Faculty of Engineering and Technology

Electrical and Computer Engineering Department

INFORMATION SECURITY AND COMPUTER NETWORK LABORATORY ENCS5121

Report IV

Experiment # 5: RSA Public-Key Encryption and Signature

Student Name: Yara Darabumukho

Student Number: 1211269

Instructor: Dr. Ahmad Alsadeh

Teaching Assistant: Eng. Tariq Odeh

Section No: 1

Date: 10/11/202

Abstract

In this experiment the main aim is to understand and apply the RSA Public-Key Encryption and Signature and learn more about the digital signature. This report contains the sixth task from the lab, Manually Verifying an X.509 Certificate.

Table of Contents

Table of Figures

Step I

Download a certificate from a real web server.

In this step the following command runs:

- openssl s_client -connect youtube.com:443 -showcerts

which retrieve the server certificate.

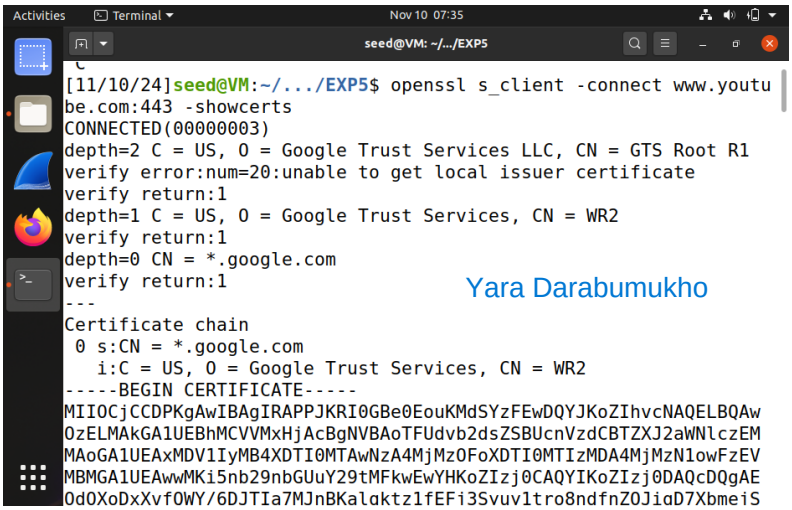

Figure 1: Certificate downloading.

This is the data shown by the pervious command, the first certificate “server’s certificate” coped in a file called c0.pem, and the second certificate “issuer’s certificate” coped in a file called c1.pem.

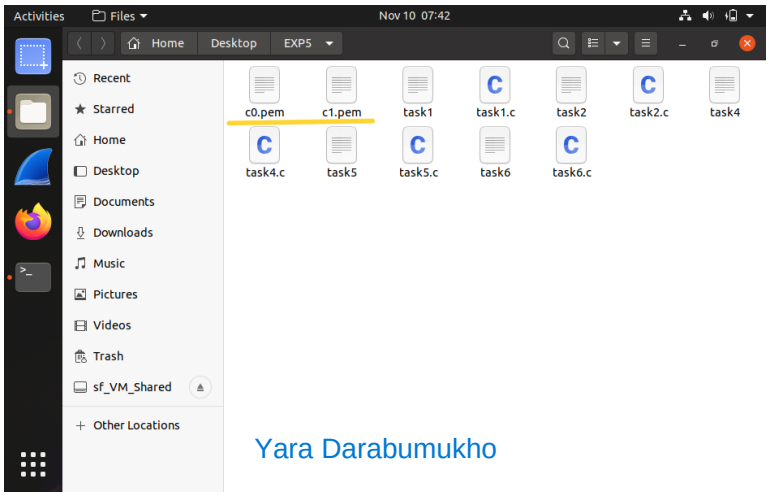

Figure 2: Files.

Step II

Extract the public key (e, n) from the issuer's certificate.

N

The value of N extracted by using the following command that applies at c1.pem file "issuer's certificate":

⇨ openssl x509 -in c1.pem -noout -modulus

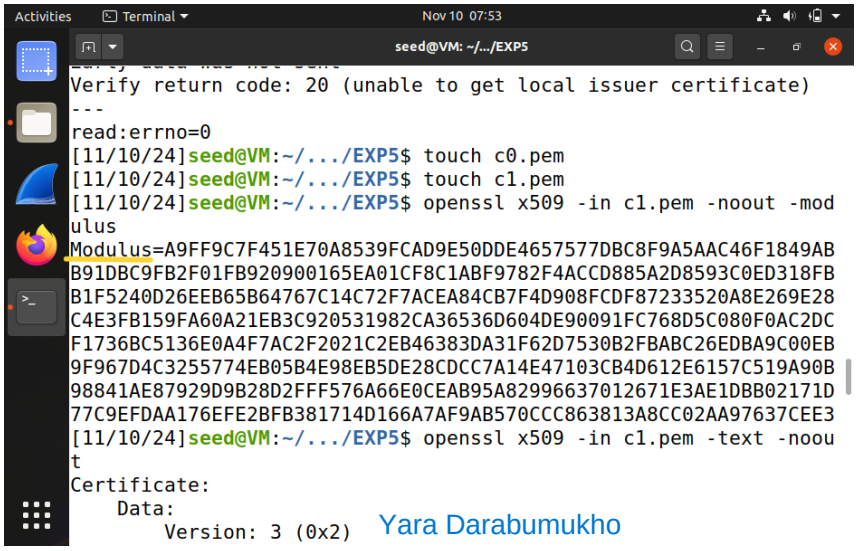

Figure 3: N extraction.

E

In the manual the value of E extracted by using the following command that applies at c1.pem file "issuer's certificate":

⇨ openssl x509 -in c1.pem -text -noout

Running the previous command shows a big amount of the data, so we can apply a grep command to find it directly insisted of find the key manually. So, this is the used command:

⇨ openssl x509 -in c1.pem -text -noout | grep "Exponent"

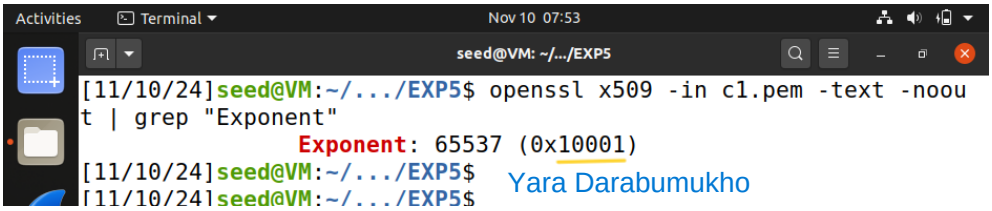

Figure 4: E extraction.

Step III

Extract the signature from the server's certificate.

In this step the following command runs:

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

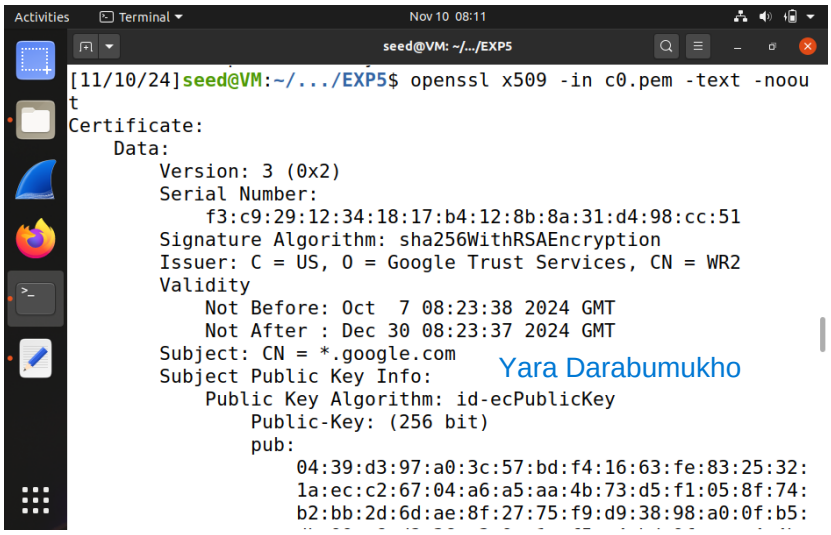

Figure 5: Command runs.

The command displays a large amount of data, and at the end of it the signature is found as shown in the following figure:

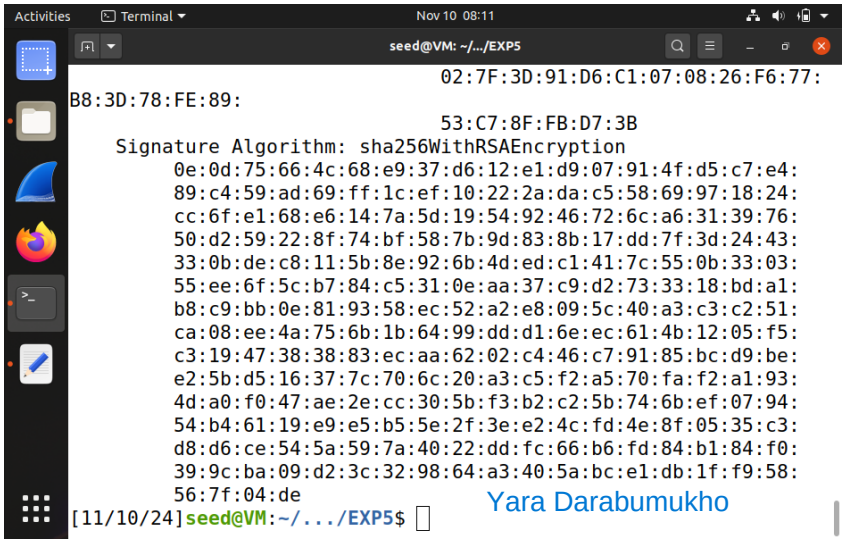

Figure 6: The signature.

The signature should be in specific format without any spaces or colons. So, the signature copes in a file called sig.txt, then apply the following command to remove them:

⇨ cat sig.txt | tr -d '[:space:]:'

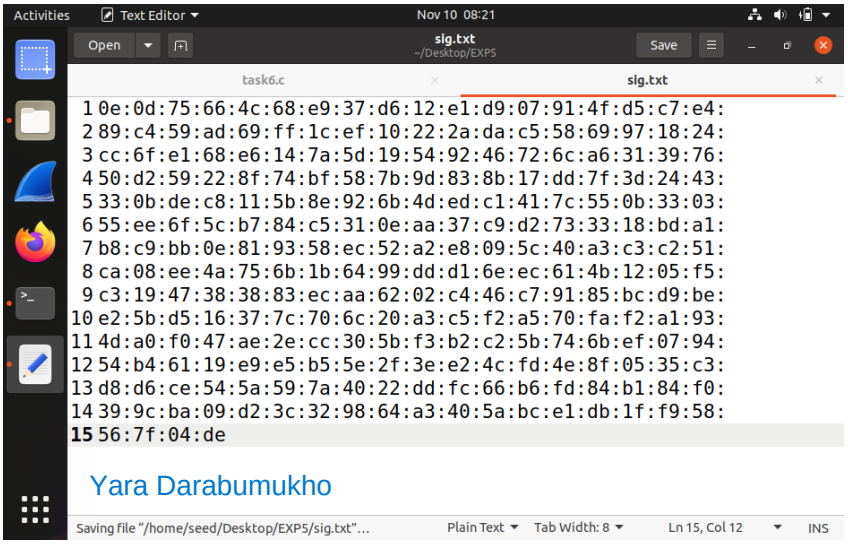

Figure 7: Signature temporary file.

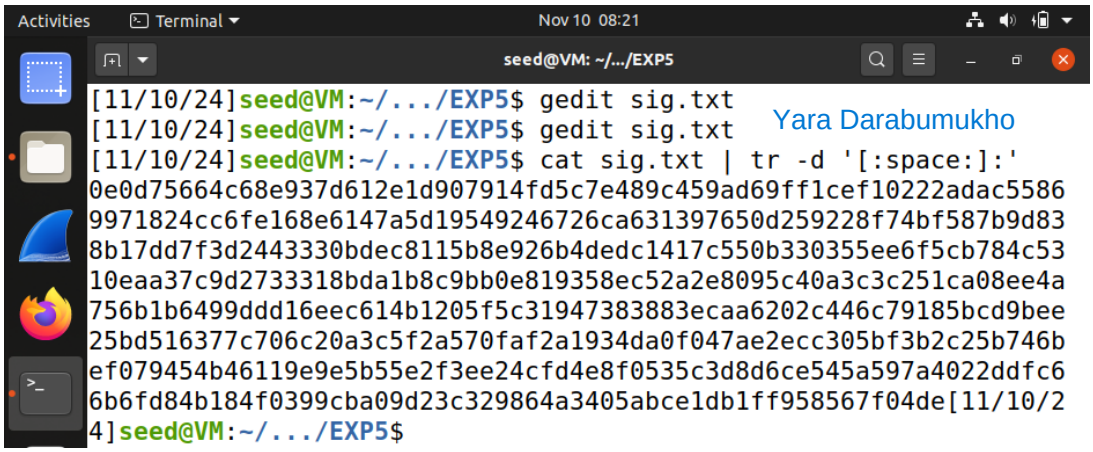

Figure 8: The required format.

Step IV

Extract the body of the server's certificate.

As a first step the offset should be determined using the following command:

- openssl asn1parse -I -in c0.pem

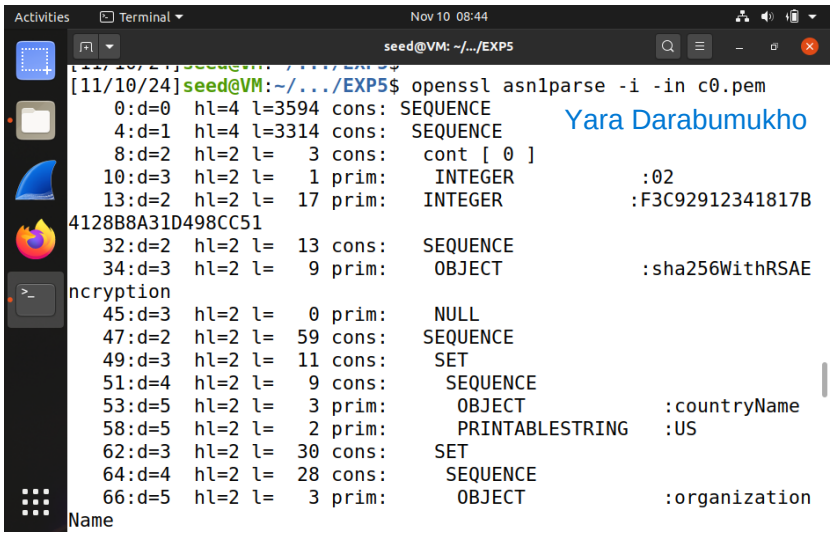

Figure 9: Offset command run.

As shown in the previous figure running the command displays a large amount of data, the number of the offset found almost in the end of the data as shown in the next figure:

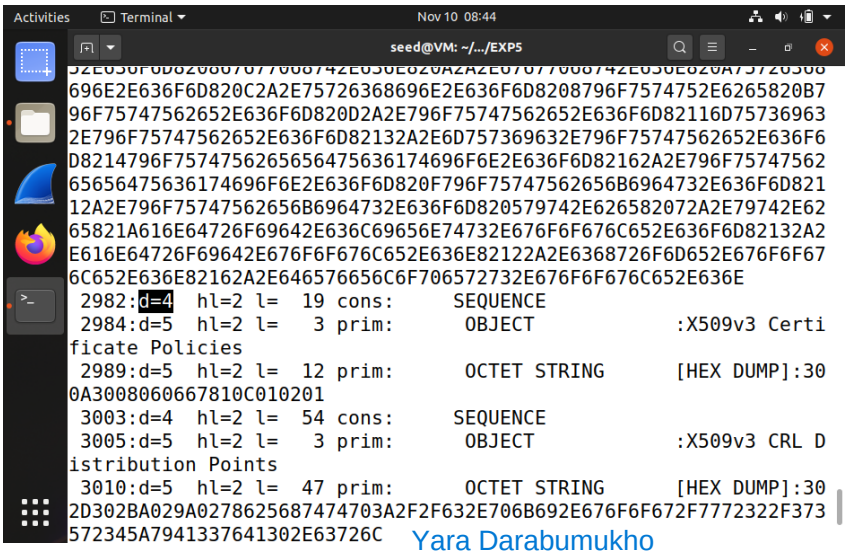

Figure 10: Offset number.

For X509 certificates the offset is always equal to 4.

In the second step the following commands run:

- openssl asn1parse -I -in c0.pem -strparse 4 -out c0_body.bin -noout
- sha256sum c0_body.bin

The first command gets the body of the certificate which stored in the c0_body.bin file, and then in the second command the hash calculated for that body.

- The result of the hash function will used in the comparison next step.

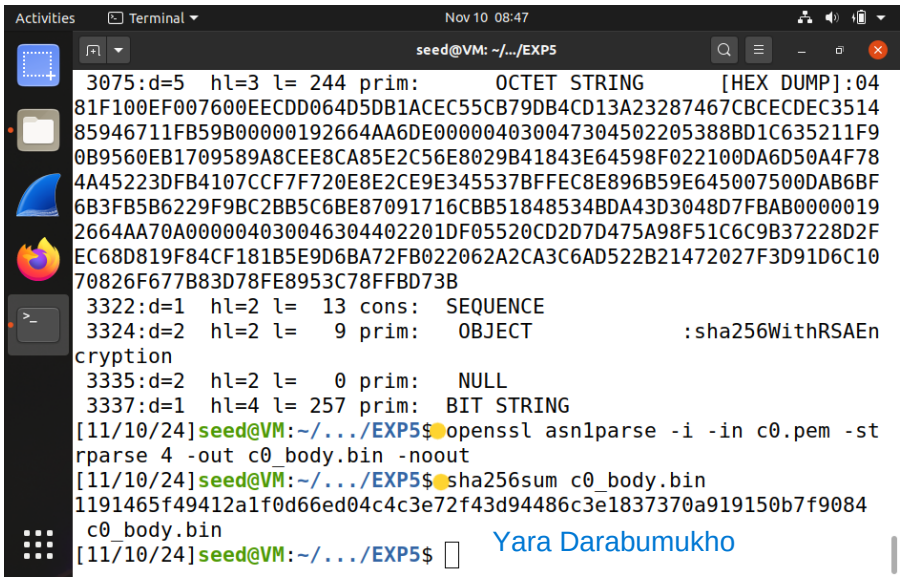

Figure 11: Hashed body result.

Step V

Verify the signature.

In this step the value of the computed hash using the public key (e, n) and the signature, which we modify and calculate from the certificate. It compared with the certificate value that generated in the previous step.

⇨ **Note:**
The computed hash generated using a code that will be attached at the end of the report in the appendix part.

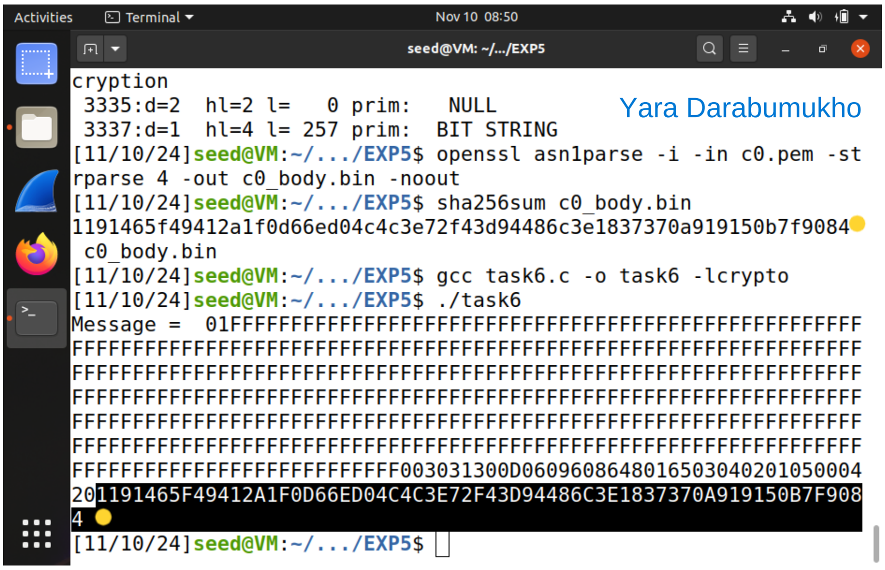

Figure 12: Verify Signature.

So, as shown in the previous figure the server certificate’s signature and the computed hash using the CA's public key same as each other. So, the certificate is valid, authentic, and was issued by the trusted CA. if they don’t match each other, then the certificate is not valid, which could imply tampering or an untrusted source.

Appendix

```
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 256

void printBN (char *msg, BIGNUM *a) // BN to Hex, then print the hex value
{
        char *num_string = BN_bn2hex(a); // bn => decimal
        printf ("%s %s\n", msg, num_string);
        OPENSSL_free(num_string);
}

int main()
{
        BN_CTX *ctx = BN_CTX_new();
        BIGNUM *s = BN_new();
        BIGNUM *n = BN_new();
        BIGNUM *e = BN_new();
        BIGNUM *msg = BN_new();
        // n, e, and s that extract from the certificate in steps number 2 and 3
        BN_hex2bn(&n,
"A9FF9C7F451E70A8539FCAD9E50DDE4657577DBC8F9A5AAC46F1849ABB91DBC9
FB2F01FB920900165EA01CF8C1ABF9782F4ACCD885A2D8593C0ED318FBB1F5240D
26EEB65B64767C14C72F7ACEA84CB7F4D908FCDF87233520A8E269E28C4E3FB159F
A60A21EB3C920531982CA36536D604DE90091FC768D5C080F0AC2DCF1736BC5136E
0A4F7AC2F2021C2EB46383DA31F62D7530B2FBABC26EDBA9C00EB9F967D4C32557
74EB05B4E98EB5DE28CDCC7A14E47103CB4D612E6157C519A90B98841AE87929D9
B28D2FFF576A66E0CEAB95A8299663701267 1E3AE1DBB02171D77C9EFDAA176EFE
2BFB381714D166A7AF9AB570CCC863813A8CC02AA97637CEE3");
        BN_hex2bn(&e, "10001");
        BN_hex2bn(&s,
"0e0d75664c68e937d612e1d907914fd5c7e489c459ad69ff1cef10222adac55869971824cc6fe
168e6147a5d19549246726ca631397650d259228f74bf587b9d838b17dd7f3d2443330bdec81
15b8e926b4dedc1417c550b330355ee6f5cb784c5310eaa37c9d2733318bda1b8c9bb0e819358
ec52a2e8095c40a3c3c251ca08ee4a756b1b6499ddd16eec614b1205f5c31947383883ecaa620
```

```
2c446c79185bcd9bee25bd516377c706c20a3c5f2a570faf2a1934da0f047ae2ecc305bf3b2c25b
746bef079454b46119e9e5b55e2f3ee24cfd4e8f0535c3d8d6ce545a597a4022ddfc66b6fd84b1
84f0399cba09d23c329864a3405abce1db1ff958567f04de");


        BN_mod_exp(msg, s, e, n, ctx); //msg = s^e mod n

        printBN("Message = ", msg);

        return 0;


}
```