- **Task 1 Deriving the Private Key (Nafeesa):**

1. What is the value of d?

The value of d is (3 5 8 7A 2 4 5 9 8E 5F 2A 2 1 D B 0 0 7D 8 9D 1 8C C 5 0A B A 5 0 7 5B A 1 9A 3 3 8 9 0F E 7C 2 8A 9B 4 9 6A E B , E 1 0 3A B D 9 4 8 9 2E 3E 7 4A F D 7 2 4B F 2 8E 7 8 3 6 6D 9 6 7 6B C C C 7 0 1 1 8B D 0A A 1 9 6 8D B B 1 4 3D 1)

2. Task1 screenshots and observations:

For the provided values

p = F7E75FDC469067FFDC4E847C51F452DF

q = E85CED54AF57E53E092113E62F436F4F

e = 0D88C3

Usage of RSA algorithm formula in the program

//For public key: n = pq //

//For private key: phi(n) = (p-1)*(q-1)//

```
BN_mod_inverse(d, e,
phi,ctx);
printBN("private key", d, n);
```

- **Task 2 Encrypting a Message (Nafeesa):**

3. What is the hex string of “This is a Secret!”?

6F B 0 7 8D A 5 5 0B 2 6 5 0 8 3 2 6 6 1E 1 4F 4F 8D 2C F A E F 4 7 5A 0D F

3A 7 5C A C D C 5D E 5C F C 5F A D C

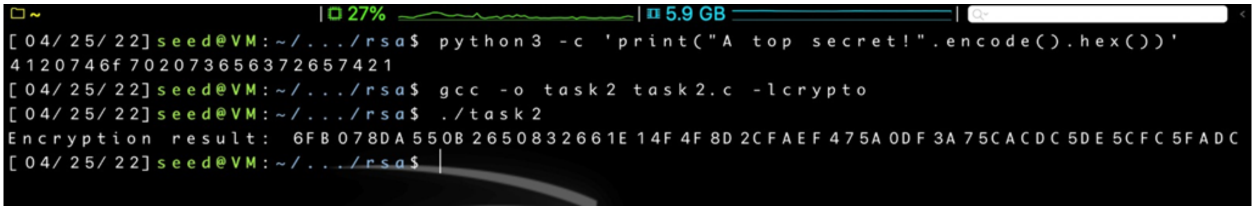
4. How do you convert it to hex?

By using the RSA algorithm formula in the bn_sample.c program.

//*encrypted Message: M^e mod n*//. This is the RSA algorithm formula to encrypt message added to the program

```
        // encrypt M: M^e mod n //
    BN_mod_exp(C, M, e, n, ctx);
    printBN("Encryption result:", C);
```

5. Task2 screenshots and observations



To the formula, M^e mod n assign the given values

n =

DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242

FB1A5

e = 010001 (in decimal= 65537)

M = A top secret!

d =

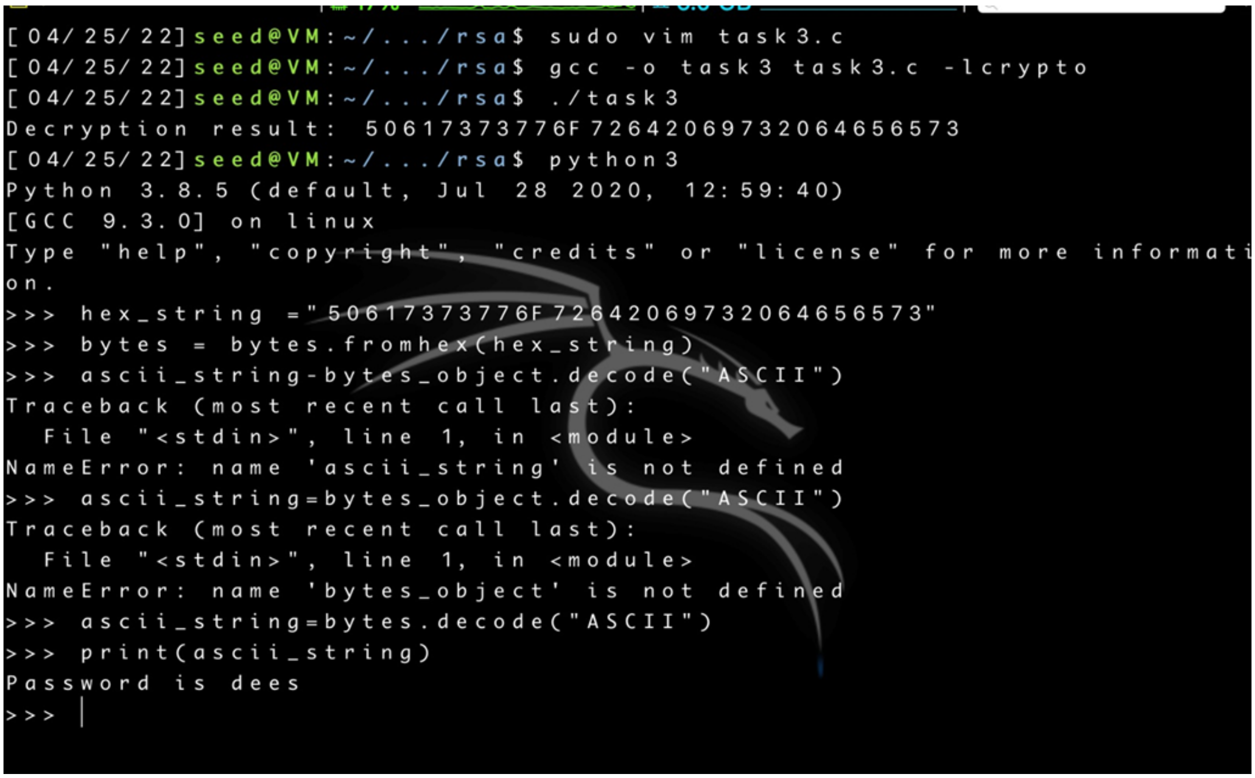
74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381C

D7D30D

Therefore, the encryption result is the result of M^e mod n

- **Task 3 Decrypting a Message (Nafeesa):**

6. Task3 screenshots and observations



```
[04/25/22]seed@VM:~/.../rsa$ sudo vim task3.c
[04/25/22]seed@VM:~/.../rsa$ gcc -o task3 task3.c -lcrypto
[04/25/22]seed@VM:~/.../rsa$ ./task3
Decryption result: 50617373776F726420697320646565673
[04/25/22]seed@VM:~/.../rsa$ python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more informati
on.
>>> hex_string ="50617373776F726420697320646565673"
>>> bytes = bytes.fromhex(hex_string)
>>> ascii_string-bytes_object.decode("ASCII")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'ascii_string' is not defined
>>> ascii_string=bytes_object.decode("ASCII")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'bytes_object' is not defined
>>> ascii_string=bytes.decode("ASCII")
>>> print(ascii_string)
Password is dees
>>> |
```

In addition to n, e, M and d values for this task the Ciphertext value is provided for decryption of the message

C =

8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F

To decrypt a message for a given ciphertext we used the RSA algorithm formula for decryption

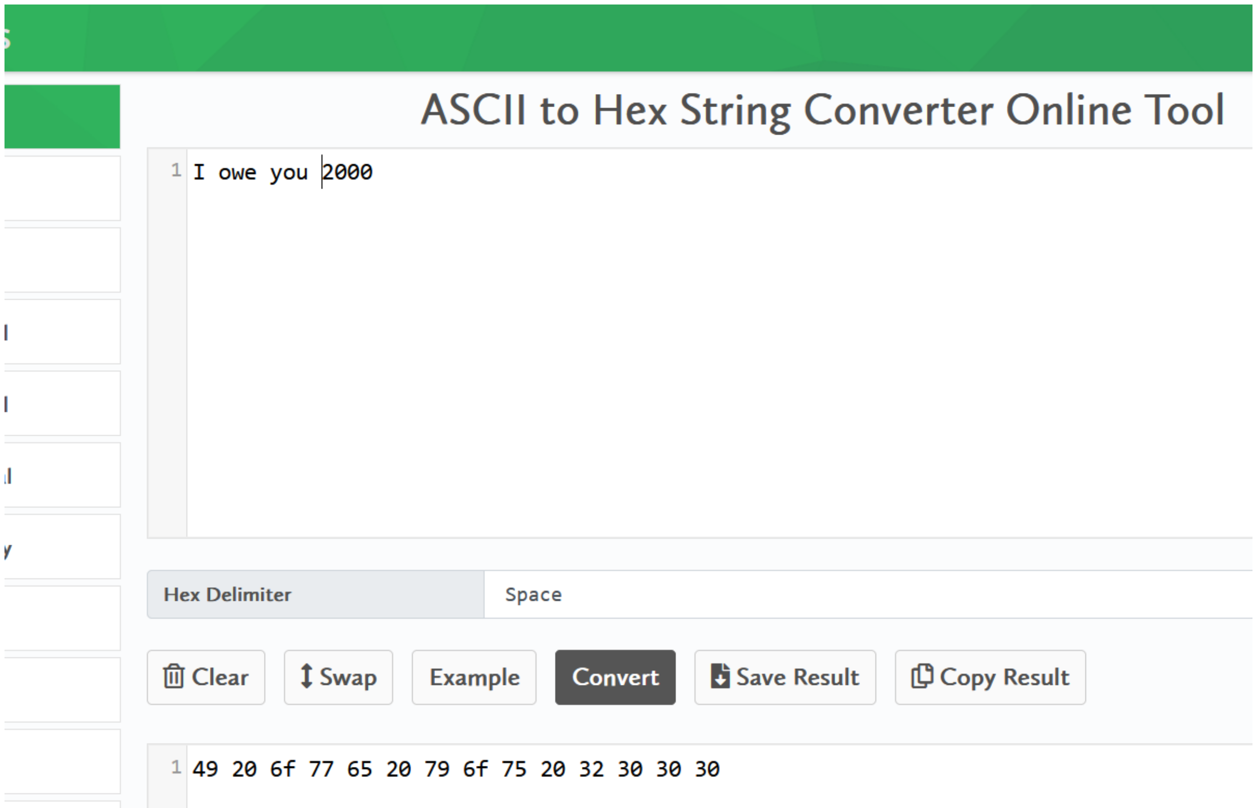
//decrypted Ciphertext: C^d mod n//

```
// decrypt C: C^d mod n//

BN_mod_exp(M, C, d, n, ctx);

printBN("Decryption result:", M);
```

- **Task 4 Signing a Message (JJ):**

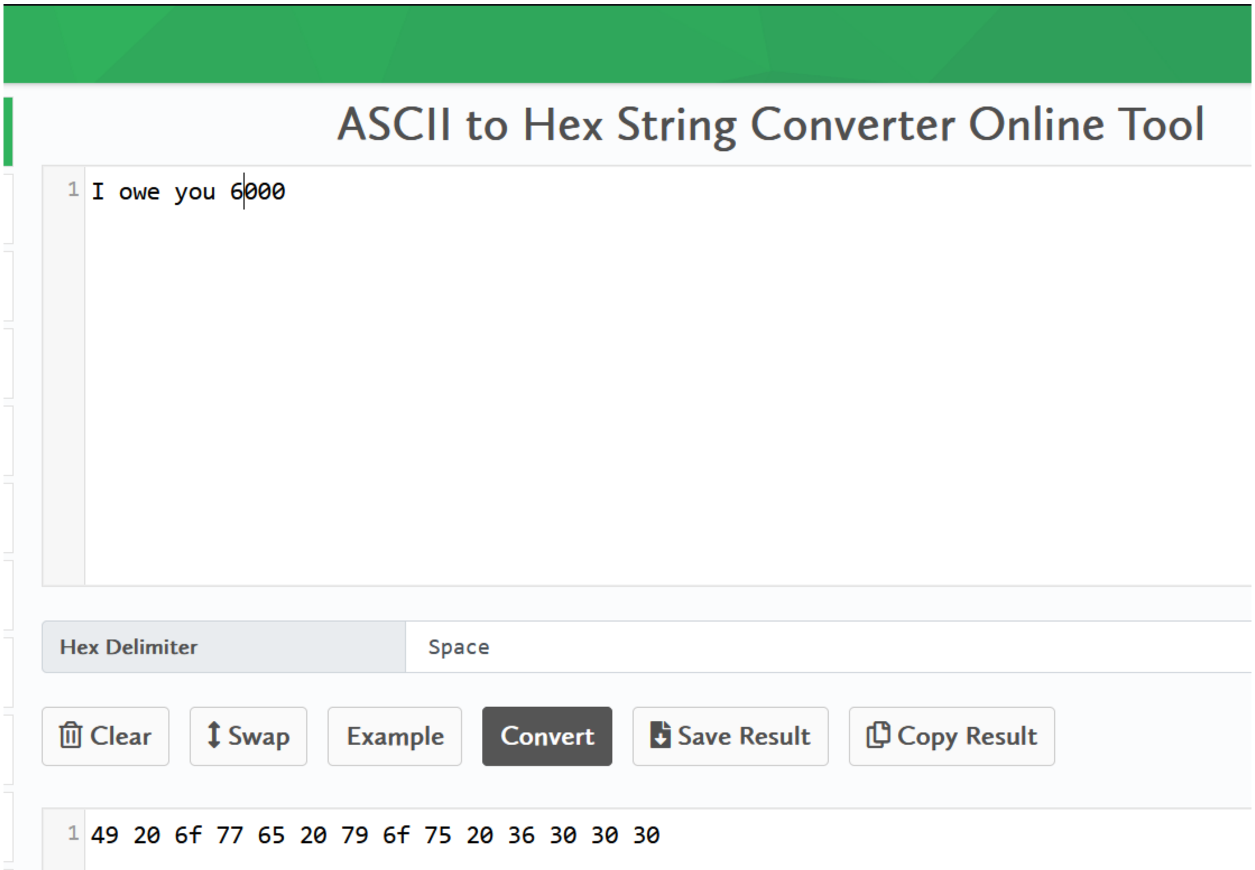
7. Please provide screenshots of generating a signature for the following message: I owe you $2000.

    The Hex for the message I owe you $2000 is the following. Be aware that the hexadecimal numbers were calculated without the dot at the end of the phrase.



8. Please provide screenshots of generating a signature for the following message: I owe you $6000

The Hexadecimal for the I owe you $6000 is the following. The Hex for the message is the following. Be aware that the hexadecimal numbers were calculated without the dot at the end of the phrase.



9. Compare both signatures above and describe what you observed.

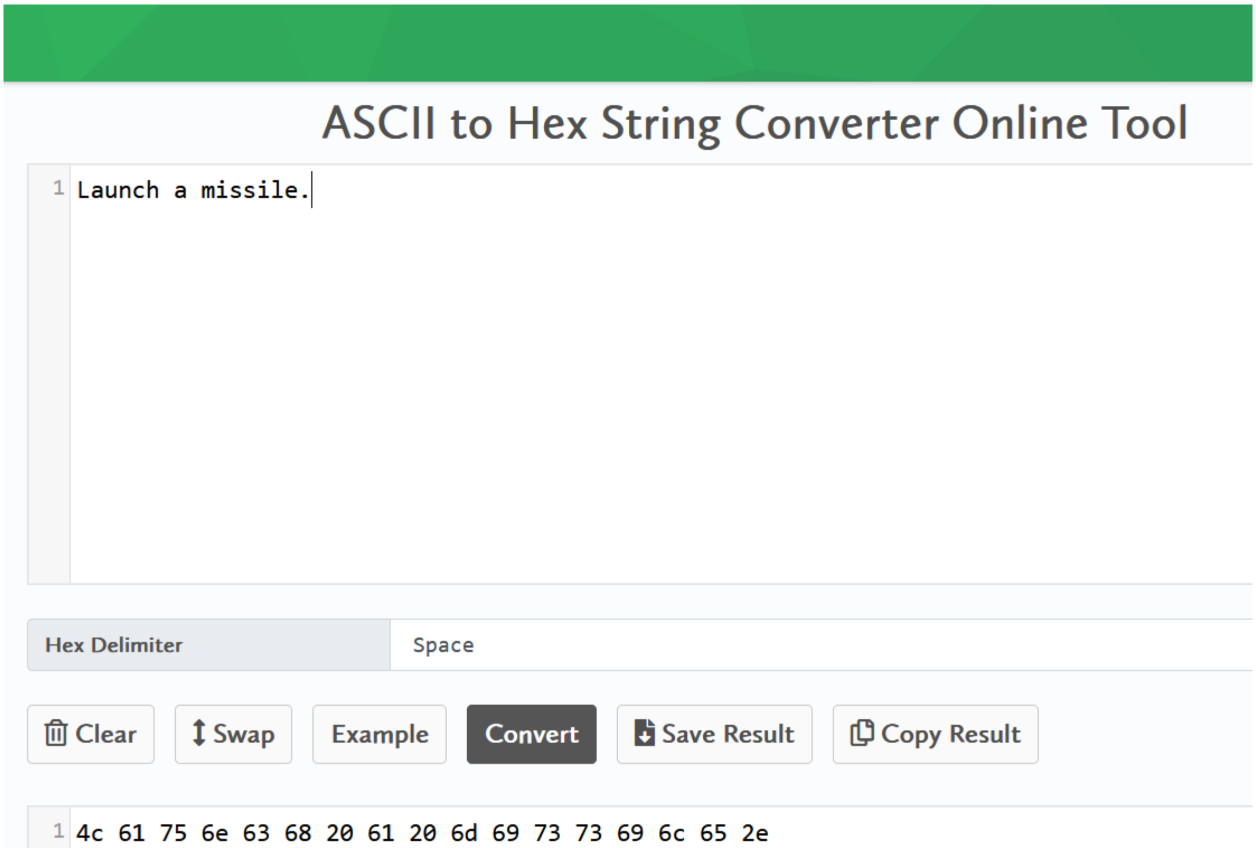
```
[04/21/22]seed@VM:~/RSA$ task4
Signature of M1: 2D5F9D8573CC275B510F4DD485511652F3718B6EDBACEEFC93C878CB6B108AF
8
Signature of M2: 18AA08A89FEFA12C53E178B7E1FCADF6E5405451D3E06ED16B799EE9B33C542
9
[04/21/22]seed@VM:~/RSA$
```

What we can see is that a small change over the message generates a huge change in the signature as-is shown in the last image.

- **Task 5 Verifying a Signature (JJ):**

10. Provide screenshots of how you verify whether the signature is Alice’s or not.

The following screenshot is the hexadecimal of the “Launch a missile.”. It includes the dot at the end.



The following is the change in the code.

```
BIGNUM *C = BN_new();
BIGNUM *S = BN_new();

// assign values
BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");
BN_dec2bn(&e, "65537");
BN_hex2bn(&M, "4c61756e63682061206d697373696c652e"); //hex encode for " Launch a missile."
BN_hex2bn(&S, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");

```

After compiling and executing the program task5 I get this notification.

```
[04/21/22]seed@VM:~/RSA$ gcc -o task5 task5.c -lcrypto
[04/21/22]seed@VM:~/RSA$ task5
Valid Signature!
[04/21/22]seed@VM:~/RSA$
```

11. Corrupt the signature above and (change the last byte of signature from 2F into 3F) repeat this task and describe what is happening. Provide your screenshots after changes.

First of all, the change is done at task5.c file as requested. 3F was written down instead of 2F as shown in the next file.

```
BIGNUM *C = BN_new();
BIGNUM *S = BN_new();

// assign values
BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");
BN_dec2bn(&e, "65537");
BN_hex2bn(&M, "4c61756e63682061206d697373696c652e"); //hex encode for " Launch a missile."
BN_hex2bn(&S, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F");
```

After the compilation and execution of the task5. The following image shows the result.

```
[04/21/22]seed@VM:~/RSA$ gcc -o task5 task5.c -lcrypto
[04/21/22]seed@VM:~/RSA$ task5
Verification fails!
[04/21/22]seed@VM:~/RSA$
```

Small Changes over the signature, may understand that the message was corrupted and the information was compromised.

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

12. What does the X.509 certificate contain?

The X.509 is a specification that defines the format of public-key certificates. The X.509 certificate contains a public key and some identities name. More specifically, it contains the X.509 version, a unique serial number identifier provided by the CA, algorithm information used in signing certificate, issuer name, date of valid certification, subject name and subject public key information.

13. What are the subject field and issuer field of the certificate? Specify the relations.

The subject field of the certificate means the target of the public key certification, and entities connect to this field when the entities need to be verified. Additionally, the issuer field of the certificate means that the stored information of the certificate authority signs the certification with a private key, and CA verifies and issues the certificate.

The relationship between subject and issuer is that the former is about the entity’s information, it could be a person or a firm. The latter one is about the verification of digital signatures.

14. What does it mean if you only get one certificate back using the command *openssl s_client -connect www.example.org:443 –showcerts*?

Basically, if we get two certificates after executing the above command, one is the subject field of certification, and the other is the issuer field of certification.

However, if we get only one certificate from the result of the command, it means that this certification is signed by the root certificate authority (CA).

15. What *asn1parse* in the OpenSSL command do?

According to step4, *asnlparse*, a parsing and Openssl command, is utilized to extract data from ASN.1 encoded format of X.509 certification. And, the input format of *asnlparse* needs to be pem format or binary format.

16. What is the Openssl command to verify the certificate for us?

Usually, considering the root certification, intimidate certification and server certification and all of the certifications are stored in PEM format.

This command can use to verify the certification.

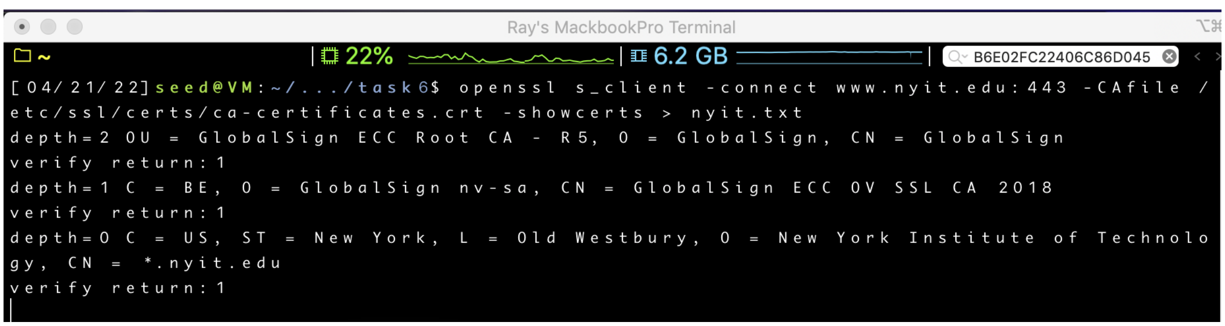
"*openssl verify -CAfile root.pem -untrusted intimidate.pem server.pem"*

In our case, we store issuers and intimidate certificates in one C1 PEM file, and the server certificate in another C0 PEM file. So, we can use this command to verify, "*openssl verify -untrusted C1 PEM.pem C0 PEM.pem*"

17. Provide screenshots of step1 to step 4 of this task6

Take *www.nyit.edu* for example, download the certification from *www.nyit.edu.*

Step1: *openssl s_client -connect www.nyit.edu:443 -CAfile /etc/ssl/certs/ca-certificates.crt -showcerts > nyit.txt*



The issuer certification is below and stored in c1.pem.


```
-----BEGIN CERTIFICATE-----
MIIDAjCCAomgAwIBAgINAe5fIpVCSQX5AZGo3DAKBggqhkjOPQQDAzBQMSQwIgYD
VQQLExtHbG9iYWxTaWduIEVDQyBSb290IENBIC0gUjUxEzARBgNVBAoTCkdsb2Jh
bFNpZ24xEzARBgNVBAMTCkdsb2JhbFNpZ24wHhcNMTgxMTIxMDAwMDAwWhcNMjgx
MTIxMDAwMDAwWjBQMQswCQYDVQQGEwJCRTEZMBcGA1UEChMQR2xvYmFsU2lnbiBu
di1zYTEmMCQGA1UEAxMdR2xvYmFsU2lnbiBFQ0MgT1YgU1NMIENBIDIwMTgwdjAQ
BgcqhkjOPQIBBgUrgQQAIgNiAATDoRGNZSPhluG7q6bQA11PTeOZD/xx44QlFam1
BM4eLeN+wfgwalsbkjzARCM9si/fnQeKNtKAlgNmNOHTmV3VfwGbocj6+22HVWZu
VeX/VeIGoWh1u7Lja/NDE7RsXaCjggEpMIIBJTAOBgNVHQ8BAf8EBAMCAYYwEgYD
VR0TAQH/BAgwBgEB/wIBADAdBgNVHQ4EFgQUWHuOdSr+YYCqkEABrtboB0ZuP0gw
HwYDVR0jBBgwFoAUPeYpSJvqB8ohREom3m7e0oPQn1kwPgYIKwYBBQUHAQEEMjAw
MC4GCCsGAQUFBzABhiJodHRwOi8vb2NzcDIuZ2xvYmFsc2lnbi5jb20vcm9vdHI1
MDYGA1UdHwQvMC0wK6ApoCeGJWh0dHA6Ly9jcmwuZ2xvYmFsc2lnbi5jb20vcm9v
dC1yNS5jcmwwRwYDVR0gBEAwPjA8BgRVHSAAMDQwMgYIKwYBBQUHAgEWJmh0dHBz
Oi8vd3d3Lmdsb2JhbHNpZ24uY29tL3JlcG9zaXRvcnkvMAoGCCqGSM49BAMDA2cA
MGQCMC4lzZGQw5mpNZBmztq8huxKf9/tRUJ5yLI4q6YU+i2fjF2FRBNA64EBmljA
7dkSOwIwL9qYB0APhsLmV0LhknrzHZVvtqzg7NQaIV18BEIDZQgK3gjxYzADjHSH
5uk4mCdW
-----END CERTIFICATE-----
```


The server certification is below and stored in c0.pem.


```
-----BEGIN CERTIFICATE-----
MIIFqTCCBS+gAwIBAgIMULHXkWWYtiBlDrHFMAoGCCqGSM49BAMDMFAxCzAJBgNV
BAYTAkJFMRkwFwYDVQQKExBHbG9iYWxTaWduIG52LXNhMSYwJAYDVQQDEx1HbG9i
YWxTaWduIEVDQyBPViBTU0wgQ0EgMjAxODAeFw0yMDA2MTExODAxNTVaFw0yMjA5
MDQyMzU5NThaMHcxCzAJBgNVBAYTAlVTMREwDwYDVQQIEwhOZXcgWW9yazEVMBMG
A1UEBxMMT2xkIFdlc3RidXJ5MSkwJwYDVQQKEyBOZXcgWW9yayBJbnN0aXR1dGUg
b2YgVGVjaG5vbG9neTETMBEGA1UEAwwKKi5ueWl0LmVkdTBZMBMGByqGSM49AgEG
CCqGSM49AwEHA0IABOmSDKv6+3gzS9uBG8k4Rx5Bp3LmQg5Yb9Y4fSKuBy+GoZbC
Hckj0nI6q/054twriT3mn3ANwEcMFDbT1rUjIKKjggPGMIIDwjAOBgNVHQ8BAf8E
BAMCA4gwgY4GCCsGAQUFBwEBBIGBMH8wRAYIKwYBBQUHMAKGOGh0dHA6Ly9zZWN1
cmUuZ2xvYmFsc2lnbi5jb20vY2FjZXJ0L2dzZWNjb3Zzc2xjYTIwMTguY3J0MDcG
CCsGAQUFBzABhitodHRwOi8vb2NzcC5nbG9iYWxzaWduLmNvbS9nc2VjY292c3Ns
Y2EyMDE4MFYGA1UdIARPME0wQQYJKwYBBAGgMgEUMDQwMgYIKwYBBQUHAgEWJmh0
dHBzOi8vd3d3Lmdsb2JhbHNpZ24uY29tL3JlcG9zaXRvcnkvMAgGBmeBDAECAjAJ
BgNVHRMEAjAAMD8GA1UdHwQ4MDYwNKAyoDCGLmh0dHA6Ly9jcmwuZ2xvYmFsc2ln
bi5jb20vZ3NlY2NvdnNzbGNhMjAxOC5jcmwwHwYDVR0RBBgwFoIKKi5ueWl0LmVk
dYIIbnlpdC5lZHUwHQYDVR0lBBYwFAYIKwYBBQUHAwEGCCsGAQUFBwMCMB8GA1Ud
IwQYMBaAFFh7jnUq/mGAqpBAAa7W6AdGbj9IMB0GA1UdDgQWBBSEHJTAIKgF0lwj
uTgciDQwCj373DCCAfkGCisGAQQB1nkCBAIEggHpBIIB5QHjAHcAb1N2rDHwMRnY
mQCkURX/dxUcEdkCwQApBo2yCJo32RMAAAFypIu1EgAABAMASDBGAiEA5e3QPzqI
J8/xkcnGPU+qJ74MfviQlKoIssddl0UbnJUCIQDNJ0b9JFElDqhK0tVI5+Nd/23b
fYh4CuUAXV/0kr4yYAB3ACJFRQdZVSRWlj+hL/H3bYbgIyZjrcBLf13Gg1xu4g8C
AAABcqSLtNIAAAQDAEgwRgIhAMvqOfDPRuWUY8/dSbbR4hwnJIbWJiqRfRKTUEcU
c6ArAiEA1b6dv49M8lDwx5uNLLhrZQ9dC/5UGa3bD6h3I2NGdioAdgApeb7wnjk5
IfBWc59jpXflvld9nGAK+PlNXSZcJV3HhAAAAXKki7e9AAAEAwBHMEUCID5ire9b
E7fLeSGyHBw265R2gd9Ub/XvenYaBlSJCoSIAiEA3Dlq4I2dJVNvXyZbzBbUHLzk
7qyF7oBw6Qfa7kzS8fkAdwBVgdTCFpA2AUrqC5tXPFPwwOQ4eHAlCBcvo6odBxPT
DAAAAXKki7TrAAAEAwBIMEYCIQCXnCeHYAx3LdHXMnY5EodPZv4LYjy5zhVW+WHx
5VxmXgIhALFA21G/3D8wzN3EOo/4aG5FftqJmoeyVGjqNFDe+nL7MAoGCCqGSM49
BAMDA2gAMGUCMBrrU8T9Ku4VzAToCJSx9ArapeZC+Mw19iOTvXeun8hleOA2p5IW
dxY+p12R5GjiMAIxAJx9e09givBQ4xz9c/2B8/3E08A9Adf2aYQvKHwC8Q40Dcg1
oK8Qc8T9LDMMIqZ4+w==
-----END CERTIFICATE-----
```

Step2:

Extract the public key from the issuer's certification and print the exponent; despite downloading a certification from different websites, the result still shows the wrong algorithm type and no exponent.

openssl x509 -in c1.pem -noout -modulus

openssl x509 -in c1.pem -text -noout | grep Exponent

```
[04/21/22]seed@VM:~/.../task6$ openssl x509 -in c1.pem -noout -modulus
Modulus=Wrong Algorithm type
[04/21/22]seed@VM:~/.../task6$ openssl x509 -in c1.pem -text -noout | grep Exponent
[04/21/22]seed@VM:~/.../task6$
```

Step3: Retrieve the server's signature and remove the space and colons.

openssl x509 -in c0.pem -text -noout

```
[04/21/22]seed@VM:~/.../task6$ cat signature | tr -d '[:space:]:'
306502301aeb53c4fd2aee15cc04e80894b1f40adaa5e642f8cc35f62393bd77ae9fc8657b4036a792167
7163ea75d91e468e230023100 9c7d7b4f608af050e31cfd73fd81f3fdc4d3c03d01d7f669842f287c02f1
0e0e0dc835a0af1073c4fd2c330c22a678fb[04/21/22]seed@VM:~/.../task6$
```

Step4: Using *asn1parse* command to extract the data from c0.perm the server's certification, and the result is the server's body certification.

As seen in the screenshots below, the field starts from offset 4, and the hash value is generated from offset 842 to 1334.

Additionally, the beginning of the signature block is from the offset 1335, which is used as "ecdsa-with-sha384."

```
[04/21/22]seed@VM:~/.../task6$ openssl asn1parse -i -in c0.pem
    0:d=0  hl=4 l=1449 cons: SEQUENCE
    4:d=1  hl=4 l=1327 cons:  SEQUENCE
    8:d=2  hl=2 l=   3 cons:   cont [ 0 ]
   10:d=3  hl=2 l=   1 prim:    INTEGER           :02
   13:d=2  hl=2 l=  12 prim:   INTEGER           :50B1D7916598B620650EB1C5
   27:d=2  hl=2 l=  10 cons:   SEQUENCE
   29:d=3  hl=2 l=   8 prim:    OBJECT            :ecdsa-with-SHA384
   39:d=2  hl=2 l=  80 cons:   SEQUENCE
   41:d=3  hl=2 l=  11 cons:    SET
   43:d=4  hl=2 l=   9 cons:     SEQUENCE
   45:d=5  hl=2 l=   3 prim:      OBJECT            :countryName
   50:d=5  hl=2 l=   2 prim:      PRINTABLESTRING   :BE
   54:d=3  hl=2 l=  25 cons:    SET
   56:d=4  hl=2 l=  23 cons:     SEQUENCE
   58:d=5  hl=2 l=   3 prim:      OBJECT            :organizationName
   63:d=5  hl=2 l=  16 prim:      PRINTABLESTRING   :GlobalSign nv-sa
   81:d=3  hl=2 l=  38 cons:    SET
   83:d=4  hl=2 l=  36 cons:     SEQUENCE
   85:d=5  hl=2 l=   3 prim:      OBJECT            :commonName
   90:d=5  hl=2 l=  29 prim:      PRINTABLESTRING   :GlobalSign ECC OV SSL CA 2018
  121:d=2  hl=2 l=  30 cons:   SEQUENCE
  123:d=3  hl=2 l=  13 prim:    UTCTIME           :200611180155Z
  138:d=3  hl=2 l=  13 prim:    UTCTIME           :220904235958Z
  153:d=2  hl=2 l= 119 cons:   SEQUENCE
  155:d=3  hl=2 l=  11 cons:    SET
  157:d=4  hl=2 l=   9 cons:     SEQUENCE
  159:d=5  hl=2 l=   3 prim:      OBJECT            :countryName
  164:d=5  hl=2 l=   2 prim:      PRINTABLESTRING   :US
  168:d=3  hl=2 l=  17 cons:    SET
  170:d=4  hl=2 l=  15 cons:     SEQUENCE
  172:d=5  hl=2 l=   3 prim:      OBJECT            :stateOrProvinceName
  177:d=5  hl=2 l=   8 prim:      PRINTABLESTRING   :New York
  187:d=3  hl=2 l=  21 cons:    SET
  189:d=4  hl=2 l=  19 cons:     SEQUENCE
  191:d=5  hl=2 l=   3 prim:      OBJECT            :localityName
  196:d=5  hl=2 l=  12 prim:      PRINTABLESTRING   :Old Westbury
  210:d=3  hl=2 l=  41 cons:    SET
  212:d=4  hl=2 l=  39 cons:     SEQUENCE
  214:d=5  hl=2 l=   3 prim:      OBJECT            :organizationName
  219:d=5  hl=2 l=  32 prim:      PRINTABLESTRING   :New York Institute of Technology
```

```
  842:d=5  hl=4 l= 489 prim:      OCTET STRING      [HEX DUMP]:048201E501E30077006F53
76AC31F03119D89900A45115FF77151C11D902C10029068DB2089A37D91300000172A48BB512000004030
04830460221000E5EDD03F3A8827CFF191C9C63D4FAA27BE0C7EF89094AA08B2C75D97451B9C9502210OCD
2746FD2451250EA84AD2D548E7E35DFF6DDB7D88780AE5005D5FCE92BE3260007700224545075955245 69
63FA12FF1F76D86E0232663ADC04B7F5DC6835C6EE20F0200000172A48BB4D2000004030048304602210 0
CBEAD1F0CF46E59463CFDD49B6D1E21C272486D6262A917D12935047147 3A02B022100D5BE9DBF8F4CF25
0F0C79B8D2CB86B650F5D0BFE5419ADDB0FA877236346762A0076002979BEF09E393921F056739F63A577
E5BE577D9C600AF8F94D5D265C255DC78400000172A48BB7BD000004030047304502203E62ADEF5B13B7C
B7921B21C1C36EB947681DF546FF5EF7A761A0654890A8488022100DC396AE08D9D25536F5F265BCC16D4
1CBCE4EEAC85EE8070E907DAEE4CD2F1F9007700558 1D4C2169036014AEA0B9B573C53F0C0E4387870250
8172FA3AA1D0713D30C00000172A48BB4EB0000040300483046022100979C2787600C772DD1D732763912
874F66FE0B623CB9CE1556F961F1E55C665E022100B140DB51BFDC3F30CCDDC43A8FF8686E457EDA899A8
7B25468EA3450DEFA72FB
 1335:d=1  hl=2 l=  10 cons:  SEQUENCE
 1337:d=2  hl=2 l=   8 prim:   OBJECT            :ecdsa-with-SHA384
 1347:d=1  hl=2 l= 104 prim:  BIT STRING
```

The screenshot below indicates using *strparse* command from offset 4, which is precisely the body of the server's certificate and then calculating the hash value to verify the certification.

```
[04/21/22]seed@VM:~/.../task6$ openssl asn1parse -i -in c0.pem -strparse 4 -out c0_bo
dy.bin -noout
[04/21/22]seed@VM:~/.../task6$ sha256sum c0_body.bin
574d0b005382c8806dc489de508560abee8bd0146936c252ffe349c8ce60a2f6  c0_body.bin
[04/21/22]seed@VM:~/.../task6$
```

References:

https://articles.assembla.com/en/articles/1623119-certificate-verification-error-20-unable-to-get-local-user-certificate