



**Faculty of Engineering & Technology
Electrical & Computer Engineering Department**

ENCS4320

RSA Public-Key Encryption and Signature Lab Report

Prepared by:

Tareq Shannak 1181404

Instructor: Dr. Hanna Al-Zughbi

Section: 2

Date: 2 January 2022

Table of Contents

Task 1: Deriving the Private Key	IV
Task 2: Encrypting a Message	IV
Task 3: Decrypting a Message.....	V
Task 4: Signing a Message	VI
Task 5: Verifying a Signature	VII
Task 6: Manually Verifying an X.509 Certificate	IX
Appendix	X
Appendix ¹	X
Appendix ²	XI
Appendix ³	XI
Appendix ⁴	XII
Appendix ⁵	XIII
Appendix ⁶	XIII

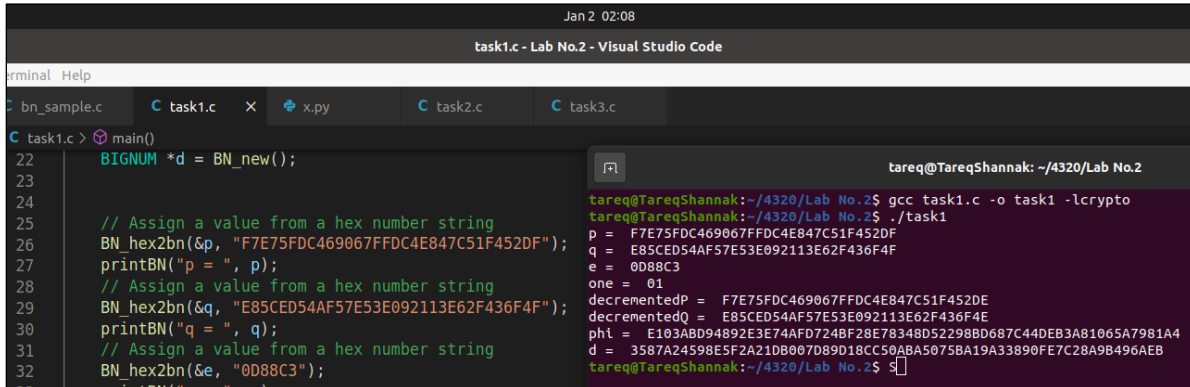
List of Figures

Figure 1 - Deriving the Private Key	IV
Figure 2 - Getting the hex of a Message	IV
Figure 3 - Encrypting a Message	IV
Figure 4 - Decrypting a Message	V
Figure 5 - Getting ASCII from a hex Message.....	V
Figure 6 - Getting the hex of messages for signing.....	VI
Figure 7 - Signing Two Messages	VI
Figure 8 - Verifying a Signature.....	VII
Figure 9 - Getting the ASCII of hex message for Verifying	VII
Figure 10 - Verifying a signature after it corrupted	VIII
Figure 11 - Invalid Message	VIII
Figure 12 - Verifying an X.509 Certificate	IX

Task 1: Deriving the Private Key

From code in [Appendix¹](#), we can derive the private key as shown on Figure 1 after computing $\phi(n)$ where:

$$\phi(n) = \phi(p * q) = (p - 1)(q - 1), \text{ where } p \text{ \& } q \text{ are prime numbers.}$$



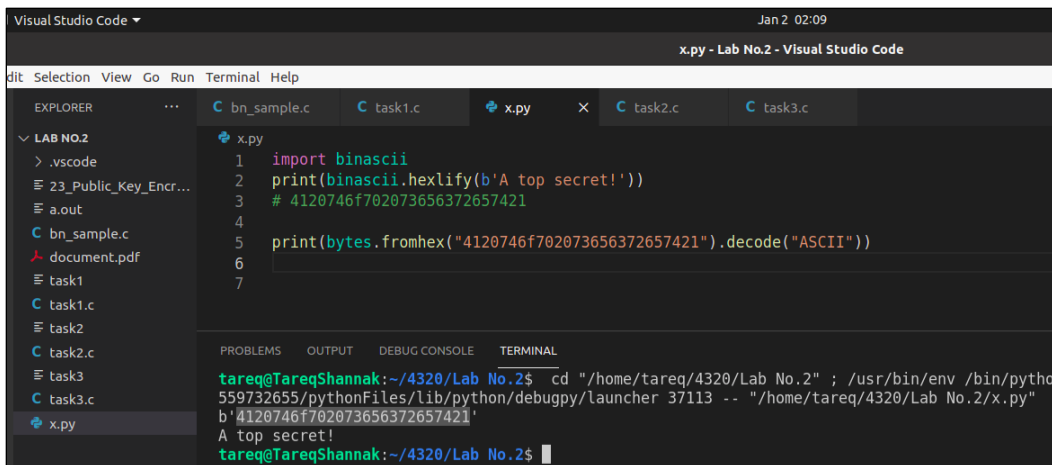
```
Jan 2 02:08
task1.c - Lab No.2 - Visual Studio Code
Terminal Help
bn_sample.c task1.c x.py task2.c task3.c
C task1.c main()
22 BIGNUM *d = BN_new();
23
24
25 // Assign a value from a hex number string
26 BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
27 printBN("p = ", p);
28 // Assign a value from a hex number string
29 BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
30 printBN("q = ", q);
31 // Assign a value from a hex number string
32 BN_hex2bn(&e, "0D88C3");
33 printBN("e = ", e);

tareq@TareqShannak: ~/4320/Lab No.2
tareq@TareqShannak:~/4320/Lab No.2$ gcc task1.c -o task1 -lcrypto
tareq@TareqShannak:~/4320/Lab No.2$ ./task1
p = F7E75FDC469067FFDC4E847C51F452DF
q = E85CED54AF57E53E092113E62F436F4F
e = 0D88C3
one = 01
decrementedP = F7E75FDC469067FFDC4E847C51F452DE
decrementedQ = E85CED54AF57E53E092113E62F436F4E
phi = E103ABD94892E3E74AFD724BF28E78348D52298BD667C44DEB3A81065A7981A4
d = 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB
tareq@TareqShannak:~/4320/Lab No.2$
```

Figure 1 - Deriving the Private Key

Task 2: Encrypting a Message

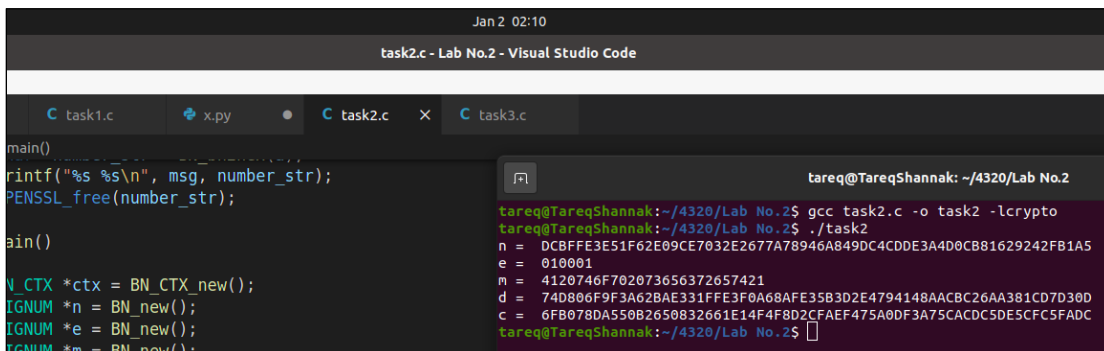
Figure 2 shows how to convert an ASCII string to hex data which is used in encryption as shown on Figure 3, the cipher text c differs than the hex of the original string and not in the same length. The code is in [Appendix²](#).



```
Visual Studio Code Jan 2 02:09
x.py - Lab No.2 - Visual Studio Code
dit Selection View Go Run Terminal Help
EXPLORER bn_sample.c task1.c x.py task2.c task3.c
LAB NO.2
.vscode
23_Public_Key_Encr...
a.out
bn_sample.c
document.pdf
task1
task1.c
task2
task2.c
task3
task3.c
x.py
1 import binascii
2 print(binascii.hexlify(b'A top secret!'))
3 # 4120746f702073656372657421
4
5 print(bytes.fromhex("4120746f702073656372657421").decode("ASCII"))
6
7

tareq@TareqShannak:~/4320/Lab No.2$ cd "/home/tareq/4320/Lab No.2" ; /usr/bin/env /bin/pytho
559732655/pythonFiles/lib/python/launcher 37113 -- "/home/tareq/4320/Lab No.2/x.py"
b'4120746f702073656372657421'
A top secret!
tareq@TareqShannak:~/4320/Lab No.2$
```

Figure 2 - Getting the hex of a Message



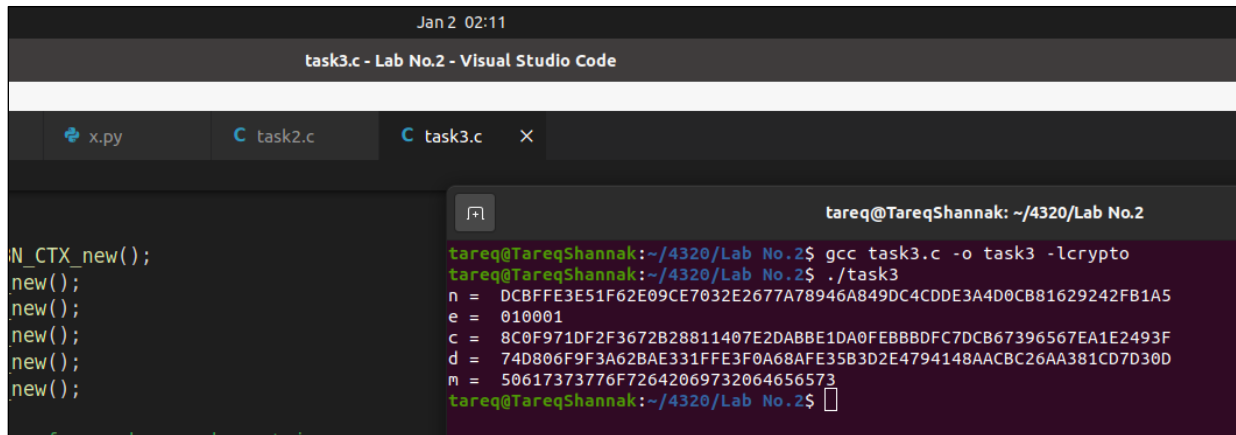
```
Jan 2 02:10
task2.c - Lab No.2 - Visual Studio Code
task1.c x.py task2.c task3.c
main()
printf("%s %s\n", msg, number_str);
PENSSL_free(number_str);
main()
CTX *ctx = BN_CTX_new();
BIGNUM *n = BN_new();
BIGNUM *e = BN_new();
BIGNUM *m = BN_new();

tareq@TareqShannak:~/4320/Lab No.2$ gcc task2.c -o task2 -lcrypto
tareq@TareqShannak:~/4320/Lab No.2$ ./task2
n = DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDE3A4D0C881629242FB1A5
e = 010001
m = 4120746f702073656372657421
d = 74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD703D0
c = 6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5E5FC5FAD
tareq@TareqShannak:~/4320/Lab No.2$
```

Figure 3 - Encrypting a Message

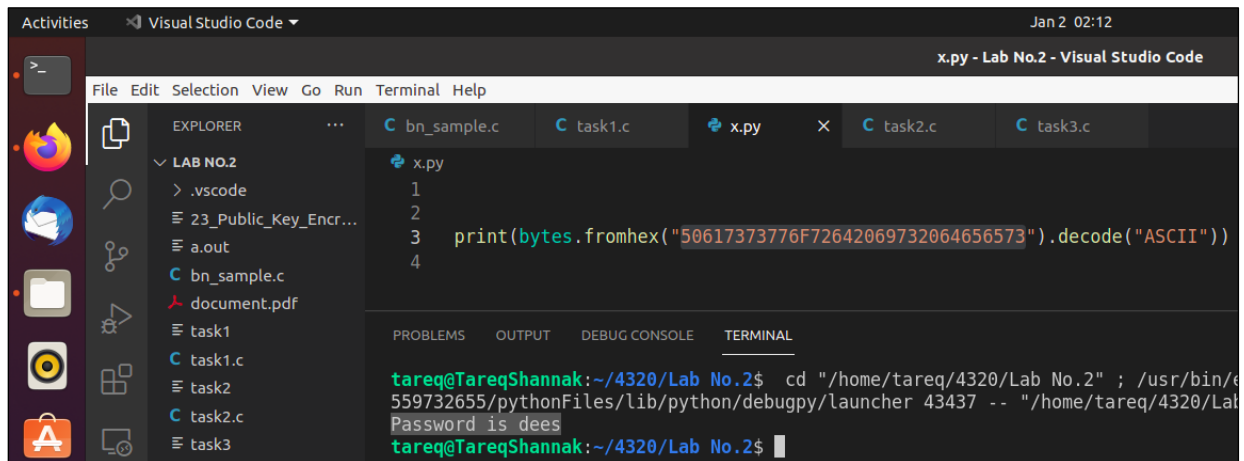
Task 3: Decrypting a Message

In this task, we decrypted a cipher text using the code in [Appendix³](#) and the result is in hex as shown on Figure 4. After that, the hex message converted to ASCII as shown on figure 5 to see the message which is “Password is dees”.



```
Jan 2 02:11
task3.c - Lab No.2 - Visual Studio Code
x.py task2.c task3.c
tareq@TareqShannak: ~/4320/Lab No.2
tareq@TareqShannak:~/4320/Lab No.2$ gcc task3.c -o task3 -lcrypto
tareq@TareqShannak:~/4320/Lab No.2$ ./task3
n = DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5
e = 010001
c = 8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBD7DCB67396567EA1E2493F
d = 74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D
m = 50617373776F72642069732064656573
tareq@TareqShannak:~/4320/Lab No.2$
```

Figure 4 - Decrypting a Message



```
Activities Visual Studio Code Jan 2 02:12
x.py - Lab No.2 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER bn_sample.c task1.c x.py task2.c task3.c
LAB NO.2
.vscode
23_Public_Key_Encr...
a.out
bn_sample.c
document.pdf
task1
task1.c
task2
task2.c
task3
x.py
1
2
3 print(bytes.fromhex("50617373776F72642069732064656573").decode("ASCII"))
4
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
tareq@TareqShannak:~/4320/Lab No.2$ cd "/home/tareq/4320/Lab No.2" ; /usr/bin/c
559732655/pythonFiles/lib/python/debugpy/launcher 43437 -- "/home/tareq/4320/Lab
Password is dees
tareq@TareqShannak:~/4320/Lab No.2$
```

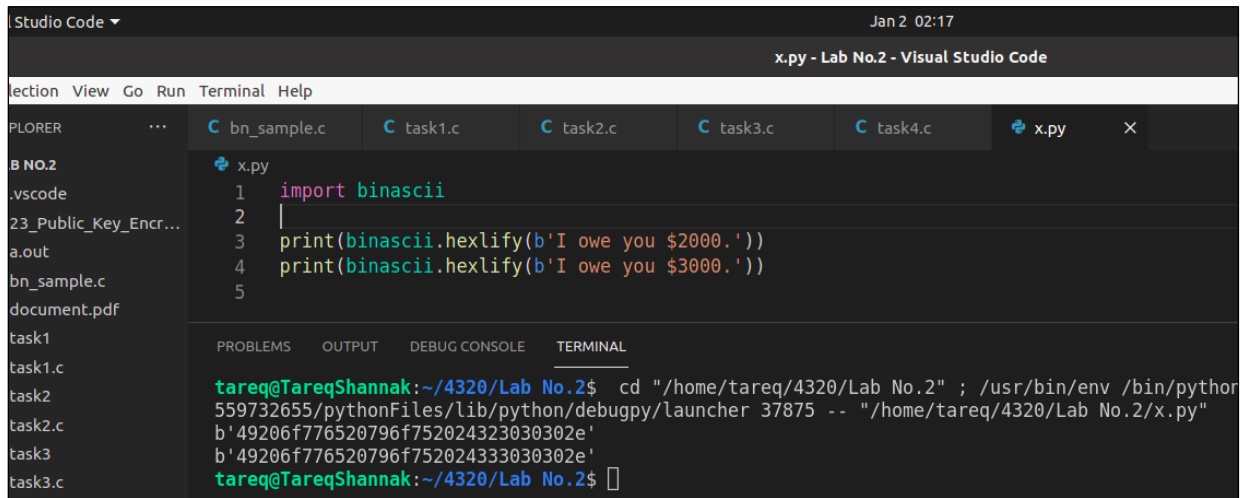
Figure 5 - Getting ASCII from a hex Message

Task 4: Signing a Message

In this task, we signed two messages using private/public keys as shown on Figure 7, first we obtained the hex of the messages in python as shown on Figure 6. The code is in [Appendix 4](#) and the signing algorithm is as following:

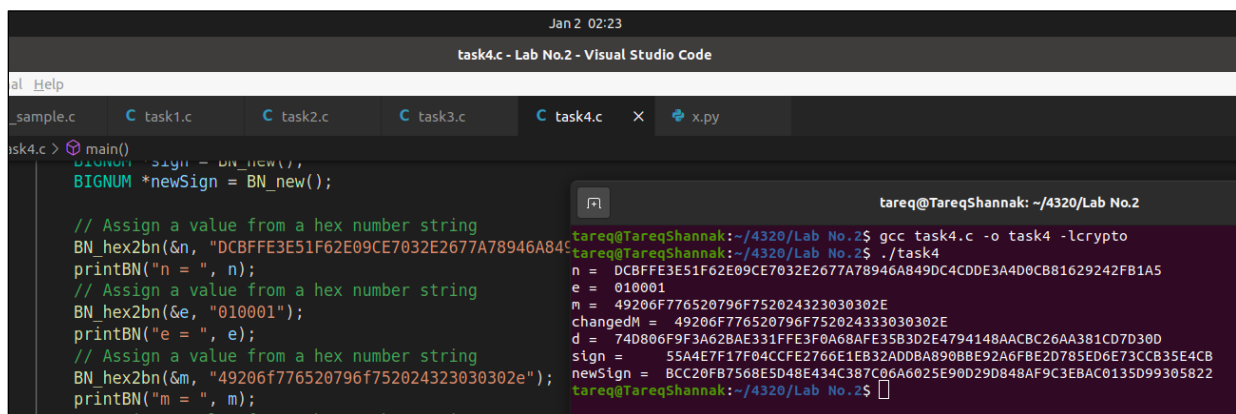
$$\text{Sign} = M^D \text{mod } N$$

We can notice the difference between the two signs because of changing one character in the message which is good.



```
Studio Code Jan 2 02:17
x.py - Lab No.2 - Visual Studio Code
File Explorer View Go Run Terminal Help
bn_sample.c task1.c task2.c task3.c task4.c x.py
x.py
1 import binascii
2
3 print(binascii.hexlify(b'I owe you $2000.))
4 print(binascii.hexlify(b'I owe you $3000.))
5
Terminal
tareq@TareqShannak:~/4320/Lab No.2$ cd "/home/tareq/4320/Lab No.2" ; /usr/bin/env /bin/python
559732655/pythonFiles/lib/python/debugpy/launcher 37875 -- "/home/tareq/4320/Lab No.2/x.py"
b'49206f776520796f752024323030302e'
b'49206f776520796f752024333030302e'
tareq@TareqShannak:~/4320/Lab No.2$
```

Figure 6 - Getting the hex of messages for signing



```
Jan 2 02:23
task4.c - Lab No.2 - Visual Studio Code
main()
{
    BIGNUM *sign = BN_new();
    BIGNUM *newSign = BN_new();

    // Assign a value from a hex number string
    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A84");
    printBN("n = ", n);
    // Assign a value from a hex number string
    BN_hex2bn(&e, "010001");
    printBN("e = ", e);
    // Assign a value from a hex number string
    BN_hex2bn(&m, "49206f776520796f752024323030302e");
    printBN("m = ", m);
}
Terminal
tareq@TareqShannak:~/4320/Lab No.2$ gcc task4.c -o task4 -lcrypto
tareq@TareqShannak:~/4320/Lab No.2$ ./task4
n = DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5
e = 010001
m = 49206f776520796f752024323030302E
changedM = 49206f776520796f752024333030302E
d = 74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D
sign = 55A4E7F17F04CCE2766E1EB32ADDBA890BBE92A6FBE2D785ED6E73CCB35E4CB
newSign = BCC20FB7568E5D48E434C387C06A6025E90D29D848AF9C3EBAC0135D99305822
tareq@TareqShannak:~/4320/Lab No.2$
```

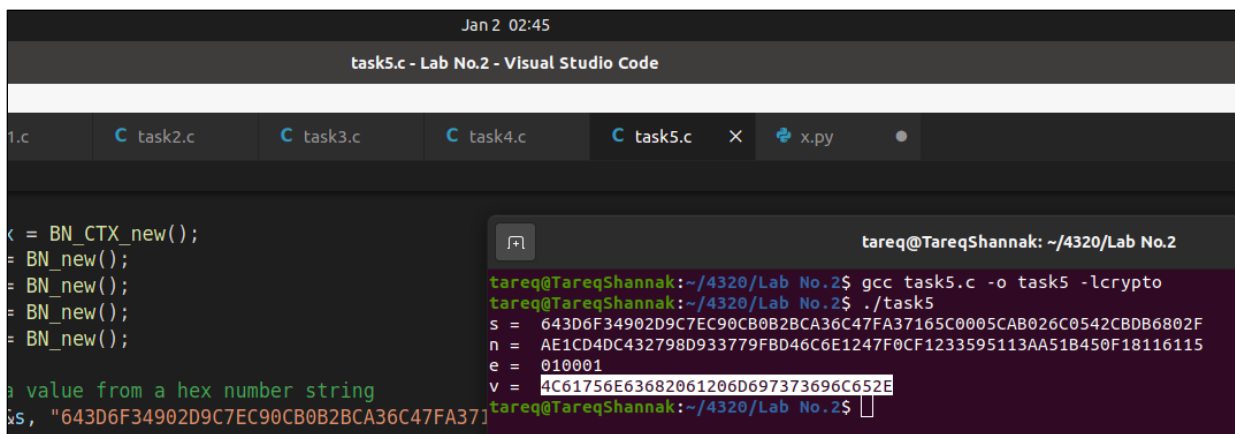
Figure 7 - Signing Two Messages

Task 5: Verifying a Signature

First, we will verify the sign as shown on Figure 8. As we know the verification algorithm as following:

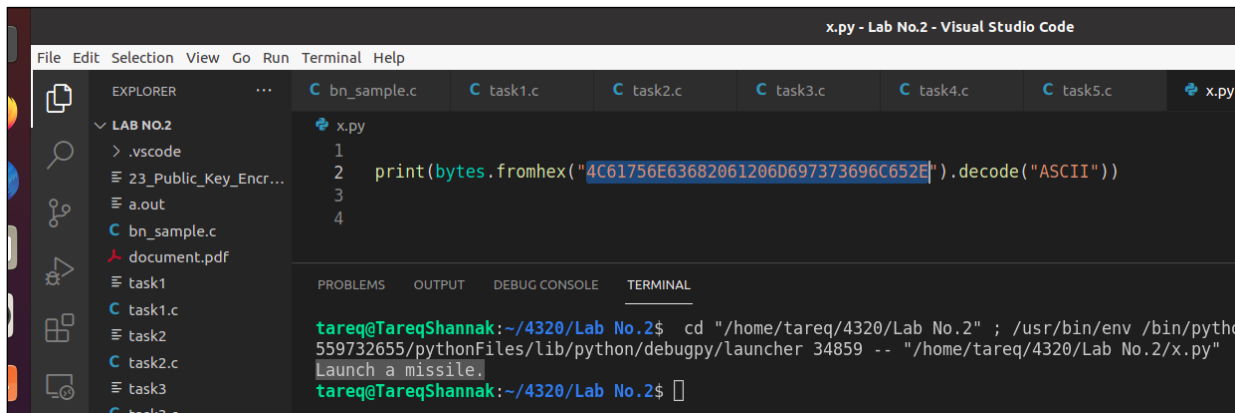
$$\text{Verification} = S^E \text{ mod } N$$

After we convert the obtained message to ASCII as shown on Figure 9, we can see that is equals to our message "Launch a missile.". Hence, the sign is verified and the receiver can trust the message in this case.



```
Jan 2 02:45
task5.c - Lab No.2 - Visual Studio Code
task2.c task3.c task4.c task5.c x.py
BN_CTX_new();
BN_new();
BN_new();
BN_new();
BN_new();
value from a hex number string
s, "643D6F34902D9C7EC90CB0B2BCA36C47FA37"
tareq@TareqShannak: ~/4320/Lab No.2
tareq@TareqShannak:~/4320/Lab No.2$ gcc task5.c -o task5 -lcrypto
tareq@TareqShannak:~/4320/Lab No.2$ ./task5
s = 643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F
n = AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115
e = 010001
v = 4C61756E63682061206D697373696C652E
tareq@TareqShannak:~/4320/Lab No.2$
```

Figure 8 - Verifying a Signature



```
x.py - Lab No.2 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
bn_sample.c task1.c task2.c task3.c task4.c task5.c x.py
LAB NO.2
.vscode
23_Public_Key_Encr...
a.out
bn_sample.c
document.pdf
task1
task1.c
task2
task2.c
task3
task3.c
x.py
1
2 print(bytes.fromhex("4C61756E63682061206D697373696C652E").decode("ASCII"))
3
4
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
tareq@TareqShannak:~/4320/Lab No.2$ cd "/home/tareq/4320/Lab No.2" ; /usr/bin/env /bin/python3 559732655/pythonFiles/lib/python/debugpy/launcher 34859 -- "/home/tareq/4320/Lab No.2/x.py"
Launch a missile.
tareq@TareqShannak:~/4320/Lab No.2$
```

Figure 9 - Getting the ASCII of hex message for Verifying

Now, we will change the sign to see the effect. Figure 10 shows the obtained message which its length taller than before and Figure 11 shows that it can't decode the message. Hence, we can't trust the received message because the obtained message isn't decodable and can't ensure that equals to the message we have "Launch a missile.". The code is in [Appendix5](#).

```

Jan 2 02:46
task5.c - Lab No.2 - Visual Studio Code
terminal Help
C bn_sample.c C task1.c C task2.c C task3.c C task4.c C task5.c X x.py
C task5.c > main()
13 {
14     BN_CTX *ctx = BN_CTX_new();
15     BIGNUM *n = BN_new();
16     BIGNUM *e = BN_new();
17     BIGNUM *s = BN_new();
18     BIGNUM *v = BN_new();
19
20     // Assign a value from a hex number string
21     BN_hex2bn(&s, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CDBD6803F");
22     printBN("s = ", s);
23     // Assign a value from a hex number string
24     BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA518450F18116115");
25     printBN("n = ", n);
26     // Assign a value from a hex number string
27     BN_hex2bn(&e, "010001");
28     printBN("e = ", e);
29
30
31
tareq@TareqShannak: ~/4320/Lab No.2
tareq@TareqShannak:~/4320/Lab No.2$ gcc task5.c -o task5 -lcrypto
tareq@TareqShannak:~/4320/Lab No.2$ ./task5
s = 643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CDBD6803F
n = AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA518450F18116115
e = 010001
v = 91471927C80DF1E42C154FB4638CE8BC726D3D66C83A4EB6B7BE0203B41AC294
tareq@TareqShannak:~/4320/Lab No.2$

```

Figure 10 - Verifying a signature after it corrupted

```

Jan 2 02:47
x.py - Lab No.2 - Visual Studio Code
Go Run Terminal Help
... C bn_sample.c C task1.c C task2.c C task3.c C task4.c C task5.c X x.py X
Encr...
1
2 print(bytes.fromhex("91471927C80DF1E42C154FB4638CE8BC726D3D66C83A4EB6B7BE0203B41AC294").decode("ASCII"))
3
4
5
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
tareq@TareqShannak:~/4320/Lab No.2$ cd "/home/tareq/4320/Lab No.2" ; /usr/bin/env /bin/python3 /home/tareq/.vscode
559732655/pythonFiles/lib/python/debugpy/launcher 45501 -- "/home/tareq/4320/Lab No.2/x.py"
Traceback (most recent call last):
  File "/home/tareq/4320/Lab No.2/x.py", line 2, in <module>
    print(bytes.fromhex("91471927C80DF1E42C154FB4638CE8BC726D3D66C83A4EB6B7BE0203B41AC294").decode("ASCII"))
UnicodeDecodeError: 'ascii' codec can't decode byte 0x91 in position 0: ordinal not in range(128)
tareq@TareqShannak:~/4320/Lab No.2$

```

Figure 11 - Invalid Message

Task 6: Manually Verifying an X.509 Certificate

First, we download a certificate from a real web server using the next command:

```
openssl s_client -connect www.example.org:443 -showcerts
```

There are two certificates that are copied and pasted in `c0.pem` and `c1.pem`. After that we extracted the public key (`e, n`) and the signature using the following commands and put them in the code which is in [Appendix 6](#). We can notice that the last part of the obtained message is the same of the signature without any changes, so we can verify the certificate on this message.

```
openssl x509 -in c1.pem -noout -modulus
```

```
openssl x509 -in c1.pem -text -noout
```

```
openssl x509 -in c0.pem -text -noout
```

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

```
openssl asn1parse -i -in c0.pem
```

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

```
sha256sum c0_body.bin
```

```
Jan 2 03:14
task6.c - Lab No.2 - Visual Studio Code
elp
ple.c task1.c task2.c task3.c task4.c task5.c task6.c x.py
> printBN(char *, BIGNUM *)
printf("%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();

// Assign a value from a hex number string
BN_hex2bn(&n, "C14BB3654770BCDD4F58DBEC9CEDC366E51F3");
printBN("n = ", n);
// Assign a value from a hex number string
BN_hex2bn(&e, "010001");
printBN("e = ", e);
// Assign a value from a hex number string
BN_hex2bn(&s, "a5543469fefb036b1a81d5a3679598f5c62a");
printBN("s = ", s);

//Calculate message
BN_mod_exp(m, s, e, n, ctx);
printBN("m = ", m);

return 0;

tareq@TareqShannak: ~/4320/Lab No.2
1587:d=2 hl=2 l= 0 prin: NULL
1589:d=1 hl=4 l= 257 prin: BIT STRING
tareq@TareqShannak:~/4320/Lab No.2$ openssl asn1parse -i -in c0.pem -strparse 4 -out c0_body.bin -noout
tareq@TareqShannak:~/4320/Lab No.2$ sha256sum c0_body.bin
8a9131f7b7cc5cdf4b7ed95c0e5d08832ee7437f5f292927356c1b65e0a8935 c0_body.bin
tareq@TareqShannak:~/4320/Lab No.2$ gcc task6.c -o task6 -lcrypto
tareq@TareqShannak:~/4320/Lab No.2$ ./task6
n = C14BB3654770BCDD4F58DBEC9CEDC366E51F311354AD4A66461F2C0AEC6407E52EDCDB90A20EDDFE3C4D09E9AA97A1D
8288E51156DB1E9F58C251E72C40D2ED292E156CBF1795FB3B87CA25037B9A52416610604F571349F0E83767830F7D3486
74C2251A6D0E9910ED57517426E27DC7CA622E131B7F238825536FC1345800884FF8B8EA75849227B96ADA289B15BCA07C
DFE951A8D5B0ED37E236B4824B62B5499AEC767D6E33EF5E3D6125E44F1BF71427D58840380B18101FAF9CA32BBB48E27872
7C52B74D4ABD697DEC364F9CACE5A256BC78178E490329AEFB494FA415B9CE25C19576D6B79A72BA2272013B5D03D40D321
300793EA99F5
e = 010001
s = A5543469FEFB036BF1A81D5A3679598F5C62A2639904D063783956440C35A2625C88AF7A10D440C14FAAD7E290395595
540E2C658440399AF3906A108D47DF4820858054390D160FC2A80AC1406A7F3A464F06B0F399E77610D2E54CF00800A
583CC1082224585020258AFB495FD143AAE662C9DC2AB7C8BF540CECA16135FD85AD39739FE7647BE1C0236FC2789453EA3
58B79C1FAF613D20831A256BF07188895D56D45DFF5FE1DE04E804A356326252084821C1EF60A2E4886422087CCFAB2FE51F
D30387D8C70A36082954480D127601E176635FAB93BA908F02E804CE3801F5BD3789FA7848AFD871169DA541CA6A148C76921
3363277354E8
m = 01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF300D060960648016503040201050004208A9131F7B7CC5CDF4B7ED95C0E5D08832EE7437F5F292927356C1
B65E0A8935
```

Figure 12 - Verifying an X.509 Certificate

Appendix

Appendix¹

```
#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 *one = BN_new();
    BIGNUM *decrementedP = BN_new();
    BIGNUM *decrementedQ = BN_new();
    BIGNUM *phi = BN_new();
    BIGNUM *d = BN_new();

    // Assign a value from a hex number string
    BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
    printBN("p = ", p);
    // Assign a value from a hex number string
    BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
    printBN("q = ", q);
    // Assign a value from a hex number string
    BN_hex2bn(&e, "0D88C3");
    printBN("e = ", e);
    // Assign a value from a hex number string
    BN_hex2bn(&one, "01");
    printBN("one = ", one);

    //Calculate decrementedP
    BN_sub(decrementedP, p, one);
    printBN("decrementedP = ", decrementedP);
    //Calculate Phi
    BN_sub(decrementedQ, q, one);
    printBN("decrementedQ = ", decrementedQ);
    //Calculate Phi
    BN_mul(phi, decrementedP, decrementedQ, ctx);
    printBN("phi = ", phi);
    //Calculate d
    BN_mod_inverse(d, e, phi, ctx);
    printBN("d = ", d);

    return 0;
}
```

Appendix²

```
#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 *n = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *m = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *c = BN_new();

    // Assign a value from a hex number string
    BN_hex2bn(&n, "DCBF3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    printBN("n = ", n);
    // Assign a value from a hex number string
    BN_hex2bn(&e, "010001");
    printBN("e = ", e);
    // Assign a value from a hex number string
    BN_hex2bn(&m, "4120746f702073656372657421");
    printBN("m = ", m);
    // Assign a value from a hex number string
    BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
    printBN("d = ", d);

    //Calculate encrypted
    BN_mod_exp(c, m, e, n, ctx);
    printBN("c = ", c);

    return 0;
}
```

Appendix³

```
#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 *n = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *m = BN_new();
```

```

BIGNUM *d = BN_new();
BIGNUM *c = BN_new();

// Assign a value from a hex number string
BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
printBN("n = ", n);
// Assign a value from a hex number string
BN_hex2bn(&e, "010001");
printBN("e = ", e);
// Assign a value from a hex number string
BN_hex2bn(&c, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBDFC7DCB67396567EA1E2493F");
printBN("c = ", c);
// Assign a value from a hex number string
BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
printBN("d = ", d);

//Calculate decrypted
BN_mod_exp(m, c, d, n, ctx);
printBN("m = ", m);

return 0;
}

```

Appendix⁴

```

#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 *n = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *m = BN_new();
    BIGNUM *changedM = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *sign = BN_new();
    BIGNUM *newSign = BN_new();

    // Assign a value from a hex number string
    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    printBN("n = ", n);
    // Assign a value from a hex number string
    BN_hex2bn(&e, "010001");
    printBN("e = ", e);
    // Assign a value from a hex number string
    BN_hex2bn(&m, "49206f776520796f752024323030302e");
    printBN("m = ", m);
    // Assign a value from a hex number string
    BN_hex2bn(&changedM, "49206f776520796f752024333030302e");
    printBN("changedM = ", changedM);
    // Assign a value from a hex number string

```

```

BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
printBN("d = ", d);

//Calculate sign
BN_mod_exp(sign, m, d, n, ctx);
printBN("sign = ", sign);

//Calculate newSign
BN_mod_exp(newSign, changedM, d, n, ctx);
printBN("newSign = ", newSign);

return 0;
}

```

Appendix⁵

```

#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 *n = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *s = BN_new();
    BIGNUM *v = BN_new();

    // Assign a value from a hex number string
    BN_hex2bn(&s, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");
    printBN("s = ", s);
    // Assign a value from a hex number string
    BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");
    printBN("n = ", n);
    // Assign a value from a hex number string
    BN_hex2bn(&e, "010001");
    printBN("e = ", e);

    //Calculate Verification
    BN_mod_exp(v, s, e, n, ctx);
    printBN("v = ", v);

    return 0;
}

```

Appendix⁶

```

#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 *n = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *s = BN_new();
    BIGNUM *m = BN_new();

    // Assign a value from a hex number string
    BN_hex2bn(&n,
"C14BB3654770BCDD4F58DBEC9CEDC366E51F311354AD4A66461F2C0AEC6407E52EDCDBC90A20EDDFE3C4D09E9AA97A1D8288
E51156DB1E9F58C251E72C340D2ED292E156CBF1795FB3BB87CA25037B9A52416610604F571349F0E8376783DFE7D34B674C225
1A6DF0E9910ED57517426E27DC7CA622E131B7F238825536FC13458008B84FFF8BEA75849227B96ADA2889B15BCA07CDFE951A8
D5B0ED37E236B4824B62B5499AECC767D6E33EF5E3D6125E44F1BF71427D58840380B18101FAF9CA32BBB48E278727C52B74D4
A8D697DEC364F9CACE53A256BC78178E490329AEFB494FA415B9CEF25C19576D6B79A72BA2272013B5D03D40D321300793EA99
F5");
    printBN("n = ", n);
    // Assign a value from a hex number string
    BN_hex2bn(&e, "010001");
    printBN("e = ", e);
    // Assign a value from a hex number string
    BN_hex2bn(&s,
"a5543469fefb036bf1a81d5a3679598f5c62a2639904d063783956440c35a2625c88af7a10d44dc14faad7e2993955955adf2c6c584403
99af3906a108d47fdf482895b8654390d160ec2a86a8c14d6a7f3a464f06eb8f399e7761db2e54cff0d8d0a583cc108222450502d6250afb
495fd143aae662c9dc2ab7c8bf546ceca16135fd85ad39739fe7647be1c0236fca27b9453ea358b70c1faf613d2d831a256bf071b8895d56
d45dff5fe1de04eb04a356326252084821c1ef60a28e48b6422007ccfab2ef51fd303b7d8c7da36d82954480d1276d1e176635fab93ba90
8f02e804ce3801f5bd37b9fa784bafd871169da541ca6a148c769213363277354e8");
    printBN("s = ", s);

    //Calculate message
    BN_mod_exp(m, s, e, n, ctx);
    printBN("m = ", m);

    return 0;
}

```