安全协议与标准课程实验要求

2025.4

1. 在 windows 下（使用 visual studio 或其他）或 LINUX 下编译最新的 OPENSSL 密码安全库（https://www.openssl.org， openssl-1.1.1），生成静态库或动态库。

2. 使用 openssl 工具完成如下操作：

（1）RSA 2048 位 密钥生成；

（2）导出公钥；

（3）生成数字证书请求；

（4）生成数字证书；

（5）生成 pkcs12 格式基于口令保护的标准格式数据包。

（6）用 ASN1 编码解析器，显示以上生成数据对象的 ASN1 编码数据。

3. 调用函数库，编写代码实现以上工具完成的功能（1-6）。

4. 参考分析 openssl 工具包原码的 s_cilent 和 s_server 代码部分，自己编写一个完整的 SSL 客户端和服务器端程序，并能安全通信（配置数字证书），显示执行过程。

5. 撰写实验报告，要求将以上过程描述并附上源代码（*.h, *.c, *.cpp）， 总结编程实验心得。

一、openssl 简介

openssl 是目前最流行的 SSL 密码库工具，其提供了一个通用、健壮、功能完备的工具套件，用以支持 SSL/TLS 协议的实现。

官网：https://www.openssl.org/source/

构成部分

1. 密码算法库
2. 密钥和证书封装管理功能
3. SSL 通信 API 接口

用途

1. 建立 RSA、DH、DSA key 参数
2. 建立 X.509 证书、证书签名请求(CSR)和 CRLs(证书回收列表)
3. 计算消息摘要
4. 使用各种 Cipher 加密/解密
5. SSL/TLS 客户端以及服务器的测试
6. 处理 S/MIME 或者加密邮件

二、RSA 密钥操作

默认情况下，openssl 输出格式为 PKCS#1-PEM

生成 RSA 私钥(无加密)

```
openssl genrsa -out rsa_private.key 2048
```

生成 RSA 公钥

```
openssl rsa -in rsa_private.key -pubout -out rsa_public.key
```

生成 RSA 私钥(使用 aes256 加密)

```
openssl genrsa -aes256 -passout pass:111111 -out rsa_aes_private.key 2048
```

其中 passout 代替 shell 进行密码输入，否则会提示输入密码；

生成加密后的内容如：

```
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-256-CBC,5584D000DDDD53DD5B12AE935F05A007
Base64 Encoded Data
-----END RSA PRIVATE KEY-----
```

此时若生成公钥，需要提供密码

```
openssl rsa -in rsa_aes_private.key -passin pass:111111 -pubout -out rsa_public.key
```

其中 passout 代替 shell 进行密码输入，否则会提示输入密码；

转换命令

私钥转非加密

```
openssl rsa -in rsa_aes_private.key -passin pass:111111 -out rsa_private.key
```

私钥转加密

```
openssl rsa -in rsa_private.key -aes256 -passout pass:111111 -out rsa_aes_private.key
```

私钥 PEM 转 DER

```
openssl rsa -in rsa_private.key -outform der-out rsa_aes_private.der
```

-inform 和-outform 参数制定输入输出格式，由 der 转 pem 格式同理

查看私钥明细

```
openssl rsa -in rsa_private.key -noout -text
```

使用-pubin 参数可查看公钥明细

私钥 PKCS#1 转 PKCS#8

```
openssl pkcs8 -topk8 -in rsa_private.key -passout pass:111111 -out pkcs8_private.key
```

其中-passout 指定了密码，输出的 pkcs8 格式密钥为加密形式，pkcs8 默认采用 des3 加密算法，内容如下：

```
-----BEGIN ENCRYPTED PRIVATE KEY-----
Base64 Encoded Data
-----END ENCRYPTED PRIVATE KEY-----
```

使用-nocrypt 参数可以输出无加密的 pkcs8 密钥，如下：

```
-----BEGIN PRIVATE KEY-----
Base64 Encoded Data
-----END PRIVATE KEY-----
```

三、生成自签名证书

生成 RSA 私钥和自签名证书

```
openssl req -newkey rsa:2048 -nodes -keyout rsa_private.key -x509 -days 365 -out cert.crt
```

req 是证书请求的子命令，-newkey rsa:2048 -keyout private_key.pem 表示生成私钥(PKCS8 格式)，

-nodes 表示私钥不加密，若不带参数将提示输入密码；

-x509 表示输出证书，-days365 为有效期，此后根据提示输入证书拥有者信息；

若执行自动输入，可使用-subj 选项：

```
openssl req -newkey rsa:2048 -nodes -keyout rsa_private.key -x509 -days 365 -out cert.crt -subj "/C=CN/ST=GD/L=SZ/O=vihoo/OU=dev/CN=vivo.com/emailAddress=yy@vivo.com"
```

使用 已有 RSA 私钥生成自签名证书

```
openssl req -new -x509 -days 365 -key rsa_private.key -out cert.crt
```

-new 指生成证书请求，加上-x509 表示直接输出证书，-key 指定私钥文件，其余选项与上述命令相同

四、生成签名请求及 CA 签名

使用 RSA 私钥生成 CSR 签名请求

```
openssl genrsa -aes256 -passout pass:111111 -out server.key 2048
openssl req -new -key server.key -out server.csr
```

此后输入密码、server 证书信息完成，也可以命令行指定各类参数

```
openssl req -new -key server.key -passin pass:111111 -out server.csr -subj "/C=CN/ST=GD/L=SZ/O=vihoo/OU=dev/CN=vivo.com/emailAddress=yy@vivo.com"
```

*** 此时生成的 csr 签名请求文件可提交至 CA 进行签发 ***

查看 CSR 的细节

```
cat server.csr
-----BEGIN CERTIFICATE REQUEST-----
Base64EncodedData
-----END CERTIFICATE REQUEST-----

openssl req -noout -text -in server.csr
```

使用 CA 证书及 CA 密钥 对请求签发证书进行签发，生成 x509 证书

```
openssl x509 -req -days 3650 -in server.csr -CA ca.crt -CAkey ca.key -passin pass:111111 -CAcreateserial -out server.crt
```

其中 CAxxx 选项用于指定 CA 参数输入

五、证书查看及转换

查看证书细节

```
openssl x509 -in cert.crt -noout -text
```

转换证书编码格式

```
openssl x509 -in cert.cer -inform DER -outform PEM -out cert.pem
```

合成 pkcs#12 证书(含私钥)

** 将 pem 证书和私钥转 pkcs#12 证书 **

```
openssl pkcs12 -export -in server.crt -inkey server.key -passin pass:111111 -password pass:111111 -out server.p12
```

其中-export 指导出 pkcs#12 证书,-inkey 指定了私钥文件,-passin 为私钥(文件)密码(nodes 为无加密),-password 指定 p12 文件的密码(导入导出)

** 将 pem 证书和私钥/CA 证书 合成 pkcs#12 证书**

```
openssl pkcs12 -export -in server.crt -inkey server.key -passin pass:111111 \
```

```
    -chain -CAfile ca.crt -password pass:111111 -out server-all.p12
```

其中-chain 指示同时添加证书链，-CAfile 指定了 CA 证书，导出的 p12 文件将包含多个证书。(其他选项：-name 可用于指定 server 证书别名；-caname 用于指定 ca 证书别名)

** pcks#12 提取 PEM 文件(含私钥) **

```
openssl pkcs12 -in server.p12 -password pass:111111 -passout pass:111111 -out out/server.pem
```

其中-password 指定 p12 文件的密码(导入导出)，-passout 指输出私钥的加密密码(nodes 为无加密)

导出的文件为 pem 格式，同时包含证书和私钥(pkcs#8)：

```
Bag Attributes
    localKeyID: 97 DD 46 3D 1E 91 EF 01 3B 2E 4A 75 81 4F 11 A6 E7 1F 79 40
subject=/C=CN/ST=GD/L=SZ/O=vihoo/OU=dev/CN=vihoo.com/emailAddress=yy@vihoo.com
issuer=/C=CN/ST=GD/L=SZ/O=viroot/OU=dev/CN=viroot.com/emailAddress=yy@viroot.com
-----BEGIN CERTIFICATE-----
MIIDazCCAlMCCQCIOlA9/dcfEjANBgkqhkiG9w0BAQUFADB5MQswCQYDVQQGEwJD
1LpQCA+2B6dn4scZwaCD
-----END CERTIFICATE-----
Bag Attributes
    localKeyID: 97 DD 46 3D 1E 91 EF 01 3B 2E 4A 75 81 4F 11 A6 E7 1F 79 40
Key Attributes: <No Attributes>
-----BEGIN ENCRYPTED PRIVATE KEY-----
MIIEvAIBADANBgkqhkiG9w0BAQEFAASCBKYwggSiAgEAAoIBAQDC/6rAc1YaPRNf
K9ZLHbyBTKVaxehjxzJHHw==
-----END ENCRYPTED PRIVATE KEY-----
```

仅提取私钥

```
 openssl pkcs12 -in server.p12 -password pass:111111 -passout pass:111111 -nocerts -out out/key.pem
```

仅提取证书(所有证书)

```
 openssl pkcs12 -in server.p12 -password pass:111111 -nokeys -out out/key.pem
```

仅提取 ca 证书

```
openssl pkcs12 -in server-all.p12 -password pass:111111 -nokeys -cacerts -out out/cacert.pem
```

仅提取 server 证书

```
openssl pkcs12 -in server-all.p12 -password pass:111111 -nokeys -clcerts -out out/cert.pem
```

六、openssl 命令参考

1. openssl list-standard-commands(标准命令)
    1) asn1parse: asn1parse 用于解释用 ANS.1 语法书写的语句(ASN 一般用于定义语法的构成)
    2) ca: ca 用于 CA 的管理

openssl ca [options]:

2.1) -selfsign

使用对证书请求进行签名的密钥对来签发证书。即"自签名"，这种情况发生在生成证书的客户端、签发证书的 CA 都是同一台机器(也是我们大多数实验中的情况)，我们可以使用同一个密钥对来进行"自签名"

2.2) -in file

需要进行处理的 PEM 格式的证书

2.3) -out file

处理结束后输出的证书文件

2.4) -cert file

用于签发的根 CA 证书

2.5) -days arg

指定签发的证书的有效时间

2.6) -keyfile arg

CA 的私钥证书文件

2.7) -keyform arg

CA 的根私钥证书文件格式:

2.7.1) PEM

2.7.2) ENGINE

2.8) -key arg

CA 的根私钥证书文件的解密密码(如果加密了的话)

2.9) -config file

配置文件

example1: 利用 CA 证书签署请求证书

openssl ca -in server.csr -out server.crt -cert ca.crt -keyfile ca.key

3) req: X.509 证书签发请求(CSR)管理

openssl req [options] <infile >outfile

3.1) -inform arg

输入文件格式

3.1.1) DER

3.1.2) PEM

3.2) -outform arg

输出文件格式

3.2.1) DER

3.2.2) PEM

3.3) -in arg

待处理文件

3.4) -out arg

待输出文件

3.5) -passin

用于签名待生成的请求证书的私钥文件的解密密码

3.6) -key file

用于签名待生成的请求证书的私钥文件

3.7) -keyform arg

3.7.1) DER

3.7.2) NET

3.7.3) PEM

3.8) -new

新的请求

3.9) -x509

输出一个 X509 格式的证书

3.10) -days

X509 证书的有效时间

3.11) -newkey rsa:bits

生成一个 bits 长度的 RSA 私钥文件，用于签发

3.12) -[digest]

HASH 算法

3.12.1) md5

3.12.2) sha1

3.12.3) md2

3.12.4) mdc2

3.12.5) md4

3.13) -config file

指定 openssl 配置文件

3.14) -text: text 显示格式

example1: 利用 CA 的 RSA 密钥创建一个自签署的 CA 证书(X.509 结构)

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

example2: 用 server.key 生成证书签署请求 CSR(这个 CSR 用于之外发送待 CA 中心等待签发)

```
openssl req -new -key server.key -out server.csr
```

example3: 查看 CSR 的细节

```
openssl req -noout -text -in server.csr
```

4) genrsa: 生成 RSA 参数

```
openssl genrsa [args] [numbits]
```

[args]

4.1) 对生成的私钥文件是否要使用加密算法进行对称加密:

4.1.1) -des: CBC 模式的 DES 加密

4.1.2) -des3: CBC 模式的 DES 加密

4.1.3) -aes128: CBC 模式的 AES128 加密

4.1.4) -aes192: CBC 模式的 AES192 加密

4.1.5) -aes256: CBC 模式的 AES256 加密

4.2) -passout arg: arg 为对称加密(des、des、aes)的密码(使用这个参数就省去了 console 交互提示输入密码的环节)

4.3) -out file: 输出证书私钥文件

[numbits]: 密钥长度

example: 生成一个 1024 位的 RSA 私钥，并用 DES 加密(密码为 1111)，保存为 server.key 文件

```
openssl genrsa -out server.key -passout pass:1111 -des3 1024
```

5) rsa: RSA 数据管理

openssl rsa [options] <infile >outfile

5.1) -inform arg

输入密钥文件格式:

5.1.1) DER(ASN1)

5.1.2) NET

5.1.3) PEM(base64 编码格式)

5.2) -outform arg

输出密钥文件格式

5.2.1) DER

5.2.2) NET

5.2.3) PEM

5.3) -in arg

待处理密钥文件

5.4) -passin arg

输入这个加密密钥文件的解密密钥(如果在生成这个密钥文件的时候，选择了加密算法了的话)

5.5) -out arg

待输出密钥文件

5.6) -passout arg

如果希望输出的密钥文件继续使用加密算法的话则指定密码

5.7) -des: CBC 模式的 DES 加密

5.8) -des3: CBC 模式的 DES 加密

5.9) -aes128: CBC 模式的 AES128 加密

5.10) -aes192: CBC 模式的 AES192 加密

5.11) -aes256: CBC 模式的 AES256 加密

5.12) -text: 以 text 形式打印密钥 key 数据

5.13) -noout: 不打印密钥 key 数据

5.14) -pubin: 检查待处理文件是否为公钥文件

5.15) -pubout: 输出公钥文件

example1: 对私钥文件进行解密

openssl rsa -in server.key -passin pass:111 -out server_nopass.key

example:2: 利用私钥文件生成对应的公钥文件

openssl rsa -in server.key -passin pass:111 -pubout -out server_public.key

6) x509:

本指令是一个功能很丰富的证书处理工具。可以用来显示证书的内容，转换其格式，给 CSR 签名等 X.509 证书的管理工作

openssl x509 [args]

6.1) -inform arg

待处理 X509 证书文件格式

6.1.1) DER

6.1.2) NET

6.1.3) PEM

6.2) -outform arg

待输出 X509 证书文件格式

6.2.1) DER

6.2.2) NET

6.2.3) PEM

6.3) -in arg

待处理 X509 证书文件

6.4) -out arg

待输出 X509 证书文件

6.5) -req

表明输入文件是一个"请求签发证书文件(CSR)"，等待进行签发

6.6) -days arg

表明将要签发的证书的有效时间

6.7) -CA arg

指定用于签发请求证书的根 CA 证书

6.8) -CAform arg

根 CA 证书格式(默认是 PEM)

6.9) -CAkey arg

指定用于签发请求证书的 CA 私钥证书文件，如果这个 option 没有参数输入，那么缺省认为私有密钥在 CA 证书文件里有

6.10) -CAkeyform arg

指定根 CA 私钥证书文件格式(默认为 PEM 格式)

6.11) -CAserial arg

指定序列号文件(serial number file)

6.12) -CAcreateserial

如果序列号文件(serial number file)没有指定，则自动创建它

example1: 转换 DER 证书为 PEM 格式

openssl x509 -in cert.cer -inform DER -outform PEM -out cert.pem

example2: 使用根 CA 证书对"请求签发证书"进行签发，生成 x509 格式证书

openssl x509 -req -days 3650 -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out server.crt

example3: 打印出证书的内容

openssl x509 -in server.crt -noout -text

7) crl: crl 是用于管理 CRL 列表

openssl crl [args]

7.1) -inform arg

输入文件的格式

7.1.1) DER(DER 编码的 CRL 对象)

7.1.2) PEM(默认的格式)(base64 编码的 CRL 对象)

7.2) -outform arg

指定文件的输出格式

7.2.1) DER(DER 编码的 CRL 对象)

7.2.2) PEM(默认的格式)(base64 编码的 CRL 对象)

7.3) -text:

以文本格式来打印 CRL 信息值。

7.4) -in filename

指定的输入文件名。默认为标准输入。

7.5) -out filename

指定的输出文件名。默认为标准输出。

7.6) -hash

输出颁发者信息值的哈希值。这一项可用于在文件中根据颁发者信息值的哈希值来查询 CRL 对象。

7.7) -fingerprint

打印 CRL 对象的标识。

7.8) -issuer

输出颁发者的信息值。

7.9) -lastupdate

输出上一次更新的时间。

7.10) -nextupdate

打印出下一次更新的时间。

7.11) -CAfile file

指定 CA 文件，用来验证该 CRL 对象是否合法。

7.12) -verify

是否验证证书。

example1: 输出 CRL 文件，包括(颁发者信息 HASH 值、上一次更新的时间、下一次更新的时间)

openssl crl -in crl.crl -text -issuer -hash -lastupdate - nextupdate

example2: 将 PEM 格式的 CRL 文件转换为 DER 格式

openssl crl -in crl.pem -outform DER -out crl.der

8) crl2pkcs7: 用于 CRL 和 PKCS#7 之间的转换

openssl crl2pkcs7 [options] <infile >outfile

转换 pem 到 spc

openssl crl2pkcs7 -nocrl -certfile venus.pem -outform DER -out venus.spc

https://www.openssl.org/docs/apps/crl2pkcs7.html

9) pkcs12: PKCS#12 数据的管理

pkcs12 文件工具，能生成和分析 pkcs12 文件。PKCS#12 文件可以被用于多个项目，例如包含 Netscape、MSIE 和 MS Outlook

openssl pkcs12 [options]

http://blog.csdn.net/as3luyuan123/article/details/16105475

https://www.openssl.org/docs/apps/pkcs12.html

10) pkcs7: PCKS#7 数据的管理

用于处理 DER 或者 PEM 格式的 pkcs#7 文件

openssl pkcs7 [options] <infile >outfile

http://blog.csdn.net/as3luyuan123/article/details/16105407

https://www.openssl.org/docs/apps/pkcs7.html

2. openssl list-message-digest-commands(消息摘要命令)

1) dgst: dgst 用于计算消息摘要

openssl dgst [args]

1.1) -hex

以 16 进制形式输出摘要

1.2) -binary

以二进制形式输出摘要

1.3) -sign file

以私钥文件对生成的摘要进行签名

1.4) -verify file

使用公钥文件对私钥签名过的摘要文件进行验证

1.5) -prverify file

以私钥文件对公钥签名过的摘要文件进行验证

verify a signature using private key in file

1.6) 加密处理

1.6.1) -md5: MD5

1.6.2) -md4: MD4

1.6.3) -sha1: SHA1

1.6.4) -ripemd160

example1: 用 SHA1 算法计算文件 file.txt 的哈西值，输出到 stdout

openssl dgst -sha1 file.txt

example2: 用 dss1 算法验证 file.txt 的数字签名 dsasign.bin，验证的 private key 为 DSA 算法产生的文件 dsakey.pem

openssl dgst -dss1 -prverify dsakey.pem -signature dsasign.bin file.txt

2) sha1: 用于进行 RSA 处理

openssl sha1 [args]

2.1) -sign file

用于 RSA 算法的私钥文件

2.2) -out file

输出文件爱你

2.3) -hex

以 16 进制形式输出

2.4) -binary

以二进制形式输出

example1: 用 SHA1 算法计算文件 file.txt 的 HASH 值,输出到文件 digest.txt

openssl sha1 -out digest.txt file.txt

example2: 用 sha1 算法为文件 file.txt 签名,输出到文件 rsasign.bin，签名的 private key 为 RSA 算法产生的文件 rsaprivate.pem

openssl sha1 -sign rsaprivate.pem -out rsasign.bin file.txt

3. openssl list-cipher-commands (Cipher 命令的列表)

1) aes-128-cbc

2) aes-128-ecb

```
3) aes-192-cbc
4) aes-192-ecb
5) aes-256-cbc
6) aes-256-ecb
7) base64
8) bf
9) bf-cbc
10) bf-cfb
11) bf-ecb
12) bf-ofb
13) cast
14) cast-cbc
15) cast5-cbc
16) cast5-cfb
17) cast5-ecb
18) cast5-ofb
19) des
20) des-cbc
21) des-cfb
22) des-ecb
23) des-ede
24) des-ede-cbc
25) des-ede-cfb
26) des-ede-ofb
27) des-ede3
28) des-ede3-cbc
29) des-ede3-cfb
30) des-ede3-ofb
31) des-ofb
32) des3
33) desx
34) rc2
35) rc2-40-cbc
36) rc2-64-cbc
37) rc2-cbc
38) rc2-cfb
39) rc2-ecb
40) rc2-ofb
41) rc4
42) rc4-40
```