山东大学网络空间安全学院

网络空间安全创新创业实践

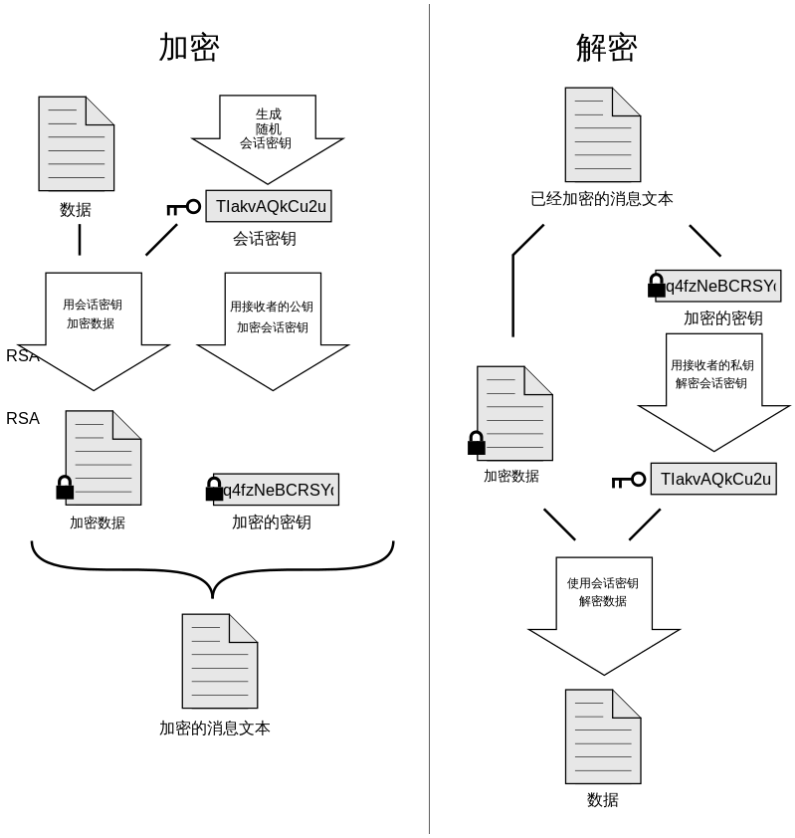
Project 6 PGP

姓名：张麟康

学号：201900301107

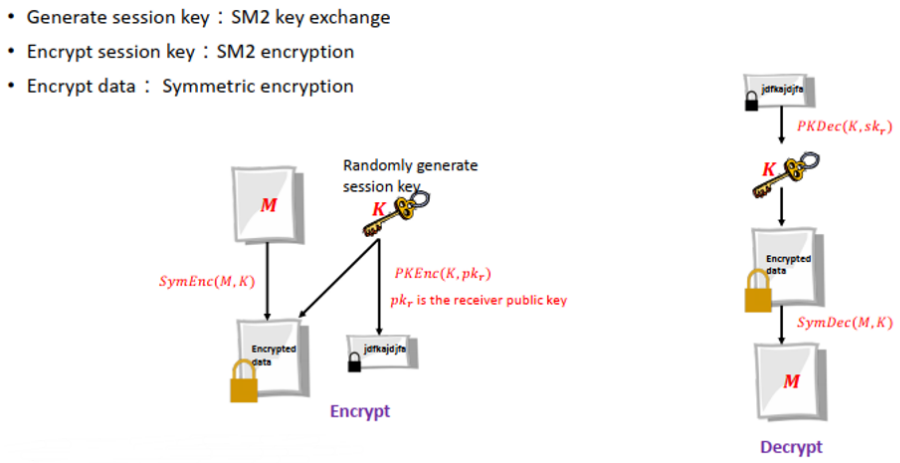
1　原理分析

PGP加密由一系列散列、数据压缩、对称密钥加密以及公钥加密算法组合而成，每个步骤均支持几种算法，根据用户需求选择使用，算法流程图如下所示。



PGP用于发送机密消息，其结合对称加密算法和公钥加密算法，使用对称加密算法对消息进行加密，对称加密密钥需要传送给接收方，因此使用公钥对对称密钥进行解密。

此外，PGP支持身份认证和完整性检查，分别用于确定消息的认证性和完整性。



PGP主要由加密和解密两个过程组成，其中加密在服务器端完成、解密在客户端完成，加解密过程中的具体环节如下所述。

1.1 加密（服务器）

- 使用密码杂凑函数**MD5**对消息M进行摘要处理得到$H(M)$
- 对消息摘要进行签名得到$\sigma(H(M))$
- 将摘要签名和消息级联作为发送的消息$\sigma(H(M))||M$
- 对发送的消息进行压缩记作$Z(\sigma(H(M))||M)$
- 对压缩消息用会话密钥K_s利用对称加密算法**IDEA**进行加密得到$EC_{K_s}(Z(\sigma(H(M))||M))$
- 将该会话密钥用公钥KU_b使用**SM2**加密算法进行加密得到$EP_{KU_b}(K_s)$
- 将加密后的会话密钥与加密后的压缩消息级联发送给接收方记作$EP_{KU_b}(K_s)||EC_{K_s}(Z(\sigma(H(M))||M))$

1.2 解密（客户端）

- 从接收到的消息中提取出$EP_{KU_b}(K_s)$和$EC_{K_s}(Z(\sigma(H(M))||M))$
- 用公钥对应的私钥KU_a对$EP_{KU_b}(K_s)$进行**SM2**解密算法解密得到会话密钥K_s
- 利用会话密钥对$EC_{K_s}(Z(\sigma(H(M))||M))$利用**IDEA**进行解密得到压缩后的签名消息对$Z(\sigma(H(M))||M)$
- 对压缩的消息进行解压缩得到$\sigma(H(M))||M$
- 分离出签名$\sigma(H(M))$和消息M
- 对解密后的邮件数据进行**MD5**压缩，验证生成的数字签名与解密所得签名是否相同

2 具体实现

本项目基于RFC1991、采用Python语言实现。由于PGP支持多种对称密码算法和公钥密码算法，本项目采用IDEA算法加密数据、用SM2算法实现数字签名和对称密钥的加密。

项目架构主要分为服务器端和客户端分别负责加密和解密，假定服务器端和客户端已经透过信赖网络实现密钥交换协议，为了省却繁复的套接字操作而只关注SM2的应用以及PGP的工作原理，本项目采用简单模拟的形式进行PGP协议的实现。

首先利用GMSSL库中的SM2算法初始化，其中SM2算法的公钥和私钥均采用GMSSL库SM2算法示例值，用于进行数字签名的随机值K设置为与私钥一致，利用random库中的getrandombits()生成128比特的对称加密密钥。

```
pk = 'B9C9A6E04E9C91F7BA880429273747D7EF5DDEB0BB2FF6317EB00BEF331A83081A699' \
     '4B8993F3F5D6EADDDB81872266C87C018FB4162F5AF347B483E24620207'
sk = '00B9AB0B828FF68872F21A837FC303668428DEA11DCD1B24429D0C99E24EED83D5'
K = sk
symmetric_key = random.getrandbits(128)
sm2_crypt = sm2.CryptSM2(public_key=pk, private_key=sk)
```

2.1 服务器端

声明server()方法，其参数为原始消息文件路径和加密消息文件路径。

2.1.1 计算哈希值

利用hashlib库中的md5算法进行文件哈希值的计算，在utils.py文件中编写如下函数用于计算对应文件的md5值。

```
def cal_md5(filepath):
    md5 = None
    if os.path.isfile(filepath):
        f = open(filepath, 'rb')
        md5_obj = hashlib.md5()
        md5_obj.update(f.read())
        hash_val = md5_obj.hexdigest()
        f.close()
        md5 = str(hash_val).lower()
    return md5
```

```
print('>>>1.计算邮件哈希值')
hash_val = cal_md5(msg_ori)
print('\t杂凑值:', hash_val)
```

2.1.2 生成文件签名

接下来首先将哈希值转换为16进制数值形式，再以大端形式转换为字节形式，利用SM2的签名算法对哈希值生成签名。

```
print('>>>2.生成文件签名')
msg_ori_bytes = int(hash_val, 16).to_bytes(16, 'big')
# print(msg_ori_bytes)
signature = sm2_crypt.sign(msg_ori_bytes, K)
print('\t数字签名:', signature)
```

2.1.3 消息签名拼接以及压缩

首先将消息和签名进行拼接，添加content和signature标识便于后续处理，利用zlib库进行压缩。

```
print('>>>3.将消息和数字签名拼接')
with open(msg_ori, "rb") as f:
    content = f.read()
data = 'content=' + str(content) + ',signature=' + str(signature)
print('\t拼接结果:', data)

print('>>>4.压缩文件')
msg_compressed = zlib.compress(str.encode(data), zlib.Z_BEST_COMPRESSION)
print('\t压缩结果:', msg_compressed)
```

2.1.4 加密压缩后的数据

实例化一个IDEA对象，将消息分块，利用ECB模式对消息进行加密。

```
print('>>>5.加密压缩后的数据')

idea = IDEA.IDEA(key=symmetric_key)
blocks = IDEA.string_to_blocks(str(msg_compressed))
# print(blocks)
encrypt_IDEA_words = []
for block in blocks:
    encrypt_IDEA_words.append(idea.encrypt(block))
# print(type(symmetric_key), symmetric_key)
```

2.1.5 加密对称密钥

根据SM2加密对象，首先将对称密钥转换为字节形式，利用SM2加密。

```
print('>>>6.加密对称会话密钥')

symmetric_key_cipher = sm2_crypt.encrypt(symmetric_key.to_bytes(16, 'big'))
print('\t原始状态:', symmetric_key)
print('\t密钥明文:', symmetric_key.to_bytes(16, 'big'))
print('\t加密结果:', symmetric_key_cipher)
print('\t解密结果:', sm2_crypt.decrypt(symmetric_key_cipher))
print('\t数值状态:', int.from_bytes(sm2_crypt.decrypt(symmetric_key_cipher),
```

2.1.5.1 级联和编码

首先将加密会话密钥和加密数据进行级联，随后利用base64算法对级联结果进行编码便于在网络上进行传输。

```
print('>>>7.将加密会话密钥和加密数据级联')
msg_to_send = 'data='
tmp = []
for m in encrypt_IDEA_words:
    tmp.append(str(m))
    tmp.append(',')
msg_to_send += ''.join(tmp)

msg_to_send = msg_to_send + 'key=' + str(symmetric_key_cipher)
print('\t级联结果:', msg_to_send.encode('utf-8'))

print('>>>8.级联结果进行Base64编码')
msg_to_send = msg_to_send.encode('utf-8')
data_encoded = base64.b64encode(msg_to_send)
# data_decoded = base64.b64decode(data_encoded)
with open(msg_enc, 'wb') as f:
    f.write(data_encoded)
print('\t最终形态:', data_encoded)
```

2.2 客户端

声明clinet()方法，其参数为加密消息文件路径和解密结果消息文件路径。

2.2.1 解码和分离

对于发送来的密态文件，首先利用base64解码，随后根据数据组织形式将数据和密钥部分分离。

```
# 服务器发来base64编码的密态消息msg_enc
print('>>>1.Base64解码')
with open(msg_enc, 'r') as f:
    msg = f.read()
msg = (base64.b64decode(msg)).decode('utf-8')
print('\t解码结果:', msg)
print('>>>2.密钥和数据分离')
data_extract = msg.split('data=')[1]
print(data_extract)
data_enc = data_extract.split('key=')[0]
key_enc = eval(msg.split('key=')[1])
print('\t数据部分:', type(data_enc), data_enc)
print('\t密钥部分:', type(key_enc), key_enc)
```

2.2.2 解密对称密钥和数据

```
print('>>>3.解密得到对称密钥')
symmetric_key_dec_bytes = sm2_crypt.decrypt(key_enc)
symmetric_key_dec_int = int.from_bytes(symmetric_key_dec_bytes, 'big')
print('\t对称密钥(bytes):', symmetric_key_dec_bytes)
print('\t对称密钥(int):', symmetric_key_dec_int)

print('>>>4.解密数据')
blocks = data_enc.split(',')[:-1]
idea = IDEA.IDEA(key=symmetric_key_dec_int)
decrypt_IDEA_words = []
for block in blocks:
    decrypt_IDEA_words.append(idea.decrypt(int(block)))
msg = blks2str(decrypt_IDEA_words)

msg = bytes(msg[2:-1], encoding='utf-8')
init_msg = codecs.escape_decode(msg, 'hex-escape')
print('压缩后消息:', init_msg)
```

2.2.3 解压缩、签名数据分离及验签

首先利用zlib库对压缩消息进行解压缩如下，然后根据数据组织形式对签名和数据部分进行分离。

```
print('>>>5.解压缩')
data_uncompressed = zlib.decompress(init_msg[0])
print('解压缩结果:', data_uncompressed)
print('>>>6.签名数据分离')
data_uncompressed = str(data_uncompressed)
# print(data_uncompressed)
tmp = data_uncompressed.split('content=', 1)[1]
content, signature = tmp.rsplit('signature=', 1)[0], tmp.rsplit('signature='
content_bytes = bytes(content[2:-2], encoding='utf-8')
with open(msg_dec, 'wb') as f:
    f.write(content_bytes)
content = content[2:-2]
print('\t数据部分:', content)
print('\t签名部分:', signature)
```

接下来计算解密文件的哈希值并将哈希值和签名值作为参数输入SM2的验签算法中，如果验签成功说明消息合法并且未被篡改，反之则消息非法。

```
print('>>>8.验签')
hash_val = cal_md5(msg_dec)
# print(hash_val)
# print(signature)

hash_val_bytes = int(hash_val, 16).to_bytes(16, 'big')
if sm2_crypt.verify(signature, hash_val_bytes):
    print('\tvalid signature.')
else:
    print('\tinvalid signature.')
```

3 运行测试

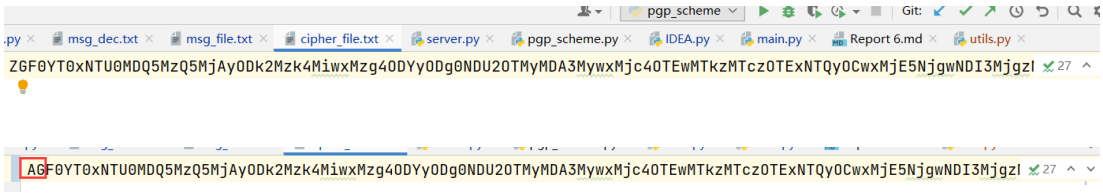
```
if __name__ == '__main__':
    msg_o = 'msg_file.txt'
    msg_e = 'cipher_file.txt'
    msg_d = 'msg_dec.txt'
    print('PGP协议模拟')
    print('************************************************************服务器（加密）**********************************************')
    server(msg_o, msg_e)
    print('************************************************************客户端（解密）**********************************************')
    client(msg_e, msg_d)
```

假设发送的消息为'sdu cyber science and technology'，假设未发生篡改，服务器端和客户端运行结果如下图所示，可见顺利通过验签算法。

```
PGP协议模拟
************************************************************服务器（加密）**********************************************
>>>1.计算邮件哈希值
    杂凑值: a1f10d7379f6d91e59198e5213f5912a
>>>2.生成文件签名
    数字签名: b9c9a6e04e9c91f7ba880429273747d8914eec243526cf4fd7c99a41471014324968fadab88cecdab97a4685e03409899293ff4dc0e9cd5ac4be35f1c29b41cc
>>>3.将消息和数字签名拼接
    拼接结果: content=b'sdu cyber science and technology',signature=b9c9a6e04e9c91f7ba880429273747d8914eec243526cf4fd7c99a41471014324968fadab88cecdab
>>>4.压缩文件
    压缩结果: b'x\xda\r\xcd9\x0e\x840\x10\x05\xd1\xab8#\x99\xc0K{\xe9\x80\xc3\xb4\xbf\xdb\x0c\xd2\xc8H`\x02n?D\xf5\xb2\xc21\xa6\x8e\xb9\xd6\xe5j\xb7\
>>>5.加密压缩后的数据
>>>6.加密对称会话密钥
    原始状态: 183298697271577484183540815140062907043
    密钥明文: b'\x89\xe6\x0b\x84\x16{r(\x8f\xc2\x95\xf4p\x96J\xa3'
    加密结果: b"8\xb0\xf7\xf34\x10\xc15y&\x04.M\r\xb4\xe9(7%\xc3\xa9\x97!e\xd7o\xf3\xa9\xe9\xa2\xa2\x1f\x1d3\xc4e\xfa\xde\xac\x98@\xa1l,K\xcb\xccR\x0
    解密结果: b'\x89\xe6\x0b\x84\x16{r(\x8f\xc2\x95\xf4p\x96J\xa3'
    数值状态: 183298697271577484183540815140062907043
>>>7.将加密会话密钥和加密数据级联
    级联结果: b'data=15540493492028963982,13888628844569320073,12789101931739115428,12196804272834593185,11789271114893372636,6542548819831910312,412
>>>8.级联结果进行Base64编码
    最终形态: b'ZGF0YT0xNTU0MDQ5MzQ5MjAyODk2Mzk4MiwxMzg4ODYyODg0NDU2OTMyMDA3MywxMjc4OTEwMTkzMTczOTExNTQyOCwxMjE5NjgwNDI3MjgzNDU5MzE4NSwxMTc4OTI3MTExN
```

```
************************************************************客户端（解密）**********************************************
>>>1.Base64解码
    解码结果: data=15540493492028963982,13888628844569320073,12789101931739115428,12196804272834593185,11789271114893372636,6542548819831910312,41263
>>>2.密钥和数据分离
15540493492028963982,13888628844569320073,12789101931739115428,12196804272834593185,11789271114893372636,6542548819831910312,412631953314978300,892
    数据部分: <class 'str'> 15540493492028963982,13888628844569320073,12789101931739115428,12196804272834593185,11789271114893372636,6542548819831910
    密钥部分: <class 'bytes'> b"8\xb0\xf7\xf34\x10\xc15y&\x04.M\r\xb4\xe9(7%\xc3\xa9\x97!e\xd7o\xf3\xa9\xe9\xa2\xa2\x1f\x1d3\xc4e\xfa\xde\xac\x98@\xa1
>>>3.解密得到对称密钥
    对称密钥(bytes): b'\x89\xe6\x0b\x84\x16{r(\x8f\xc2\x95\xf4p\x96J\xa3'
    对称密钥(int): 183298697271577484183540815140062907043
>>>4.解密数据
压缩后消息: (b"x\xda\r\xcd9\x0e\x840\x10\x05\xd1\xab8#\x99\xc0K{\xe9\x80\xc3\xb4\xbf\xdb\x0c\xd2\xc8H`\x02n?D\xf5\xb2\xc21\xa6\x8e\xb9\xd6\xe5j\xb7\xc
>>>5.解压缩
解压缩结果: b"content=b'sdu cyber science and technology',signature=b9c9a6e04e9c91f7ba880429273747d8914eec243526cf4fd7c99a41471014324968fadab88cecdab9
>>>6.签名数据分离
    数据部分: sdu cyber science and technology
    签名部分: b9c9a6e04e9c91f7ba880429273747d8914eec243526cf4fd7c99a41471014324968fadab88cecdab97a4685e03409899293ff4dc0e9cd5ac4be35f1c29b41cc
>>>8.验签
    valid signature.
```

经由服务器端加密的密文结果编码如下图所示，篡改一个字符。



再次运行客户端程序，所得结果如下图所示，可以看到解密失败。

```
D:\Python310\python.exe "F:/course-project-2022/Project 6 PGP impl/pgp_scheme.py"
PGP协议模拟
************************************************************服务器（加密）************************************************************
************************************************************客户端（解密）************************************************************
>>>1.Base64解码
    解码结果: data=1637394514805,4290669322378075559,15095529211395301157,8678430474375676884,4210822576087698279,5249903483854017980
>>>2.密钥和数据分离
1637394514805,4290669322378075559,15095529211395301157,8678430474375676884,4210822576087698279,5249903483854017980,2444664226199597
    数据部分: <class 'str'> 1637394514805,4290669322378075559,15095529211395301157,8678430474375676884,4210822576087698279,5249903483
    密钥部分: <class 'bytes'> b'\x925m\x80\xe57_|P\x10\xc5\xe5@\x9e\x85_0\x9b\xc1\xf5\xdbM\x95\xa0\xd6\x90,;^\x81E`\xc7\x98r\xb8\xa4\
>>>3.解密得到对称密钥
    对称密钥(bytes): b'\r\x0c>6\xd0\x08\xe7\xe4|\x8ac\xc0\xaf\x83\x83\xc0'
    对称密钥(int): 17343533359606174984018350429241050048
>>>4.解密数据
Traceback (most recent call last):
  File "F:\course-project-2022\Project 6 PGP impl\pgp_scheme.py", line 142, in <module>
    client(msg_e, msg_d)
  File "F:\course-project-2022\Project 6 PGP impl\pgp_scheme.py", line 102, in client
    msg = blks2str(decrypt_IDEA_words)
  File "F:\course-project-2022\Project 6 PGP impl\utils.py", line 18, in blks2str
    return ''.join([bytearray([block >> (8 * (7 - i)) & 0xFF for i in range(8)]).decode('utf-8') for block in blocks])
  File "F:\course-project-2022\Project 6 PGP impl\utils.py", line 18, in <listcomp>
    return ''.join([bytearray([block >> (8 * (7 - i)) & 0xFF for i in range(8)]).decode('utf-8') for block in blocks])
```

综上完成PGP方案的模拟实现。