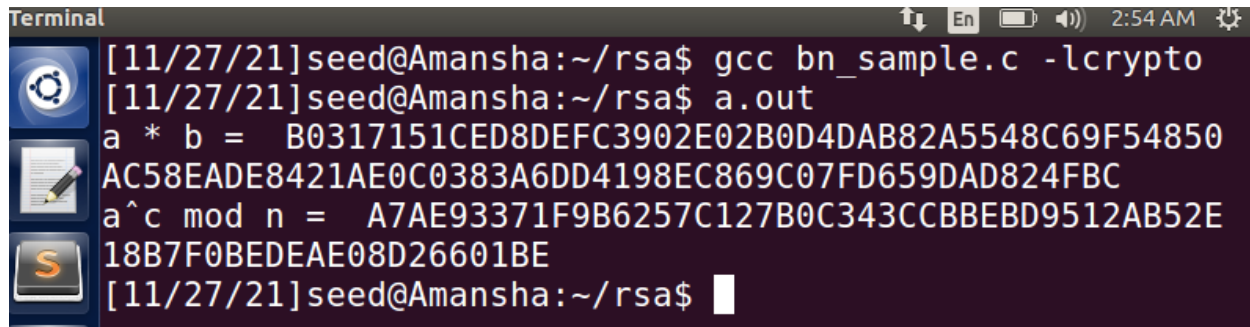


Amir Mansha
CYSE 425
RSA Public Key Encryption and Signature Lab

Pre-Task

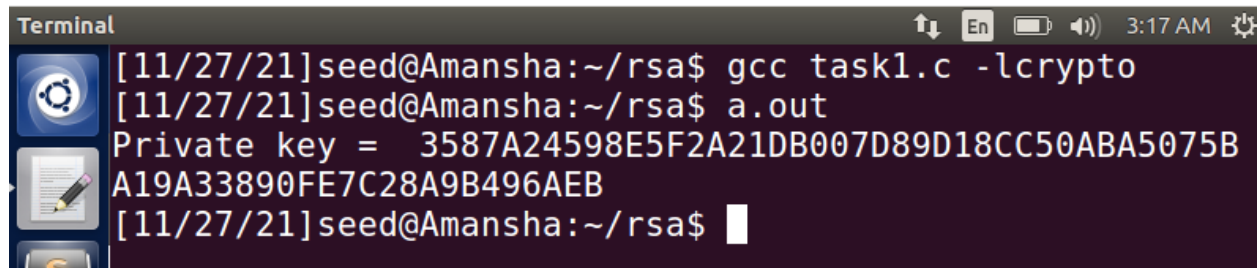
In this task, we use the `bn_sample.c` program that initialize the bignum variables and compute $a*b$ and $(a^b \bmod n)$ by using the gcc compiler and `-lcrypto` that uses crypto library. I used `a.out` to execute the program. We get the computations in the screenshot below.

A terminal window titled "Terminal" with a dark background. The prompt is "[11/27/21]seed@Amansha:~/rsa\$". The user enters "gcc bn_sample.c -lcrypto" and then "a.out". The output shows two hexadecimal results: "a * b =" followed by a long string, and "a^c mod n =" followed by another long string. The terminal has a sidebar on the left with icons for a gear, a notepad, and a folder. The top status bar shows "En", battery, and time "2:54 AM".

```
[11/27/21]seed@Amansha:~/rsa$ gcc bn_sample.c -lcrypto
[11/27/21]seed@Amansha:~/rsa$ a.out
a * b =  B0317151CED8DEFC3902E02B0D4DAB82A5548C69F54850
AC58EAD8421AE0C0383A6DD4198EC869C07FD659DAD824FBC
a^c mod n =  A7AE93371F9B6257C127B0C343CCBBEBD9512AB52E
18B7F0BEDEAE08D26601BE
[11/27/21]seed@Amansha:~/rsa$
```

Task 1

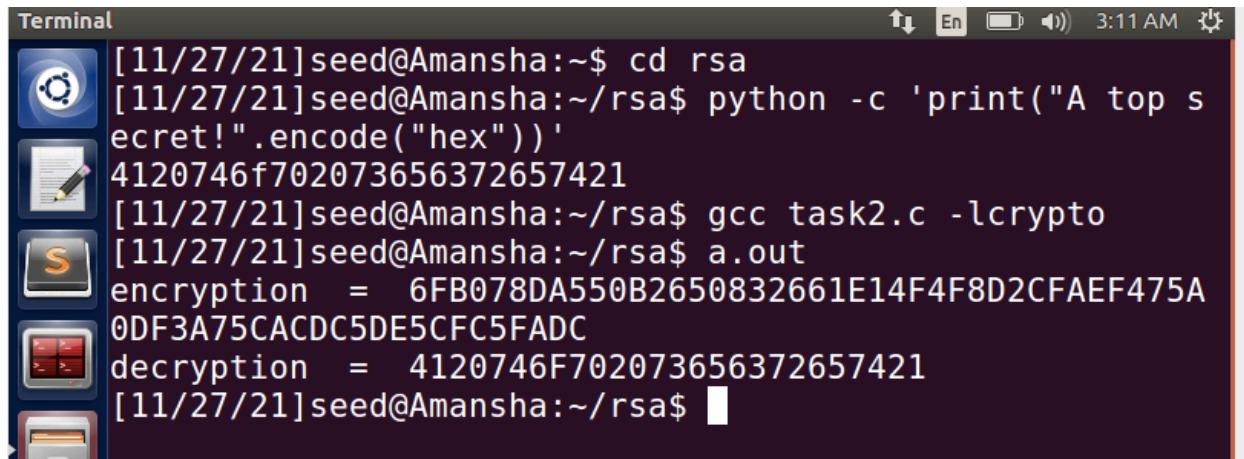
In this task, we derive the private key by using the `p,q,e` hexadecimal values given to us in the lab document. I modify the previous `bn_sample.c` code with the values given and execute the code. In return, we get the private key value "d."

A terminal window titled "Terminal" with a dark background. The prompt is "[11/27/21]seed@Amansha:~/rsa\$". The user enters "gcc task1.c -lcrypto" and then "a.out". The output shows the private key value "d" as a long hexadecimal string. The terminal has a sidebar on the left with icons for a gear, a notepad, and a folder. The top status bar shows "En", battery, and time "3:17 AM".

```
[11/27/21]seed@Amansha:~/rsa$ gcc task1.c -lcrypto
[11/27/21]seed@Amansha:~/rsa$ a.out
Private key =  3587A24598E5F2A21DB007D89D18CC50ABA5075B
A19A33890FE7C28A9B496AEB
[11/27/21]seed@Amansha:~/rsa$
```

Task 2

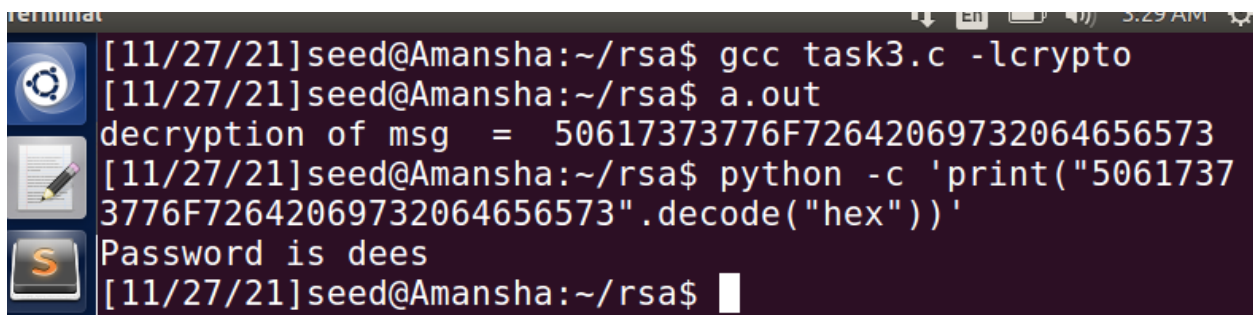
In this task, we print out the hex string of a phrase given to us in the lab document using the python script. Next, to convert the hex string to BIGNUM I edit the previous code with the API BN_hex2bn and use the public and private keys given to us in the lab document to encrypt as well as decrypt to verify the encryption result. The decryption result is the same as the hex string.

A terminal window titled 'Terminal' with a dark background and light text. The window shows a series of commands and their outputs. The user is in the directory ~/rsa. They run a python command to print the hex string of 'A top secret!'. Then they compile a C program task2.c using gcc with the -lcrypto flag. Finally, they run the resulting a.out program, which displays the encryption and decryption results, showing that the decryption matches the original hex string.

```
Terminal [11/27/21]seed@Amansha:~$ cd rsa
[11/27/21]seed@Amansha:~/rsa$ python -c 'print("A top s
ecret!".encode("hex"))'
4120746f702073656372657421
[11/27/21]seed@Amansha:~/rsa$ gcc task2.c -lcrypto
[11/27/21]seed@Amansha:~/rsa$ a.out
encryption = 6FB078DA550B2650832661E14F4F8D2CFAEF475A
0DF3A75CACDC5DE5CFC5FADC
decryption = 4120746F702073656372657421
[11/27/21]seed@Amansha:~/rsa$
```

Task 3

In this task, we decrypt a message by editing the previous code with the ciphertext “C” given to us in the lab document and the same public/private keys used in task 2. I compile and execute the program using gcc compiler and executed the program. I use the decryption result in the python script to print the plaintext “password is dees.”

A terminal window titled 'terminal' with a dark background and light text. The window shows the compilation and execution of task3.c. The user runs gcc task3.c -lcrypto, then ./a.out. The output shows the decryption of a message, resulting in the hex string 50617373776F72642069732064656573. Then, they run a python command to decode this hex string, which outputs 'Password is dees'.

```
terminal [11/27/21]seed@Amansha:~/rsa$ gcc task3.c -lcrypto
[11/27/21]seed@Amansha:~/rsa$ a.out
decryption of msg = 50617373776F72642069732064656573
[11/27/21]seed@Amansha:~/rsa$ python -c 'print("5061737
3776F72642069732064656573".decode("hex"))'
Password is dees
[11/27/21]seed@Amansha:~/rsa$
```

Task 4

In this task, we use the python script to encode the phrase given to us in the lab document to encode it into hex string. Using the same public/private keys in the previous tasks and the hex string printed out, I edit the same program to print out the digital signature of Alice.

I make a tiny change in the phrase from \$2000 to \$3000 and use the python script to print out the hex string. The hex string only has one byte change, but the digital signature value is completely different.

```
[11/27/21]seed@Amansha:~/rsa$ python -c 'print("I owe y  
ou $2000.".encode("hex"))'  
49206f776520796f752024323030302e  
[11/27/21]seed@Amansha:~/rsa$ gcc task4.c -lcrypto  
[11/27/21]seed@Amansha:~/rsa$ a.out  
encryption of msg = 55A4E7F17F04CCFE2766E1EB32ADDBA89  
0BDE92A0FBE2D785ED6E73CCB35E4CB  
[11/27/21]seed@Amansha:~/rsa$ python -c 'print("I owe y  
ou $3000.".encode("hex"))'  
49206f776520796f752024333030302e  
[11/27/21]seed@Amansha:~/rsa$ gcc task4.c -lcrypto  
[11/27/21]seed@Amansha:~/rsa$ a.out  
encryption of msg = BCC20FB7568E5D48E434C387C06A6025E  
90D29D848AF9C3EBAC0135D99305822  
[11/27/21]seed@Amansha:~/rsa$
```

Task 5

In this task, I encode the given phrase in the lab document to encode it into hex string. The lab document gives us Alice's public key and hexadecimal signature and I edit the code using the public key and signature to verify If the signature is Alice's or not. The hex message printed out is the same as the hex msg printed with the python script.

I modify the last byte of Alice's signature from 2F to 3F and repeat the same process. The result is completely different hex value.

```
[11/27/21]seed@Amansha:~/rsa$ python -c 'print("Launch  
a missile.".encode("hex"))'  
4c61756e63682061206d697373696c652e  
[11/27/21]seed@Amansha:~/rsa$ gcc task5.c -lcrypto  
[11/27/21]seed@Amansha:~/rsa$ a.out  
hex msg = 4C61756E63682061206D697373696C652E  
[11/27/21]seed@Amansha:~/rsa$ gcc task5.c -lcrypto  
[11/27/21]seed@Amansha:~/rsa$ a.out  
hex msg = 91471927C80DF1E42C154FB4638CE8BC726D3D66C83  
A4EB6B7BE0203B41AC294  
[11/27/21]seed@Amansha:~/rsa$
```

Task 6

Step 1

In this step, we find a web server to download its certificates. I use the Walmart website to download the certificates by using the openssl command and to display the information I use the -showcert command. As instructed to us, I copy and paste the first certificate displayed in the terminal to a file named c0.pem and the 2nd certificate to a file named c1.pem.

```
[11/27/21]seed@Amansha:~/rsa$ openssl s_client -connect
www.walmart.org:443 -showcerts
CONNECTED(00000003)
depth=2 OU = GlobalSign Root CA - R3, O = GlobalSign, C
N = GlobalSign
verify return:1
depth=1 C = BE, O = GlobalSign nv-sa, CN = GlobalSign R
SA OV SSL CA 2018
verify return:1
depth=0 C = US, ST = Arkansas, L = Bentonville, O = Wal
mart Inc., CN = cdn.corporate.walmart.com
verify return:1
---
Certificate chain
 0 s:/C=US/ST=Arkansas/L=Bentonville/O=Walmart Inc./CN=
cdn.corporate.walmart.com
 1 i:/C=BE/O=GlobalSign nv-sa/CN=GlobalSign RSA OV SSL
CA 2018
-----BEGIN CERTIFICATE-----
MIIJ2jCCCMKgAwIBAgIMBSG7Nllxz5J9sAm0MA0GCSqGSIb3DQEBCwU
AMFAxCzAJ
```

Step 2

I use the command given to us in the lab document to extract the public key from the Walmart web server certificate from the c1.pem file and use the grep command to extract the value of e which is the exponent value.

```
[11/27/21]seed@Amansha:~/rsa$ openssl x509 -in c1.pem
-noout -modulus
Modulus=A75AC9D50C18210023D5970FEBAEDD5C686B6B8F5060137
A81CB97EE8E8A61944B2679F604A72AFBA4DA56BBEEA0A4F07B8A7F
551F4793610D6E71513A2524082F8CE1F789D692CFAFB3A73F30EDB
5DF21AEFEF54417FDD863D92FD3815A6B5FD347B0ACF2AB3B24794F
1FC72EEAB9153A7C184C69B3B52059095E29C363E62E465BAA94904
90EB9F0F54AA1092F7C344DD0BC00C506557906CEA2D010F14843E8
B95AB59555BD31D21B3D86BEA1EC0D12DB2C9924AD47C26F03E67A7
0B570CCCD272CA58C8EC2183C92C92E736F0610569340AAA3C552FB
E5C505D669685C06B9EE5189E18A0E414D9B92900A89E9166BEFEF7
5BE7A46B8E3478A1D1C2EA74F
[11/27/21]seed@Amansha:~/rsa$ openssl x509 -in c1.pem -
text -noout | grep Exponent
      Exponent: 65537 (0x10001)
[11/27/21]seed@Amansha:~/rsa$
```

Step 3

In this step, I extract the signature from the Walmart web serve certificate from the c0.pem file using the command given to us in the lab document.

```
[11/27/21]seed@Amansha:~/rsa$ openssl x509 -in c0.pem -
text -noout
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number:
      05:21:bb:36:59:71:cf:92:7d:b0:09:b4
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: C=BE, O=GlobalSign nv-sa, CN=GlobalSign
RSA OV SSL CA 2018
    Validity
      Not Before: Oct 14 17:30:01 2021 GMT
      Not After : Nov 15 17:30:01 2022 GMT
    Subject: C=US, ST=Arkansas, L=Bentonville, O=Wa
lmart Inc., CN=cdn.corporate.walmart.com
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      Public-Key: (2048 bit)
      Modulus:
        00:c3:67:b0:82:3b:a0:af:6b:94:e8:61
```


I scroll down to find the signature block and copy and paste the information into a file named "signature."

```
Signature Algorithm: sha256WithRSAEncryption
47:d6:66:ea:94:70:e5:48:7a:67:09:46:00:c4:d8:e
d:15:5d:
99:48:90:b2:48:f8:41:88:e5:25:6e:a3:94:97:04:a
a:40:d9:
2f:b4:ae:b1:60:f1:f7:c6:70:2c:b5:d9:f6:c2:ea:3
9:fc:33:
5b:fa:33:ad:da:8a:8c:95:23:ac:bb:c8:02:26:7c:f
e:39:97:
4d:82:ed:78:71:25:91:dd:69:ad:b1:d7:e9:6a:cf:8
6:57:db:
36:82:6e:84:7b:36:35:61:b9:d0:fd:22:c6:50:1c:f
7:c6:c8:
c8:bb:d1:eb:32:0f:dd:37:e7:0f:46:3d:05:ab:16:4
a:2a:a7:
fb:77:fb:58:f7:e6:5b:5f:47:8b:35:50:34:c9:6b:8
0:a6:7e:
ae:75:04:14:04:8c:75:99:fc:cb:27:3d:b9:07:e8:a
3:49:5b:
d7:13:c4:41:c3:b4:d7:6e:4c:b4:e3:b8:7d:b0:b8:c
6:66:d5:
de:ea:56:65:7c:a5:0a:3e:ad:be:bd:ff:b2:a8:c0:4
```

Next, I use the command given to us in the lab document "cat" to extract the signature block without colons : and the spaces.

```
[11/27/21]seed@Amansha:~/rsa$ cat signature | tr -d '[:
space:]':
47d666ea9470e5487a67094600c4d8ed155d994890b248f84188e52
56ea3949704aa40d92fb4aeb160f1f7c6702cb5d9f6c2ea39fc335b
fa33adda8a8c9523acbbc802267cfe39974d82ed78712591dd69adb
1d7e96acf8657db36826e847b363561b9d0fd22c6501cf7c6c8c8bb
d1eb320fdd37e70f463d05ab164a2aa7fb77fb58f7e65b5f478b355
034c96b80a67eae750414048c7599fccb273db907e8a3495bd713c4
41c3b4d76e4cb4e3b87db0b8c666d5deea56657ca50a3eadbebdffb
2a8c04f27dccf096c32ef83a8d180f8713c0faeaba9a903336349e6
9818ee40c7c75b7ce8d3bd82ef6e921fff02609f1ad8ac69082374b
2e2aa8fb7444b8f67[11/27/21]seed@Amansha:~/rsa$
```


Step 4

In this step, I display the body of the certificate by using the openssl asn1parse command below.

```
[11/27/21]seed@Amansha:~/rsa$ openssl asn1parse -i -in c0.pem
 0:d=0  hl=4  l=2522 cons: SEQUENCE
 4:d=1  hl=4  l=2242 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= 12 prim: INTEGER           :0521B
B365971CF927DB009B4
27:d=2  hl=2  l= 13 cons: SEQUENCE
29:d=3  hl=2  l=  9 prim: OBJECT           :sha2
56WithRSAEncryption
40:d=3  hl=2  l=  0 prim: NULL
42:d=2  hl=2  l= 80 cons: SEQUENCE
44:d=3  hl=2  l= 11 cons: SET
46:d=4  hl=2  l=  9 cons: SEQUENCE
48:d=5  hl=2  l=  3 prim: OBJECT           :co
untryName
53:d=5  hl=2  l=  2 prim: PRINTABLESTRING :BE
57:d=3  hl=2  l= 25 cons: SET
59:d=4  hl=2  l= 23 cons: SEQUENCE
```

I extract the body of the certificate without the signature block and save it in a file called c0.body.bin in the command below then use the sha256sum command to calculate the hash.

```
[11/27/21]seed@Amansha:~/rsa$ openssl asn1parse -i -in c0.pem -strparse 4 -out c0_body.bin -noout
[11/27/21]seed@Amansha:~/rsa$ sha256sum c0_body.bin
bd03ae2ada697c8392a9376b0f0fe531b95976b4f060200b151b302a9581b3a4  c0_body.bin
[11/27/21]seed@Amansha:~/rsa$
```

Step 5

Lastly, I edit the code used in previous tasks with the information we found in step 2 and 3 which is the signature value and public key value. We verify if the signature of the certificate is valid or not by compiling and executing the program in the screenshot below. It is valid.

```
[11/27/21]seed@Amansha:~/rsa$ gcc task6.c -lcrypto
[11/27/21]seed@Amansha:~/rsa$ a.out
hex msg = 01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF003031300D060960864801
650304020105000420BD03AE2ADA697C8392A9376B0F0FE531B9597
6B4F060200B151B302A9581B3A4
[11/27/21]seed@Amansha:~/rsa$
```

```
// Initialize variables
BN_hex2bn
(&n, "A75AC9D50C18210023D5970FEBAEDD5C686B6B8F5060137A81CB97EE8E8A61944B2679F604A72
BN_hex2bn(&e, "010001");
BN_hex2bn
(&S, "47d666ea9470e5487a67094600c4d8ed155d994890b248f84188e5256ea3949704aa40d92fb4a
```