**Crypto and Network Security**

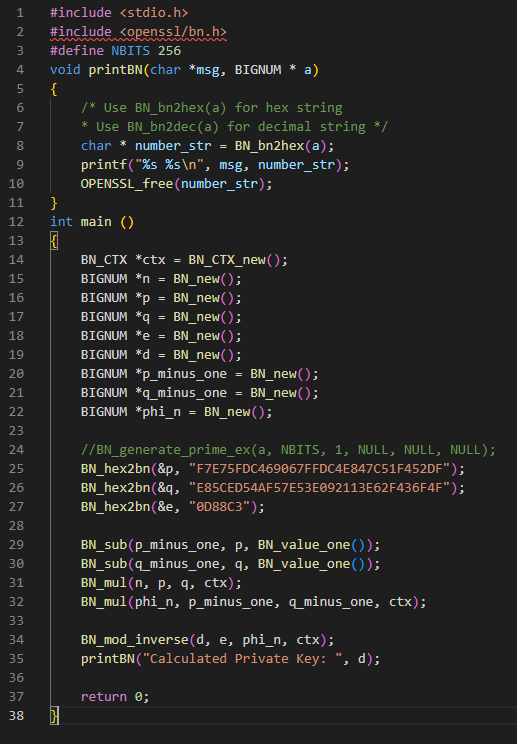
**Lab 10**

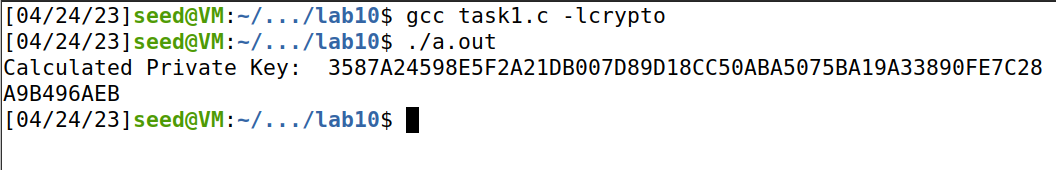
**Name: GAURAV SETTY**

**Email: settgm01@pfw.edu**

**Task 1: Deriving the private key**

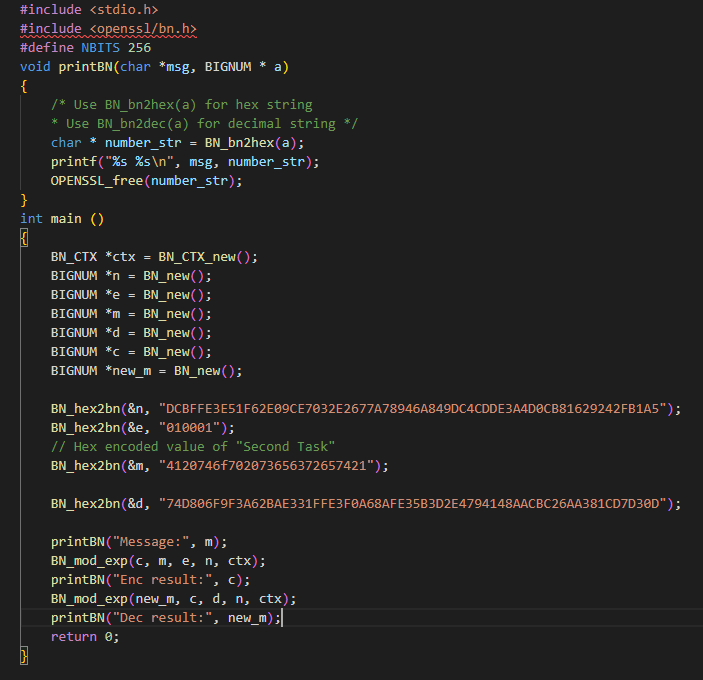
The code below utilizes the mod inverse to create a private key using the public key and the totient function.



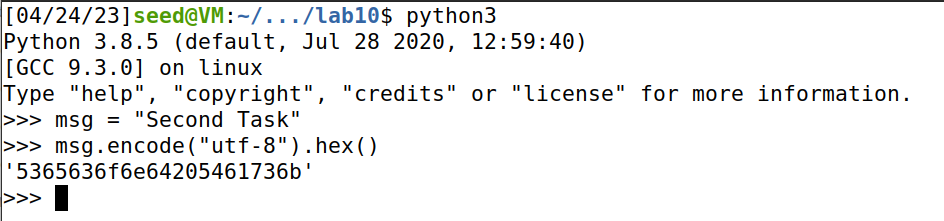


**Task 2: Encrypting a message**

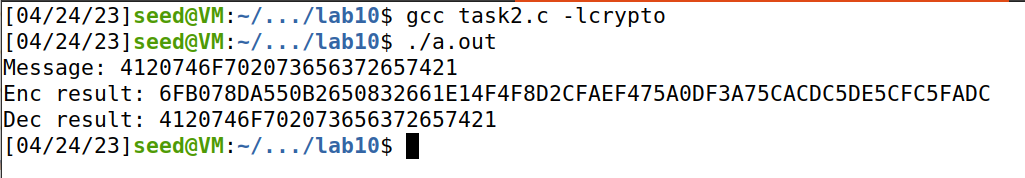
The above message's hex value is encrypted using the following code, which is then decrypted for validation.



The following calculates the hex value of any message using python’s hex() function.

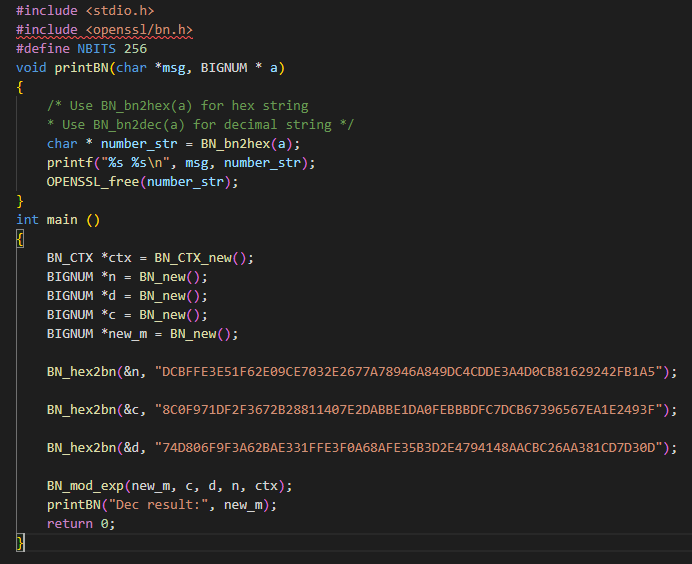


The below shows encryption and decryption outputs. Message is matching with decryption message. So, encryption is working.



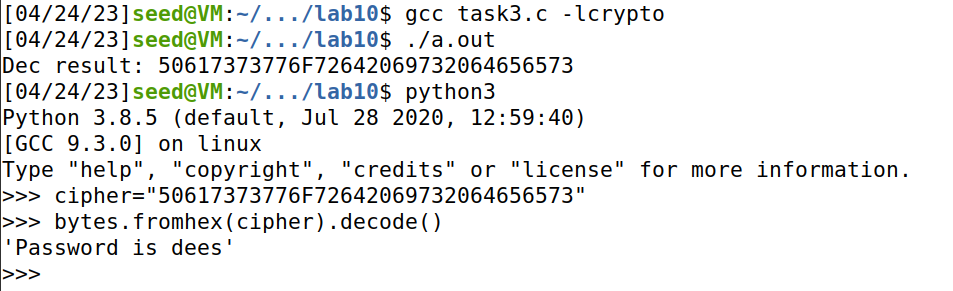
**Task 3: Decrypting a message**

The following code will decrypt a cipher text and produce the original message's hex value.



Let's utilize Python's bytes to decrypt the message with the fromhex() method, the message is transformed from hex to a utf-8 string.

'Password is dees' is the message that has been decrypted.

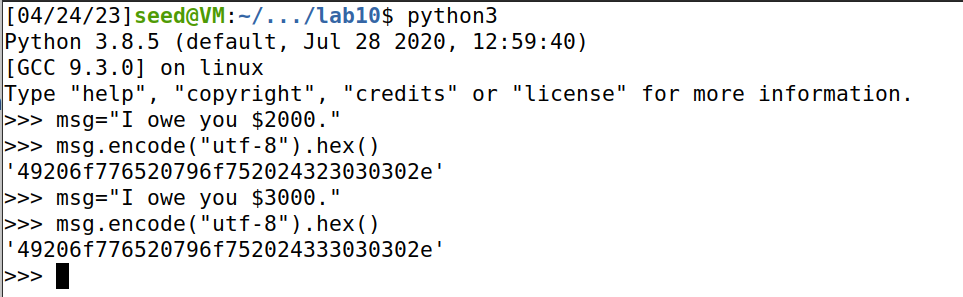


**Task 4: Signing a message**

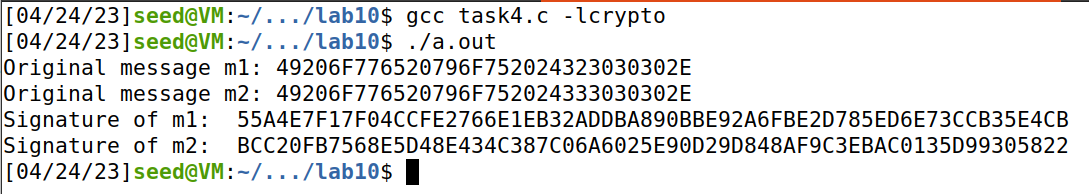
The following code takes two messages that are identical except for one digit, signs them both using a private key provided, and displays them out for comparison.



We begin by figuring out the hex values of each message. We can observe that there is only one digit between the two. The digital signature of each message is then compared by running sign.c.



The two signatures are very different.

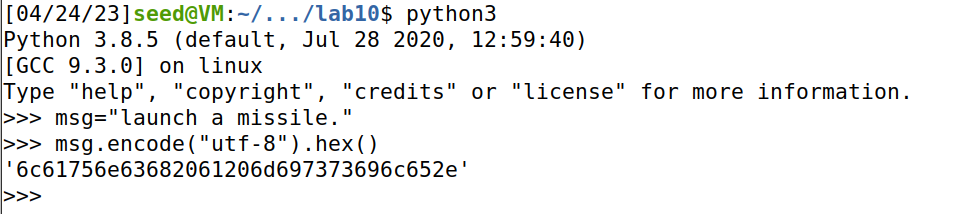


**Task 5: Verifying a signature**

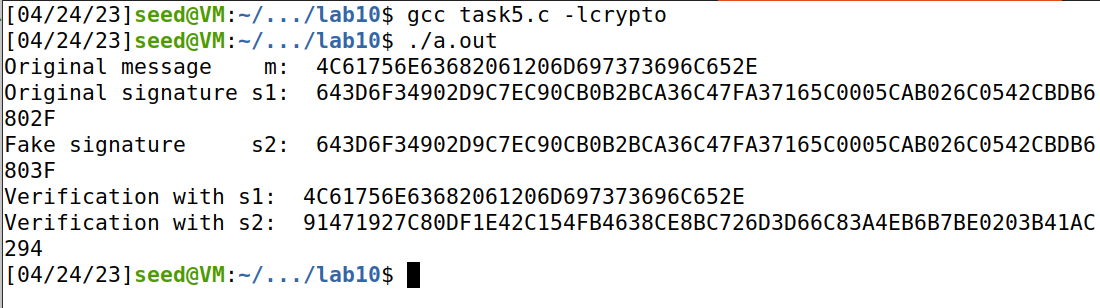
The code that follows verifies the validity of a message and the signature that goes with it. It also tries to verify the signature using a fake or corrupted one.



Prior to running verify, we determine the hex value of the message "Launch a missile."c



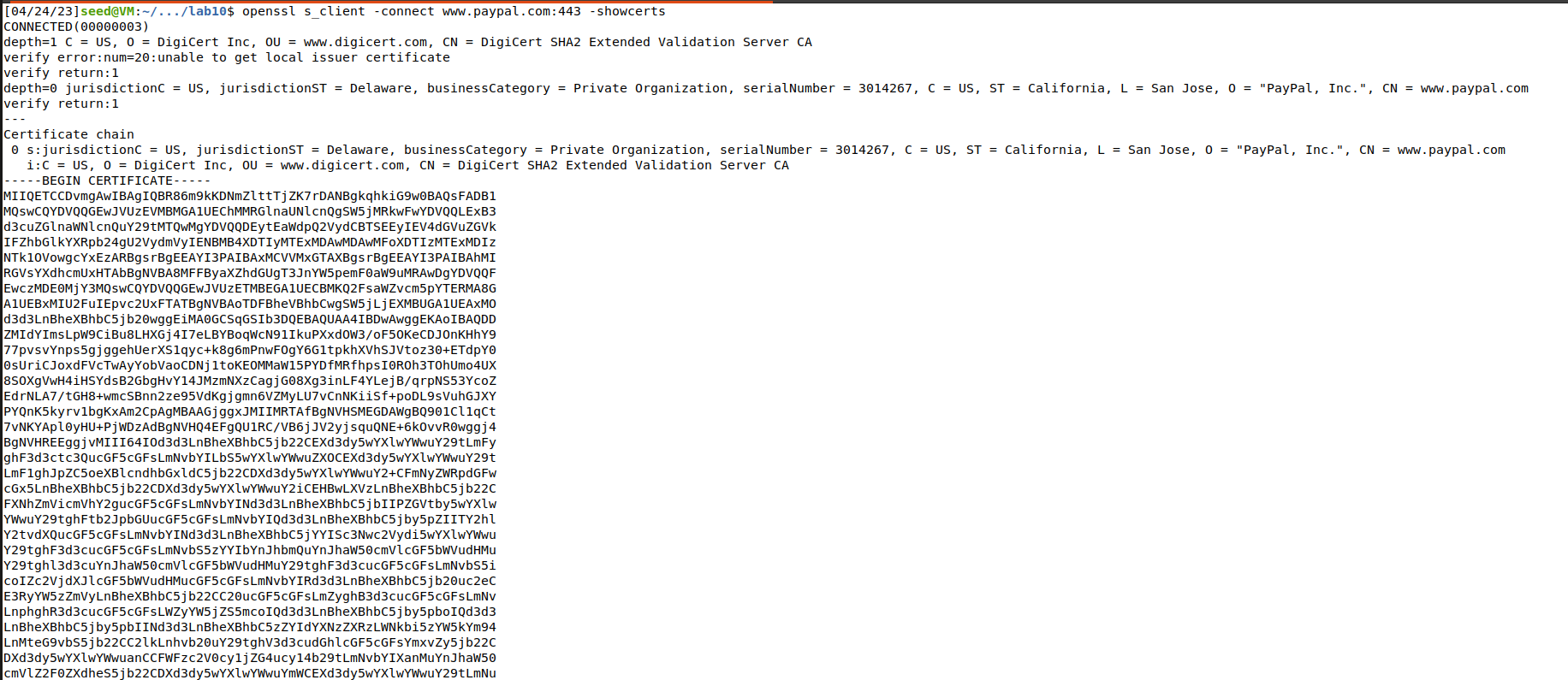
The final two bytes (2F and 3F) change between the two signatures. We can see that the first signature verifies the message, while the second signature does not.



**Task 6: Manually verifying an X.509 certificate**

**Step 1:** Download a certificate from a real web server

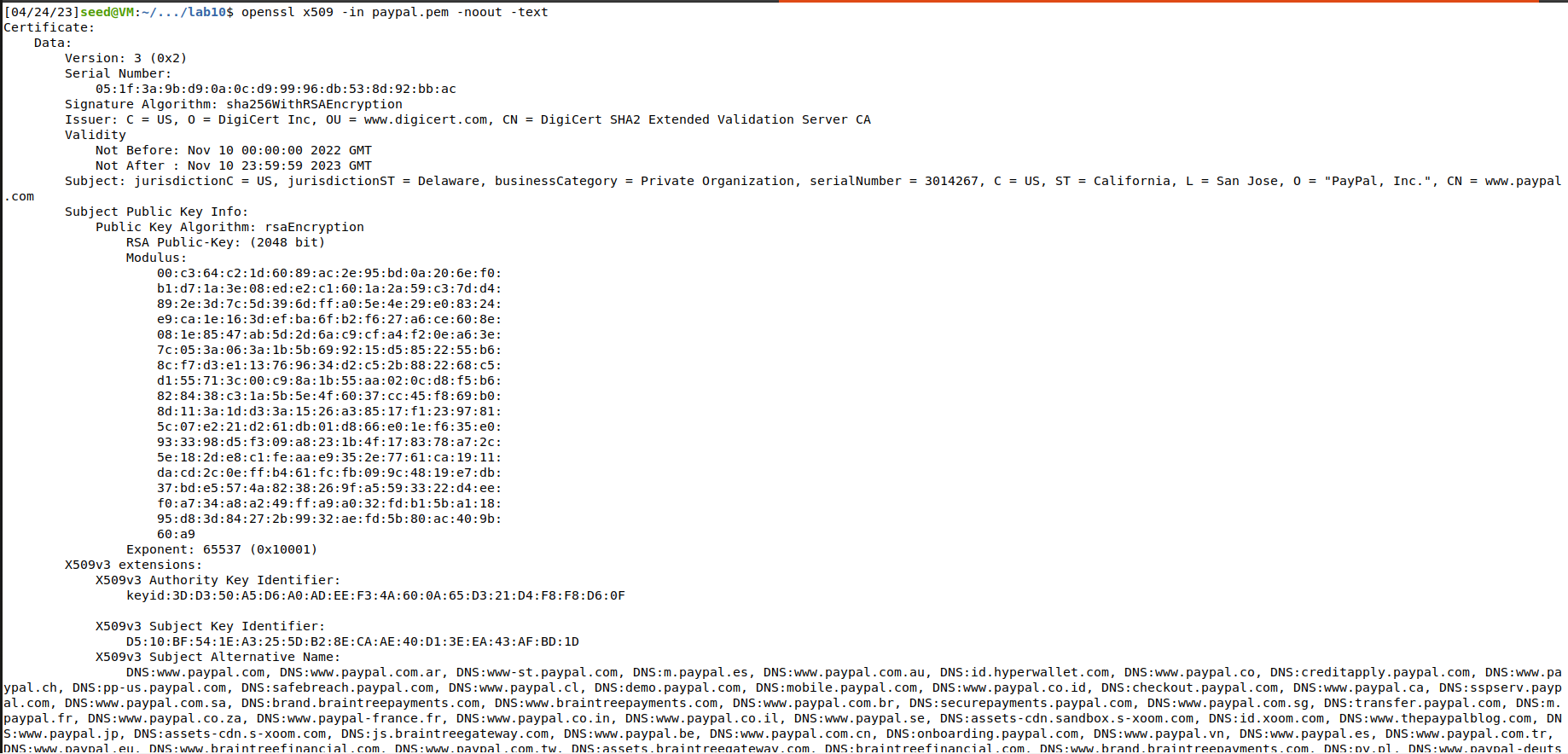
We extract the certificates for [www.paypal.com:443](http://www.paypal.com:443) using openssl s\_client -showcerts command



We copy the two certificates from the above command into paypal.pem and digicert.pem

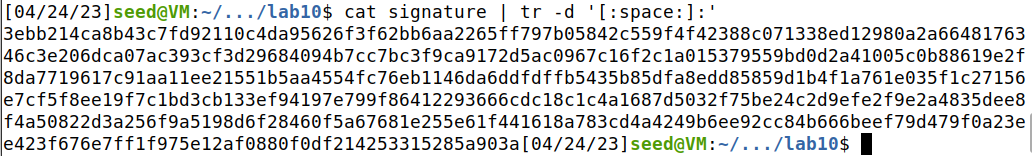
**Step 2**: Extract the public key (e, n) from the issuer’s certificate.

Then we parse the x509 certificate using openssl x509 and extract the modulus and the exponent



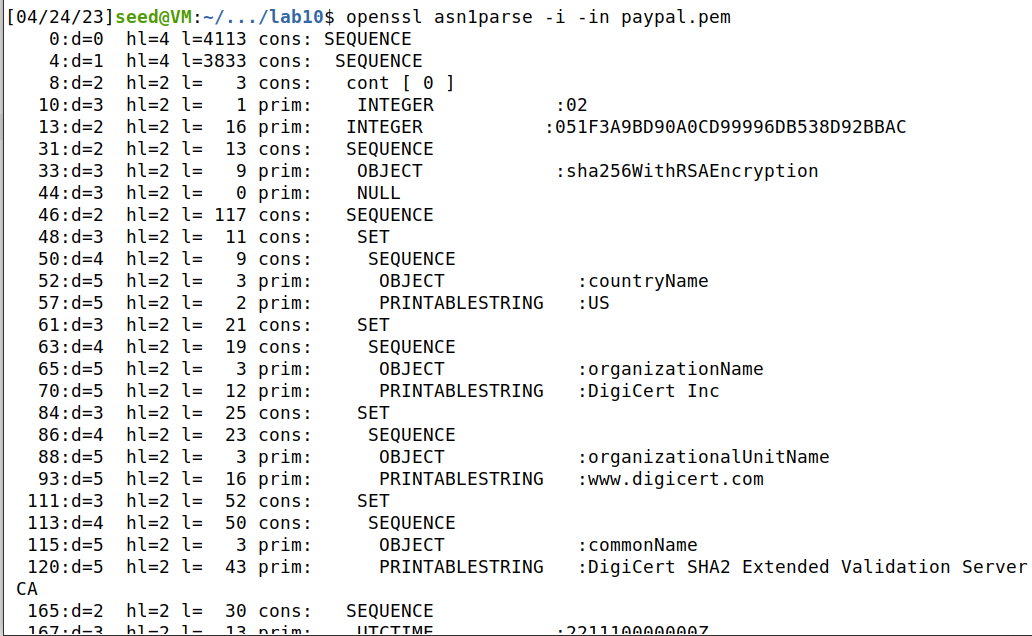
**Step 3:** Extract the signature from the server’s certificate.

To create a string, we take the signature from the command mentioned earlier and remove the spaces and colons.

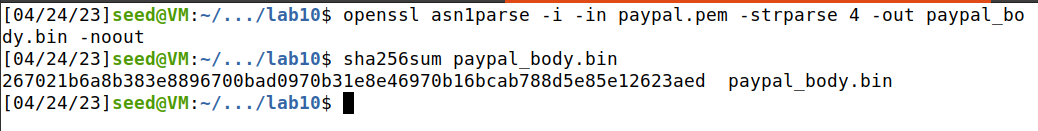


**Step 4**: Extract the body of the server’s certificate

Now we use the asn1parse command of openssl to parse the body of the certificate:



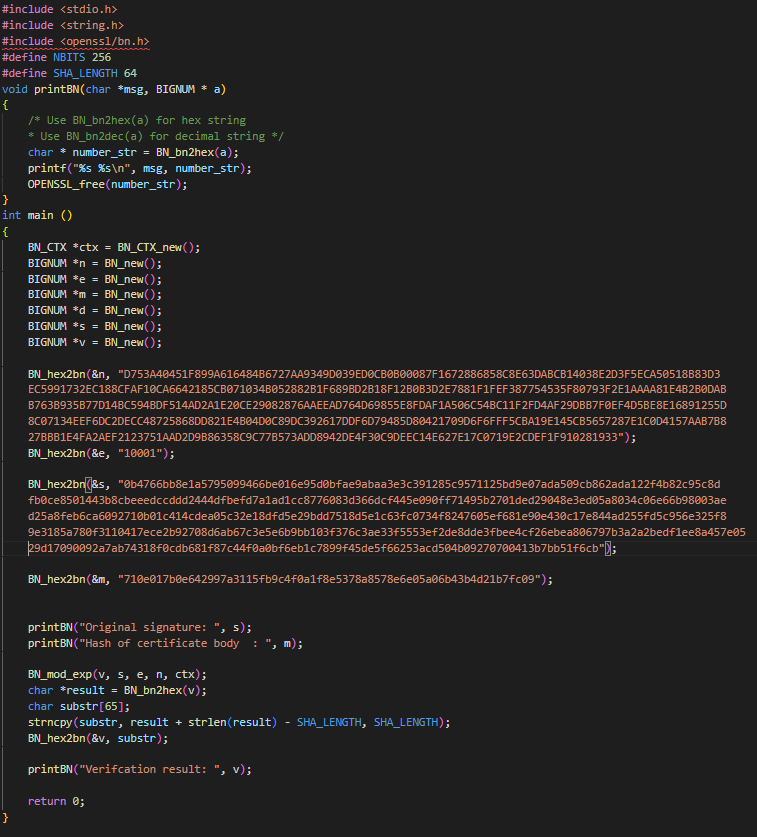
To parse the certificate starting at byte number 4, we use the -strparse 4 option.



Next, we determine the paypal\_body.bin's sha256sum.

**Step 5**: Verify the signature

The following code generates the verification result (decryption using the public key of the certificate) and displays it. It requires the modulus, exponent, signature, and hash of the certificate body.



We can observe that the hash of the certificate body and the original signature's decryption using the certificate's public key are identical.

The validity of the certificate is thus confirmed.

