CPS633 Section 07 Fall2021

Lab 07 Report

RSA Public-Key Encryption and Signature Lab

Name: Tusaif Azmat (group leader)
Student#: 500660278.
And
Name: Ankit Sodhi
Student#: 500958004

Group 04.

CPS633 Fall2021 Section 07   Group 04   Name: Tusaif Azmat - 500660278
Name: Ankit Sodhi - 500958004

CPS 633 - Lab 7 Report

RSA Public-Key Encryption and Signature Lab

Lab Tasks:

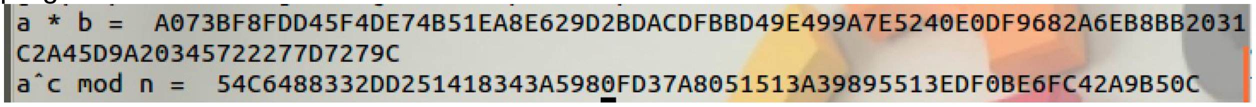
The goal of this lab is to gain hands-on experiences with the RSA algorithm. The RSA algorithm involves the computations on large numbers. Since our operators can only operate on primitive data types, such as 32-bit integers or 64-bit long integers, we must use a library that can perform arithmetic operations on integers of arbitrary size. In this lab, I used the Big Number library provided by openssl. I installed this by doing:

```
$ sudo apt-get update
$ sudo apt-get install libssl-dev
```

We compiled the program, bn_sample.c, given in the project description. We ran

```
$ gcc bn_sample.c -lcrypto
```

to compile the program using the crypto library. The following was the result from running the program.



```
a * b =  A073BF8FDD45F4DE74B51EA8E629D2BDACDFBBD49E499A7E5240E0DF9682A6EB8BB2031
C2A45D9A20345722277D7279C
a^c mod n =  54C6488332DD251418343A5980FD37A8051513A39895513EDF0BE6FC42A9B50C
```

3.1 Task 1: Deriving the Private Key

Compile Deriving-the-Private-Key.c:

```
$ gcc -o Deriving-the-Private-Key Deriving-the-Private-Key.c -lcrypto
$ ./Deriving-the-Private-Key
```

Note: Link order matters. Place `-lcrypto` flag at last.

```
public key
(0D88C3,E103ABD94892E3E74AFD724BF28E78366D9676BCCC70118BD0AA1968DBB143D1)
private key
(3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB,E103ABD9489
2E3E74AFD724BF28E78366D9676BCCC70118BD0AA1968DBB143D1)
```

The value of d (Private Key) is:

```
3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB
```

3.2 Task 2: Encrypting a Message

First, convert the string to a hex string:

```
$ python -c 'print("A top secret!".encode("hex"))'
4120746f702073656372657421
```

Compile encrypt_m.c and run:

```
$ gcc -o encrypt_m encrypt_m.c -lcrypto
$ ./encrypt_m
Encryption result: 6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC
```

3.3 Task 3: Decrypting a Message

Compile decrypt_c.c and run:

```
$ gcc -o decrypt_c decrypt_c.c -lcrypto
$ ./decrypt_c
Decryption result: 50617373776F72642069732064656573
```

decode the hex string into a plain ASCII string:

```
$ python -c 'print("50617373776F72642069732064656573".decode("hex"))'
Password is dees
```

3.4 Task 4: Signing a Message

We utilize RSA encryption as signature:

First, get the hex strings of 2 strings:

```
$ python -c 'print("I owe you $2000".encode("hex"))'
49206f776520796f75202432303030
$ python -c 'print("I owe you $3000".encode("hex"))'
49206f776520796f75202433303030
```

Adapte some subtle modification based on encrypt_m.c and write encrypt_diff.c. Compile and then run:

```
$ gcc -o encrypt_diff encrypt_diff.c -lcrypto
$ ./encrypt_diff
Signature of M1: 80A55421D72345AC199836F60D51DC9594E2BDB4AE20C804823FB71660DE7B82
Signature of M2: 04FC9C53ED7BBE4ED4BE2C24B0BDF7184B96290B4ED4E3959F58E94B1ECEA2EB
```

3.5 Task 5: Verifying a Signature

Get the hex string from M:

```
$ python -c 'print("Launch a missile.".encode("hex"))'
4c61756e63682061206d697373696c652e
```

Compile and run verify_sig.c:

```
$ gcc -o verify_sig verify_sig.c -lcrypto
$ ./verify_sig
Valid Signature!
```

It's indeed Alice's message.

If **a** valid ***S*** happens to be corrupted, even just in one byte, our program will also reject the message.

3.6 Task 6: Manually Verifying an X.509 Certificate

Step 1: Download a certificate from a real web server

```
$ openssl s_client -connect seedsecuritylabs.org:443 -showcerts > seed.txt
depth=2 C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert High Assurance EV Root CA
verify return:1
depth=1 C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert SHA2 High Assurance Server CA
verify return:1
depth=0 C = US, ST = California, L = San Francisco, O = "GitHub, Inc.", CN = www.github.com
verify return:1
```

The certificates are as below:

Save the first certification (server's CA) as c0.pem:

```
-----BEGIN CERTIFICATE-----
MIIHMDCCBhigAwIBAgIQAkk+B/qeN1otu8YdlEMPzzANBgkqhkiG9w0BAQsFADBw
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3
d3cuZGlnaWNlcnQuY29tMS8wLQYDVQQDEyZEaWdpQ2VydCBTSEEyIEhpZ2ggQXNz
dXJhbmNlIFNlcnZlciBDQTAeFw0yMDA1MDYwMDAwMDBaFw0yMjA0MTQxMjAwMDBa
MGoxCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpDYWxpZm9ybmlhMRYwFAYDVQQHEw1T
YW4gRnJhbmNpc2NvMRUwEwYDVQQKEwxHaXRIdWIsIEluYy4xFzAVBgNVBAMTDnd3
dy5naXRodWIuY29tMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAsj49
6jJ99veEXO7WdxGQZ7idtCnDcjZqQeDiy6057SwXj9yDUVnqhwo/yII8+y6Jpk3g
75LpPpYNjiOwYp/JkpWbpBAd1FWlvXJo/eZS+TwuIYb7JSc2H3NDDKt2VV5SSKQd
XOkDNqq7BisOFp2/TYwCMZboLufwRR5fKxL0nTKIOCwpnH8k//UdWpvTgIixDGLY
QCwHt0fYEo49jFeDaKD4WMBPq6Tx1iKWBhw3HVc/OyvI3yjRAx4Anf/DCSt9YTW6
f/ND4O/fOowcfW5T7zii1Kw0yw+ulBrE/xe6taVhL+QR0MXNkQV2iHNN85swidwM
tcdGI8g3fYL48bSRywIDAQABo4IDyjCCA8YwHwYDVR0jBBgwFoAUUWj/kK8CB3U8
zNllZGKiErhZcjswHQYDVR0OBBYEFIygCmlH3IkysE3GEUViXxovlk46MHsGA1Ud
EQR0MHKCDnd3dy5naXRodWIuY29tggwqLmdpdGh1Yi5jb22CCmdpdGh1Yi5jb22C
CyouZ2l0aHViLmlvgglnaXRodWIuaW+CFyouZ2l0aHVidXNlcmNvbnRlbnQuY29t
ghVnaXRodWJ1c2VyY29udGVudC5jb20wDgYDVR0PAQH/BAQDAgWgMB0GA1UdJQQW
MBQGCCsGAQUFBwMBBggrBgEFBQcDAjB1BgNVHR8EbjBsMDSgMqAwhi5odHRwOi8v
Y3JsMy5kaWdpY2VydC5jb20vc2hhMi1oYS1zZXJ2ZXItZzYuY3JsMDSgMqAwhi5o
dHRwOi8vY3JsNC5kaWdpY2VydC5jb20vc2hhMi1oYS1zZXJ2ZXItZzYuY3JsMEwG
A1UdIARFMEMwNwYJYIZIAYb9bAEBMCowKAYIKwYBBQUHAgEWHGh0dHBzOi8vd3d3
LmRpZ2ljZXJ0LmNvbS9DUFMwCAYGZ4EMAQICMIGDBggrBgEFBQcBAQR3MHUwJAYI
KwYBBQUHMAGGGGh0dHA6Ly9vY3NwLmRpZ2ljZXJ0LmNvbTBNBggrBgEFBQcwAoZB
aHR0cDovL2NhY2VydHMuZGlnaWNlcnQuY29tL0RpZ2lDZXJ0U0hBMkhpZ2hBc3N1
cmFuY2VTZXJ2ZXJDQS5jcnQwDAYDVR0TAQH/BAIwADCCAX0GCisGAQQB1nkCBAIE
ggFtBIIBaQFnAHYARqVV63X6kSAwtaKJafTzfREsQXS+/Um4havy/HD+bUcAAAFx
6y8fFgAABAMARzBFAiEA59y6w9oaoAoM2fvFq6KofYWRh0xRm4VEEaMHBtsBYUgC
IBZxJhjA7SGWUlo57YslG8u6clHngDNvoTNVw1HQtTr3AHUAIkVFB1lVJFaWP6Ev
8fdthuAjJmOtwEt/XcaDXG7iDwIAAAFx6y8evwAABAMARjBEAiBmEjiioTbc1//h
CInYIX6O8hph5oLRVGCTxrTBfSRT2wIgZz7x3ZNIKQkWPKOFaaW3AxcB0DzhFsD6
gxhkbl1p0AgAdgBRo7D1/QF5nFZtuDd4jwykeswbJ8v3nohCmg3+1IsF5QAAAXHr
Lx8JAAAEAwBHMEUCIBQ/6El+TCCtWuop7IderN0+byn5sDreTu+Xz3GiY8cLAiEA
7S83HxFFdQhQqpjjbWbIVBA88Nn/riaf5Jb8h3oJV8cwDQYJKoZIhvcNAQELBQAD
ggEBADzu/I/4dMPwG4QzMFHZmgQFlnc/xqXtaNLqONIzXPznBQmHQi481xKgAR4
jZOTTknlwOLBXnDXvV6rJQZXut3pxHSvVJk2kvuyDO3RC0uudd81AXIUsd6Pnjt2
D6Xd/ypUAoMkyE+8euYESEFk4HlnrpXtN7OSTGVYZQk0aJrDINslXdmUL9E6AQiI
YaRIpRMRdj4stG6CkPJpfSauWa19kReZ6hTQR5f89L6x50us7GuWlmH6EmVFIbhf
9EO02QA3CcU7bE1iLWMHmKcU6ythmgsvNRU5TikxvF77JFv7n1/y8GLrprmKpB6Q
Df4PA8S9ROX9Rzgwe3KTIM6qeKU=
-----END CERTIFICATE-----
```

And the second one (immediate CA, issuer's CA) as c1.pem:

```
-----BEGIN CERTIFICATE-----
MIIEsTCCA5mgAwIBAgIQBOHnpNxc8vNtwCtCuF0VnzANBgkqhkiG9w0BAQsFADBs
```

```
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3
d3cuZGlnaWNlcnQuY29tMSswKQYDVQQDEyJEaWdpQ2VydCBIaWdoIEFzc3VyYW5j
ZSBFViBSb290IENBMB4XDTEzMTAyMjEyMDAwMFoXDTI4MTAyMjEyMDAwMFowcDEL
MAkGA1UEBhMCVVMxFTATBgNVBAoTDERpZ2lDZXJ0IEluYzEZMBcGA1UECxMQd3d3
LmRpZ2ljZXJ0LmNvbTEvMC0GA1UEAxMmRGlnaUNlcnQgU0hBMiBIaWdoIEFzc3Vy
YW5jZSBTZXJ2ZXIgQ0EwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQC2
4C/CJAbIbQRf1+8KZAayfSImZRauQkCbztyfn3YHPsMwVYcZuU+UDlqUH1VWtMIC
Kq/QmO4LQNfE0DtyyBSe75CxEamu0si4QzrZCwvV1ZX1QK/IHe1NnF9Xt4ZQaJn1
itrSxwUfqJfJ3KSxgoQtxq2lnMcZgqaFD15EWCo3j/018QsIJzJa9buLnqS9UdAn
4t07QjOjBSjEuyjMmqwrIw14xnvmXnG3Sj4I+4G3FhahnSMSTeXXkgisdaScus0X
sh5ENWV/UyU50RwKmmMbGZJ0aAo3wsJSSMs5WqK24V3B3aAguCGikyZvFEohQcft
bZvySC/zA/WiaJJTL17jAgMBAAGjggFJMIIBRTASBgNVHRMBAf8ECDAGAQH/AgEA
MA4GA1UdDwEB/wQEAwIBhjAdBgNVHSUEFjAUBggrBgEFBQcDAQYIKwYBBQUHAwIw
NAYIKwYBBQUHAQEEKDAmMCQGCCsGAQUFBzABhhhodHRwOi8vb2NzcC5kaWdpY2Vy
dC5jb20wSwYDVR0fBEQwQjBAoD6gPIY6aHR0cDovL2NybDQuZGlnaWNlcnQuY29t
L0RpZ2lDZXJ0SGlnaEFzc3VyYW5jZUVWUm9vdENBLmNybDA9BgNVHSAENjA0MDIG
BFUdIAAwKjAoBggrBgEFBQcCARYcaHR0cHM6Ly93d3cuZGlnaWNlcnQuY29tL0NQ
UzAdBgNVHQ4EFgQUUWj/kK8CB3U8zNllZGKiErhZcjswHwYDVR0jBBgwFoAUsT7D
aQP4v0cB1JgmGggC72NkK8MwDQYJKoZIhvcNAQELBQADggEBABiKlYkD5m3fXPwd
aOpKj4PWUS+Na0QWnqxj9dJubISZi6qBcYRb7TROsLd5kinMLYBq8I4g4Xmk/gNH
E+r1hspZcX30BJZr01lYPf7TMSVcGDiEo+afgv2MW5gxTs14nhr9hctJqvIni5ly
/D6q1UEL2tU2ob8cbkdJf17ZSHwD2f2LSaCYJkJA69aSEaRkCldUxPUd1gJea6zu
xICaEnL6VpPX/78whQYwvwt/Tv9XBZ0k7YXDK/umdaisLRbvfXknsuvCnQsH6qqF
0wGjIChBWUMo0oHjqvbsezt3tkBigAVBRQHvFwY+3sAzm2fTYS5yh+Rp/BIAV0Ae
cPUeybQ=
-----END CERTIFICATE-----
```

Step 2: Extract the public key (e, n) from the issuer's certificate.

Extract the modulus n from an x509 certificate

```
$  openssl x509 -in c1.pem -noout -modulus
Modulus=B6E02FC22406C86D045FD7EF0A6406B27D22266516AE42409BCEDC9F9F76073EC3305
58719B94F940E5A941F5556B4C2022AAFD098EE0B40D7C4D03B72C8149EEF90B111A9AED2C8B8
433AD90B0BD5D595F540AFC81DED4D9C5F57B786506899F58ADAD2C7051FA897C9DCA4B182842
DC6ADA59CC71982A6850F5E44582A378FFD35F10B0827325AF5BB8B9EA4BD51D027E2DD3B4233
A30528C4BB28CC9AAC2B230D78C67BE65E71B74A3E08FB81B71616A19D23124DE5D79208AC75A
49CBACD17B21E4435657F532539D11C0A9A631B199274680A37C2C25248CB395AA2B6E15DC1DD
A020B821A293266F144A2141C7ED6D9BF2482FF303F5A26892532F5EE3
```

Print all attributes of the certificate, and then find the **exponent,** which is public key e:

```
$ openssl x509 -in c1.pem -text -noout | grep Exponent
                Exponent: 65537 (0x10001)
```

Step 3: Extract the signature from the server’s certificate.

Find the location of last "Signature Algorithm", the hex string is the body of the signature.

```
$ openssl x509 -in c0.pem -text -noout
Signature Algorithm: sha256WithRSAEncryption
00:f3:bb:f2:3f:e1:d3:0f:c0:6e:10:cc:c1:47:66:68:10:16:
         59:dc:ff:1a:97:b5:a3:4b:a8:e3:48:cd:73:f3:9c:14:26:1d:
         08:b8:f3:5c:4a:80:04:78:8d:93:93:4e:49:e5:c0:e2:c1:5e:
         70:d7:bd:5e:ab:25:06:57:ba:dd:e9:c4:74:af:54:99:36:92:
         fb:b2:0c:ed:d1:0b:4b:ae:75:df:35:01:72:14:b1:de:8f:9e:
         3b:76:0f:a5:dd:ff:2a:54:02:83:24:c8:4f:bc:7a:e6:04:48:
         41:64:e0:79:67:ae:95:ed:37:b3:92:4c:65:58:65:09:34:68:
         9a:c3:20:db:25:5d:d9:94:2f:d1:3a:01:08:88:61:a4:48:a5:
         13:11:76:3e:2c:b4:6e:82:90:f2:69:7d:26:ae:59:ad:7d:91:
         17:99:ea:14:d0:47:97:fc:f4:be:b1:e7:4b:ac:ec:6b:96:96:
         61:fa:12:65:45:21:b8:5f:f4:43:b4:d9:00:37:09:c5:3b:6c:
         4d:62:2d:63:07:98:a7:14:eb:2b:61:9a:0b:2f:35:15:39:4e:
         29:31:bc:5e:fb:24:5b:fb:9f:5f:f2:f0:62:eb:a6:b9:8a:a4:
         1e:90:0d:fe:0f:03:c4:bd:44:e5:fd:47:38:30:7b:72:93:20:
         ce:aa:78:a5
```

Export the body to signature and then trim all spaces and colons in it:

```
$ cat signature | tr -d '[:space:]:'
00f3bbf23fe1d30fc06e10ccc1476668101659dcff1a97b5a34ba8e348cd73f39c1426
1d08b8f35c4a8004788d93934e49e5c0e2c15e70d7bd5eab250657badde9c474af5499
3692fbb20cedd10b4bae75df35017214b1de8f9e3b760fa5ddff2a54028324c84fbc7a
e604484164e07967ae95ed37b3924c6558650934689ac320db255dd9942fd13a010888
61a448a51311763e2cb46e8290f2697d26ae59ad7d911799ea14d04797fcf4beb1e74b
acec6b969661fa12654521b85ff443b4d9003709c53b6c4d622d630798a714eb2b619a
0b2f3515394e2931bc5efb245bfb9f5ff2f062eba6b98aa41e900dfe0f03c4bd44e5fd
4738307b729320ceaa78a5
```

Step 4: Extract the body of the server's certificate.

Parse the server's certificate

```
$ openssl asn1parse -i -in c0.pem
    0:d=0  hl=4 l=1840 cons: SEQUENCE
    4:d=1  hl=4 l=1560 cons:  SEQUENCE
    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           :02493E07FA9E375A2DBBC61D94430FCF
   31:d=2  hl=2 l=  13 cons:   SEQUENCE
   33:d=3  hl=2 l=   9 prim:    OBJECT            :sha256WithRSAEncryption
   ...
 1568:d=1  hl=2 l=  13 cons:  SEQUENCE
 1570:d=2  hl=2 l=   9 prim:   OBJECT            :sha256WithRSAEncryption
 ...
```

The field starting from

```
4:d=1  hl=4 l=1560 cons:  SEQUENCE
```

is the body of the certificate that is used to generate the hash. And the line before the last "sha256WithRSAEncryption" is the beginning of the signature block.

```
1568:d=1  hl=2 l=  13 cons:  SEQUENCE
```

So, the certificate body starts from offset 4 to 1567. Use -strparse to get the field from the offset 4, which is exactly the body of server's certificate:

```
$ openssl asn1parse -i -in c0.pem -strparse 4 -out c0_body.bin -noout
```

Then calculate its hash value

```
$ sha256sum c0_body.bin
0640f8d13c0789ff0ed5437cf4bc9f2827d52146dddff38aefc2c17747d45f28  c0_body.bin
```

SHA256 with RSA utilizes PKCS#1 v1.5 to pad hash (detailed algorithm available in this answer. Since sizes of both the signature and the n are 256 bytes, we pad the hash into 256 bytes by:

```
$ Python
>>> prefix = "0001"
>>> hash = "0640f8d13c0789ff0ed5437cf4bc9f2827d52146dddff38aefc2c17747d45f28"
>>> A = "30 31 30 0D 06 09 60 86 48 01 65 03 04 02 01 05 00 04 20".replace(' ','')
>>> total_len = 256
>>> pad_len = total_len - 1 - (len(A)+len(prefix)+len(hash))//2
>>> prefix + "FF" * pad_len + "00" + A + hash
'0001FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF0030313
00D0609608648016503040201050004200640f8d13c0789ff0ed5437cf4bc9f2827d52146dddff38
aefc2c17747d45f28'
```

Simply replace the initial values of variables in verify_sig.c with all values we obtained above, we write a new program verify_ca.c. Compile and run:

```
$ gcc -o verify_ca verify_ca.c -lcrypto
$ ./verify_ca
Valid Signature!
```

The same result validated by openssl:

```
$ openssl verify -untrusted c1.pem c0.pem
```