Lab 10

Sanchit Dass

# **Task 1: Deriving the Private key**

Calculate the private key given the values of p, q and e (public key).

**p:** F7E75FDC469067FFDC4E847C51F452DF

**q:** E85CED54AF57E53E092113E62F436F4F

**e:** 0D88C3

**Source Code:**

*#include <stdio.h>*

*#include <openssl/bn.h>*

*#define NBITS 512*

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

*char \*number\_str;*

*if (type == 'd') {*

*number\_str = BN\_bn2dec(a);*

*} else if (type == 'h') {*

*number\_str = BN\_bn2hex(a);*

*}*

*printf("%s %s\n", msg, number\_str);*

*OPENSSL\_free(number\_str);*

*}*

*int main() {*

*BN\_CTX \*ctx = BN\_CTX\_new();*

*// Declare the numbers*

*BIGNUM \*p, \*q, \*n, \*e, \*d, \*phi, \*res;*

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

*// Allocate memory*

*p = BN\_new();*

*q = BN\_new();*

*n = BN\_new();*

*e = BN\_new();*

*d = BN\_new();*

*phi = BN\_new();*

*res = BN\_new();*

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

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

*// Initialize the variables*

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

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

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

*// Calculate (p-1) and (q-1)*

*BN\_sub(p\_minus\_one, p, BN\_value\_one());*

*BN\_sub(q\_minus\_one, q, BN\_value\_one());*

*// Calculate n and phi(n)*

*BN\_mul(n, p, q, ctx);*

*BN\_mul(phi, p\_minus\_one, q\_minus\_one, ctx);*

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

*// If not, exit the program*

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

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

*exit(0);*

*}*

*// Calculate the private key*

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

*printBN("Private key (hex):", d, 'h');*

*printBN("Private key (decimal):", d, 'd');*

*// Free memory*

*BN\_clear\_free(p);*

*BN\_clear\_free(q);*

*BN\_clear\_free(n);*

*BN\_clear\_free(e);*

*BN\_clear\_free(d);*

*BN\_clear\_free(phi);*

*BN\_clear\_free(res);*

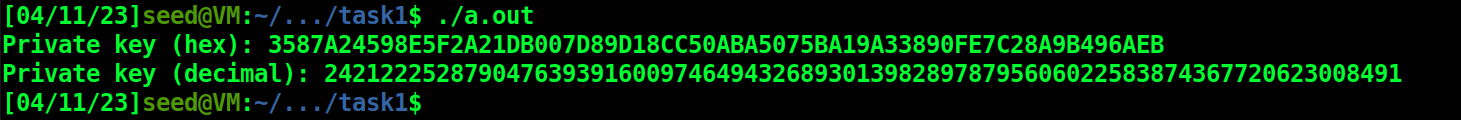
*BN\_clear\_free(p\_minus\_one);*

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

*return 0;*

*}*

**Output:**



# **Task 2: Encrypting a message**

Encrypt the message: “A top secret!”

Given,

**n:** DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5

**e:** 010001

**d:** 74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D

**m:** 4120746f702073656372657421

**Source Code:**

*#include <stdio.h>*

*#include <openssl/bn.h>*

*#define NBITS 512*

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

*char \*number\_str;*

*if (type == 'd') {*

*number\_str = BN\_bn2dec(a);*

*}*

*else if (type == 'h') {*

*number\_str = BN\_bn2hex(a);*

*}*

*printf("%s %s\n", msg, number\_str);*

*OPENSSL\_free(number\_str);*

*}*

*int main() {*

*BN\_CTX \*ctx = BN\_CTX\_new();*

*// Declare the numbers*

*BIGNUM \*n, \*e, \*d, \*c, \*m, \*dm;*

*// Allocate memory*

*n = BN\_new();*

*e = BN\_new();*

*d = BN\_new();*

*m = BN\_new();*

*c = BN\_new();*

*dm = BN\_new();*

*// Initialize the variables*

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

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

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

*BN\_hex2bn(&m, "4120746f702073656372657421");*

*// Print the message*

*printBN("Original Message:", m, 'h');*

*printf("\n----------------------------------------------------\n");*

*// Encrypt the message*

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

*printBN("Encryption result (hex):", c, 'h');*

*printBN("Encryption result (decimal):", c, 'd');*

*printf("\n----------------------------------------------------\n");*

*// Decrypt the message*

*BN\_mod\_exp(dm, c, d, n, ctx);*

*printBN("Decryption result (for verification):", dm, 'h');*

*// Free memory*

*BN\_clear\_free(n);*

*BN\_clear\_free(e);*

*BN\_clear\_free(d);*

*BN\_clear\_free(m);*

*BN\_clear\_free(c);*

*BN\_clear\_free(dm);*

*return 0;*

*}*

**Output:**

Text

Description automatically generated

# **Task 3: Decrypting a message**

Given a ciphertext, we need to decrypt and recover the original message.

**n:** DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5

**e:** 010001

**d:** 74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D

**c:** 8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F

**Source Code:**

*#include <stdio.h>*

*#include <openssl/bn.h>*

*#define NBITS 512*

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

*char \*number\_str;*

*if (type == 'd') {*

*number\_str = BN\_bn2dec(a);*

*}*

*else if (type == 'h') {*

*number\_str = BN\_bn2hex(a);*

*}*

*printf("%s %s\n", msg, number\_str);*

*OPENSSL\_free(number\_str);*

*}*

*int main() {*

*BN\_CTX \*ctx = BN\_CTX\_new();*

*// Declare the numbers*

*BIGNUM \*n, \*e, \*d, \*c, \*m;*

*// Allocate memory*

*n = BN\_new();*

*e = BN\_new();*

*d = BN\_new();*

*m = BN\_new();*

*c = BN\_new();*

*// Initialize the variables*

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

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

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

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

*// Decrypt the message*

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

*printBN("Decryption result:", m, 'h');*

*// Free memory*

*BN\_clear\_free(n);*

*BN\_clear\_free(e);*

*BN\_clear\_free(d);*

*BN\_clear\_free(m);*

*BN\_clear\_free(c);*

*return 0;*

*}*

**Output:**

**Text

Description automatically generated**

The original message: *Password is dees*

# **Task 4: Signing a message**

Sign this message: *“I owe you $2000.”*

**n:** DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5

**e:** 010001

**d:** 74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D

**Source Code:**

*#include <stdio.h>*

*#include <openssl/bn.h>*

*#define NBITS 512*

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

*char \*number\_str;*

*if (type == 'd') {*

*number\_str = BN\_bn2dec(a);*

*}*

*else if (type == 'h') {*

*number\_str = BN\_bn2hex(a);*

*}*

*printf("%s %s\n", msg, number\_str);*

*OPENSSL\_free(number\_str);*

*}*

*int main() {*

*BN\_CTX \*ctx = BN\_CTX\_new();*

*// Declare the numbers*

*BIGNUM \*n, \*e, \*d, \*s, \*m;*

*// Allocate memory*

*n = BN\_new();*

*e = BN\_new();*

*d = BN\_new();*

*m = BN\_new();*

*s = BN\_new();*

*// Initialize the variables*

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

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

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

*BN\_hex2bn(&m, "49206f776520796f752024333030302e");*

*// Print the message*

*printBN("Original Message:", m, 'h');*

*printf("\n----------------------------------------------------\n");*

*// Encrypt the message*

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

*printBN("Signature:", s, 'h');*

*// Free memory*

*BN\_clear\_free(n);*

*BN\_clear\_free(e);*

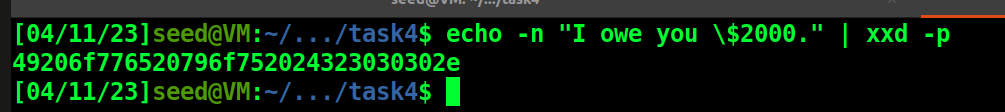
*BN\_clear\_free(d);*

*BN\_clear\_free(m);*

*BN\_clear\_free(s);*

*return 0;*

*}*



A screenshot of a computer

Description automatically generated with medium confidence

Changing the message to “*I owe you $3000.*” and signing it again:

A screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generated

The signatures are completely different even though the change in the original message is very small.

# **Task 5: Verifying a Signature**

In this task we need to verify the signature of Alice.

**M:** Launch a missile.

**S:** 643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F

**e:** 010001 (this hex value equals to decimal 65537)

**n:** AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115

**Source Code:**

*#include <stdio.h>*

*#include <openssl/bn.h>*

*#define NBITS 512*

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

*char \*number\_str;*

*if (type == 'd') {*

*number\_str = BN\_bn2dec(a);*

*}*

*else if (type == 'h') {*

*number\_str = BN\_bn2hex(a);*

*}*

*printf("%s %s\n", msg, number\_str);*

*OPENSSL\_free(number\_str);*

*}*

*int main() {*

*BN\_CTX \*ctx = BN\_CTX\_new();*

*// Declare the numbers*

*BIGNUM \*n, \*e, \*s, \*m, \*dm;*

*// Allocate memory*

*n = BN\_new();*

*e = BN\_new();*

*s = BN\_new();*

*m = BN\_new();*

*dm = BN\_new();*

*// Initialize the variables*

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

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

*BN\_hex2bn(&s, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");*

*BN\_hex2bn(&m, "4c61756e63682061206d697373696c652e");*

*// Print the original message*

*printBN("Original Message:", m, 'h');*

*// Decrypt the signature*

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

*printBN("Decryption:", dm, 'h');*

*if (BN\_cmp(m, dm) == 0) {*

*printf("The signature is valid.\n");*

*}*

*else {*

*printf("Invalid signature!\n");*

*}*

*// Free memory*

*BN\_clear\_free(n);*

*BN\_clear\_free(e);*

*BN\_clear\_free(s);*

*BN\_clear\_free(m);*

*BN\_clear\_free(dm);*

*return 0;*

*}*

Getting the hex string from the original message:

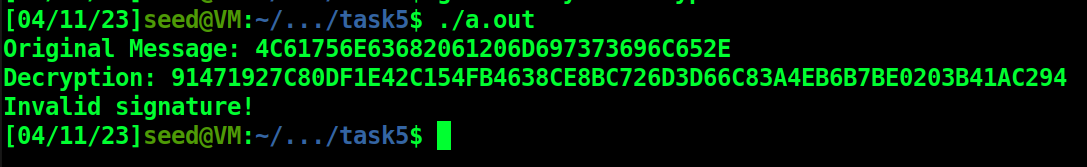
A screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generated

After changing the last 2 bits of the signature from 2F to 3F and re-running the script, the signature was invalid:



# **Task 6: Manually verifying an X.509 certificate**

Step 1: Download a certificate from a real web server.

Graphical user interface, text

Description automatically generated

c0.pem:

-----BEGIN CERTIFICATE-----

*MIIQETCCDvmgAwIBAgIQBR86m9kKDNmZlttTjZK7rDANBgkqhkiG9w0BAQsFADB1*

*MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3*

*d3cuZGlnaWNlcnQuY29tMTQwMgYDVQQDEytEaWdpQ2VydCBTSEEyIEV4dGVuZGVk*

*IFZhbGlkYXRpb24gU2VydmVyIENBMB4XDTIyMTExMDAwMDAwMFoXDTIzMTExMDIz*

*NTk1OVowgcYxEzARBgsrBgEEAYI3PAIBAxMCVVMxGTAXBgsrBgEEAYI3PAIBAhMI*

*RGVsYXdhcmUxHTAbBgNVBA8MFFByaXZhdGUgT3JnYW5pemF0aW9uMRAwDgYDVQQF*

*EwczMDE0MjY3MQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTERMA8G*

*A1UEBxMIU2FuIEpvc2UxFTATBgNVBAoTDFBheVBhbCwgSW5jLjEXMBUGA1UEAxMO*

*d3d3LnBheXBhbC5jb20wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDD*

*ZMIdYImsLpW9CiBu8LHXGj4I7eLBYBoqWcN91IkuPXxdOW3/oF5OKeCDJOnKHhY9*

*77pvsvYnps5gjggehUerXS1qyc+k8g6mPnwFOgY6G1tpkhXVhSJVtoz30+ETdpY0*

*0sUriCJoxdFVcTwAyYobVaoCDNj1toKEOMMaW15PYDfMRfhpsI0ROh3TOhUmo4UX*

*8SOXgVwH4iHSYdsB2GbgHvY14JMzmNXzCagjG08Xg3inLF4YLejB/qrpNS53YcoZ*

*EdrNLA7/tGH8+wmcSBnn2ze95VdKgjgmn6VZMyLU7vCnNKiiSf+poDL9sVuhGJXY*

*PYQnK5kyrv1bgKxAm2CpAgMBAAGjggxJMIIMRTAfBgNVHSMEGDAWgBQ901Cl1qCt*

*7vNKYApl0yHU+PjWDzAdBgNVHQ4EFgQU1RC/VB6jJV2yjsquQNE+6kOvvR0wggj4*

*BgNVHREEggjvMIII64IOd3d3LnBheXBhbC5jb22CEXd3dy5wYXlwYWwuY29tLmFy*

*ghF3d3ctc3QucGF5cGFsLmNvbYILbS5wYXlwYWwuZXOCEXd3dy5wYXlwYWwuY29t*

*LmF1ghJpZC5oeXBlcndhbGxldC5jb22CDXd3dy5wYXlwYWwuY2+CFmNyZWRpdGFw*

*cGx5LnBheXBhbC5jb22CDXd3dy5wYXlwYWwuY2iCEHBwLXVzLnBheXBhbC5jb22C*

*FXNhZmVicmVhY2gucGF5cGFsLmNvbYINd3d3LnBheXBhbC5jbIIPZGVtby5wYXlw*

*YWwuY29tghFtb2JpbGUucGF5cGFsLmNvbYIQd3d3LnBheXBhbC5jby5pZIITY2hl*

*Y2tvdXQucGF5cGFsLmNvbYINd3d3LnBheXBhbC5jYYISc3Nwc2Vydi5wYXlwYWwu*

*Y29tghF3d3cucGF5cGFsLmNvbS5zYYIbYnJhbmQuYnJhaW50cmVlcGF5bWVudHMu*

*Y29tghl3d3cuYnJhaW50cmVlcGF5bWVudHMuY29tghF3d3cucGF5cGFsLmNvbS5i*

*coIZc2VjdXJlcGF5bWVudHMucGF5cGFsLmNvbYIRd3d3LnBheXBhbC5jb20uc2eC*

*E3RyYW5zZmVyLnBheXBhbC5jb22CC20ucGF5cGFsLmZyghB3d3cucGF5cGFsLmNv*

*LnphghR3d3cucGF5cGFsLWZyYW5jZS5mcoIQd3d3LnBheXBhbC5jby5pboIQd3d3*

*LnBheXBhbC5jby5pbIINd3d3LnBheXBhbC5zZYIdYXNzZXRzLWNkbi5zYW5kYm94*

*LnMteG9vbS5jb22CC2lkLnhvb20uY29tghV3d3cudGhlcGF5cGFsYmxvZy5jb22C*

*DXd3dy5wYXlwYWwuanCCFWFzc2V0cy1jZG4ucy14b29tLmNvbYIXanMuYnJhaW50*

*cmVlZ2F0ZXdheS5jb22CDXd3dy5wYXlwYWwuYmWCEXd3dy5wYXlwYWwuY29tLmNu*

*ghVvbmJvYXJkaW5nLnBheXBhbC5jb22CDXd3dy5wYXlwYWwudm6CDXd3dy5wYXlw*

*YWwuZXOCEXd3dy5wYXlwYWwuY29tLnRygg13d3cucGF5cGFsLmV1ghp3d3cuYnJh*

*aW50cmVlZmluYW5jaWFsLmNvbYIRd3d3LnBheXBhbC5jb20udHeCG2Fzc2V0cy5i*

*cmFpbnRyZWVnYXRld2F5LmNvbYIWYnJhaW50cmVlZmluYW5jaWFsLmNvbYIfd3d3*

*LmJyYW5kLmJyYWludHJlZXBheW1lbnRzLmNvbYIFcHkucGyCGXd3dy5wYXlwYWwt*

*ZGV1dHNjaGxhbmQuZGWCDXd3dy5wYXlwYWwubWWCDGlkLnZlbm1vLmNvbYIQd3d3*

*LnBheXBhbC5jby5rcoINaWQuemV0dGxlLmNvbYITY29ycy5hcGkucGF5cGFsLmNv*

*bYINd3d3LnBheXBhbC5ka4IQbG9naW4ucGF5cGFsLmNvbYIQcHAtYXUucGF5cGFs*

*LmNvbYINd3d3LnBheXBhbC5sdYIQcHAtZXUucGF5cGFsLmNvbYIRd3d3LnBheXBh*

*bC5jb20uZXOCDXd3dy5wYXlwYWwuZGWCEHd3dy5wYXlwYWwuY28udGiCEXd3dy5w*

*YXlwYWwuY29tLm15ghF3d3cucGF5cGFsLmNvbS5teIIRc2VjdXJlLnBheXBhbC5j*

*b22CEXd3dy5wYXlwYWwuY29tLnZlgg13d3cucGF5cGFsLnBoghR3d3cucGF5cGFs*

*LWxhdGFtLmNvbYIVYnJhaW50cmVlcGF5bWVudHMuY29tghZ3d3cucGF5cGFsLWZv*

*cndhcmQuY29tggl3d3cucHkucGyCC20ucGF5cGFsLml0ghJoaXN0b3J5LnBheXBh*

*bC5jb22CEHd3dy5wYXlwYWwuY28udWuCE2JyYWludHJlZWNoYXJnZS5jb22CDHQu*

*cGF5cGFsLmNvbYIdYnJhaW50cmVlcGF5bWVudHNvbHV0aW9ucy5jb22CDWM2LnBh*

*eXBhbC5jb22CDG0ucGF5cGFsLmNvbYIPcGljcy5wYXlwYWwuY29tghd3d3cuYnJh*

*aW50cmVlY2hhcmdlLmNvbYINd3d3LnBheXBhbC5mcoITYnVzaW5lc3MucGF5cGFs*

*LmNvbYIQaWQuam9pbmhvbmV5LmNvbYIUZmluYW5jaW5nLnBheXBhbC5jb22CDnNz*

*cC5wYXlwYWwuY29tgiF3d3cuYnJhaW50cmVlcGF5bWVudHNvbHV0aW9ucy5jb22C*

*DWlkLnBheXBhbC5jb22CFGRldmVsb3Blci5wYXlwYWwuY29tghBmcGRicy5wYXlw*

*YWwuY29tghd3d3cucGF5cGFsLW1hcmtldGluZy5wbIINd3d3LnBheXBhbC5ubIIN*

*d3d3LnBheXBhbC5ub4INd3d3LnBheXBhbC5maYIVd3d3LnBheXBhbG9iamVjdHMu*

*Y29tghN3d3cucGF5cGFsLW1lbmEuY29tghB3d3cucGF5cGFsLmNvLm56ghF3d3cu*

*cGF5cGFsLmNvbS5wZYIPcXdhYy5wYXlwYWwuY29tghljb250ZW50LnBheXBhbG9i*

*amVjdHMuY29tgg13d3cucGF5cGFsLmF0ghdpZC5icmFpbnRyZWVnYXRld2F5LmNv*

*bYIJcGF5cGFsLm1lgg13d3cucGF5cGFsLmlughhwb2ludG9mc2FsZS1zLnBheXBh*

*bC5jb22CDHAucGF5cGFsLmNvbYINd3d3LnBheXBhbC5pdIIRd3d3LnBheXBhbC5j*

*b20uaGuCDXd3dy5wYXlwYWwuaWWCHXpldHRsZWludGVncmF0aW9ucy5wYXlwYWwu*

*Y29tghBwcC1pbi5wYXlwYWwuY29tggxjLnBheXBhbC5jb22CFHd3dy5wYXlwYWwt*

*Z2lmdHMuY29tgg5tLnBheXBhbC5jby51a4IOcGVwLnBheXBhbC5jb22CDXd3dy5w*

*YXlwYWwucHSCD20ucGF5cGFsLmNvbS5hdYINd3d3LnBheXBhbC5wbDAOBgNVHQ8B*

*Af8EBAMCBaAwHQYDVR0lBBYwFAYIKwYBBQUHAwEGCCsGAQUFBwMCMHUGA1UdHwRu*

*MGwwNKAyoDCGLmh0dHA6Ly9jcmwzLmRpZ2ljZXJ0LmNvbS9zaGEyLWV2LXNlcnZl*

*ci1nMy5jcmwwNKAyoDCGLmh0dHA6Ly9jcmw0LmRpZ2ljZXJ0LmNvbS9zaGEyLWV2*

*LXNlcnZlci1nMy5jcmwwSgYDVR0gBEMwQTALBglghkgBhv1sAgEwMgYFZ4EMAQEw*

*KTAnBggrBgEFBQcCARYbaHR0cDovL3d3dy5kaWdpY2VydC5jb20vQ1BTMIGIBggr*

*BgEFBQcBAQR8MHowJAYIKwYBBQUHMAGGGGh0dHA6Ly9vY3NwLmRpZ2ljZXJ0LmNv*

*bTBSBggrBgEFBQcwAoZGaHR0cDovL2NhY2VydHMuZGlnaWNlcnQuY29tL0RpZ2lD*

*ZXJ0U0hBMkV4dGVuZGVkVmFsaWRhdGlvblNlcnZlckNBLmNydDAJBgNVHRMEAjAA*

*MIIBfQYKKwYBBAHWeQIEAgSCAW0EggFpAWcAdQDoPtDaPvUGNTLnVyi8iWvJA9PL*

*0RFr7Otp4Xd9bQa9bgAAAYRj1zXXAAAEAwBGMEQCIBq2JgHX537xp1KIv1QYqFMA*

*1w7/klPjO0wPzKTqJiFdAiAGsL4fD66gTiQ0/fwrpRic+ykjdxXncfZ+Kqf0u72G*

*GgB2ALNzdwfhhFD4Y4bWBancEQlKeS2xZwwLh9zwAw55NqWaAAABhGPXNjEAAAQD*

*AEcwRQIgeYaiD0kPU0zcb7NEEN35BX+R8JxLj0BVfQw1Zf1VlbMCIQCJA4llrBZm*

*pE2JYatZ8jEC3wnNNn1ZpweID/6lVSkefgB2ALc++yTfnE26dfI5xbpY9Gxd/ELP*

*ep81xJ4dCYEl7bSZAAABhGPXNeoAAAQDAEcwRQIgAY2RXWWPn+g3KdL6gJ8zehdb*

*e0zAH4jXmVDYN8w+h20CIQD/eI1n9PrvCFlzD7cDFCrQQfmwrzGuVbJMpXBReiR7*

*BjANBgkqhkiG9w0BAQsFAAOCAQEAPrshTKi0PH/ZIRDE2pVibz9iu2qiJl/3l7BY*

*QsVZ9PQjiMBxM47RKYCipmSBdjRsPiBtygesOTzz0paECUt8x7w/nKkXLVrAlnwW*

*8sGgFTeVWb0NKkEAXAuIYZ4vjadxlhfJGqEe4hVRtapFVPx26xFG2m3f3/tUNbhd*

*+o7dhYWdG08adh4DXxwnFW589fjuGffBvTyxM++UGX55n4ZBIpNmbNwYwcShaH1Q*

*Mvdb4kwtnv4vnipINd7o9KUIItOiVvmlGY1vKEYPWmdoHiVeYfRBYYp4PNSkJJtu*

*6SzIS2Zr7vedR58KI+5CP2duf/H5deEq8IgPDfIUJTMVKFqQOg==*

*-----END CERTIFICATE-----*

c1.pem

*-----BEGIN CERTIFICATE-----*

*MIIEtjCCA56gAwIBAgIQDHmpRLCMEZUgkmFf4msdgzANBgkqhkiG9w0BAQsFADBs*

*MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3*

*d3cuZGlnaWNlcnQuY29tMSswKQYDVQQDEyJEaWdpQ2VydCBIaWdoIEFzc3VyYW5j*

*ZSBFViBSb290IENBMB4XDTEzMTAyMjEyMDAwMFoXDTI4MTAyMjEyMDAwMFowdTEL*

*MAkGA1UEBhMCVVMxFTATBgNVBAoTDERpZ2lDZXJ0IEluYzEZMBcGA1UECxMQd3d3*

*LmRpZ2ljZXJ0LmNvbTE0MDIGA1UEAxMrRGlnaUNlcnQgU0hBMiBFeHRlbmRlZCBW*

*YWxpZGF0aW9uIFNlcnZlciBDQTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoC*

*ggEBANdTpARR+JmmFkhLZyeqk0nQOe0MsLAAh/FnKIaFjI5j2ryxQDji0/XspQUY*

*uD0+xZkXMuwYjPrxDKZkIYXLBxA0sFKIKx9om9KxjxKws9LniB8f7zh3VFNfgHk/*

*LhqqqB5LKw2rt2O5Nbd9FLxZS99RStKh4gzikIKHaq7q12TWmFXo/a8aUGxUvBHy*

*/Urynbt/DvTVvo4WiRJV2MBxNO723C3sxIclho3YIeSwTQyJ3DkmF93215SF2AQh*

*cJ1vb/9cuhnhRctWVyh+HA1BV6q3uCe7seT6Ku8hI3UarS2bhjWMnHe1c63YlC3k*

*8wyd7sFOYn4XwHGeLN7x+RAoGTMCAwEAAaOCAUkwggFFMBIGA1UdEwEB/wQIMAYB*

*Af8CAQAwDgYDVR0PAQH/BAQDAgGGMB0GA1UdJQQWMBQGCCsGAQUFBwMBBggrBgEF*

*BQcDAjA0BggrBgEFBQcBAQQoMCYwJAYIKwYBBQUHMAGGGGh0dHA6Ly9vY3NwLmRp*

*Z2ljZXJ0LmNvbTBLBgNVHR8ERDBCMECgPqA8hjpodHRwOi8vY3JsNC5kaWdpY2Vy*

*dC5jb20vRGlnaUNlcnRIaWdoQXNzdXJhbmNlRVZSb290Q0EuY3JsMD0GA1UdIAQ2*

*MDQwMgYEVR0gADAqMCgGCCsGAQUFBwIBFhxodHRwczovL3d3dy5kaWdpY2VydC5j*

*b20vQ1BTMB0GA1UdDgQWBBQ901Cl1qCt7vNKYApl0yHU+PjWDzAfBgNVHSMEGDAW*

*gBSxPsNpA/i/RwHUmCYaCALvY2QrwzANBgkqhkiG9w0BAQsFAAOCAQEAnbbQkIbh*

*hgLtxaDwNBx0wY12zIYKqPBKikLWP8ipTa18CK3mtlC4ohpNiAexKSHc59rGPCHg*

*4xFJcKx6HQGkyhE6V6t9VypAdP3THYUYUN9XR3WhfVUgLkc3UHKMf4Ib0mKPLQNa*

*2sPIoc4sUqIAY+tzunHISScjl2SFnjgOrWNoPLpSgVh5oywM395t6zHyuqB8bPEs*

*1OG9d4Q3A84ytciagRpKkk47RpqF/oOi+Z6Mo8wNXrM9zwR4jxQUezKcxwCmXMS1*

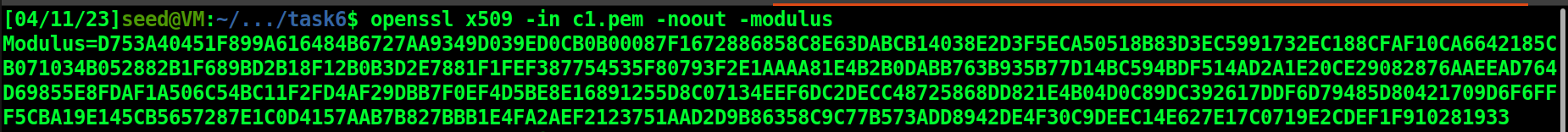
*oVWNWlZopCJwqjyBcdmdqEU79OX2olHdx3ti6G8MdOu42vi/hw15UJGQmxg7kVkn*

*8TUoE6smftX3eg==*

*-----END CERTIFICATE-----*

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

The below screenshot shows the value of n:



The below screenshot shows the value of e:

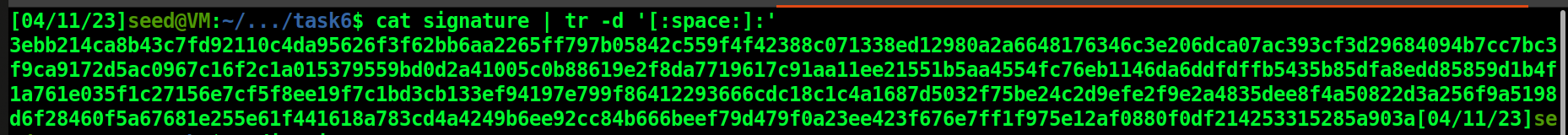
Text

Description automatically generated

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

Text

Description automatically generated



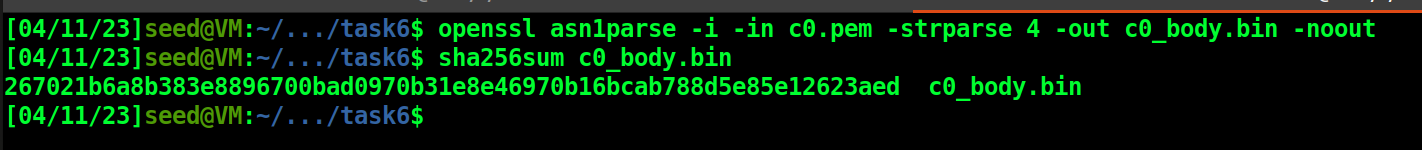
Step 4: Extract the body of the server’s certificate.

A screenshot of a computer

Description automatically generated

Graphical user interface, text

Description automatically generated



Step 5: Verify the signature.

**Source Code:**

*#include <stdio.h>*

*#include <string.h>*

*#include <openssl/bn.h>*

*#define SHA\_LENGTH 64*

*#define NBITS 512*

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

*char \*number\_str;*

*if (type == 'd') {*

*number\_str = BN\_bn2dec(a);*

*}*

*else if (type == 'h') {*

*number\_str = BN\_bn2hex(a);*

*}*

*printf("%s %s\n", msg, number\_str);*

*OPENSSL\_free(number\_str);*

*}*

*int main() {*

*BN\_CTX \*ctx = BN\_CTX\_new();*

*// Declare the numbers*

*BIGNUM \*n, \*e, \*s, \*m, \*dm;*

*// Allocate memory*

*n = BN\_new();*

*e = BN\_new();*

*s = BN\_new();*

*m = BN\_new();*

*dm = BN\_new();*

*// Initialize the variables*

*BN\_hex2bn(&n

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

*BN\_hex2bn(&s, "");*

*BN\_hex2bn(&m, "267021b6a8b383e8896700bad0970b31e8e46970b16bcab788d5e85e12623aed");*

*// Print the original message*

*printBN("Original Message:", m, 'h');*

*// Decrypt the signature*

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

*// Extract the last 64 characters (the remaining characters are part of the padding scheme)*

*char \*decrypt\_str = BN\_bn2hex(dm);*

*char substr[100];*

*strncpy(substr, decrypt\_str + strlen(decrypt\_str) - SHA\_LENGTH, SHA\_LENGTH);*

*BN\_hex2bn(&dm, substr);*

*printBN("Decryption:", dm, 'h');*

*if (BN\_cmp(m, dm) == 0) {*

*printf("The signature is valid.\n");*

*}*

*else {*

*printf("Invalid signature!\n");*

*}*

*// Free memory*

*BN\_clear\_free(n);*

*BN\_clear\_free(e);*

*BN\_clear\_free(s);*

*BN\_clear\_free(m);*

*BN\_clear\_free(dm);*

*return 0;*

*}*

Text

Description automatically generated