Dilnoza Saidova, Nathan Mahnke
TCSS381: Computer Security
November 3, 2022

Lab 2: RSA Public-Key Encryption and Signature Lab

Task 1. Deriving the Private Key

Given code solves the first task of this lab:

```
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 128
/* printBN is a helper function: receives a char and two BIGNUMs.
   It converts the BIGNUMs to hex and prints their values after the msg.
 */
void printBN(char *msg, BIGNUM *a, BIGNUM *b) {
    char *st1 = BN_bn2hex(a);
    char *st2 = BN_bn2hex(b);
    printf("%s (%s,%s)\n", msg, st1, st2);
    OPENSSL_free(st1);
    OPENSSL_free(st2);
}
int main() {
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM *p = BN_new();
    BIGNUM *q = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *phi = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *res = BN_new();
    BIGNUM *pMinusOne = BN_new();
    BIGNUM *qMinusOne = BN_new();
    /* p, q, e – arbitrary prime numbers.
       Note they are only 128 bits for simplicity - usually 512 bits
     */
    BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
    BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
    BN_hex2bn(&e, "0D88C3");
    // Define n as p*q
    BN_mul(n, p, q, ctx);
    // Print public key
    printBN("Public key: ", e, n);
    // Define pMinusOne as (p-1)
    BN_sub(pMinusOne, p, BN_value_one());
    // Define qMinusOne as (q-1)
    BN_sub(qMinusOne, q, BN_value_one());
    // Define phi(n) as (p-1)*(q-1)
    BN_mul(phi, pMinusOne, qMinusOne, ctx);
    // Output error and exit program if e and phi aren't relatively prime.
```

```
    BN_gcd(res, phi, e, ctx);
    if (!BN_is_one(res)) {
        printf("Improper input. e and/or phi are not relatively prime.
Exitting program.\n");
        exit(0);
    }
    /* Derive the private key given the values of e and phi.
       Store the result in d
     */
    BN_mod_inverse(d, e, phi, ctx);
    // Print the private key
    printBN("Private key", d, n);
    // Free the BIGNUM's
    BN_clear_free(p);
    BN_clear_free(q);
    BN_clear_free(e);
    BN_clear_free(n);
    BN_clear_free(res);
    BN_clear_free(phi);
    BN_clear_free(d);
    BN_clear_free(pMinusOne);
    BN_clear_free(qMinusOne);
    return 0;
}
```

```
[11/03/22]seed@VM:/mnt$ gcc -o RSA_Task1 RSA_Task1.c -lcrypto
[11/03/22]seed@VM:/mnt$ ./RSA_Task1
Public keys:
(0D88C3,
E103ABD94892E3E74AFD724BF28E78366D9676BCCC70118BD0AA1968DBB143D1)
Private keys:
(3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB,
E103ABD94892E3E74AFD724BF28E78366D9676BCCC70118BD0AA1968DBB143D1)
```

Task 2. Encrypting a Message

Given code solves the second task of this lab:

```
#include <stdio.h>
#include <openssl/bn.h>
/* printBN is a helper function: receives a char and a BIGNUM.
   It prints the hex of the BIGNUM after the msg.
 */
void printBN(char *msg, BIGNUM *a) {
    char *st = BN_bn2hex(a);
    printf("%s %s\n", msg, st);
    OPENSSL_free(st);
}
int main() {
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM *e = BN_new();
    BIGNUM *C = BN_new();
    BIGNUM *M = BN_new();
```

```
    BIGNUM *n = BN_new();
    //Same hex value of 010001 – part of the public key.
    BN_dec2bn(&e, "65537");
    // Part of the public key.
    BN_hex2bn(&n,
"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    // "A top secret!" in hex.
    BN_hex2bn(&M, "4120746f702073656372657421");

    // Store the result of the encrypted message given values e, n, and M.
    BN_mod_exp(C, M, e, n, ctx);
    // Output the encrypted message.
    printBN("Encryption result:", C);
    // Free the BIGNUM's.
    BN_clear_free(n);
    BN_clear_free(e);
    BN_clear_free(M);
    BN_clear_free(C);
    return 0;
}
```

```
[11/03/22]seed@VM:/mnt$ gcc -o RSA_Task2 RSA_Task2.c -lcrypto
[11/03/22]seed@VM:/mnt$ ./RSA_Task2
Encrypted Message:
 6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC
```

Task 3. Decrypting a Message

Given code solves the third task of this lab:

```
#include <stdio.h>
#include <string.h>
#include <openssl/bn.h>
// hex_to_in receives a char, in hex, and returns the int value
int hex_to_int(char c) {
    int first = c / 16 - 3;
    int second = c % 16;
    int result = first*10 + second;
    if(result > 9) result--;
    return result;
}
/* hex_to_ascii is a helper function for PrintBNHex2ASCII - receives chars.
   It assumes input to be hex and returns their ASCII value as an int.
 */
int hex_to_ascii(char c, char d) {
    int high = hex_to_int(c) * 16;
    int low = hex_to_int(d);
    return high+low;
}
/* printBN is a helper function - receives a char and a BIGNUM.
   It converts the BIGNUM to hex and prints its value after the msg.
 */
void printBN(char *msg, BIGNUM *a) {
```

```
    char *st = BN_bn2hex(a);
    printf("%s %s\n", msg, st);
    OPENSSL_free(st);
}
/* printBNHex2ASCII is a helper function - receives a char and BIGNUM.
   It assumes the BIGNUM is ASCII text in hex form.
   It converts the BIGNUM to hex and then converts the result to ASCII.
   It prints the msg received followed by the ASCII value of the BIGNUM.
 */
void printBNHex2ASCII(char *msg, BIGNUM *a) {
    char *st = BN_bn2hex(a);
    printf("%s", msg);
    int length = strlen(st);
    int i;
    char buf = 0;
    for (i = 0; i < length; i++) {
        if (i % 2 != 0) {
            printf("%c", hex_to_ascii(buf, st[i]));
        } else {
            buf = st[i];
        }
    }
    printf("\n");
    OPENSSL_free(st);
}
int main() {
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM *C = BN_new();
    BIGNUM *M = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *d = BN_new();
    // The public key.
    BN_hex2bn(&n,
"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    // The private key.
    BN_hex2bn(&d,
"74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
    // Cipher Text.
    BN_hex2bn(&C,
"8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");
    // Decrypt the cipher text (C) given n and d, and store the result in M.
    BN_mod_exp(M, C, d, n, ctx);
    // Print the decrypted text as is.
    printBN("Decryption result (in hex):", M);
    // Print the decrypted text after a conversion to ASCII.
    printBNHex2ASCII("Decryption result (in ASCII):", M);
    // Free the BIGNUMs.
    BN_clear_free(n);
    BN_clear_free(d);
    BN_clear_free(M);
```

```
    BN_clear_free(C);
    return 0;
}
[11/03/22]seed@VM:/mnt$ gcc -o RSA_Task3 RSA_Task3.c -lcrypto
[11/03/22]seed@VM:/mnt$ ./RSA_Task3
Decryption result (in hex): 50617373776F72642069732064656573
Decryption result (in ASCII):Password is dees
```

Task 4. Signing a Message

We used the RSA encryption for signature to first get the hexadecimal representations of the two strings:

```
python -c 'print("I owe you $2000".encode("hex"))'
49206f77652079f75202432303030
python -c 'print("I owe you $3000".encode("hex"))'
49206f776520796f45202433303030
```

After running a file with following encryption code, we got the signatures of the two messages (screenshot is below the code):

```
BIGNUM* encrypt(BIGNUM* message, BIGNUM* mod, BIGNUM* pub_key) {
      BN_CTX *ctx = BN_CTX_new();
      BIGNUM* enc = BN_new();
      BN_mod_exp(enc, message, mod, pub_key, ctx);
      BN_CTX_free(ctx);
      return enc;
}
```

```
    Sinature of M1: 80A55421D72345AC199836F60D51DC9594E2BDB4
AE20C804823FB71660DE7B82
    Signature of M2: 04FC9C53ED7BBE4ED4BE2C24B0BDF7184B96290
B4ED4E3959F58E94B1ECEA2EB
```

Task 5. Verifying a Signature

We got the hex string from the message M, which we then compiled and ran using code provided below to verity Alice's signature, which ended up being valid.

```
int verify() {
    // initialize
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM *n = BN_new();
BIGNUM *e = BN_new();
BIGNUM *M = BN_new();
BIGNUM *C = BN_new();
BIGNUM *S = BN_new();
// assign values
BN_hex2bn(&n,"AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F1811611
5");
BN_dec2bn(&e, "65537");
//hex encode for " Launch a missile."
BN_hex2bn(&M, "4c61756e63682061206d697373696c652e");
```

```
BN_hex2bn(&S,
"643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");
BN_mod_exp(C, S, e, n, ctx);
// validate the signature
if (BN_cmp(C, M) == 0) {
printf("Signature is valid!\n");
} else {
printf("Signature is invalid!\n");
}
    //clear data
    BN_clear_free(n);
    BN_clear_free(e);
    BN_clear_free(M);
    BN_clear_free(C);
    BN_clear_free(S);
    return 0;
}
```

Task 6. Manually Verifying an X.509 Certificate

```
#include <stdio.h>
#include <openssl/bn.h>
void printBN(char *msg, BIGNUM *a) {
    char *st = BN_bn2hex(a);
    printf("%s %s\n", msg, st);
    OPENSSL_free(st);
}
int main() {
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM *n = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *S = BN_new();
    BIGNUM *M = BN_new();
    BIGNUM *C = BN_new();
    /* Values gathered from seedsecuritylabs.org via terminal and extracted
       from c1.pem using "openssl x509 -in c1.pem -noout -modulus"
     */
    BN_hex2bn(&n,
"B6E02FC22406C86D045FD7EF0A6406B27D22266516AE42409BCEDC9F9F76073EC330558719B9
4F940E5A941F5556B4C2022AAFD098EE0B40D7C4D03B72C8149EEF90B111A9AED2C8B8433AD90
B0BD5D595F540AFC81DED4D9C5F57B786506899F58ADAD2C7051FA897C9DCA4B182842DC6ADA5
9CC71982A6850F5E44582A378FFD35F10B0827325AF5BB8B9EA4BD51D027E2DD3B4233A30528C
4BB28CC9AAC2B230D78C67BE65E71B74A3E08FB81B71616A19D23124DE5D79208AC75A49CBACD
17B21E4435657F532539D11C0A9A631B199274680A37C2C25248CB395AA2B6E15DC1DDA020B82
1A293266F144A2141C7ED6D9BF2482FF303F5A26892532F5EE3");
    /* Input extracted from c1.pem using "openssl x509 -in c1.pem -text
       noout | grep Exponent"
     */
    BN_dec2bn(&e, "65537");
    /* Input extracted from c0.pem using "openssl x509 -in c0.pem -text –
       noout"
```

```
     */
    BN_hex2bn(&S,
"00f3bbf23fe1d30fc06e10ccc1476668101659dcff1a97b5a34ba8e348cd73f39c14261d08b8
f35c4a8004788d93934e49e5c0e2c15e70d7bd5eab250657badde9c474af54993692fbb20cedd
10b4bae75df35017214b1de8f9e3b760fa5ddff2a54028324c84fbc7ae604484164e07967ae95
ed37b3924c6558650934689ac320db255dd9942fd13a01088861a448a51311763e2cb46e8290f
2697d26ae59ad7d911799ea14d04797fcf4beb1e74bacec6b969661fa12654521b85ff443b4d9
003709c53b6c4d622d630798a714eb2b619a0b2f3515394e2931bc5efb245bfb9f5ff2f062eba
6b98aa41e900dfe0f03c4bd44e5fd4738307b729320ceaa78a5");
    // Hash for the body of the certificate.
    BN_hex2bn(&M,
"0001FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFF003031300D0609608648016503040201050004200640f8d13c078
9ff0ed5437cf4bc9f2827d52146dddff38aefc2c17747d45f28");
    BN_mod_exp(C, S, e, n, ctx);
    if (BN_cmp(C, M) == 0) {
        printf("Signature is has been proven valid\n");
    } else {
        printf("Signature has failed to be proved valid\n");
    }
    // Free BIGNUM’s.
    BN_clear_free(n);
    BN_clear_free(e);
    BN_clear_free(M);
    BN_clear_free(C);
    BN_clear_free(S);
    return 0;
}
```

We first downloaded a certificate from a real web server:

```
[11/03/22]seed@VM:/mnt$ openssl s_client -connect seedsecuritylabs.org:443 -showcerts > seed.txt
depth=2 C = US, O = Internet Security Research Group, CN = ISRG Root X1
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=1 C = US, O = Let's Encrypt, CN = R3
verify return:1
depth=0 CN = seedsecuritylabs.org
verify return:1
```

We saved the certification (server's CA): as `co.pem`:

```
[11/03/22]seed@VM:/mnt$ openssl x509 -in c0.pem -out signature
[11/03/22]seed@VM:/mnt$ cat signature | tr -d '[:space:]:'
-----BEGINCERTIFICATE-----MIIHMDCCBhigAwIBAgIQAkk+B/qeN1otu8YdlEMPzzANBgkqhkiG9w0BAQsFADBwMQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRk
wFwYDVQQLExB3d3cuZGlnaWNlcnQuY29tMS8wLQYDVQQDEyZEaWdpQ2VydCBTSEEyIEhpZ2ggQXNzdXJhbmNlIFNlcnZlciBDQTAeFw0yMDA1MDYwMDAwMDBaFw0yMjA0MTQxMjAwMDBa
MGoxCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpDYWxpZm9ybmlhMRYwFAYDVQQHEw1TYW4gRnJhbmNpc2NvMRUwEwYDVQQKEwxHaXRIdWIsIEluYy4xFzAVBgNVBAMTDnd3dy5naXRodWIuY
29tMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAsj496jJ99veEXO7WdxGQZ7idtCnDcjZqQeDiy6057SwXj9yDUVnqhwo/yII8+y6Jpk3g75LpPpYNjiOwYp/JkpWbpBAd1F
WlvXJo/eZS+TwuIYb7JSc2H3NDDKt2VV5SSKQdXOkDNqq7BisOFp2/TYwCMZboLufwRR5fKxL0nTKIOCwpnH8k//UdWpvTgIixDGLYQCwHt0fYEo49jFeDaKD4WMBPq6Tx1iKWBhw3HVc
/OyvI3yjRAx4Anf/DCSt9YTW6f/ND4O/fOowcfW5T7zii1Kw0yw+ulBrE/xe6taVhL+QR0MXNkQV2iHNN85swidwMtcdGI8g3fYL48bSRywIDAQABo4IDyjCCA8YwHwYDVR0jBBgwFoAU
UWj/kK8CB3U8zNllZGKiErhZcjswHQYDVR0OBBYEFIygCmlH3IkysE3GEUViXxovlk46MHsGA1UdEQR0MHKCDnd3dy5naXRodWIuY29tggwqLmdpdGh1Yi5jb22CCmdpdGh1Yi5jb22CC
youZ2l0aHViLmlvgglnaXRodWIuaW+CFyouZ2l0aHVidXNlcmNvbnRlbnQuY29tghVnaXRodWJ1c2VyY29udGVudC5jb20wDgYDVR0PAQH/BAQDAgWgMB0GA1UdJQQWMBQGCCsGAQUFBw
MBBggrBgEFBQcDAjB1BgNVHR8EbjBsMDSgMqAwhi5odHRwOi8vY3JsMy5kaWdpY2VydC5jb20vc2hhMi1oYS1zZXJ2ZXItZzYuY3JsMDSgMqAwhi5odHRwOi8vY3JsNC5kaWdpY2VydC5
jb20vc2hhMi1oYS1zZXJ2ZXItZzYuY3JsMEwGA1UdIARFMEMwNwYJYIZIAYb9bAEBMCowKAYIKwYBBQUHAgEWHGh0dHBzOi8vd3d3LmRpZ2ljZXJ0LmNvbS9DUFMwCAYGZ4EMAQICMIGD
BggrBgEFBQcBAQR3MHUwJAYIKwYBBQUHMAGGGGh0dHA6Ly9vY3NwLmRpZ2ljZXJ0LmNvbTBNBggrBgEFBQcwAoZBaHR0cDovL2NhY2VydHMuZGlnaWNlcnQuY29tL0RpZ2lDZXJ0U0hBM
khpZ2hBc3N1cmFuY2VTZXJ2ZXJDQS5jcnQwDAYDVR0TAQH/BAIwADCCAX0GCisGAQQB1nkCBAIEggFtBIIBaQFnAHYARqVV63X6kSAwtaKJafTzfREsQXS+/Um4havy/HD+bUcAAAFx6y
8fFgAABAMARzBFAiEA59y6w9oaoAoM2fvFq6KofYWRh0xRm4VEEaMHBtsBYUgCIBZxJhjA7SGWUlo57YslG8u6clHngDNvoTNVw1HQtTr3AHUAIkVFB1lVJFaWP6Ev8fdthuAjJmOtwEt
/XcaDXG7iDwIAAAFx6y8evwAABAMARjBEAiBmEjiioTbc1//hCInYIX6O8hph5oLRVGCTxrTBfSRT2wIgZz7x3ZNIKQkWPKOFaaW3AxcB0DzhFsD6gxhkbl1p0AgAdgBRo7D1/QF5nFZt
uDd4jwykeswbJ8v3nohCmg3+1IsF5QAAAXHrLx8JAAAEAwBHMEUCIBQ/6El+TCCtWuop7IderN0+byn5sDreTu+Xz3GiY8cLAiEA7S83HxFFdQhQqpjjbWbIVBA88Nn/riaf5Jb8h3oJV
8cwDQYJKoZIhvcNAQELBQADggEBAADzu/I/4dMPwG4QzMFHZmgQFlnc/xqXtaNLqONIzXPznBQmHQi481xKgAR4jZOTTknlwOLBXnDXvV6rJQZXut3pxHSvVJk2kvuyDO3RC0uudd81AX
IUsd6Pnjt2D6Xd/ypUAoMkyE+8euYESEFk4HlnrpXtN7OSTGVYZQk0aJrDINslXdmUL9E6AQiIYaRIpRMRdj4stG6CkPJpfSauWa19kReZ6hTQR5f89L6x50us7GuWlmH6EmVFIbhf9EO
02QA3CcU7bE1iLWMHmKcU6ythmgsvNRU5TikxvF77JFv7n1/y8GLrprmKpB6QDf4PA8S9ROX9Rzgwe3KTIM6qeKU=-----ENDCERTIFICATE-----[11/03/22][11/03/22]seed@VM:
```

Then, we extracted the modulus `n` from an `x509` certificate, printing out all the attributes of the certificate to find the public key `e` – `exponent`:

```
[11/03/22]seed@VM:/mnt$ openssl x509 -in c1.pem -noout -modulus
Modulus=B6E02FC22406C86D045FD7EF0A6406B27D22266516AE42409BCEDC9F9F76073EC330558719B94F940E5A941F5556B4C
2022AAFD098EE0B40D7C4D03B72C8149EEF90B111A9AED2C8B8433AD90B0BD5D595F540AFC81DED4D9C5F57B786506899F58ADA
D2C7051FA897C9DCA4B182842DC6ADA59CC71982A6850F5E44582A378FFD35F10B0827325AF5BB8B9EA4BD51D027E2DD3B4233A
30528C4BB28CC9AAC2B230D78C67BE65E71B74A3E08FB81B71616A19D23124DE5D79208AC75A49CBACD17B21E4435657F532539
D11C0A9A631B199274680A37C2C25248CB395AA2B6E15DC1DDA020B821A293266F144A2141C7ED6D9BF2482FF303F5A26892532
F5EE3
[11/03/22]seed@VM:/mnt$ openssl x509 -in c1.pem -text -noout | grep Exponent
                Exponent: 65537 (0x10001)
```

To extract the signature from the server's certificate, we found the location of the last "Signature Algorithm", which is the hex representation of the string – body of the signature:

```
[11/03/22]seed@VM:/mnt$ openssl x509 -in c1.pem -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            04:e1:e7:a4:dc:5c:f2:f3:6d:c0:2b:42:b8:5d:15:9f
        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 High Assurance Server CA
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:b6:e0:2f:c2:24:06:c8:6d:04:5f:d7:ef:0a:64:
                    06:b2:7d:22:26:65:16:ae:42:40:9b:ce:dc:9f:9f:
                    76:07:3e:c3:30:55:87:19:b9:4f:94:0e:5a:94:1f:
                    55:56:b4:c2:02:2a:af:d0:98:ee:0b:40:d7:c4:d0:
                    3b:72:c8:14:9e:ef:90:b1:11:a9:ae:d2:c8:b8:43:
                    3a:d9:0b:0b:d5:d5:95:f5:40:af:c8:1d:ed:4d:9c:
                    5f:57:b7:86:50:68:99:f5:8a:da:d2:c7:05:1f:a8:
                    97:c9:dc:a4:b1:82:84:2d:c6:ad:a5:9c:c7:19:82:
                    a6:85:0f:5e:44:58:2a:37:8f:fd:35:f1:0b:08:27:
                    32:5a:f5:bb:8b:9e:a4:bd:51:d0:27:e2:dd:3b:42:
                    33:a3:05:28:c4:bb:28:cc:9a:ac:2b:23:0d:78:c6:
                    7b:e6:5e:71:b7:4a:3e:08:fb:81:b7:16:16:a1:9d:
                    23:12:4d:e5:d7:92:08:ac:75:a4:9c:ba:cd:17:b2:
                    1e:44:35:65:7f:53:25:39:d1:1c:0a:9a:63:1b:19:
                    92:74:68:0a:37:c2:c2:52:48:cb:39:5a:a2:b6:e1:
                    5d:c1:dd:a0:20:b8:21:a2:93:26:6f:14:4a:21:41:
                    c7:ed:6d:9b:f2:48:2f:f3:03:f5:a2:68:92:53:2f:
                    5e:e3
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:TRUE, pathlen:0
            X509v3 Key Usage: critical
                Digital Signature, Certificate Sign, CRL Sign
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
            Authority Information Access:
```

To extract the body of the server's certificate, we parsed the server's certificate:

```
[11/03/22]seed@VM:/mnt$ openssl asn1parse -i -in c0.pem
    0:d=0  hl=4 l=1840 cons: SEQUENCE
    4:d=1  hl=4 l=1560 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           :02493E07FA9E375A2DBBC61D94430FCF
   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= 112 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=  47 cons:    SET
  113:d=4  hl=2 l=  45 cons:     SEQUENCE
  115:d=5  hl=2 l=   3 prim:      OBJECT            :commonName
  120:d=5  hl=2 l=  38 prim:      PRINTABLESTRING   :DigiCert SHA2 High Assurance Server CA
  160:d=2  hl=2 l=  30 cons:   SEQUENCE
  162:d=3  hl=2 l=  13 prim:    UTCTIME           :200506000000Z
  177:d=3  hl=2 l=  13 prim:    UTCTIME           :220414120000Z
  192:d=2  hl=2 l= 106 cons:   SEQUENCE
  194:d=3  hl=2 l=  11 cons:    SET
  196:d=4  hl=2 l=   9 cons:     SEQUENCE
  198:d=5  hl=2 l=   3 prim:      OBJECT            :countryName
  203:d=5  hl=2 l=   2 prim:      PRINTABLESTRING   :US
  207:d=3  hl=2 l=  19 cons:    SET
  209:d=4  hl=2 l=  17 cons:     SEQUENCE
  211:d=5  hl=2 l=   3 prim:      OBJECT            :stateOrProvinceName
  216:d=5  hl=2 l=  10 prim:      PRINTABLESTRING   :California
  228:d=3  hl=2 l=  22 cons:    SET
  230:d=4  hl=2 l=  20 cons:     SEQUENCE
  232:d=5  hl=2 l=   3 prim:      OBJECT            :localityName
  237:d=5  hl=2 l=  13 prim:      PRINTABLESTRING   :San Francisco
```

```
[11/03/22]seed@VM:/mnt$ openssl asn1parse -i -in c0.pem -strparse 4 -out c0_body.bin -noout
[11/03/22]seed@VM:/mnt$ sha256sum c0_body.bin
0640f8d13c0789ff0ed5437cf4bc9f2827d52146dddff38aefc2c17747d45f28  c0_body.bin
```

Using the values obtained earlier, we ran the updated program, validating the signature:

```
[11/03/22]seed@VM:/mnt$ gcc -o RSA_Task6 RSA_Task6.c -lcrypto
[11/03/22]seed@VM:/mnt$ ./RSA_Task6
Signature is has been proven valid
```