

**Faculty of Engineering & Technology**

**Electrical & Computer Engineering Department**

**ENCS532**

**APPLIED CRYPTOGRAPHY**

**Crypto lab**

**Done by:**

**Amal Ziad-1192141**

**Instructor:**

**Hannah Zughbi**

**Date:**

**5/1/2021**

# **Abstract:**

The aim of this experiment is to get knowledge about how to deal with public key cryptography, the use of RSA algorithm and key generation. Also learn better about big number calculation. Then applying encryption and decryption using RSA. Also get to know what digital signature is and how it works.

**Contents**

[**Abstract:** 2](#_Toc92239737)

[ **Theoretical background:** 3](#_Toc92239738)

[ BigNum API: 3](#_Toc92239739)

[ RSA algorithm: 3](#_Toc92239740)

[ Digital Signature: 3](#_Toc92239741)

[ **Task 1:** **Deriving the Private Key** 4](#_Toc92239742)

[ **Task 2:** **Encrypting a Message** 7](#_Toc92239743)

[ **Task 3:** **Decrypting a Message** 10](#_Toc92239744)

[ **Task 4: Signing a Message** 11](#_Toc92239745)

[ **Task 5: Verifying a Signature** 15](#_Toc92239746)

[ **Task 6: Manually Verifying an X.509 Certificate** 17](#_Toc92239747)

[ **Conclusion** 18](#_Toc92239748)

# **Theoretical background:**

The RSA algorithm involves computations on large numbers. These computations cannot be directly conducted using simple arithmetic operators in programs, because those operators can only operate on primitive data types, such as 32-bit integer and 64-bit long integer types.

## BigNum API:

A BigNumber is an object which safely allows mathematical operations on numbers of any magnitude. Most operations which need to return a value will return a BigNumber and parameters which accept values will generally accept them.[[1]](#footnote-1)

## RSA algorithm:

The **RSA algorithm** is an asymmetric cryptography algorithm; this means that it uses a public key and a private key (i.e two different, mathematically linked keys). As their names suggest, a public key is shared publicly, while a private key is secret and must not be shared with anyone.[[2]](#footnote-2)

## Digital Signature:

Digital signatures are the public-key primitives of message authentication. In the physical world, it is common to use handwritten signatures on handwritten or typed messages. They are used to bind signatory to the message.[[3]](#footnote-3)

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

Let p, q, and e be three prime numbers. Let n = p\*q. We will use (e, n) as the public key. We want to calculate the private key d. The hexadecimal values of p, q, and e are as shown in fig.1.1.



Fig.1.1

The code in c language is shown in fig.1.2



Fig.1.2

Note:

I used lab\_util.h header which helped me in dealing with BIGNUM and other things. The information inside lab\_util.h are shown in fig.1.3.1, fig.1.3.2, and fig.1.3.3.

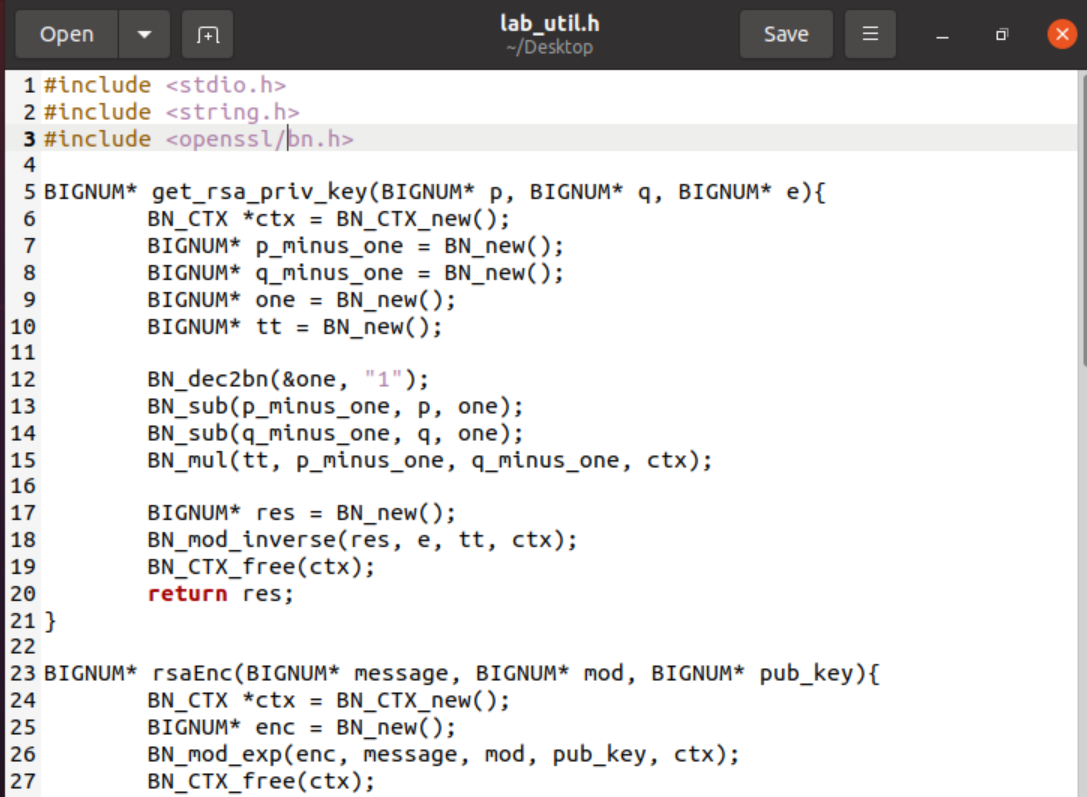


Fig.1.3.1

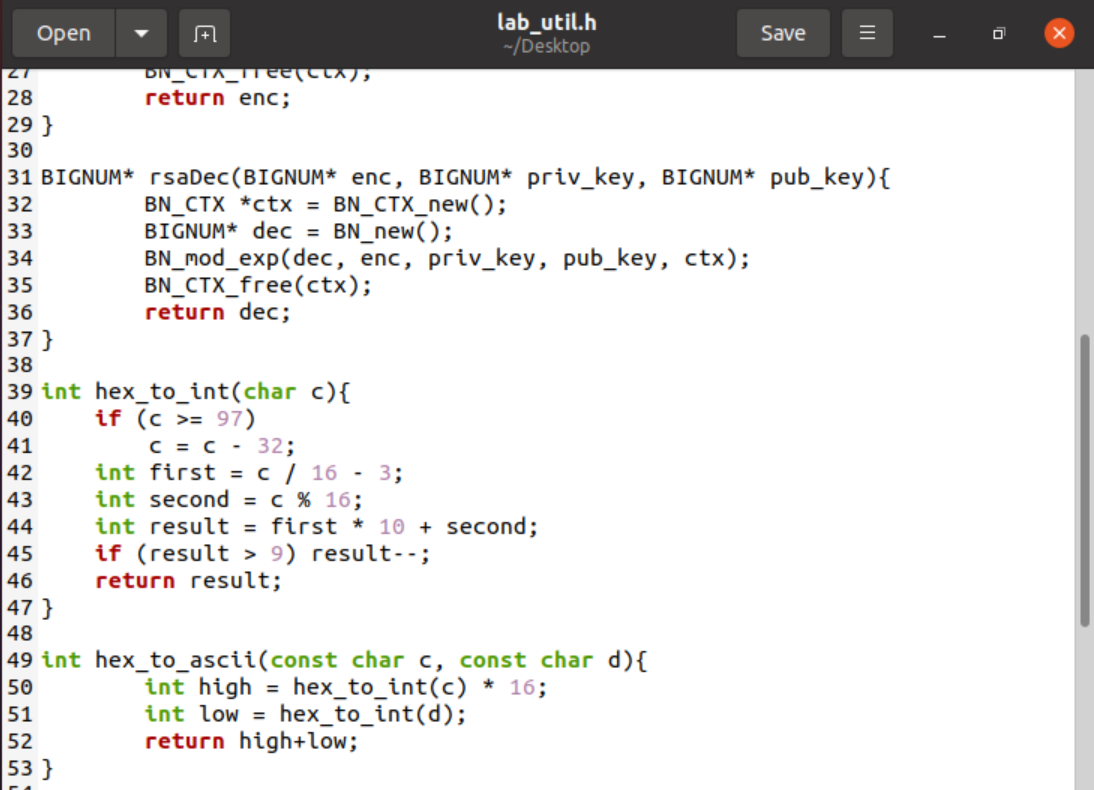


Fig.1.3.2



Fig.1.3.3

After compiling task1.c file then executing task1.out I got the following result shown in fig.1.4.

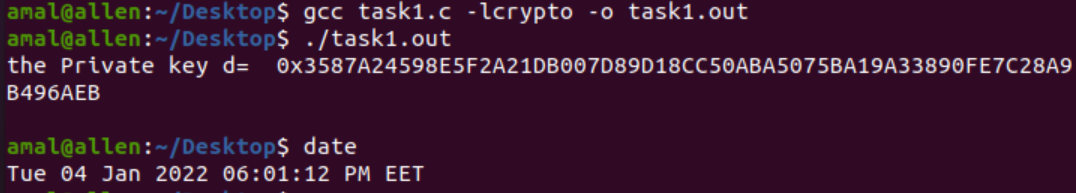


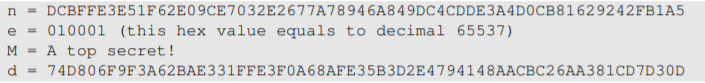
Fig.1.4

Discussion:

As we see in fig.1.4, the program gave us the private key d in hexadecimal format from inputs p, q, and e that we gave them. I noticed that the length of d is more than p and q.

# **Task 2:** **Encrypting a Message**

Let (e, n) be the public key. We want to encrypt this message :”A top secret!” (without quotations). We need to convert this ASCII string to a hex string, and then convert the hex string to a BIGNUM using the hex-to-bn API BN hex2bn(). The public keys and also the private key are shown in the following.



The code in c language is shown in fig.2.1.1 and fig.2.1.2.

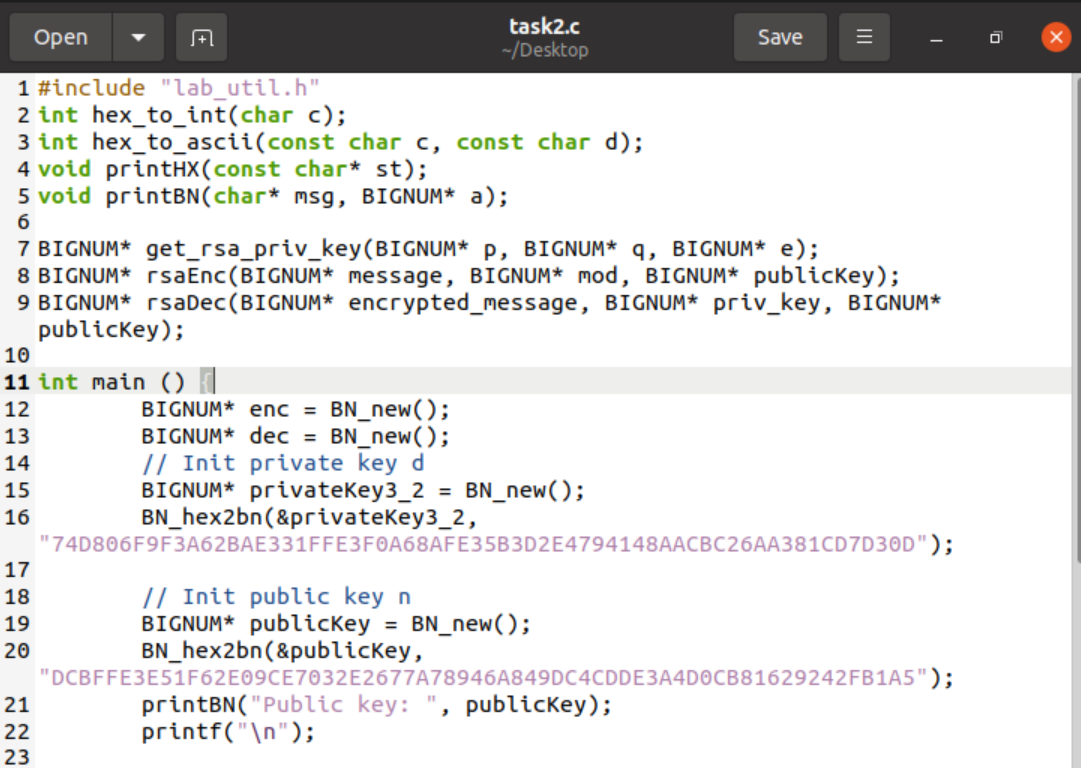


Fig.2.1.1



Fig.2.1.2

Note:

In the above code I put the message “A top secret!” in hexadecimal format and took as shown in fig.2.2.

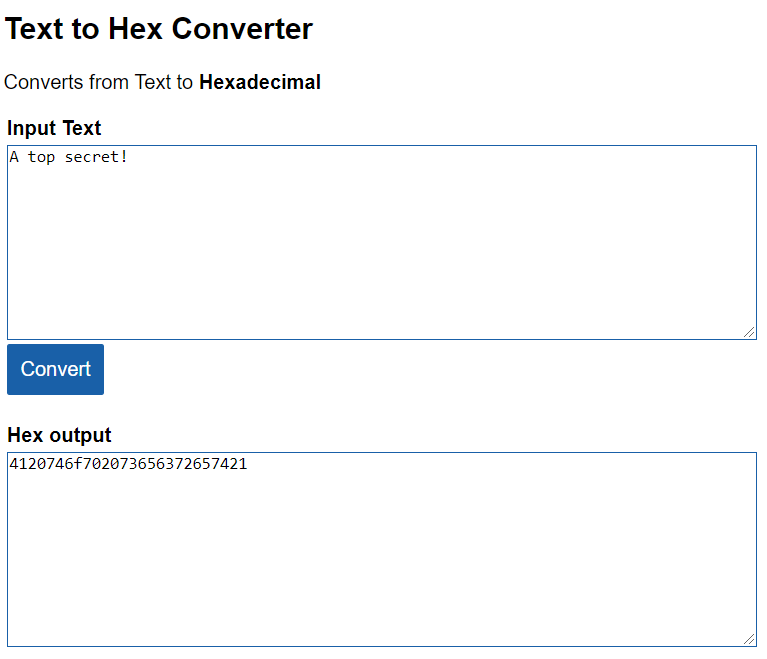


Fig.2.2

And after compiling task2.c file then executing task2.out, I got the result shown in fig.2.3



Fig.2.3

Discussion:

In c code in fig.2.1.1 and fig.2.1.2, I made it so that it takes the message we want to encrypt by the given public key, it gave us the encrypted message in hexadecimal format. I also noticed that the encrypted message here is longer than the original format. Finally I used the given private key to see if it will give us the same message or not, and it’s really gave us the same as we wished.

# **Task 3:** **Decrypting a Message**

The public/private keys used in this task are the same as the ones used in Task 2. We want to decrypt the following ciphertext C, and convert it back to a plain ASCII string.

C = 8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F

The code in c language is shown in fig.3.1.1 and fig.3.1.2.

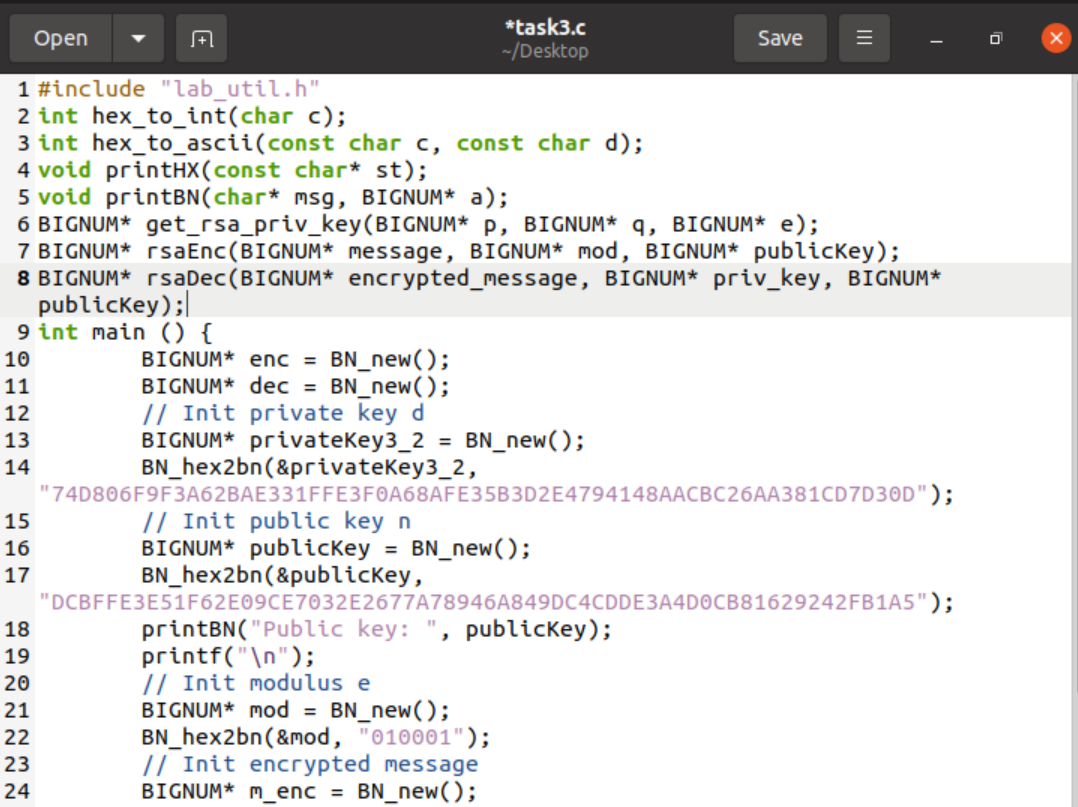


Fig.3.1.1



Fig.3.1.2

And after compiling task3.c file then executing task3.out, I got the result shown in fig.3.2



Fig.3.2

Discussion:

After decrypting the ciphertext using the given public and private key, I used BN\_bn2hex method to convert to plain ASCII string instead of using the given python command:

$python -c ’print("4120746f702073656372657421".decode("hex"))’ A top secret!

And finally I got the message which is:”Password is dees”.

# **Task 4: Signing a Message**

We want to generate a signature for the following message:

M = I owe you $2000.

The c code for this task is shown in fig.4.1.1 and fig.4.1.2.

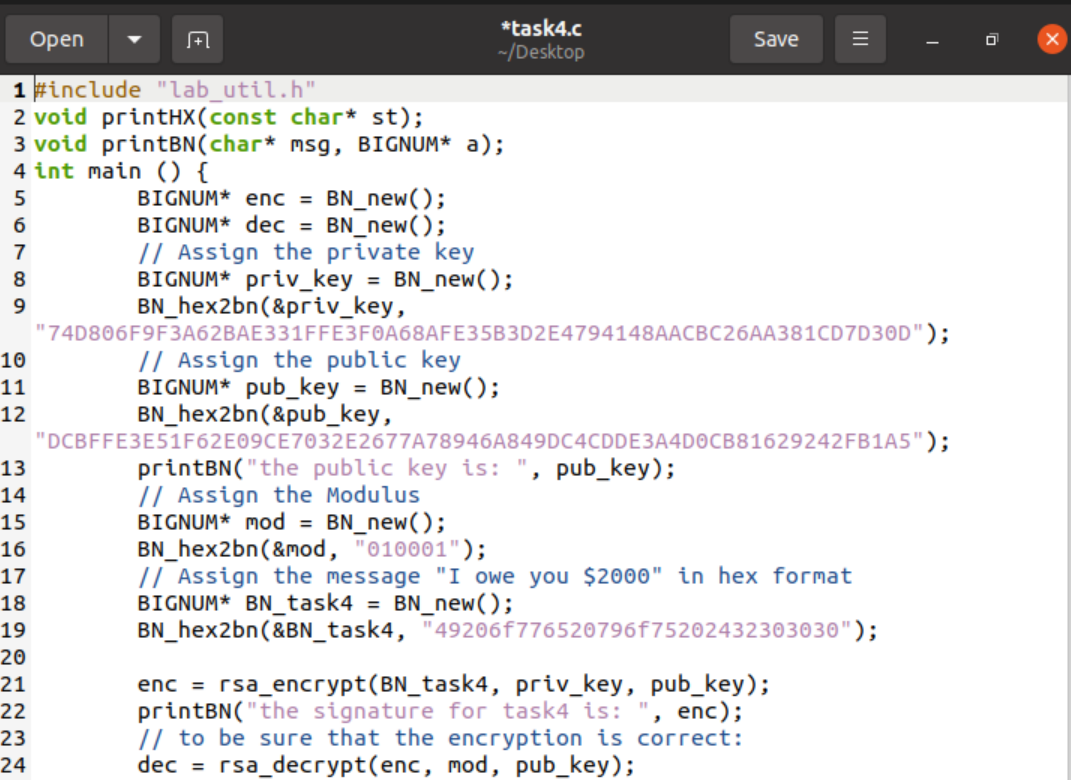


Fig.4.1.1

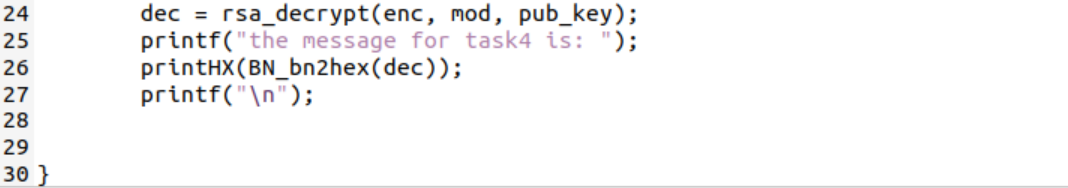


Fig.4.1.2

The message in the code above is in hex format that I have got it from the converter instead of python command:

$python -c ’print("4120746f702073656372657421".decode("hex"))’ A top secret!

As shown in fig.4.2.

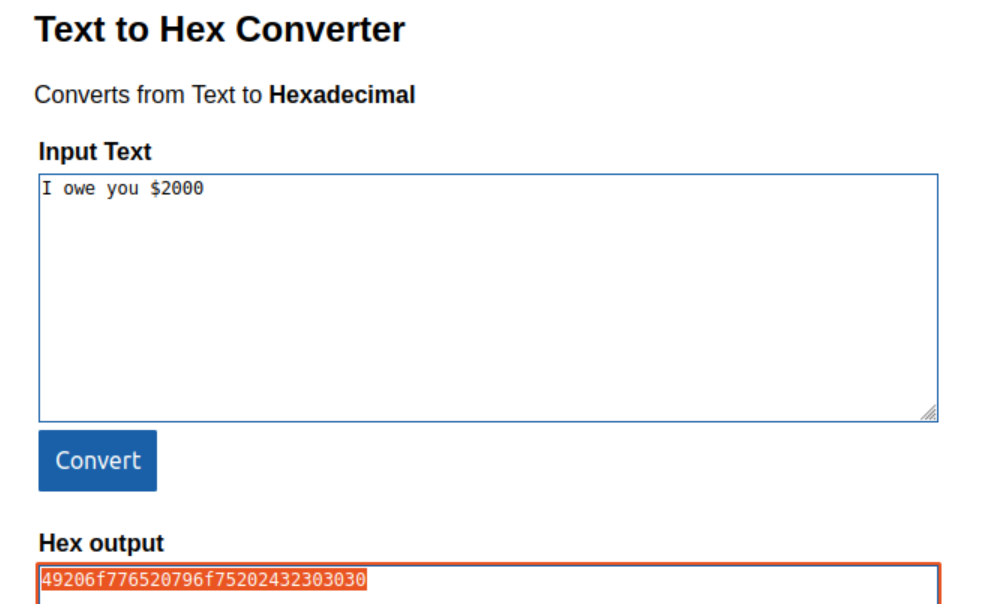


Fig.4.2

Finally I have got the result as shown in fig.4.3.

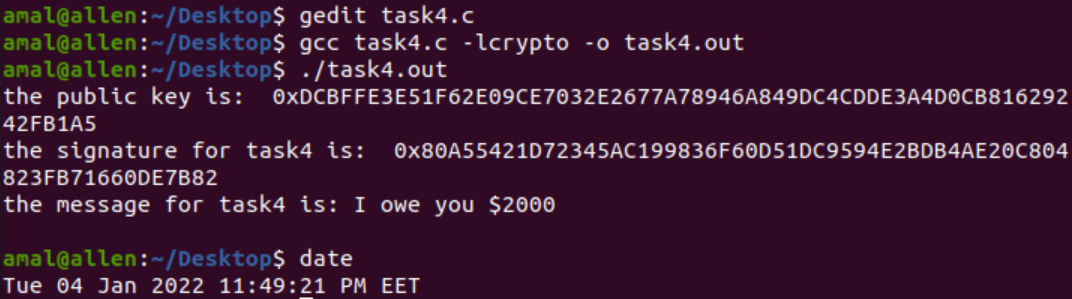


Fig.4.3

Discussion:

As we see the signature is the same length as public key. And my work is correct depending on decryption result (the message).

Let’s try to change in the message and see the difference between the two results.

As we have seen before, getting the hex format for the message: “I owe you $5000”, and 5000 instead of 2000. As shown in fig.4.4.

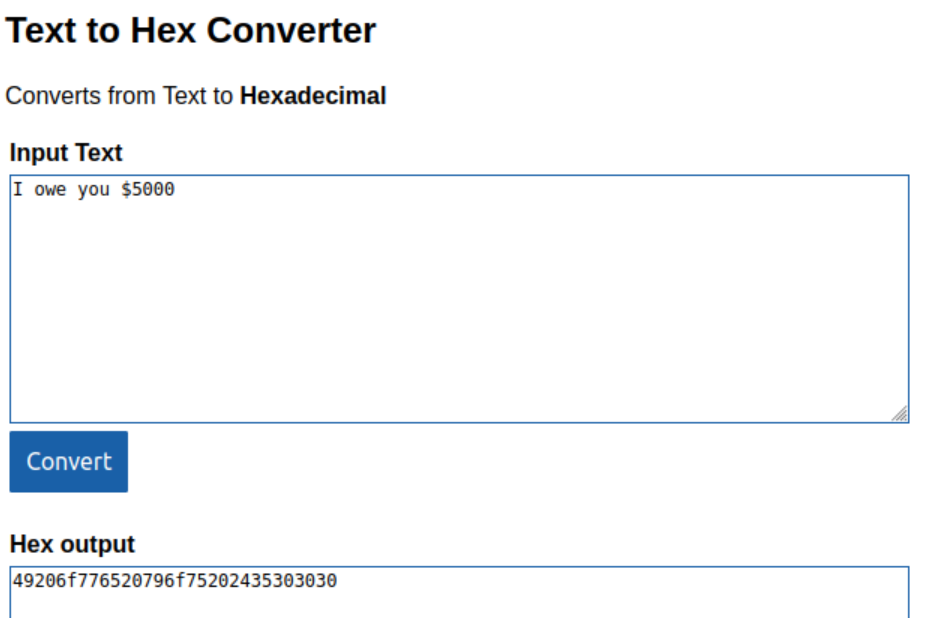


Fig.4.4

And the same code in fig.4.1.1 and fig.4.1.2 except the message format. Shown in fig.4.5.1 and fig.4.5.2.

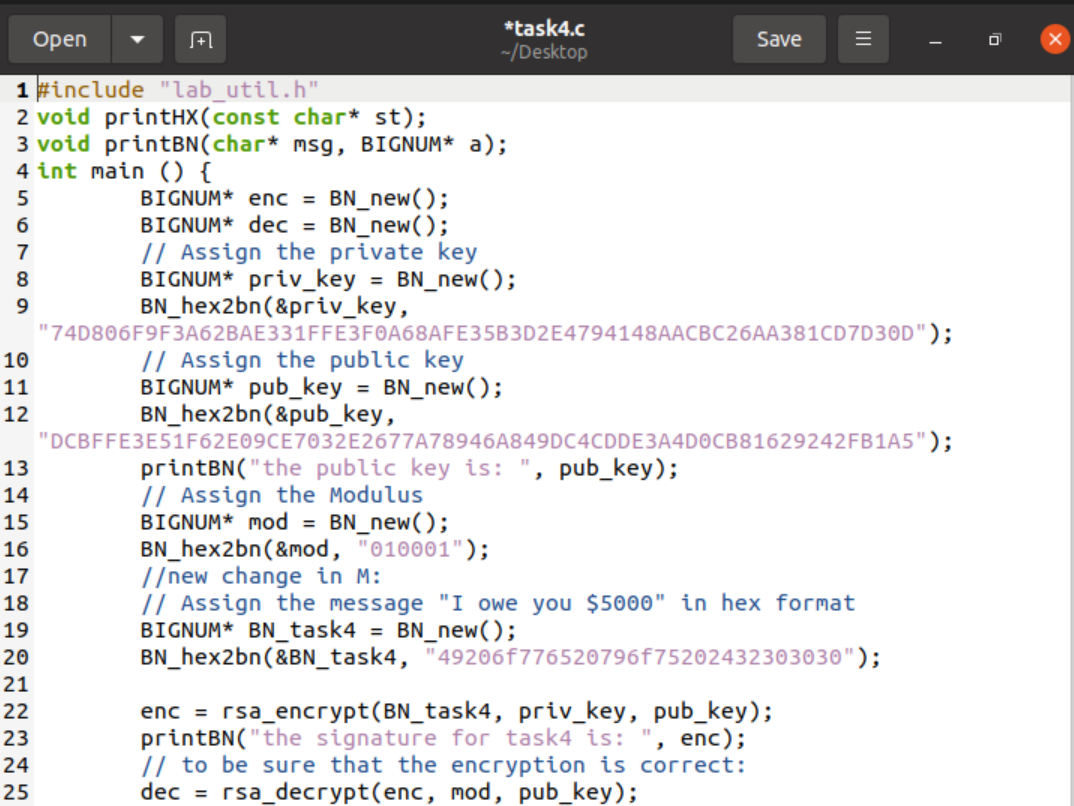


Fig.4.5.1

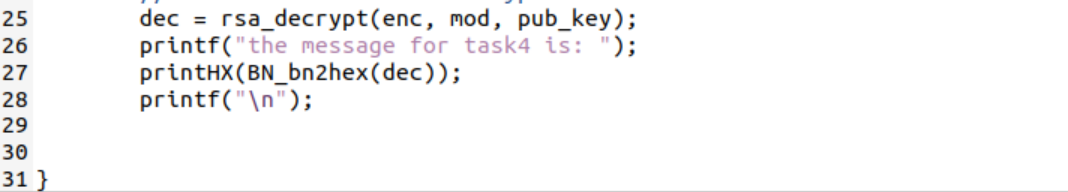


Fig.4.5.2

And the result as shown in fig.4.6.

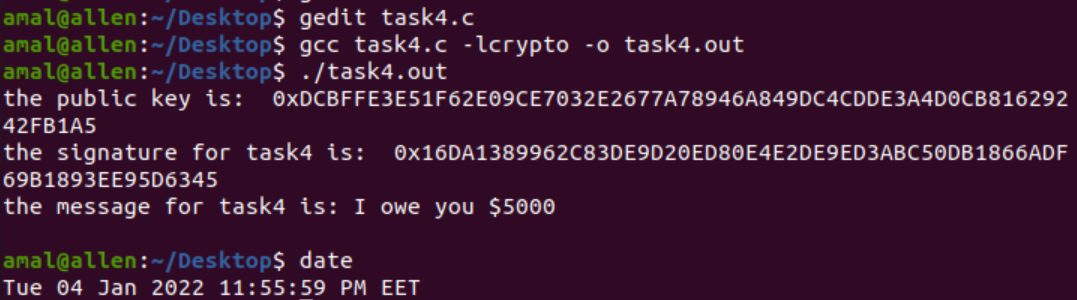


Fig.4.6

Discussion:

I have noticed that the two signatures are totally different even though I have changed a small part of the message. I concluded that the digital signature is a powerful encryption method.

# **Task 5: Verifying a Signature**

Bob receives a message M = "Launch a missile." from Alice, with her signature S. We know that Alice’s public key is (e, n). Please verify whether the signature is indeed Alice’s or not. The public key and signature (hexadecimal) are listed in the following:

M = Launch a missile.

S = 643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F

e = 010001 (this hex value equals to decimal 65537)

n = AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115

First, I got the hex format for M as shown in fig.5.1.

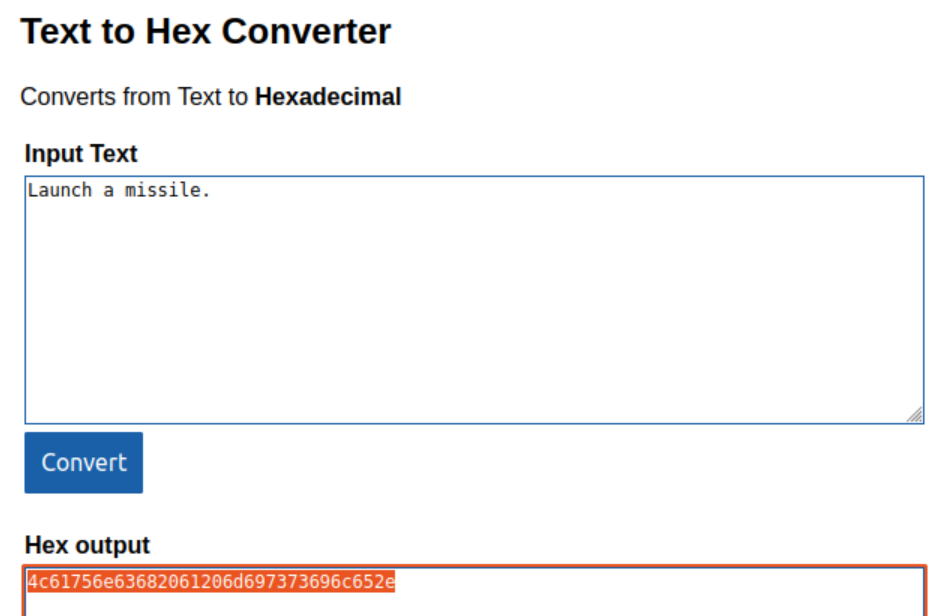


Fig.5.1

Then the c code is as shown in fig.5.2.1 and fig.5.2.2.

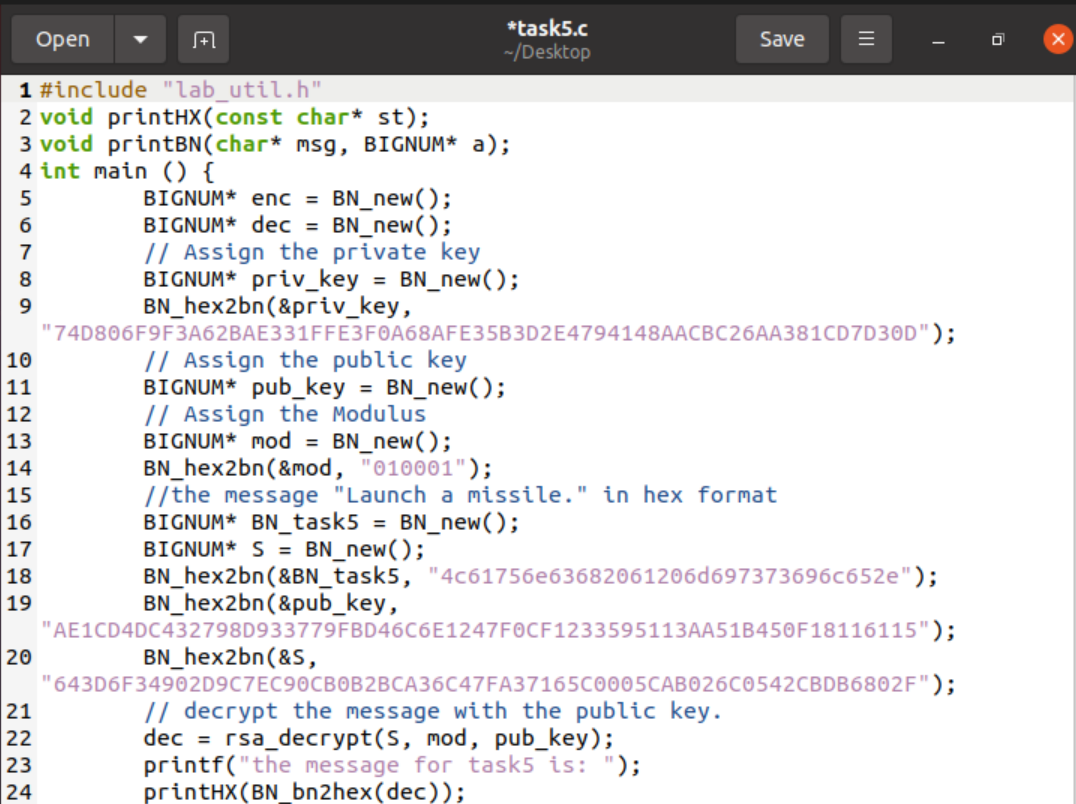


Fig.5.2.1



Fig.5.2.2

Discussion:

In the code, first I encrypted the message by the given public and private keys and signature, and after getting the result I corrupted signature by changing the last bits of the signature value by changing 2F to 3F.

And finally after executing the file and got the result shown in fig.5.3.

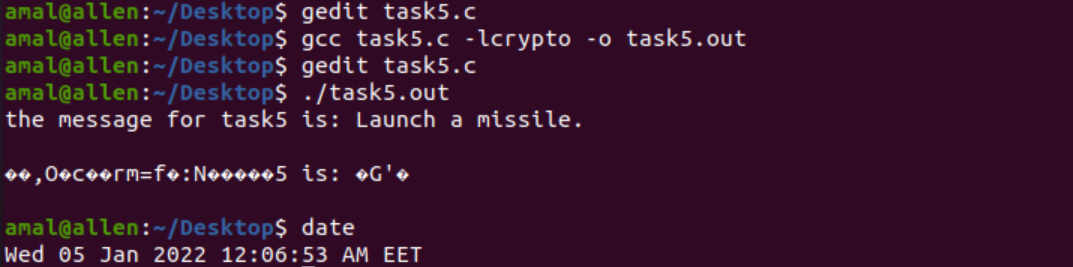


Fig.5.3

Discussion:

As we see in fig.5.3, the corrupted message is obvious.

# **Task 6: Manually Verifying an X.509 Certificate**

After downloading a real X.509 certificate from a web server, getting its issuer’s public key, and then using this public key to verify the signature on the certificate.

The c code as shown in fig.6.1.

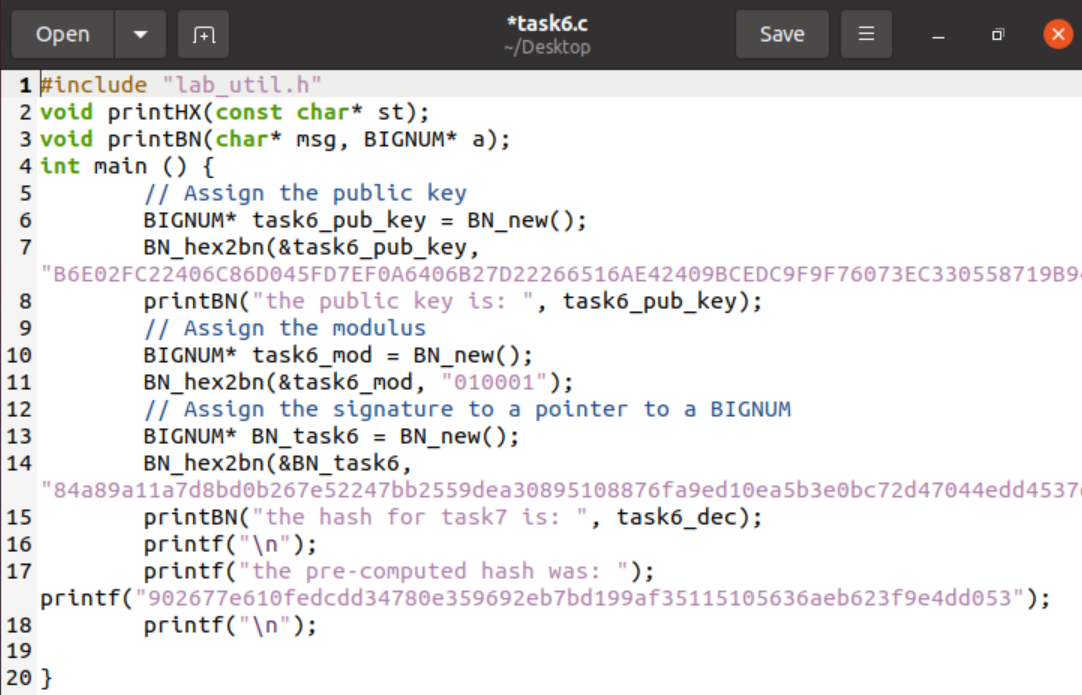


Fig.6.1

Discussion:

In the fig.6.1, I generated the hash of the certificate body like so: First, extracting the body of the certificate openssl asn1parse -i -in c0.pem -strparse 4 -out c0\_body.bin –noout , Then computing the hash: sha256sum c0\_body.bin .This hash used in comparison for when decrypting the signature. After that I decrypted the signature using the public key and modulus given from the certificate. If the signature is valid, it should match our hash of the certificate body I have computed earlier. Then printing the decrypted hash. This is a non-masked value. Follows that, I wanted to bitmask & this value with 256 bit. In other words, the first 32 bytes of this value should match the hash generated from the body of the certificate as I mentioned.

Then the final result as shown in fig.6.2.

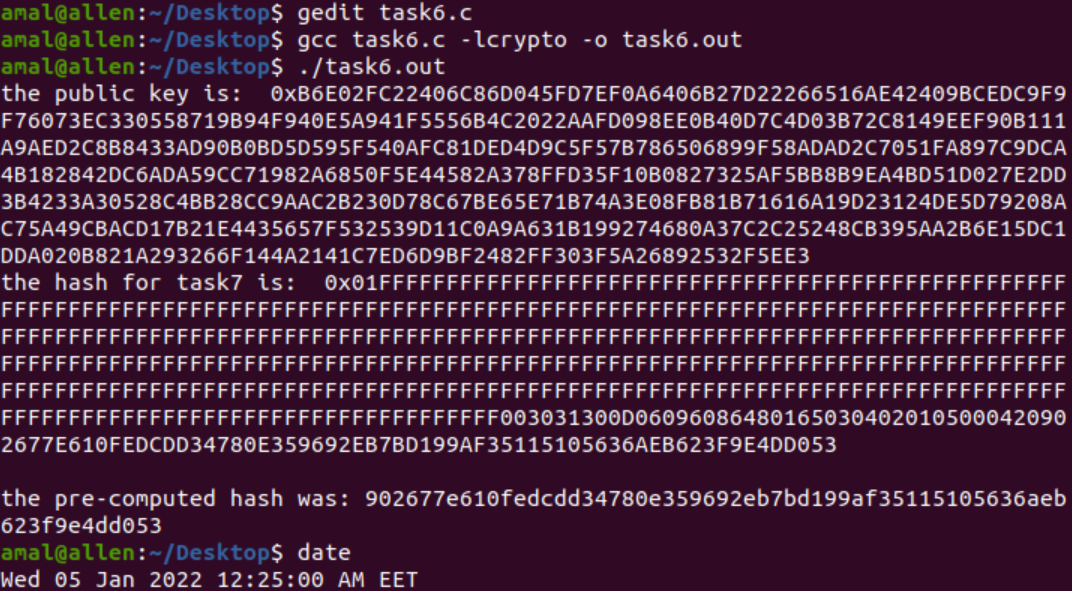


Fig.6.2

# **Conclusion**

In conclusion of this lab, I satisfied the initial abstract and got knowledge about public-key cryptography, the RSA algorithm and key generation, big number (BIGNUM) calculation. In addition, learned how to encrypt and decrypt using RSA. Also and best one for me is that I have gotten knowledge about digital signature and how to deal with it. Finally, made a X.509 certificate.

1. https://docs.ethers.io/v5/api/utils/bignumber/#:~:text=A%20BigNumber%20is%20an%20object,values%20will%20generally%20accept%20them. [↑](#footnote-ref-1)
2. https://www.educative.io/edpresso/what-is-the-rsa-algorithm [↑](#footnote-ref-2)
3. https://www.tutorialspoint.com/cryptography/cryptography\_digital\_signatures.htm [↑](#footnote-ref-3)