**RSA Encryption and Signature**

Chawin Metah

Department of Computer Science, Purdue University Fort Wayne

ACS 54500: Cryptography and Network Security

Lab 9

Dr. Zesheng Chen

April 25, 2022

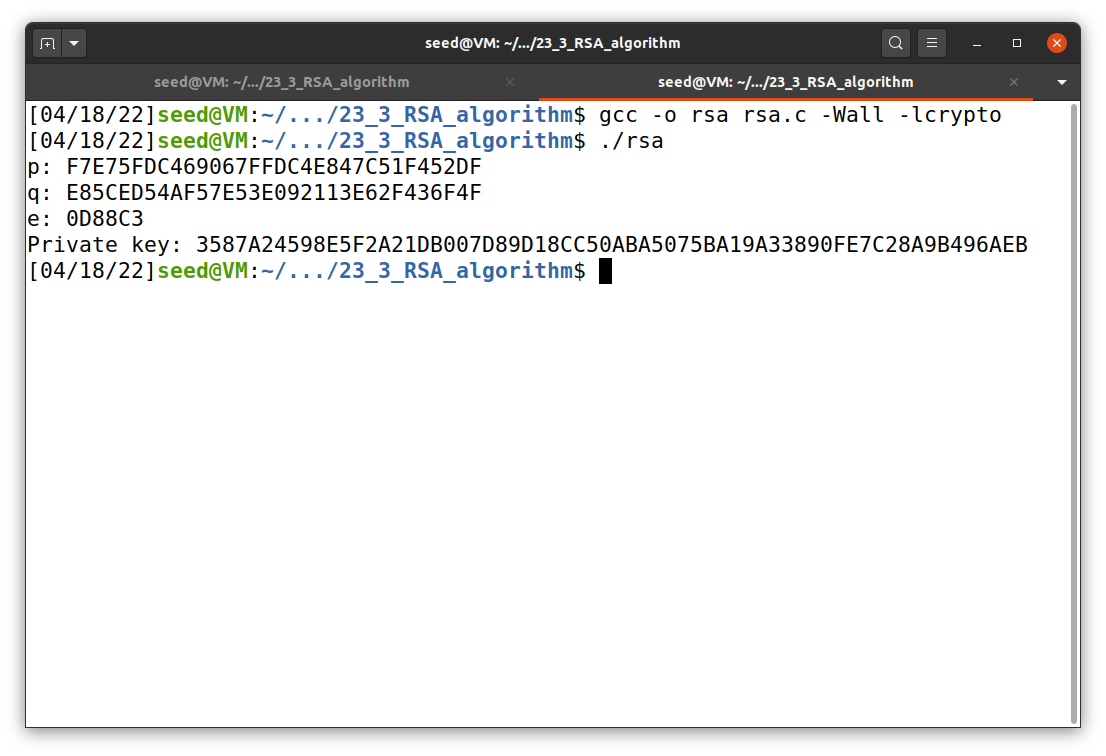
**Task 1: Deriving the Private Key**

For this task, I used the code provided on Brightspace. The “rsa.c” code that I used for each task will be reuploaded to my GitHub repository. From the direction of the task, it provides p, q, and e in hexadecimal. I utilized rsa.c code to find the hexadecimal value of the private key d. The equation e ∗ d mod φ(n) = 1 is used to find d (private key) using function BN\_mod\_inverse(). The result is shown in Figure 1.

Note: The code can be complied by using gcc -o rsa rsa.c -Wall -lcrypto

**Figure 1**

*Private key*



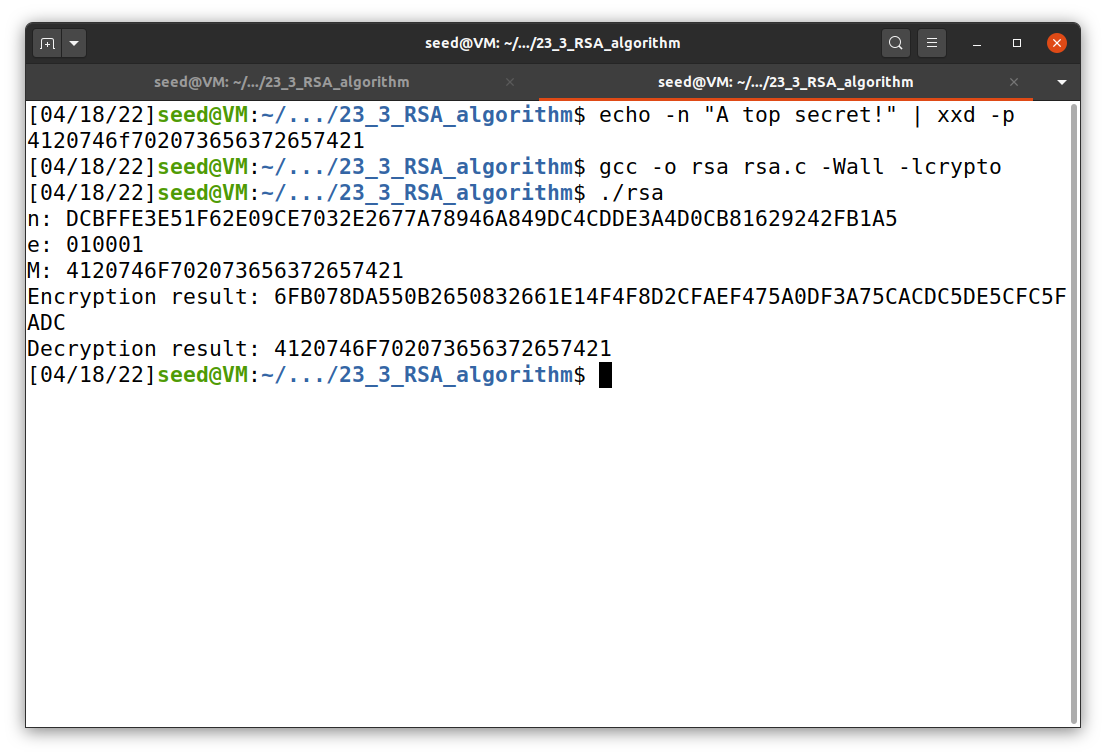
**Task 2: Encrypting a Message**

For this task, I used the code provided on Brightspace. The “rsa.c” code that I used for each task will be reuploaded to my GitHub repository. From the direction of the task, it provides n, e, and message “A top secret!”. I utilized rsa.c code to find the hexadecimal value of the encrypted message. First, I have to find the hexadecimal value of the message using the command below. Then, I reconfigured the code with new values of n, e, and message in hexadecimal. I also decrypted the message using d provided in the lab direction. In this case, the encrypted message is correct because the decrypted message is similar to the original message. The result is shown in Figure 2.

echo -n "A top secret!" | xxd -p

**Figure 2**

*Message encryption*



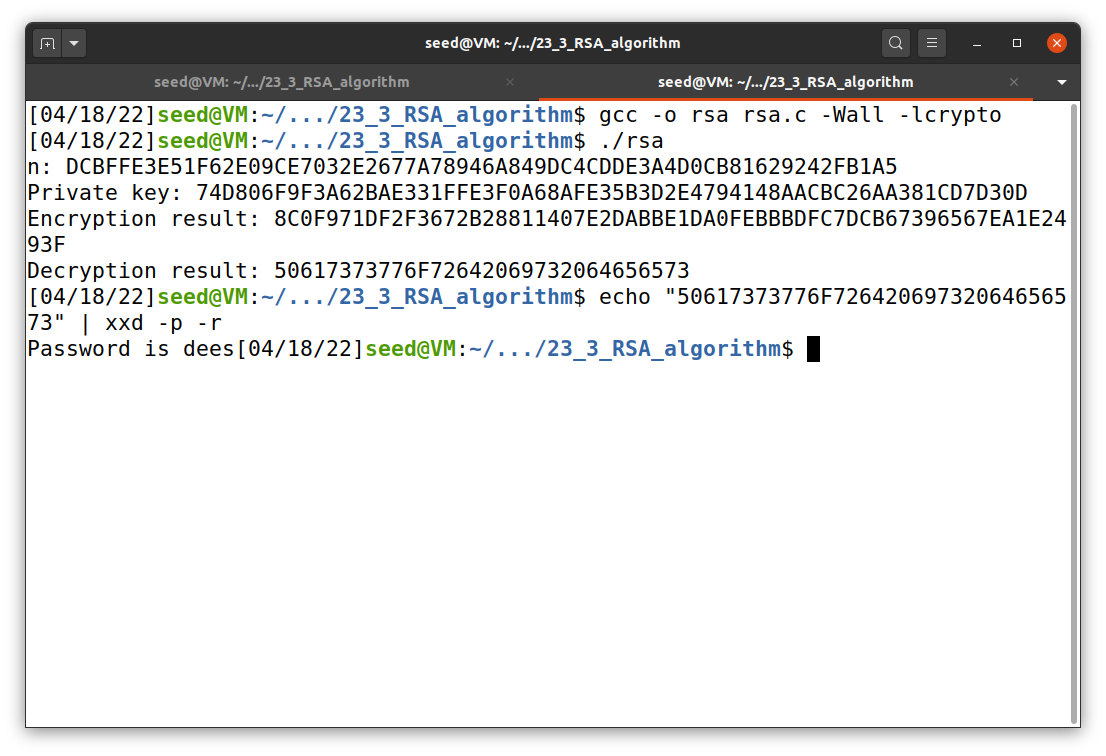
**Task 3: Decrypting a Message**

The value n and d are given in the previous task. In addition, the encrypted message c is given in the lab direction. Now, the equation M = Cd mod n is used to decrypt the message. The decrypted message is in a form of hexadecimal value. To decode this, I used the command below. The result of this task is shown in Figure 3.

echo "50617373776F72642069732064656573" | xxd -p -r

**Figure 3**

*Message decryption*

**

**Task 4: Signing a Message**

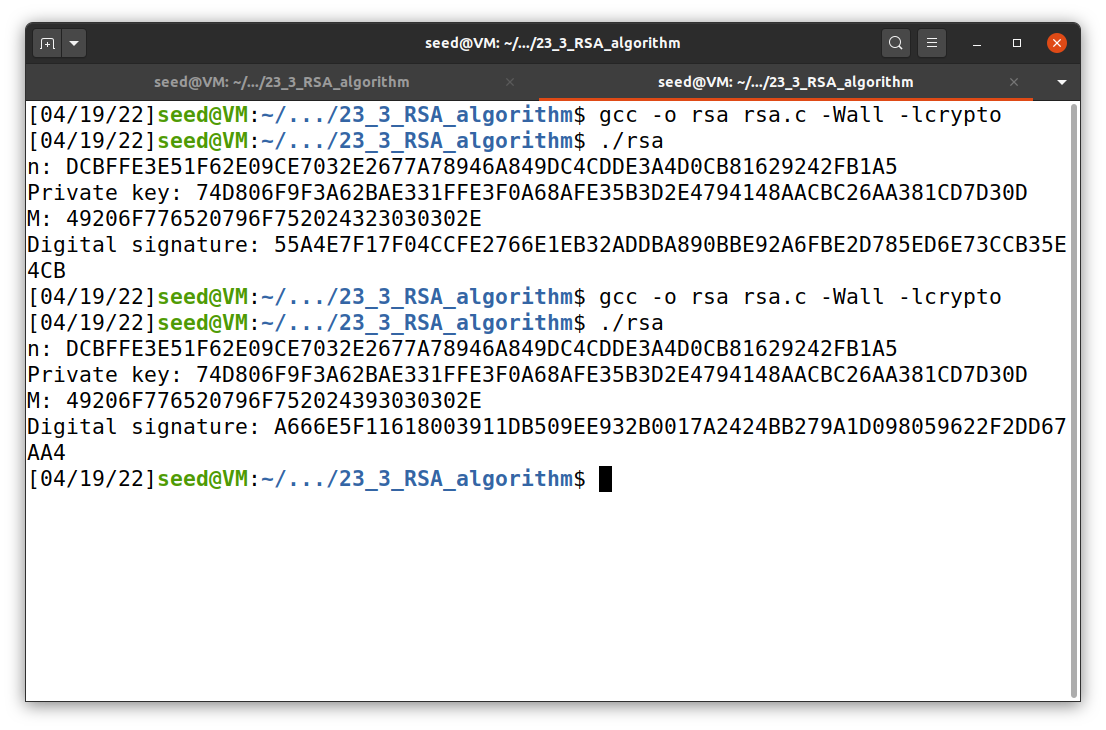
For this task, I used the code provided on Brightspace. The “rsa.c” code that I used for each task will be reuploaded to my GitHub repository. The value n and d are given in the previous task. The message is “I owe you $2000.”, which is 49206F776520796F75202432‌3030302E in hexadecimal. The commands below might be able to convert ASCII letters to hexadecimal value. However, they give an error, as shown in Figure 5. Instead, I used an external tool (online converter) to convert the string to hexadecimal value. Now, the equation S = Md mod n is used to create a digital signature. Note that d is a private key used to sign the message M. After that, I changed the message to “I owe you $9000.” and execute the program again. The results are shown in Figure 4. Even though we change only one character, the output (digital signature) is totally different.

echo -n "I owe you $2000." | xxd -p

python -c 'print("I owe you $2000.".encode("hex"))'

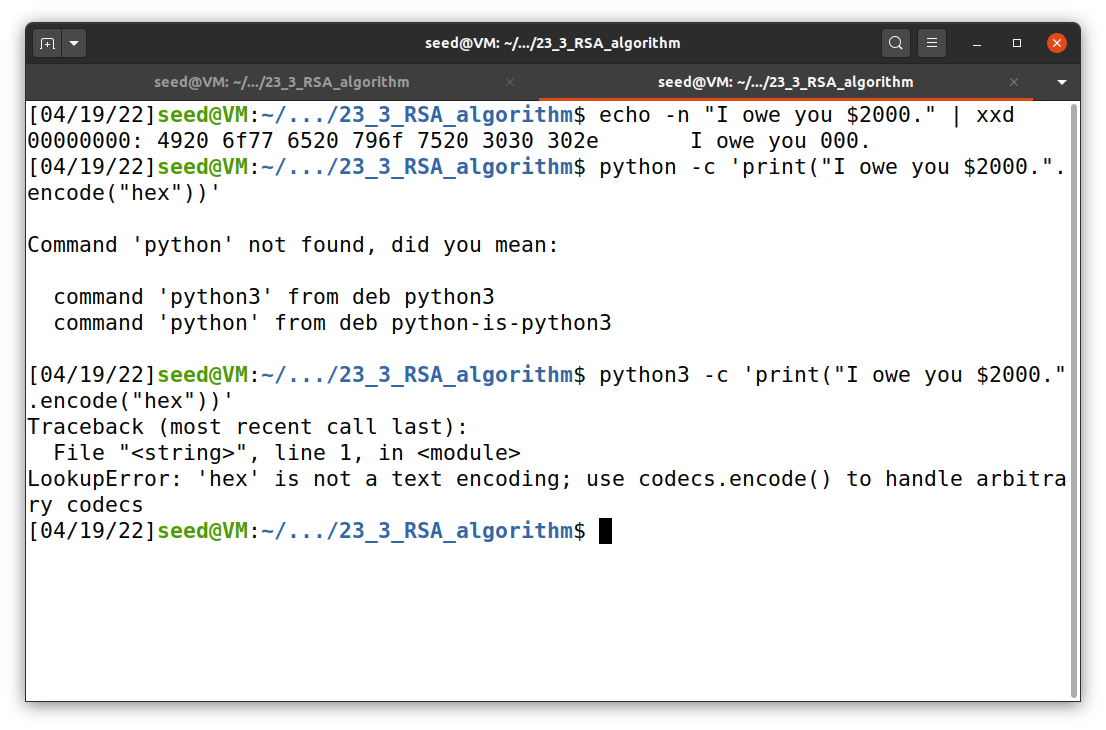
**Figure 4**

*Digital signature*



**Figure 5**

*ASCII to hex error*



**Task 5: Verifying a Signature**

For this task, I used the code provided on Brightspace. The “rsa.c” code that I used for each task will be reuploaded to my GitHub repository. The value n and d are given. The equation M = Se mod n is used to verify a digital signature. I verified the signature given in the lab direction and then change its last byte to 3F and reverify it. The results are shown in Figure 6 and 7. The verified message is “Launch a missile.” Even though I changed only one hex byte of the signature, the output is totally different with significantly longer digits. This means that it is very difficult to falsify a digital signature.

**Figure 6**

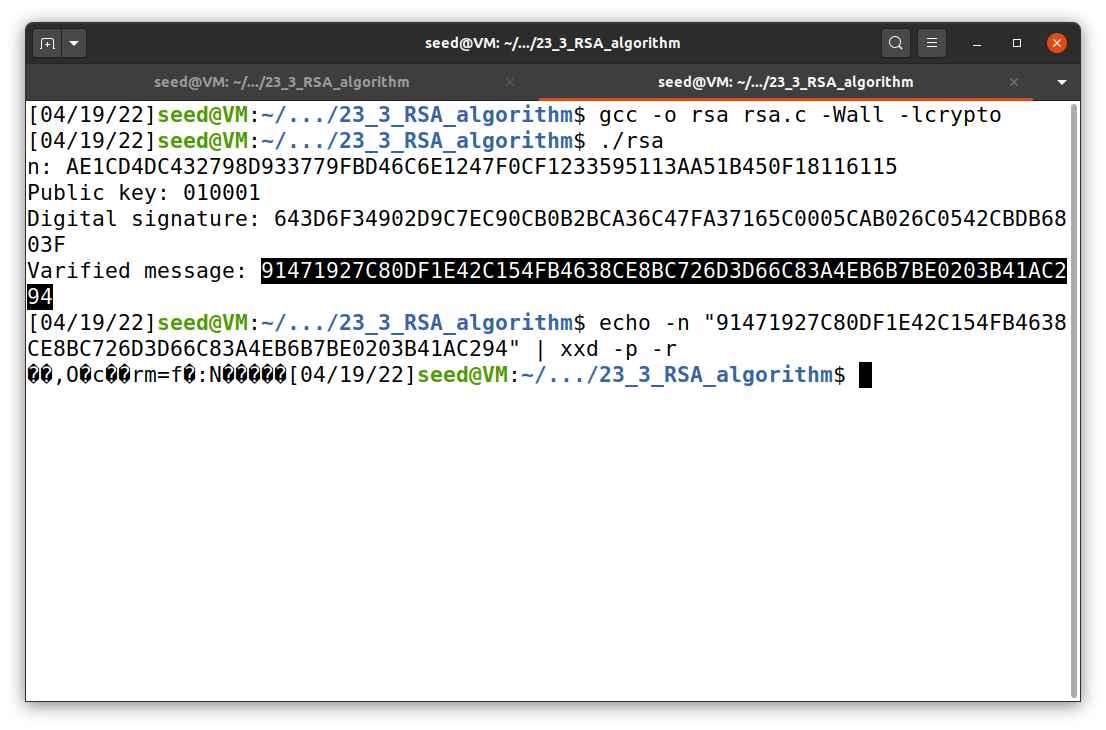
*The results for task 4*

**Graphical user interface, text, application

Description automatically generated**

**Figure 7**

*The results for task 4 with modified signature*



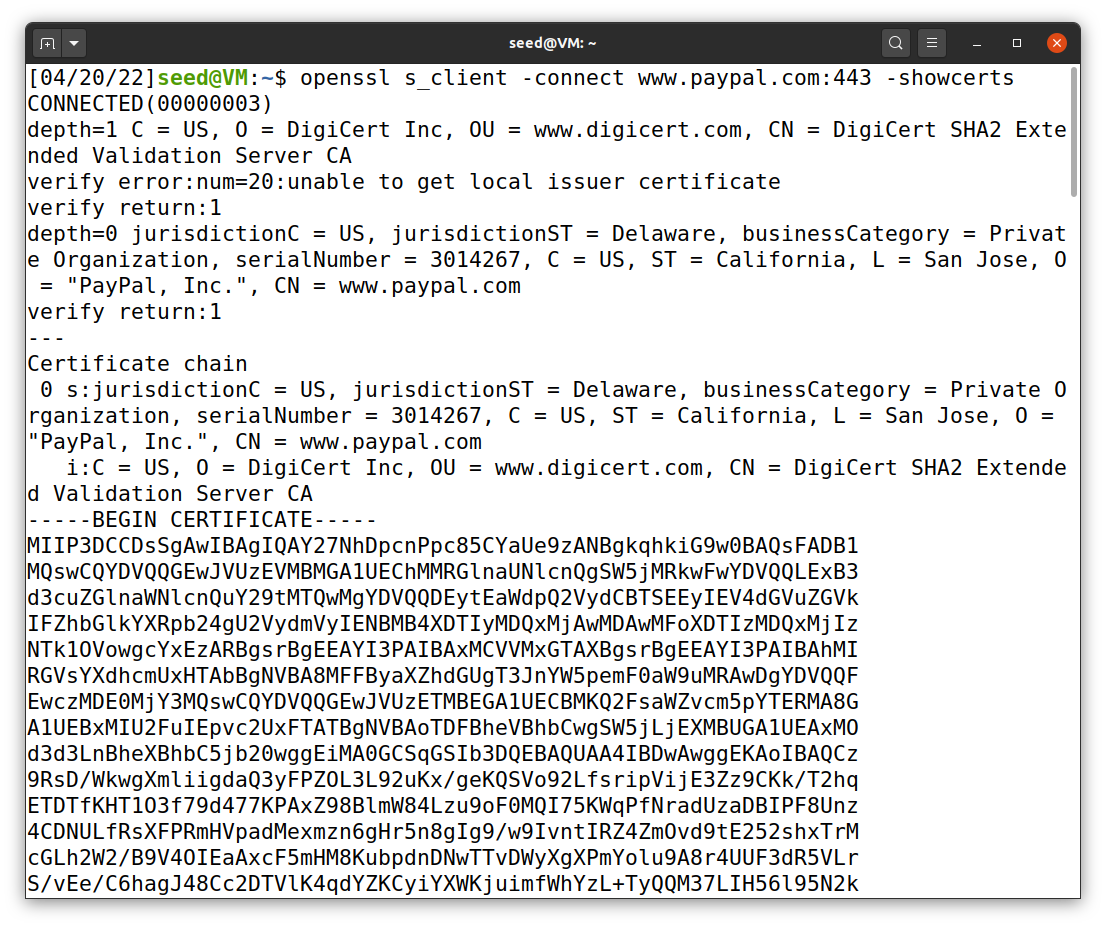
**Task 6: Manually Verifying an X.509 Certificate**

In this task, I first download a certificate from a real web server “www.paypal.com.” To do this, I used the command below. It put out the root and intermediate CAs’ certificate. It also printed other information out as shown in Figure 8 to 10. Before outputting the certificate, the command prints the subject field (the entry starting with s:) of the certificate and the issuer field (the entry starting with i:) provides the issuer’s information.

openssl s\_client -connect www.paypal.com:443 -showcerts

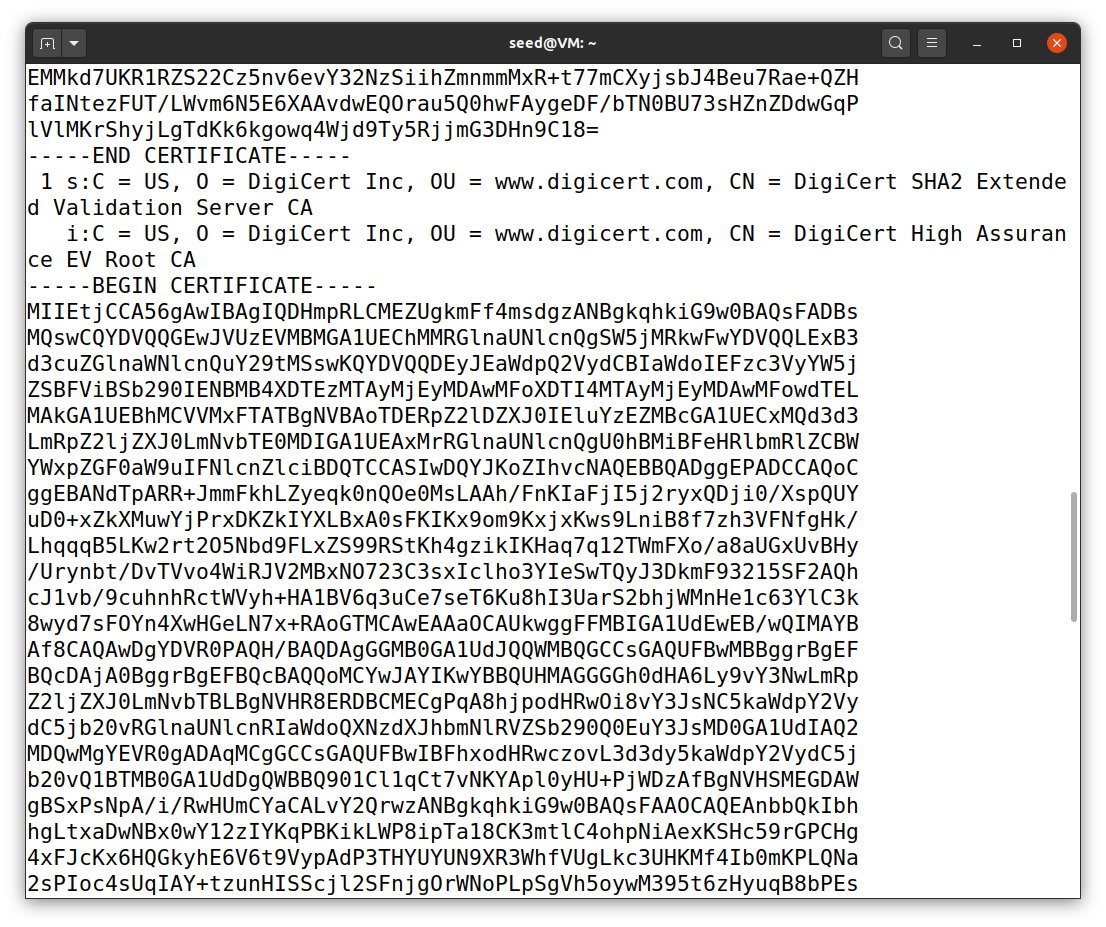
**Figure 8**

*Paypal root CA’s certificate*



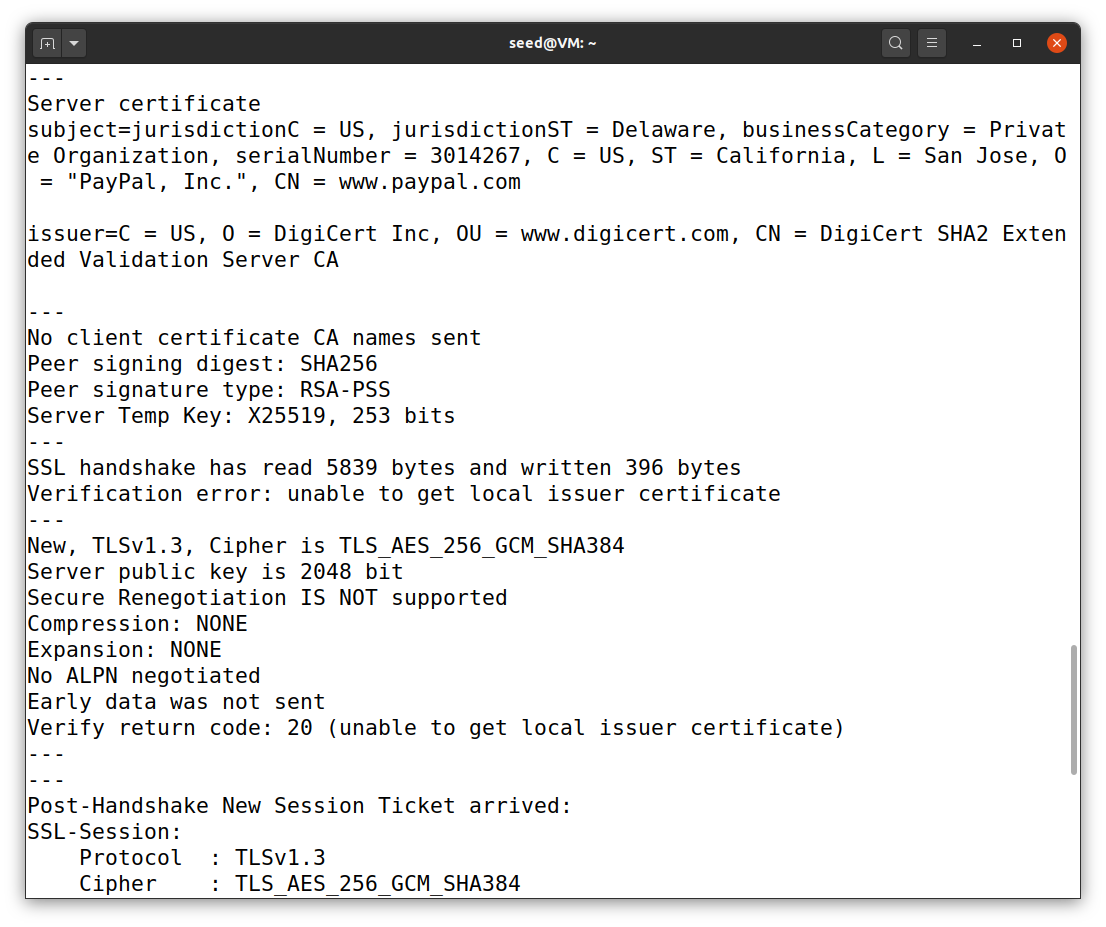
**Figure 9**

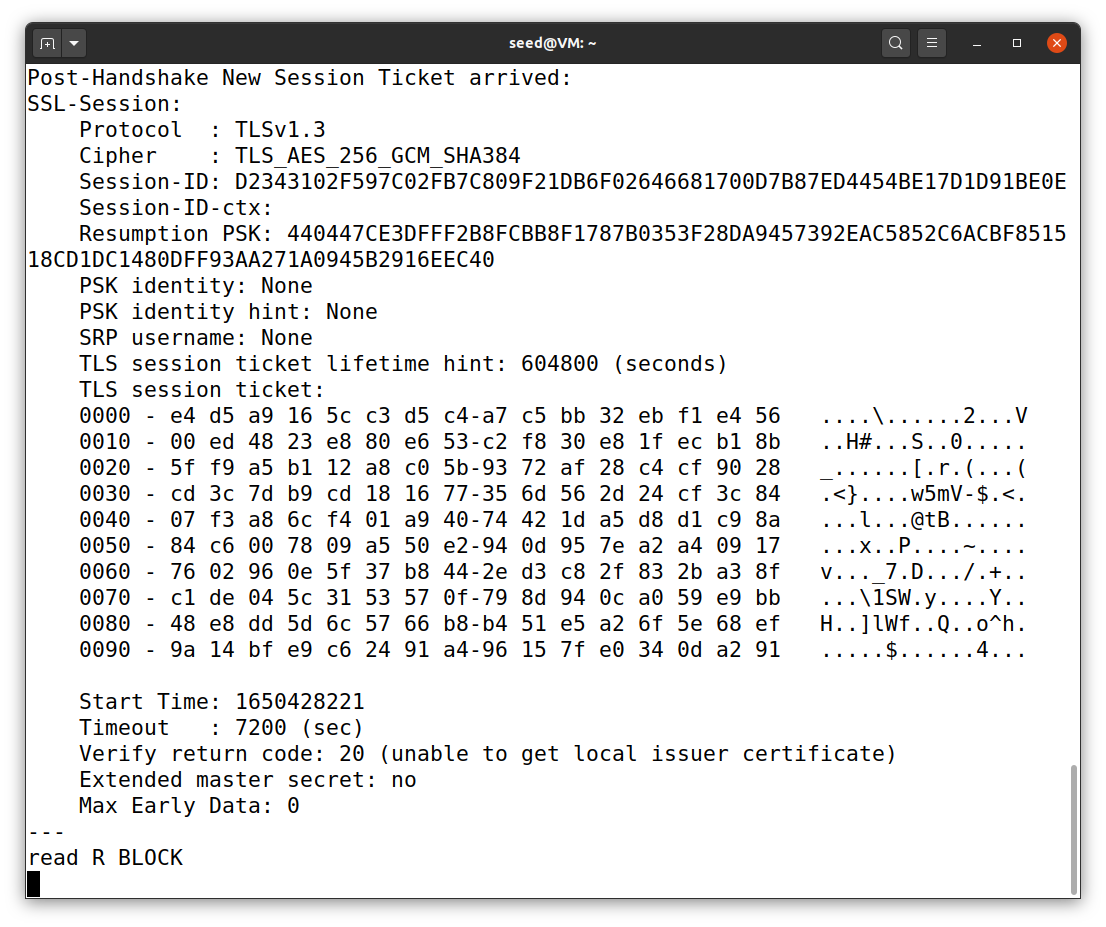
*Paypal intermediate CA’s certificate*



**Figure 10**

*Other information*





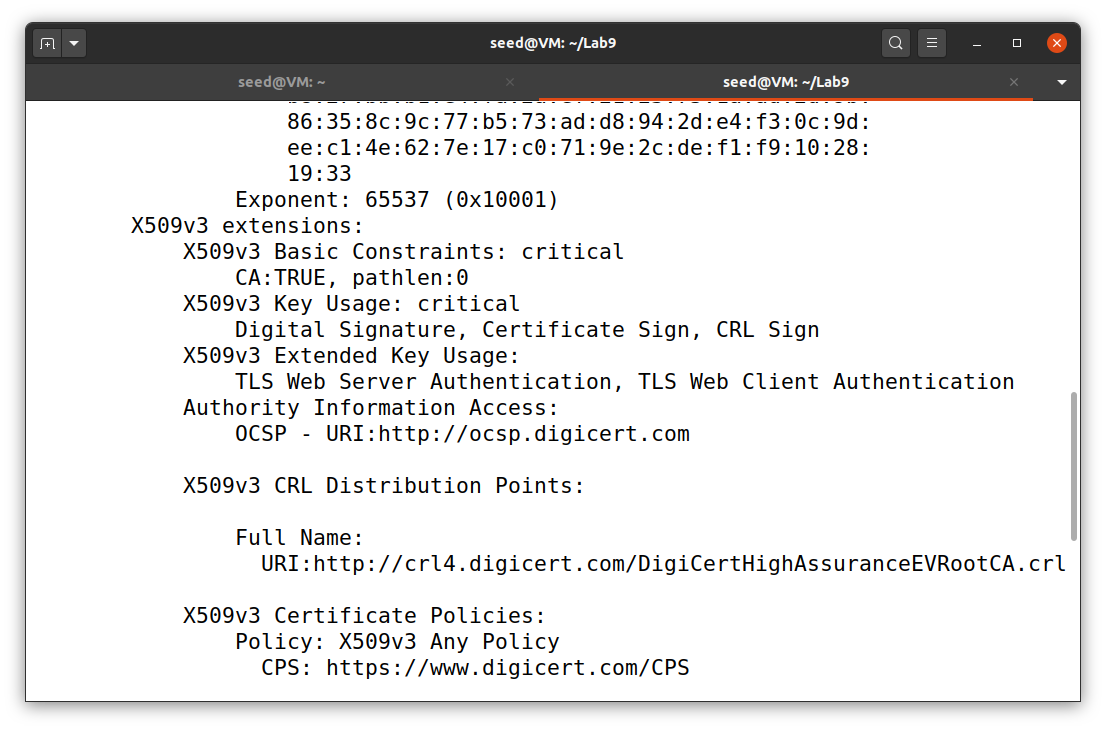
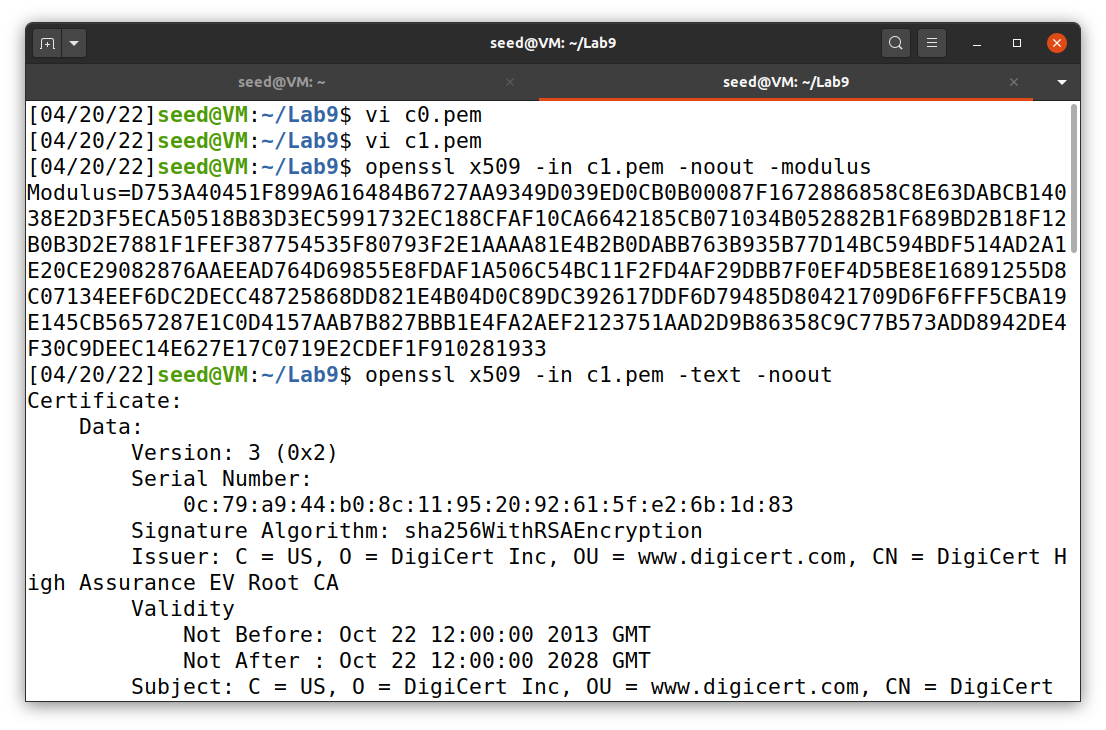
After that, I created two files called c0.pem and c1.pem using vi command. Inside the file, I copied and paste the certificate that related to the name of the files. Now, I extracted the public key (e, n) from the issuer’s certificate. To do this, I used the commands below. We can extract the value of n using -modulus. There is no specific command to extract e, but we can print out all the fields and can easily find the value of e. The modulus and e are shown in Figure 11.

openssl x509 -in c1.pem -noout -modulus

openssl x509 -in c1.pem -text -noout

**Figure 11**

*Public key*

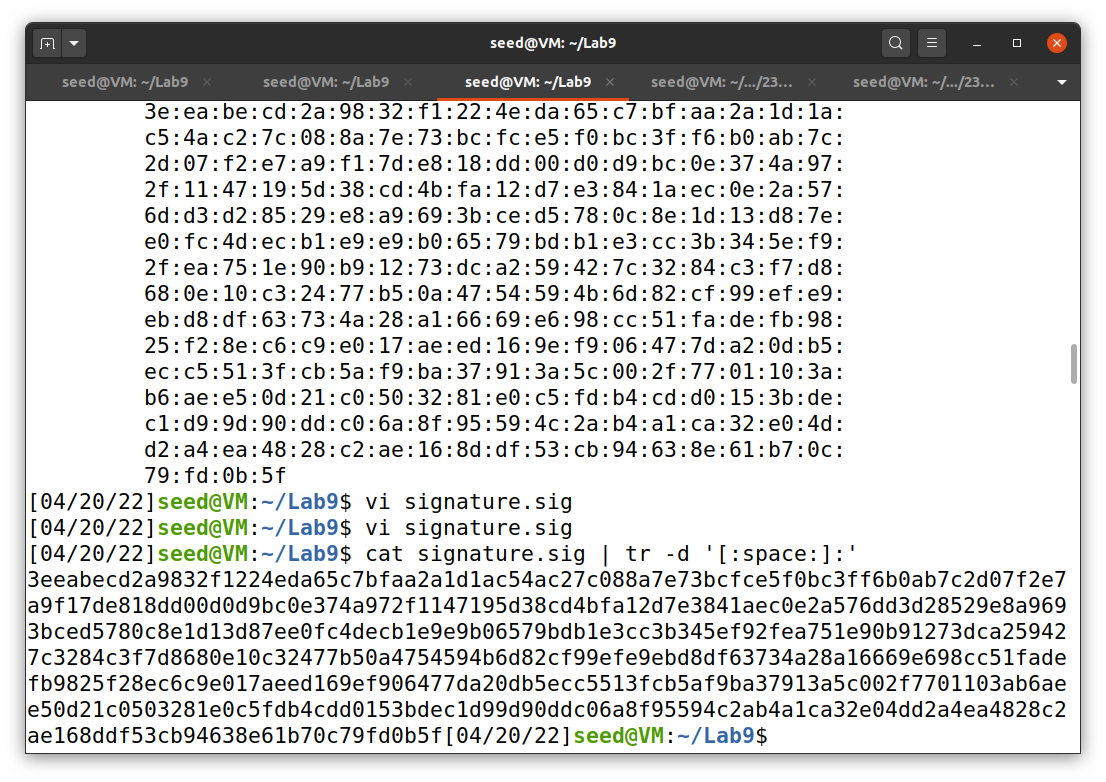


To obtain the signature from the server’s certificate, I used the openssl command below. After saving the signature to signature.sig using vi command, I removed the spaces and colons from the data using the command below. So, the output hex-string can be fed into the program. The result of this operation is shown in Figure 12.

openssl x509 -in c0.pem -text -noout

cat signature.sig | tr -d '[:space:]:'

**Figure 12**

*Signature*

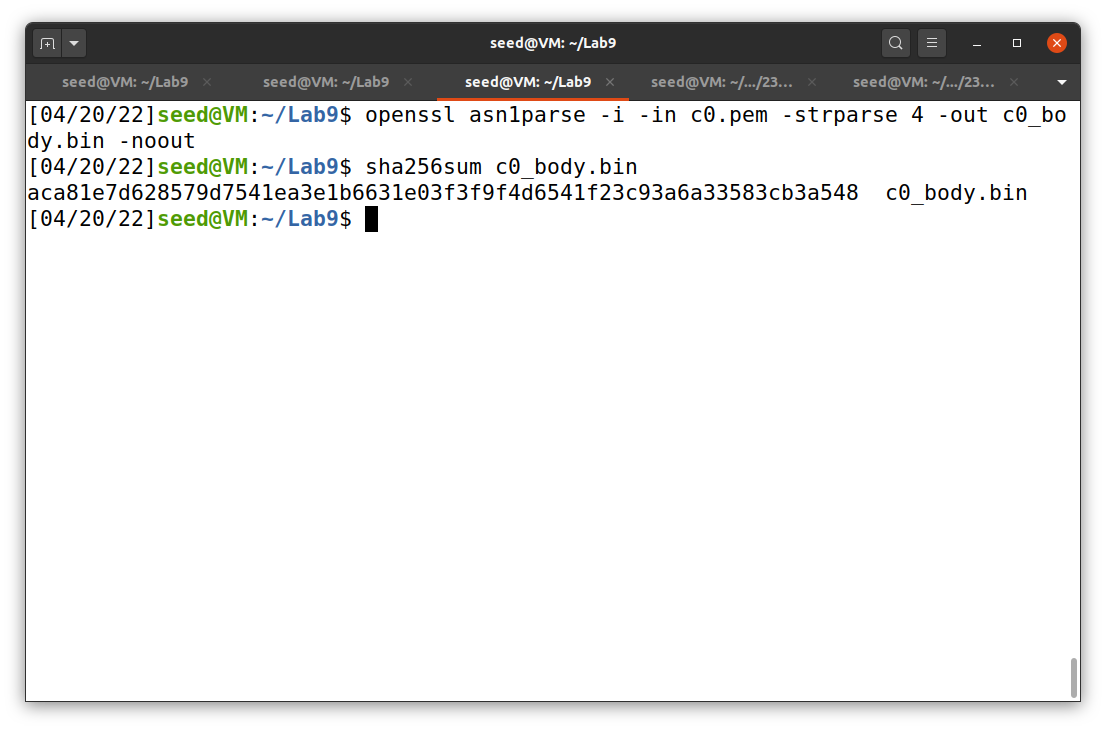
Before I can verify the signature, I need to generate the hash from a certificate. To do this, I used the command below. The hash was produced using sha256sum command. The result can be seen in Figure 13.

openssl asn1parse -i -in c0.pem -strparse 4 -out c0\_body.bin -noout

sha256sum c0\_body.bin

**Figure 13**

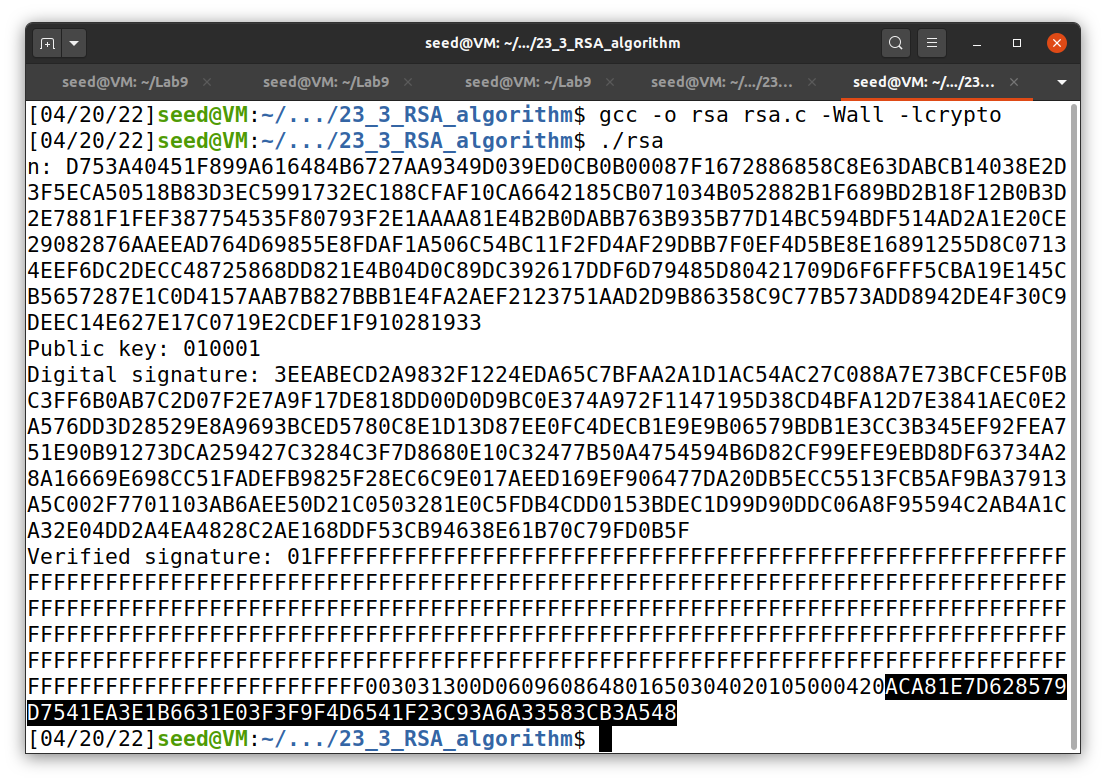
*Certificate hash*



Now we have all the information we need, including the CA’s public key (e), modulus (n), the CA’s signature (s), and the hash of the server’s certificate. I used the code provided on Brightspace. The “rsa.c” code that I used for each task will be reuploaded to my GitHub repository. The equation M = Se mod n is used to verify a digital signature. A part of verified result contains the hash of the server’s certificate, as highlighted in Figure 14.

**Figure 14**

*Certificate hash*

****

**References**

Du, W. (2019). Computer & internet security: A hands-on approach (2nd ed.). Independently published.

Du, W. (n.d.). RSA Encryption and Signature Lab. SeedLabs 2.0. https://seedsecuritylabs.org/‌Labs\_20.04/Crypto/Crypto\_RSA/