Applied Cryptography

Name	Naman Choudhary
SRN	PES2UG20CS209
Section	D

Lab 4

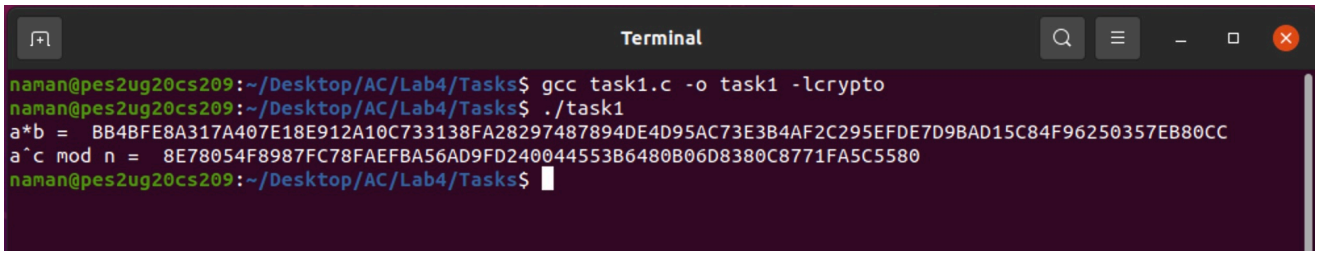
RSA Public-Key Encryption and Signature Lab

Task 1: A Complete Example of BIGNUM

Code

```c
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 256

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*a = BN_new();
    BIGNUM*b = BN_new();
    BIGNUM*n = BN_new();
    BIGNUM*res = BN_new();

    // Initialize a, b, n
    BN_generate_prime_ex(a, NBITS, 1, NULL, NULL, NULL);
    BN_dec2bn(&b, "273489463796838501848592769467194369268");
    BN_rand(n, NBITS, 0, 0);

    // res = a*b
    BN_mul(res, a, b, ctx);
    printBN("a*b = ", res);

    // res = a^b mod n
    BN_mod_exp(res, a, b, n, ctx);
    printBN("a^c mod n = ", res);
    return 0;
}
```

Screenshots



Observation: Required Big Number was computed

`a*b = BB4BFE8A317A407E18E912A10C733138FA28297487894DE4D95AC73E3B4AF2C295EFDE7D9BAD15C84F96250357EB80CC`
`a^c mod n = 8E78054F8987FC78FAEFBA56AD9FD240044553B6480B06D8380C8771FA5C5580`

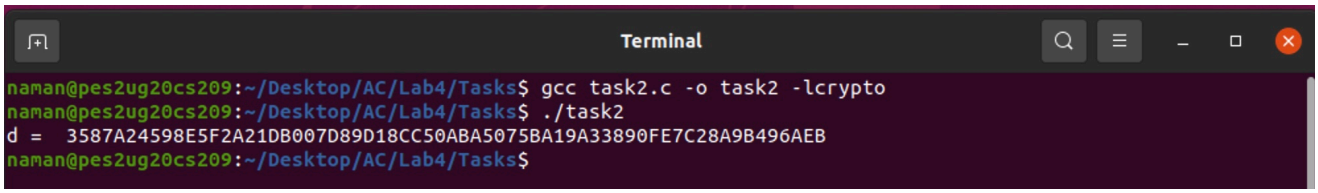
Task 2: Deriving the Private Key

Code

C

```c
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 256

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 *p= BN_new();
    BIGNUM *q= BN_new();
    BIGNUM *e= BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *res1= BN_new();
    BIGNUM *res2= BN_new();
    BIGNUM *res3= BN_new();
    BIGNUM *one = BN_new();

    //initialize
    BN_hex2bn(&p,"F7E75FDC469067FFDC4E847C51F452DF");
    BN_hex2bn(&q,"E85CED54AF57E53E092113E62F436F4F");
    BN_hex2bn(&e,"0D88C3");
    BN_dec2bn(&one,"1");

    BN_sub(res1,p,one);
    BN_sub(res2,q,one);
    BN_mul(res3,res1,res2,ctx);
    BN_mod_inverse(d,e,res3,ctx);
    printBN("d = ",d);
    return 0;
}
```

Screenshots



Observation: Private Key was found out

`d=3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB`

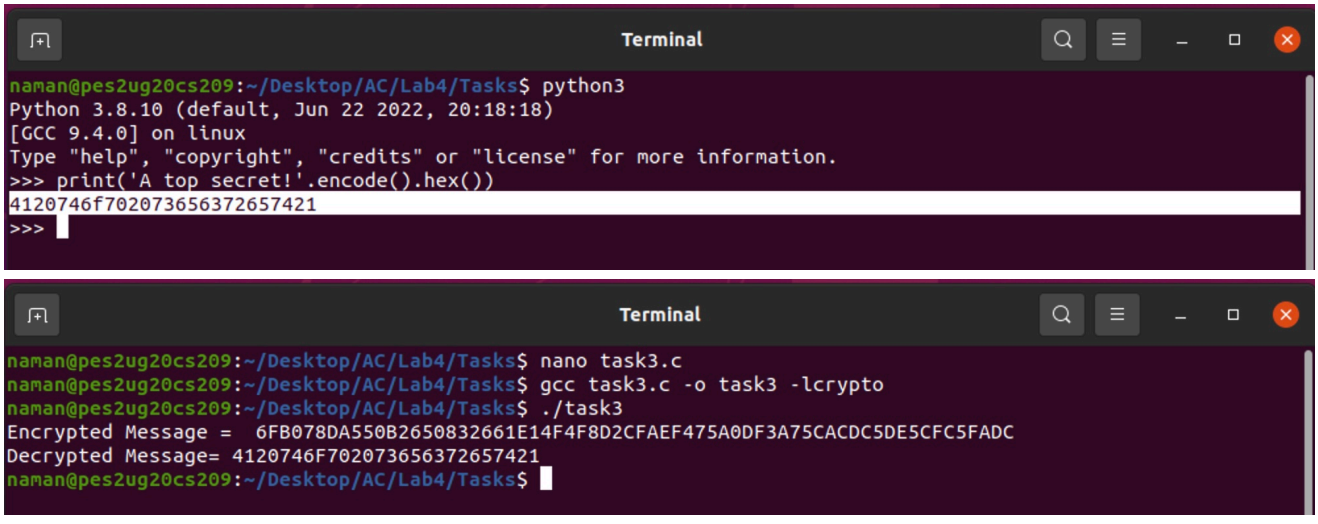
The above code multiplies first performs res3=p*q Then, it performs mod inverse of d & e with respect to res3 (and given e) The output, d is printed on the screen

Task 3: Encrypting a Message

Code

```c
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 256

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 *m = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *enc = BN_new();
    BIGNUM *dec = BN_new();

    // Initialize p, q, e
    BN_hex2bn(&m,"/*Enter the hex encoded message from previous step here*/");
    BN_hex2bn(&e,"010001");
    BN_hex2bn(&n,"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

    // Encryption : ree mod n
    BN_mod_exp(enc, m, e, n, ctx);
    printBN("Encrypted Message = ", enc);

    //Decryption: enc^d mod n
    BN_mod_exp(dec, enc, d, n, ctx);
    printBN("Decrypted Message=",dec);
    return 0;
}
```

Screenshots



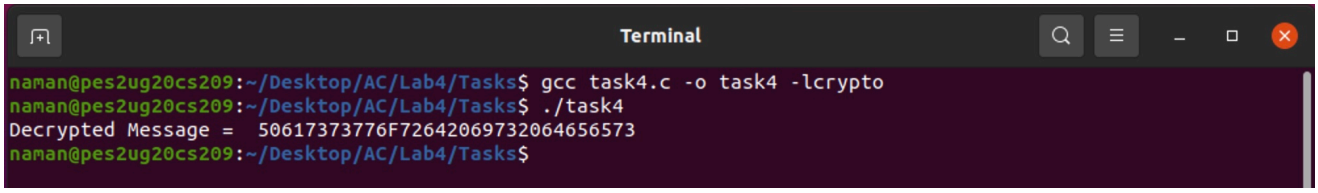
Observation: Enc and Dec verified

Task 4: Decrypting a Message

Code

```c
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 256

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 *m = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *n =BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *enc =BN_new();
    BIGNUM *dec=BN_new();

    // Initialize p, q, e
    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
    BN_hex2bn(&enc, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");

    //Decryption: enc^d mod n
    BN_mod_exp(dec, enc, d, n, ctx);
    printBN("Decrypted Message = ", dec);
    return 0;
}
```

Screenshots



Observation: Message Succesfully decrypted

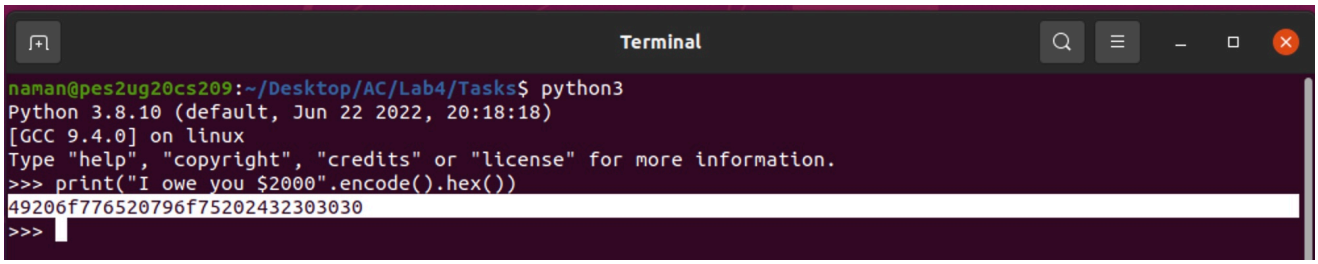
After conversion from Hex to Ascii `Password is dees` is the output

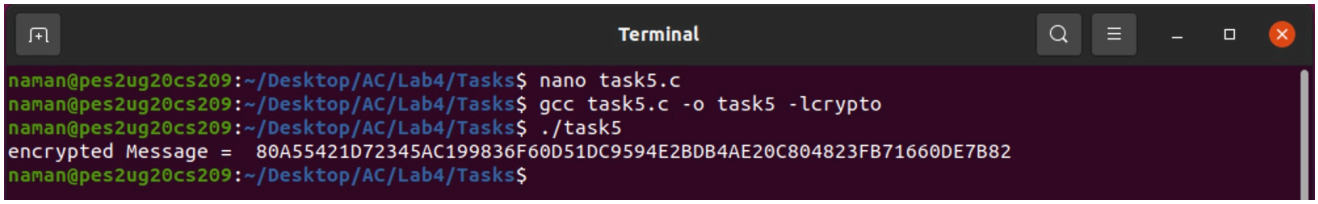
Task 5: Signing a Message

Code

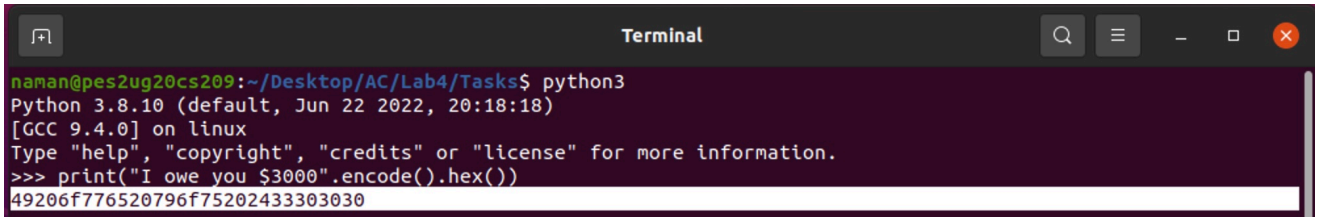
```c
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 256

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 *m = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *sign = BN_new();

    // Initialize p, q, e
    BN_hex2bn(&m,"/* hex encoded message here */");
    BN_hex2bn(&n,"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    BN_hex2bn(&d,"74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

    // signing : m^e mod n
    BN_mod_exp(sign, m, d, n, ctx);
    printBN("encrypted Message = ", sign);
    return 0;
}
```

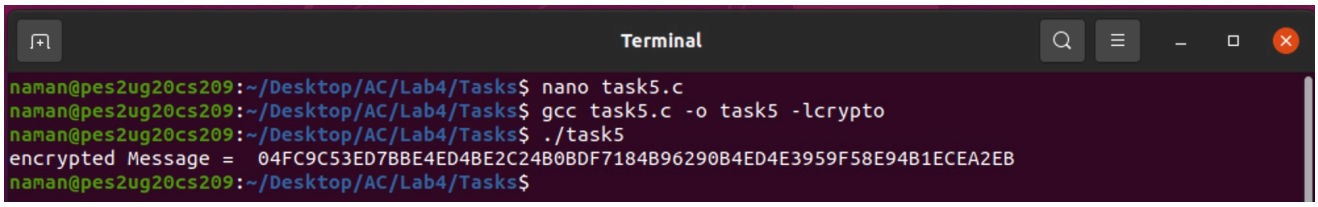
Screenshots





After Modification to `$3000`





Observation: In spite of the string having only 1 character changed, we observe that the encrypted text is completely modified

`$2000 gives output 80A55421D72345AC199836F60D51DC9594E2BDB4AE20C804823FB71660DE7B82`

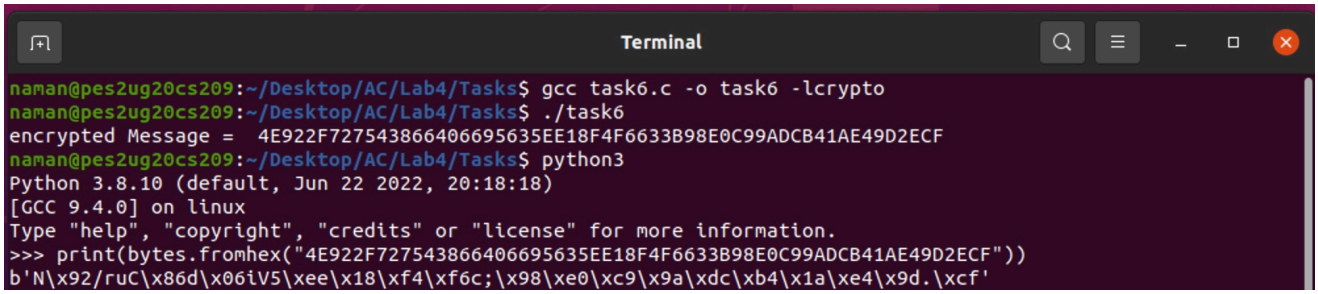
`$3000 gives output 04FC9C53ED7BBE4ED4BE2C24BOBDF7184B96290B4ED4E3959F58E94B1ECEA2EB`

Task 6: Verifying a signature

Code

```c
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 256
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 *s = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *message = BN_new();

    // Initialize p, q, e
    BN_hex2bn(&s,"643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");
    BN_hex2bn(&n,"AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18 116115");
    BN_hex2bn(&e, "010001");

    // signing : m^e mod n
    BN_mod_exp(message, s, e, n, ctx);
    printBN("encrypted Message = ", message);
    return 0;
}
```

Screenshots



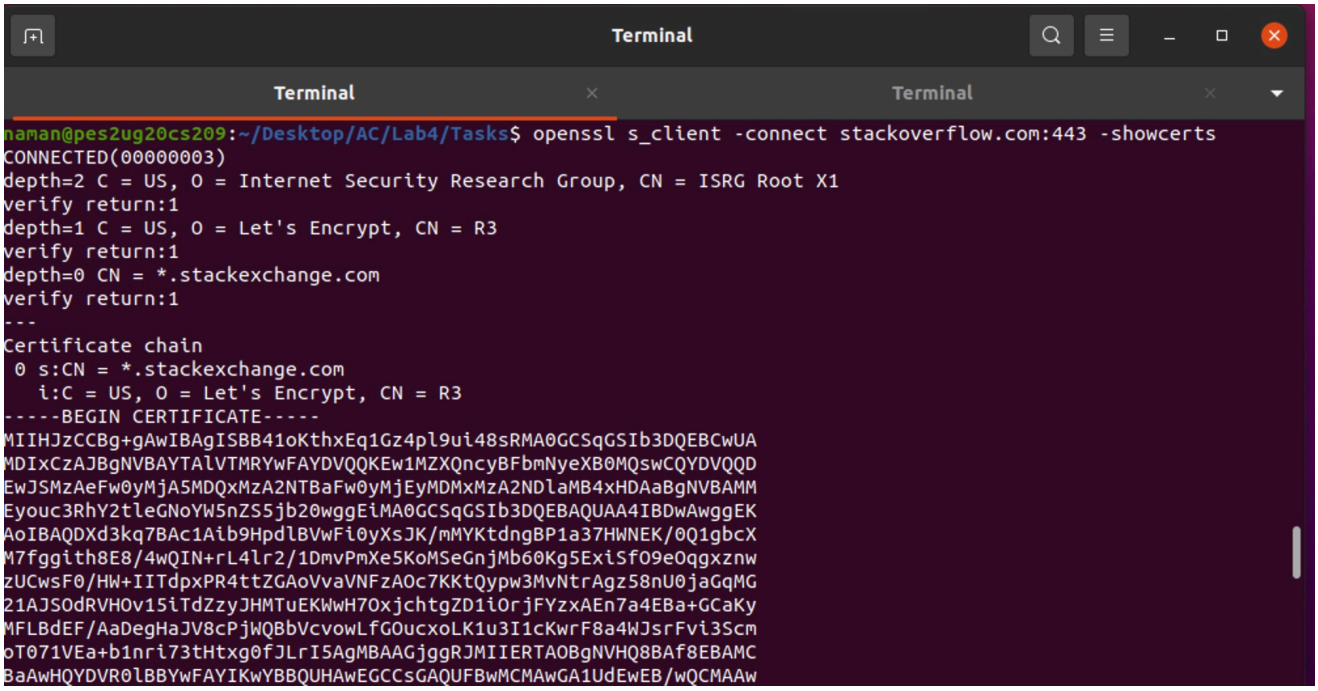
Observation: Signature generated

Task 7: Manually Verifying X.509 Certificate

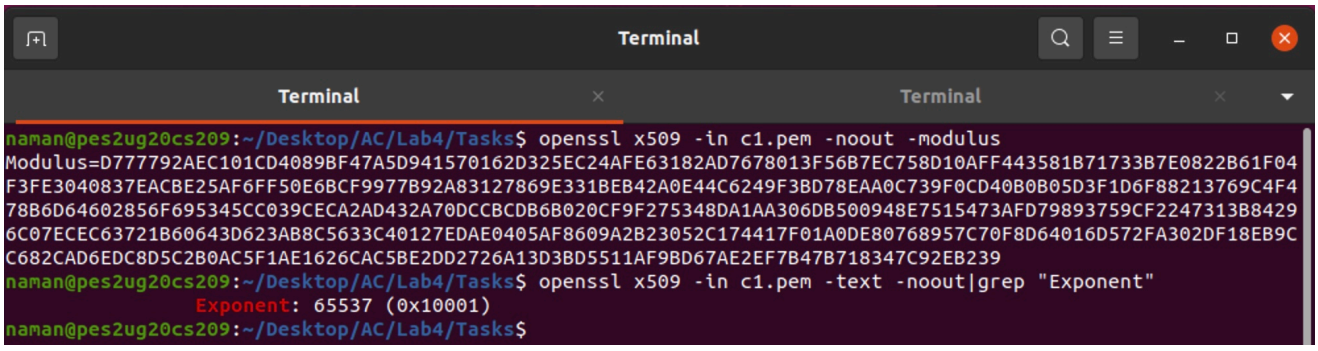
Code

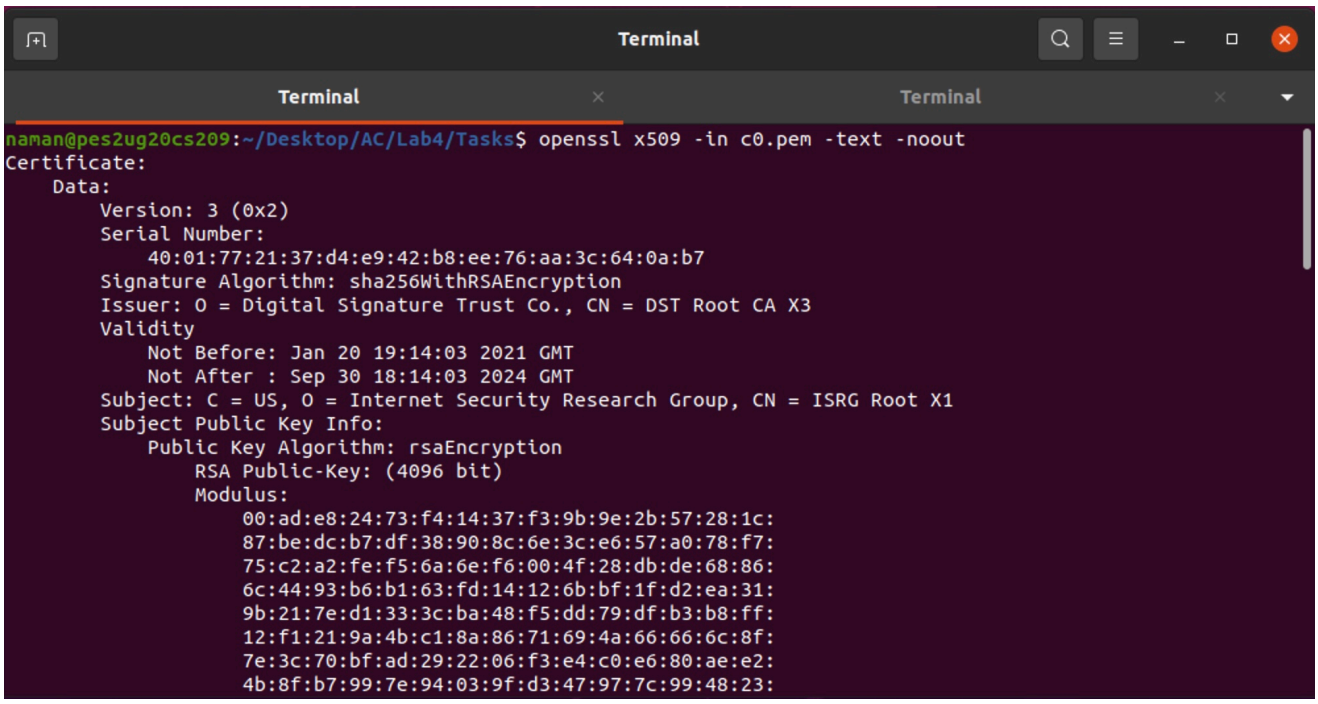
```c
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 256

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 *s = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *message = BN_new();

    // Initialize p, q, e
    /* Insert the values of n and e from step 2 */
    /* Insert the value of s from step 3 */
    BN_hex2bn(&s,"");
    BN_hex2bn(&n, "");
    BN_hex2bn(&e, "");
    // signing : m^e mod n
    BN_mod_exp(message, s, e, n, ctx);
    printBN("encrypted Message = ", message);
    return 0;
}
```

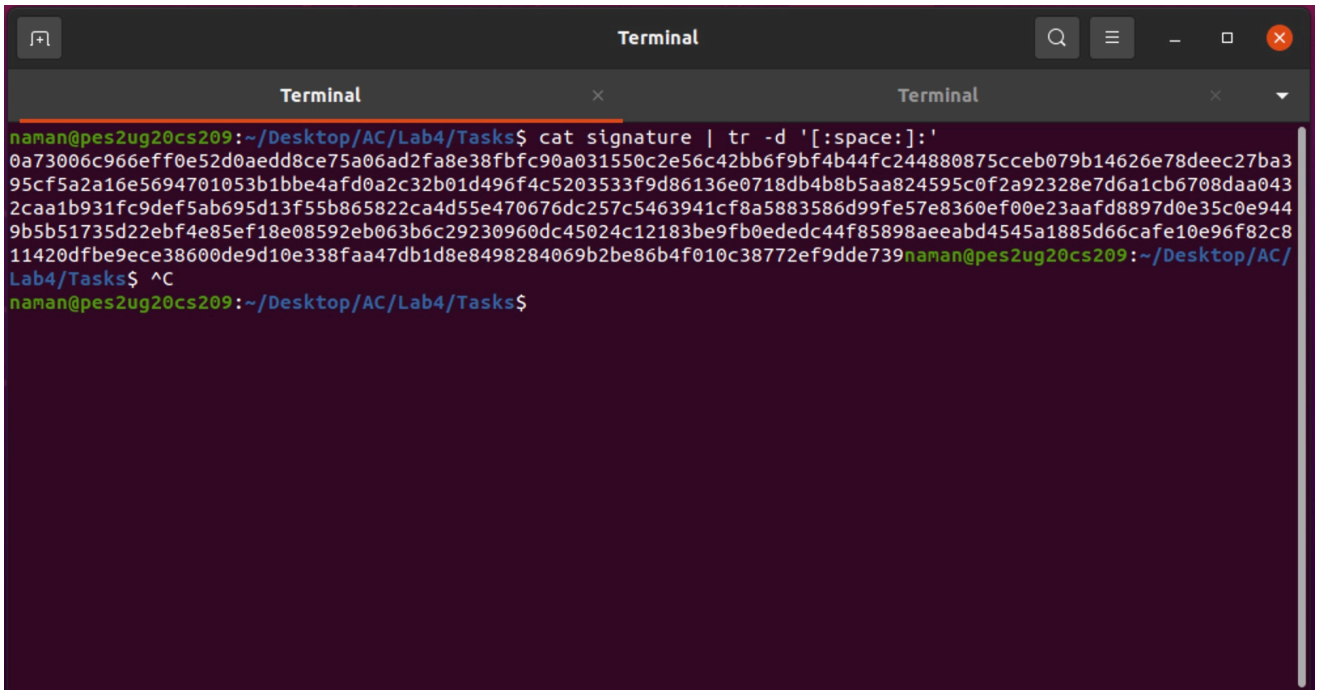
Screenshots

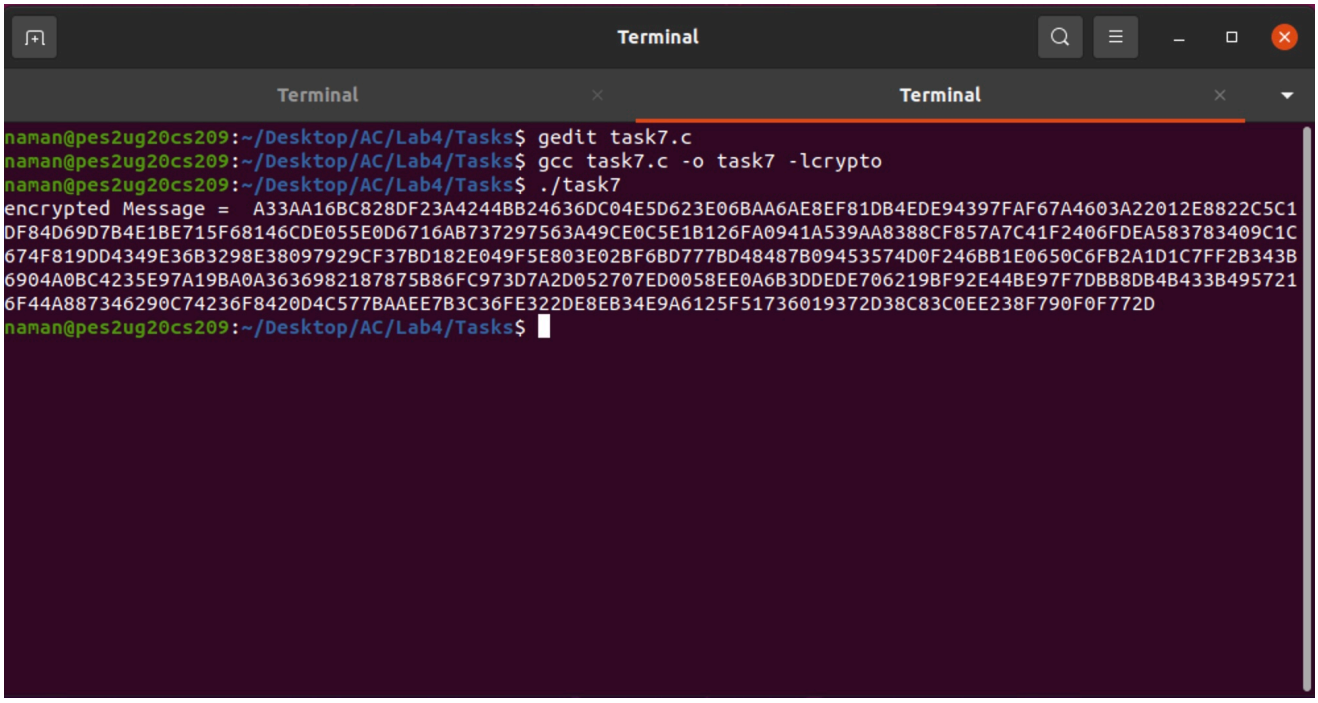

Terminal
Terminal
Terminal
naman@pes2ug20cs209:~/Desktop/AC/Lab4/Tasks$ openssl s_client -connect stackoverflow.com:443 -showcerts
CONNECTED(00000003)
depth=2 C = US, O = Internet Security Research Group, CN = ISRG Root X1
verify return:1
depth=1 C = US, O = Let's Encrypt, CN = R3
verify return:1
depth=0 CN = *.stackexchange.com
verify return:1

Certificate chain
 0 s:CN = *.stackexchange.com
   i:C = US, O = Let's Encrypt, CN = R3
-----BEGIN CERTIFICATE-----
MIIHJzCCBg+gAwIBAgISBB41oKthxEq1Gz4pl9ui48sRMA0GCSqGSIb3DQEBCwUA
MDIxCzAJBgNVBAYTAlVTMRYwFAYDVQQKEw1MZXQncyBFbmNyeXB0MQswCQYDVQQD
EwJSMzAeFw0yMjA5MDQxMzA2NTBaFw0yMjEyMDMxMzA2NDlaMB4xHDAaBgNVBAMM
Eyouc3RhY2tleGNoYW5nZS5jb20wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEK
AoIBAQDXd3kq7BAc1Aib9HpdlBVwFi0yXsJK/mMYKtdngBP1a37HWNEK/0Q1gbcX
M7fggith8E8/4wQIN+rL4lr2/1DmvPmXe5KoMSeGnjMb60Kg5ExiSfO9eOqgxznw
zUCwsF0/HW+IITdpxPR4ttZGAoVvaVNFzAOc7KKtQypw3MvNtrAgz58nU0jaGqMG
21AJSOdRVHOv15iTdZzyJHMTuEKWwH7OxjchtgZD1iOrjFYzxAEn7a4EBa+GCaKy
MFLBdEF/AaDegHaJV8cPjWQBbVcvowLfGOucxoLK1u3I1cKwrF8a4WJsrFvi3Scm
oT071VEa+b1nri73tHtxg0fJLrI5AgMBAAGjggRJMIIERTAOBgNVHQ8BAf8EBAMC
BaAwHQYDVR0lBBYwFAYIKwYBBQUHAwEGCCsGAQUFBwMCMAwGA1UdEwEB/wQCMAAw


Terminal
Terminal
Terminal
naman@pes2ug20cs209:~/Desktop/AC/Lab4/Tasks$ openssl x509 -in c1.pem -noout -modulus
Modulus=D777792AEC101CD4089BF47A5D941570162D325EC24AFE63182AD7678013F56B7EC758D10AFF443581B71733B7E0822B61F04
F3FE3040837EACBE25AF6FF50E6BCF9977B92A83127869E331BEB42A0E44C6249F3BD78EAA0C739F0CD40B0B05D3F1D6F88213769C4F4
78B6D64602856F695345CC039CECA2AD432A70DCCBCDB6B020CF9F275348DA1AA306DB500948E7515473AFD79893759CF2247313B8429
6C07ECEC63721B60643D623AB8C5633C40127EDAE0405AF8609A2B23052C174417F01A0DE80768957C70F8D64016D572FA302DF18EB9C
C682CAD6EDC8D5C2B0AC5F1AE1626CAC5BE2DD2726A13D3BD5511AF9BD67AE2EF7B47B718347C92EB239
naman@pes2ug20cs209:~/Desktop/AC/Lab4/Tasks$ openssl x509 -in c1.pem -text -noout|grep "Exponent"
            Exponent: 65537 (0x10001)
naman@pes2ug20cs209:~/Desktop/AC/Lab4/Tasks$


Terminal
Terminal
Terminal
naman@pes2ug20cs209:~/Desktop/AC/Lab4/Tasks$ openssl x509 -in c0.pem -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            40:01:77:21:37:d4:e9:42:b8:ee:76:aa:3c:64:0a:b7
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: O = Digital Signature Trust Co., CN = DST Root CA X3
        Validity
            Not Before: Jan 20 19:14:03 2021 GMT
            Not After : Sep 30 18:14:03 2024 GMT
        Subject: C = US, O = Internet Security Research Group, CN = ISRG Root X1
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (4096 bit)
                Modulus:
                    00:ad:e8:24:73:f4:14:37:f3:9b:9e:2b:57:28:1c:
                    87:be:dc:b7:df:38:90:8c:6e:3c:e6:57:a0:78:f7:
                    75:c2:a2:fe:f5:6a:6e:f6:00:4f:28:db:de:68:86:
                    6c:44:93:b6:b1:63:fd:14:12:6b:bf:1f:d2:ea:31:
                    9b:21:7e:d1:33:3c:ba:48:f5:dd:79:df:b3:b8:ff:
                    12:f1:21:9a:4b:c1:8a:86:71:69:4a:66:66:6c:8f:
                    7e:3c:70:bf:ad:29:22:06:f3:e4:c0:e6:80:ae:e2:
                    4b:8f:b7:99:7e:94:03:9f:d3:47:97:7c:99:48:23:





Observation:Signature verified