Faculty of Engineering & Technology

Electrical & Computer Engineering Department

Applied Cryptography ENCS4320

RSA public key encryption and signature lab

Prepared by:

Hasan Hamed 1190496

Shorooq Ngjar 1192415

Instructor: Dr. Ahmad Alsadeh

Section: 1

Date: 27/8/2022

1. Abstract

The aim of this lab is to learn the RSA (Rivest-Shamir-Adleman) public-key encryption which is used widely for secure communications and to learn how to generate both the private and public keys and use them for encryption and decrypting messages and finally to learn the generation of digital signatures and digital signature verification and manually verifying an X.509 certificate.

Table of Contents

Table of Figures

2. Theory

2.1. Introduction

RSA (Rivest–Shamir–Adleman) is a public-key cryptosystem that is widely used for secure data transmission. It is also one of the oldest. The acronym "RSA" comes from the surnames of Ron Rivest, Adi Shamir and Leonard Adleman, who publicly described the algorithm in 1977. An equivalent system was developed secretly in 1973 at GCHQ (the British signals intelligence agency) by the English mathematician Clifford Cocks. That system was declassified in 1997. In a public-key cryptosystem, the encryption key is public and distinct from the decryption key, which is kept secret (private). An RSA user creates and publishes a public key based on two large prime numbers, along with an auxiliary value. The prime numbers are kept secret. Messages can be encrypted by anyone, via the public key, but can only be decoded by someone who knows the prime numbers. The security of RSA relies on the practical difficulty of factoring the product of two large prime numbers, the "factoring problem". Breaking RSA encryption is known as the RSA problem. Whether it is as difficult as the factoring problem is an open question. There are no published methods to defeat the system if a large enough key is used. RSA is a relatively slow algorithm. Because of this, it is not commonly used to directly encrypt user data. More often, RSA is used to transmit shared keys for symmetric-key cryptography, which are then used for bulk encryption–decryption.

2.2. Operation

The RSA algorithm involves four steps: key generation, key distribution, encryption, and decryption.

A basic principle behind RSA is the observation that it is practical to find three very large positive integers e, d, and n, such that with modular exponentiation for all integers m (with $0 \leq m < n$):

$$(m^e)^d \equiv \mathrm{m} \ (\mathrm{mod}\ \mathrm{n})$$

and that knowing e and n, or even m, it can be extremely difficult to find d. The triple bar (≡) here denotes modular congruence. (In simple terms, modular congruence means that when you divide (me)d by n and divide m by n, each has the same remainder.)

In addition, for some operations it is convenient that the order of the two exponentiations can be changed and that this relation also implies

$$(m^d)^e \equiv \mathrm{m} \ (\mathrm{mod}\ \mathrm{n})$$

RSA involves a public key and a private key. The public key can be known by everyone and is used for encrypting messages. The intention is that messages encrypted with the public key can only be decrypted in a reasonable amount of time by using the private key. The public key is represented by the integers n and e, and the private key by the integer d (although n is also used during the decryption process, so it might be considered to be a part of the private key too). m represents the message (previously prepared with a certain technique explained below).

2.2.1. Key Generation

- Need to generate: modulus n, public key exponent e, private key exponent d
- Approach
  - Choose p,q (large random prime numbers)
  - n = pq (should be large)
  - Choose e, $1 < e < \varphi(n)$ and e is relatively prime to $\varphi(n)$
  - Find d, ed mod $\varphi(n) = 1$
- Result
  - (e,n) is public key
  - d is private key

2.2.2. Key Distribution

Suppose that Bob wants to send information to Alice. If they decide to use RSA, Bob must know Alice's public key to encrypt the message, and Alice must use her private key to decrypt the message.

To enable Bob to send his encrypted messages, Alice transmits her public key (n, e) to Bob via a reliable, but not necessarily secret, route. Alice's private key (d) is never distributed.

2.2.3. Encryption

After Bob obtains Alice's public key, he can send a message M to Alice. To do it, he first turns M (strictly speaking, the un-padded plaintext) into an integer m (strictly speaking, the padded plaintext), such that $0 \leq m < n$ by using an agreed-upon reversible protocol known as a padding scheme. He then computes the ciphertext c, using Alice's public key e, corresponding to

$$c \equiv m^e \pmod{n}$$

This can be done reasonably quickly, even for very large numbers, using modular exponentiation. Bob then transmits c to Alice. Note that at least nine values of m will yield a ciphertext c equal to m, but this is very unlikely to occur in practice.

2.2.4. Decryption

Alice can recover m from c by using her private key exponent d by computing

$$c^d \equiv (m^e)^d \equiv \text{m}$$

Given m, she can recover the original message M by reversing the padding scheme.

2.3. Digital Signature

A digital signature is a mathematical scheme for verifying the authenticity of digital messages or documents. A valid digital signature, where the prerequisites are satisfied, gives a recipient very high confidence that the message was created by a known sender (authenticity), and that the message was not altered in transit (integrity).

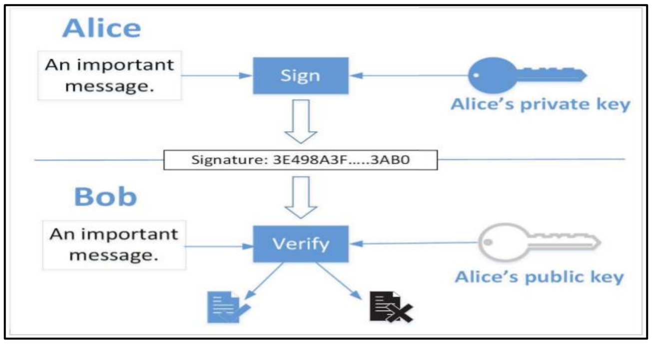

Figure 2.1: Signing Messages

For a message m that needs to be signed:

$$Digital\ signature\ = m^d\ mod\ n$$

In practice, message may be long resulting in long signature and more computing time. Instead, we generate a cryptographic hash value from the original message, and only sign the hash.

Attackers cannot generate a valid signature from a modified message because they do not know the private key. If attackers modify the message, the hash will change and it will not be able to match with the hash produced from the signature verification as will be noticed in the lab.

2.4. X.509 Certificate

An X.509 certificate is a digital certificate based on the widely accepted International Telecommunications Union (ITU) X.509 standard, which defines the format of public key infrastructure (PKI) certificates. They are used to manage identity and security in internet communications and computer networking. They are unobtrusive and ubiquitous, and we encounter them every day when using websites, mobile apps, online documents, and connected devices.

One of the structural strengths of the X.509 certificate is that it is architected using a key pair consisting of a related public key and a private key. Applied to cryptography, the public and private key pair is used to encrypt and decrypt a message, ensuring both the identity of the sender and the security of the message itself. The most common use case of X.509-based PKI is Transport Layer Security (TLS)/Secure Socket Layer (SSL), which is the basis of the HTTPS protocol, which enables secure web browsing. But the X.509 protocol is also applied to code signing for application security, digital signatures, and other critical internet protocols. In this lab, we will manually verify an X.509 certificate using our program

3. Procedure and Results

3.1. Creating BN Functions

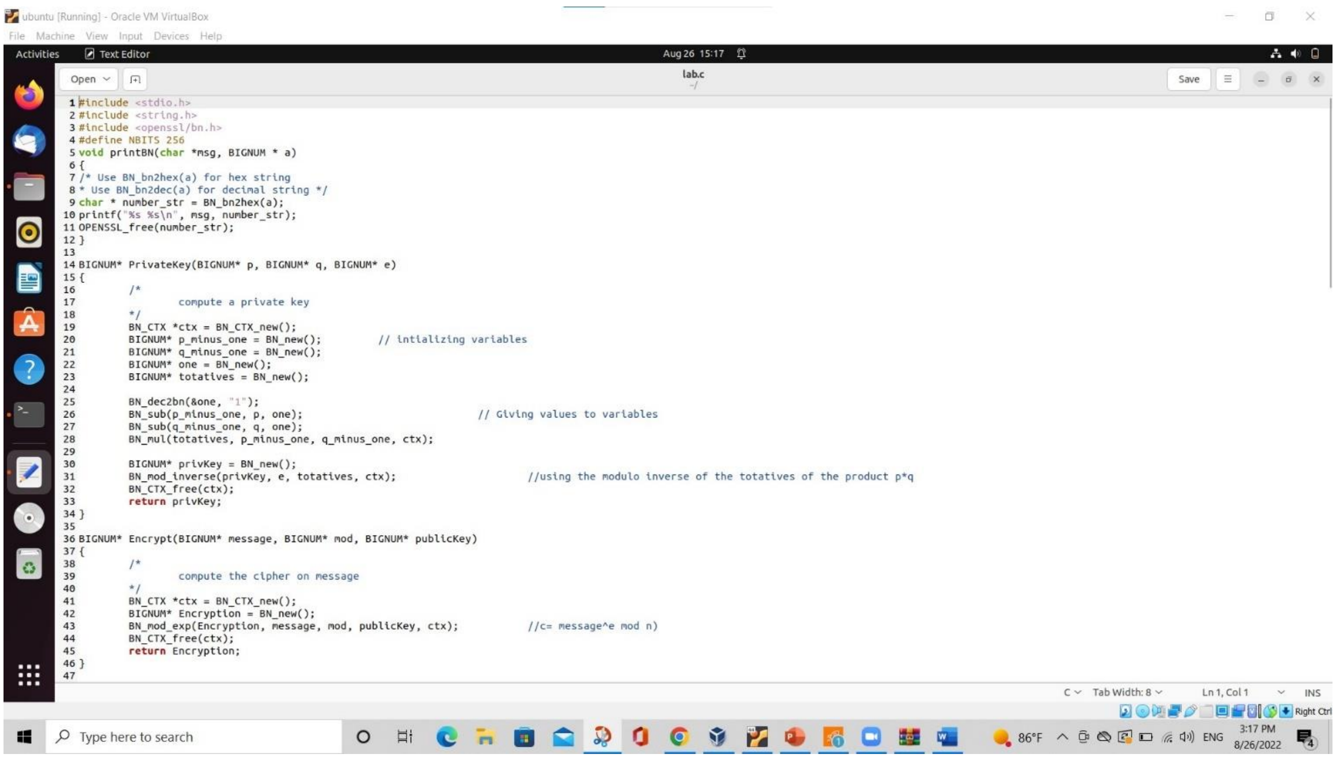

Figure 3.1: printBN, Private Key Generation, Encryption Functions Codes.

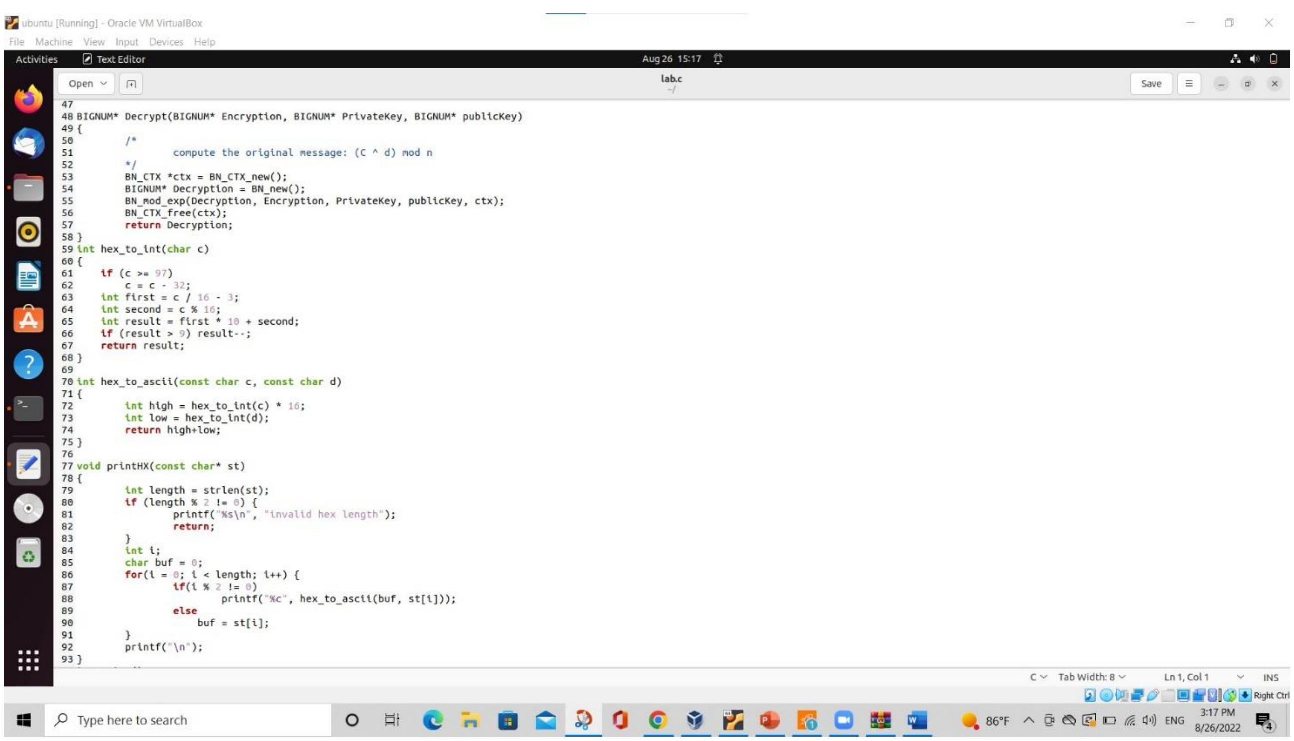

Figure 3.2: Hex2int, Hex2Ascii and printHX Functions Codes

printBN function was built to print the big numbers values using dynamic allocating, Hex2int was used to convert Hexadecimal values to integers and Hex2Ascii was used to change Hexadecimal values to Ascii which both were used in printHX function the change Hexadecimal values into characters for decryption. Private Key function was used to generate the private key for the RSA using the inverse totatives method and the encryption and decryption functions were built using the methods discussed in the theoretical part.

3.2. Task 1: Deriving the Private Key

3.2.1. Code

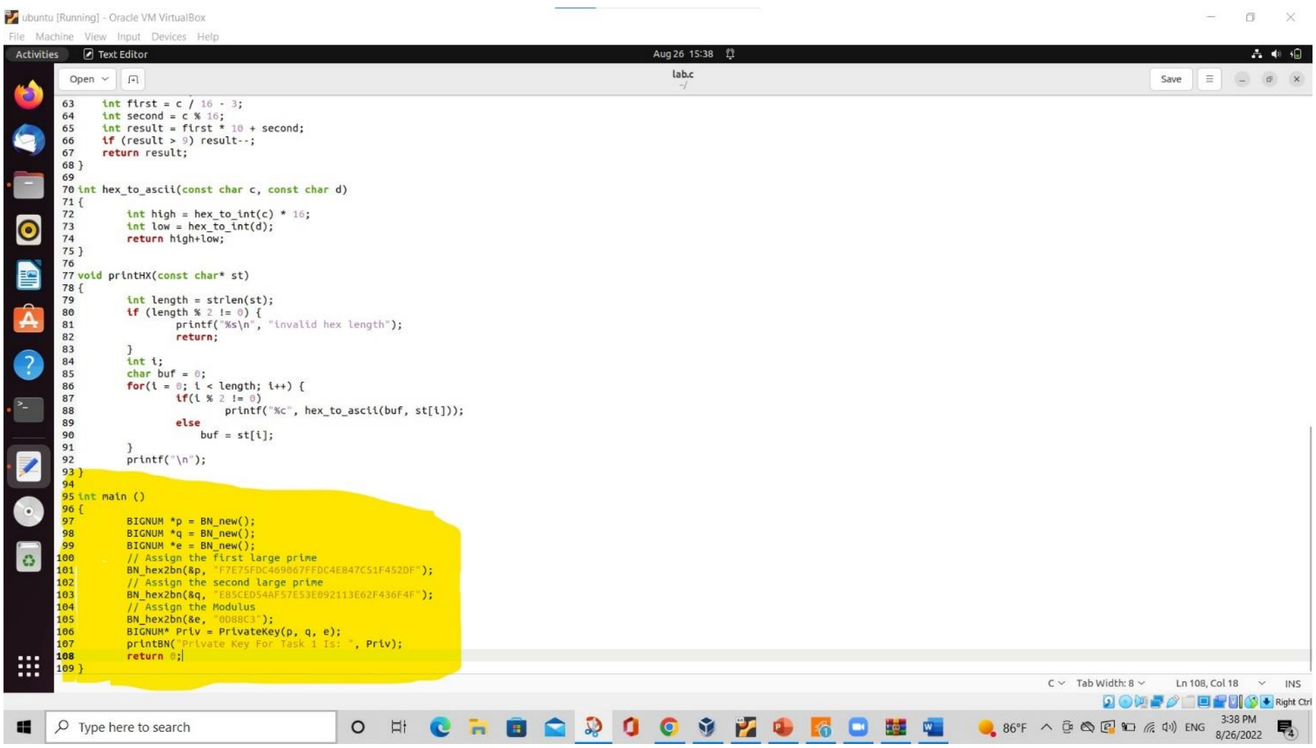

Figure 3.3: Deriving the Private Key Code

3.2.2. Code Output

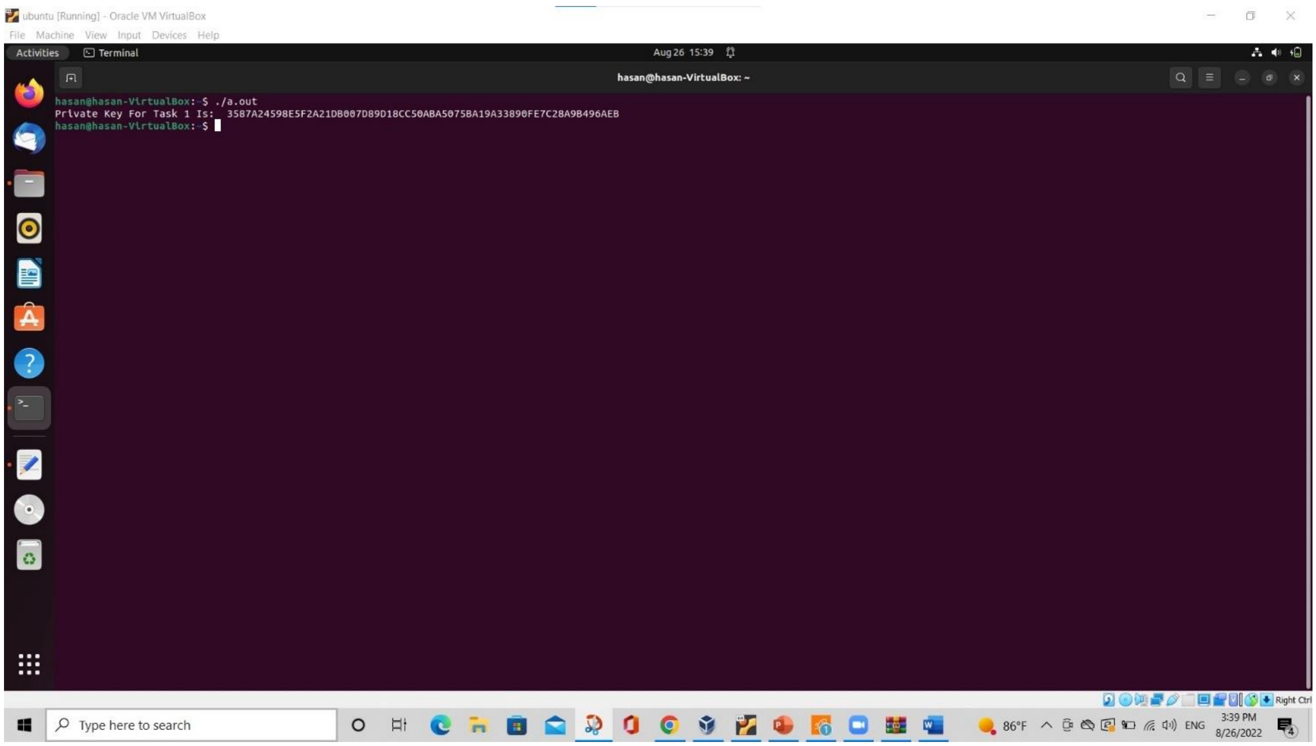

Figure 3.4: Task 1 Output

Given the values of *p,q and e* the value of private key, *d* was calculated after changing all of the values to Bignum as shown in Figure 3.3. The output of the code after it gets excuted will be the private key as shown in Figure 3.4.

3.3. Task 2: Encrypting a Message

3.3.1. Code

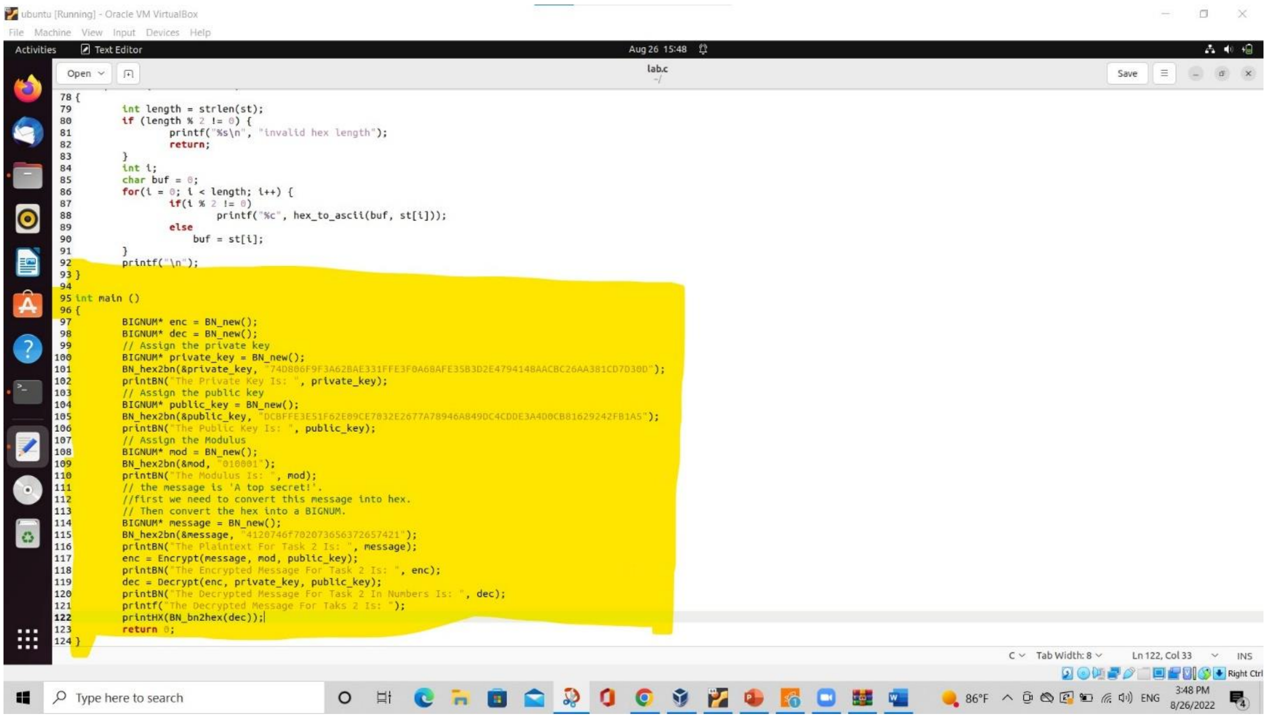

Figure 3.5: Encrypting a Message Code

3.3.2. Output

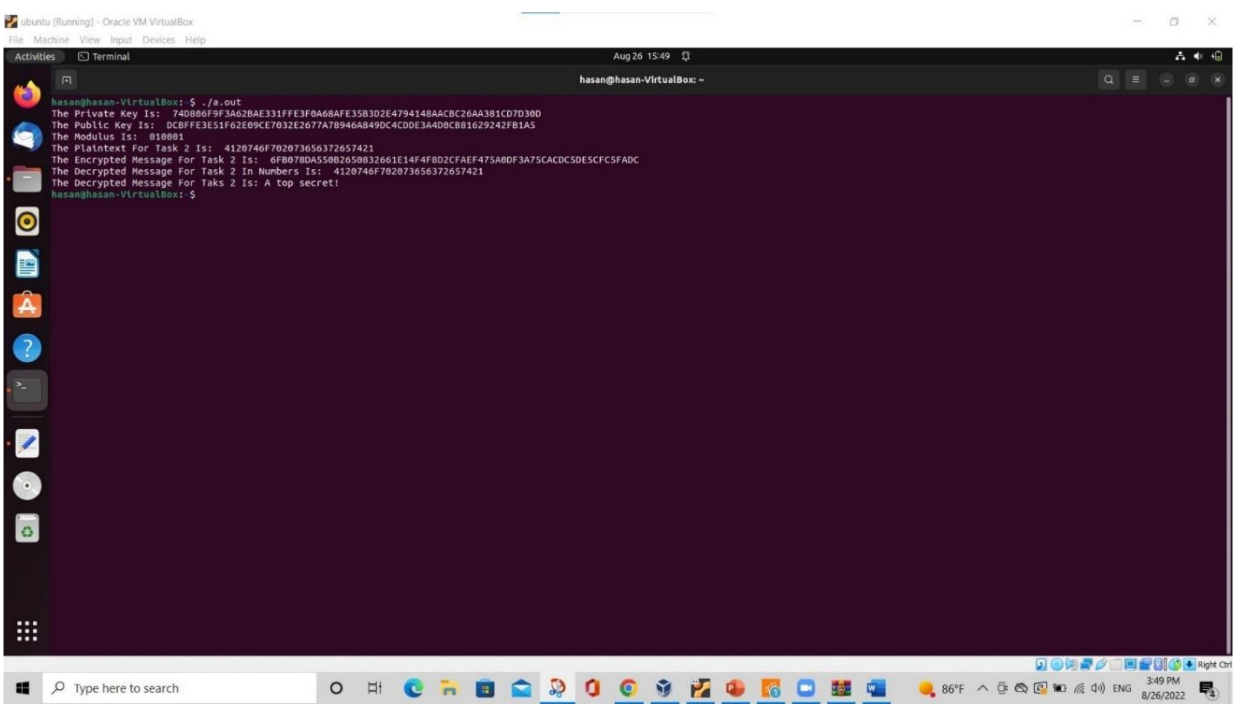

Figure 3.6: Encrypting a Message Output

In this task, the message "A top secret!" was encrypted after converting the text into hexadecimal using any converter and using the encryption function the message was encrypted using the given public keys, and after encrypting the message the given private key was used to decrypt the message to assure that it was encrypted correctly as shown in Figure 3.5.

And as shown in the output shown in Figure 3.6 after excuting the code that both the decrypted message and the original message were identical which means that the encryption was correct and it also shows the encryption of the message.

3.4. Task 3: Decrypting a Message

3.4.1. Code

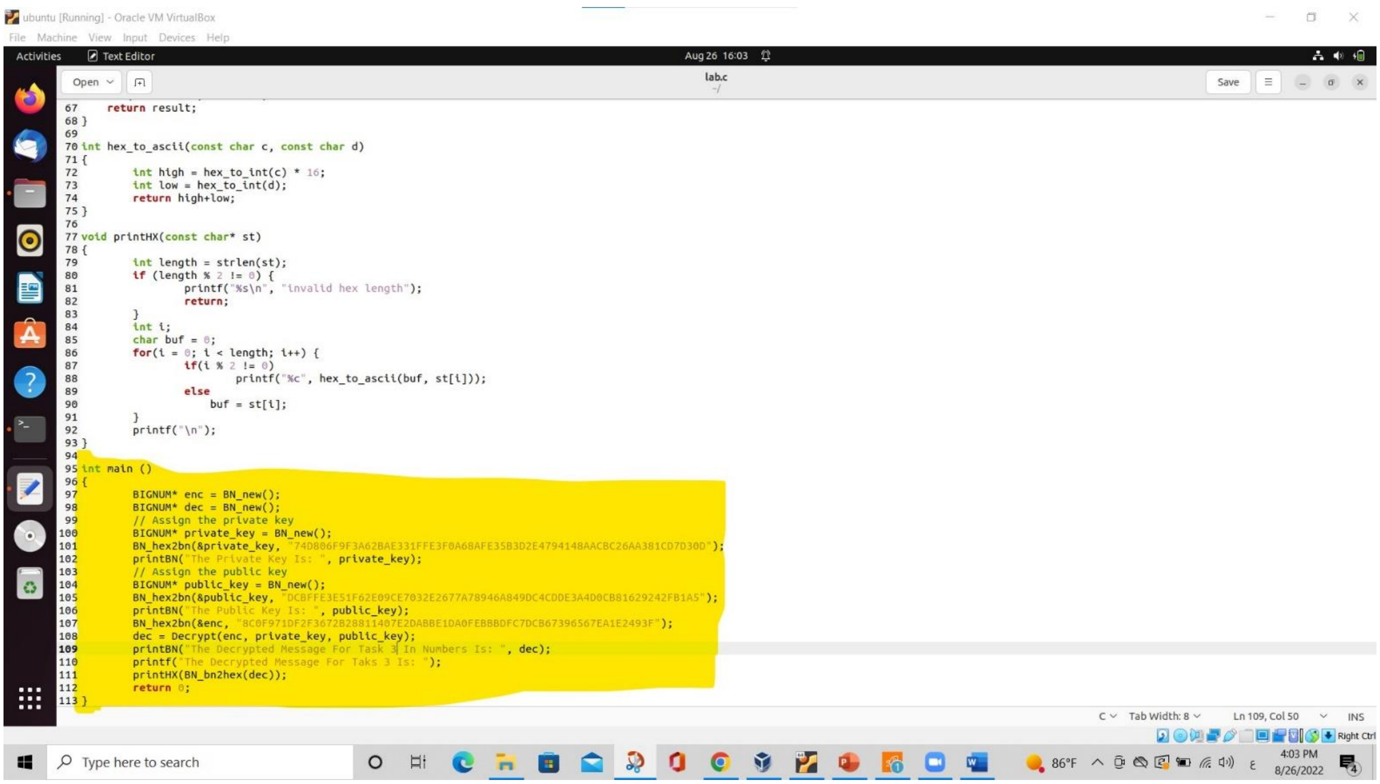

Figure 3.7: Decrypting a Message Code

3.4.2. Output

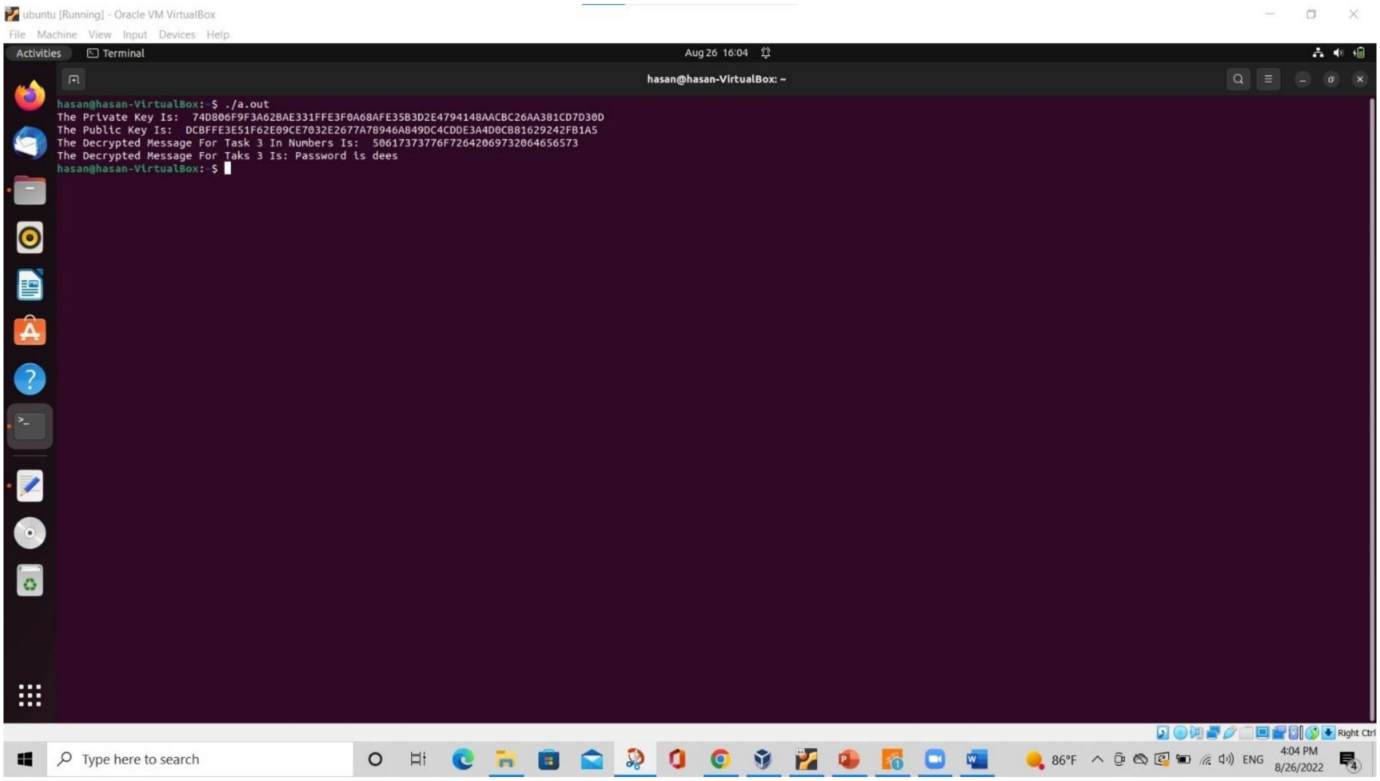

Figure 3.8: Decrypting a Message Output

The public and private keys used in this task are the same as the ones used in Task 2. The cipher c given was decrypted using the decryption function shown in the code in Figure 3.7 and then the output of the decryption was converted from hex to text as shown in the output shown in Figure 3.8 after the code execution.

3.5. Task 4: Signing a Message

3.5.1. Code

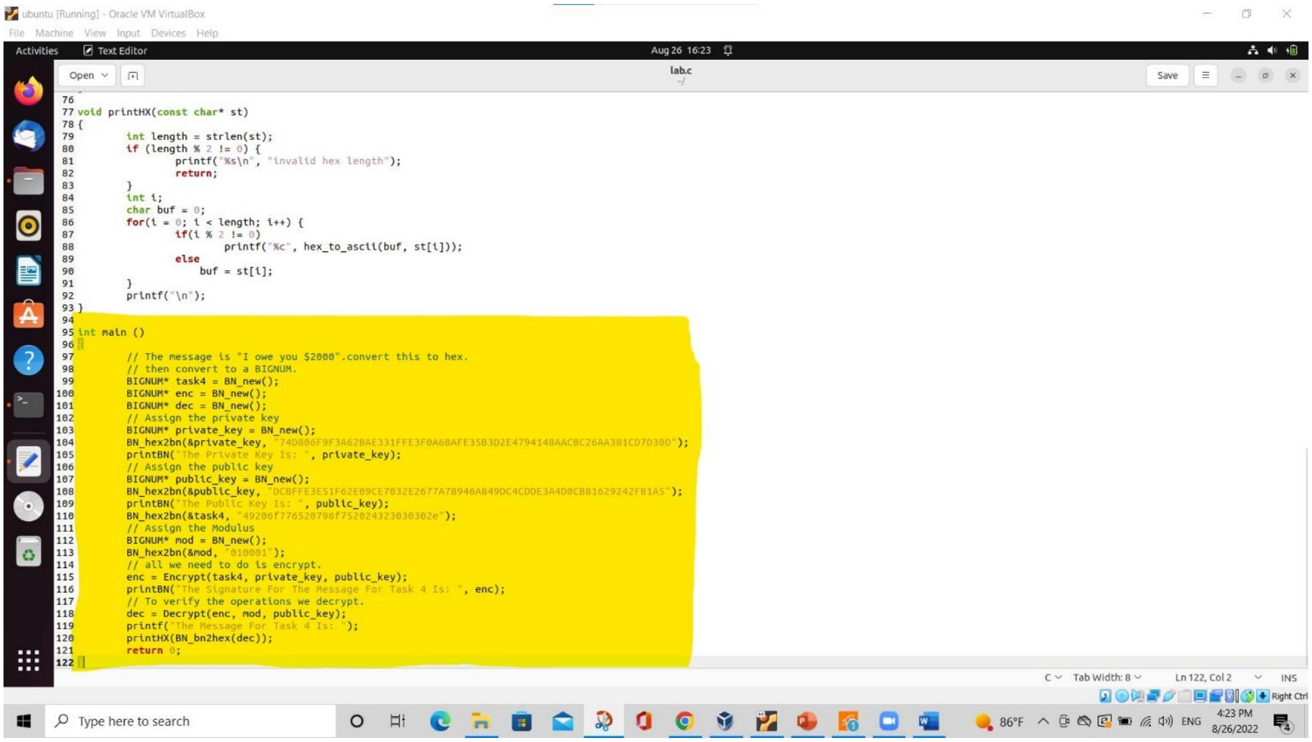

Figure 3.9: Signing a Message Code

3.5.2. Output

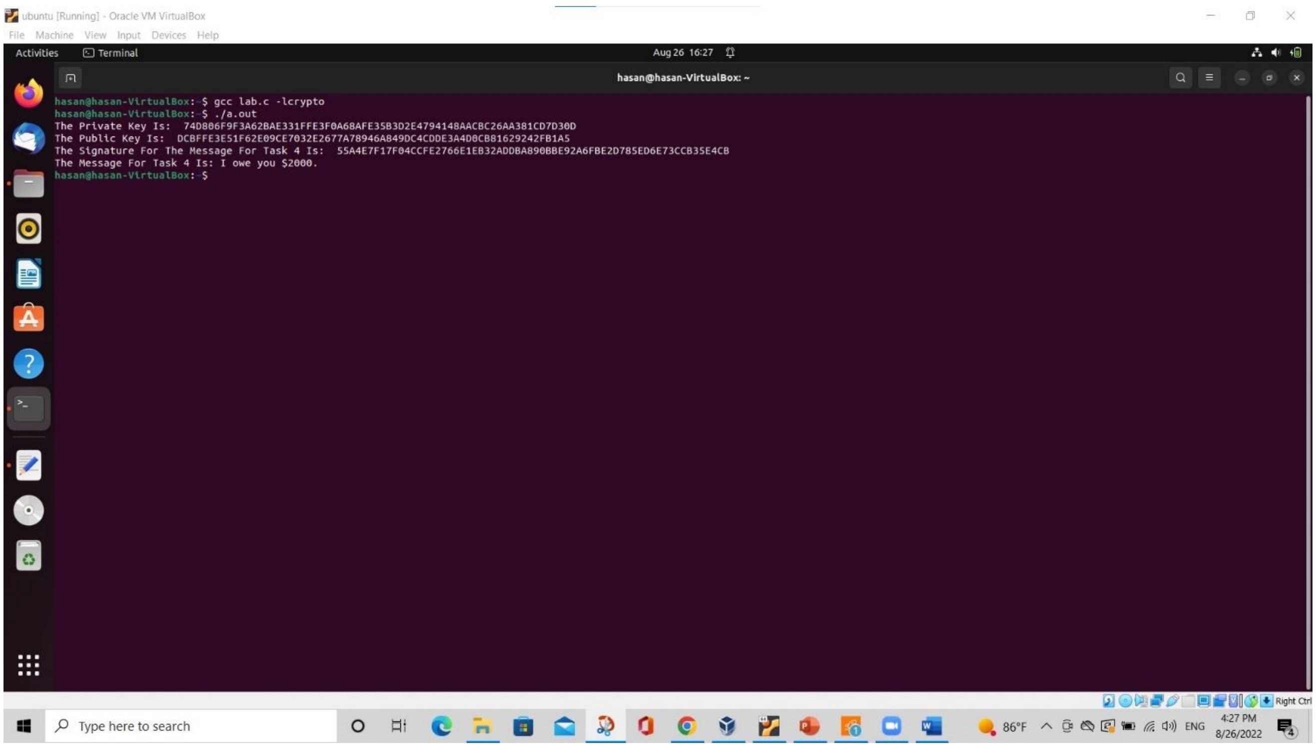

Figure 3.10: Signing a Message Output

The public and private keys used in this task are the same as the ones used in Task 2. A signature for the following message M = “I owe you $2000” was generated.

The hex value of the message “I owe you $2000.” was observed using websites. The code shown in Figure 3.9 was excuted and it outputs is shown in Figure 3.10.

Now, the hex value of the message “I owe you $3000.” was observed and the message was modified as shown in Figure 3.11, and after running the code the output was observed as shown in Figure 3.12.

3.5.3. Code

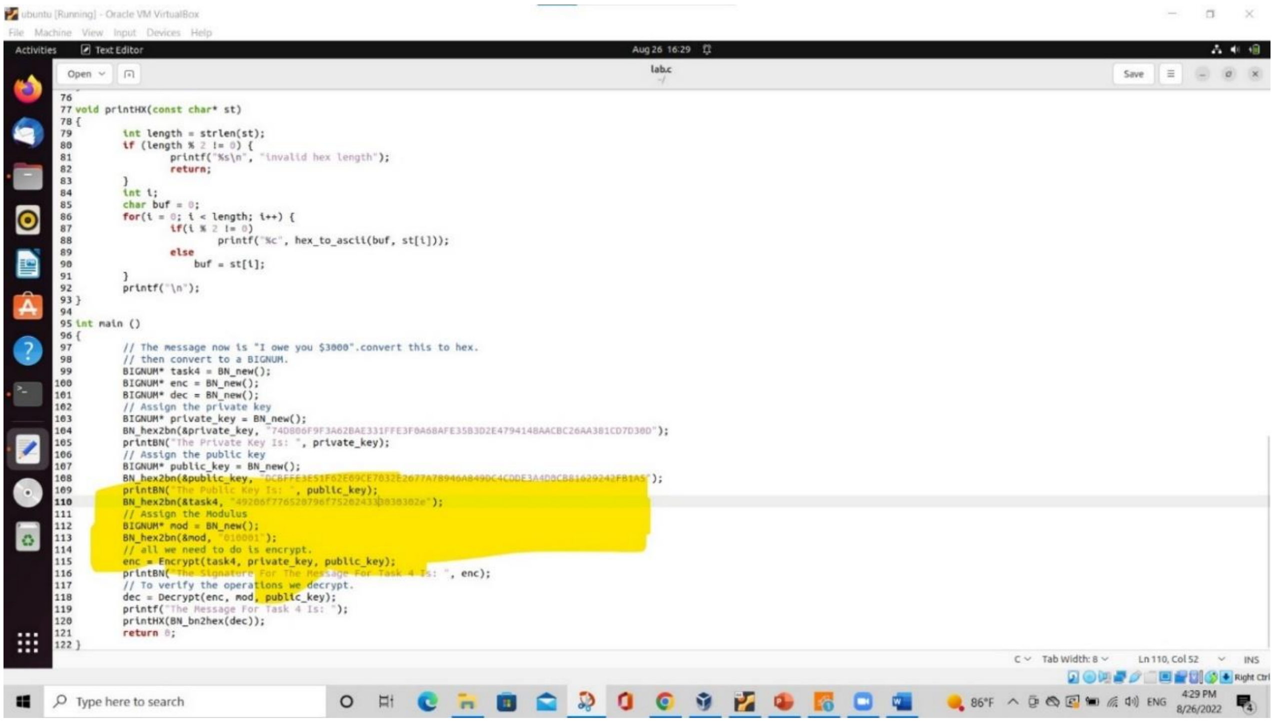

Figure 3.11: Modifying One Bit of M

3.5.4. Output

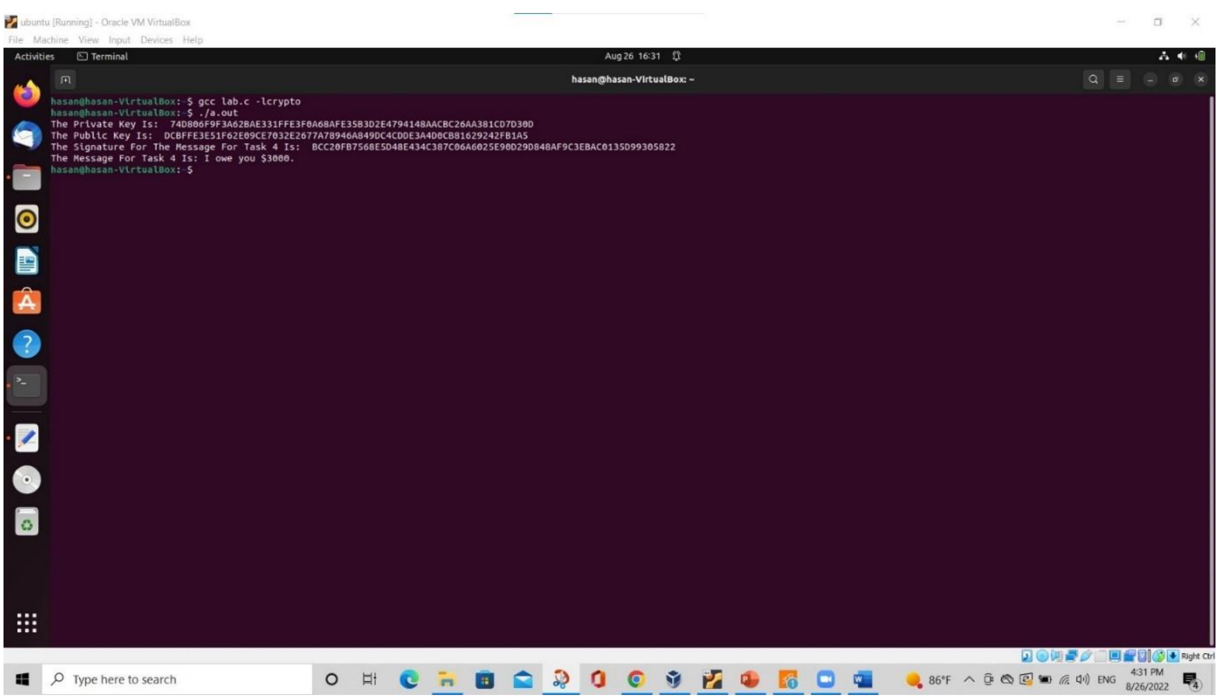

Figure 3.12: Signature for Modified Message

It was surprising that only one byte of difference in the message their signatures were completely different.

3.6. Task 5: Verifying a Signature

3.6.1. Code

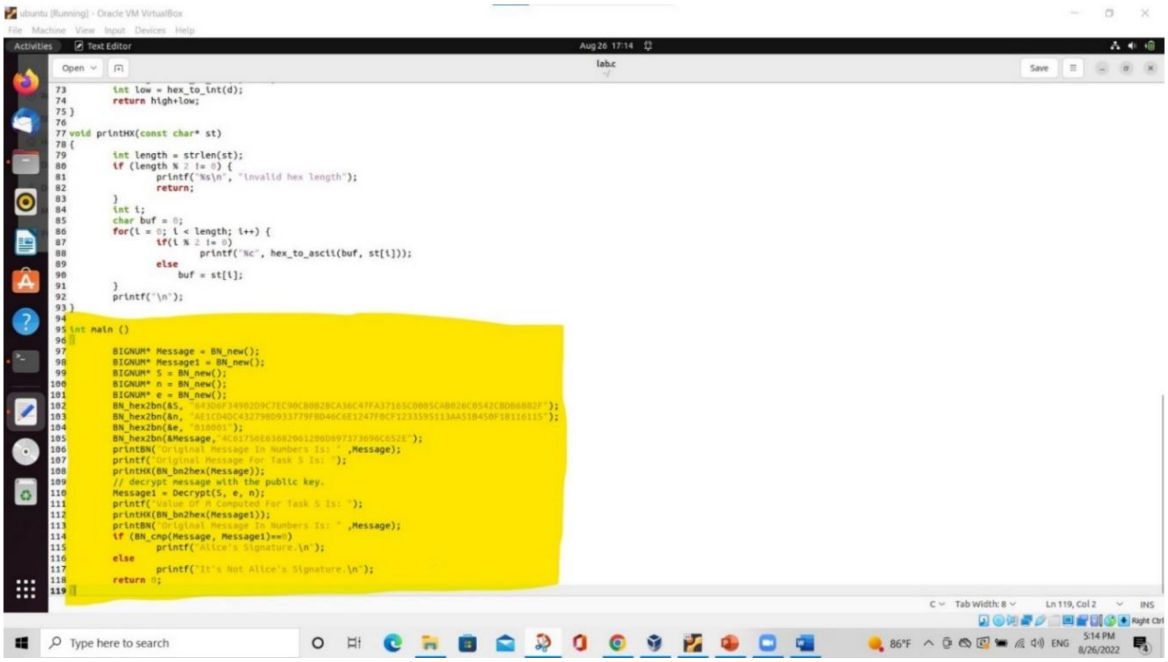

Figure 3.13: Verifying a Signature Code

3.6.2. Output

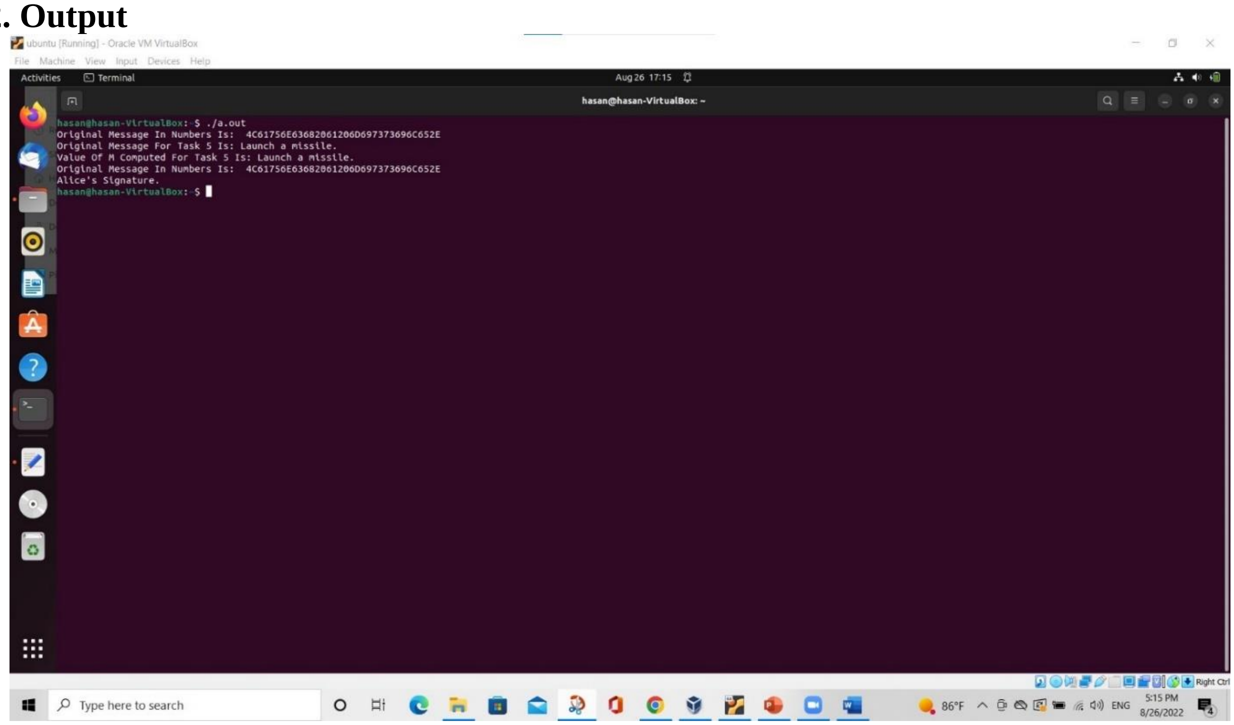

Figure 3.14: Verifying a Signature Output

Bob receives a message M = "Launch a missile." from Alice, with her signature S. Alice's public key is (e, n). The signature was verified using the decryption function by decrypting the signature using the public key as shown in the code shown in Figure 3.13, and by comparing the decrypted message with the original message if they were equal which means it's Alice signature else it's not. The output of the code after executing it is shown in Figure 3.14.

The last byte of the signature was changed from 2F to 3F and the code was modified as shown in Figure 3.15. The output was observed as shown in Figure 3.16.

3.6.3. Code

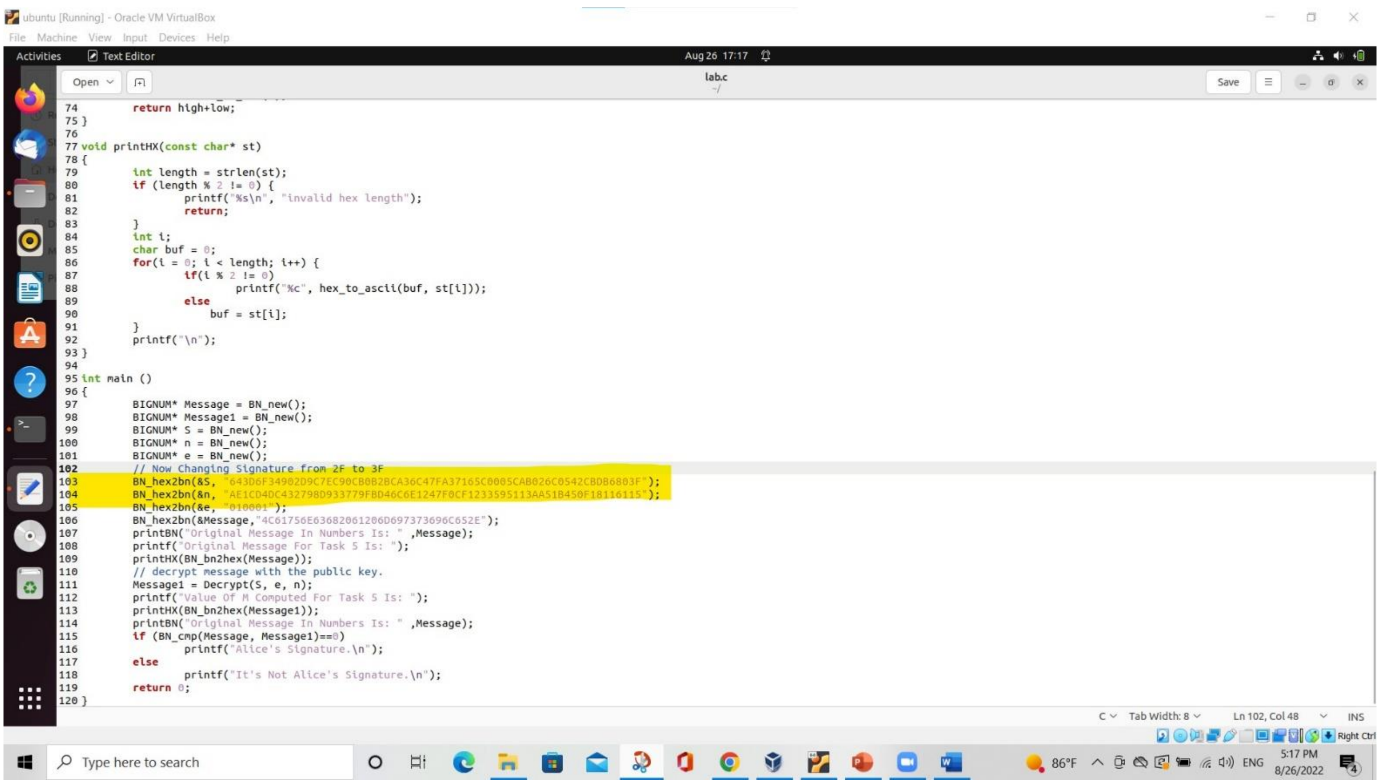

Figure 3.15: Verifying a Modified Signature Code

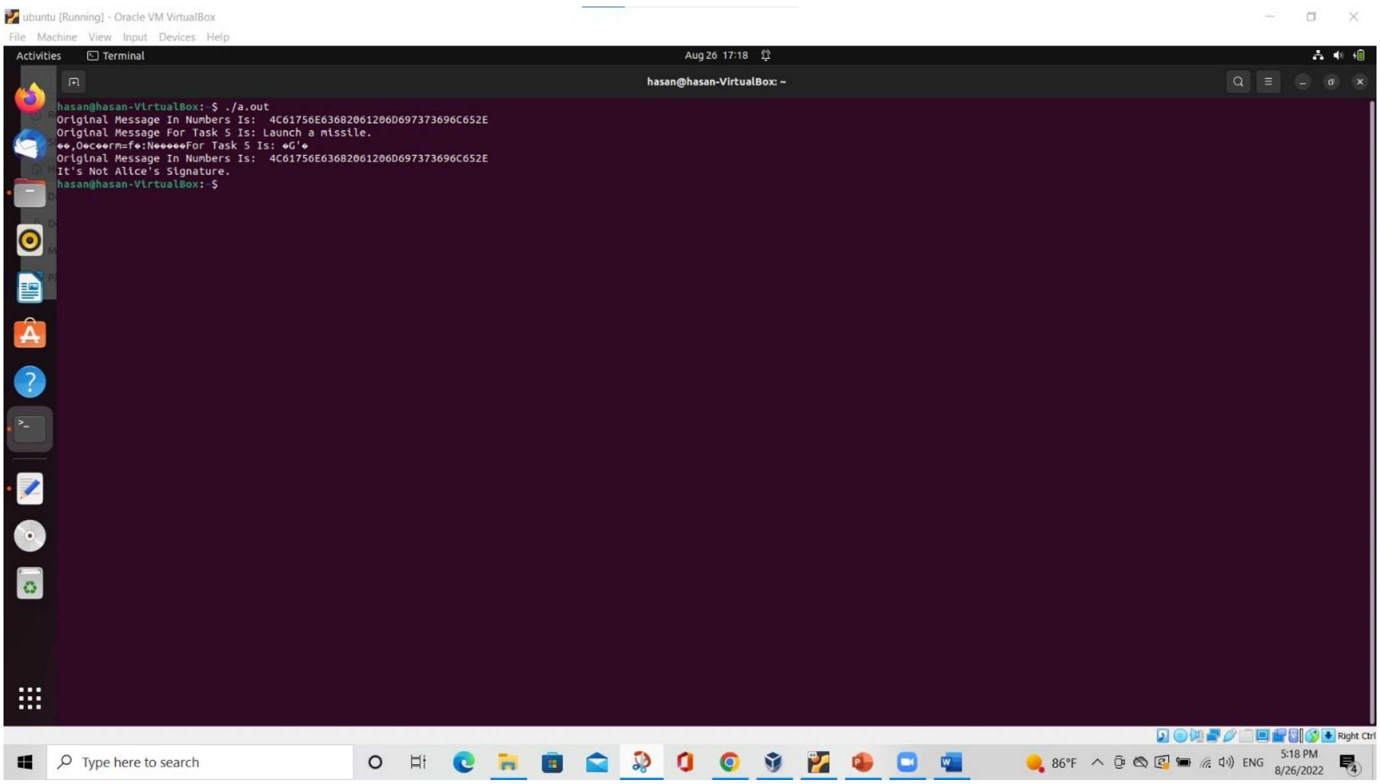

Figure 3.16: Verifying a Modified Signature Output

It was surprising that changing one byte of the signatures results changing the whole decrypted message completely which results a failed verification.

3.7. Task 6: Manually Verifying an X.509 Certificate

3.7.1. Download a certificate from a real web server

The certificate of www.blank.com was observed using the following command as shown in Figure 3.17 and 3.18:

$ openssl s_client -connect www.example.org:443 -showcerts

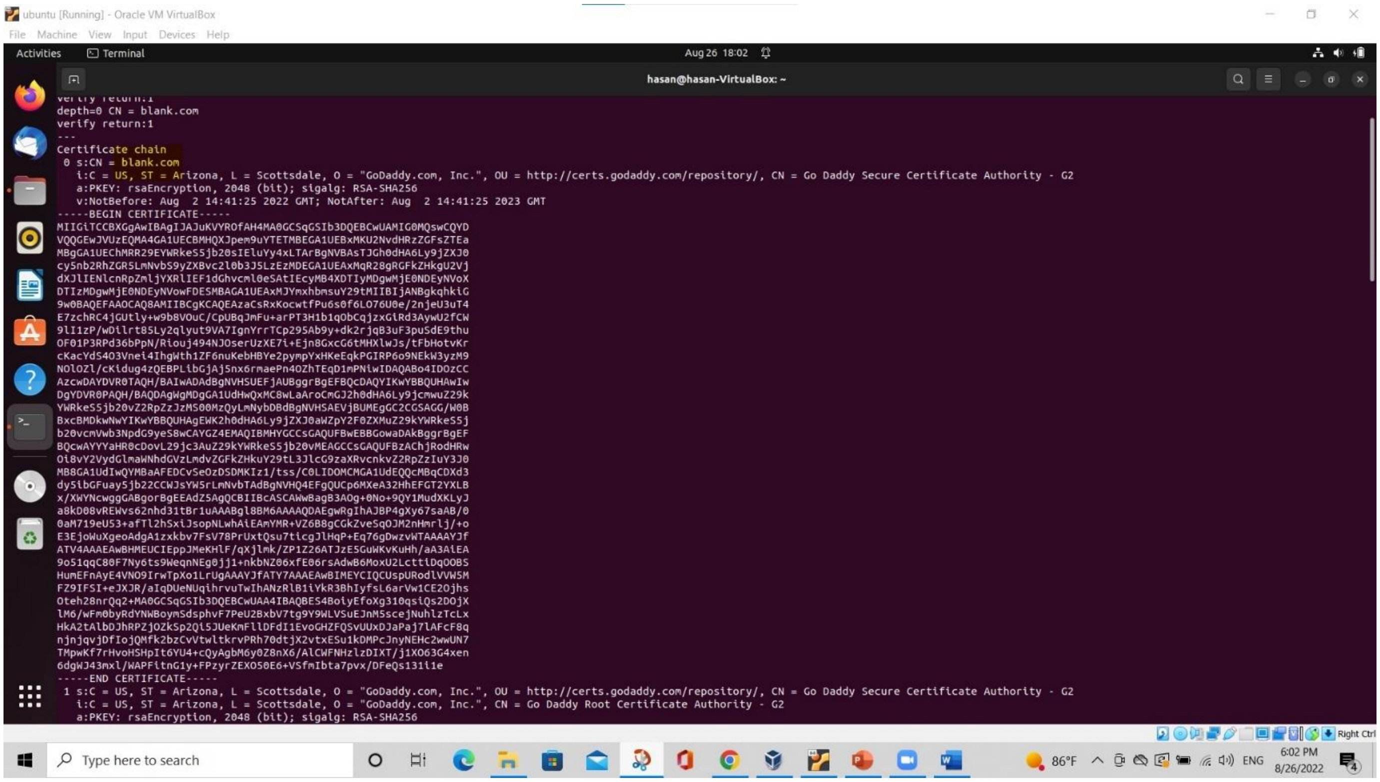

Figure 3.17: First Certificate of www.blank.com

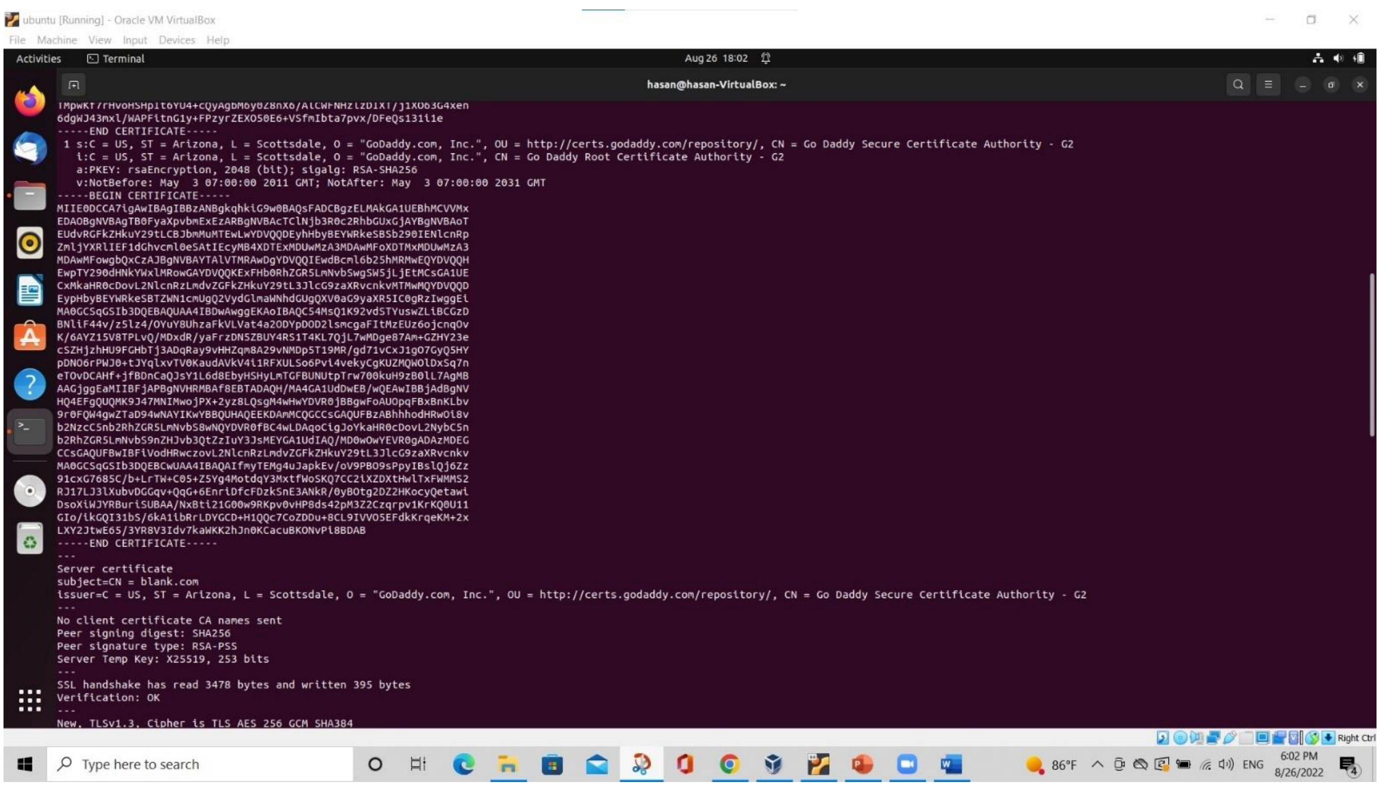

Figure 3.18: Second Certificate of www.blank.com

Each of the certificate (the text between the line containing "Begin CERTIFICATE" and the line containing "END CERTIFICATE", including these two lines) were copied and pasted to a file. c0.pem and c1.pem.

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

For modulus (n) the following command was used:

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

For the exponent, all the fields were printed out using the following command:

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

After printing all the fields, e was observed and n was observed using the first command both results are shown in figure 3.19 and 3.20.

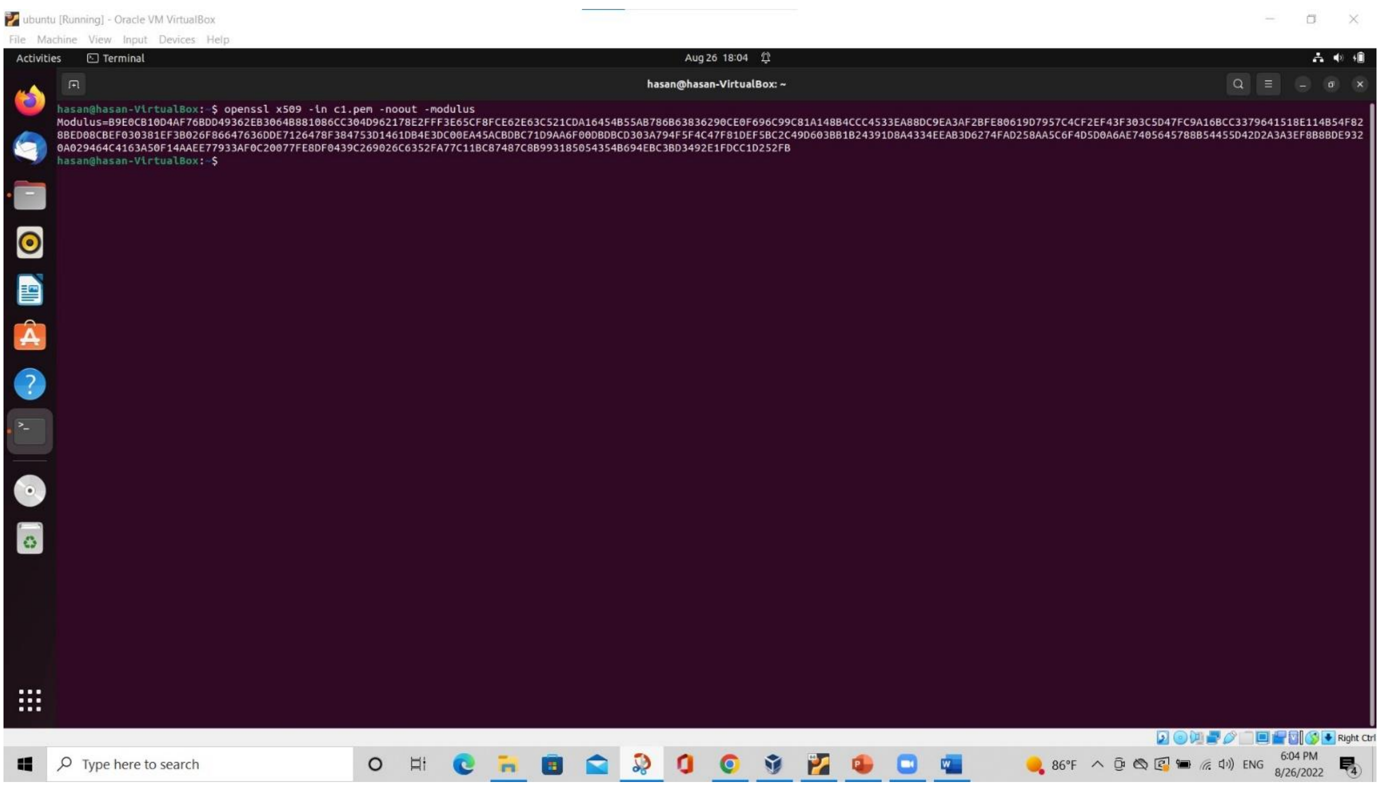

Figure 3.19: Finding Modulus (n)

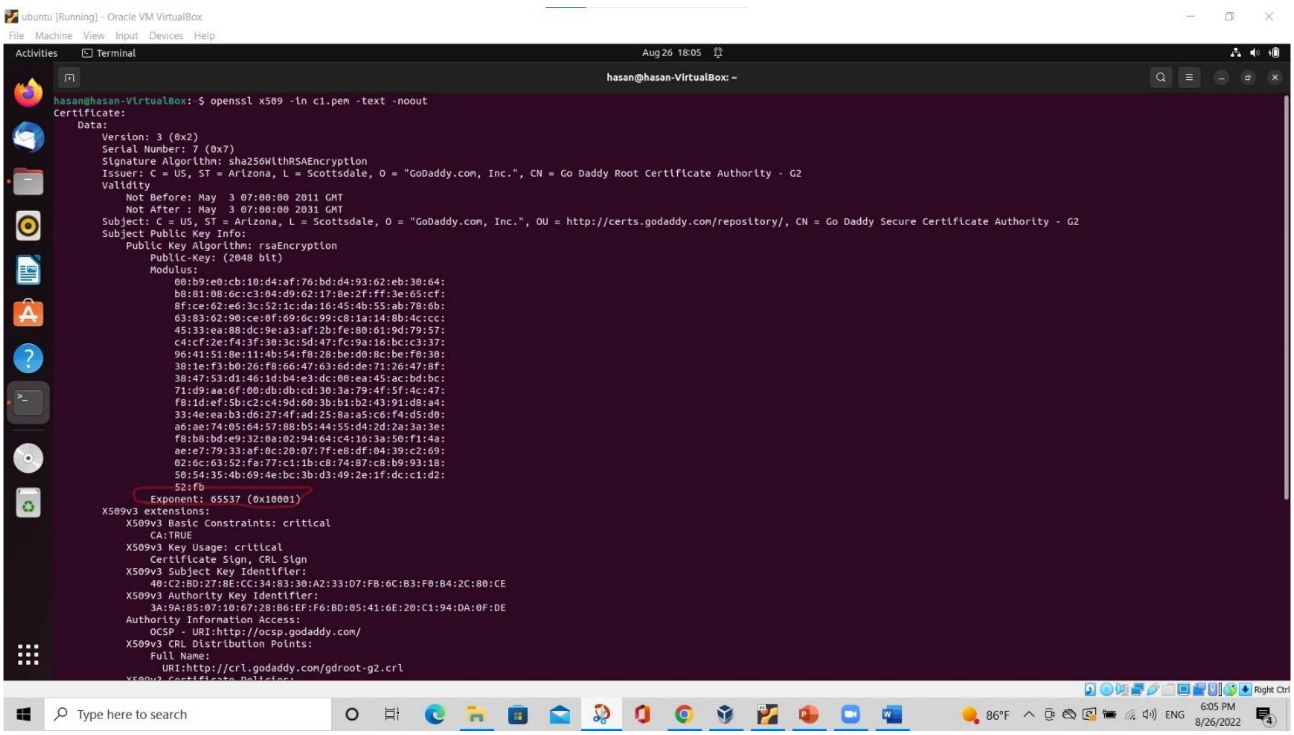

Figure 3.20: Finding Exponent

3.7.3. Extract the signature from the server's certificate

The following command was used to print out all the fields:

openssl x509 -in c0.pem -text –noout

And then the signature block was copied and pasted into a file called signature as shown in Figure 3.21.

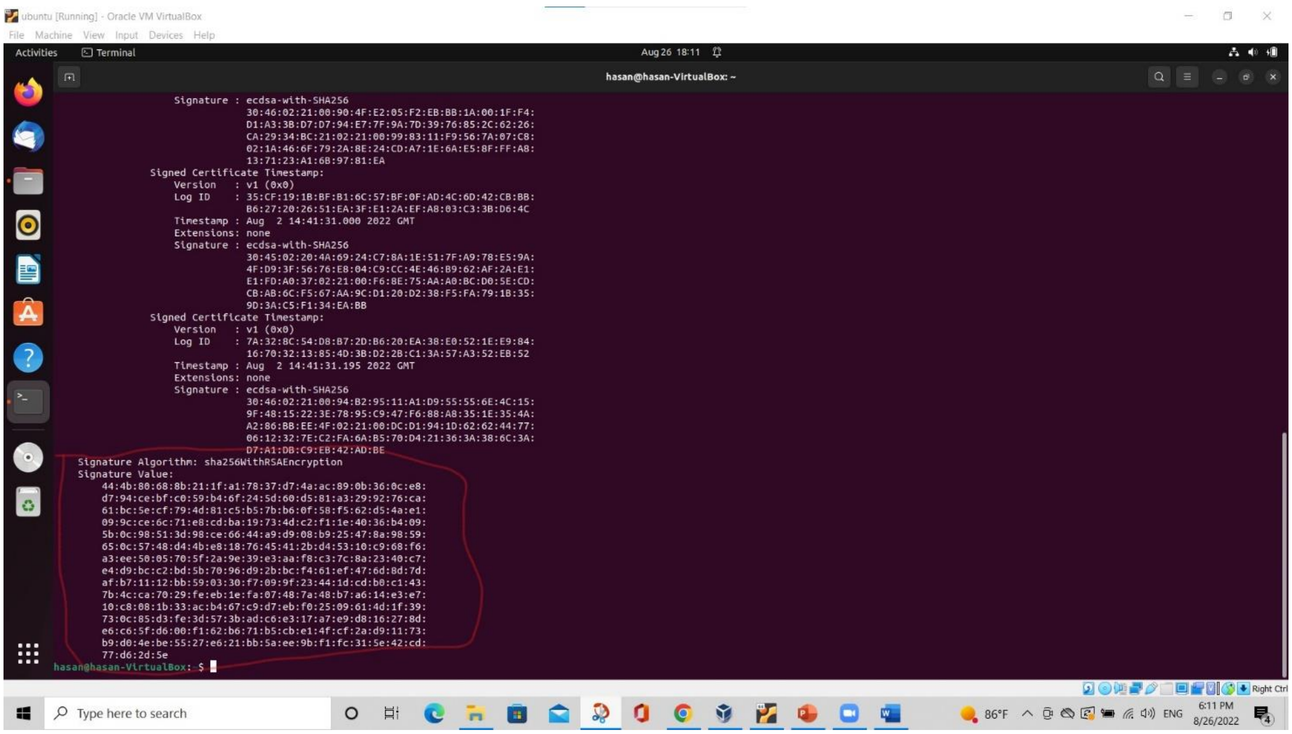

Figure 3.21: Extracting Signature Value

All the colons and spaces were removed from the signature using the tr command as shown in Figure 3.22.

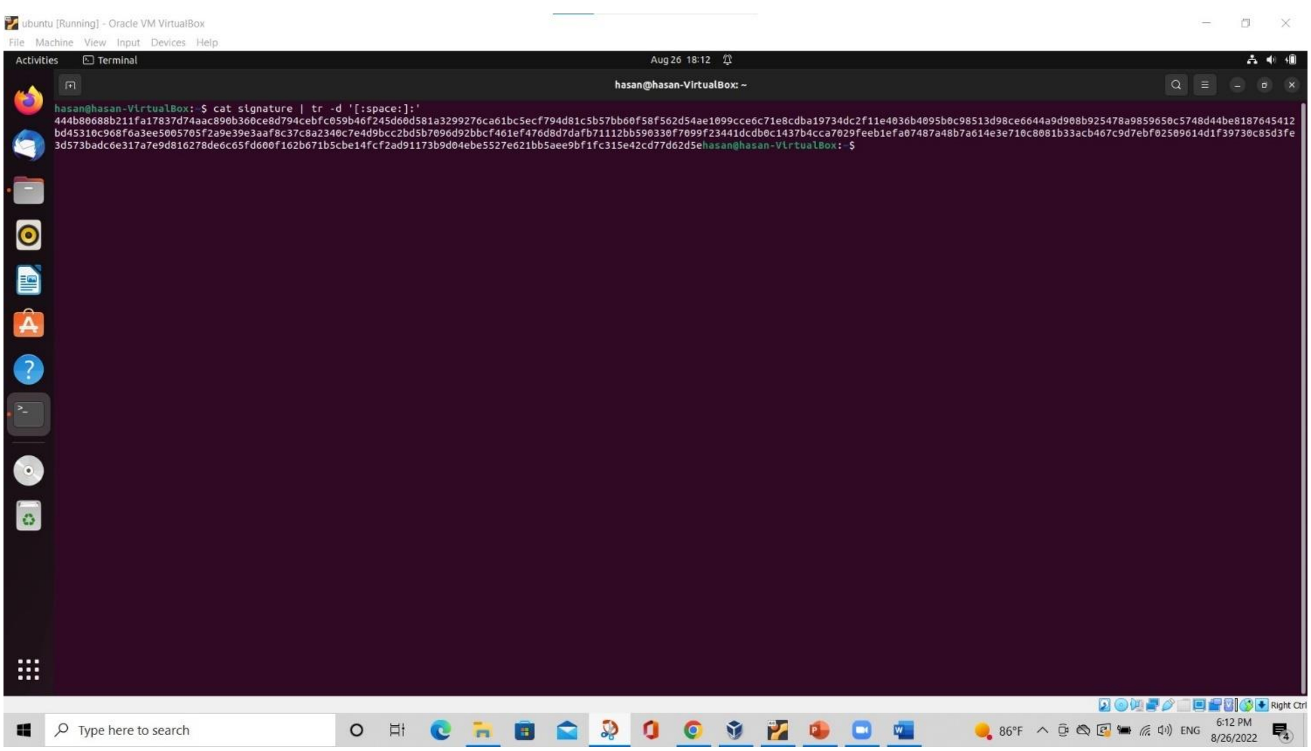

Figure 3.22: Removing Colons and Spaces From Signature File.

3.7.4. Extract the body of the server's certificate.

The following command to extract the body of the certificate as shown in Figure 3.23:

openssl asn1parse -i -in c0.pem

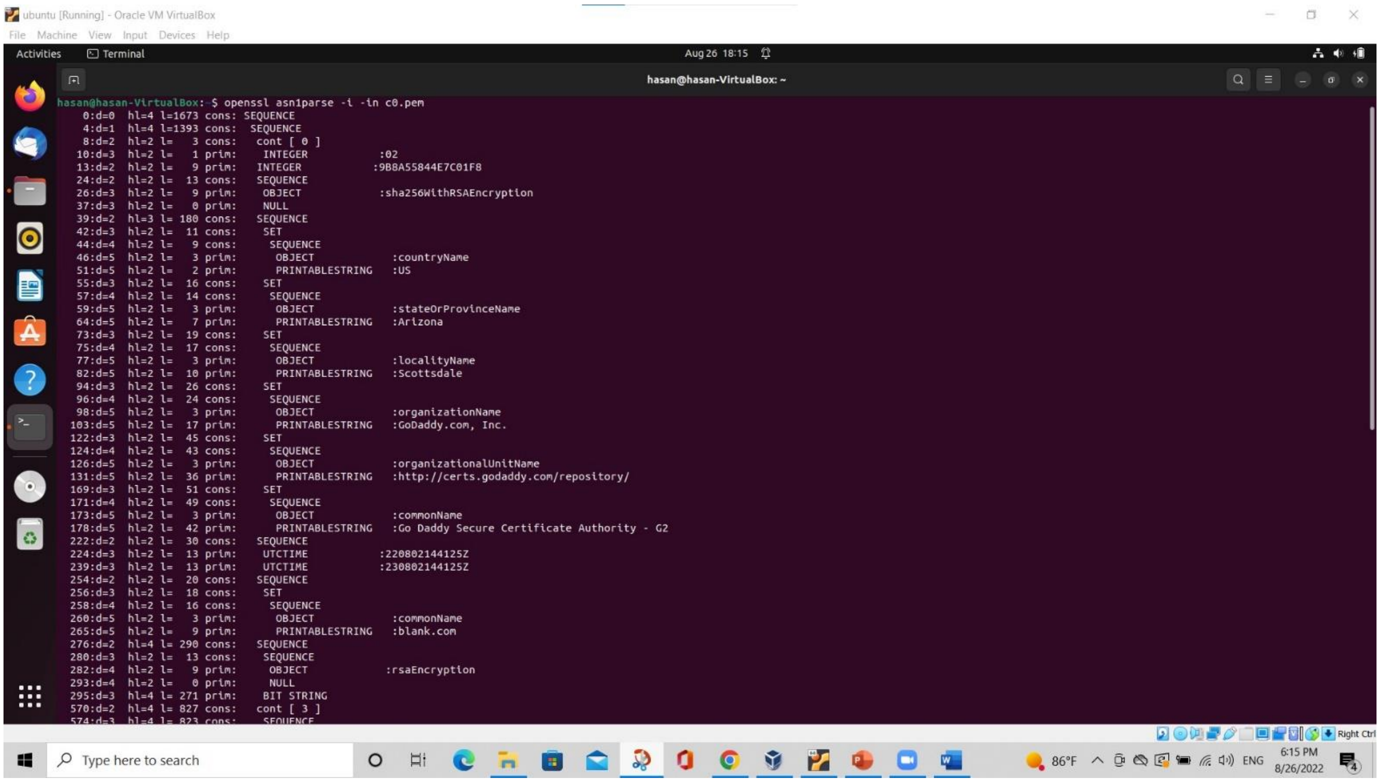

Figure 3.23: Extracting the Body of the Certificate

The -strparse option was used to get the field from the offset 4, which will give us the body of the certificate, excluding the signature block:

$openssl asn1parse -i -in c0.pem -strparse 4 -out c0_body.bin -noout

Once the body of the certificate is observed, its hash was calculated using the following command as shown in Figure 3.24:

$ sha256sum c0_body.bin

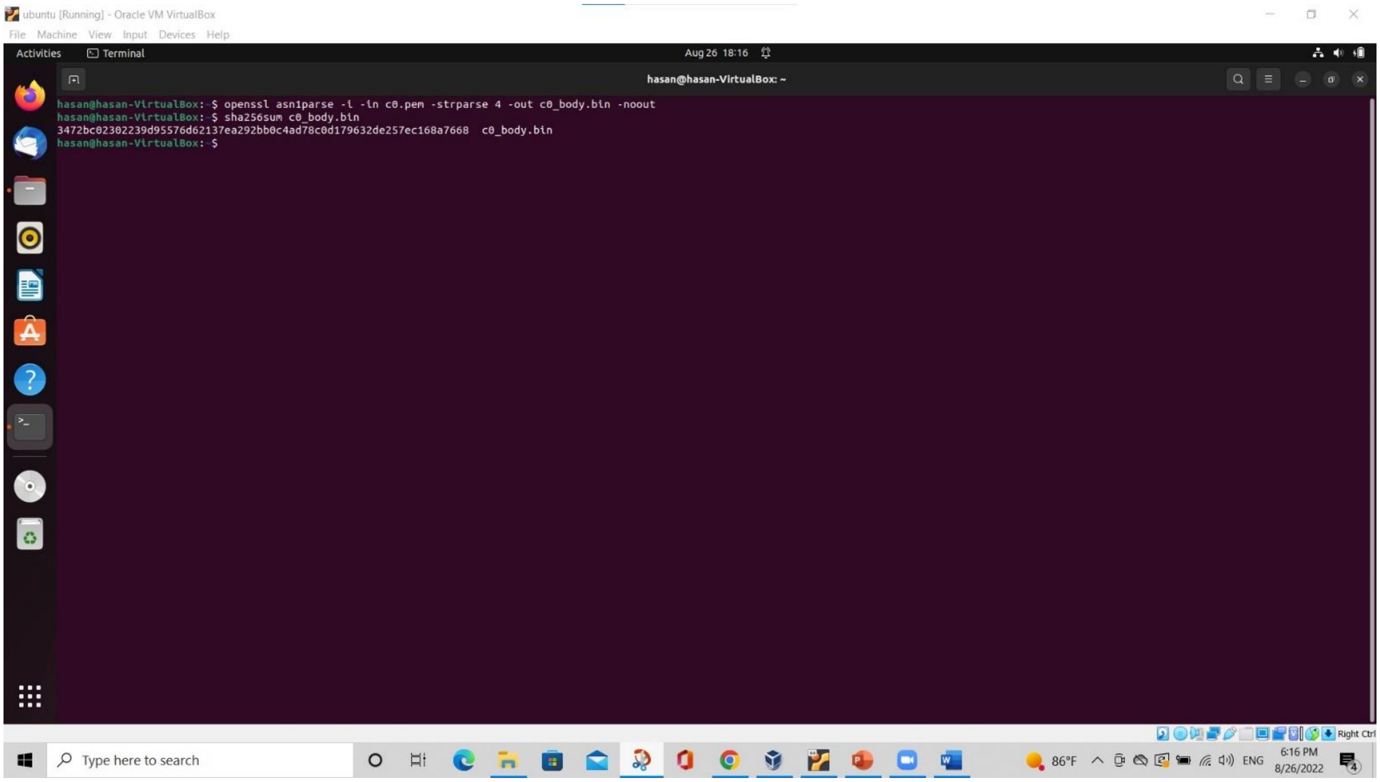

Figure 3.24: Calculating the Hash of the Certificate

3.7.5. Verify the signature

Code:

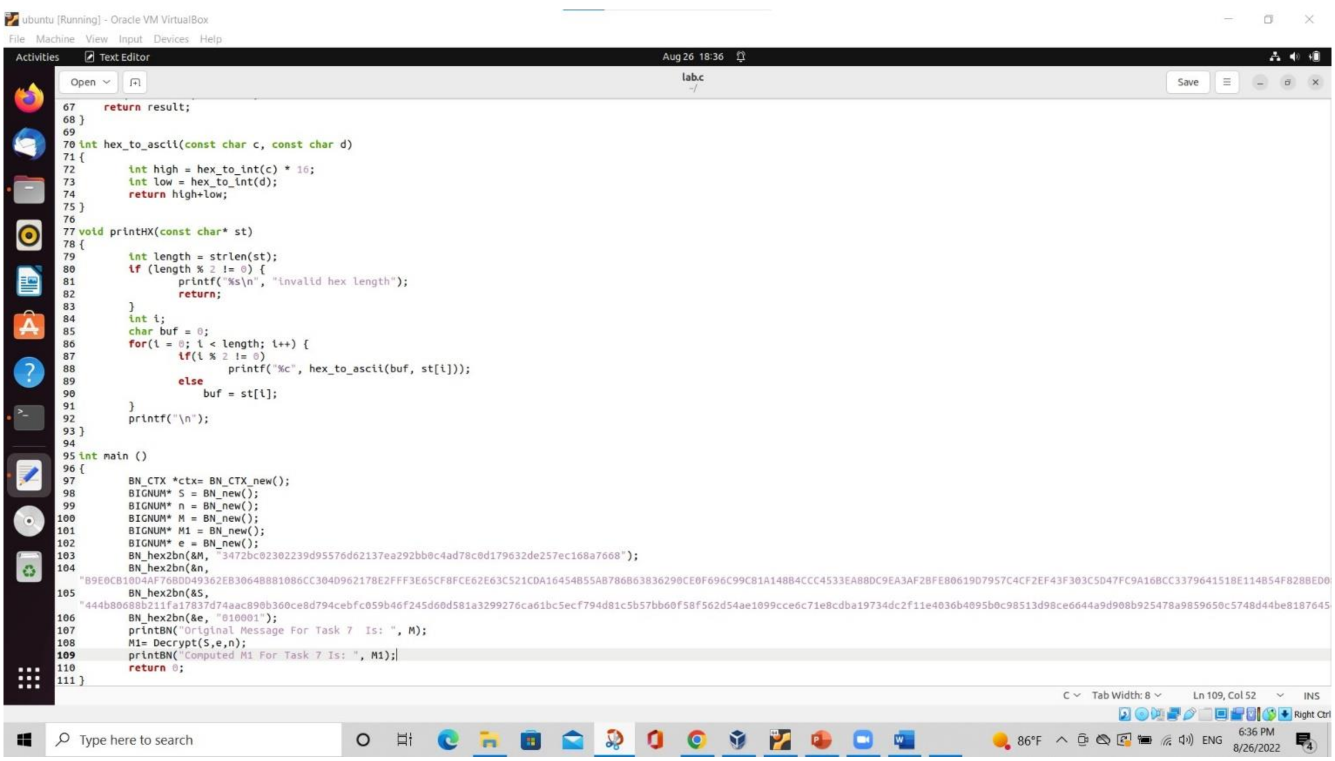

Figure 3.25: Verifying the Signature Code

The values obtained from the previous steps were used. The signature was obtained and the signature obtained was verified with the original signature as shown in Figure 3.26.

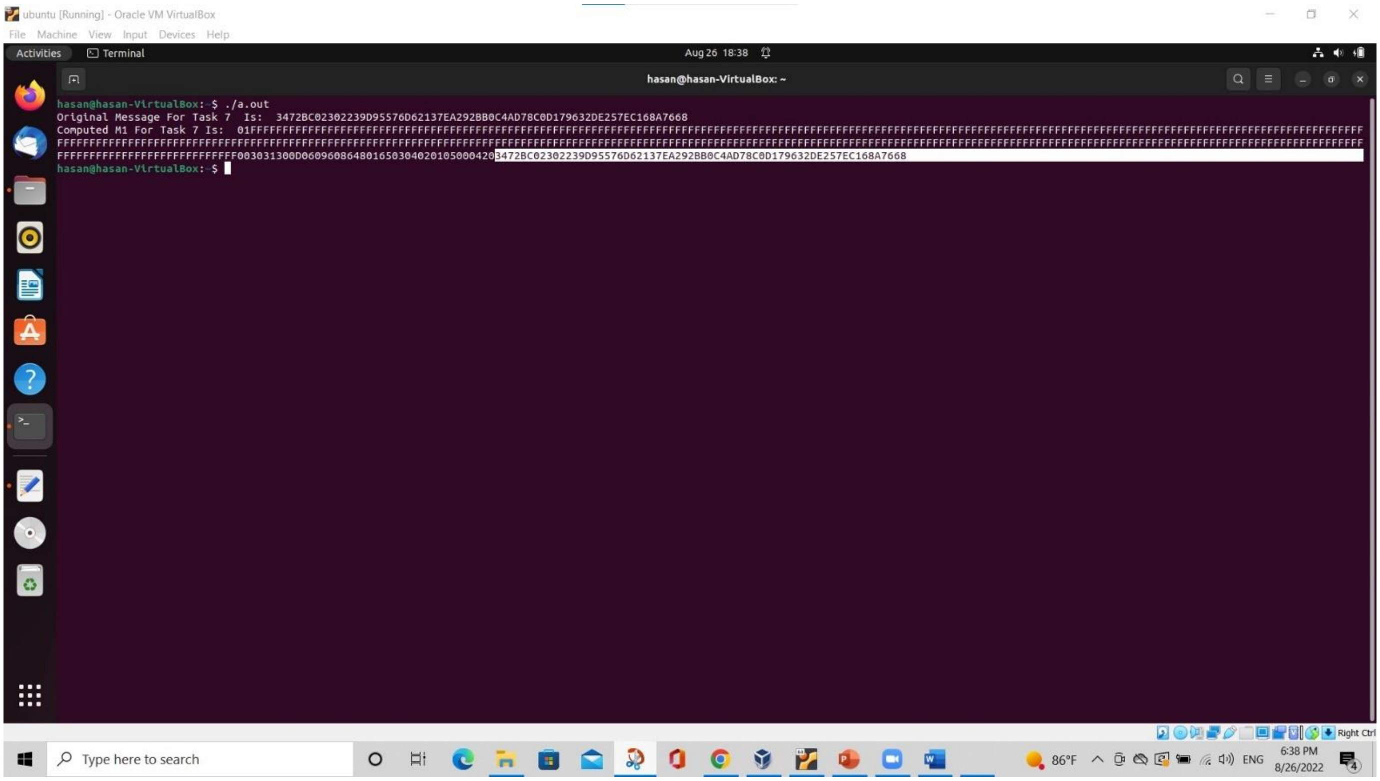

Figure 3.26: Verifying the Signature Output

The computed message's hash value and the original message have the same value. Therefore, we can say that the www.blank.com certificate has been confirmed to be right.

4. Conclusion

In conclusion, we understand the RSA public key cryptosystem, and we understand the methods: key generation, encryption and decryption using Linux and C language, and we understand how to deal with big numbers, and we learned how to generate and verify digital signatures and finally, an X.509 Certificate was manually verified using openSSL.

5. References

[1] https://en.wikipedia.org/wiki/RSA_(cryptosystem)

Accessed 27 August 2022

[2] https://en.wikipedia.org/wiki/Digital_signature

Accessed 27 August 2022

[3] https://sectigo.com/resource-library/what-is-x509-certificate

Accessed 27 August 2022

[4] Seed Labs RSA Encryption and Signature Lab PDF

Accessed 27 August 2022

[5] C03_Public_Key_Encryption.pptx Giving by Dr. Ahmad Alsadeh

Accessed 27 August 2022