

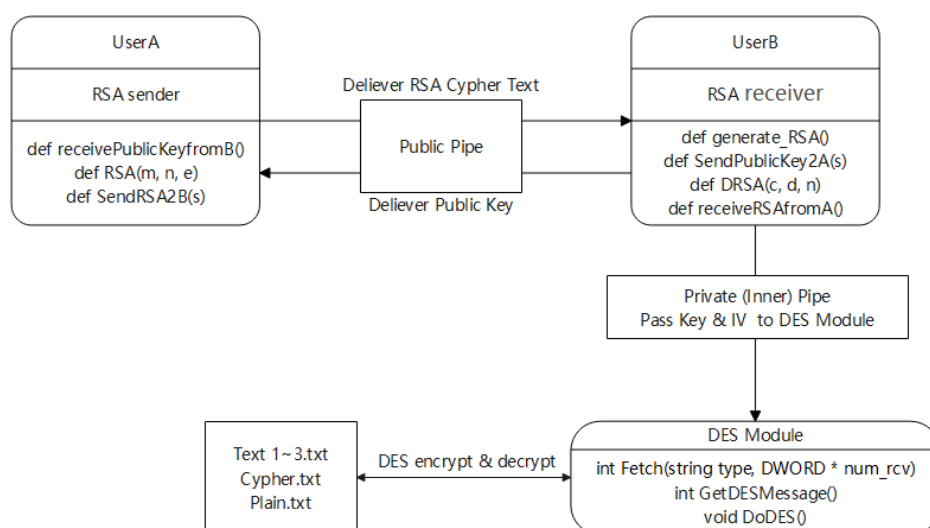
说明文档

框架介绍

PJ要求:

编程实现128bit的RSA(包括素数和密钥对产生),实现传送DES加密算法所要用的密钥, 并与DES算法一起形成传送DES加密密钥, 然后用DES加密算法加密的完整体系。

根据上述要求抽象出的PJ计划模拟两个用户A和B, 在AB之间通过公开信道传输RSA加密的DES密钥和IV值, B在收到A发出的DES加密所需的信息后进行解密再用自己的DES模块进行DES加密。由于在一台机器上很难模拟两个用户之间使用RSA进行加密的情况, 本次PJ计划用不同进程之间通过pipe进行数据传输的方式来模拟网络上的信道, 具体方式将在后续文档中详细说明, 逻辑图如下:



如图所示, 本次PJ代码部分有三个文件UserA.py, UserB.py, DES.cpp三个文件之间的数据传输通过pipe管道的方式来进行。其中User A, User B是两个python文件, 这两个进程用来模拟执行RSA加密传输的发送方和接收方, 也就是用户A和用户B, 两者间的pipe管道模拟的是网络上的公开信道。DES Module是DES.cpp文件, 由上次PJ中的代码进行了部分修改而得到, 所以DES Module这个进程在本次PJ中不当作一个用户而是用于模拟用户B的私有DES加密模块, DES Module和用户B之间的通信管道pipe用于模拟用户B本地的信息传输管道, 视为是私有信道。

运行逻辑

工作状态下, 要进行DES加密, DES加密模块向本地用户B请求所需要的Key和IV, 这一组数值由用户A给出, 以RSA加密格式通过公开信到发送给用户B。所以用户B需要生成自己的公钥和私钥, 并将公钥通过公开信到发送给用户A, 用户A接收到用户B发来的信息后将明文Key和IV进行加密, 将两个密文分别发送给用户B。用户B收到RSA密文后用自己的私钥解密得到明文Key和IV, 再通过本地私有信道将其传输给DES加密模块, DES加密模块得到信息后执行DES加密操作。在DES.cpp的控制台输出相应的信息。

代码解释

主要展示框图中的重要函数，辅助函数不做过多介绍，详见源码

UserA.py

这部分代码主要执行三件事

1. 接收用户B发来的公钥
2. 用B的公钥对DES Key 和 IV 明文进行加密
3. 将RSA加密数据传送给用户B

对应代码和注释如下

```
def receivePublicKeyfromB():
    # 连接用户B创建的发送管道SendRSA2Apipe
    fileHandle = win32file.CreateFile("\\\\.\\pipe\\SendRSA2Apipe",
                                       win32file.GENERIC_READ |
                                       win32file.GENERIC_WRITE,
                                       0, None,
                                       win32file.OPEN_EXISTING,
                                       0, None)

    # 从管道读取B发送的数据到res字符串中
    res = win32file.ReadFile(fileHandle, 4096)
    win32file.CloseHandle(fileHandle)
    # 对原始数据进行处理得到有效公钥数据
    res = str(res[1]).lstrip('b').lstrip('').rstrip('')
    # 输出确认信息
    print(res, "received from B")
    return res
```

```
def RSA(m, n, e):
    # RSA加密:  $c = m^e \bmod n$ 。n和e为上一步接收到的用户B的公钥值
    # fast_power 这个函数是模乘方运算
    c = fast_power(m, e, n)
    # 输出确认信息
    print("Cypher text =", c, "ready to be sent")
    # 将RSA加密密文传送给用户B
    SendRSA2B(str(c))
```

```
def SendRSA2B(s):
    # 发送RSA数据给用户B
    # 创建发送管道SendRSA2Bpipe, 挂起, 等待用户B连接
    p = win32pipe.CreateNamedPipe(r'\\.\\pipe\\SendRSA2Bpipe',
                                   win32pipe.PIPE_ACCESS_DUPLEX,
                                   win32pipe.PIPE_TYPE_MESSAGE | win32pipe.PIPE_WAIT,
                                   1, 65536, 65536, 300, None)
    # 检测到用户B连接管道, 发送数据
    win32pipe.ConnectNamedPipe(p, None)
    data = bytes(s.encode('utf-8'))
    win32file.WriteFile(p, data)
    # 输出确认信息
    print(data, "sent to B")
```

UserA.py的执行逻辑如下:

```

# 首先这里有一对DES需要的Key和IV值，这里沿用上次的数值，也可以通过修改此处的源码自己改变这个数值
mKey = int(0x38290fca2b38dc1f)
mIV = int(0x38273cad27b29367)

# 输出等待接收公钥的信息后挂起两秒（这是为了等待B建立传输管道，以防出现错误）
print("Receive nB from B")
time.sleep(2)
# 接收B传来的n值
n = int(receivePublicKeyfromB())
# 下面这部分接收e值同理
print("Receive eB from B")
time.sleep(2)
e = int(receivePublicKeyfromB())

# 输出确认信息，进行RSA加密及传输
print("RSA encrypt Key")
RSA(mKey, n, e)
print("RSA encrypt IV")
RSA(mIV, n, e)

```

UserB.py

这部分代码主要执行五件事

1. 生成用户B的RSA公钥密钥对
2. 把公钥传输给用户A
3. 接收用户A传来的RSA加密数据
4. 解密Key和IV
5. 将Key和IV传给本地DES加密模块进行加密（这个函数在逻辑图中忘写了）

对应代码和注释如下：

```

## 生成RSA公钥密钥对：这是一系列函数组成的算法，最终由generate_RSA()函数得到最后结果并调用传输函数传输## 公钥给用户A
def fast_power(base, power, n):
    # 模n下计算Base和power次方，输出结果base^power mod n
    result, tmp = 1, base
    while power > 0:
        if power&1 == 1:
            result = (result * tmp) % n
            tmp, power = (tmp * tmp) % n, power >> 1
    return result

def MillerRabin(n, iter_num):
    # Miller_Rabin 算法进行素性检测，返回BOOL值
    # 2 is prime
    if n == 2:
        return True
    # if n is even or less than 2, then n is not a prime
    if n&1 == 0 or n<2:
        return False
    # n-1 = (2^s)m
    m, s = n - 1, 0
    while m & 1 == 0:
        m = m >> 1
        s += 1

```

```

# M-R test
for _ in range(iter_num):
    b = fast_power(random.randint(2, n - 1), m, n)
    if b == 1 or b == n-1:
        continue
    for __ in range(s - 1):
        b = fast_power(b, 2, n)
        if b == n-1:
            break
    else:
        return False
return True

def rand64int():
    # 随机产生64bit整数
    num = 1
    for i in range(0, 62):
        num = num * 2 + random.randint(0, 1)
    num = num * 2 + 1
    return num

def generate_large_Prime():
    # 随机生成64bit整数, 用Miller_Rabin算法检测, 是素数则结束
    while True:
        x = rand64int()
        if MillerRabin(x, 10):
            return x

def extendgcd(a, b):
    # 扩展欧几里得算法, 解线性同余方程
    if b == 0:
        return 1, 0, a
    else:
        x, y, gcd = extendgcd(b, a % b)
        x, y = y, (x - (a // b) * y)
        return x, y, gcd

def generate_RSA():
    # 随机生成RSA公钥密钥对
    p = generate_large_Prime()
    q = generate_large_Prime()
    n = p * q
    phi_n = (p - 1) * (q - 1)
    d = int(0)
    e = int(0)
    while True:
        # 随机生成d, 解线性同余方程得到e
        d = random.randint(1, phi_n - 1)
        x, y, gcd = extendgcd(d, phi_n)
        # 检测是否有gcd(d, phi_n) = 1, 若不是则重新生成
        if gcd == 1:
            e = (x % phi_n + phi_n) % phi_n
            break
    # 输出确认信息
    print("Send nB to A")
    # 传送n给用户A
    SendPublicKey2A(str(n))
    # 输出确认信息

```

```

print("Send eB to A")
# 传送e给用户A
SendPublicKey2A(str(e))
# 返回公私钥对，其它信息丢弃
return e, n, d

```

```

def SendPublicKey2A(s):
    # 用户B给用户A发送公钥
    # 创建发送管道SendRSA2Apipe，挂起，等待用户A连接
    p = win32pipe.CreateNamedPipe(r'\\.\pipe\SendRSA2Apipe',
                                   win32pipe.PIPE_ACCESS_DUPLEX,
                                   win32pipe.PIPE_TYPE_MESSAGE | win32pipe.PIPE_WAIT,
                                   1, 65536, 65536, 300, None)

    # 检测到用户A连接管道后传输数据
    win32pipe.ConnectNamedPipe(p, None)
    data = bytes(s.encode('utf-8'))
    win32file.WriteFile(p, data)
    # 输出确认信息
    print(s, "sent to A")

```

```

def receiveRSAfromA():
    # 接收用户A发来的RSA加密数据
    # 连接用户A创建的发送管道SendRSA2Bpipe
    fileHandle = win32file.CreateFile("\\\\.\pipe\SendRSA2Bpipe",
                                       win32file.GENERIC_READ |
win32file.GENERIC_WRITE,
                                       0, None,
                                       win32file.OPEN_EXISTING,
                                       0, None)

    # 从管道中读取信息
    res = win32file.ReadFile(fileHandle, 4096)
    win32file.CloseHandle(fileHandle)
    # 对读到的规格化消息进行处理得到有用信息
    res = str(res[1]).lstrip('b').lstrip('').rstrip('')
    # 输出确认信息
    print(res, "Received from A")
    return res

```

```

def DRSA(c, d, n):
    # 解密RSA信息
    # 这段很简单，套公式即可。
    # d为用户B的私钥，一样是执行一个模乘方运算得到明文数据
    m = fast_power(c, d, n)
    return m

```

```

def send_des_message(key, IV):
    # 将DES Key和IV通过本地私有信道传给DES加密模块进行运算
    # 挂起2秒，等待DES.cpp创建接收Key的管道
    time.sleep(2)
    # 连接本地私有管道Innerpipe
    f=open(r'\\.\Pipe\Innerpipe', 'w')
    # 传输Key
    print("Send DES key", hex(key), "to DES.cpp")
    f.write(hex(key))

```

```

f.close()
# 挂起2秒, 等待DES.cpp重新创建接收IV的管道
time.sleep(2)
# 连接本地私有管道Innerpipe
f=open(r'\\.\Pipe\Innerpipe', 'w')
# 传输IV
print("Send DES IV", hex(IV), "to DES.cpp")
f.write(hex(IV))
f.close()

```

UserB.py的执行逻辑如下:

```

# 生成RSA公私钥对, 发送给A
e, n, d = generate_RSA()
# 输出准备接收加密的Key的信息
print("Receive RSA Key from A")
# 挂起等待传输信道建立
time.sleep(2)
# 接收加密Key
Key_RSA = int(receiverRSAfromA())

# 接下来接收IV与上面的逻辑一样
print("Receive RSA IV from A")
time.sleep(2)
IV_RSA = int(receiverRSAfromA())

# 输出确认信息, 解密Key和IV
print("DRSA for Key")
Key = DRSA(Key_RSA, d, n)
print("DRSA for IV")
IV = DRSA(IV_RSA, d, n)

# 向DES.cpp传送Key和IV数据
send_des_message(Key, IV)

```

DES.cpp

这部分代码主要执行两件事

1. 从本地用户B处获取DES加密所用Key和IV
2. 进行DES加密 (这部分和上次PJ内容完全一样, 不另作介绍)

对应代码和注释如下:

```

int Fetch(string type, DWORD * num_rcv)
{
    // 单次从用户B处申请一个数值, 用type确定是Key还是IV
    // buf_msg为全局数组用于暂存读到的数据, BUF_SIZE为1024, 也是全局的
    // 判断如果type非法, 报错退出
    if(type != "Key" && type != "IV")
    {
        cerr << "Invalid type required!\n";
        return 1;
    }
    // 建立本地私有接收管道Innerpipe, 等待用户B连接
    HANDLE h_pipe;

```

```

h_pipe = ::CreateNamedPipe("\\\\.\\pipe\\Innerpipe", PIPE_ACCESS_INBOUND,
PIPE_READMODE_BYTE | PIPE_WAIT, PIPE_UNLIMITED_INSTANCES, BUF_SIZE, BUF_SIZE, 0,
nullptr);
// 异常处理, 创建管道失败
if (h_pipe == INVALID_HANDLE_VALUE)
{
    if(type == "key")
        cerr << "Failed to create named pipe for Key translation! Error
code: " << ::GetLastError() << "\n";
    else
        cerr << "Failed to create named pipe for IV translation! Error code:
" << ::GetLastError() << "\n";
    return 1;
}
else
{
    if(type == "key")
        cout << "Innerpipe for key translation created successfully\n";
    else
        cout << "Innerpipe for IV translation created successfully\n";
}
// 等待用户B连接管道
if (::ConnectNamedPipe(h_pipe, nullptr))
{
    cout << "UserB connected\n";
    memset(buf_msg, 0, BUF_SIZE);
    // 从管道读取数据
    if (!(::ReadFile(h_pipe, buf_msg, BUF_SIZE, num_rcv, nullptr)))
    {
        if(type == "key")
            cerr << "Failed to receive Key! Error code: " <<
::GetLastError() << "\n";
        else
            cerr << "Failed to receive IV! Error code: " << ::GetLastError()
<< "\n";
        ::CloseHandle(h_pipe);
        return 1;
    }
    if(type == "key")
        cout << "Recieved Key from UserB\n";
    else
        cout << "Recieved IV from UserB\n";
}
::CloseHandle(h_pipe);
return 0;
}

```

```

int GetDESMessage()
{
    // 进行两次数据请求, 从用户B处获得Key和IV, 存在bitset<64>类型的全局变量中、
    // 先请求Key再请求IV
    DWORD Knum_rcv = 0, IVnum_rcv = 0;
    // 请求Key
    int TranslationK = Fetch("key", &Knum_rcv);
    if(TranslationK)
        return 1;
    cout << "DES key is " << buf_msg << endl;
}

```

```

// 将接收到的char数组转换为bitset<64>类型
for(int pos = Knum_rcv-1, idx_bitset = 0; pos > 1; pos --, idx_bitset += 4)
{
    int num = (buf_msg[pos]-48 < 10) ? buf_msg[pos] - 48 : buf_msg[pos] -
97 + 10;
    key[idx_bitset] = num & 0x1;
    key[idx_bitset + 1] = (num >> 1) & 0x1;
    key[idx_bitset + 2] = (num >> 2) & 0x1;
    key[idx_bitset + 3] = (num >> 3) & 0x1;
}

// 请求IV, 逻辑与上面相同
int TranslationIV = Fetch("IV", &IVnum_rcv);
if(TranslationIV)
    return 1;
cout << "DES IV is " << buf_msg << endl;
for(int pos = Knum_rcv-1, idx_bitset = 0; pos > 1; pos --, idx_bitset += 4)
{
    int num = (buf_msg[pos]-48 < 10) ? buf_msg[pos] - 48 : buf_msg[pos] -
97 + 10;
    IV[idx_bitset] = num & 0x1;
    IV[idx_bitset + 1] = (num >> 1) & 0x1;
    IV[idx_bitset + 2] = (num >> 2) & 0x1;
    IV[idx_bitset + 3] = (num >> 3) & 0x1;
}
return 0;
}

```

DES.cpp的执行逻辑如下:

```

int main()
{
    // 请求DES加密所需的Key和IV
    int msg = GetDESMessage();
    if(msg == 1)
        return 1;
    // 执行DES加密
    DoDES();
    return 0;
}

```

演示方式

本次PJ由于需要执行多个程序进程, 故编写了Start.bat文件来进行执行, 按照上述执行逻辑, 依次运行DES.cpp, UserB.py, UserA.py, 来完成数据传输和加密。成功完成信息传输后将会在DES.cpp对应的cmd窗口中显示交互信息, DES加密的执行逻辑和上次PJ一样, 按窗口提示操作即可。

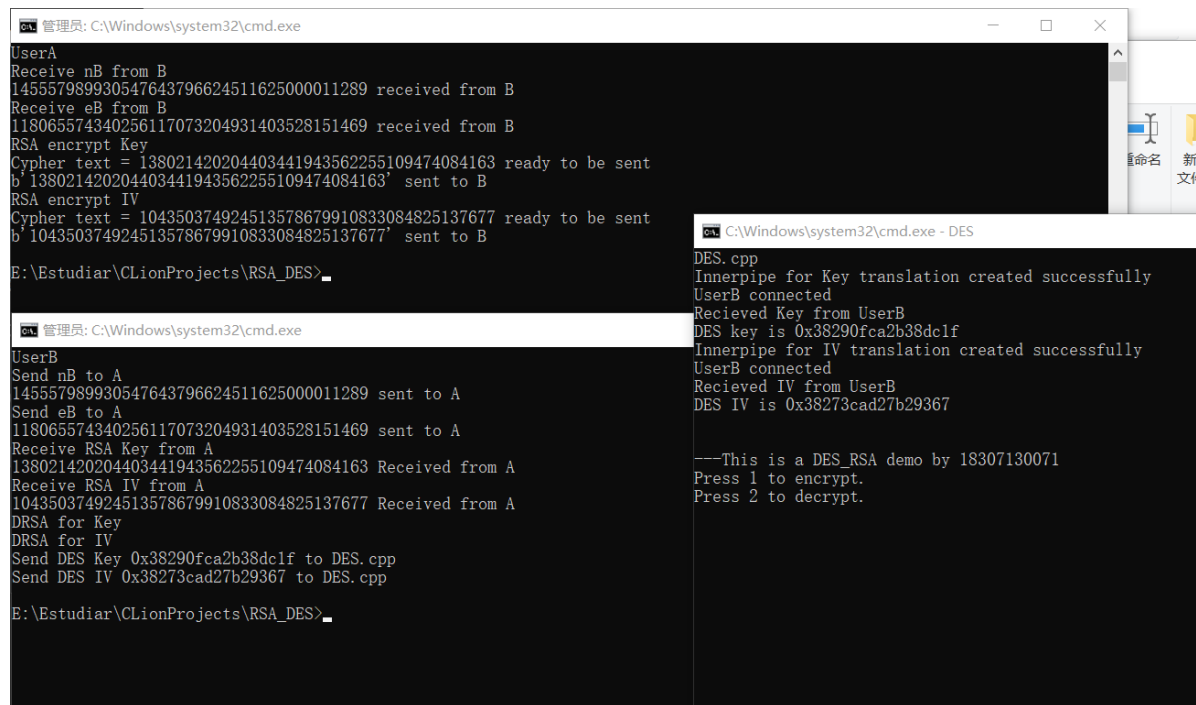
Start.bat的内容如下:

```

# 分别依次打开三个cmd窗口（顺序不能调换）来执行三个程序进程，最后在DES.cpp对应的cmd窗口中按窗口提示完成
# DES加密算法
start cmd /k "echo DES.cpp && g++ DES.cpp -o DES && DES"
start cmd /k "echo UserB && python UserB.py"
start cmd /k "echo UserA && python UserA.py"

```


双击运行Start.bat即可完成演示：



The image shows two overlapping Windows command prompt windows. The top window, titled '管理员: C:\Windows\system32\cmd.exe', shows the execution of a script for UserA. It displays the receipt of a nonce (nB) and an encrypted block (eB) from UserB, followed by the successful encryption of a key and an IV using RSA. The bottom window, titled 'C:\Windows\system32\cmd.exe - DES', shows the execution of DES.cpp. It displays the successful creation of innerpipes for key and IV translation, the receipt of the key and IV from UserB, and the successful encryption of the key and IV using DES. The output of DES.cpp includes a title '---This is a DES_RSA demo by 18307130071' and instructions to press 1 to encrypt and 2 to decrypt.

```
管理员: C:\Windows\system32\cmd.exe
UserA
Receive nB from B
145557989930547643796624511625000011289 received from B
Receive eB from B
118065574340256117073204931403528151469 received from B
RSA encrypt Key
Cypher text = 138021420204403441943562255109474084163 ready to be sent
b'138021420204403441943562255109474084163' sent to B
RSA encrypt IV
Cypher text = 104350374924513578679910833084825137677 ready to be sent
b'104350374924513578679910833084825137677' sent to B
E:\Estudiar\CLionProjects\RSA_DES>

管理员: C:\Windows\system32\cmd.exe
UserB
Send nB to A
145557989930547643796624511625000011289 sent to A
Send eB to A
118065574340256117073204931403528151469 sent to A
Receive RSA Key from A
138021420204403441943562255109474084163 Received from A
Receive RSA IV from A
104350374924513578679910833084825137677 Received from A
DRSA for Key
DRSA for IV
Send DES Key 0x38290fca2b38dc1f to DES.cpp
Send DES IV 0x38273cad27b29367 to DES.cpp
E:\Estudiar\CLionProjects\RSA_DES>

C:\Windows\system32\cmd.exe - DES
DES.cpp
Innerpipe for Key translation created successfully
UserB connected
Recieved Key from UserB
DES key is 0x38290fca2b38dc1f
Innerpipe for IV translation created successfully
UserB connected
Recieved IV from UserB
DES IV is 0x38273cad27b29367

---This is a DES_RSA demo by 18307130071
Press 1 to encrypt.
Press 2 to decrypt.
```

可以看到，UserA和UserB对应的窗口输出了相应的提示信息。待DES.cpp窗口输出DES加密的标题时，即可进行DES加解密操作了，需要加密的文档和明文密文都位与.txt文件中。DES加密的相关注意事项参考PJ1.1。

以上就是本次PJ的全部内容了