

# 《计算机网络》实验报告

信息学院 计算机系统结构 专业 2024 级

实验时间 2024 年 11 月 3 日

姓名 李昊 学号 12024115051

实验名称 实验二 安全通信编程基础

实验成绩

## 一、实验目的

试着修改之前的 TCP 客户端代码，以 PGP 为架构，服务器代码添加 DES 密钥共享（通过 RSA 加密完成）、数字签名和端点鉴别（使用 MD5 计算数据摘要，并在接收端验真）、Base64 转码（如果非 ASCII 码，需要用 Base64 转换编码）和公钥证书验证（用 CA 中心的公钥验证证书后，获得用户的公钥）。

## 二、实验仪器设备及软件

仪器设备：PC

软件：pycharm2024.2.3(Professional Edition),macos,python3.7.16

## 三、实验方案

混合密钥加密：在 DES 加密算法中，加密和解密共享一个相同的对称密钥，因此密钥的安全传输和存储可能需要借助 RSA 非对称密钥加密实现；混合密钥的思想如下：利用非对称密钥来加密对称加密的密钥，然后使用对称加密算法来加密实际需要传输的数据；混合加密结合了 DES 加密大量数据效率高的优点和 RSA 保密性强的优点；

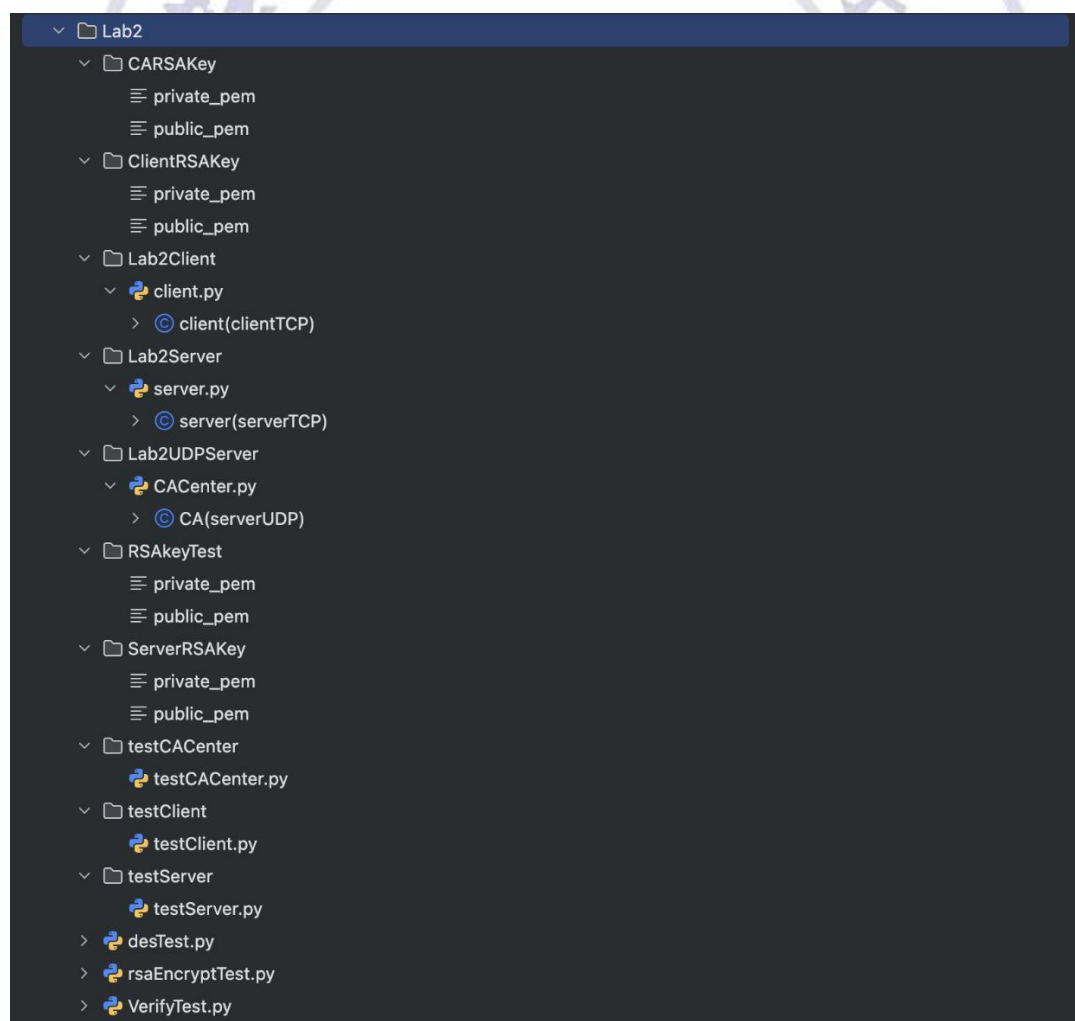
数字签名：数字签名主要用于提供消息或文件真实性和完整性检验。基于 RSA 加密，可以确保信息的发送者身份的真实性，且可以保证信息在传输过程中未被篡改；数字签名同时确保了发送方发送的不可否认性；

公钥证书验证：数字证书由可信的 CA 中心签发，证书中包含公钥和其他身份信息。可以通过这个证书来证明自己拥有对应的私钥；具体来说，发送方向 CA 中心发送个人信息和公钥；CA 打包发送方的个人信息和公钥到数据证书里，再用自己的私钥对数字证书加密得到数字签名后再把这个签名放证书里发给发送方；

**Base64**：将非 ASCII 码数据用 base64 进行转码，确保数据在传输过程中不会因为传输问题而导致数据被修改；

**Hash 计算数据摘要**：hash 计算可以将任意大小的数据输入映射到固定大小的输出，通过在接收端重算对应的 hash 值，可以检验数据在传输过程中是否被修改过；且 hash 函数是单向的，即正着计算 hash 值很简单，但是通过给定 hash 值反向逆推原始数据是不现实的；

综上，分别实现 TCPServer、TCPClient 类和 UDPCA 类，文件结构如下：



---

## 四、实验步骤

### TCPServer 类的属性:

除了继承的基类 TCP 服务器的相关信息外，还额外记录了：DES 密钥，RSA 的公密钥对，CA 中心的公钥及与 CA 中心连接的信息：

```
class server(serverTCP): 2 usages  ⚡ skyfury
    def __init__(self):  ⚡ skyfury
        super().__init__()
        self.des_key = None
        # self.des_key_client = None

        # 不要反复创建rsa密钥对
        # self.create_rsa_key()
        # 记录自己的des密钥
        self.create_des_key()
        # 记录自己的公私密钥对
        key = open("../ServerRSAKey/private.pem").read()
        self.private_key_server = RSA.importKey(key)
        key = open("../ServerRSAKey/public.pem").read()
        self.public_key_server = RSA.importKey(key)

        # 记录客户端的公私密钥对
        key = open("../ClientRSAKey/private.pem").read()
        self.private_key_client = RSA.importKey(key)
        key = open("../ClientRSAKey/public.pem").read()
        self.public_key_client = RSA.importKey(key)

        # 记录CA的公钥
        self.CA_public_key = None
        # 记录与CA的连接信息
        self.CA = {"host": "127.0.0.1", "port": 8888}
        self.CA_socket = None
```

## TCPServer 类的方法:

具体细节见注解:

创建密钥:

```
def create_rsa_key(self):  # skyfury
    """
    生成服务器的一对公私密钥,并写入文件中
    :return:
    """
    random_gen = Random.new().read
    rsa = RSA.generate(bits=1024, random_gen)
    private_pem = rsa.exportKey()
    if not os.path.exists("../ServerRSAKey"):
        os.mkdir("../ServerRSAKey")
    with open('../ServerRSAKey/private_pem', 'wb') as f:
        f.write(private_pem)

    public_pem = rsa.publickey().exportKey()
    with open('../ServerRSAKey/public_pem', 'wb') as f:
        f.write(public_pem)

def create_des_key(self):  # usage  # skyfury
    # 生成8字节的des密钥
    self.des_key = get_random_bytes(8)
```

DES 加解密:

```
def des_decrypt(self, encrypted_data):  # usage  # skyfury
    """
    用des密钥对密文进行解密, des密钥由客户端用服务器端的rsa公钥加密
    :param encrypted_data: 传入的是经过des加密的密文
    :return: 返回解密的数据
    """
    des = pyDes.des(self.des_key, pyDes.CBC, IV=b"\0\0\0\0\0\0\0\0", pad=None, padmode=pyDes.PAD_PKCS5)
    decrypted_data = des.decrypt(encrypted_data)
    return decrypted_data

def des_encrypt(self, data):  # usage  # skyfury
    """
    用des密钥对明文进行加密
    :param data:
    :return:
    """
    des = pyDes.des(self.des_key, pyDes.CBC, IV=b"\0\0\0\0\0\0\0\0", pad=None, padmode=pyDes.PAD_PKCS5)
    encrypted_data = des.encrypt(data)
    return encrypted_data
```

RSA 加解密:

```
def rsa_encrypt(self, data, key): 1 usage 2 skyfury
    pk = PKCS1_v1_5.new(key)
    # 此处的data需要base64编码?
    data_base64 = base64.b64encode(data)
    encrypt_data = pk.encrypt(data_base64)
    result = base64.b64encode(encrypt_data)
    return result

def rsa_decrypt(self, encrypted_data, key): 2 skyfury
    pk = PKCS1_v1_5.new(key)
    decrypt_base64 = base64.b64decode(encrypted_data)
    decrypt_data = pk.decrypt(decrypt_base64, sentinel=0)
    return base64.b64decode(decrypt_data)
```

处理接收客户端信息的函数:

```
# 处理接收客户端信息的部分,不是重写,接收加密数据和数字签名
def handle_data(self, encrypt_data, sign): 1 usage 2 skyfury
    # 收到的数据应该是: data+sign
    # message=input().encode(utf-8)->base64(message)+H(message)->sign(H)+des(base)
    decrypt_data = self.des_decrypt(encrypt_data)
    base64_decode_data = base64.b64decode(decrypt_data)
    message = base64_decode_data.decode("utf-8")

    #用客户端的公钥检验,如果检验通过,则直接打印数据
    self.verify(message, self.public_key_client, sign)
```

对数据进行签名:

```
def sign_data(self, data): 2 skyfury
    """
    对数据进行base64编码,并用服务器端自己的私钥进行签名
    传输数据和签名前均进行base64编码
    :param data:待处理的数据
    :return:tuple(签名, base64编码的数据)
    """
    # 用base64进行转码
    base64_data = base64.b64encode(data)
    # 用SHA256计算hash
    hash_val = SHA256.new(base64_data)

    # 用私钥对hash进行签名并对签名进行Base64编码,以便传输,返回签名和数据
    return base64.b64encode(pkcs1_15.new(self.private_key_server).sign(hash_val)), base64_data
```



## 验证签名：

```
# 接收数据、密钥、签名，检验签名是否正确
def verify(self,data,key,sign): 1 usage  1 skyfury
    """
    接收者使用发送者的公钥来验证签名：
    生成哈希值：接收者对接收到的消息内容再次进行哈希运算，得到消息摘要。
    解密签名：使用发送者的公钥解密签名，得到发送者加密的哈希值
    比较哈希值：如果解密得到的哈希值和接收者自己计算的哈希值一致，验证成功；否则，验证失败，说明消息可能被篡改或签名无效。
    :param data:
    :param key:
    :param sign:
    :return:
    """
    pk = pkcs1_15.new(key)
    try:
        data_hash = SHA256.new(base64.b64encode(data.encode('utf-8')))
        pk.verify(data_hash,sign)
        # 如果没有引发异常，则意味着通过检验
        print(data)
    except (ValueError, TypeError):
        print("warning! data has been change !")
```

## 验证证书：

```
def verify_certificate(self,client_certificate): 1 usage  1 skyfury *
    """
    验证客户端证书(从CA中心领取CA公钥)
    同时需要考虑怎么预处理接收到的certificate
    :param client_certificate:
    :return:
    """
    # 采用UDP连接
    self.CA_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    # 向CA发送公钥请求
    self.CA_socket.sendto("request public key".encode(),(self.CA["host"],self.CA["port"]))
    # 接收CA公钥的位置
    self.CA_public_key,_ = self.CA_socket.recvfrom(1024)
    # 读取CA公钥
    self.CA_public_key =RSA.importKey(open(self.CA_public_key).read())

    # 用公钥对数字证书解密并验证---要用pickle还原数据
    # tuple(information,loc_public_key)->sign(tuple)->(tuple,sign)->pickle(...)
    # 拿到签名和数字证书 (元组类型)
    certificate,sign_base64 = pickle.loads(client_certificate)
    sign = base64.b64decode(sign_base64)
    # 此时再用CA的公钥对certificate进行验证即可
    pk = pkcs1_15.new(self.CA_public_key)
    certificate_byte = pickle.dumps(certificate)
```

```

base64_data = base64.b64encode(certificate_byte)
hash_val = SHA256.new(base64_data)
try:
    pk.verify(hash_val, sign)
    # 如果没有引发异常，则意味着通过检验
    return True
except:
    return False

```

重写处理客户端请求的部分：

```

49
50 # 重写处理客户端请求的部分
51 def handle_client(self, client_socket, client_address):  # skyfury
52     # 接收客户端发送的数字证书
53     client_certificate = client_socket.recv(1024)
54     # 此处应该验证客户端的证书，如果没通过，直接关闭与客户端的连接，就不进入下面的部分了
55     if not self.verify_certificate(client_certificate):
56         client_socket.close()
57         return
58
59     # 用客户端的公钥加密des密钥，用base64编码，并发送
60     # 先对内容用base64编码，再对加密结果再次编码
61     des_key_encrypted = self.rsa_encrypt(self.des_key, self.public_key_client)
62     client_socket.send(des_key_encrypted)
63
64     while True:
65         try:
66             # 接收到的信息应该是经过des加密的数据和签名
67             encrypt_data = client_socket.recv(1024)
68             # 接收客户端的签名
69             signature = client_socket.recv(1024)
70             if encrypt_data==b"":
71                 break
72             # 断开连接的时候message为""
73             print(f"从IP: {client_address[0]} Port: {client_address[1]} 处接受到的信息: ", end='')

```

```

# 处理接收到的数据
self.handle_data(encrypt_data, signature)

response = "ACK"
client_socket.send(self.des_encrypt(response.encode()))
except ConnectionResetError:
    print("oops!! something goes wrong!")
    break

client_socket.close()
self.client.remove(client_socket)
# 断开连接时从列表中移除对应的socket
print(f"-----从IP: {client_address[0]} Port: {client_address[1]} 处断开连接-----")

```

### TCPClient 类的属性:

除了基类 TCP 客户端的相关信息外，还要记录：从 CA 中心请求的电子证书，要放到电子证书里的自己的信息，同 server 一样也要记录与 CA 中心连接的信息，以及自己的公私密钥对和服务器的公私密钥对；

```
class client(clientTCP): 2 usages  👤 skyfury
    def __init__(self):  👤 skyfury
        self.des_key_server = None

        # 记录从CA中心请求的电子证书
        self.certificate = None
        # 自己的信息
        self.information = "I am a client from YNU"

        #CA中心的信息
        self.CA = {"host":"127.0.0.1","port":8888}
        self.CA_socket = None

        # 不要反复创建rsa密钥对
        # self.create_rsa_key()
        # 记录自己的des密钥
        # self.create_des_key()
        # 记录自己的公私密钥对
        key = open("../ClientRSAKey/private_pem").read()
        self.private_key = RSA.importKey(key)
        key = open("../ClientRSAKey/public_pem").read()
        self.public_key = RSA.importKey(key)
        # 记录服务器的公私密钥对
        key = open("../ServerRSAKey/private_pem").read()
        self.private_key_server = RSA.importKey(key)
        key = open("../ServerRSAKey/public_pem").read()
        self.public_key_server = RSA.importKey(key)
```



## TCPClient 类的方法:

创建密钥:

```
def create_rsa_key(self):  # skyfury
    """
    生成客户端的一对公私密钥,并写入文件中
    :return:
    """
    random_gen = Random.new().read
    rsa = RSA.generate(bits=1024, random_gen)
    private_pem = rsa.exportKey()
    if not os.path.exists("../ClientRSAKey"):
        os.mkdir("../ClientRSAKey")
    with open('../ClientRSAKey/private_pem', 'wb') as f:
        f.write(private_pem)

    public_pem = rsa.publickey().exportKey()
    with open('../ClientRSAKey/public_pem', 'wb') as f:
        f.write(public_pem)

# def create_des_key(self):
#     # 生成8字节的des密钥
#     self.des_key = get_random_bytes(8)
```

DES 加解密:

```
def des_encrypt(self,data):  # usage  # skyfury
    des = pyDes.des(self.des_key_server, pyDes.CBC, IV=b"\0\0\0\0\0\0\0\0", pad=None, padmode=pyDes.PAD_PKCS5)
    # 使用初始化向量IV来确保每次加密的密文不同,即使相同的明文和密钥被多次使用。IV通常不需要保密,
    # 但必须确保每次加密时都是唯一的或不可预测的,以防止攻击者推测密文模式或明文内容。
    # CBC模式: 这种模式需要使用一个初始化向量,它在加密过程中会被用到,因此每个加密操作都需要一个新的IV来保证安全性
    # key:即加密密钥,8个字节
    # des算法针对字节
    encrypted_data = des.encrypt(data)
    return encrypted_data

def des_decrypt(self,encrypted_data):  # usage  # skyfury
    des = pyDes.des(self.des_key_server, pyDes.CBC, IV=b"\0\0\0\0\0\0\0\0", pad=None, padmode=pyDes.PAD_PKCS5)
    decrypted_data = des.decrypt(encrypted_data)
    return decrypted_data
```

RSA 加解密:

```
def rsa_encrypt(self, data, key):  # skyfury
    pk = PKCS1_v1_5.new(key)
    data_base64 = base64.b64encode(data)
    encrypt_data = pk.encrypt(data_base64)
    result = base64.b64encode(encrypt_data)
    return result

def rsa_decrypt(self, encrypted_data, key):  # usage  # skyfury
    # 此处收到的encrypted_data是经过base64-rs-base64编码的
    pk = PKCS1_v1_5.new(key)
    decrypt_base64 = base64.b64decode(encrypted_data)
    decrypt_data = pk.decrypt(decrypt_base64, sentinel=0)
    return base64.b64decode(decrypt_data)
```

重写发送数据的函数:

```
8 # 重写发送数据的函数
9 def send_message(self):  # usage  # skyfury
10     while self.is_running:
11         try:
12             ready, _, _ = select.select([sys.stdin], [], [], timeout=1)
13             if ready:
14                 message = input()
15                 # 以下进行数字签名:
16                 # 发送者对消息内容进行哈希运算, 并使用其私钥对哈希值进行加密, 生成签名。这个签名和消息一起发送给接收者。
17
18                 # 对要发送的数据进行base64编码
19                 message = base64.b64encode(message.encode('utf-8'))
20                 # 计算hash并签名---没有使用单独函数实现
21                 hash_val = SHA256.new(message)
22                 # 用客户端的私钥签名
23                 sign = pkcs1_15.new(self.private_key).sign(hash_val)
24                 # 使用des加密要发送的数据
25                 encrypt_data = self.des_encrypt(message)
26                 # 发送des加密数据和签名
27                 self.client_socket.send(encrypt_data)
28                 self.client_socket.send(sign)
29
30         except Exception as e:
31             self.is_running = False
32             print(f"An error occurred while sending message: {e}")
```

发送数字证书:

```
159         # 发送方向接收方发送数字证书
160         def SendCertificate(self): 1 usage  skyfury
161             try:
162                 if self.certificate is not None:
163                     self.client_socket.send(self.certificate)
164                 return
165             except Exception as e:
166                 print(f"failed to fetch a certificate: {e}")
```

向 CA 中心发送请求:

```
135         # 向CA中心发送请求
136         def requestCA(self): 1 usage  skyfury
137             """
138             发送方向CA中心发送个人信息和公钥
139             CA把发送方的个人信息和公钥打包在数字证书里
140             CA用自己的私钥对数字证书进行签名, 再把签名放到证书里
141             CA把数字证书发给发送方
142             :return:
143             """
144             # 采用UDP连接
145             self.CA_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
146             # 发送自己的信息
147             self.CA_socket.sendto(self.information.encode(), (self.CA["host"], self.CA["port"]))
148             # 发送公钥, 把自己公钥的位置发给CA, 让CA自己去读取
149             loc_public_key = "../ClientRSAKey/public.pem"
150             self.CA_socket.sendto(loc_public_key.encode(), (self.CA["host"], self.CA["port"]))
151
152             # 接收CA发送的数字证书
153             self.certificate, _ = self.CA_socket.recvfrom(1024)
154
155             # 关闭CA中心的socket
156             self.CA_socket.close()
157             return
```

重写接收服务器数据的函数:

```
02
03         # 重写接收服务器数据的函数
04         def handle_response_from_server(self, encrypted_response):  skyfury
05             response = self.des_decrypt(encrypted_response)
06             print("来自服务器信息为: ", response.decode('utf-8'))
07
```

重写启动 start 函数:

```
168 # 重写启动start函数
169 def start(self):  # skyfury
170     """
171     启动连接，并用一个新线程监听服务器状态，主线程执行发送信息
172     :return:
173     """
174     self.connect()
175     try:
176         # 先向CA中心请求数字证书
177         self.requestCA()
178         # 向接收方发送自己的数字证书
179         self.SendCertificate()
180     except:
181         # 请求证书失败，自行了断
182         self.client_socket.close()
183         return
184     try:
185         #接收服务器端的des密钥,用客户端的私钥进行解密
186         self.des_key_server = self.rsa_decrypt(self.client_socket.recv(1024),self.private_key)
187         assert self.des_key_server!=0
188         # 解密失败返回0
189     except Exception as e:
190
191     except Exception as e:
192         print(f"oops! {e}")
193         self.client_socket.close()
194         return
195
196 # 多线程，一个用于监听与服务器的连接，一个用于进行收数据
197 listener_thread = threading.Thread(target=self.listen_for_server)
198 listener_thread.start()
199 # 主线程用于和服务器发送信息
200 print("输入要传输的数据")
201 self.send_message()
202 # 等待监听线程结束
203 listener_thread.join()
```

UDPCA 类的属性:

除了继承基类的 UDP 服务器端的相关信息外，还要记录：CA 中心自己的公私密钥对；

```
class CA(serverUDP):  # 2 usages  # skyfury
    def __init__(self):  # skyfury
        super().__init__()

        # 不要反复创建rsa密钥对
        # self.create_rsa_key()

        # 记录自己的公私钥对
        key = open("../CARSAKey/private.pem").read()
        self.CA_private_key = RSA.importKey(key)
        key = open("../CARSAKey/public.pem").read()
        self.CA_public_key = RSA.importKey(key)
```

## UDPCA 类的方法:

CA 创建密钥:

```
def create_rsa_key(self):  # skyfury
    random_gen = Random.new().read
    rsa = RSA.generate(bits=1024, random_gen)
    private_pem = rsa.exportKey()
    if not os.path.exists("../CARSAKey"):
        os.mkdir("../CARSAKey")
    with open('../CARSAKey/private_pem', 'wb') as f:
        f.write(private_pem)

    public_pem = rsa.publickey().exportKey()
    with open('../CARSAKey/public_pem', 'wb') as f:
        f.write(public_pem)
```

生成数字证书:

```
# 生成数字证书 (即简单返回元组)
def generateCertificate(self, information, public_key):  # usage  # skyfury
    return information, public_key
```

对数字证书签名:

```
# 对数字证书签名
def signCertificate(self, certificate):  # usage  # skyfury
    # 考虑先把certificate(元组转换为字节)
    certificate_byte = pickle.dumps(certificate)

    base64_data = base64.b64encode(certificate_byte)
    # 用SHA256计算hash
    hash_val = SHA256.new(base64_data)
    # 用私钥对hash进行签名并对签名进行Base64编码, 以便传输, 返回签名和数据
    return base64.b64encode(pkcs1_15.new(self.CA_private_key).sign(hash_val))
```

检查请求发起者信息:

```
36
37     # 检查请求的发起者是否合法
38     def checkLegality(self, request):  # usage  # skyfury
39         return "YNU" in request
40
```

## 重写 UDP 监听函数:

```
6      #重写UDP的监听函数|
7  def listen(self):  # skyfury
8      print("-----CA center now online-----")
9      while True:
10         # 接收发送方的请求,先接收信息数据,再接收公钥的位置
11         # 请求信息也可能是接收方发起的,请求CA公钥
12         information,client_address = self.server_socket.recvfrom(1024)
13
14         if "public key" in information.decode():
15             # 把自己的公钥的位置告诉请求者
16             self.server_socket.sendto(' ../CARSAKey/public_pem'.encode(),client_address)
17             continue
18
19         loc_public_key, client_address = self.server_socket.recvfrom(1024)
20
21         # 如果请求发起者不合法,则忽略
22         if not self.checkLegality(information.decode()):
23             continue
24
25         # CA将请求发起者的信息和公钥地址打包进数字证书---元组类型
26         certificate = self.generateCertificate(information,loc_public_key)
27         # CA用自己的私钥对数字证书进行签名
28         sign = self.signCertificate(certificate)
29         # 把数字签名放证书里--又是元组类型
30         certificate = (certificate,sign)
31
32         # CA用自己的私钥对数字证书进行签名
33         sign = self.signCertificate(certificate)
34         # 把数字签名放证书里--又是元组类型
35         certificate = (certificate,sign)
36         # CA将证书发给请求发起者
37         # 怎么把元组转换为bytes? -----
38         certificate_byte = pickle.dumps(certificate)
39         self.server_socket.sendto(certificate_byte,client_address)
```





## 五、实验结果及分析

依次启动 CA 中心，服务器 server 和客户端 client:

```
/opt/anaconda3/envs/pycharm_base/bin/python /Users/lihao/PycharmProjects/socketWeb/Lab2/testCACenter/testCACenter.py
UDP Server正在监听: Host: 127.0.0.1 Port: 8888
-----CA center now online-----
```

```
/opt/anaconda3/envs/pycharm_base/bin/python /Users/lihao/PycharmProjects/socketWeb/Lab2/testServer/testServer.py
TCP Server正在监听: Host: 127.0.0.1 Port: 6666
-----来自IP: 127.0.0.1 Port: 50793 的新连接已经建立-----
```

```
/opt/anaconda3/envs/pycharm_base/bin/python /Users/lihao/PycharmProjects/socketWeb/Lab2/testClient/testClient.py
与IP: 127.0.0.1 PORT: 6666 成功建立连接!
输入要传输的数据
```

分析：分别启动 UDP 的 CA 中心和 TCPServer 和 TCPClient，并打印出了对应的提示信息，同时客户端要发送的数据从终端中进行输入；

从客户端发送数据，测试服务器端的接收情况：

```
/opt/anaconda3/envs/pycharm_base/bin/python /Users/lihao/PycharmProjects/socketWeb/Lab2/testClient/testClient.py
与IP: 127.0.0.1 PORT: 6666 成功建立连接!
输入要传输的数据
这是一条中文测试数据
上一条信息状态: 来自服务器信息为: ACK
this is an English version
上一条信息状态: 来自服务器信息为: ACK
```

分析：分别输入中文测试数据和英文测试数据，检测数据是否被正确转码并传输，客户端同时打印来自服务器的确认信息：ACK；

测试连接超时情况下各个部分的反应：

```
与IP: 127.0.0.1 PORT: 6666 成功建立连接!
输入要传输的数据
这是一条中文测试数据
上一条信息状态: 来自服务器信息为: ACK
this is an English version
上一条信息状态: 来自服务器信息为: ACK
接下来要因为超时导致断开链接了
上一条信息状态: 来自服务器信息为: ACK
oops! something goes wrong!!(probably timeout)
```

```
/opt/anaconda3/envs/pycharm_base/bin/python /Users/lihao/PycharmProjects/socketWeb/Lab2/testServer/testServer.py
TCP Server正在监听: Host: 127.0.0.1 Port: 6666
-----来自IP: 127.0.0.1 Port: 51009 的新连接已经建立-----
从IP: 127.0.0.1 Port: 51009 处接受到的信息: 这是一条中文测试数据
从IP: 127.0.0.1 Port: 51009 处接受到的信息: this is an English version
从IP: 127.0.0.1 Port: 51009 处接受到的信息: 接下来要因为超时导致断开链接了
-----从IP: 127.0.0.1 Port: 51009 处断开连接-----
```

```
/opt/anaconda3/envs/pycharm_base/bin/python /Users/Lihao/PycharmProjects/socketWeb/Lab2/testCACenter/testCACenter.py
UDP Server正在监听: Host: 127.0.0.1 Port: 8888
-----CA center now online-----
server is shutting down !
```

分析：测试当客户端长时间未进行任何操作时，能否自动因为超时断开连接，同时服务器端能不能同步检测到对应的客户端断开连接并打印相应信息（测试继承类是否存在问题，超时检测功能来自 Lab1），同时关闭 CA 服务器时，在控制台打印对应的信息（同样执行的是 Lab1 的相应函数功能）。

### 查看数据在传输过程中的加密情况:

a) 接收端检查证书及加密情况:

接收端收到加密数字证书:

```

class Server(ServerTCP):  # 2 usages  1 skyfury
150     # 重写处理客户端请求的部分
151     def handle_client(self, client_socket, client_address):  # skyfury      client_address: (
152         # 接收客户端发送的数字证书
153         client_certificate = client_socket.recv(1024)  client_certificate: b'\x80\x03C\x16I
154         # 此处应该验证客户端的证书，如果没通过，直接关闭与客户端的连接，就不进入下面的部分了
155         if not self.verify_certificate(client_certificate):
156             client_socket.close()
157             return
158
159         # 用客户端的公钥加密des密钥，用base64编码，并发送
160         # 先对内容用base64编码，再对加密结果再次编码
161         des_key_encrypted = self.rsa_encrypt(self.des_key, self.public_key_client)

```

接收端去 CA 中心请求公钥并验证证书:

此时自身信息是“I am a client from YNU”:

```
class server(serverTCP): 2 usages 1 skifyury*
def verify_certificate(self, client_certificate): 1 usage 1 skifyury* client_certificate: b'\x00\x03C\x16I am a client from YNU'
    # 读取CA的公钥
    self.CA_public_key = RSA.importKey(open(self.CA_public_key).read())

    # 用公钥对数字证书解密并验证---要用pickle还原数据
    # tuple(information, loc_public_key)->sign(tuple)->(tuple, sign)->pickle(...)
    # 拿到签名和数字证书 (元组类型)
    certificate, sign_base64 = pickle.loads(client_certificate) certificate: (b'I am a client from YNU', b'../ClientRSAKey/pk.crt')
    sign = base64.b64decode(sign_base64)

    # 此时再用CA的公钥对certificate进行验证即可
    pk = pkcs1_15.new(self.CA_public_key)
    certificate_byte = pickle.dumps(certificate)
    base64_data = base64.b64encode(certificate_byte)
    hash_val = SHA256.new(base64_data)

    try:
        pk.verify(hash_val, sign)
        # 如果没有引发异常, 则意味着通过检验
```

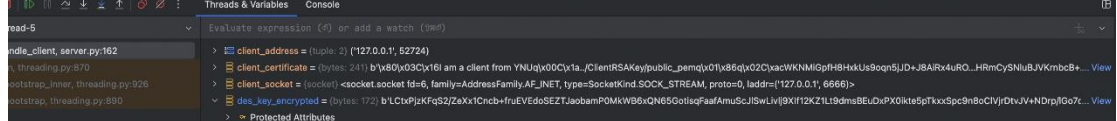
验证证书通过:

```
hash_val = SHA256.new(base64_data) hash_val: <Crypto.Hash.SHA256.SHA256Hash object at 0x7fc5102480d0>
try:
    pk.verify(hash_val, sign)
    # 如果没有引发异常, 则意味着通过检验
return True
```

混合加密, 加密 DES 密钥:

```
# 用客户端的公钥加密des密钥, 用base64编码, 并发送
# 先对内容用base64编码, 再对加密结果再次编码
des_key_encrypted = self.rsa_encrypt(self.des_key, self.public_key_client) des_key_encrypted: b'LCtxPjzKFqS2/ZexX1Cncb+f
client_socket.send(des_key_encrypted)

while True:
    try:
        # 接收到的信息应该是经过des加密的数据和签名
        encrypt_data = client_socket.recv(1024)
        # 接收客户端的签名
        signature = client_socket.recv(1024)
        if encrypt_data == "":
```



后续进入 DES 加密的数据传输环节;

b) 发送方向 CA 请求数字证书, CA 完成检验并发放数字证书:

请求者向 CA 中心请求证书:

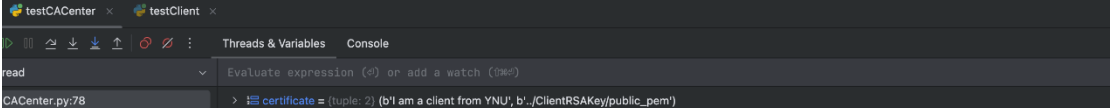
```
# 向CA中心发送请求
def requestCA(self): 1 usage 2 skyfury self: <Lab2.Lab2Client.client.client object at 0x7fbc0085bd10>
    """
    发送方向CA中心发送个人信息和公钥
    CA把发送方的个人信息和公钥打包在数字证书里
    CA用自己的私钥对数字证书进行签名, 再把签名放到证书里
    CA把数字证书发给发送方
    :return:
    """
    # 采用UDP连接
    self.CA_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    # 发送自己的信息
    self.CA_socket.sendto(self.information.encode(), (self.CA["host"], self.CA["port"]))
    # 发送公钥, 把自己公钥的位置发给CA, 让CA自己去读取
    loc_public_key = "../ClientRSAKey/public.pem"
    self.CA_socket.sendto(loc_public_key.encode(), (self.CA["host"], self.CA["port"]))
```

CA 中心检查请求者是否合法:

```
# 检查请求的发起者是否合法
def checkLegality(self, request): 1 usage 2 skyfury request: 'I am a client from YNU' self: <Lab2.L
    return "YNU" in request
```

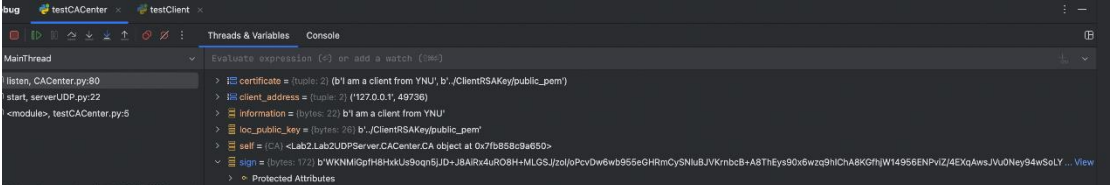
CA 中心打包数字证书:

```
# CA将请求发起者的信息和公钥地址打包进数字证书---元组类型
certificate = self.generateCertificate(information,loc_public_key)  certificate: (b'I am a client from YNU', b'./ClientRSAKey/public.pem')
# CA用自己的私钥对数字证书进行签名
sign = self.signCertificate(certificate)
# 把数字签名放证书里--又是元组类型
certificate = (certificate,sign)
# CA将证书发给请求发起者
# 怎么把元组转换为bytes? -----
certificate_byte = pickle.dumps(certificate)
self.server_socket.sendto(certificate_byte,client_address)
```



CA 中心对数字证书签名:

```
# CA将请求发起者的信息和公钥地址打包进数字证书---元组类型
certificate = self.generateCertificate(information,loc_public_key)  certificate: (b'I am a client from YNU', b'./ClientRSAKey/public.pem')
# CA用自己的私钥对数字证书进行签名
sign = self.signCertificate(certificate)  sign: b'WKNMIGpfH8HxkUs9oqn5jJD+J8A1Rx4uR08H+MLGSJ/zoL/oPcvDw6wb955e6HRmCySnuBjVKnbcB+A8ThEys90x6wzq9hChA8KGHjW14956ENPviZ/4EXqAwsJVu0Ney94wSoLY ... View'
# 把数字签名放证书里--又是元组类型
certificate = (certificate,sign)
# CA将证书发给请求发起者
# 怎么把元组转换为bytes? -----
certificate_byte = pickle.dumps(certificate)
self.server_socket.sendto(certificate_byte,client_address)
```



CA 中心对打包好的数字证书进行处理并发回给请求者:

```
# 怎么把元组转换为bytes? -----
certificate_byte = pickle.dumps(certificate)  certificate_byte: b'\x80\x03C\x16I am a client from YNUq\x00C\x1a../ClientRSAKey/public.pem'
self.server_socket.sendto(certificate_byte,client_address)
```

发送方接收 DES 密钥:

```
try:
    #接收服务器的des密钥,用客户端的私钥进行解密
    self.des_key_server = self.rsa_decrypt(self.client_socket.recv(1024),self.private_key)
    assert self.des_key_server!=0
    # 解密失败返回0
except Exception as e:
    print(f"oops! {e}")
    self.client_socket.close()
    return
# 多线程, 一个用于监听与服务器的连接, 一个用于进行收数据
listener_thread = threading.Thread(target=self.listen_for_server)
listener_thread.start()
# 主线程用于和服务器发送信息
```

后续进入 DES 加密的数据传输环节;

---

## 六、实验总结及体会

### 碰到的问题：

第一：CA 中心在对数字证书签名时，由于在实现数字证书的时候选择是数据结构是元组，即封装成：(请求者信息,公钥)的形式，所以无法对元组结构进行 base64 转码或者使用 socket 的 send 函数进行发送；

第二：无法通过实验指导书上的 pip install Crypto 安装库；

第三：PKCS1\_v1\_5 用于 RSA 加密，pkcs1\_15 用于签名和校验；

### 解决措施：

第一：使用 pickle 库，使用其中的 pickle.dump()函数将 python 对象的元组序列化为字节流，通过对字节流进行转码和传输；接收方使用 pickle.load()函数再将接收到的数据进行反序列化，得到最初的 python 对象；

第二：使用指令：pip install pycryptodome；

### 心得体会：

通过继承父类，在之前的代码上，实现了 DES+RSA 的混合密钥共享，数字签名和公钥证书验证；通过这次实验加深了对 python 中父类继承并重写相关函数的知识，学习了如何使用 crypto 库和 pyDes 库对数据进行 DES 加密，RSA 加密和签名及验证；学习了 CA 中心如何制作数字证书、对其进行签名并发还给请求方；

## 七、代码

实验代码见 Github 仓库：

[YNU-AdvancedNetwork-2024-Lab2](https://github.com/skyfuryonline/AdvancedComputerNetwork2024_Lab2)

备用 URL: [https://github.com/skyfuryonline/AdvancedComputerNetwork2024\\_Lab](https://github.com/skyfuryonline/AdvancedComputerNetwork2024_Lab)



---

## 八、教师评语

