**ACS 545 Cryptography and Network Security**

**Lab 10: RSA Public-Key Encryption and Signature Lab**

**Name:** Vijay Anand Pandian

**Task 1 – Deriving the Private Key**

**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, p and q value

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

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

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

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)) {

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);

// 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;

}

**Implementation and Output:**

**Text

Description automatically generated**

**Explanation and Observation:**If we run the above C program we will be able to derive the “Private key” for the given three prime numbers p,q and e values.

**Task 2 - Encrypting a Message**

**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 values for n, d and e

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

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

BN\_hex2bn(&e, "010001");

BN\_hex2bn(&m, "4120746f702073656372657421"); //m is the hex value of the message "A top secret!"

// Encryption: calculate m^e mod n

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;

}

**Implementation and Output:Text

Description automatically generated**

**Explanation and Observation:**

The above C program will encrypt a message using a given public key (e, n). The message to be encrypted is "A top secret!" which needs to be converted into a hex string and then converted into a BIGNUM using the hex-to-bn API BNhex2bn().

**Task 3 – Decrypting a Message:**

**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();

//Sets the value of n,d,c

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;

}

**Implementation and Output:**

**Text

Description automatically generated**

**Explanation and Observation:**

The above C program will decrypt a given ciphertext using the same public/private keys as used in Task 2. The ciphertext is in hexadecimal format and needs to be converted back to a plain ASCII string after decryption using python.

**Task 4 – Signing a Message:**

**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, \*s, \*m1;

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(); s = BN\_new(); m1 = 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(&e, "010001");

// Message: Hex to bn "I owe you $2000"

BN\_hex2bn(&m, "49206f776520796f75202432303030");

// Signature: calculate c^d mod n

BN\_mod\_exp(s, m, d, n, ctx);

printBN("Signature of Message 1:", s);

// Message: Hex to bn "I owe you $3000"

BN\_hex2bn(&m1, "49206f776520796f75202433303030");

// Signature: calculate c^d mod n

BN\_mod\_exp(s, m1, d, n, ctx);

printBN("Signature of Message 2:", s);

// 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;

}

**Implementation and Output:**

Graphical user interface, text, application

Description automatically generated

**Explanation and Observation:**

The above C program will generate a digital signature for a given message using the same public/private keys as used in Task 2. The message is "I owe you $2000", and a slight modification is made to the message by changing $2000 to $3000. Both the original message and the modified message are signed, and the signatures are compared.

**Task 5 – Verifying a Signature**

**Code:**

#include <stdio.h>

#include <openssl/bn.h>

#define NBITS 256

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, \*ch, \*res, \*s1, \*m1, \*s2;

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

p = BN\_new(); q = BN\_new(); n = BN\_new(); e = BN\_new(); s1 = BN\_new();

d = BN\_new(); m = BN\_new(); ch = BN\_new(); s2 = BN\_new(); m1 = 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, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");

BN\_hex2bn(&e, "010001");

BN\_hex2bn(&s1, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");

BN\_hex2bn(&s2, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F");

// Message: Hex to bn "Launch a missile."

BN\_hex2bn(&m, "4c61756e63682061206d697373696c65");

printBN("Message:", m);

// Signature: calculate c^d mod n

BN\_mod\_exp(ch, s1, e, n, ctx);

printBN("Verify of Message with original signature:", ch);

// Signature: calculate c^d mod n

BN\_mod\_exp(ch, s2, e, n, ctx);

printBN("Verify of Message with fake signature:", ch);

// 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);

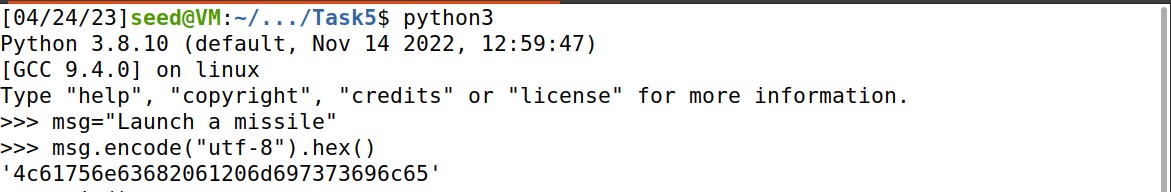
BN\_clear\_free(ch); BN\_clear\_free(res);

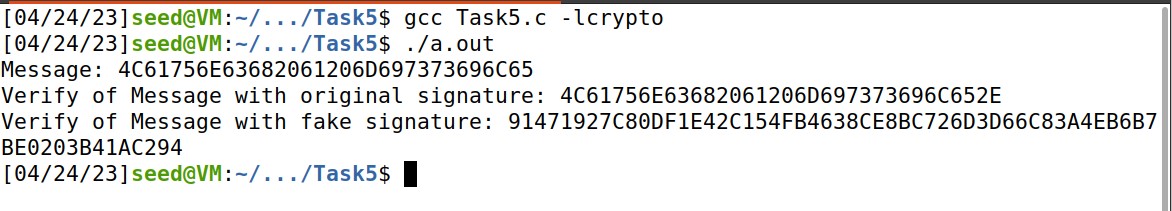
BN\_clear\_free(p\_minus\_one); BN\_clear\_free(q\_minus\_one);

return 0;

}

**Implementation and Output:**

****

****

**Explanation and Observation:**

The above C program will verify the signature of message M is "Launch a missile." and it is signed using the original private key, and the resulting signature is S. The public key (e, n) is also given in hexadecimal format. The c program will verify whether the signature is valid or not.

**Task 6 – Manually Verifying an X.509 Certificate**

**Step1: Download a certificate from a real web server**

Downloaded a real X.509 certificate from a web server, using the command "$ openssl s\_client -connect [www.paypal.com](http://www.paypal.com) -showcerts". I got two certificates back, one for the server and the other for the intermediate CA that signed the server's certificate**.**

**Text

Description automatically generated**

**Text

Description automatically generated**

**Text

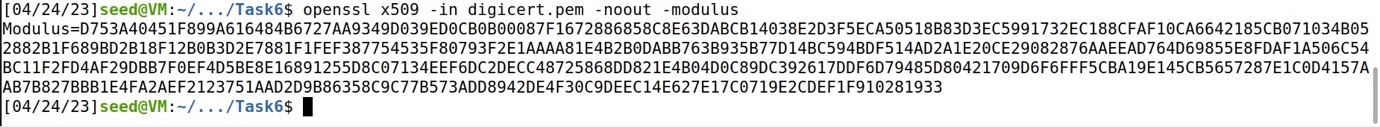
Description automatically generated**

**Step 2 – Extract the public key(e,n) from the issuer’s certificate.**

Extracted the public key (e,n) from the issuer's certificate, which is the intermediate CA's certificate. To get n, I used the command "$ openssl x509 -in digicert.pem -noout -modulus". To get e, I printed out all the fields using "$ openssl x509 -in digicert.pem -text -noout", and found the value of e in the output.

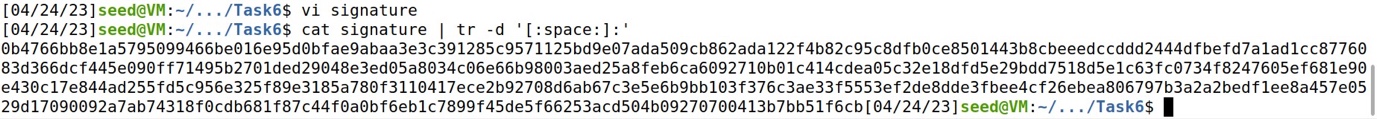
Text

Description automatically generated



**Step - 3: Extract the signature from the server’s certificate**.

Extracted the signature from the server's certificate using "$ openssl x509 -in paypal.pem -text -noout". Since the signature was displayed in hexadecimal format with spaces and colons, I removed the spaces and colons to obtain the signature as a hex-string.



**Step - 4: Extract the body of the server’s certificate.**

Text

Description automatically generated

**Step 5: Verify the signature.**

**Code:**

#include <stdio.h>

#include <openssl/bn.h>

#include <string.h>

#define NBITS 256

#define SHA\_LENGTH 64

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, \*ch, \*res, \*s;

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

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

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

BN\_hex2bn(&e, "10001");

BN\_hex2bn(&s, "");

BN\_hex2bn(&m, "710e017b0e642997a3115fb9c4f0a1f8e5378a8578e6e05a06b43b4d21b7fc09");

printBN("Message:", m);

// Signature: calculate c^d mod n

BN\_mod\_exp(ch, s, e, n, ctx);

char \*result = BN\_bn2hex(ch);

char substr[65];

strncpy(substr, result + strlen(result) - SHA\_LENGTH, SHA\_LENGTH);

BN\_hex2bn(&ch, substr);

printBN("Verify of Message with original signature:", ch);

// 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);

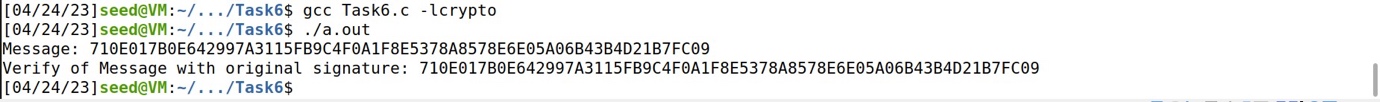
BN\_clear\_free(ch); BN\_clear\_free(res);

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

return 0;

}

**Implementation and Output:**

****

**Explanation and Observation:**

The above C program will verify the signature on the server's certificate using the issuer's public key obtained in Step 2.