Name : Linson Peter Rodrigues

Lab 8: RSA Public-Key Encryption and Signature Lab

Task 1: Deriving the Private Key

Code:

```c
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 128

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

int main() {
    BIGNUM *p = BN_new();
    BIGNUM *q = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *phi = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *res = BN_new();
    BIGNUM *p_minus_1 = BN_new();
    BIGNUM *q_minus_1 = BN_new();

    // Assign values
    BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
    BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
    BN_hex2bn(&e, "0D88C3");

    BN_CTX *ctx = BN_CTX_new();

    // n = pq
    BN_mul(n, p, q, ctx);
    printBN("Public key", n);

    // phi(n) = (p-1)*(q-1)
    BN_sub(p_minus_1, p, BN_value_one());
    BN_sub(q_minus_1, q, BN_value_one());
    BN_mul(phi, p_minus_1, q_minus_1, ctx);

    // Check if e and phi(n) are relatively prime
    BN_gcd(res, phi, e, ctx);
    if (!BN_is_one(res)) {
        printf("Error: e and phi(n) are not relatively prime\n");
        exit(0);
    }

    BN_mod_inverse(d, e, phi, ctx);
    printBN("Private key", d);

    // Clean up
    BN_free(p);
    BN_free(q);
    BN_free(n);
    BN_free(res);
    BN_free(phi);
    BN_free(e);
    BN_free(d);
    BN_free(p_minus_1);
    BN_free(q_minus_1);
    BN_CTX_free(ctx);

    return 0;
}
```

1. Initialized BIGNUM variables for p, q, n, phi, e, d, res, p_minus_1, and q_minus_1.
2. Assigned values to p, q, and e (prime numbers and the public exponent).
3. Calculated the public key modulus n (n = p * q).
4. Computed Euler's totient function phi(n) (phi = (p-1) * (q-1)).
5. Checked if e and phi(n) are relatively prime (GCD is 1); if not, it exits with an error message.
6. Calculated the private exponent d using the modular inverse of e mod phi(n).
7. Printed the public and private keys.
8. Frees memory for all BIGNUM variables and the BN_CTX structure for cleanup.

Implementation:
1. Created task1.c file
2. compiled the task1.c file an executed the code

output:

```
[10/10/23]seed@VM:~/.../lab08$ gcc -o task1 task1.c -lcrypto
[10/10/23]seed@VM:~/.../lab08$ ./task1
public key (0D88C3,E103ABD94892E3E74AFD724BF28E78366D9676BCCC70118BD0AA1968DBB143D1)
private key (3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB,E103ABD94892E3E74AFD724BF28E78366D9676BCCC70118BD0AA1968DBB143D
1)
```

Observation:
Executed the above code and was able to get the public and private key respectively.

Task 2: Encrypting a Message

Code:

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

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

int main() {
    BIGNUM *n = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *M = BN_new();
    BIGNUM *C = BN_new();
    BN_CTX *ctx = BN_CTX_new();

    // Assign values
    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    BN_dec2bn(&e, "65537");
    BN_hex2bn(&M, "4120746f702073656372657421"); // Hex encoded for "A top secret!"

    // Encrypt M: M^e mod n
    BN_mod_exp(C, M, e, n, ctx);
    printBN("Encryption result:", C);

    // Clean up
    BN_free(n);
    BN_free(e);
    BN_free(M);
    BN_free(C);
    BN_CTX_free(ctx);

    return 0;
}
```

1. Initialized BIGNUM variables for n (public key modulus), e (public exponent), M (the message to be encrypted), and C (the encrypted result).
2. Assigned values to n, e, and M.
3. Computed the encryption of M using the formula: C = M^e mod n.
4. Printed the result of the encryption.
5. Frees memory for BIGNUM variables and the BN_CTX structure for cleanup.

Implementation:

1. Created a encrypt.c file
2. Compiled the code
3. Created a task2.txt file
4. xxd -ps was used to read the binary data from the file
5. Executed the encrypt.c file

Output:

```
[10/10/23]seed@VM:~/.../lab08$ gedit encrypt.c
[10/10/23]seed@VM:~/.../lab08$ gcc -o encrypt encrypt.c -lcrypto
[10/10/23]seed@VM:~/.../lab08$ echo -n "A top secret!" > task2.txt
[10/10/23]seed@VM:~/.../lab08$ xxd -ps task2.txt
4120746f702073656372657421
[10/10/23]seed@VM:~/.../lab08$ ./task2
bash: ./task2: No such file or directory
[10/10/23]seed@VM:~/.../lab08$ ./encrypt
Encryption result: 6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC
```

Observation :

Was able to get the encryption result .

Task 3: Decrypting a Message

Code:

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

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

int main() {
    BIGNUM *n = BN_new();
    BIGNUM *M = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *C = BN_new();
    BN_CTX *ctx = BN_CTX_new();

    // Assign values
    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
    BN_hex2bn(&C, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");

    // Decrypt C: C^d mod n
    BN_mod_exp(M, C, d, n, ctx);
    printBN("Decryption result:", M);

    // Clean up
    BN_free(n);
    BN_free(d);
    BN_free(M);
    BN_free(C);
    BN_CTX_free(ctx);

    return 0;
}
```

1. initialized BIGNUM variables for n (public key modulus), M (the decrypted message), d (private exponent), and C (the ciphertext to be decrypted).
2. Assigned values to n, d, and C.
3. Used the formula M = C^d mod n to decrypt the ciphertext C using the private key (n and d).
4. Printed the result of the decryption, which is the original message M.
5. Frees memory for BIGNUM variables and the BN_CTX structure for cleanup.

Implementation:

1. Created the decrypt.c file
2. Compiled the decrypt.c file
3. Executed the decrypt.c file
4. Got the decryption result
5. Copied the decryption result.
6. Pasted the result in task3.txt file
7. And got the result

Output:

```
[10/10/23]seed@VM:~/.../lab08$ gedit decrypt.c
[10/10/23]seed@VM:~/.../lab08$ gcc -o decrypt decrypt.c -lcrypto
[10/10/23]seed@VM:~/.../lab08$ ./decrypt
Decryption result: 50617373776F72642069732064656573
[10/10/23]seed@VM:~/.../lab08$ echo -n "50617373776F72642069732064656573" > task3.txt
[10/10/23]seed@VM:~/.../lab08$ xxd -r -p task3.txt
Password is dees[10/10/23]seed@VM:~/.../lab08$
[10/10/23]seed@VM:~/.../lab08$
```

Observation:

Was able to get the output as “Password is dees”.

Task 4: Signing a Message

Code:

```c
#include <stdio.h>
#include <openssl/bn.h>
void printBN(const char *msg, const BIGNUM *a) {
    char *number_str = BN_bn2hex(a);
    printf("%s %s\n", msg, number_str);
    OPENSSL_free(number_str);
}
int main() {
    BIGNUM *n = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *M1 = BN_new();
    BIGNUM *M2 = BN_new();
    BIGNUM *C1 = BN_new();
    BIGNUM *C2 = BN_new();
    BN_CTX *ctx = BN_CTX_new();

    // Assign values
    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
    BN_hex2bn(&M1, "49206f776520796f75202432303030"); // Hex encoded for "I owe you $2000"
    BN_hex2bn(&M2, "49206f776520796f75202433303030"); // Hex encoded for "I owe you $3000"

    // Encrypt M: M^d mod n
    BN_mod_exp(C1, M1, d, n, ctx);
    BN_mod_exp(C2, M2, d, n, ctx);
    printBN("Signature of M1:", C1);
    printBN("Signature of M2:", C2);
    // Clean up
    BN_free(n);
    BN_free(d);
    BN_free(M1);
    BN_free(M2);
    BN_free(C1);
    BN_free(C2);
    BN_CTX_free(ctx);

    return 0;
}
```

1. Initialized BIGNUM variables for n (public key modulus), d (private exponent), M1 and M2 (messages to be signed), and C1 and C2 (the signatures).
2. Assigned values to n and d, which are part of the private key.
3. Assigned values to M1 and M2, which are the messages to be signed.
4. Encrypted M1 and M2 individually using the private key, resulting in C1 and C2, which are the signatures.
5. Printed the signatures of M1 and M2.
6. Frees memory for BIGNUM variables and the BN_CTX structure for cleanup.

Implementation:

1. Created to files task4a.txt, task4b.txt
2. xxd -ps was used to read the binary data from the file
3. Added the binary data in the code
4. Compiled the code
5. Executed the code

Output:

```
[10/10/23]seed@VM:~/.../lab08$ echo -n "I owe you $2000." > task4a.txt
[10/10/23]seed@VM:~/.../lab08$ echo -n "I owe you $4000." > task4b.txt
[10/10/23]seed@VM:~/.../lab08$ xxd -ps task4a.txt
49206f776520796f75203030302e
[10/10/23]seed@VM:~/.../lab08$ xxd -ps task4b.txt
49206f776520796f75203030302e
```

```
[10/10/23]seed@VM:~/.../lab08$ gcc -o diff diff.c -lcrypto
[10/10/23]seed@VM:~/.../lab08$ ./diff
Signature of M1: 80A55421D72345AC199836F60D51DC9594E2BDB4AE20C804823FB71660DE7B82
Signature of M2: 04FC9C53ED7BBE4ED4BE2C24B0BDF7184B96290B4ED4E3959F58E94B1ECEA2EB
[10/10/23]seed@VM:~/.../lab08$
```

Observation:

Was able to get the signature value for M1, M2 respectively .

Task 5: Verifying a Signature

Code:

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

void printBN(const char *msg, const BIGNUM *a) {
    char *number_str = BN_bn2hex(a);
    printf("%s %s\n", msg, number_str);
    OPENSSL_free(number_str);
}
int main() {
    BIGNUM *n = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *M = BN_new();
    BIGNUM *C = BN_new();
    BIGNUM *S = BN_new();
    BN_CTX *ctx = BN_CTX_new();
    // Assign values
    BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");
    BN_dec2bn(&e, "65537");
    BN_hex2bn(&M, "4c61756e63682061206d697373696c652e"); // Hex encoded for "Launch a missile."
    BN_hex2bn(&S, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");
    // Get S^e mod n: if S = M^d mod n, C = M
    BN_mod_exp(C, S, e, n, ctx);
    // Verify the signature
    if (BN_cmp(C, M) == 0) {
        printf("Valid Signature!\n");
    } else {
        printf("Verification fails!\n");
    }
    // Clean up
    BN_free(n);
    BN_free(e);
    BN_free(M);
    BN_free(C);
    BN_free(S);
    BN_CTX_free(ctx);

    return 0;
}
```

1. initialized BIGNUM variables for n (public key modulus), e (public exponent), M (the original message), C (the decrypted message), and S (the digital signature).
2. Assigned values to n, e, M, and S.
3. Calculated C = S^e mod n, which is the decryption of the signature using the public key.
4. Compared C with M to verify the signature. If they are equal, it indicates a valid signature; otherwise, it signifies that the verification has failed.
5. Printed the result, either "Valid Signature!" or "Verification fails!" based on the comparison.
6. Frees memory for BIGNUM variables and the BN_CTX structure for cleanup.

Implementation:

1. Created a new task5.txt file
2. Used xxd -ps to read the binary data from the file
3. Created verifysignature.c file
4. Compiled the the verifysignature.c file
5. Executed the file

Output:

```
[10/10/23]seed@VM:~/.../lab08$ echo -n "Launch a missle." > task5.txt
[10/10/23]seed@VM:~/.../lab08$ xxd -ps task5.txt
4c61756e63682061206d6973736c652e
[10/10/23]seed@VM:~/.../lab08$ gedit verifysignature.c
[10/10/23]seed@VM:~/.../lab08$ ./verifysignature
bash: ./verifysignature: No such file or directory
[10/10/23]seed@VM:~/.../lab08$ gcc -o verifysignature verifysignature.c -lcrypto
[10/10/23]seed@VM:~/.../lab08$ ./verifysignature
Valid Signature!
[10/10/23]seed@VM:~/.../lab08$
```

Observation:

Was able to get the output as a valid signature!

Task 6: Manually Verifying an X.509 Certificate

Implementation:

- Downloaded a certificate from a real web server
- Exported the certificate to seed.txt file

```
[10/15/23]seed@VM:~/.../LAB08$ 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
```

- Saved the first certification as c0.pem
- Saved the second certification as c1.pem
- Printed all attributes of the certificate, and then found the exponent, which is public key e

```
[10/15/23]seed@VM:~/.../LAB08$ gedit c0.pem
[10/15/23]seed@VM:~/.../LAB08$ gedit c1.pem
[10/15/23]seed@VM:~/.../LAB08$ openssl x509 -in c1.pem -noout -modulus
Modulus=B6E02FC22406C86D045FD7EF0A6406B27D22266516AE42409BCEDC9F9F76073EC33
0558719B94F940E5A941F5556B4C2022AAFD098EE0B40D7C4D03B72C8149EEF90B111A9AED2
C8B8433AD90B0BD5D595F540AFC81DED4D9C5F57B786506899F58ADAD2C7051FA897C9DCA4B
182842DC6ADA59CC71982A6850F5E44582A378FFD35F10B0827325AF5BB8B9EA4BD51D027E2
DD3B4233A30528C4BB28CC9AAC2B230D78C67BE65E71B74A3E08FB81B71616A19D23124DE5D
79208AC75A49CBACD17B21E4435657F532539D11C0A9A631B199274680A37C2C25248CB395A
A2B6E15DC1DDA020B821A293266F144A2141C7ED6D9BF2482FF303F5A26892532F5EE3
```

- Extracted the public key e,n from the issuer's certificate

```
[10/15/23]seed@VM:~/.../LAB08$ openssl x509 -in c1.pem -text -noout | grep
Exponent
                Exponent: 65537 (0x10001)
```

```
[10/16/23]seed@VM:~/.../LAB08$ openssl x509 -in c0.pem -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            02:49:3e:07:fa:9e:37:5a:2d:bb:c6:1d:94:43:0f:cf
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert SHA2 High Assurance Server CA
        Validity
            Not Before: May  6 00:00:00 2020 GMT
            Not After : Apr 14 12:00:00 2022 GMT
        Subject: C = US, ST = California, L = San Francisco, O = "GitHub, Inc.", CN = www.github.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:b2:3e:3d:ea:32:7d:f6:f7:84:5c:ee:d6:77:11:
                    90:67:b8:9d:b4:29:c3:72:36:6a:41:e0:e2:cb:ad:
                    39:ed:2c:17:8f:dc:83:51:59:ea:87:0a:3f:c8:82:
                    3c:fb:2e:89:a6:4d:e0:ef:92:e9:3e:96:0d:8e:23:
                    b0:62:9f:c9:92:95:9b:a4:10:1d:d4:55:a5:bd:72:
                    68:fd:e6:52:f9:3c:2e:21:86:fb:25:27:36:1f:73:
                    43:0c:ab:76:55:5e:52:48:a4:1d:5c:e9:03:36:aa:
                    bb:06:2b:0e:16:9d:bf:4d:8c:02:31:96:e8:2e:e7:
                    f0:45:1e:5f:2b:12:f4:9d:32:88:38:2c:29:9c:7f:
                    24:ff:f5:1d:5a:9b:d3:80:88:b1:0c:62:d8:40:2c:
                    07:b7:47:d8:12:8e:3d:8c:57:83:68:a0:f8:58:c0:
                    4f:ab:a4:f1:d6:22:96:06:1c:37:1d:57:3f:3b:2b:
                    c8:df:28:d1:03:1e:00:9d:ff:c3:09:2b:7d:61:35:
                    ba:7f:f3:43:e0:ef:df:3a:8c:1c:7d:6e:53:ef:38:
                    a2:d4:ac:34:cb:0f:ae:94:1a:c4:ff:17:ba:b5:a5:
                    61:2f:e4:11:d0:c5:cd:91:05:76:88:73:4d:f3:9b:
                    30:89:dc:0c:b5:c7:46:23:c8:37:7d:82:f8:f1:b4:
                    91:cb
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Authority Key Identifier:
                keyid:51:68:FF:90:AF:02:07:75:3C:CC:D9:65:64:62:A2:12:B8:59:72:3B
```

```
            Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Authority Key Identifier:
                keyid:51:68:FF:90:AF:02:07:75:3C:CC:D9:65:64:62:A2:12:B8:59:72:3B

            X509v3 Subject Key Identifier:
                8C:A0:0A:69:47:DC:89:32:B0:4D:C6:11:45:62:5F:1A:2F:96:4E:3A
            X509v3 Subject Alternative Name:
                DNS:www.github.com, DNS:*.github.com, DNS:github.com, DNS:*.github.io, DNS:github.io, DNS:*.githubusercontent.com, DNS:github
usercontent.com
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
            X509v3 CRL Distribution Points:

                Full Name:
                  URI:http://crl3.digicert.com/sha2-ha-server-g6.crl

                Full Name:
                  URI:http://crl4.digicert.com/sha2-ha-server-g6.crl

            X509v3 Certificate Policies:
                Policy: 2.16.840.1.114412.1.1
                  CPS: https://www.digicert.com/CPS
                Policy: 2.23.140.1.2.2

            Authority Information Access:
                OCSP - URI:http://ocsp.digicert.com
                CA Issuers - URI:http://cacerts.digicert.com/DigiCertSHA2HighAssuranceServerCA.crt

            X509v3 Basic Constraints: critical
                CA:FALSE
            CT Precertificate SCTs:
                Signed Certificate Timestamp:
                    Version   : v1 (0x0)
                    Log ID    : 46:A5:55:EB:75:FA:91:20:30:B5:A2:89:69:F4:F3:7D:
                                11:2C:41:74:BE:FD:49:B8:85:AB:F2:FC:70:FE:6D:47
                    Timestamp : May  6 18:11:06.134 2020 GMT
```

```
            CT Precertificate SCTs:
                Signed Certificate Timestamp:
                    Version   : v1 (0x0)
                    Log ID    : 46:A5:55:EB:75:FA:91:20:30:B5:A2:89:69:F4:F3:7D:
                                11:2C:41:74:BE:FD:49:B8:85:AB:F2:FC:70:FE:6D:47
                    Timestamp : May  6 18:11:06.134 2020 GMT
                    Extensions: none
                    Signature : ecdsa-with-SHA256
                                30:45:02:21:00:E7:DC:BA:C3:DA:1A:A0:0A:0C:D9:FB:
                                C5:AB:A2:A8:7D:85:91:87:4C:51:9B:85:44:11:A3:07:
                                06:DB:01:61:48:02:20:16:71:26:18:C0:ED:21:96:52:
                                5A:39:ED:8B:25:1B:CB:BA:72:51:E7:80:33:6F:A1:33:
                                55:C3:51:D0:B5:3A:F7
                Signed Certificate Timestamp:
                    Version   : v1 (0x0)
                    Log ID    : 22:45:45:07:59:55:24:56:96:3F:A1:2F:F1:F7:6D:86:
                                E0:23:26:63:AD:C0:4B:7F:5D:C6:83:5C:6E:E2:0F:02
                    Timestamp : May  6 18:11:06.047 2020 GMT
                    Extensions: none
                    Signature : ecdsa-with-SHA256
                                30:44:02:20:66:12:38:A2:A1:36:DC:D7:FF:E1:08:89:
                                D8:21:7E:8E:F2:1A:61:E6:82:D1:54:60:93:C6:B4:C1:
                                7D:24:53:DB:02:20:67:3E:F1:DD:93:48:29:09:16:3C:
                                A3:85:69:A5:B7:03:17:01:D0:3C:E1:16:C0:FA:83:18:
                                64:6E:5D:69:D0:08
                Signed Certificate Timestamp:
                    Version   : v1 (0x0)
                    Log ID    : 51:A3:B0:F5:FD:01:79:9C:56:6D:B8:37:78:8F:0C:A4:
                                7A:CC:1B:27:CB:F7:9E:88:42:9A:0D:FE:D4:8B:05:E5
                    Timestamp : May  6 18:11:06.121 2020 GMT
                    Extensions: none
                    Signature : ecdsa-with-SHA256
                                30:45:02:20:14:3F:E8:49:7E:4C:20:AD:5A:EA:29:EC:
                                87:5E:AC:DD:3E:6F:29:F9:B0:3A:DE:4E:EF:97:CF:71:
                                A2:63:C7:0B:02:21:00:ED:2F:37:1F:11:45:75:08:50:
                                AA:98:E3:6D:66:C8:54:10:3C:F0:D9:FF:AE:26:9F:E4:
                                96:FC:87:7A:09:57:C7
```

- Found the location of last “Signature Algorithm” hex string an exported the body to new file signature and then trimmed all the spaces and colons in it.

```
[10/16/23]seed@VM:~/.../LAB08$ cat signature | tr -d '[:space:]:'
cat: signature: No such file or directory
[10/16/23]seed@VM:~/.../LAB08$ touch signature
[10/16/23]seed@VM:~/.../LAB08$ gedit signature
[10/16/23]seed@VM:~/.../LAB08$ cat signature | tr -d '[:space:]:'
SignatureAlgorithmsha256WithRSAEncryption00f3bbf23fe1d30fc06e10ccc1476668101659dcff1a97b5a34ba8e348cd73f39c14261d08b8f35c4a8004788d93934e49e5
c0e2c15e70d7bd5eab250657badde9c474af54993692fbb20cedd10b4bae75df35017214b1de8f9e3b760fa5ddff2a54028324c84fbc7ae604484164e07967ae95ed37b3924c6
558650934689ac320db255dd9942fd13a01088861a448a51311763e2cb46e8290f2697d26ae59ad7d911799ea14d04797fcf4beb1e74bacec6b969661fa12654521b85ff443b4
d9003709c53b6c4d622d630798a714eb2b619a0b2f3515394e2931bc5efb245bfb9f5ff2f062eba6b98aa41e900dfe0f03c4bd44e5fd4738307b729320ceaa78a5[10/16/23]s
```

- Parsed the server’s certificate

```
eed@VM:~/.../LAB08$ 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
```

```
  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
  252:d=3  hl=2 l=  21 cons:    SET
  254:d=4  hl=2 l=  19 cons:     SEQUENCE
  256:d=5  hl=2 l=   3 prim:      OBJECT            :organizationName
  261:d=5  hl=2 l=  12 prim:      PRINTABLESTRING   :GitHub, Inc.
  275:d=3  hl=2 l=  23 cons:    SET
  277:d=4  hl=2 l=  21 cons:     SEQUENCE
  279:d=5  hl=2 l=   3 prim:      OBJECT            :commonName
  284:d=5  hl=2 l=  14 prim:      PRINTABLESTRING   :www.github.com
  300:d=2  hl=4 l= 290 cons:   SEQUENCE
  304:d=3  hl=2 l=  13 cons:    SEQUENCE
  306:d=4  hl=2 l=   9 prim:     OBJECT            :rsaEncryption
  317:d=4  hl=2 l=   0 prim:     NULL
  319:d=3  hl=4 l= 271 prim:    BIT STRING
  594:d=2  hl=4 l= 970 cons:   cont [ 3 ]
  598:d=3  hl=4 l= 966 cons:    SEQUENCE
  602:d=4  hl=2 l=  31 cons:     SEQUENCE
  604:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 Authority Key Identifier
  609:d=5  hl=2 l=  24 prim:      OCTET STRING      [HEX DUMP]:301680145168FF90AF0207753CCCD9656462A212B859723B
  635:d=4  hl=2 l=  29 cons:     SEQUENCE
  637:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 Subject Key Identifier
  642:d=5  hl=2 l=  22 prim:      OCTET STRING      [HEX DUMP]:04148CA00A6947DC8932B04DC61145625F1A2F964E3A
  666:d=4  hl=2 l= 123 cons:     SEQUENCE
  668:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 Subject Alternative Name
  673:d=5  hl=2 l= 116 prim:      OCTET STRING      [HEX DUMP]:3072820E7777772E6769746875622E636F6D820C2A2E6769746875622E636F6D820A6769746875
622E636F6D820B2A2E6769746875622E696F82096769746875622E696F82172A2E67697468756275736572636F6E74656E742E636F6D821567697468756275736572636F6E746
56E742E636F6D
  791:d=4  hl=2 l=  14 cons:     SEQUENCE
  793:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 Key Usage
  798:d=5  hl=2 l=   1 prim:      BOOLEAN           :255
  801:d=5  hl=2 l=   4 prim:      OCTET STRING      [HEX DUMP]:030205A0
  807:d=4  hl=2 l=  29 cons:     SEQUENCE
  809:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 Extended Key Usage
  814:d=5  hl=2 l=  22 prim:      OCTET STRING      [HEX DUMP]:301406082B06010505070301060 82B06010505070302
  838:d=4  hl=2 l= 117 cons:     SEQUENCE
  840:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 CRL Distribution Points
```

```
  791:d=4  hl=2 l=  14 cons:     SEQUENCE
  793:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 Key Usage
  798:d=5  hl=2 l=   1 prim:      BOOLEAN           :255
  801:d=5  hl=2 l=   4 prim:      OCTET STRING      [HEX DUMP]:030205A0
  807:d=4  hl=2 l=  29 cons:     SEQUENCE
  809:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 Extended Key Usage
  814:d=5  hl=2 l=  22 prim:      OCTET STRING      [HEX DUMP]:301406082B0601050507030106082B06010505070302
  838:d=4  hl=2 l= 117 cons:     SEQUENCE
  840:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 CRL Distribution Points
  845:d=5  hl=2 l= 110 prim:      OCTET STRING      [HEX DUMP]:306C3034A032A030862E687474703A2F2F63726C332E64696769636572742E636F6D2F73686132
2D68612D7365727665722D67362E63726C3034A032A030862E687474703A2F2F63726C342E64696769636572742E636F6D2F736861322D68612D7365727665722D67362E63726
C
  957:d=4  hl=2 l=  76 cons:     SEQUENCE
  959:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 Certificate Policies
  964:d=5  hl=2 l=  69 prim:      OCTET STRING      [HEX DUMP]:3043303706096086480186FD6C0101302A302806082B06010505070201161C68747470733A2F2F
7777772E64696769636572742E636F6D2F435053300806066781 0C010202
 1035:d=4  hl=3 l= 131 cons:     SEQUENCE
 1038:d=5  hl=2 l=   8 prim:      OBJECT            :Authority Information Access
 1048:d=5  hl=2 l= 119 prim:      OCTET STRING      [HEX DUMP]:3075302406082B060105050730018618687474703A2F2F6F6373702E64696769636572742E636F
6D304D06082B060105050730028641687474703A2F2F636163657274732E64696769636572742E636F6D2F446967694365727453484132486967684173737572616E636553657
276657243412E637274
 1169:d=4  hl=2 l=  12 cons:     SEQUENCE
 1171:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 Basic Constraints
 1176:d=5  hl=2 l=   1 prim:      BOOLEAN           :255
 1179:d=5  hl=2 l=   2 prim:      OCTET STRING      [HEX DUMP]:3000
 1183:d=4  hl=4 l= 381 cons:     SEQUENCE
 1187:d=5  hl=2 l=  10 prim:      OBJECT            :CT Precertificate SCTs
 1199:d=5  hl=4 l= 365 prim:      OCTET STRING      [HEX DUMP]:0482016901670076004 6A555EB75FA912030B5A28969F4F37D112C4174BEFD49B885ABF2FC70FE
6D4700000171EB2F1F16000004030047304502210 0E7DCBAC3DA1AA00A0CD9FBC5ABA2A87D8591874C519B854411A30706DB016148022016712618C0ED2196525A39ED8B251BC
BBA7251E780336FA13355C351D0B53AF7007500224545075955245696 3FA12FF1F76D86E0232663ADC04B7F5DC6835C6EE20F0200000171EB2F1EBF00000403004630440220 66
1238A2A136DCD7FFE10889D8217E8EF21A61E682D1546093C6B4C17D2453DB0220673EF1DD93482909163CA38569A5B7031701D03CE116C0FA8318646E5D69D00800760051A3B
0F5FD01799C566DB837788F0CA47ACC1B27CBF79E88429A0DFED48B05E500000171EB2F1F09000004030047304502201 43FE8497E4C20AD5AEA29EC875EACDD3E6F29F9B03ADE
4EEF97CF71A263C70B022100ED2F371F1145750850AA98E36D66C854103CF0D9FFAE269FE496FC877A0957C7
 1568:d=1  hl=2 l=  13 cons:  SEQUENCE
 1570:d=2  hl=2 l=   9 prim:   OBJECT            :sha256WithRSAEncryption
 1581:d=2  hl=2 l=   0 prim:   NULL
 1583:d=1  hl=4 l= 257 prim:  BIT STRING
```

```
 1183:d=4  hl=4 l= 381 cons:     SEQUENCE
 1187:d=5  hl=2 l=  10 prim:      OBJECT            :CT Precertificate SCTs
 1199:d=5  hl=4 l= 365 prim:      OCTET STRING      [HEX DUMP]:04820169016700760046A555EB75FA912030B5A28969F4F37D112C4174BEFD49B885ABF2FC70FE
6D4700000171EB2F1F160000040300473045022100E7DCBAC3DA1AA00A0CD9FBC5ABA2A87D8591874C519B854411A30706DB016148022016712618C0ED2196525A39ED8B251BC
BBA7251E780336FA13355C351D0B53AF70075002245450759552456963FA12FF1F76D86E0232663ADC04B7F5DC6835C6EE20F0200000171EB2F1EBF0000040300463044022066
1238A2A136DCD7FFE10889D8217E8EF21A61E682D1546093C6B4C17D2453DB0220673EF1DD93482909163CA38569A5B7031701D03CE116C0FA8318646E5D69D00800760051A3B
0F5FD01799C566DB837788F0CA47ACC1B27CBF79E88429A0DFED48B05E500000171EB2F1F090000040300473045022014 3FE8497E4C20AD5AEA29EC875EACDD3E6F29F9B03ADE
4EEF97CF71A263C70B022100ED2F371F1145750850AA98E36D66C854103CF0D9FFAE269FE496FC877A0957C7
 1568:d=1  hl=2 l=  13 cons:  SEQUENCE
 1570:d=2  hl=2 l=   9 prim:   OBJECT            :sha256WithRSAEncryption
 1581:d=2  hl=2 l=   0 prim:   NULL
 1583:d=1  hl=4 l= 257 prim:  BIT STRING
[10/16/23]seed@VM:~/.../LAB08$ openssl asn1parse -i -in c0.pem -strparse 4 -out c0_body.bin -noout
[10/16/23]seed@VM:~/.../LAB08$ sha256sum c0_body.bin
0640f8d13c0789ff0ed5437cf4bc9f2827d52146dddff38aefc2c17747d45f28  c0_body.bin
```

- The field starting from 4:d=1 hl=4 l=1560 cons: SEQUENCE
  is the body of the certificate that is used to generate the hash. And the line before the last "sha256WithRSAEncryption" is the beginning of the signature block.
  1568:d=1 hl=2 l= 13 cons: SEQUENCE
- So, the certificate body starts from offset 4 to 1567. Use -strparse to get the field from the offset 4, which is exactly the body of server's certificate
- Calculated its hash value

```
[10/16/23]seed@VM:~/.../LAB08$ python3
Python 3.8.10 (default, May 26 2023, 14:05:08)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> prefix = "0001"
>>> hash = "0640f8d13c0789ff0ed5437cf4bc9f2827d52146dddff38aefc2c17747d45f28"
>>> A = "30 31 30 0D 06 09 60 86 48 01 65 03 04 02 01 05 00 04 20".replace(' ','')
>>> total_len = 256
>>> pad_len = total_len - 1 - (len(A)+len(prefix)+len(hash))//2
>>> prefix + "FF" * pad_len + "00" + A + hash
'0001FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF003031300D0609
608648016503040201050004200640f8d13c0789ff0ed5437cf4bc9f2827d52146dddff38aefc2c17747d45f28'
>>> exit
Use exit() or Ctrl-D (i.e. EOF) to exit
>>> 
```

- Since sizes of both the signature and the n are 256 bytes, we pad the hash into 256 bytes

CODE:

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

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

int main()
{
    // init
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM *n = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *M = BN_new();
    // BIGNUM *d = BN_new();
    BIGNUM *C = BN_new();
    BIGNUM *S = BN_new();

    // assign values
    BN_hex2bn(&n,
"B6E02FC22406C86D045FD7EF0A6406B27D22266516AE42409BCEDC9F9F76073EC330558719B94F940E5A941F5556B4C2022AAFD098EE0B40D7C4D03B72C8149EEF90B111A
    BN_dec2bn(&e, "65537");
    BN_hex2bn(&M,
"0001FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
hash of the certificate body
    BN_hex2bn(&S,
"00f3bbf23fe1d30fc06e10ccc1476668101659dcff1a97b5a34ba8e348cd73f39c14261d08b8f35c4a8004788d93934e49e5c0e2c15e70d7bd5eab250657badde9c474af5

    // decrypt S: S^e mod n = (M^d mod)^e mod n = M
    BN_mod_exp(C, S, e, n, ctx);

    if (BN_cmp(C, M) == 0)
    {
        printf("Valid Signature! \n");
    }
    else
    {
        printf("Verification fails! \n");
    }

    // clear sensitive data
    BN_clear_free(n);
    BN_clear_free(e);
    BN_clear_free(M);
    BN_clear_free(C);
    BN_clear_free(S);

    return 0;
}
```

- Copied the same code which was executed in the task 5 just replaced the initial values of variables.
- Compiled the code and executed the code.

Output:

```
[10/16/23]seed@VM:~/.../LAB08$ gcc -o verify_ca verify_ca.c -lcrypto
[10/16/23]seed@VM:~/.../LAB08$ ./verify_ca
Valid Signature!
```

Observation:

Was able to Manually Verify an X.509 Certificate.