Lab-10

-By Vikramadithya Ivaturi

**Task 1:**

**Code:**

#include <stdio.h>

#include <openssl/bn.h>

#define NBITS 512

void printBN(char \*msg, BIGNUM \* a)

{

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, \*q, \*n, \*phi, \*e, \*d, \*m, \*c, \*res;

BIGNUM \*new\_m, \*p\_minus\_one, \*q\_minus\_one;

p = BN\_new(); q = BN\_new(); n = BN\_new(); e = BN\_new();

d = BN\_new(); m = BN\_new(); c = BN\_new();

res = BN\_new(); phi = BN\_new(); new\_m = BN\_new();

p\_minus\_one = BN\_new(); q\_minus\_one = BN\_new();

// Set the public key exponent e

//BN\_dec2bn(&e, "65537");

BN\_hex2bn(&e, "0D88C3");

// Generate random p and q.

//BN\_generate\_prime\_ex(p, NBITS, 1, NULL, NULL, NULL);

//BN\_generate\_prime\_ex(q, NBITS, 1, NULL, NULL, NULL);

//p = 0xF7E75FDC469067FFDC4E847C51F452DF;

//q = 0xE85CED54AF57E53E092113E62F436F4F;

BN\_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");

BN\_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");

BN\_sub(p\_minus\_one, p, BN\_value\_one()); // Compute p-1

BN\_sub(q\_minus\_one, q, BN\_value\_one()); // Compute q-1

BN\_mul(n, p, q, ctx); // Compute n=pq

BN\_mul(phi, p\_minus\_one, q\_minus\_one, ctx); // Compute (\*@$\phi(n)$@\*)

// Check whether e and (\*@$\phi(n)$@\*) are relatively prime.

BN\_gcd(res, phi, e, ctx);

if (!BN\_is\_one(res)) {

printf("They are not relatively prime, try it again.");

exit(0); // They are not relatively prime, try it again.

}

// Compute the private key exponent d, s.t. ed mod phi(n) = 1

BN\_mod\_inverse(d, e, phi, ctx);

printBN("Private key:", d);

// Encryption: calculate m^e mod n

//BN\_hex2bn(&m, "546869732069732061207365637265742e");

//BN\_mod\_exp(c, m, e, n, ctx);

//printBN("Encryption result:", c);

// Decryption: calculate c^d mod n

//BN\_mod\_exp(new\_m, c, d, n, ctx);

//printBN("Decryption result:", new\_m);

// Clear the sensitive data from the memory

BN\_clear\_free(p); BN\_clear\_free(q); BN\_clear\_free(d);

BN\_clear\_free(phi); BN\_clear\_free(m); BN\_clear\_free(new\_m);

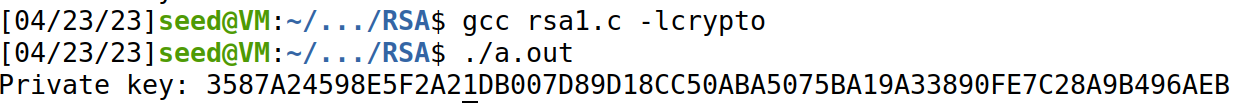
BN\_clear\_free(c); BN\_clear\_free(res);

BN\_clear\_free(p\_minus\_one); BN\_clear\_free(q\_minus\_one);

return 0;

}

**Output:**



**Task-2:**

**Code:**

#include <stdio.h>

#include <openssl/bn.h>

#define NBITS 512

void printBN(char \*msg, BIGNUM \* a)

{

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, \*q, \*n, \*phi, \*e, \*d, \*m, \*c, \*res;

BIGNUM \*new\_m, \*p\_minus\_one, \*q\_minus\_one;

p = BN\_new(); q = BN\_new(); n = BN\_new(); e = BN\_new();

d = BN\_new(); m = BN\_new(); c = BN\_new();

res = BN\_new(); phi = BN\_new(); new\_m = BN\_new();

p\_minus\_one = BN\_new(); q\_minus\_one = BN\_new();

// Set the public key exponent e

BN\_dec2bn(&e, "65537");

BN\_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");

BN\_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

// Encryption: calculate m^e mod n

BN\_hex2bn(&m, "4120746f7020736563726574210a");

BN\_mod\_exp(c, m, e, n, ctx);

printBN("Encryption result:", c);

// Decryption: calculate c^d mod n

BN\_mod\_exp(new\_m, c, d, n, ctx);

printBN("Decryption result:", new\_m);

// Clear the sensitive data from the memory

BN\_clear\_free(p); BN\_clear\_free(q); BN\_clear\_free(d);

BN\_clear\_free(phi); BN\_clear\_free(m); BN\_clear\_free(new\_m);

BN\_clear\_free(c); BN\_clear\_free(res);

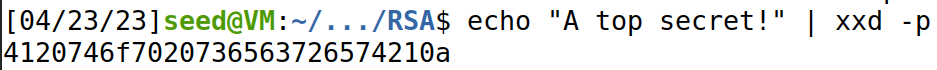
BN\_clear\_free(p\_minus\_one); BN\_clear\_free(q\_minus\_one);

return 0;

}

**Output:**

First we convert the given message into a hexadecimal format.



**(it is noticeable that there is ‘0a’ in the end of HEX string, that is a ‘new line’ character in HEX).**

Now, we compile and execute the program file.

Text

Description automatically generated

Since we got the plaintext back, we were successful in encrypting the message.

**Task-3**

**Code:**

#include <stdio.h>

#include <openssl/bn.h>

#define NBITS 512

void printBN(char \*msg, BIGNUM \* a)

{

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, \*q, \*n, \*phi, \*e, \*d, \*m, \*c, \*res;

BIGNUM \*new\_m, \*p\_minus\_one, \*q\_minus\_one;

p = BN\_new(); q = BN\_new(); n = BN\_new(); e = BN\_new();

d = BN\_new(); m = BN\_new(); c = BN\_new();

res = BN\_new(); phi = BN\_new(); new\_m = BN\_new();

p\_minus\_one = BN\_new(); q\_minus\_one = BN\_new();

BN\_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");

BN\_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

BN\_hex2bn(&c, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");

// Decryption: calculate c^d mod n

BN\_mod\_exp(new\_m, c, d, n, ctx);

printBN("Decryption result:", new\_m);

// Clear the sensitive data from the memory

BN\_clear\_free(p); BN\_clear\_free(q); BN\_clear\_free(d);

BN\_clear\_free(phi); BN\_clear\_free(m); BN\_clear\_free(new\_m);

BN\_clear\_free(c); BN\_clear\_free(res);

BN\_clear\_free(p\_minus\_one); BN\_clear\_free(q\_minus\_one);

return 0;

}

**Output:**

Text

Description automatically generated

We have successfully decrypted the message.

**Task-4:**

**Code:**

#include <stdio.h>

#include <openssl/bn.h>

#define NBITS 512

void printBN(char \*msg, BIGNUM \* a)

{

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, \*q, \*n, \*phi, \*e, \*d, \*m, \*c, \*res;

BIGNUM \*new\_m, \*p\_minus\_one, \*q\_minus\_one;

p = BN\_new(); q = BN\_new(); n = BN\_new(); e = BN\_new();

d = BN\_new(); m = BN\_new(); c = BN\_new();

res = BN\_new(); phi = BN\_new(); new\_m = BN\_new();

p\_minus\_one = BN\_new(); q\_minus\_one = BN\_new();

// Set the public key exponent e

BN\_dec2bn(&e, "65537");

BN\_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");

BN\_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

BN\_hex2bn(&c, "49206F776520796F75202432303030");

BN\_mod\_exp(new\_m, c, d, n, ctx);

printBN("Signed Message:", new\_m);

// Clear the sensitive data from the memory

BN\_clear\_free(p); BN\_clear\_free(q); BN\_clear\_free(d);

BN\_clear\_free(phi); BN\_clear\_free(m); BN\_clear\_free(new\_m);

BN\_clear\_free(c); BN\_clear\_free(res);

BN\_clear\_free(p\_minus\_one); BN\_clear\_free(q\_minus\_one);

return 0;

}

**Output:**

First thing we need to recognize is ‘49206F776520796F75202432303030’ in the program. This is the HEX representation of the message ‘I owe you $2000’.

Now, we compile the code and run the executable.

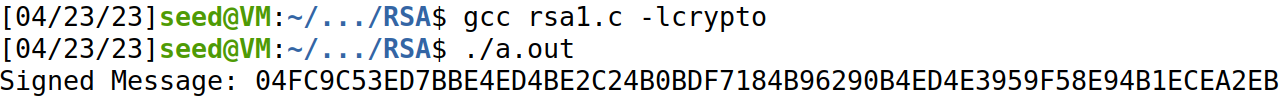
Text

Description automatically generated

So, we have signed the message.

Now, we change the message to ‘I owe you $3000’ which is ‘49206F776520796F75202433303030’ in HEX.

We compile and run the executable.



As we can see the signed messages are drastically different.

**Task-5:**

**Code:**

#include <stdio.h>

#include <openssl/bn.h>

#define NBITS 512

void printBN(char \*msg, BIGNUM \* a)

{

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, \*q, \*n, \*phi, \*e, \*d, \*m, \*c, \*res;

BIGNUM \*new\_m, \*p\_minus\_one, \*q\_minus\_one;

p = BN\_new(); q = BN\_new(); n = BN\_new(); e = BN\_new();

d = BN\_new(); m = BN\_new(); c = BN\_new();

res = BN\_new(); phi = BN\_new(); new\_m = BN\_new();

p\_minus\_one = BN\_new(); q\_minus\_one = BN\_new();

BN\_dec2bn(&e, "65537");

BN\_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");

BN\_hex2bn(&new\_m, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");

printBN("Signed Message:", new\_m);

BN\_mod\_exp(m, new\_m, e, n, ctx);

printBN("Sign Verified:", m);

// Clear the sensitive data from the memory

BN\_clear\_free(p); BN\_clear\_free(q); BN\_clear\_free(d);

BN\_clear\_free(phi); BN\_clear\_free(m); BN\_clear\_free(new\_m);

BN\_clear\_free(c); BN\_clear\_free(res);

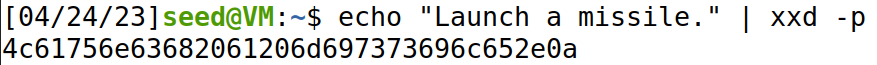
BN\_clear\_free(p\_minus\_one); BN\_clear\_free(q\_minus\_one);

return 0;

}

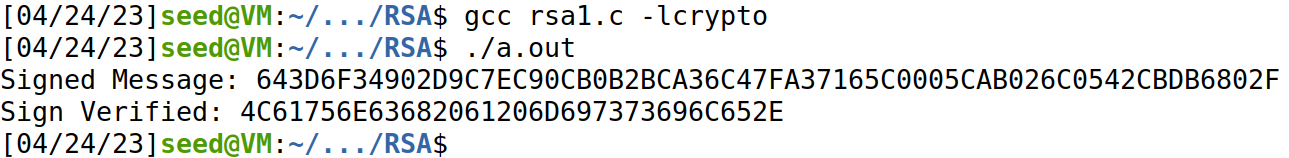
**Output:**

First we convert the input message to hex for verification.

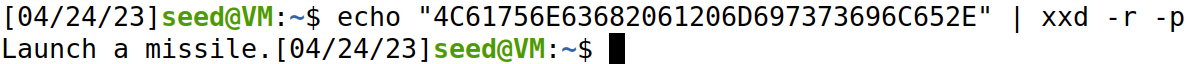
****

**(it is noticeable that there is ‘0a’ in the end of HEX string, that is a ‘new line’ character in HEX).**

Now, we compile the program and run the compiled file.

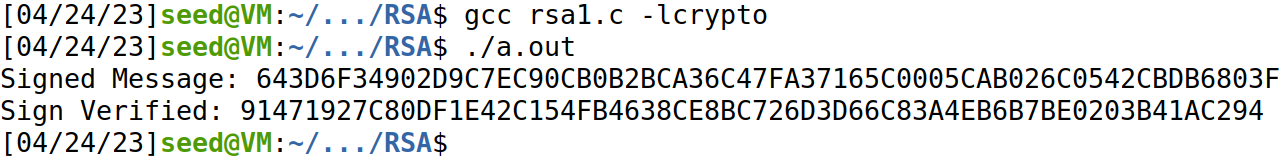


Now, we convert the output HEX string into ASCII to verify if we got back the original message.



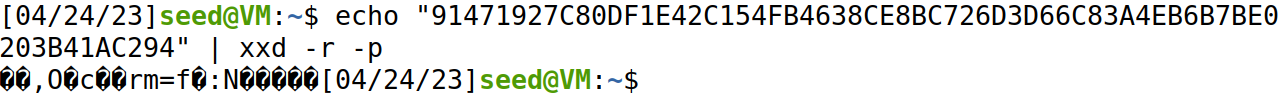
Since we got back the original message, we have verified that the message has been signed by Alice.

Now, if we try to change the value of message from ‘2F’ to ‘3F’, we will get the below output.



We can clearly observe that this HEX string is not equal to the original HEX string.

Below is the output when we convert it into ASCII format.



It is not in human readable format. Hence, we can understand that the message has been altered.

**Task-6:**

**Code:**

#include <stdio.h>

#include <openssl/bn.h>

#define NBITS 512

void printBN(char \*msg, BIGNUM \* a)

{

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, \*q, \*n, \*phi, \*e, \*d, \*m, \*c, \*res;

BIGNUM \*new\_m, \*p\_minus\_one, \*q\_minus\_one;

p = BN\_new(); q = BN\_new(); n = BN\_new(); e = BN\_new();

d = BN\_new(); m = BN\_new(); c = BN\_new();

res = BN\_new(); phi = BN\_new(); new\_m = BN\_new();

p\_minus\_one = BN\_new(); q\_minus\_one = BN\_new();

BN\_dec2bn(&e, "65537");

BN\_hex2bn(&n

BN\_hex2bn(&new\_m,"");

printBN("Sign of Cert:", new\_m);

BN\_mod\_exp(m, new\_m, e, n, ctx);

printBN("Message after verification:", m);

// Clear the sensitive data from the memory

BN\_clear\_free(p); BN\_clear\_free(q); BN\_clear\_free(d);

BN\_clear\_free(phi); BN\_clear\_free(m); BN\_clear\_free(new\_m);

BN\_clear\_free(c); BN\_clear\_free(res);

BN\_clear\_free(p\_minus\_one); BN\_clear\_free(q\_minus\_one);

return 0;

}

**Output:**

Step-1:

Getting the certificates and placing them in two files.

Text

Description automatically generated with low confidence

Text, letter

Description automatically generated

Step-2:

Getting the value of ‘n’ from the certificate. From the command given in the assignment paper.

A screenshot of a computer

Description automatically generated with medium confidence

Extracting the value of exponent ‘e’ form the certificate.

**Graphical user interface, text

Description automatically generated**

**Text

Description automatically generated**

Step-3:

Getting the signature from the server certificate.

Text

Description automatically generated

Text

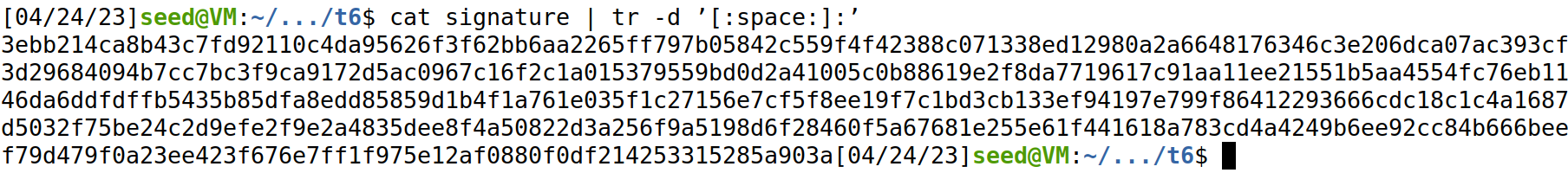
Description automatically generated

Now, pasting this block into a file.

Icon

Description automatically generated with medium confidence

Now, we are removing the ‘:’ and the spaces so that we get a pure HEX string.



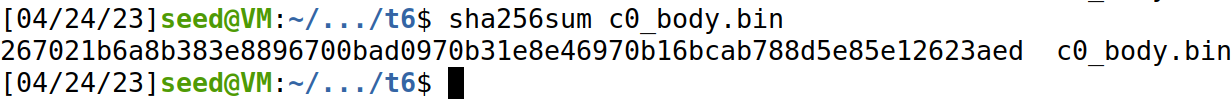
Step-4:

Extracting the body of the signature

Text

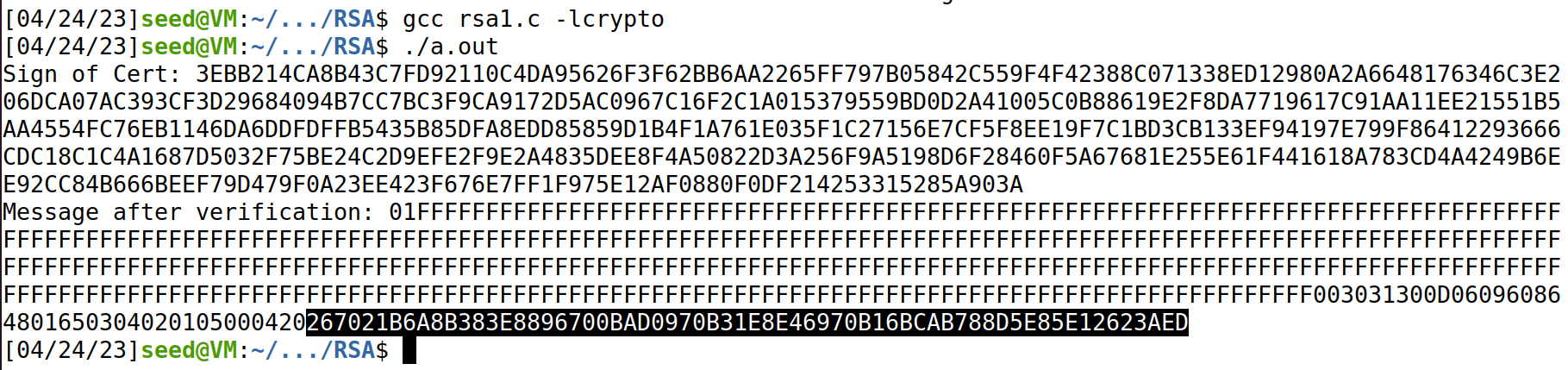
Description automatically generated with low confidence

Now, we calculate the hash value of the body file which is considered as the signed message.



Step-5:

With the values of ‘n’, ‘e’ and ‘s’ that we got from the previous steps, we compile and execute the RSA program.



We can see in the above picture that the highlighted part is equal to the hash value of the signature body.

**267021b6a8b383e8896700bad0970b31e8e46970b16bcab788d5e85e12623aed**

**267021B6A8B383E8896700BAD0970B31E8E46970B16BCAB788D5E85E12623AED**

Therefore, we have successfully verified the signature of the certificate.