Mark Gameng

CS 458 – Dong Jin

RSA

3.1 – Task 1: Deriving the private key

```
//initialize p, q ,e
BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
BN_hex2bn(&e, "0D88C3");
BN_dec2bn(&one, "1");

//n = p * q
BN_mul(n, p, q, ctx);
printBN("n = p * q = ", n);

//phi = (p - 1) * (q - 1)
BN_sub(p, p, one);
BN_sub(q, q, one);
BN_mul(phi, p, q, ctx);
printBN("Phi = (p - 1) * (q - 1) = ", phi);

//d
BN_mod_inverse(d, e, phi, ctx);
printBN("d = ", d);
```

Given **p**, **q**, and **e**, I made a similar program in the lab description and found **n** which is **p** * **q**. I then calculated the private key **d** by first calculating **phi**.

```
[10/13/20]seed@VM:~/.../RSA$ task1
n = p * q =  E103ABD94892E3E74AFD724BF28E78366D9676BCCC70118BD0AA1968DBB143D1
Phi = (p - 1) * (q - 1) =  E103ABD94892E3E74AFD724BF28E78348D52298BD687C44DEB3A8
1065A7981A4
d =  3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB
```

It was very easy to create the private key. I wonder if there is an easier process than this but also as secure or even better?

3.2 – Task 2: Encrypting a message

We want to encrypt “A top secret!” but first it needs to be turned in to a hexadecimal.

```
[10/13/20]seed@VM:~/.../RSA$ python -c 'print("A top secret!".encode("hex"))'
4120746f702073656372657421
```

Public keys are given so we can encrypt it. Encrypted = message ^ e mod n. The private key was also given so we can double check by decrypting and it should give the hex of the message.

```
//initialize var
BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
BN_hex2bn(&e, "010001");
BN_hex2bn(&m, "4120746f702073656372657421");
BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

//m^e mod n -- encrypting
BN_mod_exp(encrypted, m, e, n, ctx);
printBN("Encrpyted message = ", encrypted);

//encryption^d mod n -- decrypting
BN_mod_exp(decrypted, encrypted, d, n, ctx);
printBN("Decrypted message = ", decrypted);
```

```
[10/13/20]seed@VM:~/.../RSA$ task2
Encrpyted message =  6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC
5FADC
Decrypted message =  4120746F702073656372657421
```

Here we see that “A top secret!” was encrypted and decrypting using the private key **d**, we get the same hex of “A top secret!”. Thus, we have successfully encrypted a message and decrypted it using a private key.

3.3 – Task 3: Decrypting a message

We are given ciphertext, **C,** to decode and using the same values, we can decrypt it.

```
//initialize var
BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
BN_hex2bn(&encrypted, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");
/*ciphertext C given*/

//encryption^d mod n -- decrypting
BN_mod_exp(decrypted, encrypted, d, n, ctx);
printBN("Decrypted message = ", decrypted);
```

```
[10/13/20]seed@VM:~/.../RSA$ task3
Decrypted message =  50617373776F72642069732064656573
```

```
[10/13/20]seed@VM:~/.../RSA$ python -c 'print("50617373776F72642069732064656573"
.decode("hex"))'
Password is dees
```

Getting the decrypted hex and then converting it to ASCII, we get that the decrypted ciphertext **C**, is “Password is dees”.

3.4 – Task 4: Signing a message

```
[10/13/20]seed@VM:~/.../RSA$ python -c 'print("I owe you $2000.".encode("hex"))'

49206f776520796f75202432303030302e
[10/13/20]seed@VM:~/.../RSA$ python -c 'print("I owe you $3000.".encode("hex"))'

49206f776520796f75202433303030302e
```

Image above shows two messages with its hex values. Those two will be used to compare signatures. Signature is just RSA function with the private key.

```
//initialize var
BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
BN_hex2bn(&e, "010001");
BN_hex2bn(&m1, "49206f776520796f75202432303030302e"); /* I owe you $2000 */
BN_hex2bn(&m2, "49206f776520796f75202433303030302e"); /* I owe you $3000 */
BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

//m^e mod n -- signature -- RSA func with private key
BN_mod_exp(m1s, m1, d, n, ctx);
printBN("M1 Signature = ", m1s);
BN_mod_exp(m2s, m2, d, n, ctx);
printBN("M2 Signature = ", m2s);
```

```
M1 Signature =  55A4E7F17F04CCFE2766E1EB32ADDBA890BBE92A6FBE2D785ED6E73CCB35E4CB
M2 Signature =  BCC20FB7568E5D48E434C387C06A6025E90D29D848AF9C3EBAC0135D99305822
```

Here we see that even though we only made a slight change to the message **M**, we get very different signatures. I thought it would be quite similar because the hex of the messages are similar except for one byte. That's shows how good the function is and use of exponent and modulus.

3.5 Task 5: Verifying a signature

Given a message with signature and public key, we can verify the signature is legit.

```
//initialize var
BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");
BN_hex2bn(&m ,"4c61756e63682061206d697373696c652e");//M = Launch a missile.
BN_hex2bn(&s, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");
BN_hex2bn(&e, "010001");

BN_mod_exp(ver, s, e, n, ctx);
printBN("Verifying = ", ver);
```

```
[10/13/20]seed@VM:~/.../RSA$ task5
Verifying =  4C61756E63682061206D697373696C652E
[10/13/20]seed@VM:~/.../RSA$ python -c 'print("4C61756E63682061206D697373696C652E".decode("hex"))'
Launch a missile.
```

To verify the signature, we do signature^e mod n, and we get the message. Cross checking the message, we see that the messages are equal, so the signature is indeed Alices.

Changing the last byte of the signature, this is what happens:

```
[10/13/20]seed@VM:~/.../RSA$ task5
Verifying =  91471927C80DF1E42C154FB4638CE8BC726D3D66C83A4EB6B7BE0203B41AC294
[10/13/20]seed@VM:~/.../RSA$ python -c 'print("91471927C80DF1E42C154FB4638CE8BC726D3D66C83A4EB6B7BE0203B41AC294".decode("hex"))'
��,��c���rm=f�:N������
```

We can clearly see that the signature is not Alice's or is corrupted because the hex is different and decoding it got something random. This is surprising because we only changed the last byte of the signature, and it completely changed it. I would have thought it would only made a small difference.

3.6 – Task 6

Step 1:

I used www.google.com and got the certificate and copied it to c0.pem and c1.pem.


```
-----BEGIN CERTIFICATE-----
MIIFkTCCBHmgAwIBAgIQB5q2sT+2NbIIAAAAAFst1zANBgkqhkiG9w0BAQsFADBC
MQswCQYDVQQGEwJVUzEeMBwGA1UEChMVR29vZ2xlIFRydXN0IFNlcnZpY2VzMRMw
EQYDVQQDEwpHVFMgQ0EgMU8xMB4XDTIwMDkyMjE1MTYxOVoXDTIwMTIxNTE1MTYx
OVowaDELMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNhbGlmb3JuaWExFjAUBgNVBAcT
DU1vdW50YWluIFZpZXcxEzARBgNVBAoTCkdvb2dsZSBMTEMxFzAVBgNVBAMTDnd3
dy5nb29nbGUuY29tMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAnG1D
Z2Tl8l5ruTFxLz5PmUKZjYFEihvXKsmiu2d0NTKv8RgWvMItgvP/1IfIz+O76jn3
meC71lP10LST+wI+vFPE9vEL/Zie5veUijpXE0bUApeud+Rnlsw4UJ1x50PCBdSN
r41sNVCAAU98ibSGlc5n2Qr1/YYhs2qnqKKG8i8zNMbjDYyzOWyjfmRVi8W7eU2Z
FpJVwc3D6WEACqQrLATPcFNK0Jb3f5j7eohea9eMCslB13e43iIbA2zHb8owE2Mu
F7GKhYiTr8OEFy4aq7meEeIFHDB/O5eO+6nSxV0p3dBsdxXu/BlEVRuwnigCtMAn
LVS5OEhwBuz1Wl+VWwIDAQABo4ICWzCCAlcwDgYDVR0PAQH/BAQDAgWgMBMGA1Ud
JQQMMAoGCCsGAQUFBwMBMAwGA1UdEwEB/wQCMAAwHQYDVR0OBBYEFDDRC4ywBM9P
OAk0QWnLJlg3HjmJMB8GA1UdIwQYMBaAFJjR+G4Q68+b7GCfGJAboOt9Cf0rMGgG
CCsGAQUFBwEBBFwwWjArBggrBgEFBQcwAYYfaHR0cDovL29jc3AucGtpLmdvb2cv
Z3RzMW8xY29yZTArBggrBgEFBQcwAoYfaHR0cDovL3BraS5nb29nL2dzcjIvR1RT
MU8xLmNydDAZBgNVHREEEjAQgg53d3cuZ29vZ2xlLmNvbTAhBgNVHSAEGjAYMAgG
BmeBDAECAjAMBgorBgEEAdZ5AgUDMDMGA1UdHwQsMCowKKAmoCSGImh0dHA6Ly9j
cmwucGtpLmdvb2cvR1RTMU8xY29yZS5jcmwwggEDBgorBgEEAdZ5AgQCBIH0BIHx
AO8AdQDwlaRZ8gDRgkAQLS+TiI6tS/4dR+OZ4dA0prCoqo6ycwAAAXS2mgE4AAAE
AwBGMEQCIBA000C/IxSaE2sVhS+dJtnXsh7fSYjeybHOnFOtoRFCAiBnTcymqGeb
lwBe5U3nJyG3tngeH9YCfBdkeShmHdf6DgB2ALIeBcyLos2KIE6HZvkruYolIGdr
2vpw57JJUy3vi5BeAAABdLaaADoAAAQDAEcwRQIgBFJa178fY3/4Pb95N5hh2JfR
3EJ9AUbb0vff31qx/NsCIQDJTBFLs1QdubTe6S+Xc7EuQC9rh3YVLSYc8+dSRZuV
3zANBgkqhkiG9w0BAQsFAAOCAQEAfznjbmvP1GKbyj7RIT5L/x6dkPBCWp6u6toi
1ak4chqHN7mkJkazcb+DGoSAkz7DWfvrVt6Kruh7Vq93Z90g9Nnp5ZiMvkHd5+JM
VqVq3SEK0x+Bd//cW7364zsqnCP97Dg1kvPZz/Rqkq04i9ajSGNxkiMjkkFG4klO
tBuMXOmjoIPwa81iXT1tpV8TqV3uQj0FJ+WZXrYP33HSFGgEXO4VJq6cAh1o5V4z
+3KOF1Si5pVAzIHjEbTB9RP2WK7XQ2VkmyfqcnEJYCEnLDAReQkkGQMmJVO14jHh
SRcKMTc/HXPKihjE7cPmhElEehHzWkJLsYwYZcIGOP+shpz1rw==
-----END CERTIFICATE-----
```

```
-----BEGIN CERTIFICATE-----
MIIESjCCAzKgAwIBAgINAeO0mqGNiqmBJWlQuDANBgkqhkiG9w0BAQsFADBMMSAw
HgYDVQQLExdHbG9iYWxTaWduIFJvb3QgQ0EgLSBSMjETMBEGA1UEChMKR2xvYmFs
U2lnbjETMBEGA1UEAxMKR2xvYmFsU2lnbjAeFw0xNzA2MTUwMDAwNDJaFw0yMTEy
MTUwMDAwNDJaMEIxCzAJBgNVBAYTAlVTMR4wHAYDVQQKExVHb29nbGUgVHJ1c3Qg
U2VydmljZXMxEzARBgNVBAMTCkdUUyBDQSAxTzEwggEiMA0GCSqGSIb3DQEBAQUA
A4IBDwAwggEKAoIBAQDQGM9F1IvN05zkQO9+tN1pIRvJzzyOTHW5DzEZhD2ePCnv
UA0Qk28FgICfKqC9EksC4T2fWBYk/jCfC3R3VZMdS/dN4ZKCEPZRrAzDsiKUDzRr
mBBJ5wudgzndIMYcLe/RGGFl5yODIKgjEv/SJH/UL+dEaltN11BmsK+eQmMF++Ac
xGNhr59qM/9il71I2dN8FGfcddwuaej4bXhp0LcQBbjxMcI7JP0aM3T4I+DsaxmK
FsbjzaTNC9uzpFlgOIg7rR25xoynUxv8vNmkq7zdPGHXkxWY7oG9j+JkRyBABk7X
rJfoucBZEqFJJSPk7XA0LKW0Y3z5oz2D0c1tJKwHAgMBAAGjggEzMIIBLzAOBgNV
HQ8BAf8EBAMCAYYwHQYDVR0lBBYwFAYIKwYBBQUHAwEGCCsGAQUFBwMCMBIGA1Ud
EwEB/wQIMAYBAf8CAQAwHQYDVR0OBBYEFJjR+G4Q68+b7GCfGJAboOt9Cf0rMB8G
A1UdIwQYMBaAFJviB1dnHB7AagbeWbSaLd/cGYYuMDUGCCsGAQUFBwEBBCkwJzAl
BggrBgEFBQcwAYYZaHR0cDovL29jc3AucGtpLmdvb2cvZ3NyMjAyBgNVHR8EKzAp
MCegJaAjhiFodHRwOi8vY3JsLnBraS5nb29nL2dzcjIvZ3NyMi5jcmwwPwYDVR0g
BDgwNjA0BgZngQwBAgIwKjAoBggrBgEFBQcCARYcaHR0cHM6Ly9wa2kuZ29vZy9y
ZXBvc2l0b3J5LzANBgkqhkiG9w0BAQsFAAOCAQEAGoA+Nnn78y6pRjd9XlQWNa7H
TgiZ/r3RNGkmUmYHPQq6Scti9PEajvwRT2iWTHQr02fesqOqBY2ETUwgZQ+lltoN
FvhsO9tvBCOIazpswWC9aJ9xju4tWDQH8NVU6YZZ/XteDSGU9YzJqPjY8q3MDxrz
mqepBCf5o8mw/wJ4a2G6xzUr6Fb6T8McDO22PLRL6u3M4Tzs3A2M1j6bykJYi8wW
IRdAvKLWZu/axBVbzYmqmwkm5zLSDW5nIAJbELCQCZwMH56t2Dvqofxs6BBcCFIZ
USpxu6x6td0V7SvJCCosirSmIatj/9dSSVDQibet8q/7UK4v4ZUN80atnZz1yg==
-----END CERTIFICATE-----
```


Step 2:

Can extract the value of **n** and **e** using the following commands, **openssl x509 -in c1.pem -noout -modulus**, and **openssl x509 -in c1.pem -text -noout**

```
[10/13/20]seed@VM:~/.../RSA$ openssl x509 -in c1.pem -noout -modulus
Modulus=D018CF45D48BCDD39CE440EF7EB4DD69211BC9CF3C8E4C75B90F3119843D9E3C29EF500D
10936F0580809F2AA0BD124B02E13D9F581624FE309F0B747755931D4BF74DE1928210F651AC0CC3
B222940F346B981049E70B9D8339DD20C61C2DEFD1186165E7238320A82312FFD2247FD42FE7446A
5B4DD75066B0AF9E426305FBE01CC46361AF9F6A33FF6297BD48D9D37C1467DC75DC2E69E8F86D78
69D0B71005B8F131C23B24FD1A3374F823E0EC6B198A16C6E3CDA4CD0BDBB3A4596038883BAD1DB9
C68CA7531BFCBCD9A4ABBCDD3C61D7931598EE81BD8FE264472040064ED7AC97E8B9C05912A14925
23E4ED70342CA5B4637CF9A33D83D1CD6D24AC07
```

```
  Modulus:
      00:d0:18:cf:45:d4:8b:cd:d3:9c:e4:40:ef:7e:b4:
      dd:69:21:1b:c9:cf:3c:8e:4c:75:b9:0f:31:19:84:
      3d:9e:3c:29:ef:50:0d:10:93:6f:05:80:80:9f:2a:
      a0:bd:12:4b:02:e1:3d:9f:58:16:24:fe:30:9f:0b:
      74:77:55:93:1d:4b:f7:4d:e1:92:82:10:f6:51:ac:
      0c:c3:b2:22:94:0f:34:6b:98:10:49:e7:0b:9d:83:
      39:dd:20:c6:1c:2d:ef:d1:18:61:65:e7:23:83:20:
      a8:23:12:ff:d2:24:7f:d4:2f:e7:44:6a:5b:4d:d7:
      50:66:b0:af:9e:42:63:05:fb:e0:1c:c4:63:61:af:
      9f:6a:33:ff:62:97:bd:48:d9:d3:7c:14:67:dc:75:
      dc:2e:69:e8:f8:6d:78:69:d0:b7:10:05:b8:f1:31:
      c2:3b:24:fd:1a:33:74:f8:23:e0:ec:6b:19:8a:16:
      c6:e3:cd:a4:cd:0b:db:b3:a4:59:60:38:88:3b:ad:
      1d:b9:c6:8c:a7:53:1b:fc:bc:d9:a4:ab:bc:dd:3c:
      61:d7:93:15:98:ee:81:bd:8f:e2:64:47:20:40:06:
      4e:d7:ac:97:e8:b9:c0:59:12:a1:49:25:23:e4:ed:
      70:34:2c:a5:b4:63:7c:f9:a3:3d:83:d1:cd:6d:24:
      ac:07
  Exponent: 65537 (0x10001)
```

From those, we now have the **n** and **e**.

Step 3:

To get the signature, I did openssl x509 -in c0.pem -text -noout

```
Signature Algorithm: sha256WithRSAEncryption
     7f:39:e3:6e:6b:cf:d4:62:9b:ca:3e:d1:21:3e:4b:ff:1e:9d:
     90:f0:42:5a:9e:ae:ea:da:22:d5:a9:38:72:1a:87:37:b9:a4:
     26:46:b3:71:bf:83:1a:84:80:93:3e:c3:59:fb:eb:56:de:8a:
     ae:e8:7b:56:af:77:67:dd:20:f4:d9:e9:e5:98:8c:be:41:dd:
     e7:e2:4c:56:a5:6a:dd:21:0a:d3:1f:81:77:ff:dc:5b:bd:fa:
     e3:3b:2a:9c:23:fd:ec:38:35:92:f3:d9:cf:f4:6a:92:ad:38:
     8b:d6:a3:48:63:71:92:23:23:92:41:46:e2:49:4e:b4:1b:8c:
     5c:e9:a3:a0:83:f0:6b:cd:62:5d:3d:6d:a5:5f:13:a9:5d:ee:
     42:3d:05:27:e5:99:5e:b6:0f:df:71:d2:14:68:04:5c:ee:15:
     26:ae:9c:02:1d:68:e5:5e:33:fb:72:8e:17:54:a2:e6:95:40:
     cc:81:e3:11:b4:c1:f5:13:f6:58:ae:d7:43:65:64:9b:27:ea:
     72:71:09:60:21:27:2c:30:11:79:09:24:19:03:26:25:53:b5:
     e2:31:e1:49:17:0a:31:37:3f:1d:73:ca:8a:18:c4:ed:c3:e6:
     84:49:44:7a:11:f3:5a:42:4b:b1:8c:18:65:c2:06:38:ff:ac:
     86:9c:f5:af
```

Inputting those values in a text file and removing the spaces and colons, we get the signature.

```
[10/13/20]seed@VM:~/.../RSA$ cat sig.txt | tr -d '[:space:]:'
7f39e36e6bcfd4629bca3ed1213e4bff1e9d90f0425a9eaeeada22d5a938721a8737b9a42646b371
bf831a8480933ec359fbeb56de8aaee87b56af7767dd20f4d9e9e5988cbe41dde7e24c56a56add21
0ad31f8177ffdc5bbdfae33b2a9c23fdec383592f3d9cff46a92ad388bd6a348637192232392414б
e2494eb41b8c5ce9a3a083f06bcd625d3d6da55f13a95dee423d0527e5995eb60fdf71d21468045c
ee1526ae9c021d68e55e33fb728e1754a2e69540cc81e311b4c1f513f658aed74365649b27ea7271
096021272c30117909241903262553b5e231e149170a31373f1d73ca8a18c4edc3e68449447a11f3
5a424bb18c1865c20638ffac869cf5af[10/13/20]seed@VM:~/.../RSA$
```

Step 4:

To get the body of the certificate, without the signature block, and put it in a file:

```
[10/13/20]seed@VM:~/.../RSA$ openssl asn1parse -i -in c0.pem -strparse 4 -out c0
_body.bin -noout
```

Calculating the hash of the body of the certificate:

```
[10/13/20]seed@VM:~/.../RSA$ sha256sum c0_body.bin
fe5623d9b8e79ffa12b3b6471ae96c3ebc8ee5b6144305320c8c7de06443269a  c0_body.bin
```

This will be used to verify the certificate, similar to task 5.

Step 5:

Using a similar program in task 5, but using the **n**, **e**, and **signature** above, we can verify the signature:

```
   //initialize var
   BN_hex2bn(&n,
"D018CF45D48BCDD39CE440EF7EB4DD69211BC9CF3C8E4C75B90F3119843D9E3C29EF500D10936F0580809F2AA0I
   BN_hex2bn(&s,
"7f39e36e6bcfd4629bca3ed1213e4bff1e9d90f0425a9eaeeada22d5a938721a8737b9a42646b371bf831a8480
   BN_hex2bn(&e, "010001");

   // decrypting
   BN_mod_exp(body, s, e, n, ctx);
   printBN("Body = ", body);
```

Running the program results in:

```
[10/13/20]seed@VM:~/.../RSA$ task6
Body =  01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFF003031300D060960864801650304020105000420FE5623D9B8E79FFA12B3B6471A
E96C3EBC8EE5B6144305320C8C7DE06443269A
```

Comparing with the hash in step 5:

```
[10/13/20]seed@VM:~/.../RSA$ sha256sum c0_body.bin
fe5623d9b8e79ffa12b3b6471ae96c3ebc8ee5b6144305320c8c7de06443269a  c0_body.bin
```

The hash looks different, but at the end, they are the same. Thus, we have verified that the signature is valid. I wonder why we get all those irrelevant? hex values in front.

Final Thoughts

This was very mind opening and quite surprising at certain points. In task 5, I knew changing just one byte would make a difference, but never expected it to be that big of a difference. It went from being a legible message to being gibberish due to one byte being off in the signature. It just goes to show how secure RSA. RSA is secure and easy to implement, however I wonder in a security perspective, would it better to have a cipher that is as secure but harder to implement?