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

List of Figures

Task 1: Deriving the Private Key

From code in Appendix[1], we can derive the private key as shown on Figure 1 after computing ϕ(n) where:

$$\emptyset(n) = \emptyset(p * q) = (p-1)(q-1), where\ p\ \&\ q\ are\ prime\ numbers.$$

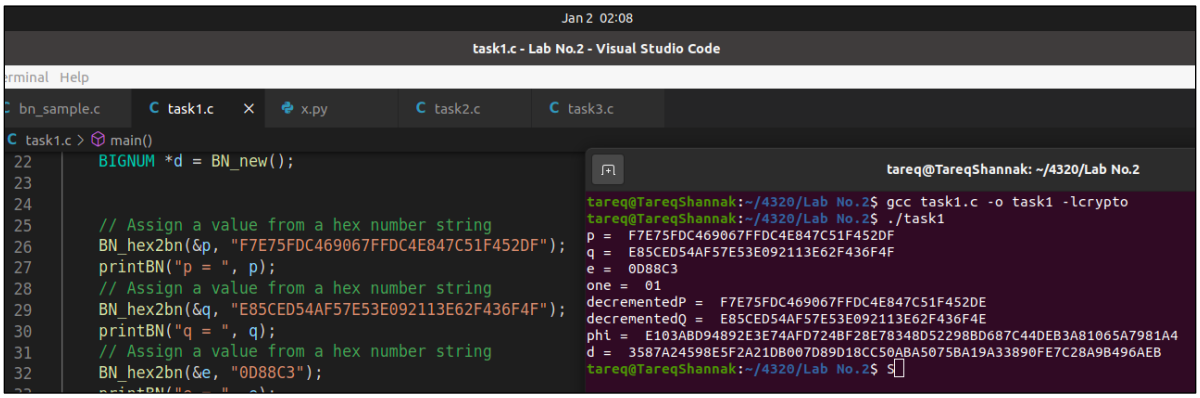

Figure 1 - Deriving the Private Key

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 Appendix[2].

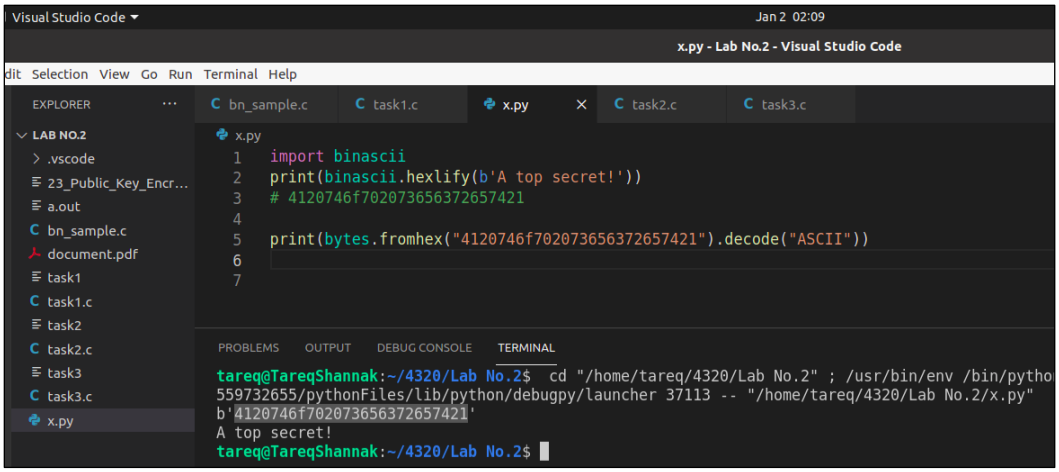

Figure 2 – Getting the hex of a Message

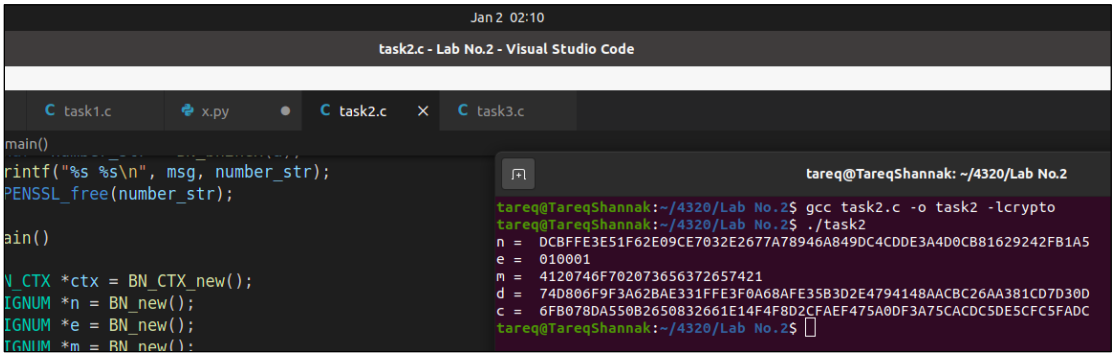

Figure 3 - Encrypting a Message

Task 3: Decrypting a Message

In this task, we decrypted a cipher text using the code in Appendix[3] 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”.

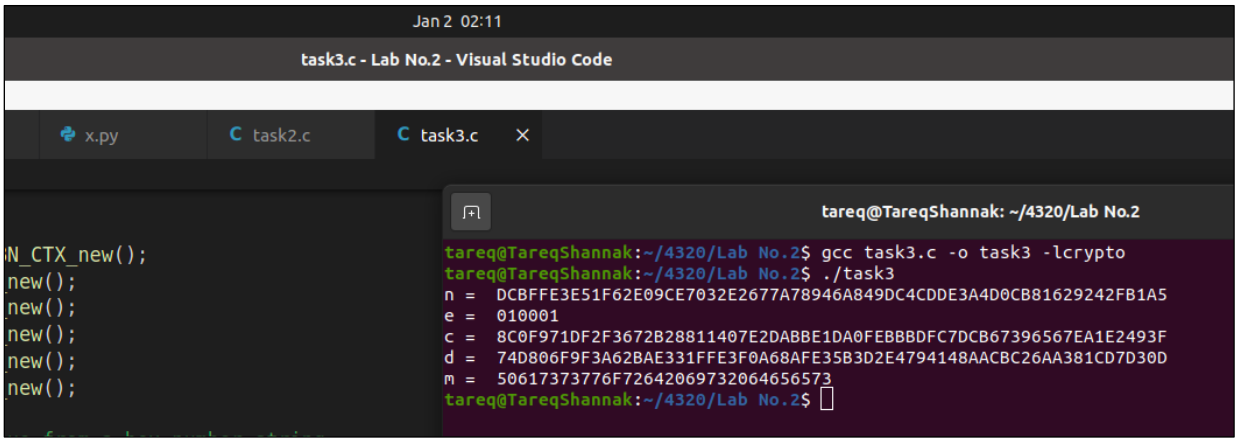

Figure 4 - Decrypting a Message

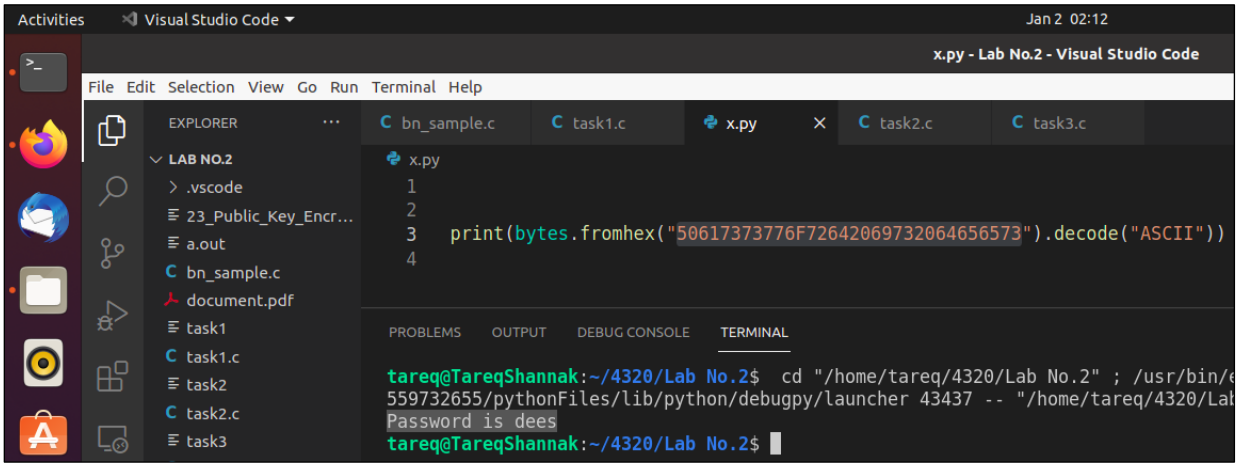

Figure 5 - Getting ASCII from a hex Message

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:

$$Sign = M^D mod\ N$$

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

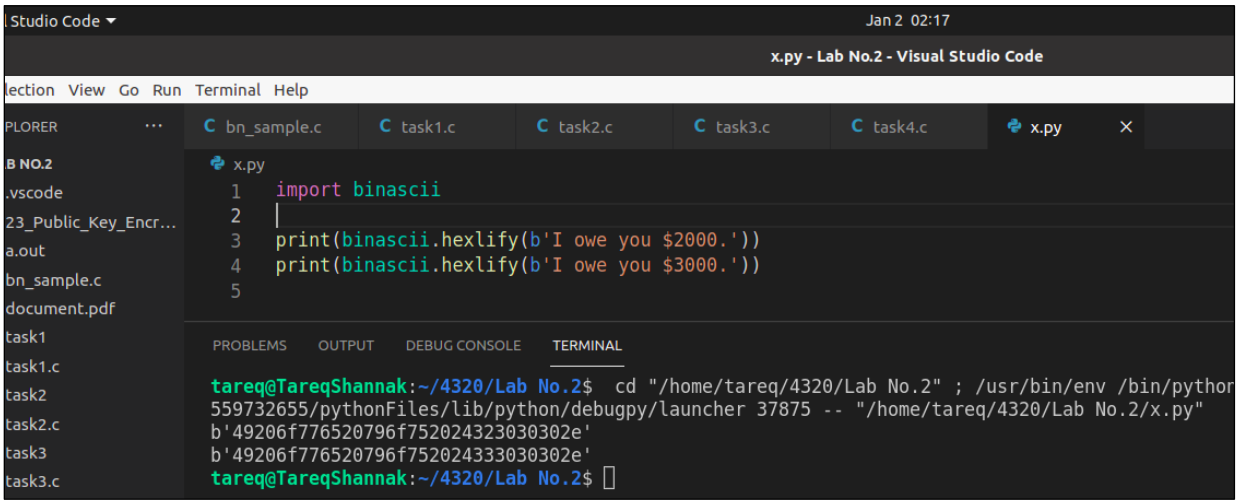

Figure 6 - Getting the hex of messages for signing

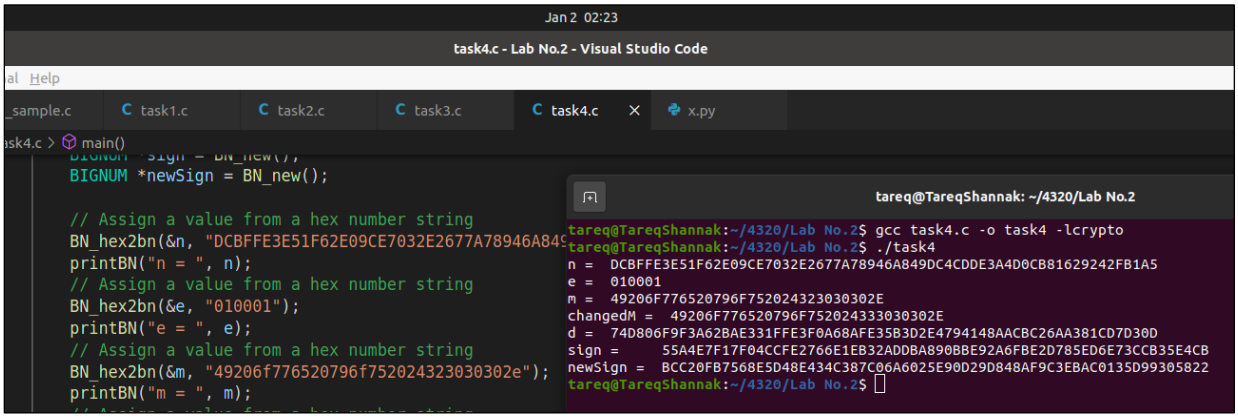

Figure 7 - Signing Two Messages

Task 5: Verifying a Signature

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

$$Verification = S^E mod\ N$$

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.

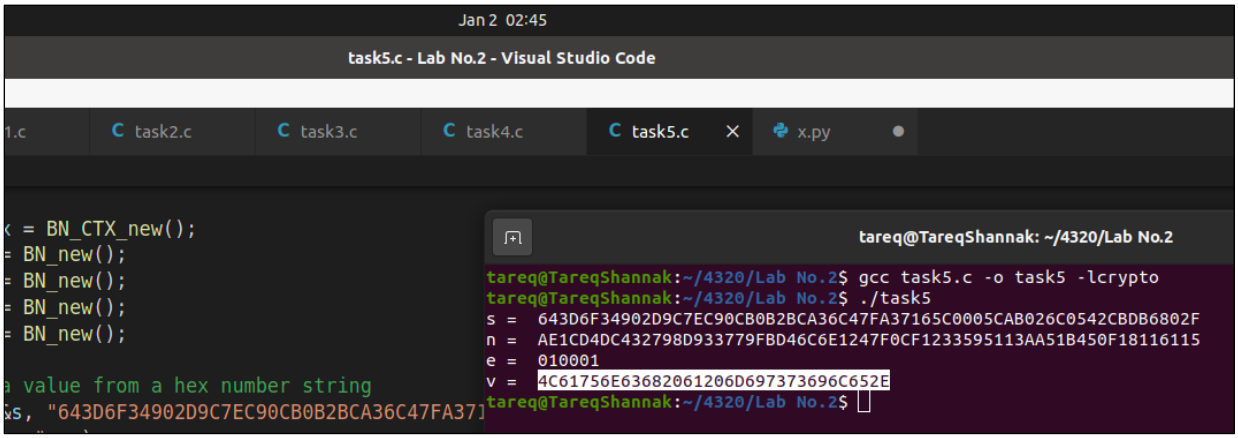

Figure 8 - Verifying a Signature

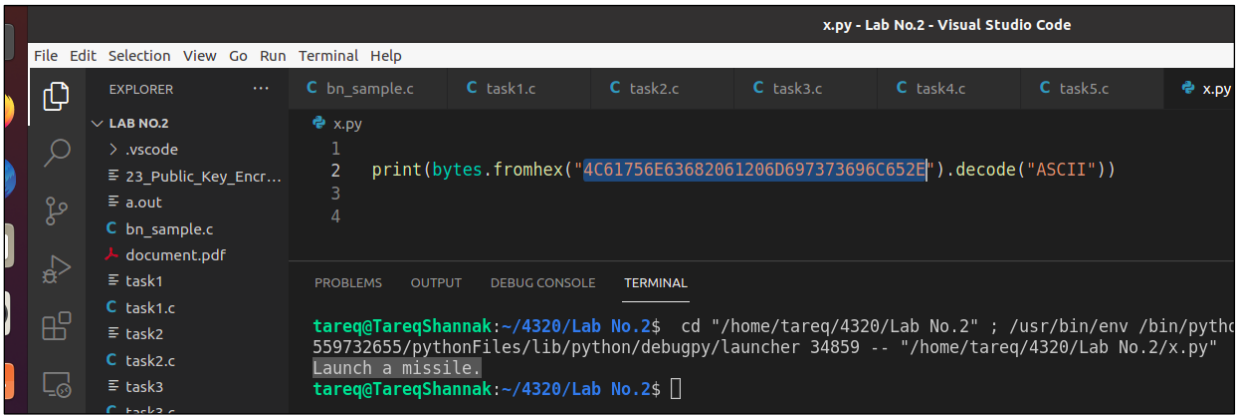

Figure 9 - Getting the ASCII of hex message for Verifying

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 Appendix[5].

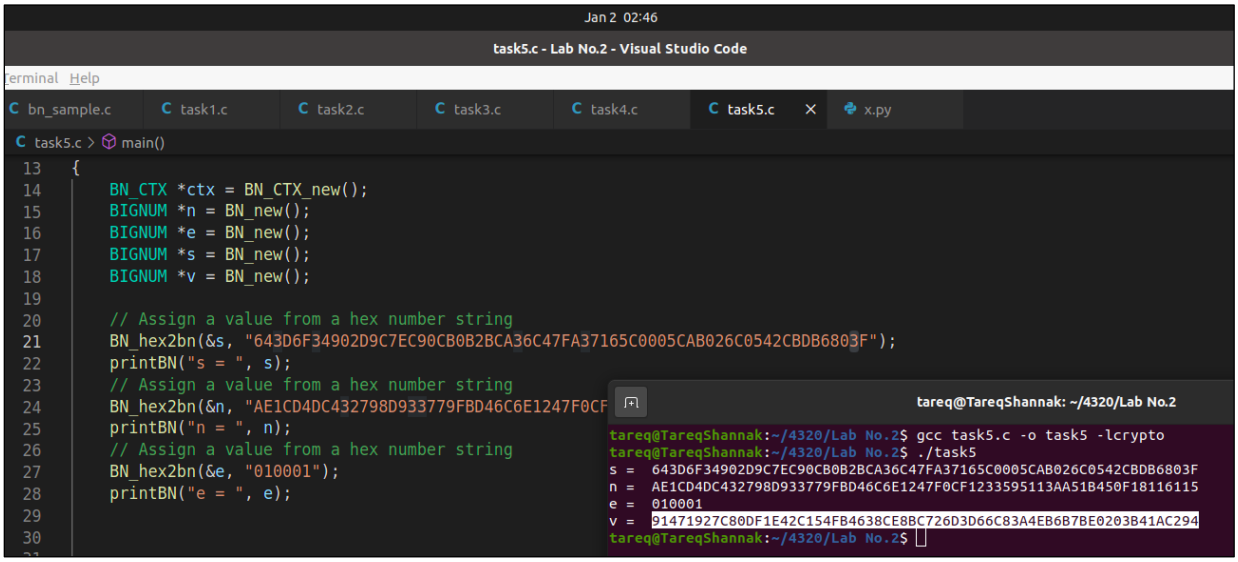

Figure 10 - Verifying a signature after it corrupted

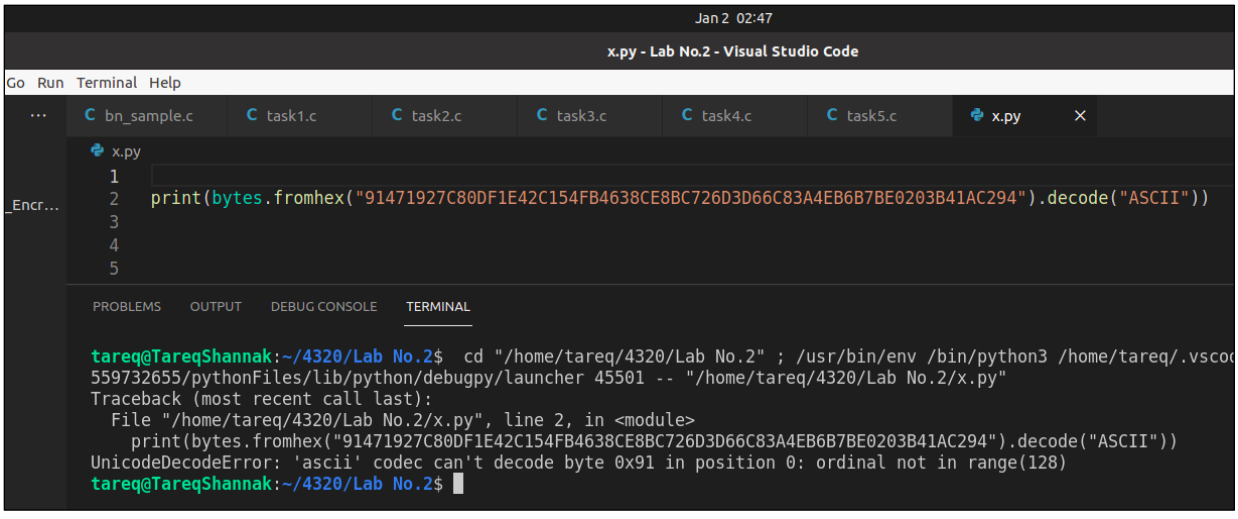

Figure 11 - Invalid Message

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 verified the certificate on this message.

$$openssl\ x509\ -in\ c1.pem\ -noout - modulus$$

$$openssl\ x509\ -in\ c1.pem\ -text - noout$$

$$openssl\ x509\ -in\ c0.pem\ -text - noout$$

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

$$openssl\ asn1parse\ -i\ -in\ c0.pem$$

$$openssl\ asn1parse\ -i\ -in\ c0.pem\ -strparse\ 4\ -out\ c0\_body.bin - noout$$

$$sha256sum\ c0\_body.bin$$

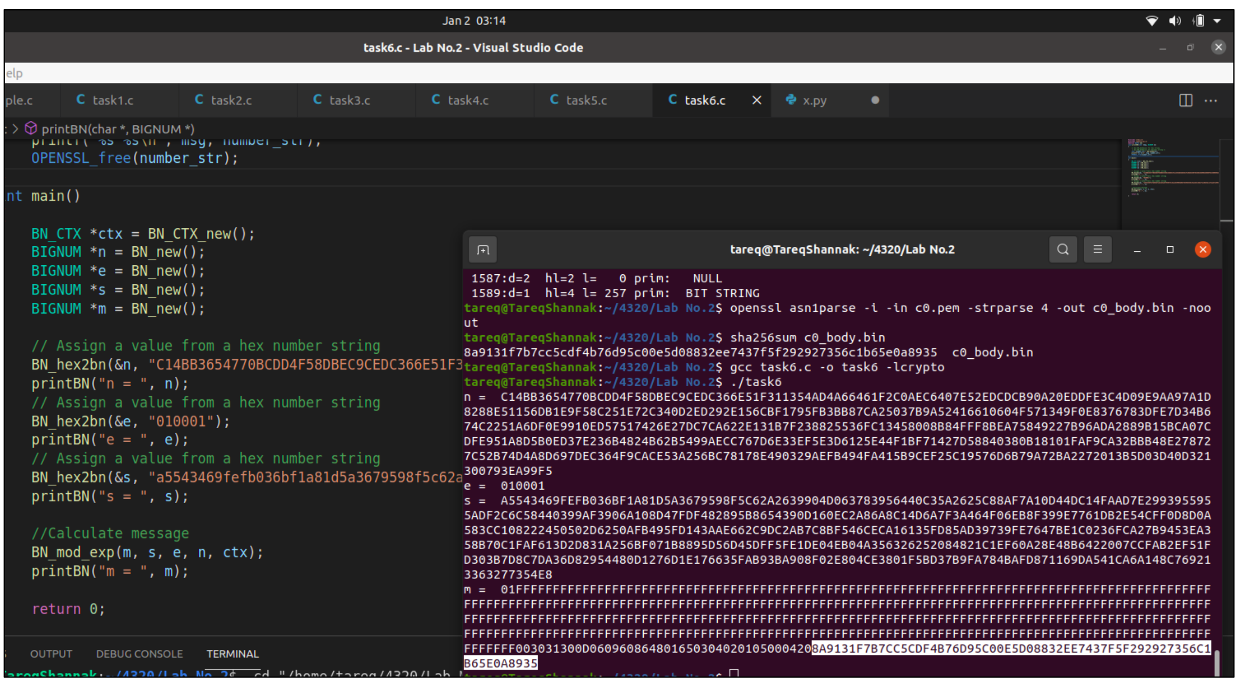

Figure 12 - Verifying an X.509 Certificate

Appendix

Appendix[1]

```
#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, "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, "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, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");
    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;
}
```

Appendix[4]

```
#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,
"C14BB3654770BCDD4F58DBEC9CEDC366E51F311354AD4A66461F2C0AEC6407E52EDCDCB90A20EDDFE3C4D09E9AA97A1D8288
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;
}
```