

《信息安全综合实践》实验报告

实验名称: 加解密实验姓名: 黎锦灏 学号: 518021910771 邮箱: ljh2000@sjtu.edu.cn 实验时长: 90 分钟

一、实验目的

1. 了解密码技术的应用
2. 学习使用 PGP 软件
3. 学习 OpenSSL 的相关命令及应用
4. 学习和理解数字证书的管理

二、实验内容 (补充填写下表内容) (20 分)

序	内容	实验步骤
1)	使用 PGP 收发加密及签名邮件	1. 利用 PGP 软件 (Windows10 虚拟机已预装, 或自行安装) 生成公私钥; 2. 撰写邮件, 并利用相应密钥对邮件进行加密、签名或加密并签名操作, 发送给合作人; 3. 接收相应邮件, 体验安全邮件功能。 操作系统: Windows 10 邮件软件: 交大邮箱 (mail.sjtu.edu.cn) 合作人及学号: 刘子涵 518021910690
2)	OpenSSL 加解密	文件对称加解密: 1. 加解密算法名称: des3 2. 命令: 加密: openssl enc -des3 -in test.txt -out encrypt.txt -pass pass:123456 解密: openssl enc -des3 -d -in encrypt.txt -out decrypt.txt -pass pass:123456
3)		计算文件摘要 1. 摘要算法名称: sha-1 2. 命令: openssl sha1 -out digest1 test.txt
4)	OpenSSL 证书管理	签发 CA 根证书 (命令) openssl req -config openssl.cnf -new -x509 -days 3650 -key ca.key -out ca.crt
5)		签发客户证书 (命令) openssl req -config openssl.cnf -new -key client.key -out client.csr openssl ca -config openssl.cnf -keyfile ca.key -cert ca.crt -in client.csr -out

		client.pem -days 730
6)		撤销客户证书，并查看证书撤销列表（命令） openssl ca -keyfile ca.key -cert ca.crt -revoke client.pem openssl ca -gencrl -keyfile ca.key -cert ca.crt -out test.crl

三、分析和思考（70 分）

1. 请贴图展示 PGP 中邮件加/解密结果和签名验证结果（不超过 6 张图片），并说明发送者或接收者的公私钥在签名和加密过程中各起什么作用。分析公私钥各自面临怎样的安全威胁，可采取怎样的保护措施。（20 分）

一、实验场景：

在 Windows 10 虚拟机环境下，利用 PGP 生成公钥，邮件交换公钥，发送者用接收者公钥加密，用自己的私钥签名，然后发送加密信息，接收者可解密并通过签名验证发送者身份，完整实现 PGP 中的邮件加/解密。

发送者邮箱（本人 A）：ljh2000@sjtu.edu.cn

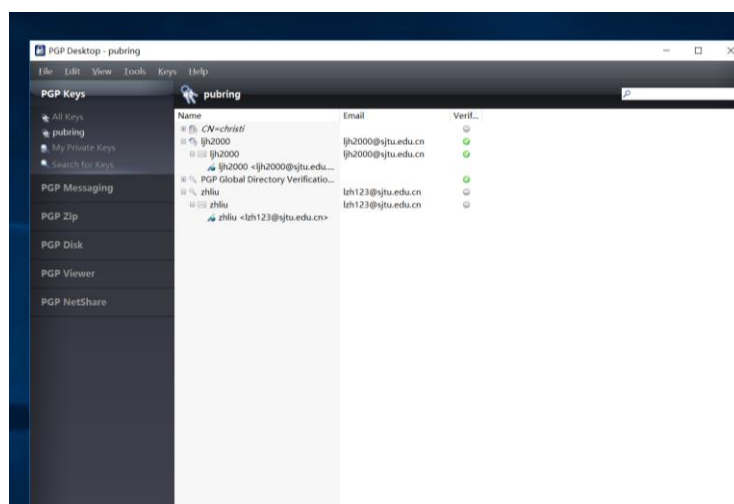
接收者邮箱（合作者 B）：lzh123@sjtu.edu.cn

二、实验过程：

首先，A 与 B 分别生成公钥并交换信息，对于 A 来说，需要传输文件时，先用 B 的公钥加密，再用 A 自己的私钥签名，将加密并签名过的文件发送给 B。B 在接收到邮件之后，可以用 A 的公钥进行签名验证，并使用自己的私钥解密，对比 B 解密后的文件和 A 发送的源文件是否一致，可以检验 PGP 中邮件加解密和签名验证的实现结果。

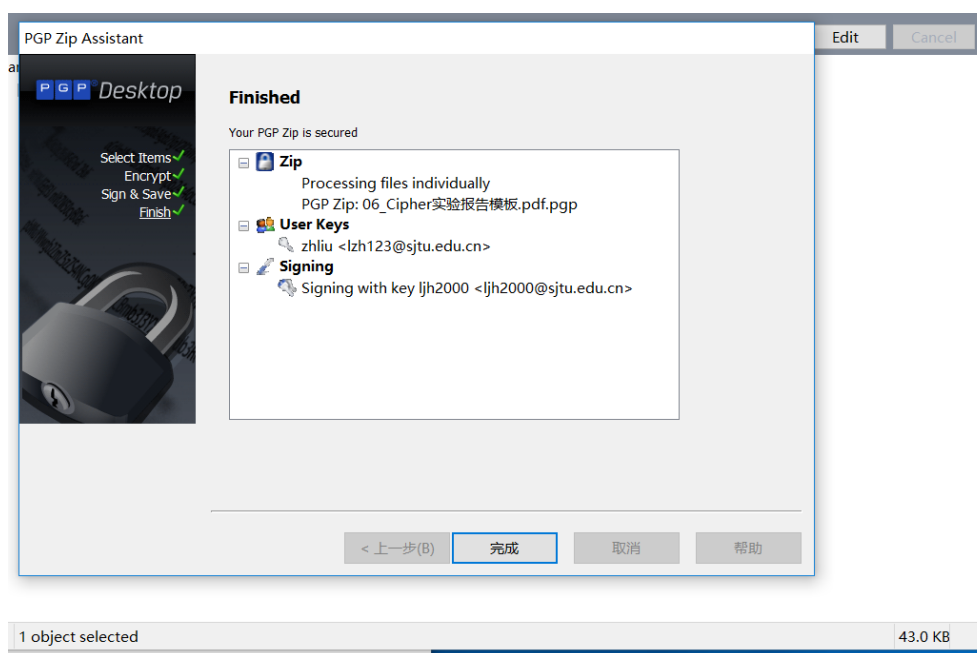
Step 1: 生成密钥并交换公钥

首先利用 PGP 软件生成本人的密钥（公私钥），以邮件形式将公钥文件 pubring.pkr 发送给合作者，同时接收合作者的公钥，实现公钥交换。将收到的 pubring.pkr 在 PGP 软件里导入，可以看到接收者的公钥在列表中显示，导入结果如下图所示：

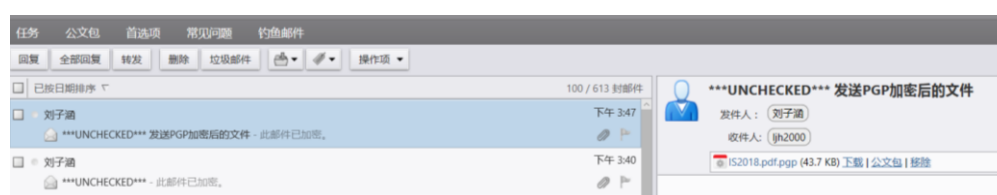


Step 2: 加密并签名待传输文件

在 PGP Zip 中，对本次加解密实验的实验报告模板利用合作者的公钥（User Keys）进行加密，利用自己的私钥签名（Signing），加密与签名后结果如下图所示：



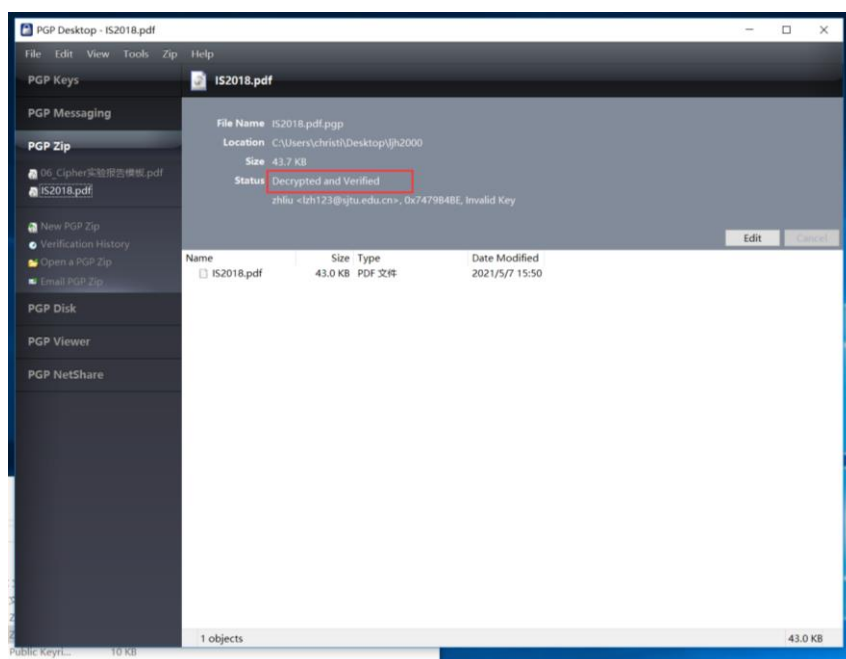
利用邮件发送 PGP 加密后的文件，如下图所示：



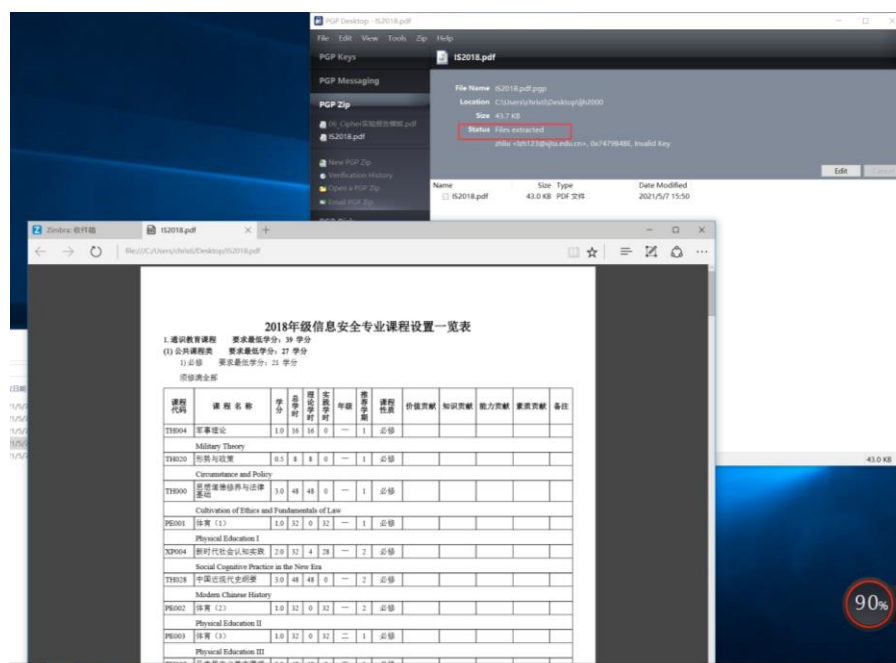
Step 3: 验证签名并解密文件

接收者可以对收到的文件进行解密并验证签名，具体方式为：使用发送者的公钥可以验证发送者的身份，然后用自己的私钥解密可以得到原文件，与发送的原始文件对比可验证加解密过程。

在这里，刘子涵同学加密签名并传输给我的是 2018 级信息安全专业的培养计划，可以看到解密后能正常访问文件，如下图所示：



点击 Extract，可以得到解密文件，与原始文件一致，如下图所示：



三、思考题：

- 公私钥在签名、加密过程中的作用

发送方和接收方都需要保存自己的公私钥，并通过交换得到对方的公钥

接收方公钥：用于发送方对发送文件进行加密；

发送方私钥：用于发送方对发送文件进行签名；

发送方公钥：用于接收方对接受文件的签名验证；

接收方私钥：用于接收方对加密文件进行解密。

- 公私钥面临的**安全威胁：**

公钥本身是公开的，在密钥管理时需要注意分发与交换的权限分配。

私钥是通过 PGP 软件直接生成的，如果 PGP 软件被攻击，攻击者取得了软件权限或直接得到了生成的私钥，可能导致私钥的泄露。用户对密钥管理的疏忽或在不安全的环境中使用私钥，也可能造成私钥被窃取。

- 公私钥的**保护措施：**

公私钥应合理存储并采取最小化权限原则，在本地管理公私钥时应使用风险低、安全性好的公私钥管理软件（例如：PGP）。

需要特别注意的是，在传输私钥时应防止窃听，并且尽量不要随意备份密钥，在备份到云端时应格外关注加密传输的安全问题。

对于个人或组织，建议使用信任的第三方证书管理机构来协助管理密钥，可靠性强，安全性高。

2. 在 OpenSSL 的文件加密、文件摘要以及公私钥生成实验中你采用的是何种算法和命令，请截图说明（不超过 6 张图片）。并用 `openssl speed` 命令分别测试这三种算法的速度，对结果进行分析，并总结不同密码算法的特点和用途。（20 分）

一、文件加解密

OpenSSL 的文件加密过程，我采用的是 **DES3 算法**，其中加解密命令如下：

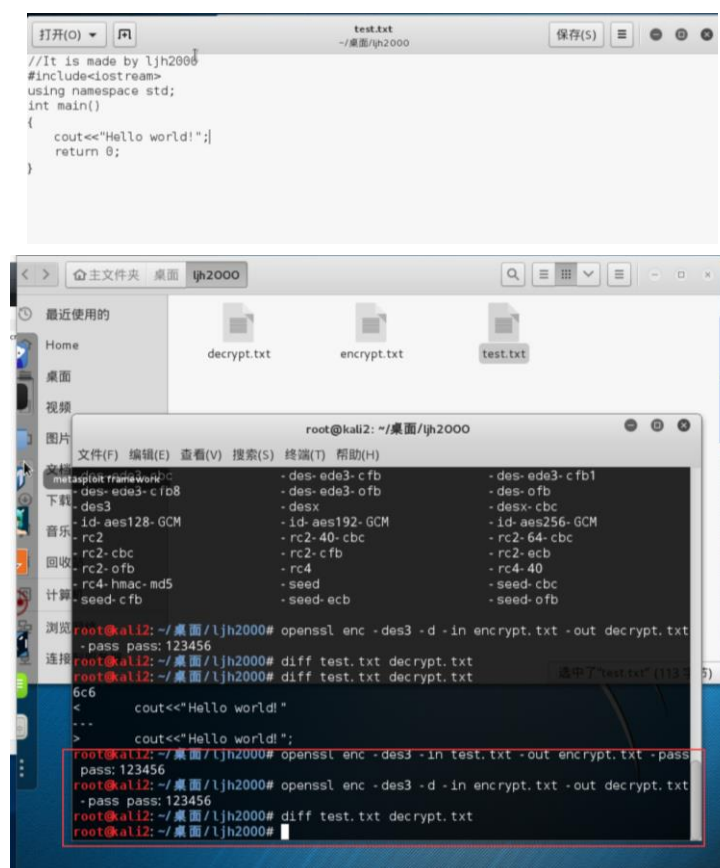
加密：`openssl enc -des3 -in test.txt -out encrypt.txt -pass pass:123456`

解密：`openssl enc -des3 -d -in encrypt.txt -out decrypt.txt -pass pass:123456`

源文件信息后面在终端窗口展示。

对比源文件与解密后的文件，使用 `diff` 命令比较 `decrypt.txt` 和 `test.txt`，发现完全一致，即加解密成功。

加解密结果如下图所示：

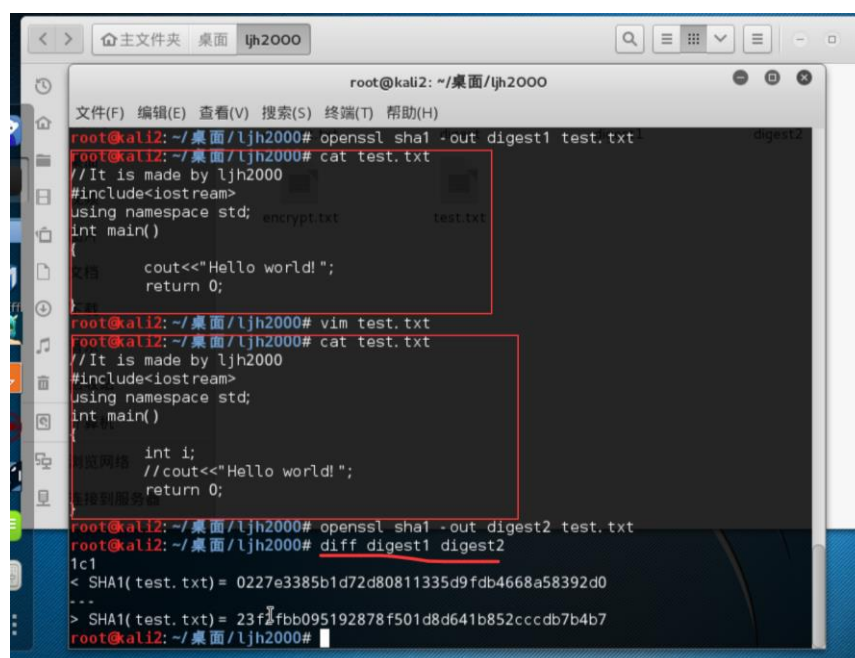


二、文件摘要

在生成文件摘要时，我使用的是 **SHA-1 算法**，同样对加解密实验中用到的 `test.txt` 操作，命令如下：

生成文件摘要：`openssl sha1 -out digest1 test.txt`

修改 `test.txt`，可以看到注释了一行并加入了一行新代码，并对修改后的文件生成摘要，比较修改前后文件摘要的内容差异，如下图所示：



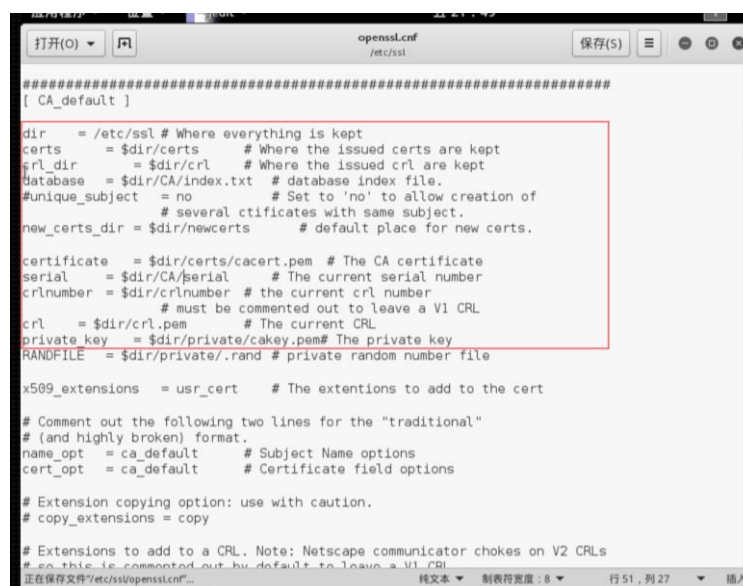
A terminal window on a Kali Linux system. The user is in the directory ~/桌面/ljh2000. The terminal shows the following commands and output:

```
root@kali2: ~/桌面/ljh2000# openssl sha1 -out digest1 test.txt
root@kali2: ~/桌面/ljh2000# cat test.txt
//It is made by ljh2000
#include<iostream>
using namespace std;
int main()
{
    cout<<"Hello world!";
    return 0;
}
root@kali2: ~/桌面/ljh2000# vim test.txt
root@kali2: ~/桌面/ljh2000# cat test.txt
//It is made by ljh2000
#include<iostream>
using namespace std;
int main()
{
    int i;
    //cout<<"Hello world!";
    return 0;
}
root@kali2: ~/桌面/ljh2000# openssl sha1 -out digest2 test.txt
root@kali2: ~/桌面/ljh2000# diff digest1 digest2
1c1
< SHA1( test.txt) = 0227e3385b1d72d80811335d9fdb4668a58392d0
---
> SHA1( test.txt) = 23f2fbb095192878f501d8d641b852cccd7b4b7
root@kali2: ~/桌面/ljh2000#
```

cat 可以看到修改前后 test.txt 的文件具体内容, diff 可以比对并查看摘要的变化。

三、公私钥生成

在公私钥生成时使用的是 3DES+RSA 算法, 按教程要求修改 OpenSSL 的配置文件, 如下图所示:



A screenshot of the OpenSSL configuration file (openssl.cnf) in a text editor. The file is located at /etc/ssl. The following lines are highlighted with a red box:

```
[ CA_default ]

dir = /etc/ssl # Where everything is kept
certs = $dir/certs # Where the issued certs are kept
crl_dir = $dir/crl # Where the issued crl are kept
database = $dir/CA/index.txt # database index file.
#unique_subject = no # Set to 'no' to allow creation of
# several certificates with same subject.
new_certs_dir = $dir/newcerts # default place for new certs.

certificate = $dir/certs/cacert.pem # The CA certificate
serial = $dir/CA/serial # The current serial number
crlnumber = $dir/crlnumber # the current crl number
# must be commented out to leave a V1 CRL
crl = $dir/crl.pem # The current CRL
private_key = $dir/private/cakey.pem # The private key
RANDFILE = $dir/private/.rand # private random number file

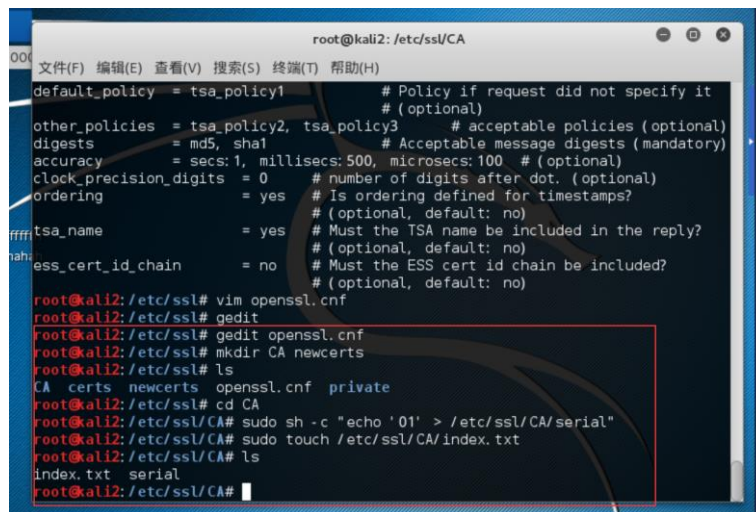
x509_extensions = usr_cert # The extensions to add to the cert

# Comment out the following two lines for the "traditional"
# (and highly broken) format.
name_opt = ca_default # Subject Name options
cert_opt = ca_default # Certificate field options

# Extension copying option: use with caution.
# copy_extensions = copy

# Extensions to add to a CRL. Note: Netscape communicator chokes on V2 CRLs
# so this is commented out by default to leave a V1 CRL
```


修改完配置文件后，在 `/etc/ssl/` 目录下建立两个目录 `CA` 和 `newcerts` 用来存储证书等信息，并在 `/etc/ssl/CA` 目录下建立两个文件，如图所示：



```
root@kali2: /etc/ssl/CA
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
default_policy = tsa_policy1 # Policy if request did not specify it
                                # (optional)
other_policies = tsa_policy2, tsa_policy3 # acceptable policies (optional)
digests = md5, sha1 # Acceptable message digests (mandatory)
accuracy = secs:1, millisecs:500, microsecs:100 # (optional)
clock_precision_digits = 0 # number of digits after dot. (optional)
ordering = yes # Is ordering defined for timestamps?
                                # (optional, default: no)
tsa_name = yes # Must the TSA name be included in the reply?
                                # (optional, default: no)
ess_cert_id_chain = no # Must the ESS cert id chain be included?
                                # (optional, default: no)

root@kali2: /etc/ssl# vim openssl.cnf
root@kali2: /etc/ssl# gedit
root@kali2: /etc/ssl# gedit openssl.cnf
root@kali2: /etc/ssl# mkdir CA newcerts
root@kali2: /etc/ssl# ls
CA certs newcerts openssl.cnf private
root@kali2: /etc/ssl# cd CA
root@kali2: /etc/ssl/CA# sudo sh -c "echo '01' > /etc/ssl/CA/serial"
root@kali2: /etc/ssl/CA# sudo touch /etc/ssl/CA/index.txt
root@kali2: /etc/ssl/CA# ls
index.txt serial
root@kali2: /etc/ssl/CA#
```

然后生成 CA 自签名证书，生成私钥文件 `ca.key`，并签发 CA 公钥证书 `ca.crt`，命令如下：

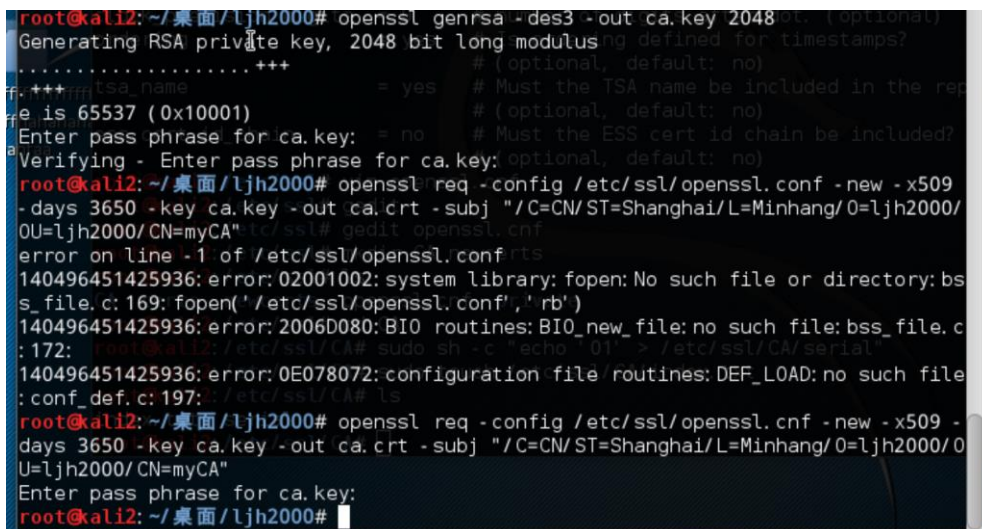
生成私钥文件：

```
openssl req -config openssl.cnf -new -x509 -days 3650 -key ca.key -out ca.crt
```

签发 CA 公钥证书：

```
openssl req -config openssl.cnf -new -x509 -days 3650 -key ca.key -out ca.crt
```

实现结果如下图所示：



```
root@kali2: ~/桌面/ljh2000# openssl genrsa -des3 -out ca.key 2048
Generating RSA private key, 2048 bit long modulus
.....+++
Enter pass phrase for ca.key:
Verifying - Enter pass phrase for ca.key:
root@kali2: ~/桌面/ljh2000# openssl req -config /etc/ssl/openssl.cnf -new -x509
-days 3650 -key ca.key -out ca.crt -subj "/C=CN/ST=Shanghai/L=Minhang/O=ljh2000/
OU=ljh2000/CN=myCA"
root@kali2: /etc/ssl/CA# gedit openssl.cnf
error on line -1 of /etc/ssl/openssl.cnf
140496451425936: error: 02001002: system library: fopen: No such file or directory: bs
s_file.c: 169: fopen('/etc/ssl/openssl.cnf', 'rb')
140496451425936: error: 2006D080: BIO routines: BIO_new_file: no such file: bss_file.c
: 172:
root@kali2: /etc/ssl/CA# sudo sh -c "echo '01' > /etc/ssl/CA/serial"
140496451425936: error: 0E078072: configuration file routines: DEF_LOAD: no such file
: conf_def.c: 197:
root@kali2: /etc/ssl/CA# ls
index.txt serial
root@kali2: /etc/ssl/CA#
```


之后将生成的 CA 公钥证书文件 ca.crt 和私钥文件 ca.key 转移至 /etc/ssl/certs 和 /etc/ssl/private 目录下 (cp 命令, 注意 sudo 权限), 如下图所示:

```
root@kali2: ~/桌面/ljh2000# sudo cp ca.crt /etc/ssl/certs/ca.crt
root@kali2: ~/桌面/ljh2000# sudo cp ca.key /etc/ssl/private/ca.key
root@kali2: ~/桌面/ljh2000#
```

四、OpenSSL speed 测试三种算法的速度

上述三个操作分别采用了:

文件加密: 3DES

文件摘要: SHA-1

公私钥生成: 3DES+RSA

算法速度测试结果如下:

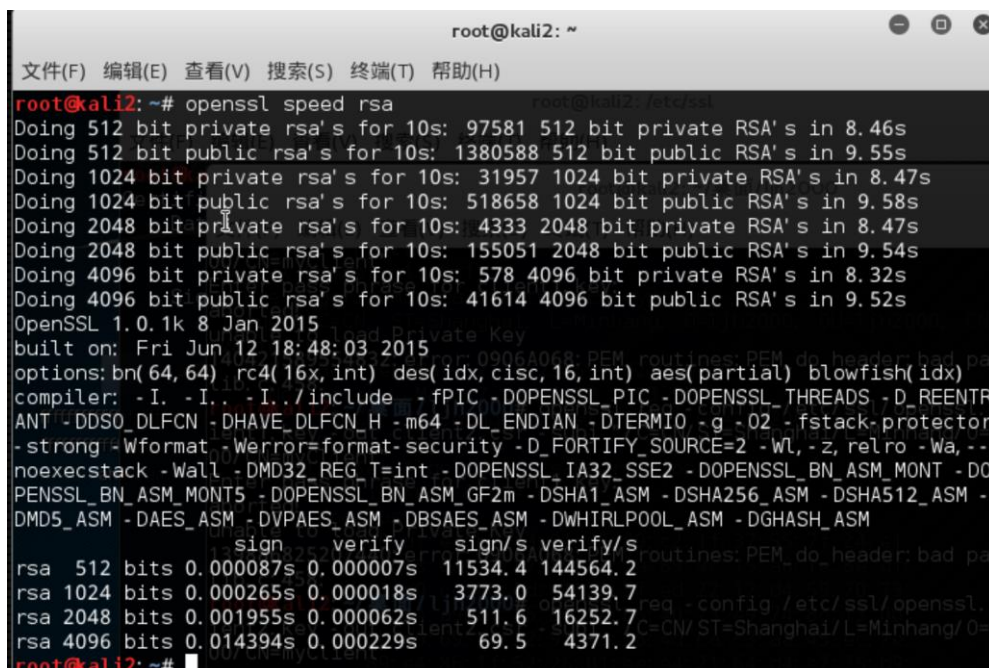
1、3DES

```
root@kali2: ~# openssl speed des-ede3
Doing des ede3 for 3s on 16 size blocks: 2045392 des ede3's in 2.43s
Doing des ede3 for 3s on 64 size blocks: 535211 des ede3's in 2.52s
Doing des ede3 for 3s on 256 size blocks: 133819 des ede3's in 2.52s
Doing des ede3 for 3s on 1024 size blocks: 33102 des ede3's in 2.50s
Doing des ede3 for 3s on 8192 size blocks: 4081 des ede3's in 2.50s
OpenSSL 1.0.1k 8 Jan 2015
built on: Fri Jun 12 18:48:03 2015
options:bn(64,64) rc4(16x,int) des(idx,cisc,16,int) aes(partial) blowfish(idx)
compiler:-I.-I.-I.-I./include -fPIC -DOPENSSL_PIC -DOPENSSL_THREADS -D_REENTRANT -DDSO_DLFCN -DHAVE_DLFCN_H -m64 -DL_ENDIAN -DTERMIO -g -O2 -fstack-protector-strong -Wformat -Werror=format-security -D_FORTIFY_SOURCE=2 -Wl,-z,relro -Wl,-noexecstack -Wall -DMD32_REG_T=int -DOPENSSL_IA32_SSE2 -DOPENSSL_BN_ASM_MONT -DOPENSSL_BN_ASM_MONT5 -DOPENSSL_BN_ASM_GF2m -DSHA1_ASM -DSHA256_ASM -DSHA512_ASM -DMD5_ASM -DAES_ASM -DVPAES_ASM -DBSAES_ASM -DWHIRLPOOL_ASM -DGHASH_ASM
The 'numbers' are in 1000s of bytes per second processed.
type      16 bytes    64 bytes    256 bytes    1024 bytes    8192 bytes
des ede3  13467.60k    13592.66k    13594.31k    13558.58k    13372.62k
root@kali2: ~#
```

2、SHA-1:

```
root@kali2: ~
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
root@kali2: ~# openssl speed sha1
Doing sha1 for 3s on 16 size blocks: 6010090 sha1's in 2.51s
Doing sha1 for 3s on 64 size blocks: 4194349 sha1's in 2.51s
Doing sha1 for 3s on 256 size blocks: 2303666 sha1's in 2.58s
Doing sha1 for 3s on 1024 size blocks: 859678 sha1's in 2.65s
Doing sha1 for 3s on 8192 size blocks: 123882 sha1's in 2.58s
OpenSSL 1.0.1k 8 Jan 2015
built on: Fri Jun 12 18:48:03 2015
options:bn(64,64) rc4(16x,int) des(idx,cisc,16,int) aes(partial) blowfish(idx)
compiler:-I.-I.-I.-I./include -fPIC -DOPENSSL_PIC -DOPENSSL_THREADS -D_REENTRANT -DDSO_DLFCN -DHAVE_DLFCN_H -m64 -DL_ENDIAN -DTERMIO -g -O2 -fstack-protector-strong -Wformat -Werror=format-security -D_FORTIFY_SOURCE=2 -Wl,-z,relro -Wl,-noexecstack -Wall -DMD32_REG_T=int -DOPENSSL_IA32_SSE2 -DOPENSSL_BN_ASM_MONT -DOPENSSL_BN_ASM_MONT5 -DOPENSSL_BN_ASM_GF2m -DSHA1_ASM -DSHA256_ASM -DSHA512_ASM -DMD5_ASM -DAES_ASM -DVPAES_ASM -DBSAES_ASM -DWHIRLPOOL_ASM -DGHASH_ASM
The 'numbers' are in 1000s of bytes per second processed.
type      16 bytes    64 bytes    256 bytes    1024 bytes    8192 bytes
sha1     38311.33k    106947.54k    228580.81k    332192.56k    393349.36k
root@kali2: ~#
```

3、RSA:



```
root@kali2: ~  
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)  
root@kali2: ~# openssl speed rsa  
Doing 512 bit private rsa's for 10s: 97581 512 bit private RSA's in 8.46s  
Doing 512 bit public rsa's for 10s: 1380588 512 bit public RSA's in 9.55s  
Doing 1024 bit private rsa's for 10s: 31957 1024 bit private RSA's in 8.47s  
Doing 1024 bit public rsa's for 10s: 518658 1024 bit public RSA's in 9.58s  
Doing 2048 bit private rsa's for 10s: 4333 2048 bit private RSA's in 8.47s  
Doing 2048 bit public rsa's for 10s: 155051 2048 bit public RSA's in 9.54s  
Doing 4096 bit private rsa's for 10s: 578 4096 bit private RSA's in 8.32s  
Doing 4096 bit public rsa's for 10s: 41614 4096 bit public RSA's in 9.52s  
OpenSSL 1.0.1k 8 Jan 2015  
built on: Fri Jun 12 18:48:03 2015  
options: bn(64,64) rc4(16x,int) des(idx,cisc,16,int) aes(partial) blowfish(idx)  
compiler: -I. -I. -I../include -fPIC -DOPENSSL_PIC -DOPENSSL_THREADS -D_REENTR  
ANT -DDSO_DLFCN -DHAVE_DLFCN_H -m64 -DL_ENDIAN -DTERMIO -g -O2 -fstack-protector  
-strong -Wformat -Werror=format-security -D_FORTIFY_SOURCE=2 -Wl,-z,relro -Wa,--  
noexecstack -Wall -DMD32_REG_T=int -DOPENSSL_IA32_SSE2 -DOPENSSL_BN_ASM_MONT -DO  
PENSSL_BN_ASM_MONT5 -DOPENSSL_BN_ASM_GF2m -DSHA1_ASM -DSHA256_ASM -DSHA512_ASM -D  
MD5_ASM -DAES_ASM -DVPAES_ASM -DBSAES_ASM -DWHIRLPOOL_ASM -DGHASH_ASM  
sign verify sign/s verify/s  
rsa 512 bits 0.000087s 0.000007s 11534.4 144564.2  
rsa 1024 bits 0.000265s 0.000018s 3773.0 54139.7  
rsa 2048 bits 0.001955s 0.000062s 511.6 16252.7  
rsa 4096 bits 0.014394s 0.000229s 69.5 4371.2  
root@kali2: ~#
```

从 OpenSSL speed 的测试中可以看出, 随着明文长度的增加, 三种算法的速度都几乎不变。而对于相同大小的明文来说, 3DES 比 SHA-1 略慢, 且两种算法都比 RSA 快很多, 这是由算法复杂度本身决定的。

五、不同密码算法的特点和用途

3DES: 用于对称加密算法

优点: 1、密钥长度较短;

2、加解密速度快, 处理量大;

3、适用于大文件。

缺点: 1、密钥需要定期更换;

2、大型网络需要保存的密钥量大, 管理难度大;

3、安全性偏低, 密钥需要保密, 而且传递较难。

应用:

3DES 算法主要用于对数据的直接加密, 保证信息的机密性, 还可用于构造各种加密体制(产生伪随机数等)。

SHA-1: 消息摘要算法

SHA-1 是常用的消息摘要算法，将以变长的消息压缩成一个定长的鉴别码。SHA-1 算法不需要密钥，且具有不可逆性，能保证消息的完整性，在输入的消息改变时，输出的摘要也会随之变化。

SHA-1 算法产生了消息摘要，计算速度较快（比需要使用密钥的 MAC 要显著更快），哈希碰撞率较低。

应用：

生成消息摘要，附在正文之前，提供不可抵赖性，接收方也可以通过鉴别码判断消息完整性。

RSA: 公钥加密算法

- 优点：**
- 1、安全性很高，公开公钥，只需要对私钥保密；
 - 2、密钥生命周期较长；
 - 3、可以用于数字签名和密钥交换。

- 缺点：**
- 1、加密速度慢，数据处理量小；
 - 2、密钥长度较长。

应用：

RSA 算法可以在无密钥传输的情况下实现保密通信，常用于数据加解密、密钥协商交换、数字签名，RSA 加密可以保证信息的不可抵赖性、完整性、机密性。（一般公钥用于加密对称加密中的密钥，私钥用于解密和进行数字签名。）

3. 请贴图展示证书签发、CRL 列表签发的结果（不超过 4 张图片）。试分析在证书管理、尤其是证书撤销列表 CRL 机制中可能存在的 3-4 个安全威胁，并给出防范建议。（20 分）

一、证书与 CRL 列表签发

首先生成私钥长度和有效期分别为 1024、2 年，1024、3 年，2048、3 年的客户证书，以生成第一个客户的命令为例，如下图所示：

```
root@kali2: ~/桌面/ljh2000# openssl genrsa -des3 -out client1.key 1024 -days 730
Generating RSA private key, 1024 bit long modulus
.....+++++
.....+++++
e is 65537 (0x10001)
Enter pass phrase for client1.key:
Verifying - Enter pass phrase for client1.key:
root@kali2: ~/桌面/ljh2000# openssl req -config /etc/ssl/openssl.cnf -new -key client1.key -out client1.csr -subj "/C=CN/ST=Shanghai/L=Minhang/O=ljh2000/OU=ljh2000/CN=myClient"
Enter pass phrase for client1.key:
root@kali2: ~/桌面/ljh2000#

Data Base Updated
root@kali2: ~/桌面/ljh2000# openssl ca -config /etc/ssl/openssl.cnf -keyfile ca.key -cert ca.crt -in client1.csr -out client.pem -days 730
```

查看证书详细信息，如下图所示：

实验指导书 课程资料

\$ sudo touch /etc/ssl/CA/index.txt

2.2 签发CA自签名证书

1)生成自签名证书

2)将生成的CA公钥证书文件和私钥文件分别转移至/etc/ssl/certs和/etc/ssl/private目录下

2.3 发放客户证书

1)生成私钥长度和有效期分别为1024、2年，1024、3年，2048、3年的客户证书。

2)查看/etc/ssl/CA和/etc/ssl/newcerts两个目录下有关文件的内容。

2.4 证书撤销

1)撤销刚才发放的客户证书中的前两张证书，并检查证书的更改情况。

2)发布crl列表。

root@kali2: ~/桌面/ljh2000

Using configuration from /etc/ssl/openssl.cnf

Enter pass phrase for ca.key:

Check that the request matches the signature

Signature ok

Certificate Details:

Serial Number: 1 (0x1)

Validity

Not Before: May 7 14:23:59 2021 GMT

Not After : May 7 14:23:59 2023 GMT

Subject:

countryName = CN

stateOrProvinceName = Shanghai

organizationName = ljh2000

organizationalUnitName = ljh2000

commonName = myClient

X509v3 extensions:

X509v3 Basic Constraints:

CA: FALSE

Netscape Comment:

OpenSSL Generated Certificate

X509v3 Subject Key Identifier:

CC: 8B: 98: 0A: 22: DC: 3B: D4: 1B: FC: F1: E7: C0: 72: 40: 27: B7: C4: C7: 8D

X509v3 Authority Key Identifier:

keyid: 23: E0: D4: 13: 5E: 20: EA: 5B: F1: 17: 77: F4: 78: B5: 01: E6: A1: A7: B1: 9

Certificate is to be certified until May 7 14:23:59 2023 GMT (730 days)

Sign the certificate? [y/n]:

然后撤销发布的前两张证书，如下图所示：

```
root@kali2: ~/桌面# openssl ca -keyfile ca.key -cert ca.crt -revoke /etc/ssl/newcerts/01.pem
Using configuration from /usr/lib/ssl/openssl.cnf
Enter pass phrase for ca.key:
Revoking Certificate 01.
Data Base Updated
root@kali2: ~/桌面# openssl ca -keyfile ca.key -cert ca.crt -revoke /etc/ssl/newcerts/02.pem
Using configuration from /usr/lib/ssl/openssl.cnf
Enter pass phrase for ca.key:
Revoking Certificate 02.
Data Base Updated
```

根据实验要求，撤销后检查证书的更改情况。这里查看/etc/ssl/CA 目录下的 index.txt 文件，可以看到因为我们前一步撤销了前两张证书，所以三个证书中前两个已经显示“R”，表示已撤销，而第三个仍是“V”，表示证书有效，如下图所示：

The screenshot shows a terminal window displaying the contents of the index.txt file in the /etc/ssl/CA directory. The file lists three certificates with their serial numbers, unique identifiers, and status. The first two certificates are marked 'R' (Revoked) and the third is marked 'V' (Valid).

Serial Number	Unique Identifier	Status	Details
230507142356Z	210507143325Z	01	unknown / C=CN/ ST=Shanghai/ O=SJTU / OU=IS/ CN=myClient1
240506142503Z	210507143347Z	02	unknown / C=CN/ ST=Shanghai/ O=SJTU / OU=IS/ CN=myClient2
240506142523Z	03	03	unknown / C=CN/ ST=Shanghai/ O=SJTU / OU=IS/ CN=myClient3

二、证书管理（尤其是证书撤销列表 CRL 机制中）的安全威胁及防范建议

安全威胁：

1、攻击者会利用可信的 SSL 证书进行中间人攻击，攻击者可能通过某种途径获得了一个与某网站域名完全相同的适用于网络浏览协议的 SSL 证书，而且该证书被用户的浏览器信任。那么从证书验证的角度来说，这是一个可以被信任的有效证书，则该攻击者就有可能在位于用户与网站之间的网络通路上，伪装为中间人进行攻击，窃取用户的私密信息。

2、OCSP 应答器不可信任时将对证书撤销列表 CRL 机制产生巨大威胁：在 OCSP 中，证书撤销的相关信息认证被委托给信赖的 OCSP 应答器，即证书撤销信息的认证不直接由 CA 的数字签名所担保。此时，OCSP 应答器

必须保证是可信任的,而且由于 OCSP 响应必须经由 OCSP 响应器数字签名来保证完整性,还可能会影响性能,甚至有 DDoS 攻击的风险。

3、对证书撤销列表 CRL 可能存在**重放攻击**: CRL 有两个时间戳,分别是**生成日期**和**下次更新日期**,其中, nextUpdate 日期为 PKIX 配置文件的必填项。在证书被吊销的过程中,如果使用了不安全的通道,那么旧的 CRL 可以重放在 nextUpdate 之前。

4、浏览器开发人员将证书从管理中心删除需要一段时间,在证书需要被撤销但操作尚未被执行的过程中(即尚未进入证书撤销列表 CRL 中),不法分子可能利用该证书进行欺诈行为。

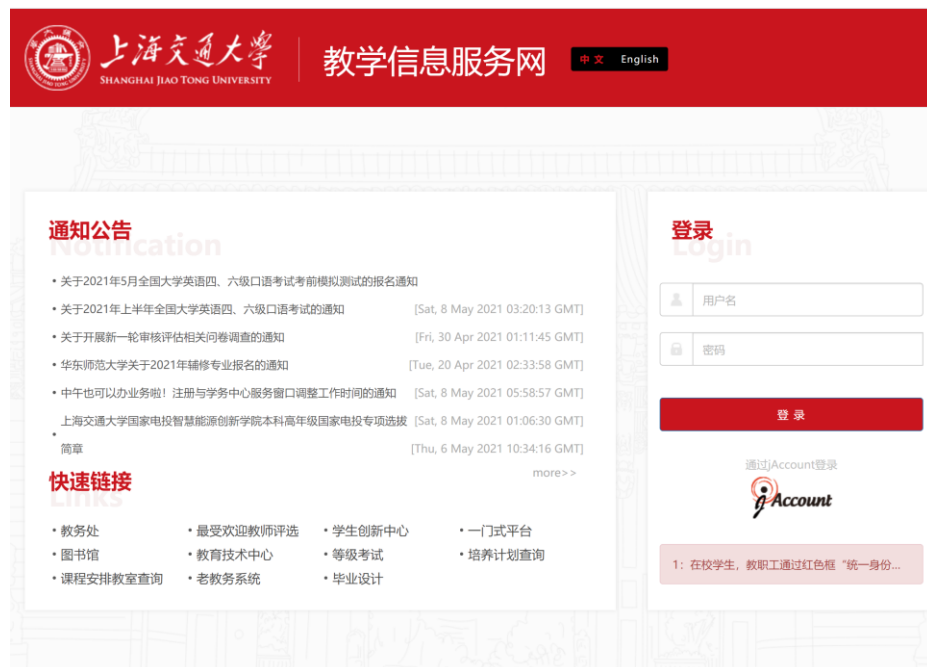
5、Web 浏览器带有大量内置的受信任的根证书列表,这些组织可以颁发任意证书,且会被包含其根证书的 Web 浏览器视为真实证书,但实际上其中许多组织用户并不熟悉。Web 浏览器在证书管理时可能会被这些组织利用,发布欺诈性证书。这种情况只能依靠浏览器管理人员在运行管理过程发现并控制,而且开发人员将这些证书从管理中心删除需要一定时间。

防范建议:

- 1、证书认证机构需鉴别证书申请者提交的身份信息的真伪;
- 2、防止私钥泄露和被破解,通过执行最为严格苛刻的安全管理措施和技术手段来实现私钥保护;
- 3、确保提供服务的 OCSP 应答器是可以信赖,安全性有保障;
- 4、证书提供者需在政府的管辖之下,有关部门出于执法目的可以要求提供者生成任何证书,证书提供者生成证书需备案。

4. 查看一个常用网页的 SSL 证书，分析其证书所给出的信息，如证书所有方、证书发行方等。（10 分）

我选择了平时常用的教学信息服务网（i.sjtu.edu.cn），查看其 SSL 证书。



证书信息：

证书所有方：*.sjtu.edu.cn（即域名 sjtu.edu.cn 下的所有子域名）

证书发行方：TrustAsia OV TLS Pro CA G2

有效期：2020/6/1 到 2022/6/8

签名算法：SHA256 RSA

签名哈希算法：SHA256

使用者：中国上海市上海交通大学网络信息中心

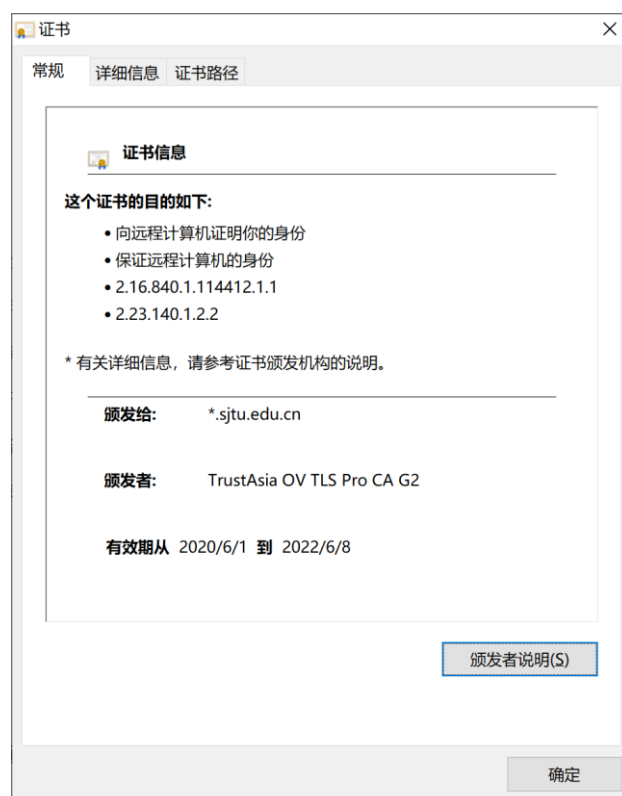
公钥：RSA（2048 bits）

公钥参数：05 00

使用者密钥：f444ee927b8e64c68b1eee6e09aee32af839d159

密钥用法：Digital Signature, Key Encipherment (a0)

概况如下图所示：



证书详细信息如下图所示：



四、实验总结（收获和心得）（5 分）

本次实验主要熟悉了 PGP 和 OpenSSL 的相关命令及应用，对于密码学课程内容有了更深入的认识，也对数字证书管理的工作模式有了基本了解。

在 PGP 实验中，我和刘子涵同学合作完成了收发加密及签名邮件，熟悉了 PGP 的工作原理。由于这部分没有给出实验教程，我在搜了许多博客教学之后，才大致清楚了 PGP 加密、签名、解密的操作方式和完整过程。OpenSSL 实验让我接触了证书的签发流程，加深了对数字证书管理的理解。在签发用户证书的时候遇到了一些问题，在咨询过助教之后我才意识到遗漏了生成 pem 文件这一关键步骤。实验过程中也让我认识到证书对于安全保障的重要作用，在完成课后思考题分析网站的证书信息也让我对了解实际应用中证书签发的流程产生了浓厚兴趣。

总的来说，在使用这些软件进行一些基本的加解密操作的过程中，我了解了加密领域的操作规范和实践细节，将现代密码学课程中学到的 CA 证书、数字签名等知识应用到实际的案例中。

五、尚存问题或疑问、建议（5 分）

问题：

本次实验中，我们主要熟悉了 OpenSSL 的自签名、自认证等证书管理，但对于目前实际应用中的证书管理机构工作模式仍不太了解，在查看了教学信息服务网的 SSL 证书信息之后，我想知道证书管理结构在签发证书和撤销证书时是如何具体管理操作的？

建议：

尽管有预习报告，提前熟悉了实验中的命令，但在实验过程中仍有很多同学遇到许多困难，建议之后在实验前下发预习报告的参考答案，或者在实验手册里添加更详细的指令说明，这样也许能加深同学们对 OpenSSL 的印象，也对具体命令有更深刻的认识。