# project-14

Implement a PGP scheme with SM2 基于SM2搭建PGP结构并在真实网络中实现

# 实验原理

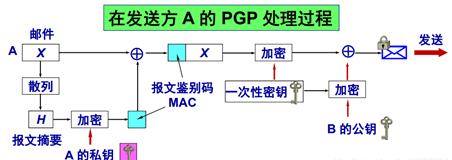
为了防止电子邮件被监听或者截取，保护邮件内容不被第三者所获取，我们使用PGP和公钥体系来为电子邮件提供保密性（Privacy)和认证性(Authentication）。

假设发送者和接收者互相知道对方的公匙。发送者就用接收者的公匙加密邮件寄出，接收者收到后就可以用自己的私匙解密出发送的原文。由于没别人知道接收者的私匙，所以即使是发送者本人也无法解密那封信，这就解决了信件保密的问题。另一方面由于每个人都知道接收者的公匙，他们都可以给该接收者发信，这时接收者就无法确信发送者的身份。这时候就需要用数字签名来认证。

于是我们要求发送者用自己的私匙将邮件的特征值加密，附加在邮件后，再用接收者的公匙将整个邮件加密。这样这份密文被收到以后，接收者再用自己的私匙将邮件解密，得到发送的原文和签名，接收者的PGP也从原文计算出一个特征值来和用甲的公匙解密签名所得到的数比较，这样就确定了发送人的身份。这样两个安全性要求都得到了满足。

# 发送方实现方式

发送者的工作方式如图：

[](https://user-images.githubusercontent.com/129512207/255407100-64019189-0817-4d9b-8938-926c992a8219.png)

1) 对明文邮件 X 进行 hash 运算，得出报文摘要 H。用 A 的私钥对 H 进行加密（即数字签名），得出报文鉴别码 MAC，把它拼接在明文 X 后面，得到扩展的邮件 X || MAC。

2) 使用 A 自己生成的一次性密钥Ks对扩展的邮件X || MAC进行加密。

3) 用 B 的公钥对 A 生成的一次性密钥进行加密，即EB公钥(Ks)。因为加密所用的密钥是一次性的，即密钥只会使用一次，不会出现因为密钥泄露导致之前的加密内容被解密。即使密钥被泄露了，也只会影响一次通信过程。

4) 把加了密的一次性密钥和加了密的扩展的邮件连接（即EB公钥(Ks) ||EKs(Z(sig(H(M)) ||M))）发送给 B。

# 发送者sender的部分实现代码：

def PGP\_sender():

key = b'dfqjlw202100460092'

iv = b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'

crypt\_sm4 = sm4.CryptSM4()

dA,PA=get\_key()

dB, PB = get\_key()

IDA = 12345

IDB = 54321

msg = "dfq202100460092"

#print(sm3\_sign(msg,IDA,dA,PA))

H=sm3.sm3\_hash(list(msg.encode()))

temp=str(sm3\_sign(H,IDA,dA,PA))

#print(temp)

MAC=(temp+msg).encode()

#print(MAC)

crypt\_sm4.set\_key(key, SM4\_ENCRYPT)

encrypt\_value = crypt\_sm4.crypt\_ecb(MAC)

#print(encrypt\_value)# bytes类型

crypt\_sm4.set\_key(key, SM4\_DECRYPT)

MAC = crypt\_sm4.crypt\_ecb(encrypt\_value)

#print(MAC)

encrypt\_value=bytes\_to\_int(encrypt\_value)

encrypt\_key=sm3\_en(key.decode(),PB)

encrypt\_key=bytes\_to\_int(str(encrypt\_key).encode())

print("待加密的明文为："+msg)

print("发送的加密消息：")

print(encrypt\_value)

print(encrypt\_key)

s = socket.socket() # 创建 socket 对象

host = socket.gethostname() # 获取本地主机名

port = 12345 # 设置端口

s.bind((host, port)) # 绑定端口

print("等待接收方链接...")

s.listen(5)

c, addr = s.accept()

print("检测到成功链接")

c.send(str(encrypt\_value).encode())

c.send(str(encrypt\_key).encode())

c.send(str(dB).encode())

c.send(str(len(PA)).encode())

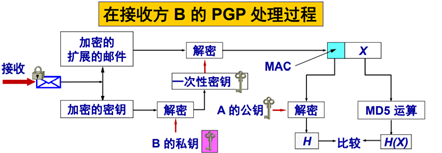
for s in PA:

s=str(s)

c.send(s.encode())

# 接收方实现方式

接收方工作方式如图：

[](https://user-images.githubusercontent.com/129512207/255408066-3686e2d3-48e4-4783-9949-d3d315c47c19.png)

1) 把被加密的一次性密钥EB公钥(Ks)和被加密的扩展报文X || MAC分离开。

2) 用 B 自己的私钥解出 A 的一次性密钥Ks。

3) 用解出的一次性密钥Ks对报文进行解密，然后分离出明文 X 和MAC。

4) 用 A 的公钥对 MAC 进行解密（即签名核实），得出报文摘要 H。这个报文摘要就是 A 原先用明文邮件 X 通过hash运算生成的那个报文摘要。

5) 对签名进行验证：对分离出的明文邮件 X 进行报文摘要运算，得出另一个报文摘要 H(X)。把 H(X) 和前面得出的 H 进行比较，是否和一样。如一样，则对邮件的发送方的鉴别就通过了，报文的完整性也得到肯定。

# 发送者sender的部分实现代码：

def PGP\_receiver():

iv = b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'

crypt\_sm4 = sm4.CryptSM4()

IDA=12345

IDB=54321

def PGP\_get(encrypt\_value,encrypt\_key,dB,PA):

encrypt\_key=int\_to\_bytes(encrypt\_key)

encrypt\_key=json.loads(encrypt\_key.decode())

key=sm3\_de(encrypt\_key,dB).encode()

encrypt\_value = int\_to\_bytes(encrypt\_value)

crypt\_sm4.set\_key(key, SM4\_DECRYPT)

MAC = crypt\_sm4.crypt\_ecb(encrypt\_value)

MAC=MAC.decode()

result = re.match("\[.\*\]", MAC)

sign=json.loads(result.group())

msg\_for\_get=MAC[MAC.find(']')+1:]

print("解密出的消息为: "+msg\_for\_get)

H = sm3.sm3\_hash(list(msg\_for\_get.encode()))

print("验证结果为：",verif\_sign(H, sign, IDA, PA))

print("正在链接发送方...")

s = socket.socket() # 创建 socket 对象

host = socket.gethostname() # 获取本地主机名

port = 12345 # 设置端口号

s.connect((host, port))

print("成功链接发送方")

encrypt\_value = s.recv(1024)

encrypt\_value =int(encrypt\_value.decode())

encrypt\_key = s.recv(1024)

encrypt\_key =int(encrypt\_key.decode())

dB=s.recv(1024)

dB=int(dB.decode())

count = s.recv(1024)

count = int(count.decode())

PA=[0 for i in range(count)]

for i in range(count):

p=s.recv(1024)

p=int(p.decode())

PA[i]=p

print("接收到一次性密钥和加了密的扩展的邮件连接:",encrypt\_value)

print("和",encrypt\_key)

PGP\_get(encrypt\_value,encrypt\_key,dB,PA)

# 实验环境

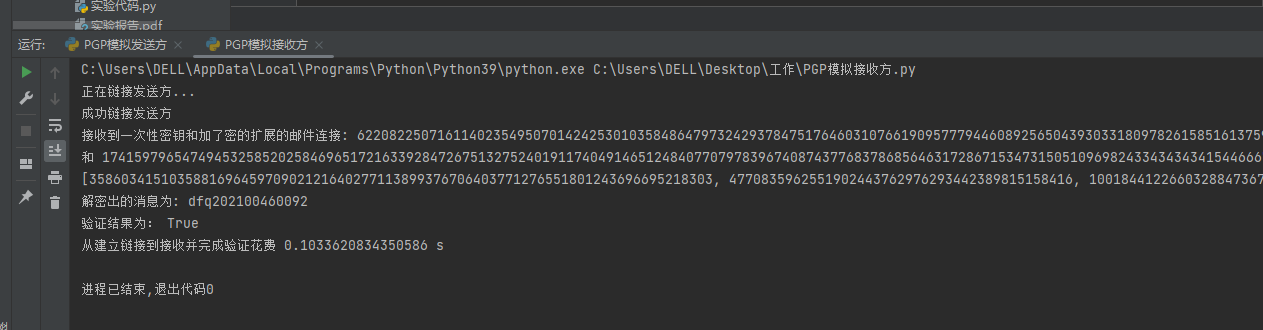
Windows10

visual studio 2022

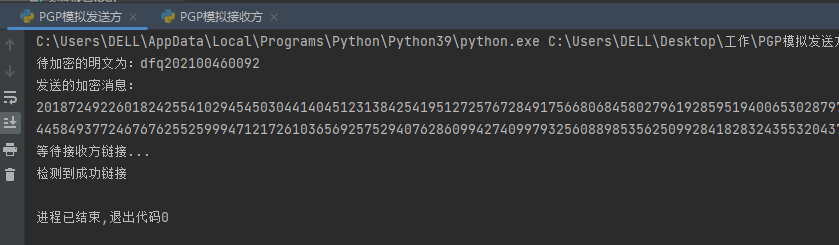
CPU：11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz

# 实验结果

接收方：

[](https://user-images.githubusercontent.com/129512207/255409246-8ba3943d-cab0-44d8-8c79-1a6eed73d7ea.png)

发送方：

[](https://user-images.githubusercontent.com/129512207/255409419-48768f22-e556-4b45-9972-bbe08120a9ed.png)

运行速度：从建立链接到接收并完成验证花费 0.1033620834350586 s

# 小组分工

202100460092 戴方奇单人完成project14