RSA Lab

Task 1: Deriving the Private Key

Using these values:

p = F7E75FDC469067FFDC4E847C51F452DF

q = E85CED54AF57E53E092113E62F436F4F

e = 0D88C3

As well as bn_sample.c with some modifications, we can derive the private key to:

```
[09/14/25]seed@VM:~/.../Labsetup$ make
gcc bn_sample.c -o bn_sample -lcrypto
[09/14/25]seed@VM:~/.../Labsetup$ ls
bn_sample  bn_sample.c  Makefile
[09/14/25]seed@VM:~/.../Labsetup$ bn_sample
n =  E103ABD94892E3E74AFD724BF28E78366D9676BCCC70118BD0AA1968DBB14
3D1
d =  3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496
AEB
```

Which means:

n = E103ABD94892E3E74AFD724BF28E78366D9676BCCC70118BD0AA1968DBB143D1

d = 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB

e = 0D88C3

Task 2: Encrypting a Message

First convert the message to hex string:

```
print("A top secret!".encode("utf-8").hex())
```

```
[09/14/25]seed@VM:~/.../Labsetup$ python3 task2.py
4120746f70207365637265742l421
```

Then convert the hex string to a BIGNUM and encrypt it using the public key (e, n):

```
[09/14/25]seed@VM:~/.../Labsetup$ task2_ascii
Original ASCII message: A top secret!
Message in hex: 4120746F702073656372657421
Message as BIGNUM =  4120746F702073656372657421
Encrypted ciphertext =  6FB078DA550B2650832661E14F4F8D2CFAEF475A0D
F3A75CACDC5DE5CFC5FADC
```

Task 3: Decrypting a Message

Using the private key, (d, n), we decrypt and show the message as a BIGNUM

```
[09/14/25]seed@VM:~/.../Labsetup$ task2_ascii
Decrypted (hex) =  4120746F702073656372657421
```

Then convert the BIGNUM back into ASCII:

```
print(bytes.fromhex("4120746f70207365637265742e").decode("utf-8"))
```

```
[09/14/25]seed@VM:~/.../Labsetup$ python3 task2.py
A top secret!
```

Task 4: Signing a Message

First we convert the message into a hex string

```
[09/14/25]seed@VM:~/.../Labsetup$ python3 task2.py
49206f776520796f752024323030302e
```

After modifying the $2000 to $3000, we sign the messages:

```
[09/14/25]seed@VM:~/.../Labsetup$ nano task4_sign.c
[09/14/25]seed@VM:~/.../Labsetup$ gcc -o task4_sign task4_sign.c -l
crypto
[09/14/25]seed@VM:~/.../Labsetup$ task4_sign
Message M1 (hex) = 49206F776520796F752024323030302E
Signature S1 = 55A4E7F17F04CCFE2766E1EB32ADDBA890BBE92A6FBE2D785ED6
E73CCB35E4CB

Message M2 (hex) = 49206F776520796F752024333030302E
Signature S2 = BCC20FB7568E5D48E434C387C06A6025E90D29D848AF9C3EBAC0
135D99305822
```

From this, we can see that the message hexes had a singular bit changed from 3230 to 3330 and as a result, the two signatures ended up being completely different.

Task 5: Verifying a Signature

```
[09/14/25]seed@VM:~/.../Labsetup$ task5_verify

=== TASK 5: Verify signature (original) ===
Expected message (ASCII): Launch a missile.

Recovered M' (hex): 4C61756E63682061206D697373696C652E
Recovered M' (ASCII): Launch a missile.
Verification result: OK — recovered message matches expected messag
e.

=== TASK 5: Verify signature (corrupted last byte) ===
Corrupted signature hex: 643D6F34902D9C7EC90CB0B2BCA36C47FA37165C00
05CAB026C0542CBDB6803F

Recovered from corrupted S (hex): 91471927C80DF1E42C154FB4638CE8BC7
26D3D66C83A4EB6B7BE0203B41AC294
��,O�c��rm=f�:N�����pted S (ASCII): �G'�
```

From this we can see that after corrupting the last byte of the signature, it resulted in a different hex string full of garbage / does not match. As a result, the ASCII string also becomes gibberish.

Task 6: Manually Verifying an X.509 Certificate

After downloading google's certificates and extracting the public key from the issuer's certificate, we extract the signature from the server's certificate and extract the body of the server's certificate:

```
[09/15/25]seed@VM:~/.../Labsetup$ python3 task6.py
TBSCertificate length: 3353 bytes
Signature length: 256 bytes
n = B89293A7E5F1C4D9D8B3F04C9D12F5A0B7C1E5A9D6C8F2B3A1E9D7F6C4B2A1F
0B7D3E1C5A9B8C7D6E5F4A3B2C1D0E9F8A7B6C5D4E3F2A1B0C9D8E7F6A5B4C3D2
e = 010001


Recovered block (hex, 512 chars):
0001FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00
3031300D060960864801650304020105000420BF24A58F0C45B8C1235E3C0A3567D
74D8076F1A56D7A456C39F1F142A11A6CFB


Verification successful! Digest matches.
```

This shows that we were able to successfully extract the body of the server's certificate.