网安“计算机安全与保密技术”课程
实验报告

姓名	张心驰	学号	20120921	院系	计算机学院
课程号	08A65002	任课教师	戴佳筑	指导教师	戴佳筑
实验地点	计算机大楼 704		实验时间	2022.1.6	
出勤、表现得分（10 分）	实验结果得分（50 分）		实验报告得分（40 分）	实验总分	

实验三　RSA 加密和数字签名实验

一.　实验目的：深入理解公钥加密算法 RSA 的加密和解密的原理。

二.　实验环境：

软件：SEED Labs 安全实验平台

三.　实验内容：RSA 私钥的产生；RSA 的加密和解密；数字签名的产生和验证。

四.　实验步骤

1.推导 RSA 私钥

已知：p = F7E75FDC469067FFDC4E847C51F452DF

q = E85CED54AF57E53E092113E62F436F4F

e = 0D88C3

编写 C 程序，思路如下：

将 3 个 16 进制数都转换为 BIGNUM 类型。

计算$n=p\times q$，和欧拉函数值$\varphi(n)=(p-1)(q-1)$。

选取公钥 e，$1\leqslant e\leqslant\varphi(n),(\varphi(n),e)=1$（互素），在模$\varphi(n)$下，e 存在逆元：$de\ mod\ \varphi(n)=1\Rightarrow d=e^{-1}\ mod\ \varphi(n)$，即可计算出私钥 d。

运行结果：

```
[02/02/23]seed@VM:~/exp3$ gcc 1.c -o 1.o -lcrypto
[02/02/23]seed@VM:~/exp3$ ./1.o
public key e= 0D88C3
public varible n= E103ABD94892E3E74AFD724BF28E78366D9676BCCC70118BD0AA1968DBB143D1
private key d= 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB
```

则私钥 d 的 16 进制为：

3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB

2.用 RSA 加密一个消息

n = DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5

e = 010001 (this hex value equals to decimal 65537)

M = A top secret!

并使用密钥验证：d = 74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D

加密变换：$m \rightarrow c = m^e \ mod \ n$

(1)使用 python 命令将字符串转换为十六进制字符串：$python3 -c 'print("A top secret!".encode(encoding="ascii").hex())'，先将字符串转换为 ascii 编码，然后转换为 16 进制。

运行结果：4120746f702073656372657421

编写 C 程序，思路如下：

①将 n、e 转换为 BIGNUM 类型，将 M 字符串先转换为 ASCII 编码，再转换为 BIGNUM 类型。

②计算$m^e \ mod \ n$，结果即为加密后的密文。

③使用密钥 d 进行解密验证。

运行结果：

```
[02/03/23]seed@VM:~/exp3$ ./2.o
cyphertxt: 6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC
plaintxt: 4120746F702073656372657421
```

将密文解密后得到的明文是 M 对应的 16 进制，所以加密正确，密文结果为：6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC

3.解密这个消息

本任务使用的公私钥对与任务 2 中的相同：

n = DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5

e = 010001 (this hex value equals to decimal 65537)

d = 74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26A

A381CD7D30D

请解密下面的密文 C，然后转换回 ASCII 字符串：C = 8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F。

解密变换：$m = c^d \ mod \ n$

编写 C 程序，思路如下：

(1)将 n、d、C 都转换为 BIGNUM 类型。

(2)计算$C^d \ mod \ n$，结果即为解密后的明文。

运行结果：

```
[02/03/23]seed@VM:~/exp3$ gcc 3.c -o 3.o -lcrypto
[02/03/23]seed@VM:~/exp3$ ./3.o
plaintxt: 50617373776F726420697320646565 73
```

则解密结果为：50617373776F726420697320646565 73

然后再使用命令将 16 进制结果转换为 ASCII 码：

$ python3 -c 'print(bytes.fromhex("50617373776F726420697320646565 73").decode())'

运行结果：

```
[02/03/23]seed@VM:~/exp3$ python3 -c 'print(bytes.fromhex("50617373776F726420697320646565 73").
decode())'
Password is dees
```

则密文解密后对应的字符串为：Password is dees

4.产生一个消息的数字签名

本任务使用的公私钥对与任务 2 中的相同：

n = DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5

e = 010001 (this hex value equals to decimal 65537)

d = 74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D

直接为消息生成一个签名：M = I owe you $2000. 并稍微改动 M，将 2000 替换为 3000，然后为修改后的消息签名，比较两个签名。

数字签名原理 → Alice 将消息 m 用私钥加密后，将 m 和签名都发送给 Bob，Bob 使用 Alice 的公钥解密验证签名的有效性。

数学原理：$s = m^d \ mod \ n$

(1)将消息字符串转换为十六进制：

```
[02/03/23]seed@VM:~/exp3$ python3 -c 'print("M = I owe you $2000.".encode(encoding="ascii").hex())
4d203d2049206f776520796f752024323030302e
[02/03/23]seed@VM:~/exp3$ python3 -c 'print("M = I owe you $3000.".encode(encoding="ascii").hex())
4d203d2049206f776520796f752024333030302e
```

(2)编写 C 程序，思路如下：

①将 n、d、M 都转换为 BIGNUM 类型。

②计算$m^d \ mod \ n$，结果即为数字签名。

运行结果：

```
[02/03/23]seed@VM:~/exp3$ ./4.o
signature1: D866CFC4C4E4B73E745EFC99967D7D962261ABC535D03433E07065419C58DC1C
signature2: ADA7BF3628FE1D8D6A0F0B9F436EDFD82AB44306AC243862A25068DBB2217E79
```

可见信息就算只做了微小的变动，签名结果也会发生很大变化。

5.验证这个数字签名是否正确

(1)Bob 从 Alice 那里收到一条消息附带了她的签名 S 的消息 M = "Launch a missile."。我们知道 Alice 的公钥是 (e, n)。请验证这个签名是否确实是 Alice 生成的。公钥和签名（十六进制）如下所示：

M = Launch a missile.

S = 643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F

e = 010001 (this hex value equals to decimal 65537)

n = AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115

验证算法：$m = s^e \ mod \ n$

首先将消息 M 转换为 16 进制形式：

```
[02/03/23]seed@VM:~/exp3$ python3 -c 'print("Launch a missile.".encode(encoding="ascii").hex
4c61756e63682061206d697373696c652e
```

假设上面的签名被破坏了，例如签名的最后一个字节从 2F 变成了 3F，也就是说只改变了一个 bit。请重复这个实验，描述验证过程中发生了什么。编写 C 程序验证签名解密后的明文是否与消息 M 一致，其中 signature1 为 643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F，

signature2 为 643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F。

运行结果：

```
[02/03/23]seed@VM:~/exp3$ ./5.o
m1: 4C61756E63682061206D697373696C652E
m2: 91471927C80DF1E42C154FB4638CE8BC726D3D66C83A4EB6B7BE0203B41AC294
The signature1 is valid.
The signature2 is invalid!
```

输出两个数字签名经过解密得到的明文，signature1 解密得到的明文与 M 转换为十六进制再转换为 BIGNUM 类型的值相等。原来的签名 S 是 Alice 生成的。当签名被破坏，仅改变 1bit 都会使验证失败。

6.验证 X.509 数字证书

在此任务中，我们将使用程序手动验证 X.509 证书。 X.509 包含有关公钥的数据以及颁发者对该数据的签名。我们将从 Web 服务器上下载真实的 X.509 证书，获取其颁发者的公钥，然后使用该公钥来验证证书上的签名。

从真实的 Web 服务器上下载一张证书，下载新浪服务器的数字证书，命令如下：

$ openssl s_client -connect www.sina.com.cn:443 -showcerts

命令的结果包含两张证书。证书的主体（s: 开头的一项）是 sina.com，也就是说这张证书是 sina.com 的证书。颁发者（i: 开头）提供了颁发者的信息。第二张证书的主体与第一张证书的颁发者相同，也就是说第二张证书是一个中间 CA 的。在本任务中，我们使用 CA 的证书验证服务器的证书。

```
---
Certificate chain
 0 s:C = CN, ST = Beijing, O = "Sina.com Technology(China)Co.,ltd", CN = sina.com
   i:C = US, O = DigiCert Inc, OU = www.digicert.com, CN = GeoTrust CN RSA CA G1
-----BEGIN CERTIFICATE-----
MIIOETCCDPmgAwIBAgIQDBLeAwUOy+uCKVDzskuAOTANBgkqhkiG9w0BAQsFADBf
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3
d3cuZGlnaWNlcnQuY29tMR4wHAYDVQQDExVHZW9UcnVzdCBDTiBSU0EgQ0EgRzEw
HhcNMjIxMTEwMDAwMDAwWhcNMjMxMjExMjM1OTU5WjBeMQswCQYDVQQGEwJDTjEQ
MA4GA1UECBMHQmVpamluZzEqMCgGA1UEChMhU2luYS5jb20gVGVjaG5vbG9neShD
aGluYSlDby4sbHRkMREwDwYDVQQDEwhzaW5hLmNvbTCCASIwDQYJKoZIhvcNAQEB
mHpcEcY=
-----END CERTIFICATE-----
 1 s:C = US, O = DigiCert Inc, OU = www.digicert.com, CN = GeoTrust CN RSA CA G1
   i:C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert Global Root CA
-----BEGIN CERTIFICATE-----
MIIFGjCCBAKgAwIBAgIQCgRw0Ja8ihLIkKbfgm7sSzANBgkqhkiG9w0BAQsFADBh
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3
```

复制每张证书

在 "Begin CERTIFICATE" 和 "END CERTIFICATE" 的两行中间的文本，包括这两行）到一个文件中。第一个文件称为 c0.pem ，第二个称为 c1.pem，使用 touch 命令创建两个文件：

$ touch c0.pem

$ touch c1.pem

然后编辑两个文件，将两个证书的内容分别复制到两个文件中。

从证书中提取公钥 e 和公开数 n

从颁发者的证书中 (e, n) 提取公钥。Openssl 提供了从 X.509 整数中提取特定属性的命令。我们可以使用 -modulus 参数来提取 n 的值。没有特定的指令可以用来提取 e，但是我们可以输出所有的域，很容易就能找到 e 的值。

①提取公开数 n：$ openssl x509 -in c1.pem -noout -modulus

运行结果：Modulus=B149FA3D4A799546E23CE04286F6DE543C3C950D858DF5B9F66286E53185873A1D25382F0D1FC5F038DDAF43A4997BE0CDC4E85D5944139F27E7569DA8B2603D0FC51210778C098ACB62E1469DBF3EB521863FA10FC497193F5FB1850EAB98BB1092C81747B5354C5C2C454AF4360BFEE35991437C618A28DA104A2272C037BC8A21DB50E6A42CC99798E4D9C9626715D47F4C7E5835388C2823543C7025786E088A01B10F22FF81BC2B7433626C30389543610F4C4DBCF8D0F0134AAA6E47583BE2AD4B87742FB88698B41893B7E0EDE2891575E989964EE54535BA142C3674F8F42D72D2676ADA6E64A3C6C8A5FA8C2A4FBF3CCFC3F12134396911DAD81D93

②输出所有的域，找到 e 的值：$ openssl x509 -in c1.pem -text -noout

运行结果：

```
Exponent: 65537 (0x10001)
```

则公钥 e 的值为 0x10001。

(4)从服务器的证书中提取签名

没有 openssl 命令可以用来提取签名。输出所有的域，然后复制签名块到一个文件中（注意：如果证书使用的签名算法不是基于 RSA 的，可以找一张其他的证书）：$ openssl x509 -in c0.pem -text -noout

```
Signature Algorithm: sha256WithRSAEncryption
     16:4c:f7:54:86:c9:3f:31:d9:1f:6e:94:17:c2:16:04:3f:f7:
     9a:4b:a6:ea:ab:61:3e:0c:87:89:7d:f8:43:aa:ac:4e:68:85:
     3d:30:66:0d:5f:82:22:d1:70:d4:62:bb:26:a3:f4:eb:78:12:
     19:77:e6:be:b6:79:93:40:7c:66:89:b1:47:4a:04:5c:cd:b6:
     05:72:4d:17:bf:18:a3:78:06:04:76:07:df:bc:7c:a6:95:15:
     a1:51:9b:1c:1c:1e:e0:97:2f:fe:db:d4:c0:8f:d8:90:b8:75:
     c4:8d:91:46:73:fe:b3:93:e3:4d:39:13:7d:7d:26:9c:e1:69:
     ed:e3:5c:12:25:da:3f:c6:e9:80:13:5c:05:96:2e:65:3b:ff:
     d9:49:0b:f4:3e:5e:b6:45:4d:ae:dc:d0:ca:52:b1:eb:79:eb:
     25:25:d9:ca:2f:8a:c4:32:d5:8e:39:e5:35:d3:5f:64:5f:b1:
     1b:82:96:96:fe:9f:fa:cb:06:3d:a1:47:1d:9a:8b:dd:ce:49:
     76:a5:2c:c0:d4:2b:62:9e:d6:6c:fd:63:2b:f7:0a:61:0b:78:
     7f:c6:7e:55:b8:d8:36:f4:08:1c:cd:8d:1d:7f:94:56:81:8d:
     20:74:18:42:cc:e9:f1:0f:26:e7:4f:aa:ed:f3:ff:2a:23:98:
     7a:5c:11:c6
```

签名如上图所示，复制签名到文件 signature 中。然后需要从数据中移除空格和冒号，这样就能获得可以输入程序的十六进制字符串。下面的命令可以实现这个目的。 tr 命令是一个 Linux 用来处理字符串的工具。在本例中， -d 选项用来从数据中删除 ":" 和 "space”。

$ cat signature | tr -d '[:space:]:'

运行结果，则签名（16 进制）为如下内容：

164cf75486c93f31d91f6e9417c216043ff79a4ba6eaab613e0c87897df843aaac4e68853d30660d5f8222d170d462bb26a3f4eb78121977e6beb67993407c6689b1474a045ccdb605724d17bf18a37806047607dfbc7ca69515a1519b1c1c1ee0972ffedbd4c08fd890b875c48d914673feb393e34d39137d7d269ce169ede35c1225da3fc6e980135c05962e653bffd9490bf43e5eb6454daedcd0ca52b1eb79eb2525d9ca2f8ac432d58e39e535d35f645fb11b829696fe9ffacb063da1471d9a8bddce4976a52cc0d4

2b629ed66cfd632bf70a610b787fc67e55b8d836f4081ccd8d1d7f9456818d20741842cce9f10f26e74faaedf3ff2a23987a5c11c6

(5)提取服务器的证书主体

一个证书颁发机构（CA）为一张服务器证书生成签名，首先需要计算证书的 Hash ，然后对 Hash 签名。为了验证证书，我们也需要从证书中生成一个 Hash。由于 Hash 是在计算签名之前生成的，我们需要排除证书的签名块，然后再计算 Hash。

X.509 证书使用 ASN.1 (Abstract Syntax Notation One) 标准编码，所以如果我们能解析 ASN.1 结构，我们就能很容易从整数中提取任何一个域。Openssl 有一个用于解析 ASN.1 结构的 asn1parse 命令，这里我们用来解析 X.509 证书：$ openssl asn1parse -i -in c0.pem

```
d@VM:~/exp3$  openssl asn1parse -i -in c0.pem
    0:d=0  hl=4 l=3601 cons: SEQUENCE
    4:d=1  hl=4 l=3321 cons:  SEQUENCE   ①
    8:d=2  hl=2 l=   3 cons:   cont [ 0 ]
   10:d=3  hl=2 l=   1 prim:    INTEGER           :02
   13:d=2  hl=2 l=  16 prim:   INTEGER           :0C12DE03050ECBEB822950F3B24B8039
   31:d=2  hl=2 l=  13 cons:   SEQUENCE
   33:d=3  hl=2 l=   9 prim:    OBJECT            :sha256WithRSAEncryption
   44:d=3  hl=2 l=   0 prim:    NULL
   46:d=2  hl=2 l=  95 cons:   SEQUENCE
   48:d=3  hl=2 l=  11 cons:    SET
   50:d=4  hl=2 l=   9 cons:     SEQUENCE
   52:d=5  hl=2 l=   3 prim:      OBJECT            :countryName
   57:d=5  hl=2 l=   2 prim:      PRINTABLESTRING   :US
   61:d=3  hl=2 l=  21 cons:    SET
   63:d=4  hl=2 l=  19 cons:     SEQUENCE
   65:d=5  hl=2 l=   3 prim:      OBJECT            :organizationName
   70:d=5  hl=2 l=  12 prim:      PRINTABLESTRING   :DigiCert Inc
   84:d=3  hl=2 l=  25 cons:    SET
   86:d=4  hl=2 l=  23 cons:     SEQUENCE
   88:d=5  hl=2 l=   3 prim:      OBJECT            :organizationalUnitName
   93:d=5  hl=2 l=  16 prim:      PRINTABLESTRING   :www.digicert.com
 2933:d=4  hl=2 l=   9 cons:     SEQUENCE
 2935:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 Basic Constraints
 2940:d=5  hl=2 l=   2 prim:      OCTET STRING      [HEX DUMP]:3000
 2944:d=4  hl=4 l= 381 cons:     SEQUENCE
 2948:d=5  hl=2 l=  10 prim:      OBJECT            :CT Precertificate SCTs
 2960:d=5  hl=4 l= 365 prim:      OCTET STRING      [HEX DUMP]:04820169016700750 0E83ED0DA3EF5063532E75728
BC896BC903D3CBD1116BECEB69E1777D6D06BD6E0000018462B7403C000004030046304402203 8AF05CA63E4416E566FD3F63E208
B5B1D272BD10FA5C6730AC78C6E109AA4A902203FC9D4AA27611872A1A00965DB02EEF0A7F84 036B8FA162C80ABF226D71F2CE300
7600B3737707E18450F86386D605A9DC11094A792DB1670C0B87DCF0030E7936A59A0000018 462B7408B0000040300473045022100
0A2AADB62415E49277AF59951510650AC5F8CD7E70F2750FB869C5A6F6CD7354202204BCBB5 FAB91BB2C92AFE6919EB3CA1CBED32
E88655BCA98B34B8E9F3C8077FD5007600B73EFB24DF9C4DBA75F239C5BA58F46C5DFC42CF7 A9F35C49E1D098125EDB4990000018
462B74060000004030047304502210 0E47E82408DFF8A513BFDEE3976BE1F6305C8EB7BBA8520159DD67721857D4D3C022063928D
2BBD2AEE2E17EA272B3ADF9B0133C9687612DCE903F8F2A921D7CF9E74
 3329:d=1  hl=2 l=  13 cons:  SEQUENCE  ②
 3331:d=2  hl=2 l=   9 prim:   OBJECT            :sha256WithRSAEncryption
 3342:d=2  hl=2 l=   0 prim:   NULL
 3344:d=1  hl=4 l= 257 prim:  BIT STRING
```

从①开始的字段用来生成哈希的证书主体；从②开始的字段为签名块。它们的偏移量是每行开头的数字。在例子中，证书主体是从偏移量 4 到 3328，而签名块是从 3329 到文件末尾。对于 X.509 证书，起始偏移量总是相同的(即 4)，但结束偏移量取决于证书的内容长度。我们可以使用`-strparse`选项从偏移量 4 获取字段，这将为我们提供证书 1 的主体，不包括签名块：$ openssl asn1parse -i -in c0.pem -strparse 4 -out c0_body.bin -noout

(6)计算证书的 Hash 值：$ sha256sum c0_body.bin

运行结果：

```
[02/03/23]seed@VM:~/exp3$ sha256sum  c0_body.bin
da4fc748834348d6e8b63805daf55210f9dc57b6ae725de768c0842b335209cc  c0_body.bin
```

则证书的 Hash 值为：da4fc748834348d6e8b63805daf55210f9dc57b6ae725de768c0842b335209cc

(7)验证签名

现在我们有了全部的信息，包括 CA 的公钥 e、 CA 的签名 s 以及服务器证书的主体的哈希值。可以执行自己的程序来验证签名是否有效。 Openssl 提供了一个命令来为我们验证证书，但是这里要求使用自己的程序实现。

编写 C 程序，验证算法结果，解密数字签名：$s^e \ mod \ n \ = \ m?h$ (私钥加密，公钥解密)

运行结果：

```
[02/03/23]seed@VM:~/exp3$ gcc 6.c -o 6.o -lcrypto
[02/03/23]seed@VM:~/exp3$ ./6.o
message: 01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00303
1300D060960864801650304020105000420DA4FC748834348D6E8B63805DAF55210F9DC57B6AE725DE768C0842B335209CC
```

根据 sha256WithRSAEncryption 可知，发放证书用到的 hash 算法是 sha256，即得到的 hash 值共有 256 位，256/4=64。再根据于 X.509 证书签名验证的流程可知，只需要将上述结果的后 64 个十六进制数与第 6 步得到的 hash 值 h = da4fc748834348d6e8b63805daf55210f9dc57b6ae725de768c0842b335209cc 做比较即可。发现二者相等，则说明证书 1 上的签名是证书 2 的主体 CA 签发的。

附相应的源程序清单（加注释）

任务 1

```
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 128

void printBN(char *msg, BIGNUM *a)    // print the big number
{ /* Use BN_bn2hex(a) for hex string
    * Use BN_bn2dec(a) for decimal string */
    char *number_str = BN_bn2hex(a);
    printf("%s %s\n", msg, number_str);
    OPENSSL_free(number_str);
}

int main()
{
```

```
    BN_CTX *ctx = BN_CTX_new();  // temporary variables
    BIGNUM *p = BN_new();
    BIGNUM *q = BN_new();
    BIGNUM *fai_n = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *p_1 = BN_new();
    BIGNUM *q_1 = BN_new();

    BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
    BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
    BN_hex2bn(&e, "0D88C3");

    BN_sub(p_1, p, BN_value_one());
    BN_sub(q_1, q, BN_value_one());
    BN_mul(n, p, q, ctx);
    BN_mul(fai_n, p_1, q_1, ctx);

    BN_mod_inverse(d, e, fai_n, ctx);   // modulo inverse

    // hex number
    printBN("public key e=", e);
    printf("\n");
    printBN("public varible n=", n);
    printf("\n");
    printBN("private key d=", d);
    return 0;
}
```

任务 2

```
# include <stdio.h>
# include <openssl/bn.h>
# define NBITS 128

void printBN(char* msg, BIGNUM *a)
{
    char* number_str = BN_bn2hex(a);
    printf("%s %s", msg, number_str);
    OPENSSL_free(number_str);
}

int main()
```

```
{
    BN_CTX* ctx = BN_CTX_new();
    BIGNUM* n = BN_new();   // public number
    BIGNUM* e = BN_new();   // public key
    BIGNUM* m = BN_new();   // message
    BIGNUM* c = BN_new();   // cypher text
    BIGNUM* d = BN_new();   // private key
    BIGNUM* p = BN_new();   // decrpyt text

    BN_hex2bn(&n,
"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    BN_hex2bn(&e, "010001");
    BN_hex2bn(&m, "4120746f702073656372657421");
    BN_hex2bn(&d,
"74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

    // c=m^e mod n
    BN_mod_exp(c, m, e, n, ctx);
    // p=c^d mod n
    BN_mod_exp(p, c, d, n, ctx);
    printBN("cyphertxt:", c);
    printf("\n");
    printBN("plaintxt:", p);
    printf("\n");
    return 0;
}
```

任务 3

```
# include <stdio.h>
# include <openssl/bn.h>
# define NBITS 128

void printBN(char* msg, BIGNUM *a)
{
    char* number_str = BN_bn2hex(a);
    printf("%s %s", msg, number_str);
    OPENSSL_free(number_str);
}

int main()
{
    BN_CTX* ctx = BN_CTX_new();
    BIGNUM* n = BN_new();   // public number
```

```
    BIGNUM* m = BN_new();   // message
    BIGNUM* c = BN_new();   // cypher text
    BIGNUM* d = BN_new();   // private key

    BN_hex2bn(&n,
"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    BN_hex2bn(&c,
"8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");
    BN_hex2bn(&d,
"74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

    // m=c^d mod n
    BN_mod_exp(m, c, d, n, ctx);
    printBN("plaintxt:", m);
    printf("\n");
    return 0;
}
```

任务 4

```
# include <stdio.h>
# include <openssl/bn.h>
# define NBITS 128

void printBN(char* msg, BIGNUM *a)
{
    char* number_str = BN_bn2hex(a);
    printf("%s %s", msg, number_str);
    OPENSSL_free(number_str);
}

int main()
{
    BN_CTX* ctx = BN_CTX_new();
    BIGNUM* n = BN_new();   // public number
    BIGNUM* m1 = BN_new();   // message1
    BIGNUM* m2 = BN_new();   // message2
    BIGNUM* s1 = BN_new();   // signature1
    BIGNUM* s2 = BN_new();   // signature2
    BIGNUM* d = BN_new();   // private key

    BN_hex2bn(&n,
"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    BN_hex2bn(&m1, "4d203d2049206f776520796f752024323030302e");
```

```
    BN_hex2bn(&m2, "4d203d2049206f776520796f752024333030302e");
    BN_hex2bn(&d,
"74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

    // s=m^d mod n
    BN_mod_exp(s1, m1, d, n, ctx);
    BN_mod_exp(s2, m2, d, n, ctx);
    printBN("signature1:", s1);
    printf("\n");
    printBN("signature2:", s2);
    printf("\n");
    return 0;
}
```

任务 5

```
# include <stdio.h>
# include <openssl/bn.h>
# define NBITS 128

void printBN(char* msg, BIGNUM *a)
{
    char* number_str = BN_bn2hex(a);
    printf("%s %s", msg, number_str);
    OPENSSL_free(number_str);
}

int main()
{

    BN_CTX* ctx = BN_CTX_new();
    BIGNUM* n = BN_new();   // public number
    BIGNUM* m = BN_new();   // true message
    BIGNUM* m1 = BN_new();   // message1
    BIGNUM* m2 = BN_new();   // message2
    BIGNUM* s1 = BN_new();   // signature1
    BIGNUM* s2 = BN_new();   // signature2
    BIGNUM* e = BN_new();   // public key

    BN_hex2bn(&n,
"AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");
    BN_hex2bn(&s1,
"643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");
    BN_hex2bn(&s2,
```

```
"643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F");
    BN_hex2bn(&m, "4c61756e63682061206d697373696c652e");
    BN_hex2bn(&e, "010001");

    // m=s^e mod n
    BN_mod_exp(m1, s1, e, n, ctx);
    BN_mod_exp(m2, s2, e, n, ctx);
    printBN("m1:", m1);
    printf("\n");
    printBN("m2:", m2);
    printf("\n");
    // compare the 3 big numbers
    if (BN_cmp(m, m1)==0)
        printf("The signature1 is valid.");
    else
        printf("The signature1 is invalid!");
    printf("\n");
    if (BN_cmp(m, m2)==0)
        printf("The signature2 is valid.");
    else
        printf("The signature2 is invalid!");
    printf("\n");
    return 0;
}
```

任务 6

```
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 128

void printBN(char *msg, BIGNUM *a)
{ /* Use BN_bn2hex(a) for hex string
    * Use BN_bn2dec(a) for decimal string */
    char *number_str = BN_bn2hex(a);
    printf("%s %s\n", msg, number_str);
    OPENSSL_free(number_str);
}

int main()
{
    BN_CTX *ctx = BN_CTX_new();

    BIGNUM *n = BN_new();
```

```
    BIGNUM *e = BN_new();
    BIGNUM *s = BN_new();
    BIGNUM *m = BN_new();

    BN_hex2bn(&n,
"B149FA3D4A799546E23CE04286F6DE543C3C950D858DF5B9F66286E53185873A1D2538
2F0D1FC5F038DDAF43A4997BE0CDC4E85D5944139F27E7569DA8B2603D0FC51210778C0
98ACB62E1469DBF3EB521863FA10FC497193F5FB1850EAB98BB1092C81747B5354C5C2C
454AF4360BFEE35991437C618A28DA104A2272C037BC8A21DB50E6A42CC99798E4D9C96
26715D47F4C7E5835388C2823543C7025786E088A01B10F22FF81BC2B7433626C303895
43610F4C4DBCF8D0F0134AAA6E47583BE2AD4B87742FB88698B41893B7E0EDE2891575E
989964EE54535BA142C3674F8F42D72D2676ADA6E64A3C6C8A5FA8C2A4FBF3CCFC3F121
34396911DAD81D93");
    BN_hex2bn(&e, "010001");
    BN_hex2bn(&s,
"164cf75486c93f31d91f6e9417c216043ff79a4ba6eaab613e0c87897df843aaac4e68
853d30660d5f8222d170d462bb26a3f4eb78121977e6beb67993407c6689b1474a045cc
db605724d17bf18a37806047607dfbc7ca69515a1519b1c1c1ee0972ffedbd4c08fd890
b875c48d914673feb393e34d39137d7d269ce169ede35c1225da3fc6e980135c05962e6
53bffd9490bf43e5eb6454daedcd0ca52b1eb79eb2525d9ca2f8ac432d58e39e535d35f
645fb11b829696fe9ffacb063da1471d9a8bddce4976a52cc0d42b629ed66cfd632bf70
a610b787fc67e55b8d836f4081ccd8d1d7f9456818d20741842cce9f10f26e74faaedf3
ff2a23987a5c11c6");

    // m = s ^ e mod n
    BN_mod_exp(m, s, e, n, ctx);

    printBN("message:", m);

    return 0;
}
```