Lab 1: RSA Public-Key Encryption and Signature Lab

Marlee Bryant

CS 548 Network Security

5 March 2022

Introduction

RSA is a form of asymmetric encryption using two keys, public and private, to encrypt, decrypt, or sign messages sent between two parties. Encryption and signatures are two important aspects of cryptography which help ensure the confidentiality, authenticity, and integrity of transmitted messages. RSA's security stems from the difficulty of factoring an extremely large number. For this reason, RSA's security is directly related to key size, with larger keys providing greater security.

Goals

The goal of this lab is to test knowledge of RSA key generation, encryption, decryption, and signature generation and verification by implementing these algorithms in C. Implementation of the algorithm requires thorough understanding of the computation required for each step. Testing will provide even deeper understanding of the algorithm through observation of how changes to the input can alter the output. Verification of an X.509 certificate from a real web server exemplifies the possible use cases for RSA.

Environment Setup

To ensure proper behavior of the RSA program during testing, the SEED Ubuntu 20.04 Virtual Machine should be used. The provided pre-built VM can be added to Virtual Box, and testing can be performed using the terminal for this machine. While many Integrated Development Environments are available for C programming, the IDE used to create the RSA program for this lab is Atom.

Task 1

For this task, a function was developed which generates an RSA private key using provided p, q, and e values. P and q are large prime numbers and e is an integer such that e < phi(n) – 1 and GCD(e, phi(n)) = 1. Since these prime numbers are unusually large, they cannot be represented by primitive data types, but are instead represented by the BIGNUM type provided by openssl. N is the public key, which can be determined by multiplying p and q. Phi(n) is the product of p-1 and q-1, and must be calculated to determine the private key. The equation for the private key is d = (1 mod phi(n))/e, but this relationship can be reinterpreted as 1 = (d*e mod phi(n)), so d can be calculated using modular inverse. Figure 1 shows the section of the main function related to Task 1, including p, q, and e initialization, calculation of public key n, and the call to the generatePrivKey function. Figure 2 depicts this function definition, including the calculation of phi(n) and the modular inverse calculation. Figure 3 shows the output of the function displaying the calculated private key value.

```c
BN_CTX *ctx = BN_CTX_new();
BIGNUM *p = BN_new();
BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
BIGNUM *q = BN_new();
BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
BIGNUM *e = BN_new();
BN_hex2bn(&e, "0D88C3");

BIGNUM *n = BN_new();
BN_mul(n, p, q, ctx);

BIGNUM *d = BN_new();
d = generatePrivKey(p,q,e,ctx);
char * privKey = BN_bn2hex(d);
printf("Private Key: %s", privKey);
```

Fig 1. Generate private key function call in main

```c
BIGNUM * generatePrivKey(BIGNUM * p, BIGNUM * q, BIGNUM * e, BN_CTX * ctx) {
  BIGNUM * phi = BN_new();
  BIGNUM * pDec = BN_new();
  BIGNUM * qDec = BN_new();
  BIGNUM * dec = BN_new();
  BIGNUM * d = BN_new();

  BN_dec2bn(&dec, "1");
  BN_sub(pDec, p, dec);
  BN_sub(qDec, q, dec);
  BN_mul(phi, pDec, qDec, ctx);

  BN_mod_inverse(d, e, phi, ctx);
  return d;
}
```

Fig 2. Generate private key function definition

```
[03/05/22]seed@VM:~/shared$ gcc rsa.c -lcrypto
[03/05/22]seed@VM:~/shared$ ./a.out
Private Key: 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB
Cipher Text: C33FA48B095C885D96E9603286A1D04BF2E73206DA07A3852E0D3B02A973ED97
Decrypted Message: 50617373776F72642069732064656573
```

Fig 3. Task 1 through 3 output

Task 2

In this task, the RSA encryption algorithm was implemented. Encryption requires the public key pair (n, e) in addition to the message. Modular exponentiation is used to convert the numerical representation of the message to a cipher text. Figure 4 shows the section of the main function dealing with encryption, including initializing n, e, and m values, where m is the message “MarleeBryant+11796088” in hex. Figure 5 shows the encrypt function definition where modular exponentiation is performed. Figure 3 above shows the function’s output of the generated cipher text.

```
BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
BN_hex2bn(&e, "010001");

BIGNUM *m = BN_new();
BN_hex2bn(&m, "4D61726C6565427279616E742B3131373936303838");

BIGNUM *cipher = BN_new();
cipher = encrypt(n,e,m,ctx);
char * ciphText = BN_bn2hex(cipher);
printf("\nCipher Text: %s", ciphText);
```

Fig 4. Encrypt function call in main

```
BIGNUM * encrypt(BIGNUM * n, BIGNUM * e, BIGNUM * m, BN_CTX * ctx) {
  BIGNUM * c = BN_new();

  BN_mod_exp(c, m, e, n, ctx);
  return c;
}
```

Fig 5. Encrypt function definition

Task 3

In this task, the RSA decryption algorithm was implemented. Decryption requires the public key n and private key d in addition to the ciphertext. Similar to encryption, modular exponentiation is used to convert the cipher text to a numerical representation of the message. Figure 6 shows the section of the main function dealing with decryption, including initializing c. Figure 7 shows the decrypt function definition where modular exponentiation is performed. Figure 3 above shows the function’s output, the decrypted message which is “The password is dees.”

```
BIGNUM *c = BN_new();
BN_hex2bn(&c, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");
char * decMes = decrypt(n,d,c,ctx);
printf("\nDecrypted Message: %s", decMes);
```

Fig 6. Decrypt function call in main

```
char * decrypt(BIGNUM * n, BIGNUM * d, BIGNUM * c, BN_CTX * ctx) {
  BIGNUM * m = BN_new();

  BN_mod_exp(m, c, d, n, ctx);
  char * hexM = BN_bn2hex(m);
  return hexM;
}
```

Fig 7. Decrypt function definition

Task 4

In this task, the RSA signature generation algorithm was implemented. Signature Generation requires the public key n and private key d in addition to the message to be signed. Typically a hash of the message is used to generate the signature, but in this case the message's hex value is used directly. The two messages used are "Marlee owes you $2000" and "Marlee owes you $3000." Similar to encryption and decryption, modular exponentiation is used to convert the message to a signature. Figure 8 shows the section of the main function dealing with signature generation, including initializing the signature messages. Figure 9 shows the signature generation function definition where modular exponentiation is performed. Figure 10 shows the signature created for each message, and even though the messages differ by only one character, the signatures are entirely different by means of confusion and diffusion in the algorithm.

```
BIGNUM *sigM1 = BN_new();
BIGNUM *sigM2 = BN_new();
char * sig1hex = "4D61726C6565206F77657320796F75202432303030";
char * sig2hex = "4D61726C6565206F77657320796F75202433303030";
BN_hex2bn(&sigM1, sig1hex);
BN_hex2bn(&sigM2, sig2hex);

BIGNUM *sig1 = BN_new();
sig1 = generateSig(sigM1,n,d,ctx);
BIGNUM *sig2 = BN_new();
sig2 = generateSig(sigM2,n,d,ctx);
char * sig1text = BN_bn2hex(sig1);
printf("\n\nSignature 1: %s", sig1text);
char * sig2text = BN_bn2hex(sig2);
printf("\nSignature 2: %s", sig2text);
```

Fig 8. Signature generation function call in main

```
BIGNUM * generateSig(BIGNUM * m, BIGNUM * n, BIGNUM * d, BN_CTX * ctx) {
  BIGNUM * s = BN_new();

  BN_mod_exp(s, m, d, n, ctx);
  return s;
}
```

Fig 9. Signature generation function definition

```
Signature 1: 1BD60A2210A724597EDF555606B8C26DCA7197C577504A11238312CDB0661AA0
Signature 2: 6FB5D4FF65B5316F894B502433ED89DB3622F6C892DD7B4B41D4C8700927F8CE
```

Fig 10. Generate signature output

Task 5

In this task, the RSA signature verification algorithm was implemented. Signature verification requires the public key pair (n, e) in addition to the message to be signed. The two signatures used vary by only one hex character. Similar to encryption and decryption, modular exponentiation is used to verify a signature. Figure 11 shows the section of the main function dealing with signature verification, including initializing the signatures and a new value of n, as well as comparing the message generated from the signature to the expected message “Launch a missile.” Figure 12 shows the signature verification function definition where modular exponentiation is performed. Figure 13 shows the output for this function, and similar to signature generation, a one character change in the input created drastically different output.

```
BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");

BIGNUM *sigTest = BN_new();
BN_hex2bn(&sigTest, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");
BIGNUM *sigTest2 = BN_new();
BN_hex2bn(&sigTest2, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F");
char *origM = "4C61756E63682061206D697373696C652E";
char *sigM = verifySig(sigTest,e,n,ctx);
char *sigM0 = verifySig(sigTest2,e,n,ctx);

printf("\n\nSent Message: %s", origM);
printf("\nMessage from Signature 1: %s", sigM);
if(strcmp(origM,sigM) == 0)
  printf("\nThis message was sent by Alice.");
else
  printf("\nThis message was not sent by Alice.");
printf("\nMessage from Signature 2: %s", sigM0);
if(strcmp(origM,sigM0) == 0)
  printf("\nThis message was sent by Alice.\n");
else
  printf("\nThis message was not sent by Alice.\n");
```

Fig 11. Verify signature function call in main

```
char * verifySig(BIGNUM * s, BIGNUM * e, BIGNUM * n, BN_CTX * ctx) {
  BIGNUM * m = BN_new();

  BN_mod_exp(m, s, e, n, ctx);
  char * hexM = BN_bn2hex(m);
  return hexM;
}
```

Fig 12. Verify signature function definition

```
Sent Message: 4C61756E63682061206D697373696C652E
Message from Signature 1: 4C61756E63682061206D697373696C652E
This message was sent by Alice.
Message from Signature 2: 91471927C80DF1E42C154FB4638CE8BC726D3D66C83A4EB6B7BE02
03B41AC294
This message was not sent by Alice.
```

Fig 13. Signature verification output

Task 6

This task displays one of the use cases of RSA in the form of signatures on X.509 certificates. Provided openssl commands can be used to extract the certificate, Figure 14, and to parse the issuer's certificate for the public key pair, Figure 15, and the server's certificate for the signature and body, Figure 16 and 17. The body must then be hashed using SHA256, Figure 18, since the signature was generated with a hash of the certificate body. Using the signature verification algorithm created in Task 5 with the values parsed from the certificates generates a message equivalent to the hash of the certificate body if the signature is valid. The website used for this example is www.andrewsfcu.org.

```
mabryant4@DESKTOP-8QNKU9J:/mnt/c/users/marle/desktop/cs548$  openssl s_client -connect www.andrewsfcu.org:443 -showcerts
CONNECTED(00000003)
depth=2 C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert High Assurance EV Root CA
verify return:1
depth=1 C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert SHA2 Extended Validation Server CA
verify return:1
depth=0 businessCategory = Private Organization, jurisdictionC = US, serialNumber = 5754, C = US, ST = Maryland, L = Suitland, O = ANDREWS FEDERAL CREDIT UNION, CN = andr
sfcu.org
verify return:1
---
Certificate chain
 0 s:/businessCategory=Private Organization/jurisdictionC=US/serialNumber=5754/C=US/ST=Maryland/L=Suitland/O=ANDREWS FEDERAL CREDIT UNION/CN=andrewsfcu.org
   i:/C=US/O=DigiCert Inc/OU=www.digicert.com/CN=DigiCert SHA2 Extended Validation Server CA
-----BEGIN CERTIFICATE-----
MIIHOTCCBiGgAwIBAgIQA/0RXe8powrzRD3OrbGMszANBgkqhkiG9w0BAQsFADB1
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3
d3cuZGlnaWNlcnQuY29tMTQwMgYDVQQDEytEaWdpQ2VydCBTSEEyIEV4dGVuZGVk
IFZhbGlkYXRpb24gU2VydmVyIENBMB4XDTIxMDkyMTAwMDAwMFoXDTIyMTAwMTIz
NTk1OVowgbYxHTAbBgNVBA8MFFByaXZhdGUgT3JnYW5pemF0aW9uMRMwEQYLKwYB
BAGCNzwCAQMTAlVTMQ0wCwYDVQQFEwQ1NzU0MQswCQYDVQQGEwJVUzERMA8GA1UE
CBMITWFyeWxhbmQxETAPBgNVBAcTCFN1aXRsYW5kMSUwIwYDVQQKExxBTkRSRVdT
IEZFREVSQUwgQ1JFRElUIFVOSU9OMRcwFQYDVQQDEw5hbmRyZXdzZmN1Lm9yZzCC
ASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAKnhDGZ/5tQh19K0bEfAmWhM
UK6UtjVMhu4ML5wO+JFOL7DIWln/jSQoZIPJijtsoQKEhbSXffYH5TUShQqfHwYE
neSxrRVOpHYAxtk+mlzt7UEecz/v2aszWqUviKEaem7QCoG/Q+NrEgPySZucD8KD
buvu5kRZO4SETRfq6FUeZjC6u736UO0g9ENVmIyPj8Q0O6JJnYWRIHnktBoCRT1S
PwbQWJiz/X+NtHeNEtBE66F/TTkydXB/snJBsRcLbGBBNwq9gE++lweeeyjteY6B
qFf8XI/6y30pT3g649aUGfImIZhxweSsgoBvxbRshDFMO+wnlkVgrHT6IQU5nZUC
AwEAAaOCA4EwggN9MB8GA1UdIwQYMBaAFD3TUKXWoK3u80pgCmXTIdT4+NYPMB0G
A1UdDgQWBBQszlhKRuU71+rBW++SVoKq+pMsfzAtBgNVHREEJjAkgg5hbmRyZXdz
ZmN1Lm9yZ4ISd3d3LmFuZHJld3NmY3Uub3JnMA4GA1UdDwEB/wQEAwIFoDAdBgNV
HSUEFjAUBggrBgEFBQcDAQYIKwYBBQUHAwIwdQYDVR0fBG4wbDA0oDKgMIYuaHR0
cDovL2NybDMuZGlnaWNlcnQuY29tL3NoYTItZXYtc2VydmVyLWczLmNybDA0oDKg
MIYuaHR0cDovL2NybDQuZGlnaWNlcnQuY29tL3NoYTItZXYtc2VydmVyLWczLmNy
bDBKBgNVHSAEQzBBMAsGCWCGSAGG/WwCATAyBgVngQwBATApMCcGCCsGAQUFBwIB
FhtodHRwOi8vd3d3LmRpZ2ljZXJ0LmNvbS9DUFMwgYgGCCsGAQUFBwEBBHwwejAk
BggrBgEFBQcwAYYYaHR0cDovL29jc3AuZGlnaWNlcnQuY29tMFIGCCsGAQUFBzAC
hkZodHRwOi8vY2FjZXJ0cy5kaWdpY2VydC5jb20vRGlnaUNlcnRTSEEyRXh0ZW5k
ZWRWYWxpZGF0aW9uU2VydmVyQ0EuY3J0MAwGA1UdEwEB/wQCMAAwggF/BgorBgEE
AdZ5AgQCBIIBbwSCAWsBaQB2ACl5vvCeOTkh8FZzn2Old+W+V32cYAr4+U1dJlwl
XceEAAABfAmX9PgAAAQDAEcwRQIgHuw2O7MKawNHV5gciGltBkxNJ9+odbVun/Qw
```

Fig 14. Task 6 Step 1: obtaining certificates

```
mabryant4@DESKTOP-8QNKU9J:/mnt/c/users/marle/desktop/cs548$ openssl x509 -in c2.pem -noout -modulus
Modulus=D753A40451F899A616484B6727AA9349D039ED0CB0B00087F1672886858C8E63DABCB14038E2D3F5ECA50518B83D3EC5991732EC188CFAF10CA6642185CB0710
1FEF387754535F80793F2E1AAAA81E4B2B0DABB763B935B77D14BC594BDF514AD2A1E20CE29082876AAEEAD764D69855E8FDAF1A506C54BC11F2FD4AF29DBB7F0EF4D5BE
8DD821E4B04D0C89DC392617DDF6D79485D80421709D6F6FFF5CBA19E145CB5657287E1C0D4157AAB7B827BBB1E4FA2AEF2123751AAD2D9B86358C9C77B573ADD8942DE4
1933
mabryant4@DESKTOP-8QNKU9J:/mnt/c/users/marle/desktop/cs548$ openssl x509 -in c2.pem -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            0c:79:a9:44:b0:8c:11:95:20:92:61:5f:e2:6b:1d:83
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, O=DigiCert Inc, OU=www.digicert.com, CN=DigiCert High Assurance EV Root CA
        Validity
            Not Before: Oct 22 12:00:00 2013 GMT
            Not After : Oct 22 12:00:00 2028 GMT
        Subject: C=US, O=DigiCert Inc, OU=www.digicert.com, CN=DigiCert SHA2 Extended Validation Server CA
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:d7:53:a4:04:51:f8:99:a6:16:48:4b:67:27:aa:
                    93:49:d0:39:ed:0c:b0:b0:00:87:f1:67:28:86:85:
                    8c:8e:63:da:bc:b1:40:38:e2:d3:f5:ec:a5:05:18:
                    b8:3d:3e:c5:99:17:32:ec:18:8c:fa:f1:0c:a6:64:
                    21:85:cb:07:10:34:b0:52:88:2b:1f:68:9b:d2:b1:
                    8f:12:b0:b3:d2:e7:88:1f:1f:ef:38:77:54:53:5f:
                    80:79:3f:2e:1a:aa:a8:1e:4b:2b:0d:ab:b7:63:b9:
                    35:b7:7d:14:bc:59:4b:df:51:4a:d2:a1:e2:0c:e2:
                    90:82:87:6a:ae:ea:d7:64:d6:98:55:e8:fd:af:1a:
                    50:6c:54:bc:11:f2:fd:4a:f2:9d:bb:7f:0e:f4:d5:
                    be:8e:16:89:12:55:d8:c0:71:34:ee:f6:dc:2d:ec:
                    c4:87:25:86:8d:d8:21:e4:b0:4d:0c:89:dc:39:26:
                    17:dd:f6:d7:94:85:d8:04:21:70:9d:6f:6f:ff:5c:
                    ba:19:e1:45:cb:56:57:28:7e:1c:0d:41:57:aa:b7:
                    b8:27:bb:b1:e4:fa:2a:ef:21:23:75:1a:ad:2d:9b:
                    86:35:8c:9c:77:b5:73:ad:d8:94:2d:e4:f3:0c:9d:
                    ee:c1:4e:62:7e:17:c0:71:9e:2c:de:f1:f9:10:28:
                    19:33
                Exponent: 65537 (0x10001)
        X509v3 extensions:
```

Fig 15. Task 6 Step 2: obtaining public key pair (n, e)

```
Signature Algorithm: sha256WithRSAEncryption
     28:4a:2a:0d:1c:48:cf:00:41:87:df:13:a0:3f:8a:aa:7c:33:
     fc:a8:20:12:69:4e:99:0c:91:b1:3c:44:33:b7:6b:5b:82:de:
     4e:4f:09:27:29:85:9c:b1:66:c2:c9:ca:ef:83:7d:6e:57:40:
     ac:99:02:a5:3a:21:80:d2:f2:0c:3f:84:61:e9:73:90:8e:c9:
     f7:82:1b:8d:4a:7a:21:18:61:2e:a7:05:08:c4:01:8b:bb:df:
     f9:02:28:1c:7a:52:e0:94:2c:64:29:aa:f9:c5:d9:aa:12:d8:
     cd:f8:9b:90:bb:87:c9:cc:25:55:aa:cb:64:e5:76:fb:76:76:
     ba:11:2f:14:22:02:a6:30:c3:b4:92:89:60:ab:e2:2e:c0:7a:
     2b:34:b5:57:55:e1:f4:9f:12:d0:c4:a9:47:21:ef:ed:c0:77:
     7c:77:d8:c9:77:c5:92:f8:3d:95:0e:f9:59:ba:27:c2:0b:09:
     a8:c3:03:02:99:cc:32:90:63:b4:b0:3b:1b:e3:63:5c:ce:6e:
     bd:6e:f5:2b:54:a9:58:d1:c7:7e:68:c2:ec:7f:a4:89:25:bd:
     f3:06:8f:11:cd:8a:5c:2f:01:46:98:60:c5:79:d6:dc:cb:41:
     c7:08:b1:cc:e1:39:b4:85:8b:8e:d2:e9:e5:58:b3:68:e7:4c:
     1d:1d:61:11
```

Fig 16. Task 6 Step 3: obtaining the signature

```
mabryant4@DESKTOP-8QNKU9J:/mnt/c/users/marle/desktop/cs548$  openssl asn1parse -i -in c1.pem
    0:d=0  hl=4 l=1849 cons: SEQUENCE
    4:d=1  hl=4 l=1569 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=  16 prim:   INTEGER           :03FD115DEF29A30AF3443DCEADB18CB3
   31:d=2  hl=2 l=  13 cons:   SEQUENCE
   33:d=3  hl=2 l=   9 prim:    OBJECT            :sha256WithRSAEncryption
   44:d=3  hl=2 l=   0 prim:    NULL
   46:d=2  hl=2 l= 117 cons:   SEQUENCE
   48:d=3  hl=2 l=  11 cons:    SET
   50:d=4  hl=2 l=   9 cons:     SEQUENCE
   52:d=5  hl=2 l=   3 prim:      OBJECT            :countryName
   57:d=5  hl=2 l=   2 prim:      PRINTABLESTRING   :US
   61:d=3  hl=2 l=  21 cons:    SET
   63:d=4  hl=2 l=  19 cons:     SEQUENCE
   65:d=5  hl=2 l=   3 prim:      OBJECT            :organizationName
   70:d=5  hl=2 l=  12 prim:      PRINTABLESTRING   :DigiCert Inc
   84:d=3  hl=2 l=  25 cons:    SET
   86:d=4  hl=2 l=  23 cons:     SEQUENCE
   88:d=5  hl=2 l=   3 prim:      OBJECT            :organizationalUnitName
   93:d=5  hl=2 l=  16 prim:      PRINTABLESTRING   :www.digicert.com
  111:d=3  hl=2 l=  52 cons:    SET
  113:d=4  hl=2 l=  50 cons:     SEQUENCE
  115:d=5  hl=2 l=   3 prim:      OBJECT            :commonName
  120:d=5  hl=2 l=  43 prim:      PRINTABLESTRING   :DigiCert SHA2 Extended Validation Server CA
  165:d=2  hl=2 l=  30 cons:   SEQUENCE
  167:d=3  hl=2 l=  13 prim:    UTCTIME           :210921000000Z
  182:d=3  hl=2 l=  13 prim:    UTCTIME           :221001235959Z
  197:d=2  hl=3 l= 182 cons:   SEQUENCE
  200:d=3  hl=2 l=  29 cons:    SET
  202:d=4  hl=2 l=  27 cons:     SEQUENCE
  204:d=5  hl=2 l=   3 prim:      OBJECT            :businessCategory
  209:d=5  hl=2 l=  20 prim:      UTF8STRING        :Private Organization
  231:d=3  hl=2 l=  19 cons:    SET
  233:d=4  hl=2 l=  17 cons:     SEQUENCE
  235:d=5  hl=2 l=  11 prim:      OBJECT            :jurisdictionCountryName
  248:d=5  hl=2 l=   2 prim:      PRINTABLESTRING   :US
  252:d=3  hl=2 l=  13 cons:    SET
```

Fig 17. Task 6 Step 4 part 1: obtaining the certificate body

```
mabryant4@DESKTOP-8QNKU9J:/mnt/c/users/marle/desktop/cs548$ openssl dgst -sha256 c0_body.bin
SHA256(c0_body.bin)= abb7edf2d755f5472b63c161d2976c6e0e173dde1bdf442572d902a05f4542e3
mabryant4@DESKTOP-8QNKU9J:/mnt/c/users/marle/desktop/cs548$
```

Fig 18. Task 6 Step 4 part 2: generating a hash of the certificate body