CSE644 Lab9

Yishi Lu

4/10/2018

**Task1: Deriving the Private Key**

Code for task1

#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 \*one = BN\_new();

BIGNUM \*p = BN\_new();

BIGNUM \*q = BN\_new();

BIGNUM \*e = BN\_new();

BIGNUM \*x = BN\_new();

BIGNUM \*d = BN\_new();

// Initialize p, q, e, n, and one

BN\_hex2bn (&one, "00001"); //hex value of number 1

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

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

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

//calculate d by formular e\*d mod (p-1)\*(q-1) = 1

BN\_sub (p, p, one); // p=p-1

BN\_sub (q, q, one); // q=q-1

BN\_mul (x, p, q, ctx); //x=(p-1)\*(q-1)

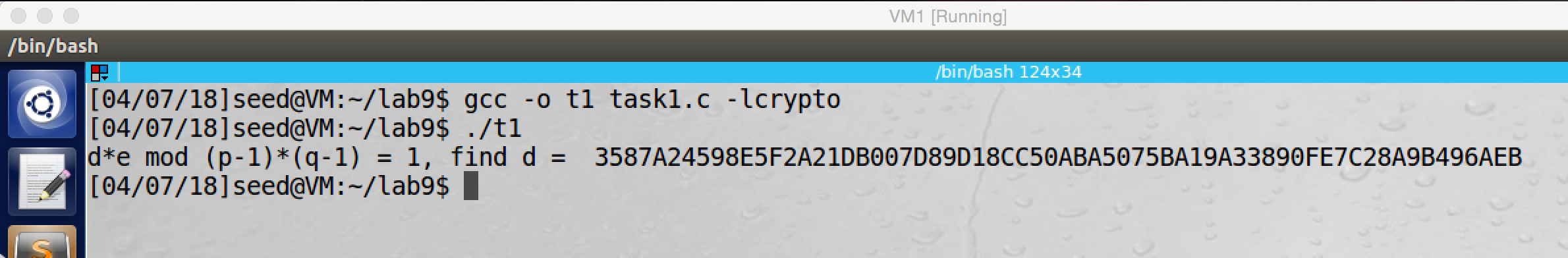
BN\_mod\_inverse(d, e, x, ctx); // calculate d from e\*d mod (p-1)\*(q-1) = 1

//print result

printBN("d\*e mod (p-1)\*(q-1) = 1, find d = ", d);

return 0;

}



screenshot1. After run the above code, we got result of d.

**Observation and Explanation:**

For this task, we are given p, q, and e, and we need to find d. Therefore, we use Extended Euclidian Algorithm: e\*d mod (p-1)\*(q-1) = 1. After we compile and run the code, we got result of d which is shown in screenshot1.

**Task2: Encrypting a Message**

Code for task 2

#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 \*e = BN\_new();

BIGNUM \*n = BN\_new();

BIGNUM \*d = BN\_new();

BIGNUM \*M = BN\_new();

BIGNUM \*C = BN\_new();

// Initialize e, n, d, M

BN\_hex2bn (&M, "4120746f702073656372657421");

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

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

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

//calculate ciphertext C by formular M^e mod n

BN\_mod\_exp(C, M, e, n, ctx);

printBN("restul of M^e mod n is ", C);

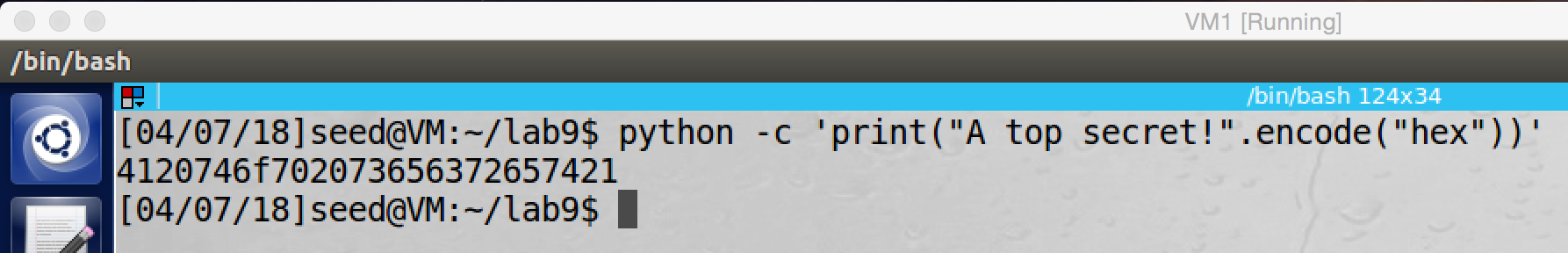
// result verification, decrypting the ciphertext by formula C^d mod n

BN\_mod\_exp(M, C, d, n, ctx);

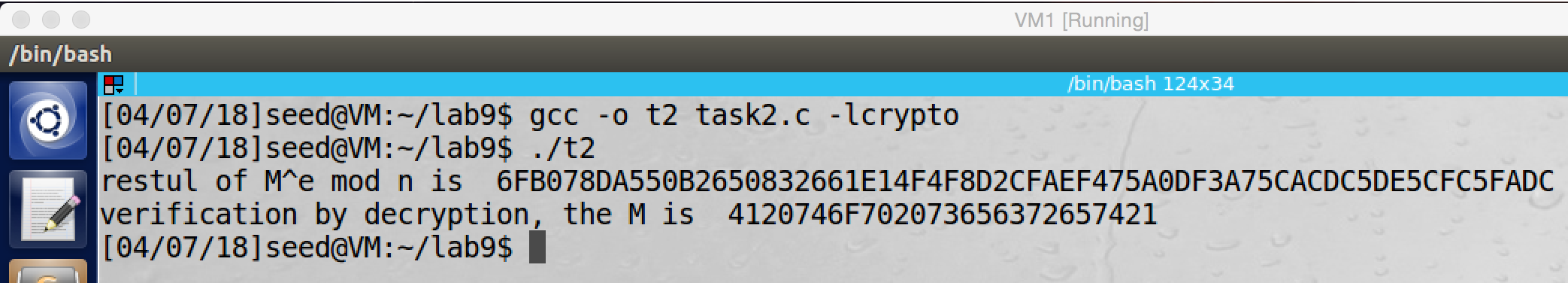
printBN("verification by decryption, the M is ", M);

return 0;

}



screenshot1. Using python to convert “A top secret!” to hex value



screenshot2. Compile and run task2.c program, we get ciphertext. Moreover, we also decrypt the ciphertext, and we get a M, which is exactly same as the original M.

**Observation and Explanation:**

In this task, we are given n, e, d, and M. we use formula (M^e mod n) to get cipher text C. And then we use (C^d mod n) to get M for verification.

First, we convert the string M to hex value (screenshot1). And then we compile and run taske2.c. As the screenshot2 shows, by running the program, we successfully get the cipher text C. And then we also use the private key to decrypt the cipher text, and we get message M, which is exactly same as the original M.

**Task3: Decrypting a Message**

Code for task3:

#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 \*e = BN\_new();

BIGNUM \*n = BN\_new();

BIGNUM \*d = BN\_new();

BIGNUM \*C = BN\_new();

BIGNUM \*M = BN\_new();

// Initialize e, n, d, C

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

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

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

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

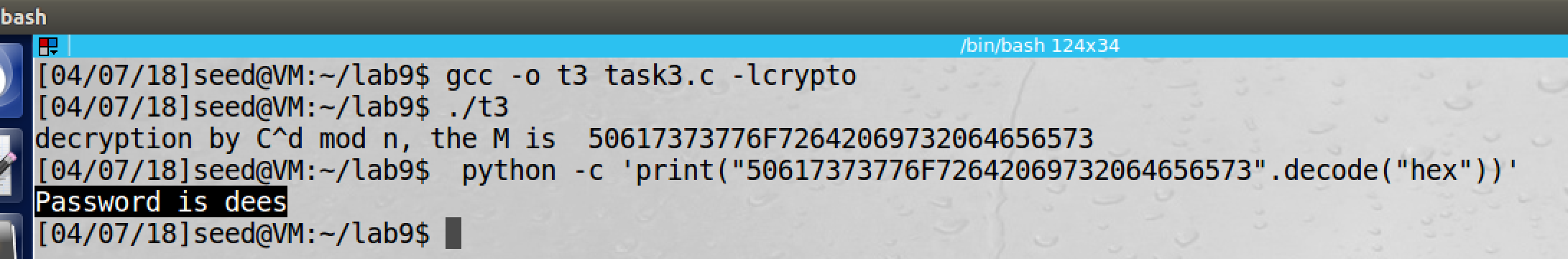
//decoding ciphertext C to get message M by formula C^d mod n

BN\_mod\_exp(M, C, d, n, ctx);

printBN("decryption by C^d mod n, the M is ", M);

return 0;

}



screenshot1. Decrypting the cipher text by the private key and n, and then we get hex value of the message. Then we use python to decode the hex value, and we get the message “Password is dees”

**Observation and Explanation:**

In this task, we are give e, n, d, and C. we use formula (C^d mod n) to decrypt the cipher text.

As the screenshot1 show, we compile and run the program task3.c, and then we get the hex value of the M. Afterwards, we run python to decode the hex value. Finally, we get the message which is “Password is dees”.

**Task4: Signing a Message**

#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 \*e = BN\_new();

BIGNUM \*n = BN\_new();

BIGNUM \*d = BN\_new();

BIGNUM \*C1 = BN\_new();

BIGNUM \*C2 = BN\_new();

BIGNUM \*M1 = BN\_new();

BIGNUM \*M2 = BN\_new();

// initialize M1, M2, d, n, e

BN\_hex2bn (&M1, "49206f776520796f752024323030302e");

BN\_hex2bn (&M2, "49206f776520796f752024333030302e");

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

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

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

//generate signature C1 by M1^d mod n

BN\_mod\_exp(C1, M1, d, n, ctx); // generate signature for "I owe you $2000"

printBN("The signature for I owe you $2000. is ", C1);

BN\_mod\_exp(M1, C1, e, n, ctx); // answer verification

printBN("Verification by decryption ", M1);

//generate signature C2 by M2^d mod n

BN\_mod\_exp(C2, M2, d, n, ctx); // generate signature for "I owe you $3000"

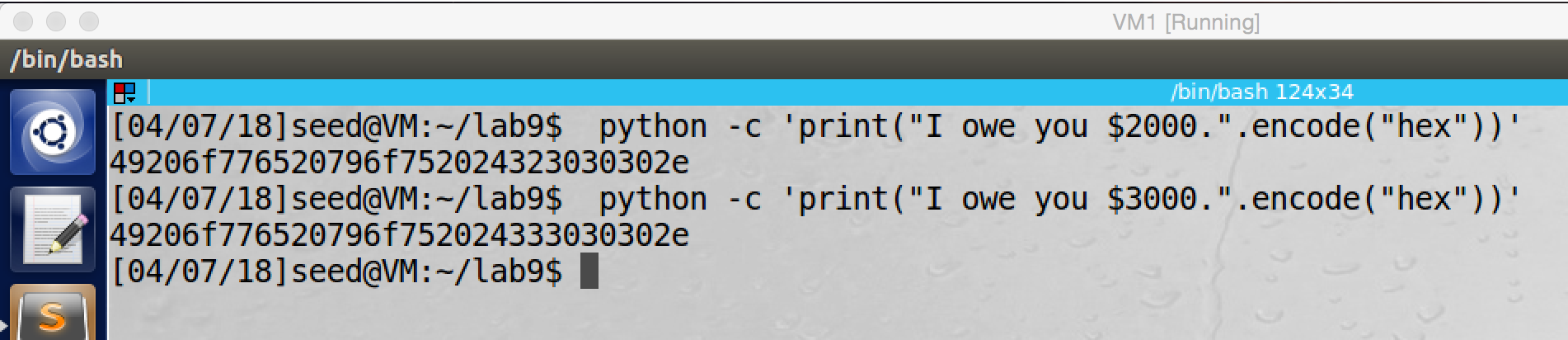
printBN("The signature for I owe you $3000. is ", C2);

BN\_mod\_exp(M2, C2, e, n, ctx); // answer verification

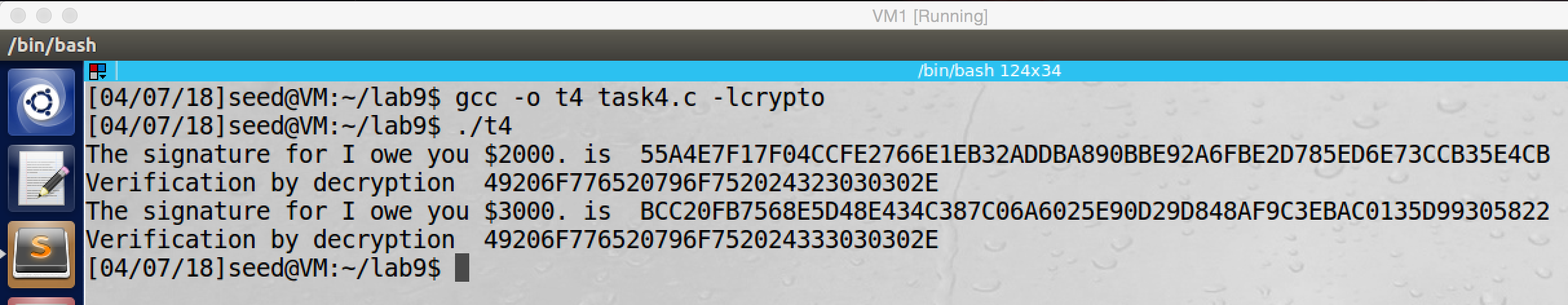
printBN("Verification by decryption ", M2);

return 0;

}



screenshot1. Convert messages into hex value



screenshot2. Compile and run program. We get the signature for M1 “I owe you $2000.”, and signature for M2 “I owe you $3000.”. Otherwise, we also use the public key to generate the hex value for M1 and M2, this is jut for answer verification.

**Observation and Explanation:**

For this task, we are given e, n, d, and two message M1 (“I owe you $2000.”), M2 (“I owe you $3000.”). We use (M^d mod n) to generate signature for M1 and M2.

First we convert M1 and M2 into hex value (screenshot1). After we compile and run the program task4.c, we get signature for M1 and M2 (screenshot2). Even there is just a slightly change, the signature of M1 and M2 are totally different. Moreover, after we get signatures (C1 for M1, and C2 for M2), we also use (C^e mod n) to generate M1 and M2 for verification.

**Task5: Verifying a Signature**

#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 \*e = BN\_new();

BIGNUM \*n = BN\_new();

BIGNUM \*d = BN\_new();

BIGNUM \*S = BN\_new();

BIGNUM \*M = BN\_new();

// Initialize S, n, e

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

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

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

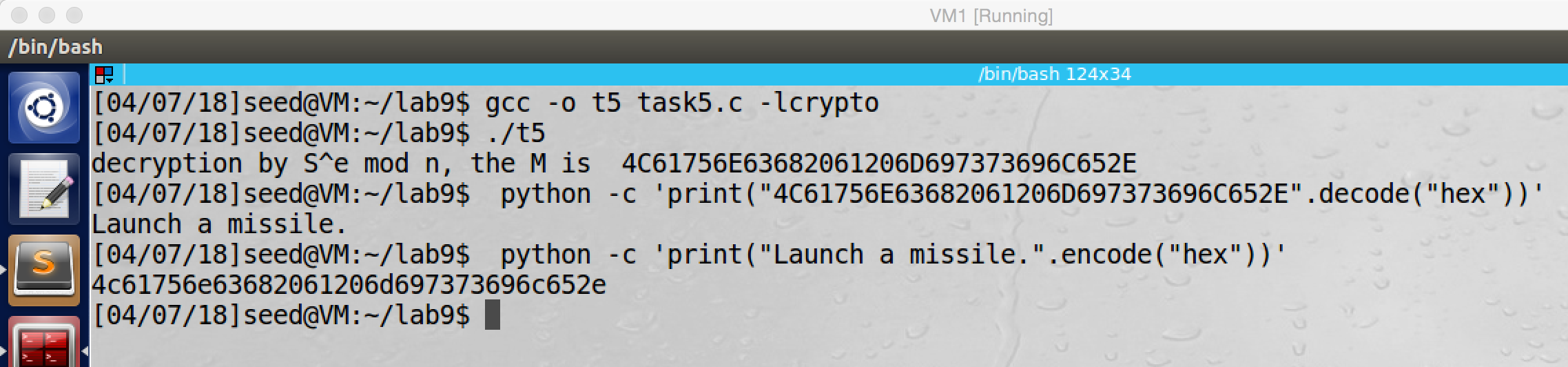
//decoding signature S to get message M by S^e mod n

BN\_mod\_exp(M, S, e, n, ctx);

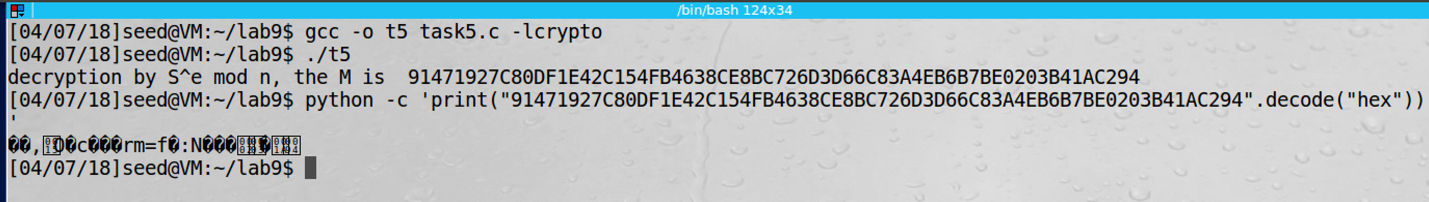
printBN("decryption by S^e mod n, the M is ", M);

return 0;

}



screenshot1. After compile and run program task5.c, we generate hex value for M. and then we decode the hex value, and we get message “Launch a missile.” which is exactly same as the original message. So this signature is indeed Alice’s.



screenshot2. After we change the last byte of the signature from 2F to 3F and run the program again, we get a very different hex value for the message M. After we decode the hex value, we get some corrupted characters. We have no idea what this is.

**Observation and Explanation:**

In this task, we are given e, n, S, and original message M. Our task is to verify the signature S is generated from M. For this purpose, we can use (S^e mod n) to get message M’. If M and M’ are exactly same, then the signature S is indeed Alice’s.

First, we compile and run the program task4.c, and it prints the message M which is generated by formula (S^e mod n) (screenshot1). Because the M is hex value, so we decode the M, and we get string “Launch a missile.”, this message is exactly same as the Alice’s message (screenshot1). So the signature is indeed Alice’s.

As the screenshot2 shows, if we changed the last byte of the signature S from 2F to 3F, then after we run the program, we got a totally different hex value for M. After we decode the hex value, we got some corrupted characters. And these corrupted characters are not the message “Launch a missile.” Obviously. So the signature is not Alice’s.

**Task6: Manually Verifying an X.509 Certificate**

Code for task 6

#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 \*e = BN\_new();

BIGNUM \*n = BN\_new();

BIGNUM \*d = BN\_new();

BIGNUM \*S = BN\_new();

BIGNUM \*M = BN\_new();

// Initialize S, n, e

BN\_hex2bn (&S, "");

BN\_hex2bn (&n

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

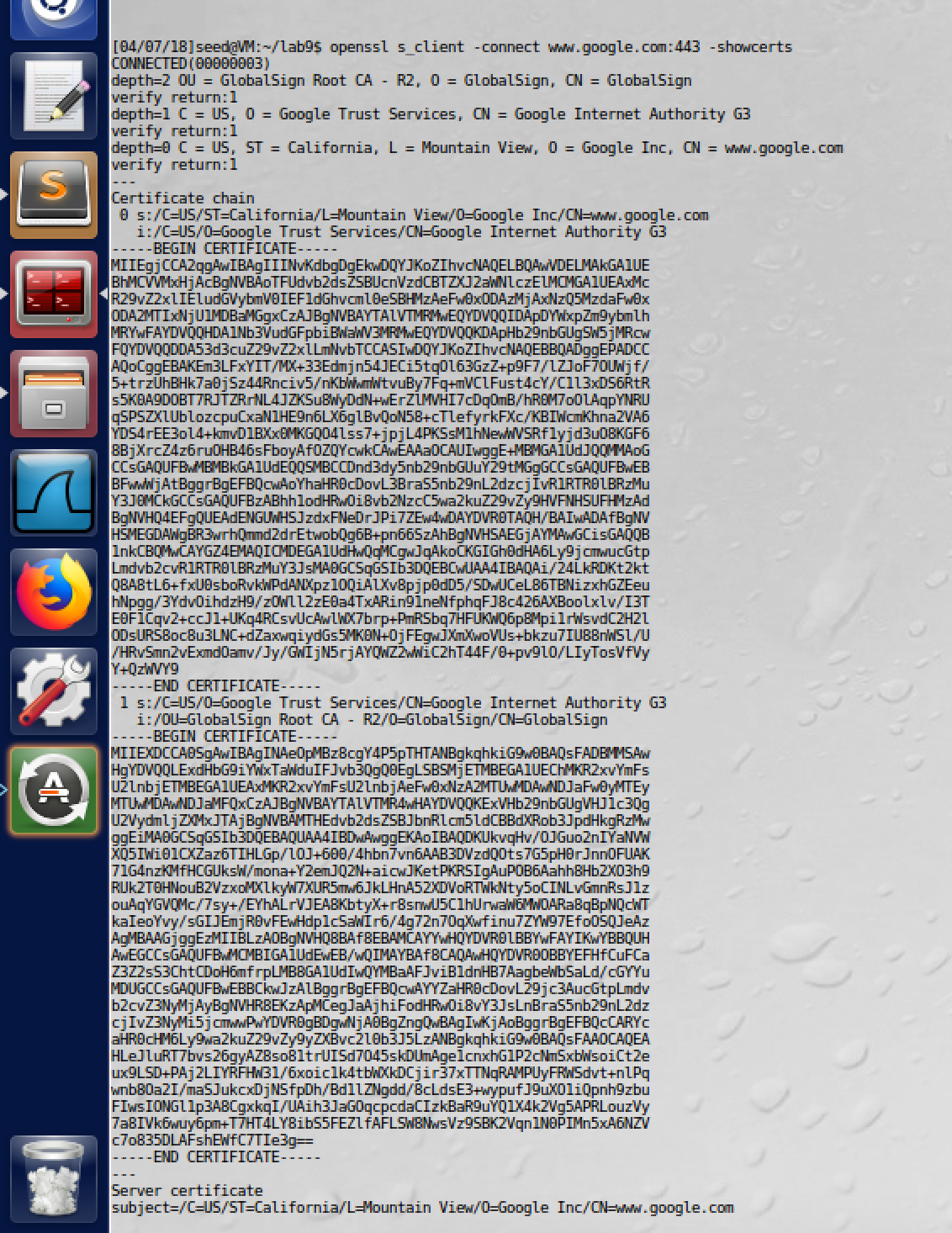
//decoding signature S to get certificate body M by S^e mod n

BN\_mod\_exp(M, S, e, n, ctx);

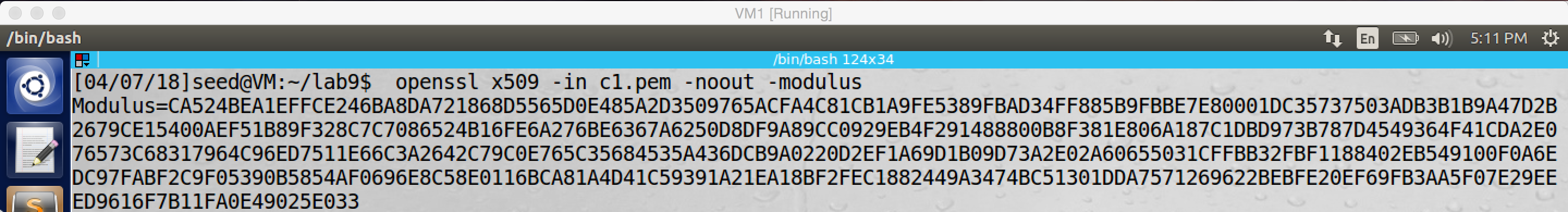
printBN("decryption by S^e mod n, the M is ", M); //print message M

return 0;

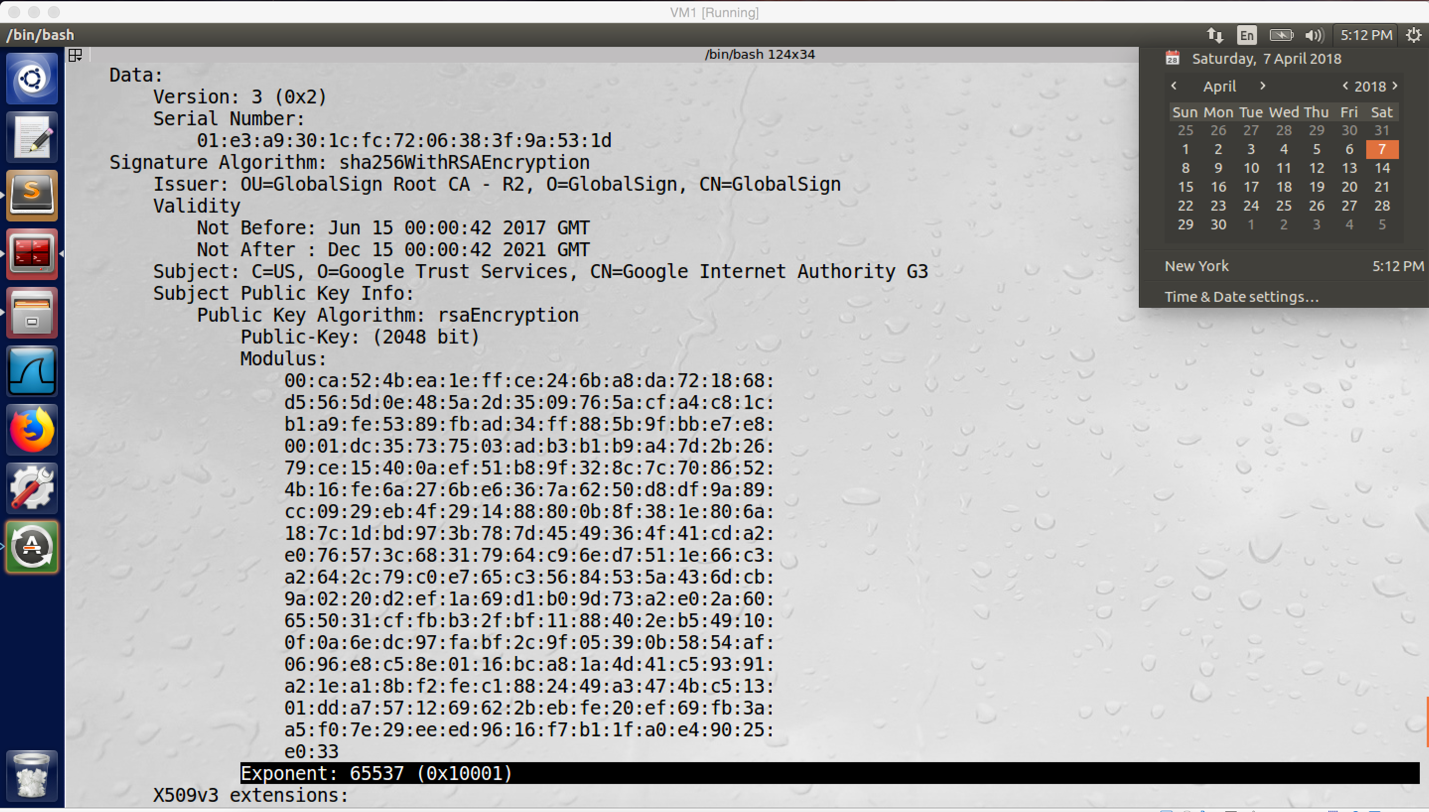
}



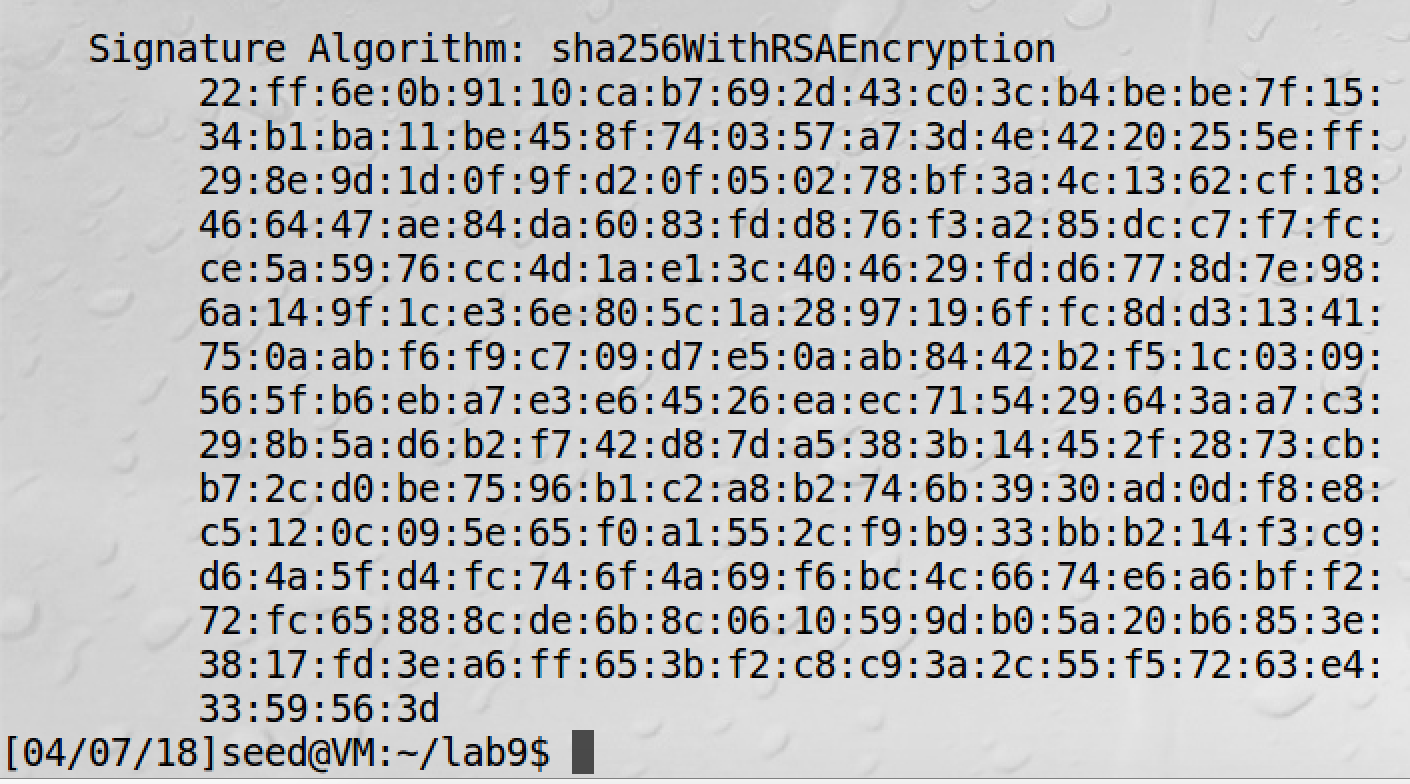
screenshot1. Run command “openssl s\_client -connect www.google.com:443 -showcerts” to get Google’s certificates. And we save the above two certificates into c0.pem and c1.pem



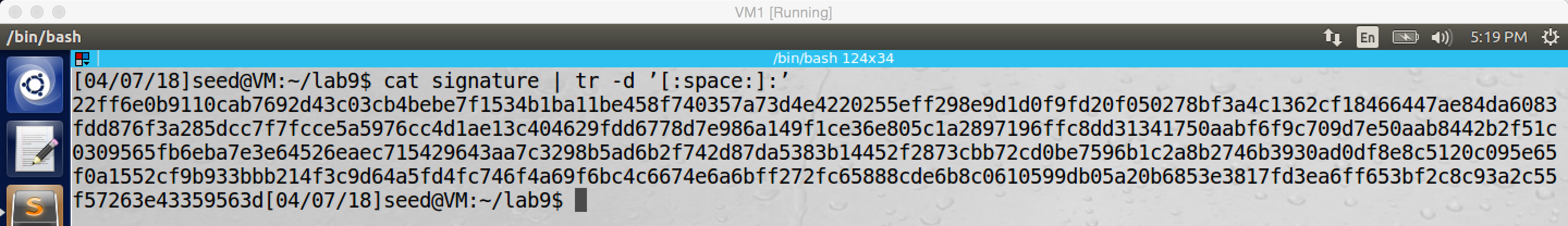
screenshot2. By running -modulus command, we can get “n”



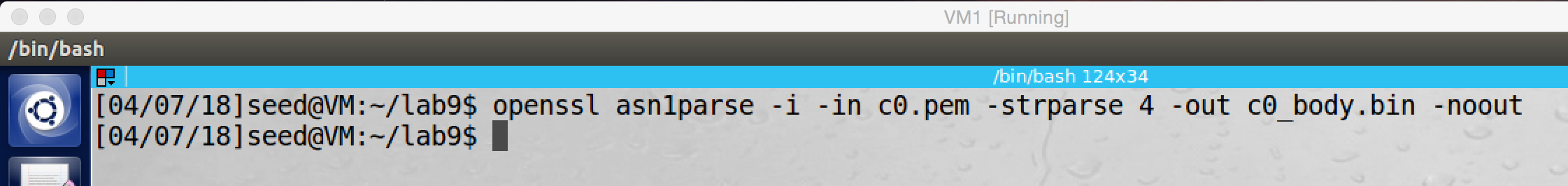
screenshot3. We get “e” by running command -noout



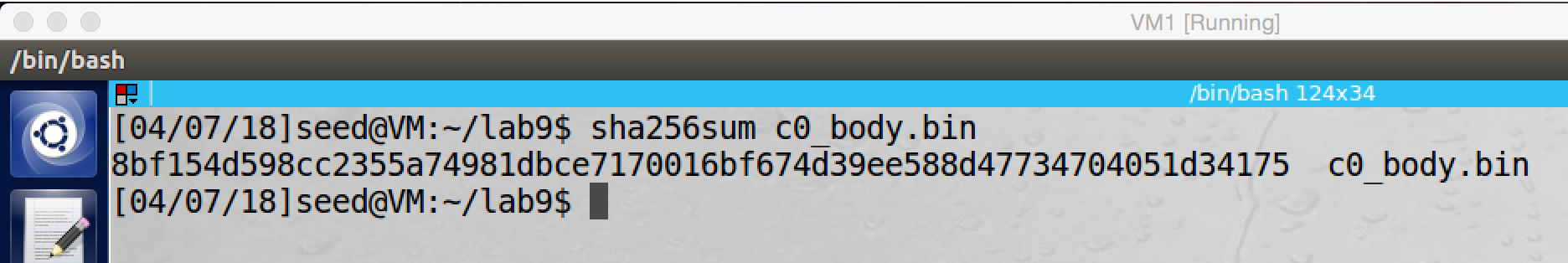
screenshot4. Get signature by using command “openssl x509 -in c0.pem -text -noout”



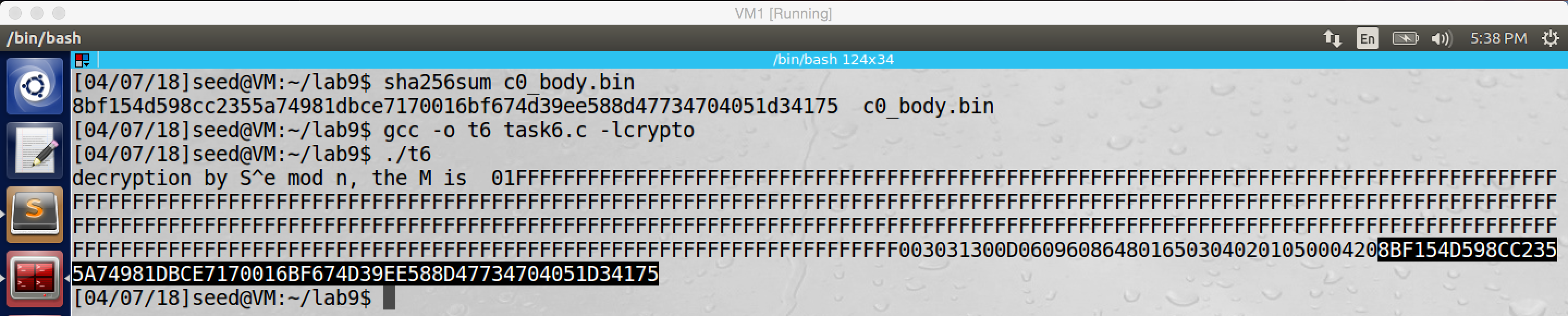
screenshot5. Removing “:” and “ ” in the signature



screenshto6. We can use the -strparse option to get the field from the offset 4, which will give us the body of the certificate, excluding the signature block



screenshot7. Calculate the hash value for the body of the certificate



screenshot8. After running the program, we get the message M, it is same as the hash value for the body of the certificate, so we verified that the signature is valid.

**Explanation and Observation:**

In this task, we want to verify the X.509 certificate of www.google.com. As the screenshot 1 to screenshot7 shows, I follow the steps on the lab description to get all necessary information, which includes the public key (n, e), the signature, the certificate body, and the hash value of the certificate body.

To verify the signature is valid or not, we can use formula (S^e mod n) to get the hash value of the certificate body, now we already have these values (S, e, n), and we also have the hash value of the certificate body. If the M generated by my program is same as the hash value of the certificate body, then we verified that the signature is valid.

For this purpose, we feed n, e, and signature S into the program task6.c, and then we compile and run it. As screenshot8 shows, the message M is printed. We see that the last 64bytes are exactly same as the hash value of the certificate body, so we verified that the signature is valid.