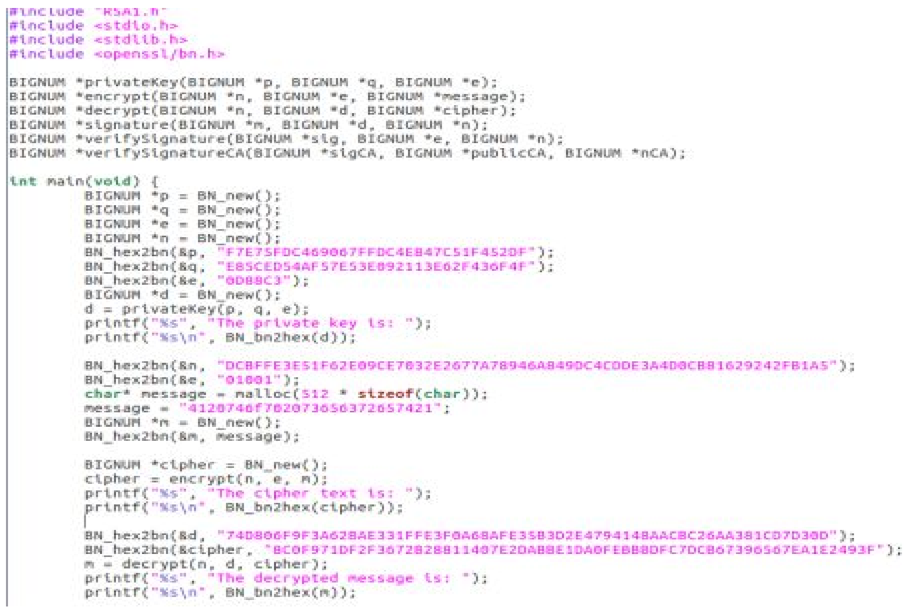
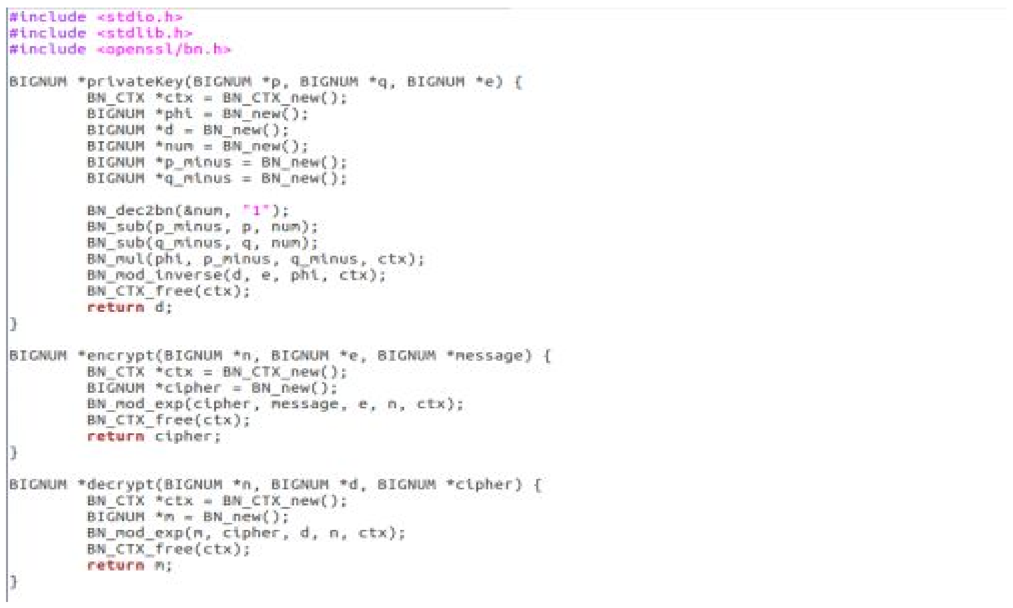
Nazim Zerrouki
Greg Gertsen

<u>Task 1, 2, &3:</u>

This is the code that was used to compute the RSA private key and encrypt&decrypt data:

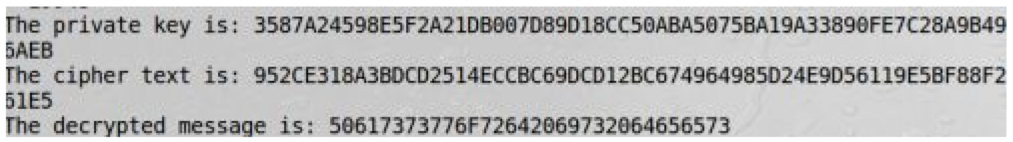


```c
#include "RSA1.h"
#include <stdio.h>
#include <stdlib.h>
#include <openssl/bn.h>

BIGNUM *privateKey(BIGNUM *p, BIGNUM *q, BIGNUM *e);
BIGNUM *encrypt(BIGNUM *n, BIGNUM *e, BIGNUM *message);
BIGNUM *decrypt(BIGNUM *n, BIGNUM *d, BIGNUM *cipher);
BIGNUM *signature(BIGNUM *m, BIGNUM *d, BIGNUM *n);
BIGNUM *verifySignature(BIGNUM *sig, BIGNUM *e, BIGNUM *n);
BIGNUM *verifySignatureCA(BIGNUM *sigCA, BIGNUM *publicCA, BIGNUM *nCA);

int main(void) {
        BIGNUM *p = BN_new();
        BIGNUM *q = BN_new();
        BIGNUM *e = BN_new();
        BIGNUM *n = BN_new();
        BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
        BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
        BN_hex2bn(&e, "0D88C3");
        BIGNUM *d = BN_new();
        d = privateKey(p, q, e);
        printf("%s", "The private key is: ");
        printf("%s\n", BN_bn2hex(d));

        BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
        BN_hex2bn(&e, "01001");
        char* message = malloc(512 * sizeof(char));
        message = "4120746f702073656372657421";
        BIGNUM *m = BN_new();
        BN_hex2bn(&m, message);

        BIGNUM *cipher = BN_new();
        cipher = encrypt(n, e, m);
        printf("%s", "The cipher text is: ");
        printf("%s\n", BN_bn2hex(cipher));

        BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
        BN_hex2bn(&cipher, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");
        m = decrypt(n, d, cipher);
        printf("%s", "The decrypted message is: ");
        printf("%s\n", BN_bn2hex(m));
```



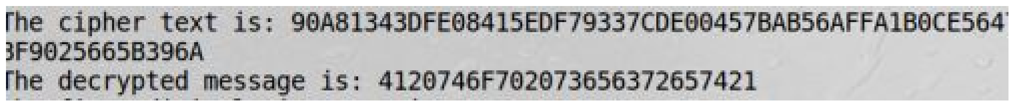
```c
#include <stdio.h>
#include <stdlib.h>
#include <openssl/bn.h>

BIGNUM *privateKey(BIGNUM *p, BIGNUM *q, BIGNUM *e) {
        BN_CTX *ctx = BN_CTX_new();
        BIGNUM *phi = BN_new();
        BIGNUM *d = BN_new();
        BIGNUM *num = BN_new();
        BIGNUM *p_minus = BN_new();
        BIGNUM *q_minus = BN_new();

        BN_dec2bn(&num, "1");
        BN_sub(p_minus, p, num);
        BN_sub(q_minus, q, num);
        BN_mul(phi, p_minus, q_minus, ctx);
        BN_mod_inverse(d, e, phi, ctx);
        BN_CTX_free(ctx);
        return d;
}

BIGNUM *encrypt(BIGNUM *n, BIGNUM *e, BIGNUM *message) {
        BN_CTX *ctx = BN_CTX_new();
        BIGNUM *cipher = BN_new();
        BN_mod_exp(cipher, message, e, n, ctx);
        BN_CTX_free(ctx);
        return cipher;
}

BIGNUM *decrypt(BIGNUM *n, BIGNUM *d, BIGNUM *cipher) {
        BN_CTX *ctx = BN_CTX_new();
        BIGNUM *m = BN_new();
        BN_mod_exp(m, cipher, d, n, ctx);
        BN_CTX_free(ctx);
        return m;
}
```

And the produced results:



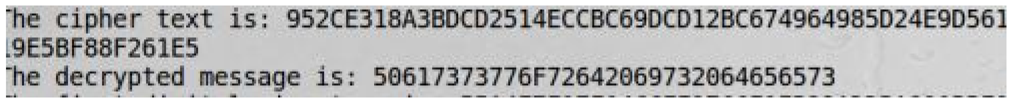
```
The private key is: 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B49
5AEB
The cipher text is: 952CE318A3BDCD2514ECCBC69DCD12BC674964985D24E9D56119E5BF88F2
61E5
The decrypted message is: 50617373776F72642069732064656573
```

Using the same modulus and exponent gets me to the same message when I use RSA encryption and decryption as well:



```
The cipher text is: 90A81343DFE08415EDF79337CDE00457BAB56AFFA1B0CE564
3F9025665B396A
The decrypted message is: 4120746F702073656372657421
```

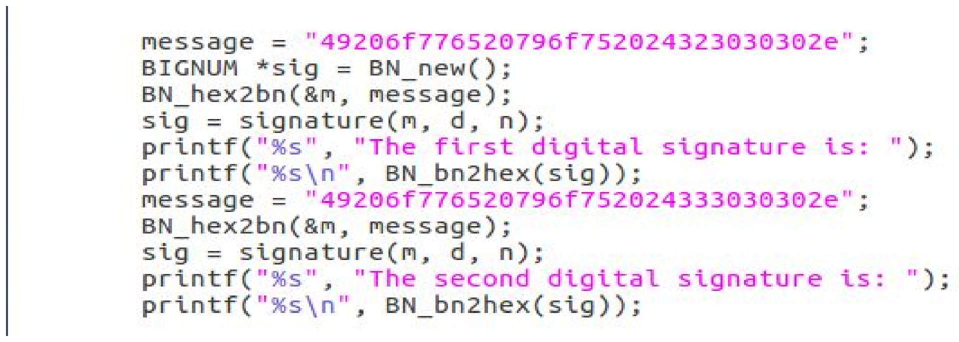
Correction, the decrypted message was wrong when using the new values, here is the new ciphertext and decrypted message:



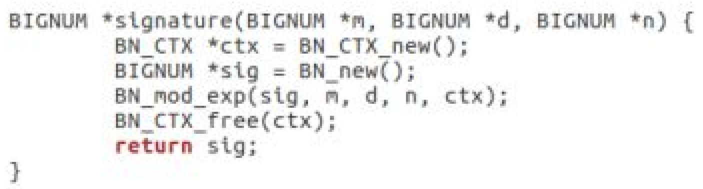
```
The cipher text is: 952CE318A3BDCD2514ECCBC69DCD12BC674964985D24E9D561
19E5BF88F261E5
The decrypted message is: 50617373776F72642069732064656573
```

Task 4:

This is the code that was used to produce the digital signatures:

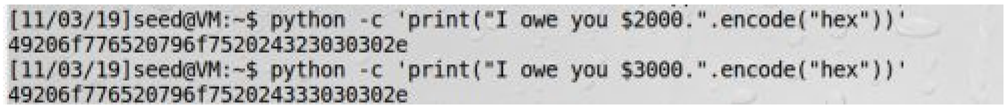


```
        message = "49206f776520796f752024323030302e";
        BIGNUM *sig = BN_new();
        BN_hex2bn(&m, message);
        sig = signature(m, d, n);
        printf("%s", "The first digital signature is: ");
        printf("%s\n", BN_bn2hex(sig));
        message = "49206f776520796f752024333030302e";
        BN_hex2bn(&m, message);
        sig = signature(m, d, n);
        printf("%s", "The second digital signature is: ");
        printf("%s\n", BN_bn2hex(sig));
```

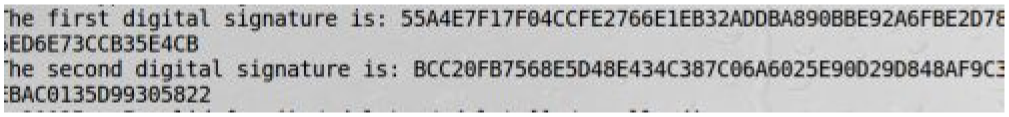


```
BIGNUM *signature(BIGNUM *m, BIGNUM *d, BIGNUM *n) {
        BN_CTX *ctx = BN_CTX_new();
        BIGNUM *sig = BN_new();
        BN_mod_exp(sig, m, d, n, ctx);
        BN_CTX_free(ctx);
        return sig;
}
```

And the produced results:



```
[11/03/19]seed@VM:~$ python -c 'print("I owe you $2000.".encode("hex"))'
49206f776520796f752024323030302e
[11/03/19]seed@VM:~$ python -c 'print("I owe you $3000.".encode("hex"))'
49206f776520796f752024333030302e
```

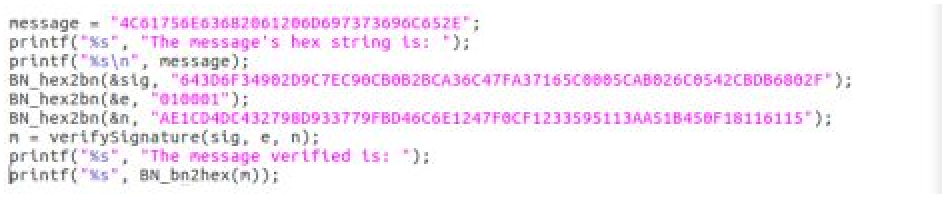


```
he first digital signature is: 55A4E7F17F04CCFE2766E1EB32ADDBA890BBE92A6FBE2D7
ED6E73CCB35E4CB
he second digital signature is: BCC20FB7568E5D48E434C387C06A6025E90D29D848AF9C
BAC0135D99305822
```

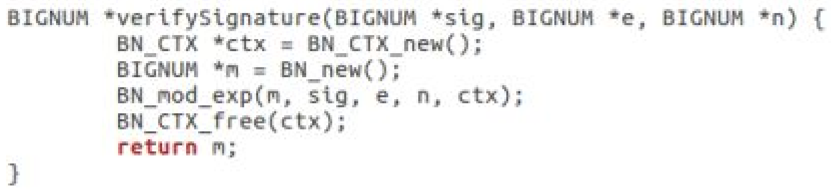
Changing the message only resulted in one bit flip, but one bit flip was enough to change the entire byte sequence of the digital signature.

Task 5:

This is the code that was used to verify the signature:

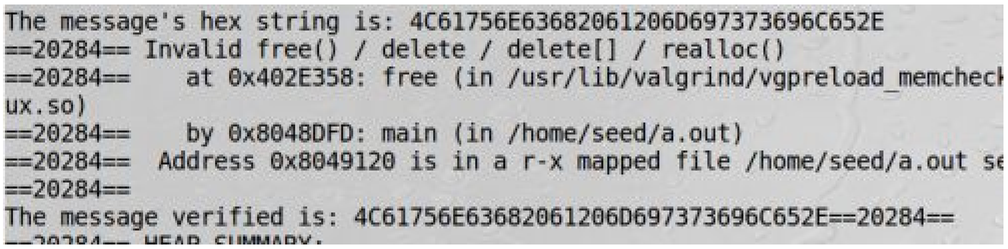


```
message = "4C61756E63682061206D697373696C652E";
printf("%s", "The message's hex string is: ");
printf("%s\n", message);
BN_hex2bn(&sig, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");
BN_hex2bn(&e, "010001");
BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");
m = verifySignature(sig, e, n);
printf("%s", "The message verified is: ");
printf("%s", BN_bn2hex(m));
```



```
BIGNUM *verifySignature(BIGNUM *sig, BIGNUM *e, BIGNUM *n) {
        BN_CTX *ctx = BN_CTX_new();
        BIGNUM *m = BN_new();
        BN_mod_exp(m, sig, e, n, ctx);
        BN_CTX_free(ctx);
        return m;
}
```

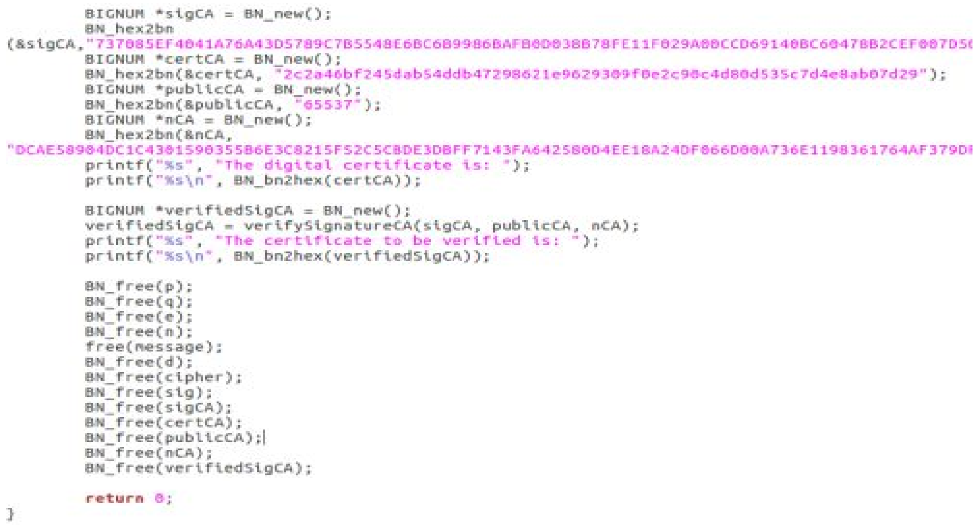
And this is the produced result:



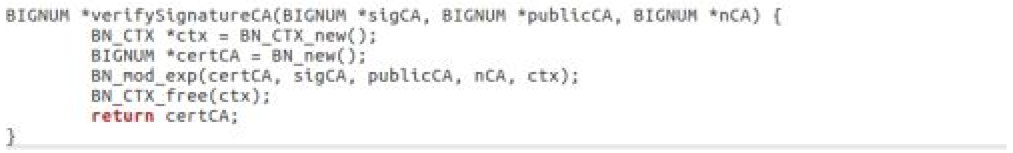
```
The message's hex string is: 4C61756E63682061206D697373696C652E
==20284== Invalid free() / delete / delete[] / realloc()
==20284==    at 0x402E358: free (in /usr/lib/valgrind/vgpreload_memcheck
ux.so)
==20284==    by 0x8048DFD: main (in /home/seed/a.out)
==20284==  Address 0x8049120 is in a r-x mapped file /home/seed/a.out se
==20284==
The message verified is: 4C61756E63682061206D697373696C652E==20284==
```

Task 6:

This is the code that was used to verify the digital signature:

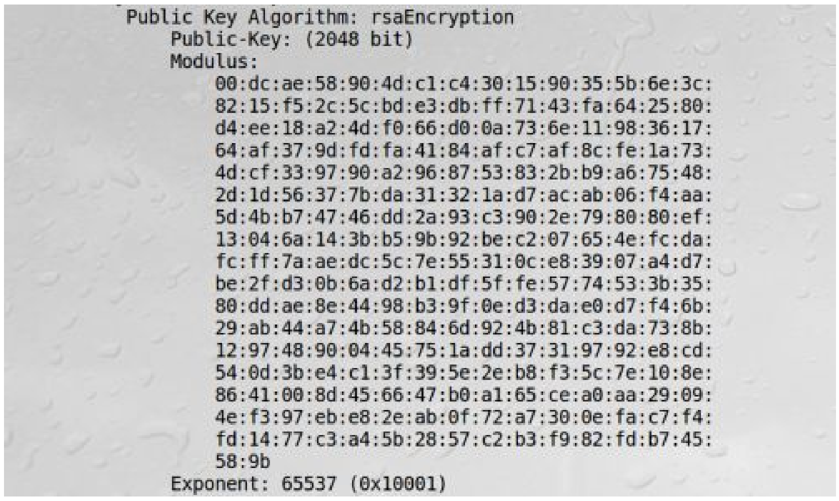


```
        BIGNUM *sigCA = BN_new();
        BN_hex2bn
(&sigCA,"737085EF4041A76A43D5789C7B5548E6BC6B9986BAFB0D038B78FE11F029A00CCD69140BC60478B2CEF007D5
        BIGNUM *certCA = BN_new();
        BN_hex2bn(&certCA, "2c2a46bf245dab54ddb47298621e9629309f0e2c98c4d80d535c7d4e8ab07d29");
        BIGNUM *publicCA = BN_new();
        BN_hex2bn(&publicCA, "65537");
        BIGNUM *nCA = BN_new();
        BN_hex2bn(&nCA,
"DCAE58904DC1C4301590355B6E3C8215F52C5CBDE3DBFF7143FA642580D4EE18A24DF066D00A736E1198361764AF379D
        printf("%s", "The digital certificate is: ");
        printf("%s\n", BN_bn2hex(certCA));

        BIGNUM *verifiedSigCA = BN_new();
        verifiedSigCA = verifySignatureCA(sigCA, publicCA, nCA);
        printf("%s", "The certificate to be verified is: ");
        printf("%s\n", BN_bn2hex(verifiedSigCA));

        BN_free(p);
        BN_free(q);
        BN_free(e);
        BN_free(n);
        free(message);
        BN_free(d);
        BN_free(cipher);
        BN_free(sig);
        BN_free(sigCA);
        BN_free(certCA);
        BN_free(publicCA);
        BN_free(nCA);
        BN_free(verifiedSigCA);

        return 0;
}
```

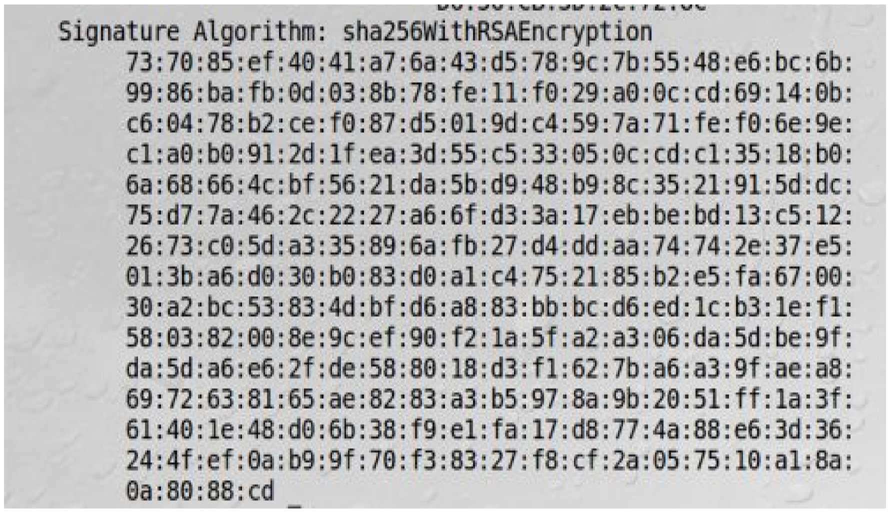


```
BIGNUM *verifySignatureCA(BIGNUM *sigCA, BIGNUM *publicCA, BIGNUM *nCA) {
        BN_CTX *ctx = BN_CTX_new();
        BIGNUM *certCA = BN_new();
        BN_mod_exp(certCA, sigCA, publicCA, nCA, ctx);
        BN_CTX_free(ctx);
        return certCA;
}
```

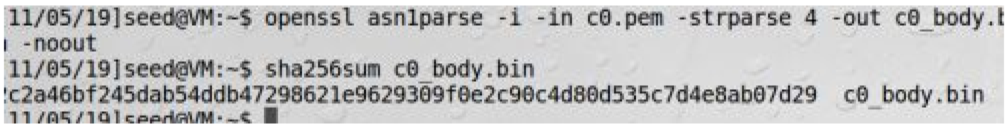
The steps to produce the code:



```
        Public Key Algorithm: rsaEncryption
            Public-Key: (2048 bit)
            Modulus:
                00:dc:ae:58:90:4d:c1:c4:30:15:90:35:5b:6e:3c:
                82:15:f5:2c:5c:bd:e3:db:ff:71:43:fa:64:25:80:
                d4:ee:18:a2:4d:f0:66:d0:0a:73:6e:11:98:36:17:
                64:af:37:9d:fd:fa:41:84:af:c7:af:8c:fe:1a:73:
                4d:cf:33:97:90:a2:96:87:53:83:2b:b9:a6:75:48:
                2d:1d:56:37:7b:da:31:32:1a:d7:ac:ab:06:f4:aa:
                5d:4b:b7:47:46:dd:2a:93:c3:90:2e:79:80:80:ef:
                13:04:6a:14:3b:b5:9b:92:be:c2:07:65:4e:fc:da:
                fc:ff:7a:ae:dc:5c:7e:55:31:0c:e8:39:07:a4:d7:
                be:2f:d3:0b:6a:d2:b1:df:5f:fe:57:74:53:3b:35:
                80:dd:ae:8e:44:98:b3:9f:0e:d3:da:e0:d7:f4:6b:
                29:ab:44:a7:4b:58:84:6d:92:4b:81:c3:da:73:8b:
                12:97:48:90:04:45:75:1a:dd:37:31:97:92:e8:cd:
                54:0d:3b:e4:c1:3f:39:5e:2e:b8:f3:5c:7e:10:8e:
                86:41:00:8d:45:66:47:b0:a1:65:ce:a0:aa:29:09:
                4e:f3:97:eb:e8:2e:ab:0f:72:a7:30:0e:fa:c7:f4:
                fd:14:77:c3:a4:5b:28:57:c2:b3:f9:82:fd:b7:45:
                58:9b
            Exponent: 65537 (0x10001)
```

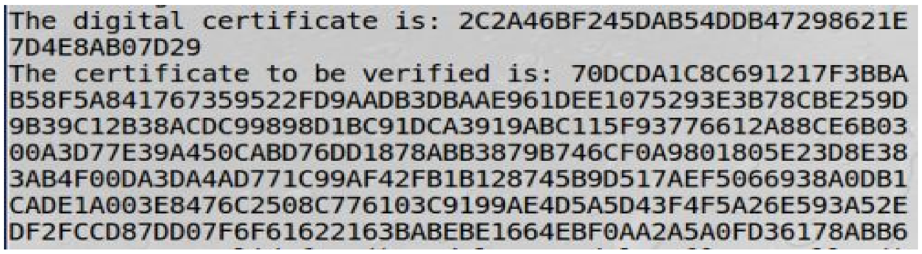


```
    Signature Algorithm: sha256WithRSAEncryption
         73:70:85:ef:40:41:a7:6a:43:d5:78:9c:7b:55:48:e6:bc:6b:
         99:86:ba:fb:0d:03:8b:78:fe:11:f0:29:a0:0c:cd:69:14:0b:
         c6:04:78:b2:ce:f0:87:d5:01:9d:c4:59:7a:71:fe:f0:6e:9e:
         c1:a0:b0:91:2d:1f:ea:3d:55:c5:33:05:0c:cd:c1:35:18:b0:
         6a:68:66:4c:bf:56:21:da:5b:d9:48:b9:8c:35:21:91:5d:dc:
         75:d7:7a:46:2c:22:27:a6:6f:d3:3a:17:eb:be:bd:13:c5:12:
         26:73:c0:5d:a3:35:89:6a:fb:27:d4:dd:aa:74:74:2e:37:e5:
         01:3b:a6:d0:30:b0:83:d0:a1:c4:75:21:85:b2:e5:fa:67:00:
         30:a2:bc:53:83:4d:bf:d6:a8:83:bb:bc:d6:ed:1c:b3:1e:f1:
         58:03:82:00:8e:9c:ef:90:f2:1a:5f:a2:a3:06:da:5d:be:9f:
         da:5d:a6:e6:2f:de:58:80:18:d3:f1:62:7b:a6:a3:9f:ae:a8:
         69:72:63:81:65:ae:82:83:a3:b5:97:8a:9b:20:51:ff:1a:3f:
         61:40:1e:48:d0:6b:38:f9:e1:fa:17:d8:77:4a:88:e6:3d:36:
         24:4f:ef:0a:b9:9f:70:f3:83:27:f8:cf:2a:05:75:10:a1:8a:
         0a:80:88:cd _
```



```
11/05/19]seed@VM:~$ openssl asn1parse -i -in c0.pem -strparse 4 -out c0_body.b
 -noout
11/05/19]seed@VM:~$ sha256sum c0_body.bin
c2a46bf245dab54ddb47298621e9629309f0e2c90c4d80d535c7d4e8ab07d29  c0_body.bin
11/05/19]seed@VM:~$ █
```

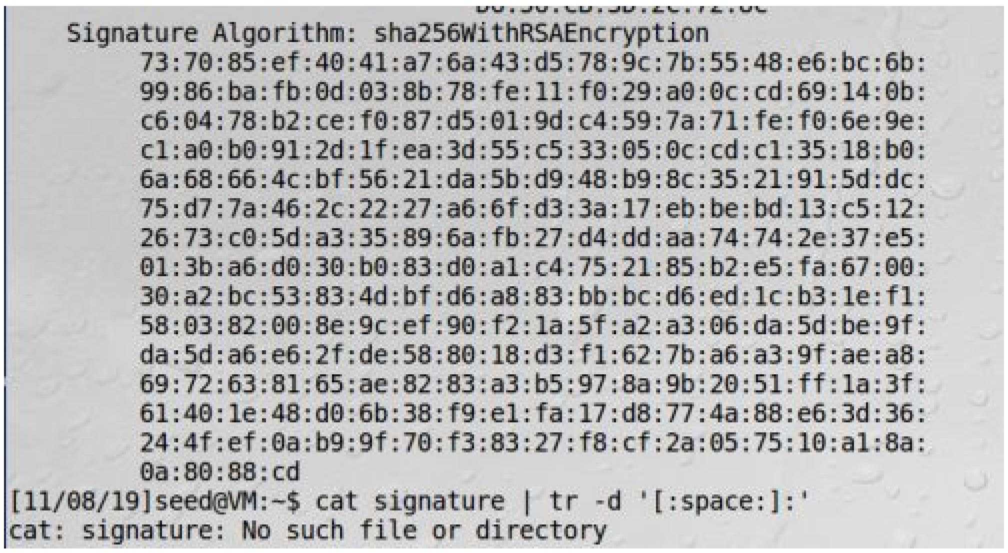
And the produced results:



```
The digital certificate is: 2C2A46BF245DAB54DDB47298621E
7D4E8AB07D29
The certificate to be verified is: 70DCDA1C8C691217F3BBA
B58F5A841767359522FD9AADB3DBAAE961DEE1075293E3B78CBE259D
9B39C12B38ACDC99898D1BC91DCA3919ABC115F93776612A88CE6B03
00A3D77E39A450CABD76DD1878ABB3879B746CF0A9801805E23D8E38
3AB4F00DA3DA4AD771C99AF42FB1B128745B9D517AEF5066938A0DB1
CADE1A003E8476C2508C776103C9199AE4D5A5D43F4F5A26E593A52E
DF2FCCD87DD07F6F61622163BABEBE1664EBF0AA2A5A0FD36178ABB6
```

The code to be used to verify the message should be correct even though the produced output is wrong. To get the certificate, we take (sig(cert)^m) mod nCA and that should get the right result. Unfortunately the result is wrong and we don't understand why. Hopefully with more time, we can figure out why.

Update:



```
    Signature Algorithm: sha256WithRSAEncryption
         73:70:85:ef:40:41:a7:6a:43:d5:78:9c:7b:55:48:e6:bc:6b:
         99:86:ba:fb:0d:03:8b:78:fe:11:f0:29:a0:0c:cd:69:14:0b:
         c6:04:78:b2:ce:f0:87:d5:01:9d:c4:59:7a:71:fe:f0:6e:9e:
         c1:a0:b0:91:2d:1f:ea:3d:55:c5:33:05:0c:cd:c1:35:18:b0:
         6a:68:66:4c:bf:56:21:da:5b:d9:48:b9:8c:35:21:91:5d:dc:
         75:d7:7a:46:2c:22:27:a6:6f:d3:3a:17:eb:be:bd:13:c5:12:
         26:73:c0:5d:a3:35:89:6a:fb:27:d4:dd:aa:74:74:2e:37:e5:
         01:3b:a6:d0:30:b0:83:d0:a1:c4:75:21:85:b2:e5:fa:67:00:
         30:a2:bc:53:83:4d:bf:d6:a8:83:bb:bc:d6:ed:1c:b3:1e:f1:
         58:03:82:00:8e:9c:ef:90:f2:1a:5f:a2:a3:06:da:5d:be:9f:
         da:5d:a6:e6:2f:de:58:80:18:d3:f1:62:7b:a6:a3:9f:ae:a8:
         69:72:63:81:65:ae:82:83:a3:b5:97:8a:9b:20:51:ff:1a:3f:
         61:40:1e:48:d0:6b:38:f9:e1:fa:17:d8:77:4a:88:e6:3d:36:
         24:4f:ef:0a:b9:9f:70:f3:83:27:f8:cf:2a:05:75:10:a1:8a:
         0a:80:88:cd
[11/08/19]seed@VM:~$ cat signature | tr -d '[:space:]:'
cat: signature: No such file or directory
```

I could not solve the issue. We repeated the steps and fixed the exponent (should be 010001), but the issue was not resolved. Unfortunately this command didn't work, so we had to type it directly into my program. We believe the issue isn't with our function because the idea behind it is to decrypt the signature using the CA's public key and modding it by n. Therefore, we believe the issue is due to user error.