RSA LAB

Maryam Fahmi (501096276)
Sidra Musheer (501122840)
Veezish Ahmad (501080184)

CPS633 Sec. 10 - Group 61
Computer Security
Toronto Metropolitan University

Task 1

Code for getting the private key

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

void printBN(char *msg, BIGNUM * a) {
   /* Convert the BIGNUM to a decimal string and print */
   char *number_str = BN_bn2hex(a);
   printf("%s %s\n", msg, number_str);
   OPENSSL_free(number_str);
}

int main() {
   // Initialize the context
   BN_CTX *ctx = BN_CTX_new();

   // Step 1: Initialize p, q, and e with the provided hex values
   BIGNUM *p = BN_new();
   BIGNUM *q = BN_new();
   BIGNUM *e = BN_new();
   BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
   BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
   BN_hex2bn(&e, "0D88C3");

   // Step 2: Compute n = p * q
   BIGNUM *n = BN_new();
   BN_mul(n, p, q, ctx);
   printBN("n = p * q = ", n);

   // Step 3: Compute phi(n) = (p - 1) * (q - 1)
   BIGNUM *p_minus1 = BN_new();
   BIGNUM *q_minus1 = BN_new();
   BIGNUM *phi_n = BN_new();
   BN_sub(p_minus1, p, BN_value_one()); // p - 1
   BN_sub(q_minus1, q, BN_value_one()); // q - 1
   BN_mul(phi_n, p_minus1, q_minus1, ctx);
   printBN("phi(n) = (p - 1) * (q - 1) = ", phi_n);

   // Step 4: Compute d, the modular inverse of e mod phi(n)
   BIGNUM *d = BN_new();
   BN_mod_inverse(d, e, phi_n, ctx);
   printBN("d = e^(-1) mod phi(n) = ", d);

   // Cleanup
   BN_free(p);
   BN_free(q);
   BN_free(e);
   BN_free(n);
   BN_free(p_minus1);
   BN_free(q_minus1);
   BN_free(phi_n);
   BN_free(d);
```

```
    BN_CTX_free(ctx);

    return 0;
}
```

Running it:

```
[10/27/24]seed@VM:~/.../Labsetup$ gcc task1.c -o rsa_task1 -lcrypto
[10/27/24]seed@VM:~/.../Labsetup$ ./rsa_task1
n = p * q =  E103ABD94892E3E74AFD724BF28E78366D9676BCCC70118BD0AA1968DBB143D1
phi(n) = (p - 1) * (q - 1) =  E103ABD94892E3E74AFD724BF28E78348D52298BD687C44DEB3A81065A7981A4
d = e^(-1) mod phi(n) =  3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB
```

Task 2

Code for encryption

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

void printBN(char *msg, BIGNUM * a) {
    /* Convert the BIGNUM to a hexadecimal string and print */
    char *number_str = BN_bn2hex(a);
    printf("%s %s\n", msg, number_str);
    OPENSSL_free(number_str);
}

int main() {
    // Initialize the context
    BN_CTX *ctx = BN_CTX_new();

    // Step 1: Initialize n and e
    BIGNUM *n = BN_new();
    BIGNUM *e = BN_new();
    BN_hex2bn(&n,
"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    BN_hex2bn(&e, "010001");

    // Step 2: Convert the message "A top secret!" to hexadecimal and then to BIGNUM
    BIGNUM *M = BN_new();
    BN_hex2bn(&M, "4120746F702073656372657421");  // "A top secret!" in hex

    // Step 3: Encrypt the message using C = M^e mod n
    BIGNUM *C = BN_new();
    BN_mod_exp(C, M, e, n, ctx);
    printBN("Encrypted message C = ", C);

    // Cleanup
    BN_free(n);
    BN_free(e);
    BN_free(M);
    BN_free(C);
    BN_CTX_free(ctx);

    return 0;
}
```

Running it

```
[10/27/24]seed@VM:~/.../Labsetup$ ./rsa_task2
Encrypted message C =  6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC
```

Task 3

Code for decryption

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

void printBN(char *msg, BIGNUM * a) {
   /* Convert the BIGNUM to a hexadecimal string and print */
   char *number_str = BN_bn2hex(a);
   printf("%s %s\n", msg, number_str);
   OPENSSL_free(number_str);
}

int main() {
   // Initialize the context
   BN_CTX *ctx = BN_CTX_new();

   // Step 1: Initialize n and d with the values from Task 2
   BIGNUM *n = BN_new();
   BIGNUM *d = BN_new();
   BN_hex2bn(&n,
"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
   BN_hex2bn(&d,
"74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

   // Step 2: Initialize the ciphertext C
   BIGNUM *C = BN_new();
   BN_hex2bn(&C,
"8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");

   // Step 3: Decrypt the ciphertext using M = C^d mod n
   BIGNUM *M = BN_new();
   BN_mod_exp(M, C, d, n, ctx);
   printBN("Decrypted message M = ", M);

   // Cleanup
   BN_free(n);
   BN_free(d);
   BN_free(C);
   BN_free(M);
   BN_CTX_free(ctx);

   return 0;
}
```

Running it:

```
[10/27/24]seed@VM:~/.../Labsetup$ gcc rsa_task3.c -o rsa_task3 -lcrypto
[10/27/24]seed@VM:~/.../Labsetup$ ./rsa_task3
Decrypted message M =  50617373776F72642069732064656573
[10/27/24]seed@VM:~/.../Labsetup$ python3 -c 'print(bytes.fromhex("50617373776F72642069732064656573").decode("utf-8"))'
Password is dees
```

Task 4

Code for signing a message

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

void printBN(char *msg, BIGNUM * a) {
   /* Convert the BIGNUM to a hexadecimal string and print */
   char *number_str = BN_bn2hex(a);
   printf("%s %s\n", msg, number_str);
   OPENSSL_free(number_str);
}

int main() {
   // Initialize the context
   BN_CTX *ctx = BN_CTX_new();

   // Step 1: Initialize n and d with the values from Task 2
   BIGNUM *n = BN_new();
   BIGNUM *d = BN_new();
   BN_hex2bn(&n,
"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
   BN_hex2bn(&d,
"74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");  //
Replace with actual value of d

   // Step 2: Convert the original message "I owe you $2000." to hexadecimal and then to BIGNUM
   BIGNUM *M1 = BN_new();
   BN_hex2bn(&M1, "49206F776520796F752024323030302E");  // Hex for "I owe you $2000."

   // Step 3: Sign the original message M1 using S1 = M1^d mod n
   BIGNUM *S1 = BN_new();
   BN_mod_exp(S1, M1, d, n, ctx);
   printBN("Signature for original message (I owe you $2000) S1 = ", S1);

   // Step 4: Convert the modified message "I owe you $3000." to hexadecimal and then to
BIGNUM
   BIGNUM *M2 = BN_new();
   BN_hex2bn(&M2, "49206F776520796F752024333030302E");  // Hex for "I owe you $3000."

   // Step 5: Sign the modified message M2 using S2 = M2^d mod n
   BIGNUM *S2 = BN_new();
   BN_mod_exp(S2, M2, d, n, ctx);
   printBN("Signature for modified message (I owe you $3000) S2 = ", S2);

   // Cleanup
   BN_free(n);
   BN_free(d);
   BN_free(M1);
   BN_free(S1);
   BN_free(M2);
   BN_free(S2);
   BN_CTX_free(ctx);
```

```
    return 0;
}
```

Running it for both:

```
[10/27/24]seed@VM:~/.../Labsetup$ gcc rsa_task4.c -o rsa_task4 -lcrypto
[10/27/24]seed@VM:~/.../Labsetup$ ./rsa_task4
Signature for original message (I owe you $2000) S1 =  55A4E7F17F04CCFE2766E1EB32ADDBA890BBE92A6FBE2D785ED6E73CCB35E4CB
Signature for modified message (I owe you $3000) S2 =  BCC20FB7568E5D48E434C387C06A6025E90D29D848AF9C3EBAC0135D99305822
```

Observation

- **Comparison of Signatures**:
  - S1 and S2 are different because even a minor change in the message content (like changing "$2000" to "$3000") produces a completely different signature.
  - This demonstrates the **sensitivity of RSA signatures to message content**—even a single-character change results in a unique signature, which is essential for integrity verification.

This approach ensures that any tampering with the message will make the original signature invalid, as the signature is tightly bound to the specific message content.

Task 5

Code for verifying a signature

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

void printBN(char *msg, BIGNUM * a) {
    /* Convert the BIGNUM to a hexadecimal string and print */
    char *number_str = BN_bn2hex(a);
    printf("%s %s\n", msg, number_str);
    OPENSSL_free(number_str);
}

int main() {
    // Initialize the context
    BN_CTX *ctx = BN_CTX_new();

    // Step 1: Initialize n and e for the public key
    BIGNUM *n = BN_new();
    BIGNUM *e = BN_new();
    BN_hex2bn(&n,
"AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");
    BN_hex2bn(&e, "010001");

    // Step 2: Initialize the original message M and the provided signature S
    BIGNUM *M = BN_new();
    BIGNUM *S = BN_new();
    BN_hex2bn(&M, "4C61756E63682061206D697373696C652E"); // Hex for "Launch a missile."
    BN_hex2bn(&S,
"643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");

    // Step 3: Verify the signature by computing M' = S^e mod n
```

```
BIGNUM *M_prime = BN_new();
BN_mod_exp(M_prime, S, e, n, ctx);
printBN("Computed message from signature M' = ", M_prime);

// Check if M and M' are the same
if (BN_cmp(M, M_prime) == 0) {
    printf("Signature is valid for the original message.\n");
} else {
    printf("Signature is invalid for the original message.\n");
}

// Step 4: Corrupt the signature slightly (e.g., change the last byte) and re-verify
BIGNUM *S_corrupt = BN_dup(S);  // Duplicate S to corrupt
BN_set_word(S_corrupt, BN_get_word(S_corrupt) ^ 0x01);  // Flip the last bit

// Verify the corrupted signature
BN_mod_exp(M_prime, S_corrupt, e, n, ctx);
printBN("Computed message from corrupted signature M' = ", M_prime);

// Check if the corrupted signature is valid
if (BN_cmp(M, M_prime) == 0) {
    printf("Corrupted signature is (unexpectedly) valid.\n");
} else {
    printf("Corrupted signature is invalid, as expected.\n");
}

// Cleanup
BN_free(n);
BN_free(e);
BN_free(M);
BN_free(S);
BN_free(M_prime);
BN_free(S_corrupt);
BN_CTX_free(ctx);

return 0; }
```

```
[10/29/24]seed@VM:~/.../Labsetup$ gcc rsa_task5.c -o rsa_task5 -lcrypto
[10/29/24]seed@VM:~/.../Labsetup$ ./rsa_task5
Computed message from signature M' =  4C61756E63682061206D697373696C652E
Signature is valid for the original message.
Computed message from corrupted signature M' =  5C7F948B174A367CE59D0937829B8453873D749B1CE700113A538B09187E92E0
Corrupted signature is invalid, as_expected.
```

Explanation of Observed Results

- **Original Signature**: If the computed message M’ matches M, the signature is valid.
- **Corrupted Signature**: Any modification to the signature should result in an invalid verification, showing that RSA signatures are highly sensitive to changes in the signed message or signature itself.

This verifies that the signature is uniquely tied to both the private key and the specific message content, ensuring message integrity.

Task 6

Step 1:

Purpose: Obtain the certificate chain for the target website. This chain includes the **server certificate** and possibly **intermediate CA certificates**.

```
[10/29/24]seed@VM:~/.../Labsetup$ openssl s_client -connect google.com:443 -showcerts
CONNECTED(00000003)
depth=2 C = US, O = Google Trust Services LLC, CN = GTS Root R1
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=1 C = US, O = Google Trust Services, CN = WR2
verify return:1
depth=0 CN = *.google.com
verify return:1
---
```


```
-----BEGIN CERTIFICATE-----
MIIFCzCCAvOgAwIBAgIQf/AFoHxM3tEArZ1mpRB7mDANBgkqhkiG9w0BAQsFADBH
MQswCQYDVQQGEwJVUzEiMCAGA1UEChMZR29vZ2xlIFRydXN0IFNlcnZpY2VzIExM
QzEUMBIGA1UEAxMLR1RTIFJvb3QgUjEwHhcNMjMxMjEzMDkwMDAwWhcNMjkwMjIw
MTQwMDAwWjA7MQswCQYDVQQGEwJVUzEeMBwGA1UEChMVR29vZ2xlIFRydXN0IFNl
cnZpY2VzMQwwCgYDVQQDEwNXUjIwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEK
AoIBAQCp/5x/RR5wqFOfytnlDd5GV1d9vI+aWqxG8YSau5HbyfsvAfuSCQAWXqAc
+MGr+XgvSszYhaLYWTwO0xj7sfUkDSbutltkdnwUxy96zqhMt/TZCPzfhyM1IKji
aeKMTj+xWfpgoh6zySBTGYLKNlNtYE3pAJH8do1cCA8Kwtzxc2vFE24KT3rC8gIc
LrRjg9ox9i11MLL7q8Ju26nADrn5Z9TDJVd06wW06Y613ijNzHoU5HEDy01hLmFX
xRmpC5iEGuh5KdmyjS//V2pm4M6rlagplmNwEmceOuHbsCFx13ye/aoXbv4r+zgX
FNFmp6+atXDMyGOBOozAKql2N87jAgMBAAGjgf4wgfswDgYDVR0PAQH/BAQDAgGG
MB0GA1UdJQQWMBQGCCsGAQUFBwMBBggrBgEFBQcDAjASBgNVHRMBAf8ECDAGAQH/
AgEAMB0GA1UdDgQWBBTeGx7teRXUPjckwyG77DQ5bUKyMDAfBgNVHSMEGDAWgBTk
rysmcRorSCeFL1JmLO/wiRNxPjA0BggrBgEFBQcBAQQoMCYwJAYIKwYBBQUHMAKG
GGh0dHA6Ly9pLnBraS5nb29nL3IxLmNydDArBgNVHR8EJDAiMCCgHqAchhpodHRw
Oi8vYy5wa2kuZ29vZy9yL3IxLmNybDATBgNVHSAEDDAKMAgGBmeBDAECATANBgkq
hkiG9w0BAQsFAAOCAgEARXWL5R87RBOWGqtY8TXJbz3S0DNKhjO6V1FP7sQ02hYS
TL8Tnw3UVOlIecAwPJQl8hr0ujKUtjNyC4XuCRElNJThb0Lbgpt7fyqaqf9/qdLe
SiDLs/sDA7j4BwXaWZIvGEaYzq9yviQmsR4ATb0IrZNBRAq7x9UBhb+TV+PfdBJT
DhEl05vc3ssnbrPCuTNiOcLgNeFbpwkuGcuRKnZc8d/KI4RApW//mkHgte8y0YWu
ryUJ8GLFbsLIbjL9uNrizkqRSvOFVU6xddZIMy9vhNkSXJ/UcZhjJY1pXAprffJB
vei7j+Qi151lRehMCofa6WBmiA4fx+FOVsV2/7R6V2nyAiIJJkEd2nSi5SnzxJrl
Xdaqev3htytmOPvoKWa676ATL/hzfvDaQBEcXd2Ppvy+275W+DKcH0FBbX62xevG
iza3F4ydzxl6NJ8hk8R+dDXSqv1MbRT1ybB5W0k8878XSOjvmiYTDIfyc9acxVJr
```

```
Y/cykHipa+te1pOhv7wYPYtZ9orGBV5SGOJm4NrB3K1aJar0RfzxC3ikr7Dyc6Qw
qDTBU39CluVIQeuQRgwG3MuSxl7zRERDRilGoKb8uY45JzmxWuKxrfwT/478JuHU
/oTxUFqOl2stKnn7QGTq8z29W+GgBLCXSBxC9epaHM0myFH/FJlniXJfHeytWt0=
-----END CERTIFICATE-----
 2 s:C = US, O = Google Trust Services LLC, CN = GTS Root R1
   i:C = BE, O = GlobalSign nv-sa, OU = Root CA, CN = GlobalSign Root CA
-----BEGIN CERTIFICATE-----
MIIFYjCCBEqgAwIBAgIQd70NbNs2+RrqIQ/E8FjTDTANBgkqhkiG9w0BAQsFADBX
MQswCQYDVQQGEwJCRTEZMBcGA1UEChMQR2xvYmFsU2lnbiBudi1zYTEQMA4GA1UE
CxMHUm9vdCBDQTEbMBkGA1UEAxMSR2xvYmFsU2lnbiBSb290IENBMB4XDTIwMDYx
OTAwMDA0MloXDTI4MDEyODAwMDA0MlowRzELMAkGA1UEBhMCVVMxIjAgBgNVBAoT
GUdvb2dsZSBUcnVzdCBTZXJ2aWNlcyBMTEMxFDASBgNVBAMTC0dUUyBSb290IFIx
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAthECix7joXebO9y/lD63
ladAPKH9gvl9MgaCcfb2jH/76Nu8ai6Xl6OMS/kr9rH5zoQdsfnFl97vufKj6bwS
iV6nqlKr+CMny6SxnGPb15l+8Ape62im9MZaRw1NEDPjTrETo8gYbEvs/AmQ351k
KSUjB6G00j0uYODP0gmHu81I8E3CwnqIiru6z1kZ1q+PsAewnjHxgsHA3y6mbWwZ
DrXYfiYaRQM9sHmklCitD38m5agI/pboPGiUU+6DOogrFZYJsuB6jC511pzrp1Zk
j5ZPaK49l8KEj8C8QMALXL32h7M1bKwYUH+E4EzNktMg6TO8UpmvMrUpsyUqtEj5
cuHKZPfmghCN6J3Cioj6OGaK/GP5Afl4/Xtcd/p2h/rs37EOeZVXtL0m79YB0esW
CruOC7XFxYpVq9Os6pFLKcwZpDIlTirxZUTQAs6qzkm06p98g7BAe+dDq6dso499
iYH6TKX/1Y7DzkvgtdizjkXPdsDtQCv9Uw+wp9U7DbGKogPeMa3Md+pvez7W35Ei
Eua++tgy/BBjFFFy3l3WFpO9KWgz7zpm7AeKJt8T11dleCfeXkkUAKIAf5qoIbap
sZWwpbkNFhHax2xIPEDgfg1azVY80ZcFuctL7TlLnMQ/0lUTbiSw1nH69MG6zO0b
9f6BQdgAmD06yK56mDcYBZUCAwEAAaOCATgwggE0MA4GA1UdDwEB/wQEAwIBhjAP
BgNVHRMBAf8EBTADAQH/MB0GA1UdDgQWBBTkrysmcRorSCeFL1JmLO/wiRNxPjAf
BgNVHSMEGDAWgBRge2YaRQ2XyolQL30EzTSo//z9SzBgBggrBgEFBQcBAQRUMFIw
JQYIKwYBBQUHMAGGGWh0dHA6Ly9vY3NwLnBraS5nb29nL2dzcjEwKQYIKwYBBQUH
MAKGHWh0dHA6Ly9wa2kuZ29vZy9nc3IxL2dzcjEuY3J0MDIGA1UdHwQrMCkwJ6Al
oCOGIWh0dHA6Ly9jcmwucGtpLmdvb2cvZ3NyMS9nc3IxLmNybDA7BgNVHSAENDAy
MAgGBmeBDAECATAIBgZngQwBAgIwDQYLKwYBBAHWeQIFAwIwDQYLKwYBBAHWeQIF
AwMwDQYJKoZIhvcNAQELBQADggEBADSkHrEoo9C0dhemMXoh6dFSPsjbdBZBiLg9
NR3t5P+T4Vxfq7vqfM/b5A3Ri1fyJm9bvhdGaJQ3b2t6yMAYN/olUazsaL+yyEn9
WprKASOshIArAoyZl+tJaox118fessmXn1hIVw41oeQa1v1vg4Fv74zPl6/AhSrw
9U5pCZEt4Wi4wStz6dTZ/CLANx8LZh1J7QJVj2fhMtfTJr9w4z30Z209fOU0iOMy
+qduBmpvvYuR7hZL6Dupszfnw0Skfths18dG9ZKb59UhvmaSGZRVbNQpsg3BZlvi
d0lIKO2d1xozclOzgjXPYovJJIultzkMu34qQb9Sz/yilrbCgj8=
-----END CERTIFICATE-----
```


Step 2:

Purpose: To verify the signature on the server's certificate, we need the **public key** of the issuer (intermediate CA). This public key consists of the modulus n and exponent e.

Exponent = 65537

```
[10/29/24]seed@VM:~/.../Labsetup$ openssl x509 -in c1.pem -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            7f:f0:05:a0:7c:4c:de:d1:00:ad:9d:66:a5:10:7b:98
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = US, O = Google Trust Services LLC, CN = GTS Root R1
        Validity
            Not Before: Dec 13 09:00:00 2023 GMT
            Not After : Feb 20 14:00:00 2029 GMT
        Subject: C = US, O = Google Trust Services, CN = WR2
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:a9:ff:9c:7f:45:1e:70:a8:53:9f:ca:d9:e5:0d:
                    de:46:57:57:7d:bc:8f:9a:5a:ac:46:f1:84:9a:bb:
                    91:db:c9:fb:2f:01:fb:92:09:00:16:5e:a0:1c:f8:
                    c1:ab:f9:78:2f:4a:cc:d8:85:a2:d8:59:3c:0e:d3:
                    18:fb:b1:f5:24:0d:26:ee:b6:5b:64:76:7c:14:c7:
                    2f:7a:ce:a8:4c:b7:f4:d9:08:fc:df:87:23:35:20:
                    a8:e2:69:e2:8c:4e:3f:b1:59:fa:60:a2:1e:b3:c9:
                    20:53:19:82:ca:36:53:6d:60:4d:e9:00:91:fc:76:
                    8d:5c:08:0f:0a:c2:dc:f1:73:6b:c5:13:6e:0a:4f:
```

Modulus =
A9FF9C7F451E70A8539FCAD9E50DDE4657577DBC8F9A5AAC46F1849ABB91DBC9FB2F01F
B920900165EA1CF8C1ABF9782F4ACCD885A2D8593COED318FBB1F5240D26EEB65B64767C
14C72F7ACEA84CB7F4D908FCDF87233520A8E269E28C4E3FB159FA60A1EB3C920531982CA
36536D604DE90091FC768D5C080F0AC2

```
[10/29/24]seed@VM:~/.../Labsetup$ openssl x509 -in c1.pem -noout -modulus
Modulus=A9FF9C7F451E70A8539FCAD9E50DDE4657577DBC8F9A5AAC46F1849ABB91DBC9FB2F01FB920900165EA01CF8C1ABF9782F4ACCD885A2D8593C0ED318FBB1F5240D26E
EB65B64767C14C72F7ACEA84CB7F4D908FCDF87233520A8E269E28C4E3FB159FA60A21EB3C920531982CA36536D604DE90091FC768D5C080F0AC2DCF1736BC5136E0A4F7AC2F2
021C2EB46383DA31F62D7530B2FBABC26EDBA9C00EB9F967D4C3255774EB05B4E98EB5DE28CDCC7A14E47103CB4D612E6157C519A90B98841AE87929D9B28D2FFF576A66E0CEA
B95A82996637012671E3AE1DBB02171D77C9EFDAA176EFE2BFB381714D166A7AF9AB570CCC863813A8CC02AA97637CEE3
```

Step 3:

Purpose: The signature is a cryptographic value that certifies the authenticity of the certificate. We'll compare this with the hash of the certificate body to verify it.

Signature Algorithm: sha384WithRSAEncryption
38:96:0a:ee:3d: b4:96:le:5f:ef:9d:9c: 0b:33:9f:2b:e0:ca:
fd:d2:8e:0a:1f:41:74:a5:7c:aa:84:d4:e5:f2:1e:e6:37:52:
32:9c:0b:d1:61:1d:bf:28:c1:b6:44:29:35:75:77:98: b2:7c:
d9: bd: 74: ac:8a:68:e3: a9:31:09:29:01:60:73:e3:47:7c:53: a8:90:4a:27:ef:4b:
d7:9f:93:e7:82:36:ce:9a:68:0c:82:e7:
cf:d4:10:16:6f:5f:0e:99:5c:f6:1f:71:7d:ef:ef:7b:2f:7e:
ea:36:d6:97:70:0b:15:ee:d7:5c:56:6a:33:a5: e3:49:38:0c: b8:7d: fb:8d:85: a4:b1:59:5e:
f4:6a:el:dd:al: f6:64:44:ae: e6:51:83:21:66: c6:11:3e:f3: ce: 47:ee:9c:28:1f:25: da:ff:
ac:66:95:dd:35:0f:5c:ef:20:2c:62: fd:91:ba: a9: cc:fc:5a: 9c:93:81:83:29:97:4a:7c:5a:72: b4:39:
d0:b7:77: cb:79:fd:
69:3a:92:37:ed:6e:38:65:46:7e:e9:60:bd:79:88:97:5f:38:
12: f4:ee:af:5b:82:c8:86: d5:e1:99:6d:8c:04: f2:76: ba:49: f6:6e:e9:6d:le:5f:a0:ef:27:82:76:40: f8:
a6: d3:58:5c:0f:
2c:42: da:42:c6:7b:88:34:c7:c1:d8:45:9b:c1:3e:c5:61:1d:
d9:63:50:49:f6:34:85:6a:e0:18:c5:6e:47:ab:41:42:29:9b:
f6:60:0d:d2:31:d3:63:98:23:93:5a:00:81:48: b4:ef:cd:8a:

cd:c9:cf:99: ee: d9:9e:aa:36:e1:68:4b:71:49:14:36:28:3a: 3d:1d:ce:9a:8f:25:e6:80:71:61:2b: b5:7b: cc:f9:25:16:81: el:31:5f:al: a3:7e:16:a4:9c:16:6a:97:18:bd:76:72:a5:0b: 9e:1d:36:e6:2f:al:2f: be:70:91:0f: a8:e6: da: f8: c4:92:40: 6c:25:7e:7b:b3:09:dc:b2:17:ad:80:44:f0:68:a5:8f:94:75: ff:74:5a:e8: a8:02:7c:0c:09:e2: a9:4b:0b: a0:85:0b:62:b9: ef:a1:31:92: fb:ef:f6:51:04:89:6c:e8: a9:74:al:bb:17:b3: b5: fd:49:0f:7c:3c:ec:83:18:20:43:4e: d5:93: ba:b4:34:b1: 1f:16:36:1f:0c:e6:64:39:16:4c:dc: e0: fe:1d:c8:a9:62:3d: 40:ea: ca: c5:34:02: b4: ae: 89:88:33:35:dc:2c:13:73: d8:27: f1:d0:72: ee:75:3b:22:de: 98:68:66:5b:f1: c6:63:47:55:1c: ba:a5:08:51:75: a6:48:25

Punctuation removed cat signature | tr -d ’[:space:]:’

38960aee3db496le5fef9d9c0b339f2be0cafdd28e0a1f4174a57caa84d4e5f21ee63752329c0bd1611 dbf28c1b6442935757798b27cd9bd74ac8a68e3a9310929016073e3477c53a8904a27ef4bd79f93e 78236ce9a680c82e7cfd410166f5f0e995cf61f717defef7b2f7eea36d697700b15eed75c566a33a5e3 49380cb87dfb8d85a4b1595ef46aelddalf66444aee651832166c6113ef3ce47ee9c281f25daffac6695 dd350f5cef202c62fd91baa9ccfc5a9c93818329974a7c5a72b439d0b777cb79fd693a9237ed6e3865 467ee960bd7988975f3812f4eeaf5b82c886d5e1996d8c04f276ba49f66ee96dle5fa0ef27827640f8a 6d3585c0f2c42da42c67b8834c7c1d8459bc13ec5611dd9635049f634856ae018c56e47ab4142299 bf6600dd231d3639823935a008148b4efcd8acdc9cf99eed99eaa36e1684b71491436283a3d1dce9 a8f25e68071612bb57bccf9251681el315fala37e16a49c166a9718bd7672a50b9e1d36e62fal2fbe70 910fa8e6daf8c492406c257e7bb309dcb217ad8044f068a58f9475ff745ae8a8027c0c09e2a94b0ba0 850b62b9efa13192fbeff65104896ce8a974albb17b3b5fd490f7c3cec831820434ed593bab434b11f1 6361f0ce66439164cdce0fe1dc8a9623d40eacac53402b4ae89883335dc2c1373d827f1d072ee753b 22de9868665bf1c66347551cbaa5085175a64825

```
[10/29/24]seed@VM:~/.../Labsetup$ cat signature | tr -d '[:space:]:'
38960aee3db496le5fef9d9c0b339f2be0cafdd28e0a1f4174a57caa84d4e5f21ee63752329c0bd1611dbf28c1b6442935757798b27cd9bd74ac8a68e3a9310929016073e3477
c53a8904a27ef4bd79f93e78236ce9a680c82e7cfd410166f5f0e995cf61f717defef7b2f7eea36d697700b15eed75c566a33a5e349380cb87dfb8d85a4b1595ef46aelddalf6
6444aee651832166c6113ef3ce47ee9c281f25daffac6695dd350f5cef202c62fd91baa9ccfc5a9c93818329974a7c5a72b439d0b777cb79fd693a9237ed6e3865467ee960bd7
988975f3812f4eeaf5b82c886d5e1996d8c04f276ba49f66ee96dle5fa0ef27827640f8a6d3585c0f2c42da42c67b8834c7c1d8459bc13ec5611dd9635049f634856ae018c56e
47ab4142299bf6600dd231d3639823935a008148b4efcd8acdc9cf99eed99eaa36e1684b71491436283a3d1dce9a8f25e68071612bb57bccf9251681el315fala37e16a49c166
a9718bd7672a50b9e1d36e62fal2fbe70910fa8e6daf8c492406c257e7bb309dcb217ad8044f068a58f9475ff745ae8a8027c0c09e2a94b0ba0850b62b9efa13192fbeff65104
896ce8a974albb17b3b5fd490f7c3cec831820434ed593bab434b11f16361f0ce66439164cdce0fe1dc8a9623d40eacac53402b4ae89883335dc2c1373d827f1d072ee753b22d
e9868665bf1c66347551cbaa5085175a64825[10/29/24]seed@VM:~/.../Labsetup$
```

Step 4:

Purpose: The body of the certificate contains all information except the signature. This body is hashed to produce a value, which should match the decrypted signature if the certificate is authentic.

```
[10/29/24]seed@VM:~/.../Labsetup$ openssl asn1parse -i -in c0.pem
    0:d=0  hl=4 l=1370 cons: SEQUENCE
    4:d=1  hl=4 l= 834 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           :6E47A9C54B470C0DEC33D089B91CF4E1
   31:d=2  hl=2 l=  13 cons:   SEQUENCE
   33:d=3  hl=2 l=   9 prim:    OBJECT            :sha384WithRSAEncryption
   44:d=3  hl=2 l=   0 prim:    NULL
   46:d=2  hl=2 l=  71 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=  34 cons:    SET
   63:d=4  hl=2 l=  32 cons:     SEQUENCE
   65:d=5  hl=2 l=   3 prim:      OBJECT            :organizationName
   70:d=5  hl=2 l=  25 prim:      PRINTABLESTRING   :Google Trust Services LLC
   97:d=3  hl=2 l=  20 cons:    SET
   99:d=4  hl=2 l=  18 cons:     SEQUENCE
  101:d=5  hl=2 l=   3 prim:      OBJECT            :commonName
  106:d=5  hl=2 l=  11 prim:      PRINTABLESTRING   :GTS Root R1
  119:d=2  hl=2 l=  30 cons:   SEQUENCE
  121:d=3  hl=2 l=  13 prim:    UTCTIME           :160622000000Z
  136:d=3  hl=2 l=  13 prim:    UTCTIME           :360622000000Z
```

```
 46:d=2  hl=2 l=  71 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=  34 cons:    SET
 63:d=4  hl=2 l=  32 cons:     SEQUENCE
 65:d=5  hl=2 l=   3 prim:      OBJECT            :organizationName
 70:d=5  hl=2 l=  25 prim:      PRINTABLESTRING   :Google Trust Services LLC
 97:d=3  hl=2 l=  20 cons:    SET
 99:d=4  hl=2 l=  18 cons:     SEQUENCE
101:d=5  hl=2 l=   3 prim:      OBJECT            :commonName
106:d=5  hl=2 l=  11 prim:      PRINTABLESTRING   :GTS Root R1
119:d=2  hl=2 l=  30 cons:   SEQUENCE
121:d=3  hl=2 l=  13 prim:    UTCTIME           :160622000000Z
136:d=3  hl=2 l=  13 prim:    UTCTIME           :360622000000Z
151:d=2  hl=2 l=  71 cons:   SEQUENCE
153:d=3  hl=2 l=  11 cons:    SET
155:d=4  hl=2 l=   9 cons:     SEQUENCE
157:d=5  hl=2 l=   3 prim:      OBJECT            :countryName
162:d=5  hl=2 l=   2 prim:      PRINTABLESTRING   :US
166:d=3  hl=2 l=  34 cons:    SET
168:d=4  hl=2 l=  32 cons:     SEQUENCE
170:d=5  hl=2 l=   3 prim:      OBJECT            :organizationName
175:d=5  hl=2 l=  25 prim:      PRINTABLESTRING   :Google Trust Services LLC
202:d=3  hl=2 l=  20 cons:    SET
```

```
204:d=4  hl=2 l=  18 cons:     SEQUENCE
206:d=5  hl=2 l=   3 prim:      OBJECT            :commonName
211:d=5  hl=2 l=  11 prim:      PRINTABLESTRING   :GTS Root R1
224:d=2  hl=4 l= 546 cons:   SEQUENCE
228:d=3  hl=2 l=  13 cons:    SEQUENCE
230:d=4  hl=2 l=   9 prim:     OBJECT            :rsaEncryption
241:d=4  hl=2 l=   0 prim:     NULL
243:d=3  hl=4 l= 527 prim:    BIT STRING
774:d=2  hl=2 l=  66 cons:   cont [ 3 ]
776:d=3  hl=2 l=  64 cons:    SEQUENCE
778:d=4  hl=2 l=  14 cons:     SEQUENCE
780:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 Key Usage
785:d=5  hl=2 l=   1 prim:      BOOLEAN           :255
788:d=5  hl=2 l=   4 prim:      OCTET STRING      [HEX DUMP]:03020106
794:d=4  hl=2 l=  15 cons:     SEQUENCE
796:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 Basic Constraints
801:d=5  hl=2 l=   1 prim:      BOOLEAN           :255
804:d=5  hl=2 l=   5 prim:      OCTET STRING      [HEX DUMP]:30030101FF
811:d=4  hl=2 l=  29 cons:     SEQUENCE
813:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 Subject Key Identifier
818:d=5  hl=2 l=  22 prim:      OCTET STRING      [HEX DUMP]:0414E4AF2B26711A2B4827852F52662CEFF08913713E
842:d=1  hl=2 l=  13 cons:  SEQUENCE
844:d=2  hl=2 l=   9 prim:   OBJECT            :sha384WithRSAEncryption
855:d=2  hl=2 l=   0 prim:   NULL
857:d=1  hl=4 l= 513 prim:  BIT STRING
```

```
[10/29/24]seed@VM:~/.../Labsetup$ openssl asn1parse -i -in c0.pem -strparse 4 -out c0_body.bin -noout
[10/29/24]seed@VM:~/.../Labsetup$ sha256sum c0_body.bin
18f299911772e1f826f1870e4742a217e2650f975685e8450a1158cd467a8156  c0_body.bin
```

Step 5:

Purpose: Use the C program to manually verify that the signature on the certificate matches the hash of the body, proving the certificate's authenticity.

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

```
#include <openssl/sha.h>

void printBN(char *msg, BIGNUM *a) {
   char *number_str = BN_bn2hex(a);
   printf("%s %s\n", msg, number_str);
   OPENSSL_free(number_str);
}

int main() {
   // Initialize variables and OpenSSL context
   BN_CTX *ctx = BN_CTX_new();
   BIGNUM *n = BN_new();   // Modulus
   BIGNUM *e = BN_new();   // Exponent
   BIGNUM *signature = BN_new();
   BIGNUM *hash_bn = BN_new();
   BIGNUM *decrypted_hash = BN_new();

   // Step 1: Set the issuer's public key (modulus n and exponent e)
   BN_hex2bn(&n,
"A9FF9C7F451E70A8539FCAD9E50DDE4657577DBC8F9A5AAC46F1849ABB91DBC9FB2F01F
B920900165EA1CF8C1ABF9782F4ACCD885A2D8593COED318FBB1F5240D26EEB65B64767C
14C72F7ACEA84CB7F4D908FCDF87233520A8E269E28C4E3FB159FA60A1EB3C920531982CA
36536D604DE90091FC768D5C080F0AC2");
   BN_hex2bn(&e, "65537");

   // Step 2: Set the signature (hex from the signature file)
   BN_hex2bn(&signature,
"38960aee3db496le5fef9d9c0b339f2be0cafdd28e0a1f4174a57caa84d4e5f21ee63752329c0bd161
1dbf28c1b6442935757798b27cd9bd74ac8a68e3a9310929016073e3477c53a8904a27ef4bd79f93
e78236ce9a680c82e7cfd410166f5f0e995cf61f717defef7b2f7eea36d697700b15eed75c566a33a5e
349380cb87dfb8d85a4b1595ef46aelddalf66444aee651832166c6113ef3ce47ee9c281f25daffac669
5dd350f5cef202c62fd91baa9ccfc5a9c93818329974a7c5a72b439d0b777cb79fd693a9237ed6e386
5467ee960bd7988975f3812f4eeaf5b82c886d5e1996d8c04f276ba49f66ee96dle5fa0ef27827640f8
a6d3585c0f2c42da42c67b8834c7c1d8459bc13ec5611dd9635049f634856ae018c56e47ab414229
9bf6600dd231d3639823935a008148b4efcd8acdc9cf99eed99eaa36e1684b71491436283a3d1dce
9a8f25e68071612bb57bccf9251681el315fala37e16a49c166a9718bd7672a50b9e1d36e62fal2fbe7
0910fa8e6daf8c492406c257e7bb309dcb217ad8044f068a58f9475ff745ae8a8027c0c09e2a94b0ba
0850b62b9efa13192fbeff65104896ce8a974albb17b3b5fd490f7c3cec831820434ed593bab434b11f
16361f0ce66439164cdce0fe1dc8a9623d40eacac53402b4ae89883335dc2c1373d827f1d072ee753
b22de9868665bf1c66347551cbaa5085175a64825");

   // Step 3: Calculate the decrypted hash = signature^e mod n
   BN_mod_exp(decrypted_hash, signature, e, n, ctx);

   // Step 4: Compute the expected hash of the certificate body
   // Load the hash manually (SHA-256 hash of c0_body.bin in hex)
   BN_hex2bn(&hash_bn,
"18f299911772e1f826f1870e4742a217e2650f975685e8450a1158cd467a8156");

   // Step 5: Verify if decrypted_hash matches the expected hash
   if (BN_cmp(decrypted_hash, hash_bn) == 0) {
      printf("Signature is valid.\n");
   } else {
      printf("Signature is invalid.\n");
   }
```

```
    // Clean up
    BN_free(n);
    BN_free(e);
    BN_free(signature);
    BN_free(hash_bn);
    BN_free(decrypted_hash);
    BN_CTX_free(ctx);

    return 0;
}
```

```
[10/29/24]seed@VM:~/.../Labsetup$ gcc verify_certificate.c -o verify_certificate -lcrypto
[10/29/24]seed@VM:~/.../Labsetup$ ./verify_certificate
Signature is valid.
```
