;
        x0=x1; y0=y1;
        x1=x; y1=y;
        m=n; n=r; r=m%n;
        q=(m-r)/n;
    }
    return n;
}

//RSA 初始化
void RSA_Initialize()
{
    //取出 1000 内素数保存在 prime[]数组中
    int prime[5000];
    int count_Prime =ProducePrimeNumber(prime);
    //随机取两个素数 p,q
    srand((unsigned)time(NULL));
    int ranNum1 =rand()%count_Prime;
    int ranNum2 =rand()%count_Prime;
    int p =prime[ranNum1], q =prime[ranNum2];
    n =p*q;
    int On =(p-1)*(q-1);

    //用欧几里得扩展算法求 e,d
    for(int j =3; j <On; j+=1331)
    {
        int gcd =Exgcd(j, On, d);
        if( gcd ==1 && d >0)
        {
            e =j;
            break;
        }
    }
}

//RSA 加密
void RSA_Encrypt()
{
    cout<<"Public Key (e, n) : e ="<<e<<" n ="<<n<<'\n';
    cout<<"Private Key (d, n) : d ="<<d<<" n ="<<n<<'\n'<<'\n';
    int i =0;
```

```
    for(i =0; i <100; i++)
        Ciphertext[i] =Modular_Exonentiation(Plaintext[i], e, n);
    cout<<"Use the public key (e, n) to encrypt:"<<'\n';
    for(i =0; i <100; i++)
        cout<<Ciphertext[i]<<" ";
    cout<<'\n'<<'\n';
}

//RSA 解密
void RSA_Decrypt()
{
    int i =0;
    for(i =0; i <100; i++)
        Ciphertext[i] =Modular_Exonentiation(Ciphertext[i], d, n);
    cout<<"Use private key (d, n) to decrypt:"<<'\n';
    for(i =0; i <100; i++)
        cout<<Ciphertext[i]<<" ";
    cout<<'\n'<<'\n';
}

//算法初始化
void Initialize()
{
    int i;
    srand((unsigned)time(NULL));
    for(i =0; i <100; i++)
        Plaintext[i] =rand()%1000;
    cout<<"Generate 100 random numbers:"<<'\n';
    for(i =0; i <100; i++)
        cout<<Plaintext[i]<<" ";
    cout<<'\n'<<'\n';
}

int main()
{
    Initialize();
    while(!e)
        RSA_Initialize();
    RSA_Encrypt();
    RSA_Decrypt();
    return 0;
}
```

运行结果如图 3-3 所示。

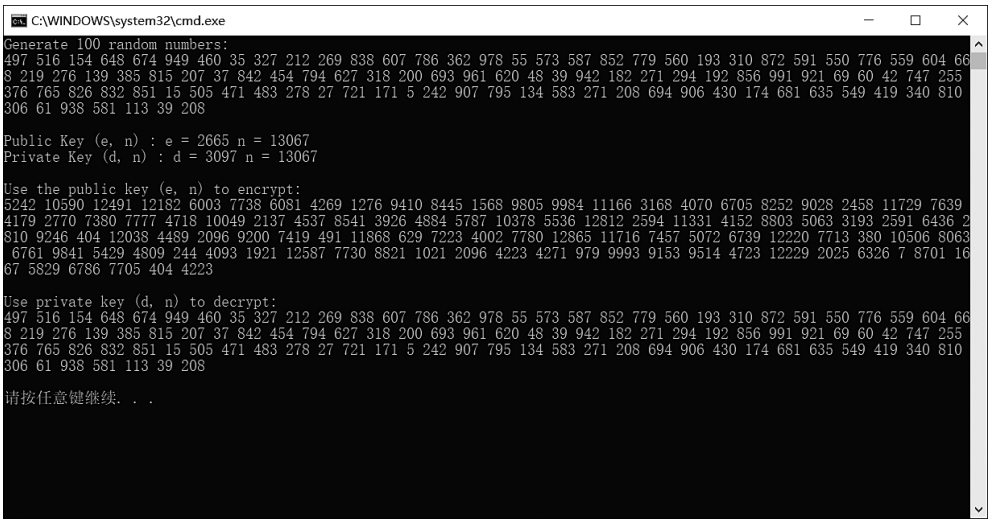

图 3-3　第 3 章实验 2 运行结果

3.5　实验结果分析与总结

1. 实验 1——DES 算法的程序实现

设置一个密钥为数组 char key[8] = { 1,9,8,0,9,1,7,2 },要加密的字符串数组为“test”,利用 Des_SetKey(key)设置加密的密钥,调用 Des_Run()对输入的明文进行加密,其中,第一个参数 str 是输出的密文,第二个参数 str 是输入的明文,枚举值 ENCRYPT 设置进行加密运算。

2. 实验 2——RSA 算法的程序实现

RSA 是一种非对称加密算法,在公开密钥和电子商业中被广泛使用。它基于一个很简单的数论事实,两个素数相乘很容易,对两素数乘积因式分解很困难。

3.6　思考题

假设需要加密的明文信息为 $m=85$,选择:$e=7$,$p=11$,$q=13$,说明使用 RSA 算法的加密和解密过程。

解析:

$n=pq=11\times13=143$,

$\varphi(n)=(p-1)(q-1)=(11-1)\times(13-1)=120$,

根据 $ed\equiv1 \bmod \varphi(n)$,

又有 $7d \bmod 120=1$,

得出 $d=103$，

公钥为$(e,n)=(7,143)$，

加密公式为 $c=m\hat{}e \bmod n$，

根据公钥加密明文 m 计算得出 $C=85\hat{}7 \bmod 143=123$，

私钥为$(d,n)=(103,143)$，

解密公式为 $m=c\hat{}d \bmod n$，

根据私钥解密 C 计算得出 $m=123\hat{}103 \bmod 143=85$。

根据上述过程理解 RSA 算法的流程。

3.7 练习

(1) 就目前计算机设备的计算能力而言，数据加密标准(DES)不能抵抗对密钥的穷举搜索攻击，其原因是(　　)。(答案：B)

A. DES 算法是公开的

B. DES 的密钥较短

C. DES 除了其中 S 盒是非线性变换外，其余变换均为线性变换

D. DES 算法简单

解析：DES 所使用的密钥是 64 位，实际用到了 56 位，第 8、16、24、32、40、48、56、64 位是校验位，使得每个密钥都有奇数个 1。

(2) 密码技术的分类有很多种，根据加密和解密所使用的密钥是否相同，可以将加密算法分为对称密码体制和(非对称密码体制)，其中，对称密码体制又可分为两类，按字符逐位加密的(序列密码)和按固定数据块大小加密的(分组密码)。

(3) 为了保障数据的存储和传输安全，需要对一些重要数据进行加密。由于对称密码算法(　①　)，所以特别适合对大量的数据进行加密。DES 实际的密钥长度是(　②　)位。(答案：①C　②A)

① A. 比非对称密码算法更安全

B. 比非对称密码算法密钥长度更长

C. 比非对称密码算法效率更高

D. 还能同时用于身份认证

② A. 56　　B. 64　　C. 128　　D. 256

(4) 以下不属于对称密码算法的是(　　)。(答案：D)

A. IDEA　　B. RC　　C. DES　　D. RSA

(5) 密码系统的安全性取决于用户对于密钥的保护，实际应用中的密钥种类有很多，从密钥管理的角度可以分为(初始密钥)、(会话密钥)、密钥加密密钥和(主密钥)。