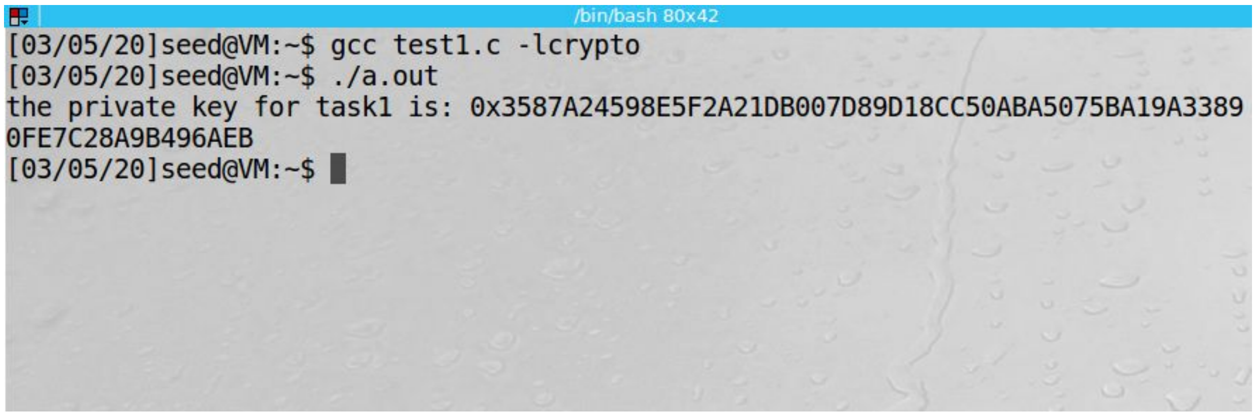
Ethan Booker
RSA Public-Key Encryption and Signature Lab
3/5/2020

Description: To get hands-on experience on the RSA algorithm of going through each essential step of the RSA algorithm on actual numbers

Section 3.1 Task 1: Deriving the Private Key

Description: The goal was to derive the private key with the given values. We had our two primes and mod value, then just called a private function to the private key.
Evidence:



```
[03/05/20]seed@VM:~$ gcc test1.c -lcrypto
[03/05/20]seed@VM:~$ ./a.out
the private key for task1 is: 0x3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A3389
0FE7C28A9B496AEB
[03/05/20]seed@VM:~$ 
```

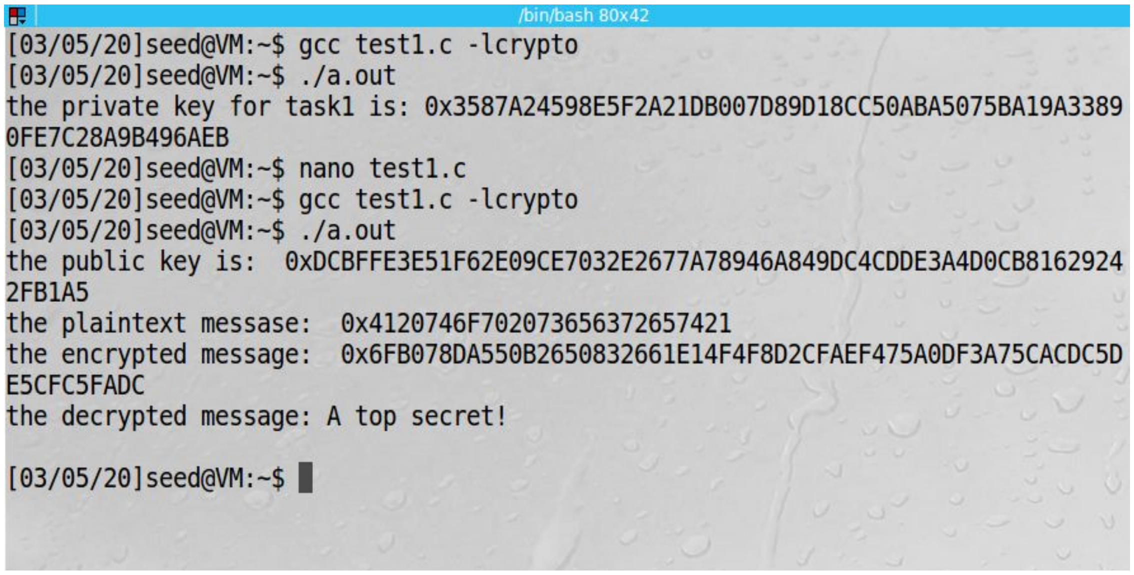
Analysis: The private key was generated by solving these formulas

- Φ(n) = (P-1)(Q-1)
- d = (k*Φ(n) + 1) / e

3.2 Task 2: Encrypting a Message

Description: The goal was to learn to encrypt a message using the given values.

Evidence:

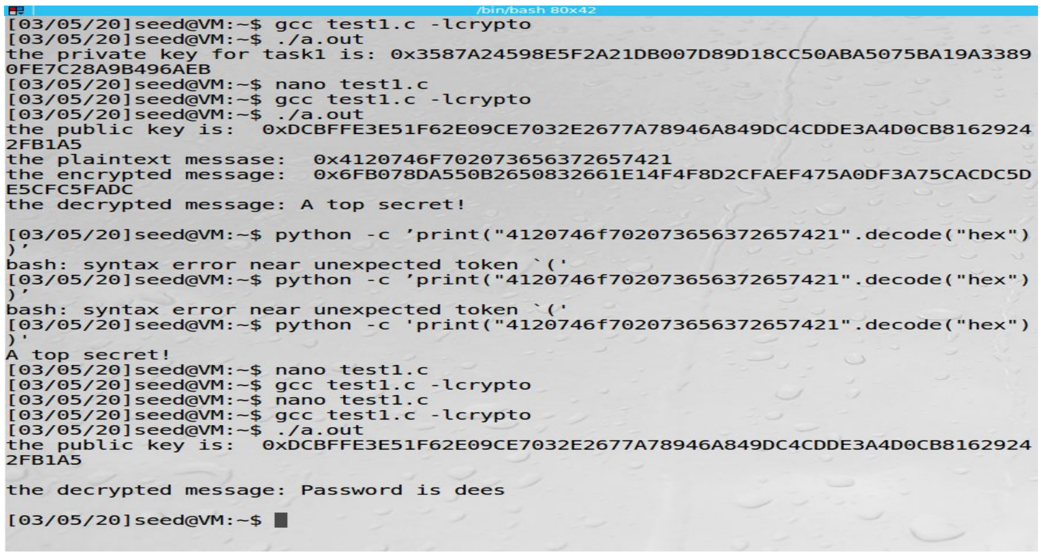


```
[03/05/20]seed@VM:~$ gcc test1.c -lcrypto
[03/05/20]seed@VM:~$ ./a.out
the private key for task1 is: 0x3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A3389
0FE7C28A9B496AEB
[03/05/20]seed@VM:~$ nano test1.c
[03/05/20]seed@VM:~$ gcc test1.c -lcrypto
[03/05/20]seed@VM:~$ ./a.out
the public key is:  0xDCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB8162924
2FB1A5
the plaintext message:  0x4120746F702073656372657421
the encrypted message:  0x6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5D
E5CFC5FADC
the decrypted message: A top secret!

[03/05/20]seed@VM:~$ 
```

Analysis: Since we had the public key and private to verify, we just had to encrypt the message and verify by decrypting. And since they matched their respective counter part values, it indeed worked.

3.3 Task 3: Decrypting a Message

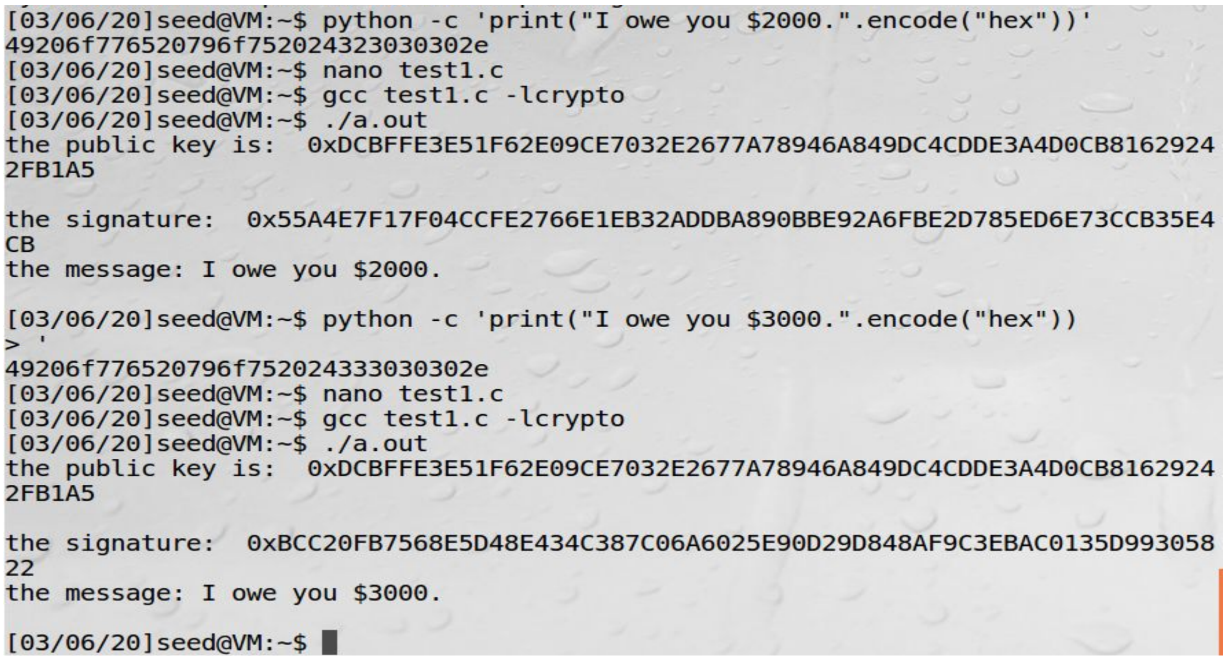
Description: The goal was to decrypt a message with the following values and convert it back to plain ASCII string.

Evidence: 

```
[03/05/20]seed@VM:~$ gcc test1.c -lcrypto
[03/05/20]seed@VM:~$ ./a.out
the private key for task1 is: 0x3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A3389
0FE7C28A9B496AEB
[03/05/20]seed@VM:~$ nano test1.c
[03/05/20]seed@VM:~$ gcc test1.c -lcrypto
[03/05/20]seed@VM:~$ ./a.out
the public key is:  0xDCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB8162924
2FB1A5
the plaintext message:  0x4120746F702073656372657421
the encrypted message:  0x6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5D
E5CFC5FADC
the decrypted message: A top secret!

[03/05/20]seed@VM:~$ python -c ’print("4120746f702073656372657421".decode("hex")
)’
bash: syntax error near unexpected token `('
[03/05/20]seed@VM:~$ python -c ’print("4120746f702073656372657421".decode("hex")
)’
bash: syntax error near unexpected token `('
[03/05/20]seed@VM:~$ python -c 'print("4120746f702073656372657421".decode("hex")
)'
A top secret!
[03/05/20]seed@VM:~$ nano test1.c
[03/05/20]seed@VM:~$ gcc test1.c -lcrypto
[03/05/20]seed@VM:~$ nano test1.c
[03/05/20]seed@VM:~$ gcc test1.c -lcrypto
[03/05/20]seed@VM:~$ ./a.out
the public key is:  0xDCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB8162924
2FB1A5

the decrypted message: Password is dees

[03/05/20]seed@VM:~$ 
```

Analysis: We are given our public and private key value and interestingly enough it printed out the the root password. I am not sure if that is a bug or not or that’s intended but this process was similar to 3.2 task. So I just used my decrypt function on the ciphertext.

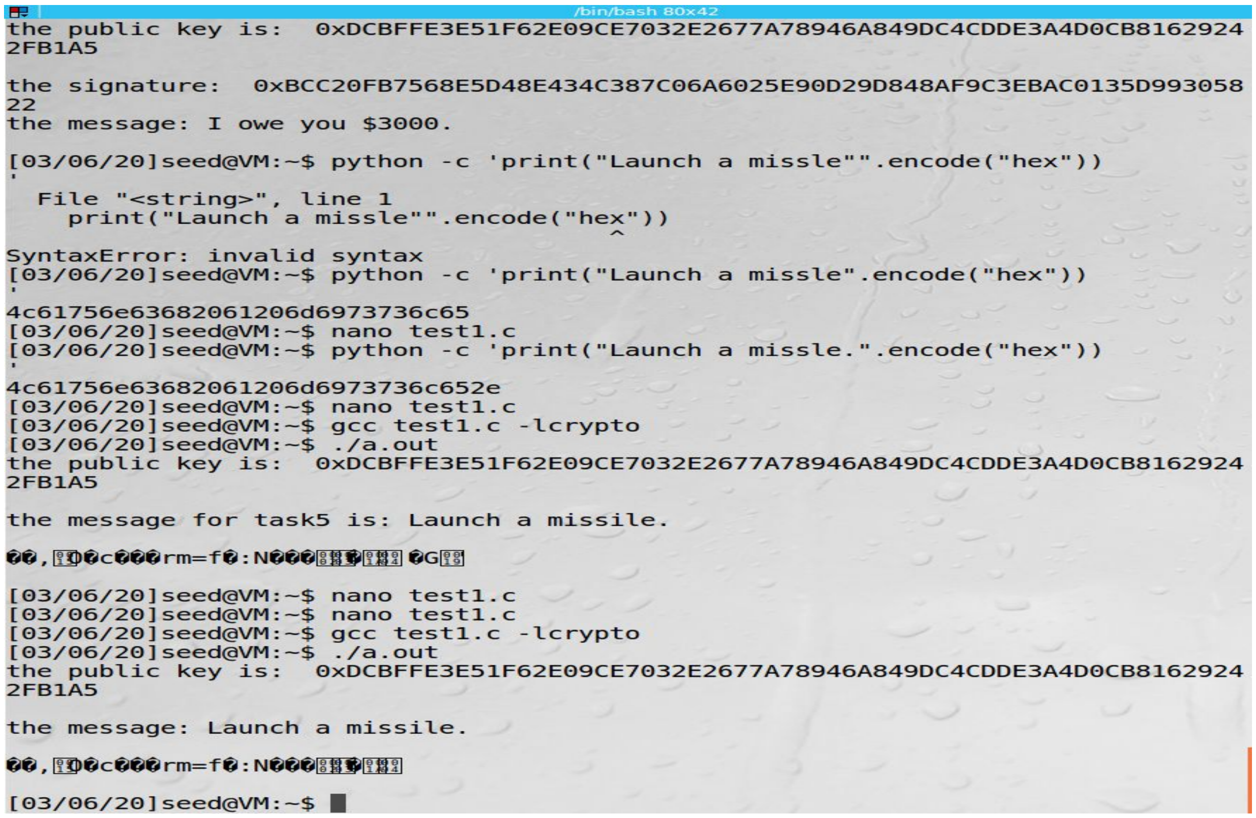
3.4 Task 4: Signing a Message
Description: The goal was to generate a signature for the following message and see what happens when you change the message.
Evidence:



```
[03/06/20]seed@VM:~$ python -c 'print("I owe you $2000.".encode("hex"))'
49206f776520796f752024323030302e
[03/06/20]seed@VM:~$ nano test1.c
[03/06/20]seed@VM:~$ gcc test1.c -lcrypto
[03/06/20]seed@VM:~$ ./a.out
the public key is:  0xDCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB8162924
2FB1A5

the signature:  0x55A4E7F17F04CCFE2766E1EB32ADDBA890BBE92A6FBE2D785ED6E73CCB35E4
CB
the message: I owe you $2000.

[03/06/20]seed@VM:~$ python -c 'print("I owe you $3000.".encode("hex"))
> '
49206f776520796f752024333030302e
[03/06/20]seed@VM:~$ nano test1.c
[03/06/20]seed@VM:~$ gcc test1.c -lcrypto
[03/06/20]seed@VM:~$ ./a.out
the public key is:  0xDCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB8162924
2FB1A5

the signature:  0xBCC20FB7568E5D48E434C387C06A6025E90D29D848AF9C3EBAC0135D993058
22
the message: I owe you $3000.

[03/06/20]seed@VM:~$ █
```

Analysis: we just needed to convert the message to a hex value and since we have the private key we could just encrpyt. To verify we just decrypted. One thing to notice is that the public key remains the same but the signature value will be different since our message is of diferent value

3.5 Task 5: : Verifying a Signature
Description: The goal was to verify if the signature is Alice's or not. We also explore what happens if the signature had slightly different values.
Evidence:



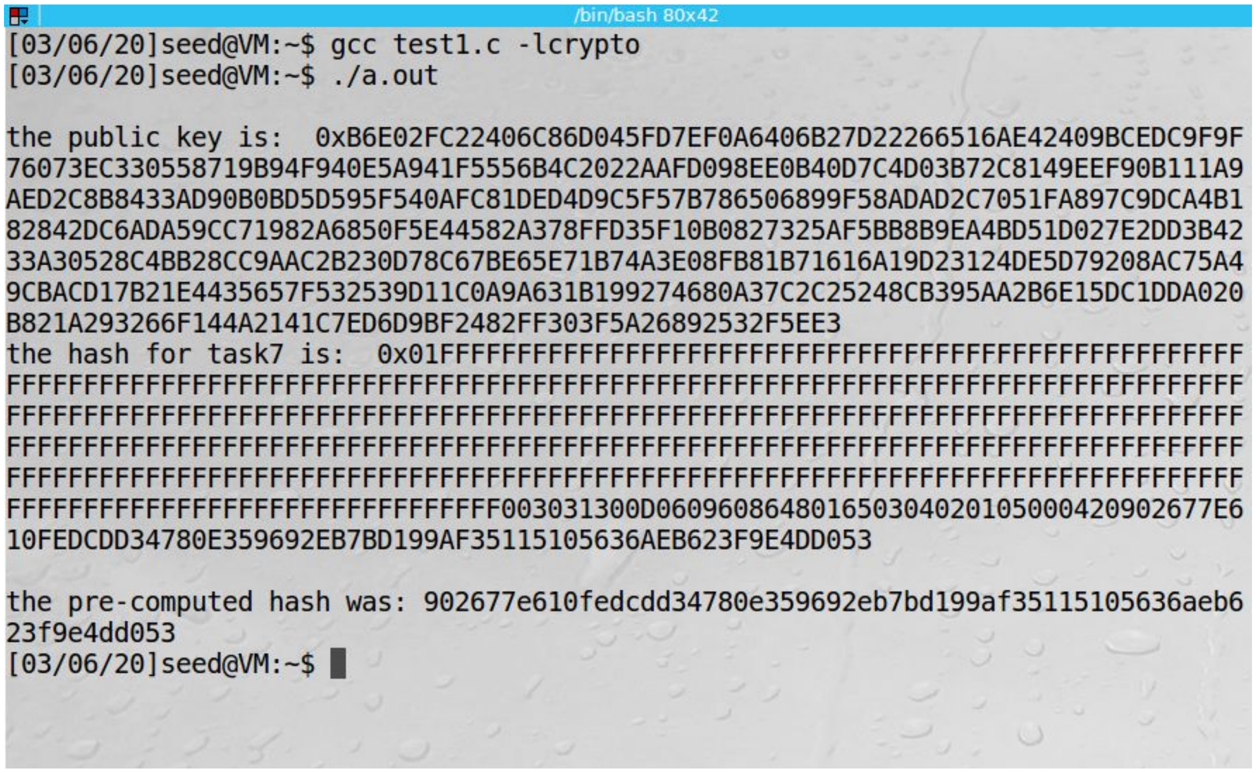
```
the public key is:  0xDCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB8162924
2FB1A5

the signature:  0xBCC20FB7568E5D48E434C387C06A6025E90D29D848AF9C3EBAC0135D993058
22
the message: I owe you $3000.

[03/06/20]seed@VM:~$ python -c 'print("Launch a missle"".encode("hex"))
'
  File "<string>", line 1
    print("Launch a missle"".encode("hex"))
                                      ^
SyntaxError: invalid syntax
[03/06/20]seed@VM:~$ python -c 'print("Launch a missle".encode("hex"))
'
4c61756e63682061206d6973736c65
[03/06/20]seed@VM:~$ nano test1.c
[03/06/20]seed@VM:~$ python -c 'print("Launch a missle.".encode("hex"))
'
4c61756e63682061206d6973736c652e
[03/06/20]seed@VM:~$ nano test1.c
[03/06/20]seed@VM:~$ gcc test1.c -lcrypto
[03/06/20]seed@VM:~$ ./a.out
the public key is:  0xDCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB8162924
2FB1A5

the message for task5 is: Launch a missile.

��,�0�c���rm=f�:N�������� �G�

[03/06/20]seed@VM:~$ nano test1.c
[03/06/20]seed@VM:~$ nano test1.c
[03/06/20]seed@VM:~$ gcc test1.c -lcrypto
[03/06/20]seed@VM:~$ ./a.out
the public key is:  0xDCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB8162924
2FB1A5

the message: Launch a missile.

��,�0�c���rm=f�:N��������

[03/06/20]seed@VM:~$ 
```

Analysis: We used our public key to decrypt a message that has been encrypted with a private key. Then we compared the results. We corrupted the signature then tried to verify again but as a result you can see that the output is corrupted since that's not the expected value we wanted from the encryption.

3.6 Task 6: Manually Verifying an X.509 Certificate

Description: The goal was to manually verify an X.509 certificate using our program.

Evidence:



```
[03/06/20]seed@VM:~$ gcc test1.c -lcrypto
[03/06/20]seed@VM:~$ ./a.out

the public key is:  0xB6E02FC22406C86D045FD7EF0A6406B27D22266516AE42409BCEDC9F9F
76073EC330558719B94F940E5A941F5556B4C2022AAFD098EE0B40D7C4D03B72C8149EEF90B111A9
AED2C8B8433AD90B0BD5D595F540AFC81DED4D9C5F57B786506899F58ADAD2C7051FA897C9DCA4B1
82842DC6ADA59CC71982A6850F5E44582A378FFD35F10B0827325AF5BB8B9EA4BD51D027E2DD3B42
33A30528C4BB28CC9AAC2B230D78C67BE65E71B74A3E08FB81B71616A19D23124DE5D79208AC75A4
9CBACD17B21E4435657F532539D11C0A9A631B199274680A37C2C25248CB395AA2B6E15DC1DDA020
B821A293266F144A2141C7ED6D9BF2482FF303F5A26892532F5EE3
the hash for task7 is:  0x01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF003031300D060960864801650304020105000420902677E6
10FEDCDD34780E359692EB7BD199AF35115105636AEB623F9E4DD053

the pre-computed hash was: 902677e610fedcdd34780e359692eb7bd199af35115105636aeb6
23f9e4dd053
[03/06/20]seed@VM:~$ 
```

Analysis: We extracted the public key using openssl x509 -in c1.pem -noout -modulus, we then assigned the modulus, then with a slightly different flags for command for the public key we extracted the RSA-signed sha256 signature. Then we extracted the body then the hash, then decrypted the signature then print the decrypted signature. We can then assume that the first 32 bytes should match the hash generated from the body.

Conclusion:

It seems as though that RSA encryption is quite complex and required very intelligent minds to think of a clever algorithm to encrypt and decrypt such a highly regarded cryptography algorithm. There also seems to be a large amount of API support along with websites to test this algorithm and to display the potential of RSA encryption. I have a better understanding of the process of the RSA algorithm must do to ensure that the signature is who they are supposed to be and what happens if values are incorrect in the RSA algorithm in order to detect tampered values. I think this was a nice lab to learn more cryptography and one of it’s most popular encryption.