



南开大学
Nankai University

南 开 大 学

网 络 空 间 安 全 学 院

2022 年球季密码学实验报告

密码学大作业

于文明

年级：2020 级

专业：信息安全

指导教师：古力

2023 年 1 月 6 日

摘要

关键字：Parallel

目录

一、 实验内容	1
(一) 实验环境	1
(二) 实验内容	1
(三) 实验要求	1
二、 保密通信协议设计	1
三、 程序流程图	2
(一) RSA 加解密	2
1. 大素数生成	2
2. Miller-Rabin 检测	3
3. RSA 加解密	3
(二) AES 加解密 (CBC 模式)	4
1. AES 加密	4
2. AES 解密	5
(三) 加密解密过程	5
(四) 用户端/服务端交互图	7
四、 RSA 算法介绍	7
(一) 大素数生成	7
1. 生成大随机数	7
2. 素数表筛查	7
3. Rabin-Miller 检测	8
(二) RSA 加解密	8
1. 公钥	8
2. 私钥	8
3. 加密算法	8
4. 解密算法	8
五、 AES 算法介绍	8
(一) CBC 模式	8
1. 概念	8
2. 初始向量 IV	8
(二) 数据填充 (PKCS7)	9
(三) 算法原理	9
(四) 字节代换 ByteSub 部件	9
(五) 行移位变换 ShiftRow	9
(六) 列混合运算 MixColumn	9

(七) 密钥加 AddRoundKey	10
(八) 密钥编排	10
六、 信息传输过程	10
(一) 用户端	10
(二) 服务端	10
七、 核心代码	10
(一) 项目整体介绍	10
(二) RSA 算法	11
(三) AES 算法	11
1. CBC 模式实现 (以加密部分为例)	11
2. 和偏移向量异或	13
3. 类型转换函数	13
(四) 初始化	14
1. Socket 初始化	14
2. RSA 公钥和密钥初始化	14
3. AES 密钥初始化	15
(五) 通信过程	16
1. 用户端主程序	16
2. 服务端主程序	16
八、 安全性分析	17
(一) RSA - AES 混合加密的安全性	17
(二) 中间人攻击	18
九、 实验结果	18
(一) 建立连接	18
(二) RSA 算法的实现	18
1. 用户端	18
2. 服务端	19
(三) AES 算法的实现	19
1. 用户端	19
2. 服务端	19
(四) 信息传输	19
1. 用户端	19
2. 服务端	20
十、 附录	20

一、 实验内容

(一) 实验环境

运行 Windows 操作系统的 PC 机, 具有 VC 等语言编译环境

(二) 实验内容

设计一个保密通信的协议, 具体要求为: 利用 RSA 公钥密码算法, 为双方分配一个 AES 算法的会话密钥, 然后利用 AES 加密算法和分配的会话密钥, 加解密传送的信息。

假设条件: 假设通讯双方为 A 和 B, 并假设发方拥有自己的 RSA 公钥 PKA 和私钥 SKA, 同时收方拥有自己的 RSA 公钥 PKB 和私钥 SKB, 同时收发双方已经通过某种方式知道了双方的公钥。

(三) 实验要求

- 1、作业需要先设计出保密通讯协议的过程和具体步骤;
- 2、分别编写 A, B 两个用户端的程序, 写清楚两个程序分别要完成的功能, 并能够在两个程序间进行通讯;
- 3、对 AES 加密的保密信息, 要求采用 CBC 模式进行加密;
- 4、大作业的提交方式同实验报告的提交, 也就是说既要提交程序实现的说明文档, 也要提交源代码和可执行程序。

二、 保密通信协议设计

- 服务端和用户端分别生成 RSA 公钥和密钥, 之后交换公钥
- 用户端生成 AES 的密钥 aes_key, 使用服务端的 RSA 公钥加密 aes_key 发送给服务端
- 服务端使用自己的 RSA 私钥解密后得到用户端的 AES 密钥 aes_key
- 用户端发送消息, 使用 AES 加密后 send 给服务端
- 服务端接收到加密消息后, 使用解密得到的 aes_key 解密

三、 程序流程图

(一) RSA 加解密

1. 大素数生成

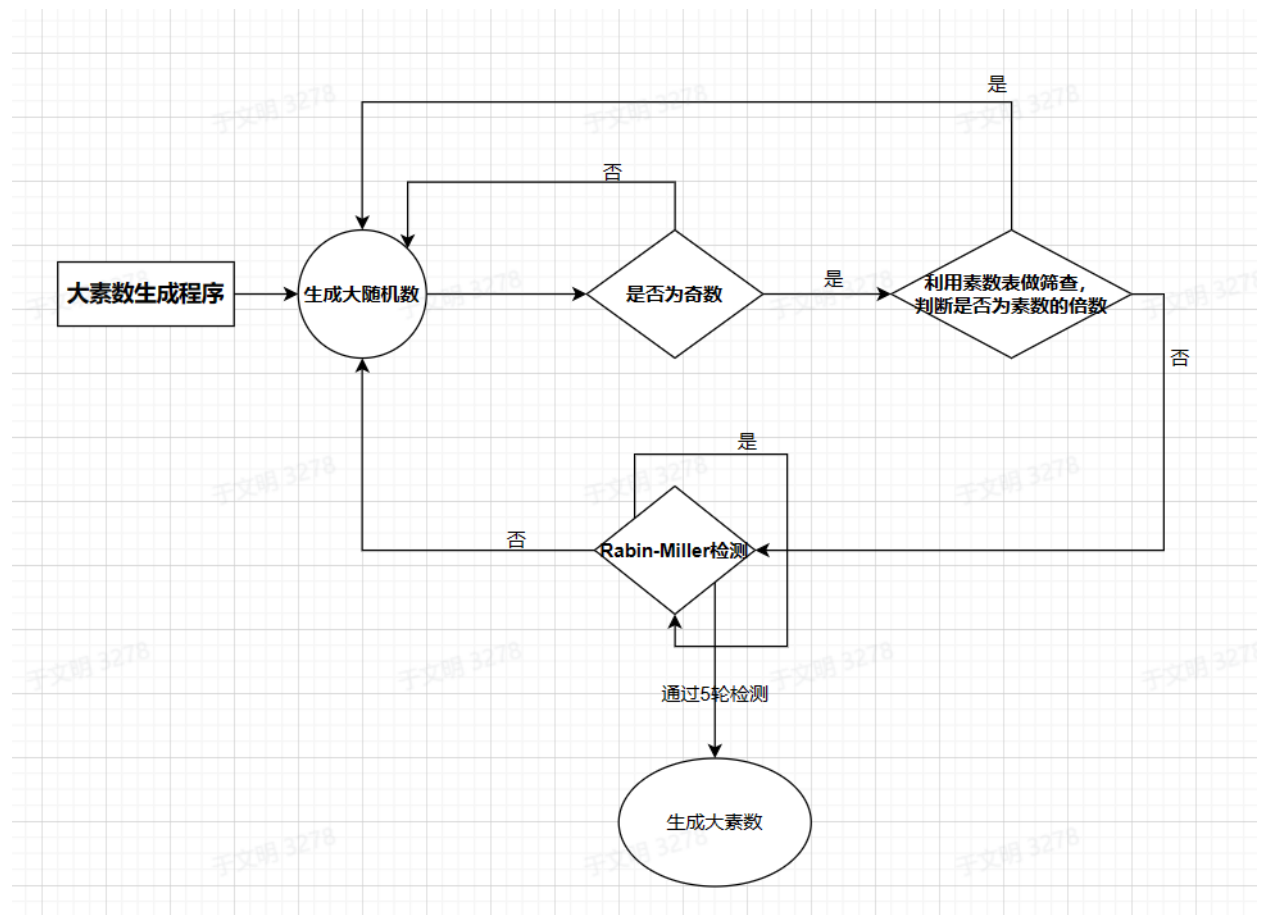


图 1: 大素数生成

2. Miller-Rabin 检测

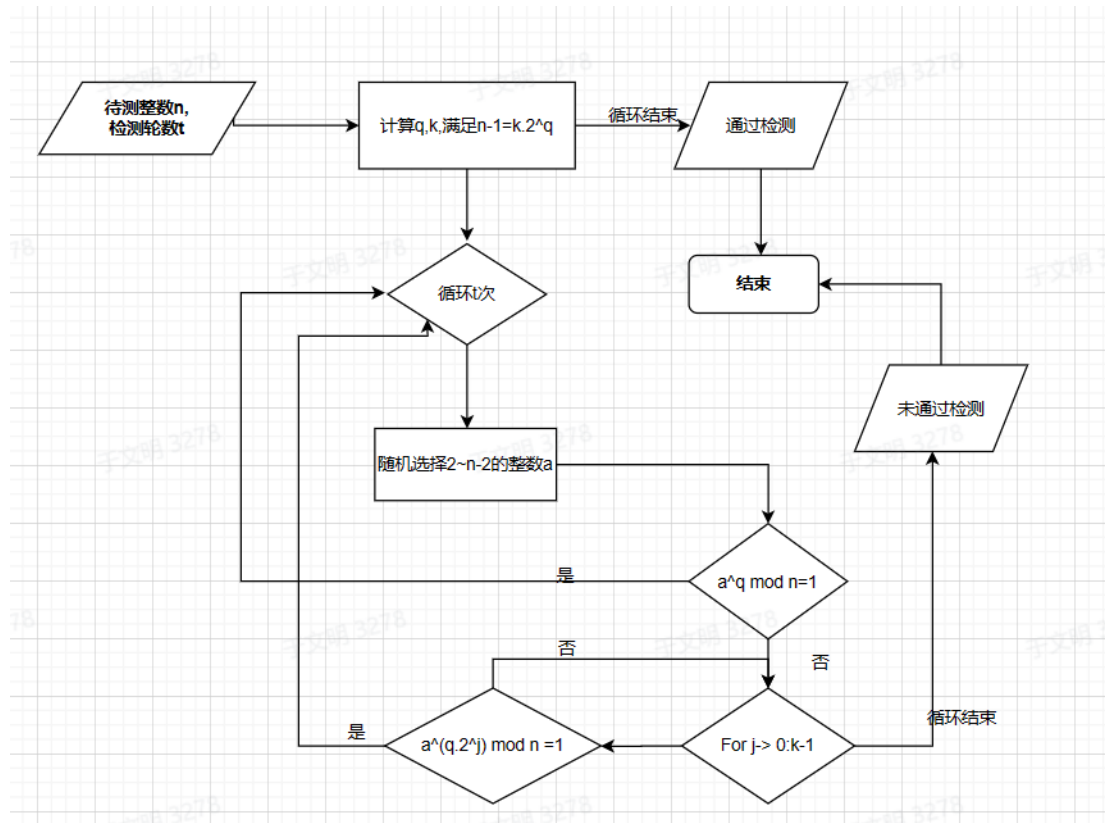


图 2: Rabin-Miller 检测

3. RSA 加解密

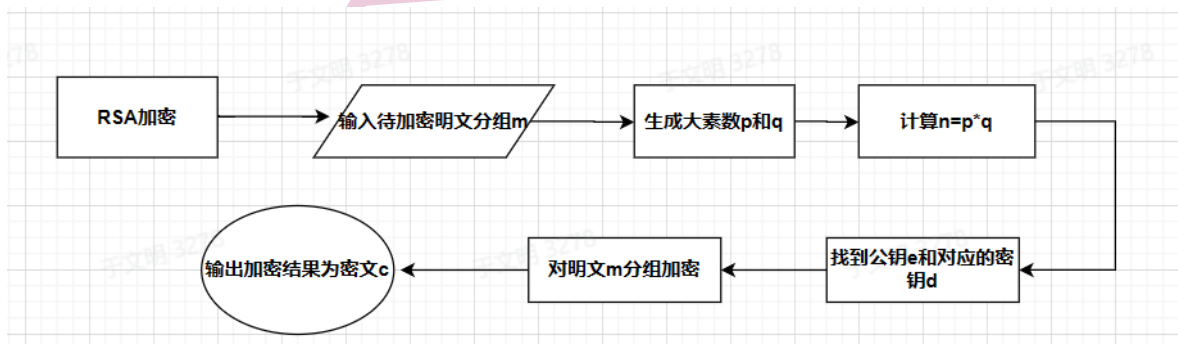


图 3: RSA 加密流程图

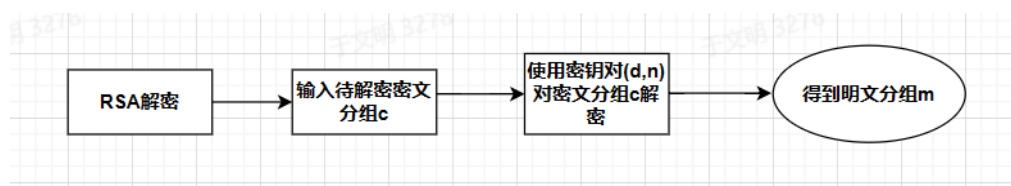


图 4: RSA 解密流程图

(二) AES 加解密 (CBC 模式)

1. AES 加密

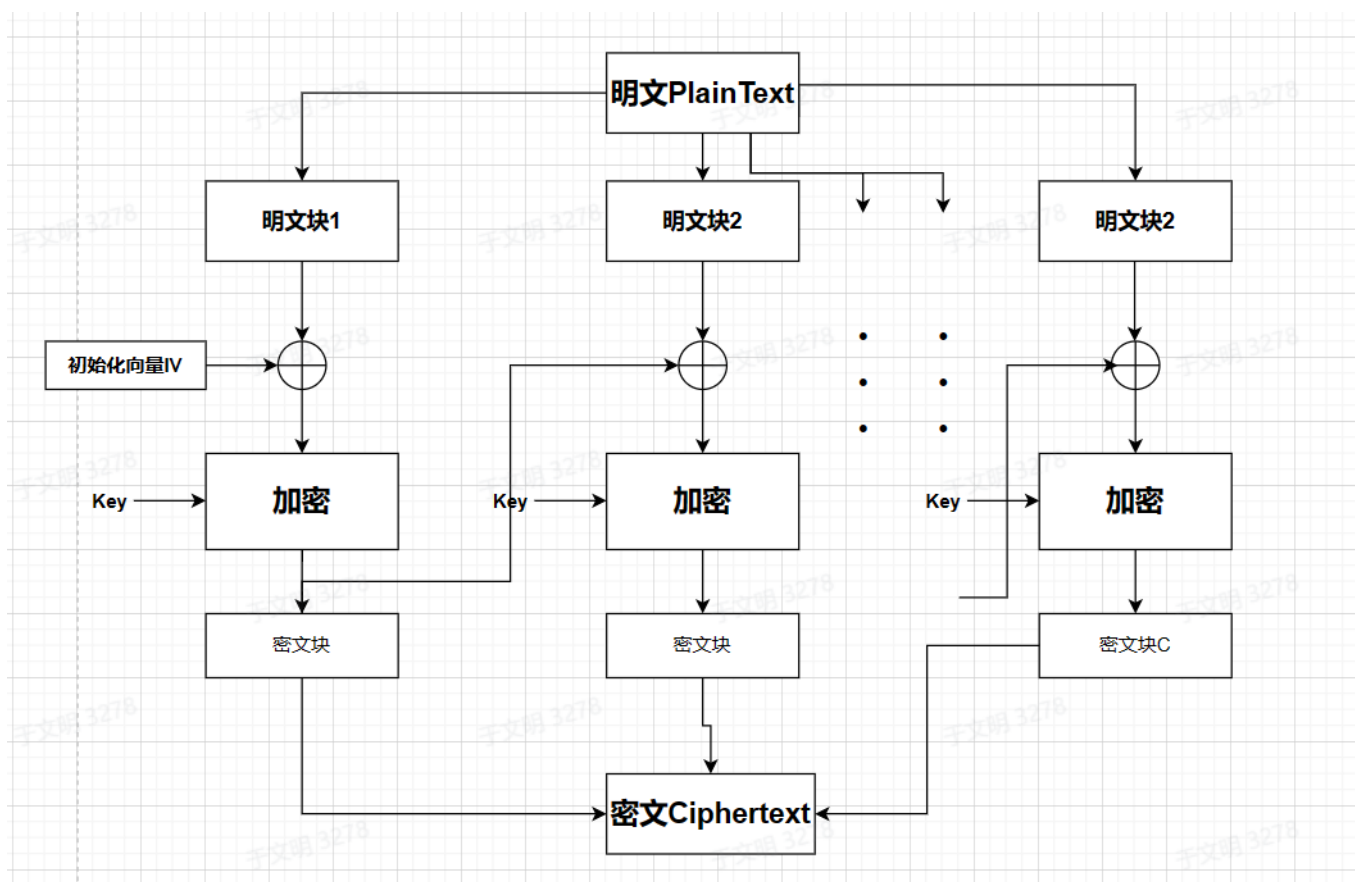


图 5: AES 加密流程图

2. AES 解密

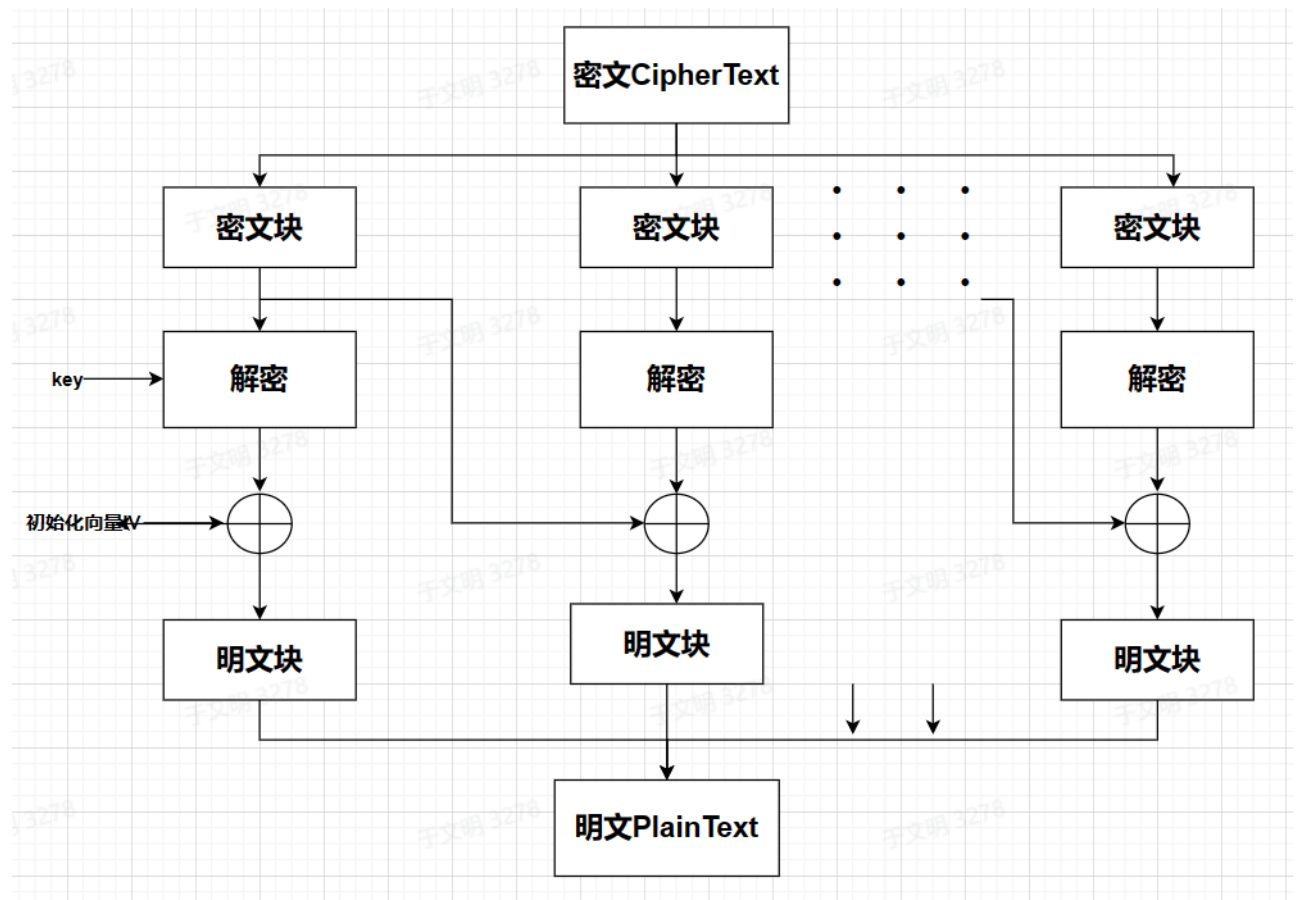


图 6: AES 解密流程图

(三) 加密解密过程

加密解密过程和之前 ECB 模式的类似，这里就直接用当时的流程图

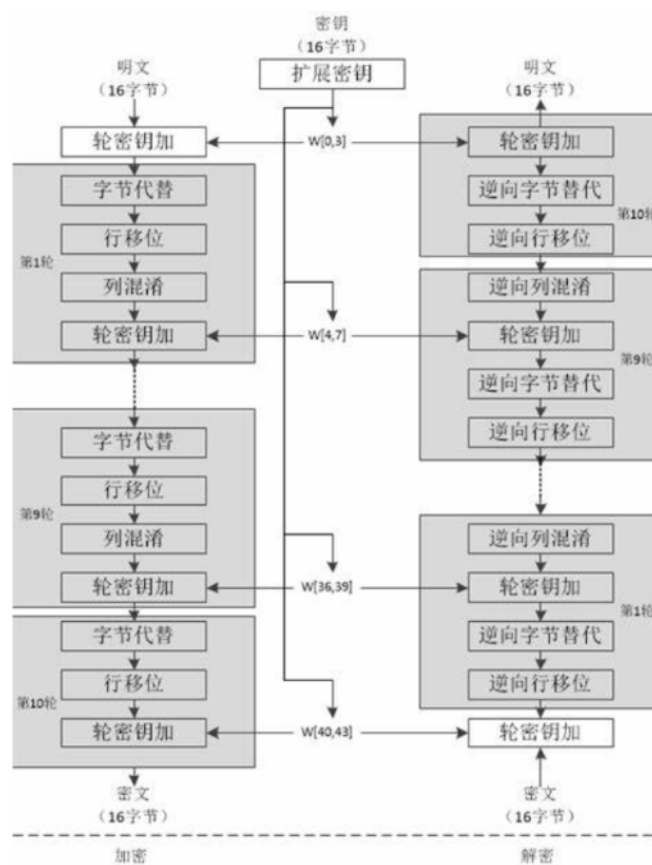


图 7: 加密解密具体流程

(四) 用户端/服务端交互图

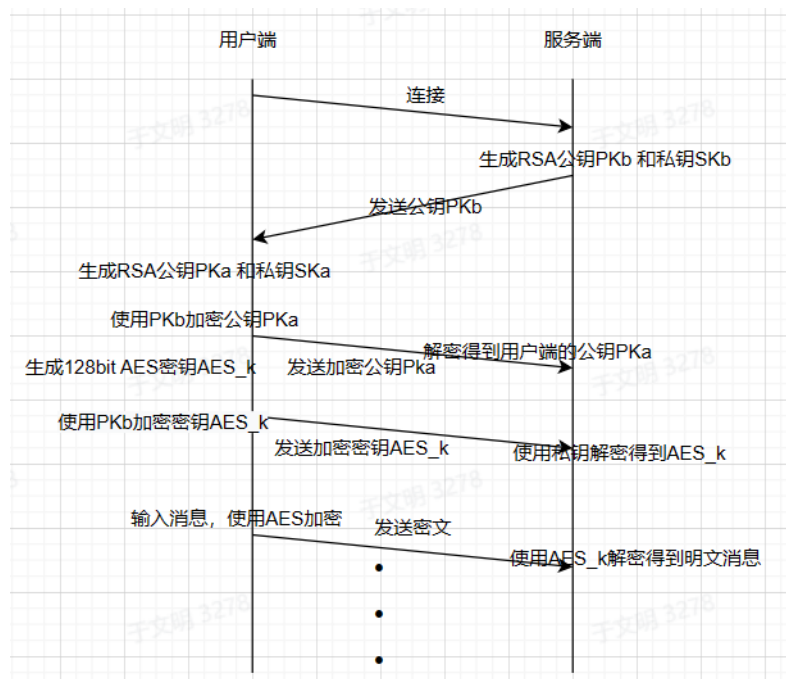


图 8: 交互图

四、 RSA 算法介绍

(一) 大素数生成

1. 生成大随机数

因为库函数 `rand()` 最大只能产生 `0X7FFF` 的数, 为了能产生 32 位的随机数, 需要 3 次 `rand()` 操作, 即 `rand()«17 + rand()«2 + rand()`, 同时根据需要生成的大整数的 `size`, 在外层添加循环 `For i->1:size: data[i]=rand()«17 + rand()«2 + rand();`

2. 素数表筛查

定义 `IsPrime()` 函数判断是否是素数, 然后从小到大生成 1000 个素数填充进素数表 `prime[1000]`。使用 `Random()` 生成一个大随机数, 进行奇偶性检测, 判断是否为奇数。使用 `prime[1000]` 进行筛查, 使之不能被其中的任一素数整除。伪代码

itob

```

1 IsPrime();
2 genPrime();
3 while(!flag):
4     n.Random();
5     while(!n.isOdd())
6         n.Random();
7     for i -> 0: sizeof(prime)/sizeof(int)
8         if n//prime[i]:
9             flag=false;
  
```

10

`break;`

3. Rabin-Miller 检测

Miller Rabin 算法的依据是费马小定理: $a^{p-1} \equiv 1 \pmod{P}$

假设需要判断的数是 p

我们把 $p-1$ 分解为 $2^k * t$

当 p 是素数, 有 $a^{(2^k * t)} \equiv 1 \pmod{p}$

然后随机选择一个数 a , 计算出 $a^t \pmod{p}$

让其不断的自乘, 同时结合二次探测定理进行判断

如果我们自乘后的数 $\pmod{p}=1$, 但是之前的数 $\pmod{p} \neq \pm 1$

那么这个数就是合数 (违背了二次探测定理)

这样乘 k 次, 最后得到的数就是 a^{p-1}

那么如果最后计算出的数不为 1, 这个数也是合数 (费马小定理)

(二) RSA 加解密

1. 公钥

选择两个不同的大素数 p 和 q , n 是二者的乘积, 即 $\varphi(n) = pq$

e e $n e$

2. 私钥

求出正数 d , 使其满足 $e * d \equiv 1 \pmod{\varphi(n)}$

$n d$

3. 加密算法

对于明文 m , 由 $c \equiv m^e \pmod{n}$, c

4. 解密算法

对于密文 c , 由 $m \equiv c^d \pmod{n}$, m

五、 AES 算法介绍

(一) CBC 模式

1. 概念

密文分组像链条一样相互连接在一起。在 CBC 模式中, 首先将明文分组与前一个密文分组进行 XOR 运算, 然后再进行加密

2. 初始向量 IV

当加密第一个明文分组时, 由于不存在“前一个密文分组”, 因此需要事先准备一个长度为一个分组的比特序列来代替“前一个密文分组”, 这个比特序列称为初始化向量 (Initialization Vector), 通常缩写为 IV, 一般来说, 每次加密时都会随机产生一个不同的比特序列来作为初始化向量。在程序中为了方便起见, 我们将其初始化为 0000...

(二) 数据填充 (PKCS7)

数据填充通常有两个作用一是按要求将数据补足到要就的块长度来满足加密算法的应用需求；二是通过增加填充数据来进一步提高密文的安全性。PKCS7 是当下各大加密算法都遵循的填充算法，且 OpenSSL 加密算法默认填充算法就是 PKCS7。PKCS7Padding 的填充方式为当数据长度不足数据块长度时，缺几位补几个几，eg. 对于 AES128 算法其数据块为 16Byte（数据长度需要为 16Byte 的倍数），如果数据为“00112233445566778899AA”一共 11 个 Byte，缺了 5 位，采用 PKCS7Padding 方式填充之后的数据为“00112233445566778899AA0505050505”。

特别注意的一点是如果是数据刚好满足数据块长度也要在元数据后在按 PKCS7 规则填充一个数据块数据，这样做的目的是为了区分有效数据和补齐数据。

(三) 算法原理

AES 算法本质上是一种对称分组密码体制，采用代替/置换网络，每轮由三层组成：线性混合层确保多轮之上的高度扩散，非线性层由 16 个 S 盒并置起到混淆的作用，密钥加密层将子密钥异或到中间状态。Rijndael 是一个迭代分组密码，其分组长度和密钥长度都是可变的，只是为了满足 AES 的要求才限定处理的分组大小为 128 位，而密钥长度为 128 位、192 位或 256 位，相应的迭代轮数 N，为 10 轮、12 轮、14 轮。AES 汇聚了安全性能、效率、可实现性、灵活性等优点。最大的优点是可以给出算法的最佳差分特征的概率，并分析算法抵抗差分密码分析及线性密码分析的能力。

(四) 字节代换 ByteSub 部件

字节代换是非线性变换，独立地对状态的每个字节进行。代换表（即 S-盒）是可逆的，由以下两个变换的合成得到：

- (1) 首先，将字节看作 GF(28) 上的元素，映射到自己的乘法逆元，‘00’映射到自己。
- (2) 其次，对字节 z 作 (GF(2) 上的，可逆的) 仿射变换：

(五) 行移位变换 ShiftRow

行移位是将状态阵列的各行进行循环移位，不同状态行的位移量不同。第 0 行不移动，第 1 行循环左移 C1 个字节，第 2 行循环左移 C2 个字节，第 3 行循环左移 C3 个字节。位移量 C1、C2、C3 的取值与 Nb 有关。ShiftRow 的逆变换是对状态阵列的后 3 列分别以位移量 Nb-C1、Nb-C2、Nb-C3 进行循环移位，使得第 i 行第 j 列的字节移位到 $(j+Nb-Ci) \bmod Nb$ 。

(六) 列混合运算 MixColumn

在列混合变换中，将状态阵列的每个列视为系数为 GF(28) 上的多项式，再与一个固定的多项式 $c(x)$ 进行模 x^4+1 乘法。当然要求 $c(x)$ 是模 x^4+1 可逆的多项式，否则列混合变换就是不可逆的，因而会使不同的输入分组对应的输出分组可能相同。Rijndael 的设计者给出的 $c(x)$ 为（系数用十六进制数表示）： $c(x) = '03'x^3 + '01'x^2 + '01'x + '02'$ $c(x)$ 是与 x^4+1 互素的，因此是模 x^4+1 可逆的。列混合运算也可写为矩阵乘法。设 $b(x) = c(x) a(x)$ ，这个运算需要做 GF(28) 上的乘法，但由于所乘的因子是 3 个固定的元素 02、03、01，所以这些乘法运算仍然是比较简单的。列混合运算的逆运算是类似的，即每列都用一个特定的多项式 $d(x)$ 相乘。 $d(x)$ 满足 $('03'x^3 + '01'x^2 + '01'x + '02') d(x) = '01'$ 由此可得

$$d(x) = '0B'x^3 + '0D'x^2 + '09'x + '0E'$$

(七) 密钥加 AddRoundKey

密钥加是将轮密钥简单地与状态进行逐比特异或。轮密钥由种子密钥通过密钥编排算法得到，轮密钥长度等于分组长度 N_b 。密钥加运算的逆运算是其自身。

(八) 密钥编排

密钥编排指从种子密钥得到轮密钥的过程，它由密钥扩展和轮密钥选取两部分组成。其基本原则如下：

- (1) 轮密钥的字数（4 比特 32 位的数）等于分组长度乘以轮数加 1；
- (2) 种子密钥被扩展成为扩展密钥；(3) 轮密钥从扩展密钥中取，其中第 1 轮轮密钥取扩展密钥的前 N_b 个字，第 2 轮轮密钥取接下来的 N_b 个字，如此下去。具体的内容请见教材和课程 ppt

六、 信息传输过程

(一) 用户端

用户端会经历以下过程:

- 初始化 Socket
- 生成 RSA 公钥 PK_A , 私钥 SK_A , 使用 Random() 生成 128bitAES 密钥 AES_k
- 和 Server 建立连接
- 接收服务端 RSA 公钥 PK_B , 传递自己的公钥 PK_A
- 使用 PK_B AES AES_K, AES

(二) 服务端

服务端会经历以下过程:

- 初始化 Socket, 等待建立连接
- 和用户端建立连接
- 生成 RSA 公钥 PK_B , 私钥 SK_B , 将公钥 PK_B 发送给用户端
- 接受用户端传来的加密后的密钥 AES_k, 在服务端使用 RSA 算法解密
- 接受用户端传来的加密后的消息 (密文), 使用 AES 算法解密后得到明文

七、 核心代码

(一) 项目整体介绍

这里以用户端为例

- include
 - aes.h aes 算法头文件

- bigInt.h 大整数类头文件
- conf.h 全局变量
- gen.h 生成大素数
- init.h 初始化函数
- prime.h 素数判断
- tools.h 辅助工具函数
- src
 - aes.cpp
 - bigInt.cpp
 - ClientMain.cpp 用户端主程序
 - conf.cpp
 - init.cpp

服务端目录结构同用户端类似

(二) RSA 算法

具体实现请见 RSA 实验部分

(三) AES 算法

这里只列举和 ECB 部分不同的代码

1. CBC 模式实现 (以加密部分为例)

itob

```
1 int AES::encrypt_cbc() {
2
3     unsigned char e, B[4][4], iv[4][4];
4     unsigned char keys[4][60];
5     int i, j;
6     int level, padLength;
7
8     int cArray[4][4];
9     int k, l;
10
11     plen = strlen(plain);
12
13     padLength = 16 - plen % 16; //需要填充多少个字符
14     for (i = plen; i < padLength + plen; i++) {
15         plain[i] = (char)padLength;
16     }
17     plen += padLength;
18     plain[plen] = '\0';
```

```
19
20
21     putchar('\n');
22
23     setKey(S_BOX, keys);
24
25     k = 0;
26
27     for (l = 0; l < plen; l += 16) {
28
29         k = l;
30         for (i = 0; i <= 3; i++) {
31             for (j = 0; j <= 3; j++) {
32                 B[j][i] = getIntFromChar(plain[k]);
33                 k++;
34             }
35         }
36         if (l == 0) {
37             for (i = 0; i < 4; i++) {
38                 for (j = 0; j < 4; j++) {
39                     iv[i][j] = 0;
40                 }
41             }
42
43         }
44
45         xorWithiv(iv, B); //和偏移向量进行异或
46
47         //轮密钥加
48         for (i = 0; i <= 3; i++)
49             for (j = 0; j <= 3; j++) {
50                 B[i][j] ^= keys[i][j];
51             }
52
53         for (level = 1; level <= beforeTimes; level++) { //1到9轮循环
54             bytesub(S_BOX, B); //字节代换
55             shiftrow(B); //行移位
56             mixcolumn(B); //列混合
57
58             //这里似乎又是轮密钥加
59             for (i = 0; i <= 3; i++)
60                 for (j = 0; j <= 3; j++)
61                     B[i][j] ^= keys[i][level * 4 + j];
62         }
63
64         bytesub(S_BOX, B); //第10轮循环
65         shiftrow(B);
66
```

```

67     for (i = 0; i <= 3; i++)
68         for (j = 0; j <= 3; j++) {
69             B[i][j] ^= keys[i][wordLength - 4 + j];
70             cArray[i][j] = (int)B[i][j];
71         }
72
73     convertArrayToStr(cArray, cipher + 1);
74
75     for (i = 0; i < 4; i++) {
76         for (j = 0; j < 4; j++) {
77             iv[i][j] = B[i][j];
78         }
79     }
80     for (i=0;i<4;i++)
81         for (j = 0; j < 4; j++)
82         {
83             printf("%x ", B[j][i]);
84         }
85     }
86     cipher[plen] = '\0';
87     clen = plen;
88     return 0;
89 }

```

2. 和偏移向量异或

itob

```

1 //和偏移向量进行异或
2 void AES::xorWithiv(unsigned char iv[4][4], unsigned char B[4][4]) {
3     for (int i = 0; i <= 3; i++) {
4         for (int j = 0; j <= 3; j++) {
5             B[i][j] ^= iv[i][j];
6         }
7     }
8 }

```

3. 类型转换函数

itob

```

1 //把4X4数组转回字符串
2 static void convertArrayToStr(int array[4][4], char* str) {
3     int i, j;
4     for (i = 0; i < 4; i++)
5         for (j = 0; j < 4; j++)
6             *str++ = (char)array[j][i];
7 }

```



```

8 //把4X4数组放进十六进制存放的字符串
9 static void convertArrayToStr16(unsigned char chArray[], unsigned char B
    [4][4], int l) {
10     int i, j;
11     for (i = 0; i < 4; i++) {
12         for (j = 0; j < 4; j++) {
13             chArray[l] = B[j][i];
14             l++;
15         }
16     }
17 }

```

(四) 初始化

1. Socket 初始化

itob

```

1 int SockInit()
2 {
3     sockServer = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
4     addrSer.sin_family = AF_INET;
5     addrSer.sin_port = ServerPort;
6     addrSer.sin_addr.S_un.S_addr = ServerIP;
7
8     if (bind(sockServer, (SOCKADDR*)&addrSer, sizeof(SOCKADDR)) == -1)
9         return 0;
10    else
11        return 1;
12 }

```

2. RSA 公钥和密钥初始化

itob

```

1 void RSAInit()
2 {
3     printf("生成RSA公钥密钥\n");
4     genPrime();
5     //产生大素数
6     printf("开始生成大素数p\n");
7     BigInt p = GeneratePrime();
8     p.display();
9
10    BigInt q = GeneratePrime();
11    q.display();
12
13    /*printf("公钥n=p*q\n");*/

```

```

14
15     n2 = p * q;
16
17     BigInt t = (p - 1) * (q - 1);
18     //y用于参与扩展欧几里得运算，存储t模e的乘法逆元
19     BigInt y;
20
21     BigInt temp;
22     while (1)
23     {
24         //产生与t互质的e
25         e2.Random(false);
26         while (!(gcd(e2, t) == 1))
27         {
28             e2.Random(false);
29         }
30
31         //用扩展欧几里德算法试图求出e模t的乘法逆元
32         temp = Extended_gcd(e2, t, d, y);
33
34         //e*d模t结果为1，说明d确实是e模t的乘法逆元
35         temp = (e2 * d) % t;
36         if (temp == 1)
37             break;
38         //否则重新生成e
39     }
40 }

```

3. AES 密钥初始化

itob

```

1 void PKInit()
2 {
3     //发送PKa
4     printf("调用PKInit\n");
5     n1.display();
6     send(sockClient, (char*)&n1, sizeof(n1), 0);
7     e1.display();
8     send(sockClient, (char*)&e1, sizeof(e1), 0);
9     //接收PKb
10    recv(sockClient, (char*)&n2, sizeof(n2), 0);
11    recv(sockClient, (char*)&e2, sizeof(e2), 0);
12
13    //生成AES的密钥
14    AES_k.Random(true);
15    AES_k.display();
16    //加密发送AES_k

```

```

17     BigInt tmp;
18     tmp = Power(AES_k,e2,n2);
19     send(sockClient, (char*)&tmp, sizeof(tmp), 0);
20
21     printf("PKInit Success\n");
22 }

```

(五) 通信过程

1. 用户端主程序

itob

```

1  int main()
2  {
3      AES aes;
4      Init();
5      //connect
6      if (connect(sockClient, (SOCKADDR*)&addrSer, sizeof(SOCKADDR)) == -1)
7          perror("connect");
8      else
9          printf("Connect to Server\n");
10     PKInit();
11
12     while (1)
13     {
14         memset(aes.plain, 0, sizeof(aes.plain));
15         printf("请输入消息:");
16         cin.getline(aes.plain, 1024, '\n');
17         aes.encrypt_cbc();
18         memset(sendBuffer, 0, sizeof(sendBuffer));
19         memcpy(sendBuffer, aes.cipher, sizeof(aes.cipher));
20         if (send(sockClient, sendBuffer, sizeof(sendBuffer), 0) ==
21             -1)
22             perror("send");
23         else
24             printf("发送成功\n");
25     }
26     WSACleanup();
27     system("pause");
28 }

```

2. 服务端主程序

itob

```

1  int main()
2  {

```

```

3      AES aes;
4      int len = sizeof(SOCKADDR);
5      Init();
6      //开始监听
7      if (listen(sockServer, 5) == -1) {
8          perror("listen");
9      }
10     while (1)
11     {
12         SOCKET clientSock= accept(sockServer, (SOCKADDR*)&addrCli, &
13             len);
14         PKInit(clientSock);
15         while (1)
16         {
17             memset(recvBuffer, 0, sizeof(recvBuffer));
18             if (recv(clientSock, recvBuffer, sizeof(recvBuffer),
19                 0) == -1)
20                 perror("recv");
21             else
22             {
23                 //开始解密
24                 memset(aes.cipher, 0, sizeof(aes.cipher));
25                 memcpy(aes.cipher, recvBuffer, sizeof(
26                     recvBuffer));
27                 aes.clen = strlen(aes.cipher);
28                 aes.decrypt_cbc();
29             }
30         }
31     }
32 }

```

八、 安全性分析

(一) RSA - AES 混合加密的安全性

RSA 加密机制：属于非对称加密，公钥用于对数据进行加密，私钥对数据进行解密，两者不可逆。公钥和私钥是同时生成的，且一一对应。比如：A 拥有公钥，B 拥有公钥和私钥。A 将数据通过公钥进行加密后，发送密文给 B，B 可以通过私钥和公钥进行解密。

AES 加密机制：属于对称加密，就是说，A 用密码对数据进行 AES 加密后，B 用同样的密码对密文进行 AES 解密。利用 RSA 来加密传输 AES 的密钥，用 AES 的密钥来加密数据。这样做：既利用了 RSA 的灵活性，可以随时改动 AES 的密钥；又利用了 AES 的高效性，可以高效传输数据。

单纯的使用 RSA（非对称加密）方式的话，效率会很低，因为非对称加密解密方式虽然很保险，但是过程复杂，需要时间长；但是，RSA 优势在于数据传输安全，且对于几个字节的数据，加密和解密时间基本可以忽略，所以用它加密 AES 密钥（一般 16 个字节）再合适不过了；单纯的使用 AES（对称加密）方式的话，死板且不安全。这种方式使用的密钥是一个固定的密

钥，客户端和服务端是一样的，一旦密钥被人获取，那么，我们所发的每一条数据都会被对方破解；但是，AES 有个很大的优点，那就是加密解密效率很高，而我们传输正文数据时，正号需要这种加解密效率高的，所以这种方式适合于传输量大的数据内容；

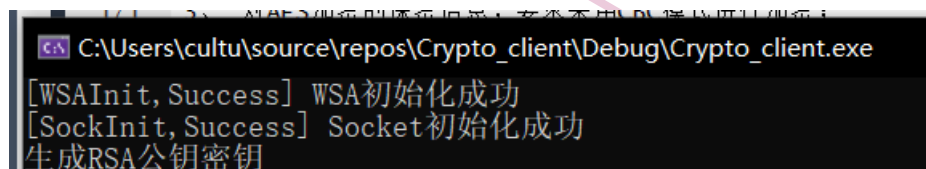
(二) 中间人攻击

设想：B 生成一对 RSA 密钥 P_b 、 S_b ，将公钥 P_b 发送给 A。而 AB 中有 C。C 截获了 P_b ，而自己生成了一对 RSA 密钥 P_c 、 S_c ，将 P_c 发送给 A。A 用 P_c 加密了会话密钥 K，发送给 B，被 C 截获。C 用 S_c 解密得到 K，再用 P_b 加密后给 B。这时 C 完成了中间人攻击。

所以说：RSA 的公钥在端与端间传递时，存在中间人攻击问题。最好的方法是服务端持有私钥，客户端直接内置好公钥，就不用担心中间人攻击了。当然 TLS 协议采取了身份认证数字签名的方式，这也是一种解决方案。

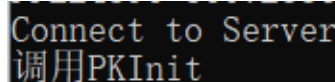
九、 实验结果

(一) 建立连接



```
C:\Users\cultu\source\repos\Crypto_client\Debug\Crypto_client.exe
[WSAInit, Success] WSA初始化成功
[SocketInit, Success] Socket初始化成功
生成RSA公钥密钥
```

图 9: 初始化

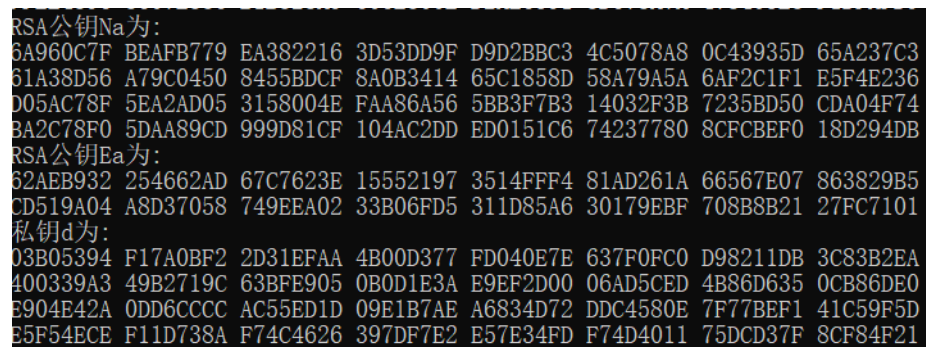


```
Connect to Server
调用PKInit
```

图 10: 建立连接

(二) RSA 算法的实现

1. 用户端



```
RSA公钥Na为:
6A960C7F BEAFB779 EA382216 3D53DD9F D9D2BBC3 4C5078A8 0C43935D 65A237C3
61A38D56 A79C0450 8455BDCF 8A0B3414 65C1858D 58A79A5A 6AF2C1F1 E5F4E236
D05AC78F 5EA2AD05 3158004E FAA86A56 5BB3F7B3 14032F3B 7235BD50 CDA04F74
BA2C78F0 5DAA89CD 999D81CF 104AC2DD ED0151C6 74237780 8CFCBEF0 18D294DB
RSA公钥Ea为:
62AEB932 254662AD 67C7623E 15552197 3514FFF4 81AD261A 66567E07 863829B5
CD519A04 A8D37058 749EEA02 33B06FD5 311D85A6 30179EBF 708B8B21 27FC7101
私钥d为:
03B05394 F17A0BF2 2D31EFAA 4B00D377 FD040E7E 637F0FC0 D98211DB 3C83B2EA
400339A3 49B2719C 63BFE905 0B0D1E3A E9EF2D00 06AD5CED 4B86D635 0CB86DE0
E904E42A 0DD6CCCC AC55ED1D 09E1B7AE A6834D72 DDC4580E 7F77BEF1 41C59F5D
E5F54ECE F11D738A F74C4626 397DF7E2 E57E34FD F74D4011 75DCD37F 8CF84F21
```

图 11: RSA 公钥 PK_a 和私钥 SK_a

2. 服务端

```

RSA公钥Nb为:
6A960C7F BEAFB779 EA382216 3D53DD9F D9D2BBC3 4C5078A8 0C43935D 65A237C3
61A38D56 A79C0450 8455BDCF 8A0B3414 65C1858D 58A79A5A 6AF2C1F1 E5F4E236
D05AC78F 5EA2AD05 3158004E FAA86A56 5BB3F7B3 14032F3B 7235BD50 CDA04F74
BA2C78F0 5DAA89CD 999D81CF 104AC2DD ED0151C6 74237780 8CFCBEF0 18D294DB
RSA公钥Eb为:
62AEB932 254662AD 67C7623E 15552197 3514FFF4 81AD261A 66567E07 863829B5
CD519A04 A8D37058 749EEA02 33B06FD5 311D85A6 30179EBF 708B8B21 27FC7101
私钥d为:
03B05394 F17A0BF2 2D31EFAA 4B00D377 FD040E7E 637F0FC0 D98211DB 3C83B2EA
400339A3 49B2719C 63BFE905 0B0D1E3A E9EF2D00 06AD5CED 4B86D635 0CB86DE0

```

图 12: RSA 公钥 PKb 和私钥 SKb

(三) AES 算法的实现

1. 用户端

```

AES的密钥AES_k为:
7EBD6FEE F2663098
FD6F3BFA F579F277
使用RSA加密发送AES_k
加密后的AES_k为:
15E4454C DAE4FAC4 7EA47DF9 F539851F 251367DE A2083B39 C9A72910 DA7C4E64
CF499AE9 53093AD3 4B2A867A 123342C3 4579B817 44C5D2C2 6FD9846B 394BE979

```

图 13: 生成 AES 密钥并使用 RSA 加密

2. 服务端

```

接受到加密后的AES密钥，开始解密...
密文为:
15E4454C DAE4FAC4 7EA47DF9 F539851F 251367DE A2083B39 C9A72910 DA7C4E64
CF499AE9 53093AD3 4B2A867A 123342C3 4579B817 44C5D2C2 6FD9846B 394BE979
F1AC581E DBE77B87 FC6044A6 274C444F 061F7707 E09E26C1 C14324F2 A3472878
CBD749FB 68CE2DA9 FB26B455 96EB1D48 305F6E36 3EFD35D9 C6C86842 9B62AD55
解密后的明文(AES_k)为:
7EBD6FEE F2663098
FD6F3BFA F579F277

```

图 14: 接收到加密后的 AES 密钥并使用 RSA 密钥解密

(四) 信息传输

1. 用户端

```

请输入消息:I like nankai
加密后的密文为(16进制):
1f 4f fc e3 43 de c0 7e 3b 10 93 17 5a b2 de 25 发送成功
请输入消息:terrible cryptography xxxxxxxxxxxx yyyyyyyyyy
加密后的密文为(16进制):
d4 cd 73 85 72 2d 57 b3 94 b7 9b 34 b6 d7 83 cd d1 e0 85 b1 a1 ee fa a1 c 3b f0 b 1e 2 34 91 22 1e 42 37 d1 c2 20 6a 14
af 8b d8 3b c8 ff cb 发送成功
请输入消息:

```

图 15: 接受明文输入并使用 AES 加密

2. 服务端

```
接受到的密文为(16进制):  
1f 4f fc e3 43 de c0 7e 3b 10 93 17 5a b2 de 25  
[RECV] 解密后的消息为I like nankai  
接受到的密文为(16进制):  
d4 cd 73 85 72 2d 57 b3 94 b7 9b 34 b6 d7 83 cd d1 e0 85 b1 a1 ee fa a1 c 3b f0 b 1e 2 34 91 22 1e 42 37 d1 c2 20 6a 14  
af 8b d8 3b c8 ff cb  
[RECV] 解密后的消息为terrible cryptography xxxxxxxxxxxx yyyyyyyyyy
```

图 16: 接收密文并使用 AES 解密

参考文献 [1]

十、 附录

本次实验相关资源已经上传至 [github:https://github.com/Metetor/NKU_Cryptography](https://github.com/Metetor/NKU_Cryptography)

NKU

参考文献

- [1] 吴世忠、宋晓龙、郭涛等译 Paul Garrett 著. *An Introduction to Cryptology*. 机械工业出版社, 2003.

NIKU