

# RSA

Q1)

```
Computer Security CW2 > RSA > C rsa_q1.c
1  #include <stdio.h>
2  #include <openssl/bn.h>
3
4  int main() {
5      BN_CTX *ctx = BN_CTX_new();
6
7      BIGNUM *p = BN_new();
8      BIGNUM *q = BN_new();
9      BIGNUM *n = BN_new();
10     BIGNUM *phi = BN_new();
11     BIGNUM *e = BN_new();
12     BIGNUM *d = BN_new();
13     BIGNUM *M = BN_new();
14     BIGNUM *C = BN_new();
15     BIGNUM *temp1 = BN_new();
16     BIGNUM *temp2 = BN_new();
17
18     // Set values
19     BN_dec2bn(&p, "71");
20     BN_dec2bn(&q, "97");
21     BN_dec2bn(&n, "6887");
22     BN_dec2bn(&e, "143");
23     BN_dec2bn(&M, "1234");
24
25     // phi = (p-1)(q-1)
26     BN_sub(temp1, p, BN_value_one()); // temp1 = p - 1
27     BN_sub(temp2, q, BN_value_one()); // temp2 = q - 1
28     BN_mul(phi, temp1, temp2, ctx); // phi = (p-1)(q-1)
29
30     // Compute d = e^-1 mod phi
31     BN_mod_inverse(d, e, phi, ctx);
32
33     // Compute C = M^e mod n
34     BN_mod_exp(C, M, e, n, ctx);
35
36     // Print results
37     char *d_str = BN_bn2dec(d);
38     char *C_str = BN_bn2dec(C);
39
40     printf("Decryption key d: %s\n", d_str);
41     printf("Ciphertext C: %s\n", C_str);
42
43     // Free
44     OPENSSL_free(d_str);
45     OPENSSL_free(C_str);
46
47     BN_free(p); BN_free(q); BN_free(n); BN_free(phi); BN_free(e); BN_free(d);
48     BN_free(M); BN_free(C); BN_free(temp1); BN_free(temp2);
49     BN_CTX_free(ctx);
50
51     return 0;
52 }
53
```

Output :

```
>>> Compiling rsa_q1.c
>>> Running rsa_q1
-----
Decryption key d: 47
Ciphertext C: 4347
-----
```

Q2)

```

Computer Security CW2 > RSA > C rsa_q2.c
1  #include <stdio.h>
2  #include <openssl/bn.h>
3
4  int main() {
5      BN_CTX *ctx = BN_CTX_new();
6
7      BIGNUM *p = BN_new();
8      BIGNUM *q = BN_new();
9      BIGNUM *n = BN_new();
10     BIGNUM *phi = BN_new();
11     BIGNUM *d = BN_new();
12     BIGNUM *e = BN_new();
13     BIGNUM *M = BN_new();
14     BIGNUM *C = BN_new();
15     BIGNUM *temp1 = BN_new();
16     BIGNUM *temp2 = BN_new();
17
18     // Set known values
19     BN_dec2bn(&p, "223");
20     BN_dec2bn(&q, "311");
21     BN_dec2bn(&n, "69353");
22     BN_dec2bn(&d, "29401");
23     BN_dec2bn(&M, "12345");
24
25     // phi = (p - 1) * (q - 1)
26     BN_sub(temp1, p, BN_value_one()); // temp1 = p - 1
27     BN_sub(temp2, q, BN_value_one()); // temp2 = q - 1
28     BN_mul(phi, temp1, temp2, ctx); // phi = (p-1)(q-1)
29
30     // e = d^-1 mod phi
31     BN_mod_inverse(e, d, phi, ctx);
32
33     // Encrypt C = M^e mod n
34     BN_mod_exp(C, M, e, n, ctx);
35
36     // Print results
37     char *e_str = BN_bn2dec(e);
38     char *C_str = BN_bn2dec(C);
39
40     printf("Encryption key e: %s\n", e_str);
41     printf("Ciphertext C: %s\n", C_str);
42
43     // Free memory
44     OPENSSL_free(e_str);
45     OPENSSL_free(C_str);
46
47     BN_free(p); BN_free(q); BN_free(n); BN_free(phi); BN_free(e); BN_free(d);
48     BN_free(M); BN_free(C); BN_free(temp1); BN_free(temp2);
49     BN_CTX_free(ctx);
50
51     return 0;
52 }

```

Output :

```

>>> Compiling rsa_q2.c
>>> Running rsa_q2
-----
Encryption key e: 4321
Ciphertext C: 58684
-----

```

Q3)

```

Computer Security CW2 > RSA > C rsa_q3.c
1  #include <stdio.h>
2  #include <openssl/bn.h>
3
4  int main() {
5      BN_CTX *ctx = BN_CTX_new();
6
7      BIGNUM *p = BN_new();
8      BIGNUM *q = BN_new();
9      BIGNUM *n = BN_new();
10     BIGNUM *phi = BN_new();
11     BIGNUM *e = BN_new();
12     BIGNUM *d = BN_new();
13     BIGNUM *C = BN_new();
14     BIGNUM *M = BN_new();
15     BIGNUM *temp1 = BN_new();
16     BIGNUM *temp2 = BN_new();
17
18     // Set values
19     BN_dec2bn(&p, "37");
20     BN_dec2bn(&q, "67");
21     BN_dec2bn(&e, "169");
22     BN_dec2bn(&C, "1744");
23
24     // n = p * q
25     BN_mul(n, p, q, ctx);
26
27     // phi = (p - 1)(q - 1)
28     BN_sub(temp1, p, BN_value_one());
29     BN_sub(temp2, q, BN_value_one());
30     BN_mul(phi, temp1, temp2, ctx);
31
32     // d = e^-1 mod phi
33     BN_mod_inverse(d, e, phi, ctx);
34
35     // M = C^d mod n
36     BN_mod_exp(M, C, d, n, ctx);
37
38     // Print results
39     char *n_str = BN_bn2dec(n);
40     char *phi_str = BN_bn2dec(phi);
41     char *d_str = BN_bn2dec(d);
42     char *M_str = BN_bn2dec(M);
43
44     printf("n (modulus)      : %s\n", n_str);
45     printf("phi(n)           : %s\n", phi_str);
46     printf("Decryption key d : %s\n", d_str);
47     printf("Decrypted M      : %s\n", M_str);
48
49     // Free memory
50     OPENSSL_free(n_str);
51     OPENSSL_free(phi_str);
52     OPENSSL_free(d_str);
53     OPENSSL_free(M_str);
54
55     BN_free(p); BN_free(q); BN_free(n); BN_free(phi); BN_free(e); BN_free(d);
56     BN_free(C); BN_free(M); BN_free(temp1); BN_free(temp2);
57     BN_CTX_free(ctx);
58
59     return 0;
60 }

```

Output :

```

>>> Compiling rsa_q3.c
>>> Running rsa_q3
-----
n (modulus)      : 2479
phi(n)           : 2376
Decryption key d : 2137
Decrypted M      : 1234
-----

```

Q4)

```

Computer Security CW2 > RSA > C rsa_q4.c
1  #include <stdio.h>
2  #include <openssl/bn.h>
3
4  int main() {
5      BN_CTX *ctx = BN_CTX_new();
6
7      BIGNUM *S = BN_new();
8      BIGNUM *n = BN_new();
9      BIGNUM *e = BN_new();
10     BIGNUM *M_verify = BN_new();
11
12     // Set known values
13     BN_dec2bn(&S, "1459");
14     BN_dec2bn(&n, "2479");
15     BN_dec2bn(&e, "169");
16
17     // Compute M' = S^e mod n
18     BN_mod_exp(M_verify, S, e, n, ctx);
19
20     // Print result
21     char *M_str = BN_bn2dec(M_verify);
22     printf("Verified Message from Signature: %s\n", M_str);
23
24     // Free
25     OPENSSL_free(M_str);
26     BN_free(S); BN_free(n); BN_free(e); BN_free(M_verify);
27     BN_CTX_free(ctx);
28
29     return 0;
30 }

```

Output :

```

>>> Compiling rsa_q4.c
>>> Running rsa_q4
-----
Verified Message from Signature: 1233
-----

```

Comments : The signature  $S=1459$  was verified using Alice's public key ( $e=169, n=2479$ ), resulting in  $M'=1233$ .

Since the expected message was  $M=1234$ , and the recovered message from the signature does **not** match ( $M' \neq M$ ), we conclude that:

This signature was not generated using Alice's private key.

Therefore, it cannot be considered authentic, and may have been forged or created with a different key.

# Diffie-Hellman

Q1)

```
Computer Security CW2 > Diffie-Hellman > C dh_q1.c
1  #include <stdio.h>
2  #include <openssl/bn.h>
3
4  int main() {
5      BN_CTX *ctx = BN_CTX_new();
6
7      BIGNUM *p = BN_new();
8      BIGNUM *g = BN_new();
9      BIGNUM *xa = BN_new();
10     BIGNUM *xb = BN_new();
11     BIGNUM *ya = BN_new();
12     BIGNUM *yb = BN_new();
13     BIGNUM *kab1 = BN_new();
14     BIGNUM *kab2 = BN_new();
15
16     // Set values
17     BN_dec2bn(&p, "773");
18     BN_dec2bn(&g, "200");
19     BN_dec2bn(&xa, "333");
20     BN_dec2bn(&xb, "603");
21
22     // y_a = g^x_a mod p
23     BN_mod_exp(ya, g, xa, p, ctx);
24
25     // y_b = g^x_b mod p
26     BN_mod_exp(yb, g, xb, p, ctx);
27
28     // K_ab = y_b^x_a mod p
29     BN_mod_exp(kab1, yb, xa, p, ctx);
30
31     // (Optional: K_ab from other side to double-check)
32     BN_mod_exp(kab2, ya, xb, p, ctx);
33
34     // Print results
35     char *ya_str = BN_bn2dec(ya);
36     char *yb_str = BN_bn2dec(yb);
37     char *kab_str = BN_bn2dec(kab1);
38     char *kab2_str = BN_bn2dec(kab2);
39
40     printf("Alice's public key (y_a): %s\n", ya_str);
41     printf("Bob's public key (y_b): %s\n", yb_str);
42     printf("Shared secret key (K_ab): %s\n", kab_str);
43     printf("Verification (K_ab from other side): %s\n", kab2_str);
44
45     // Cleanup
46     OPENSSL_free(ya_str);
47     OPENSSL_free(yb_str);
48     OPENSSL_free(kab_str);
49     OPENSSL_free(kab2_str);
50     BN_free(p); BN_free(g); BN_free(xa); BN_free(xb);
51     BN_free(ya); BN_free(yb); BN_free(kab1); BN_free(kab2);
52     BN_CTX_free(ctx);
53
54     return 0;
55 }
```

Output:

```
>>> Compiling dh_q1.c
>>> Running dh_q1
-----
Alice's public key (y_a): 164
Bob's public key (y_b): 405
Shared secret key (K_ab): 476
Verification (K_ab from other side): 476
-----
```

Q2)

```

Computer Security CW2 > Diffie-Hellman > C dh_q2.c
1  #include <stdio.h>
2  #include <openssl/bn.h>
3
4  int main() {
5      BN_CTX *ctx = BN_CTX_new();
6
7      BIGNUM *p = BN_new();
8      BIGNUM *g = BN_new();
9      BIGNUM *xa = BN_new();
10     BIGNUM *xb = BN_new();
11     BIGNUM *ya = BN_new();
12     BIGNUM *yb = BN_new();
13     BIGNUM *kab1 = BN_new();
14     BIGNUM *kab2 = BN_new();
15
16     // Set input values for DH Q2
17     BN_dec2bn(&p, "1553");
18     BN_dec2bn(&g, "307");
19     BN_dec2bn(&xa, "1333");
20     BN_dec2bn(&xb, "807");
21
22     // Compute public keys
23     BN_mod_exp(ya, g, xa, p, ctx); // y_a = g^x_a mod p
24     BN_mod_exp(yb, g, xb, p, ctx); // y_b = g^x_b mod p
25
26     // Compute shared keys
27     BN_mod_exp(kab1, yb, xa, p, ctx); // K_ab = y_b^x_a mod p
28     BN_mod_exp(kab2, ya, xb, p, ctx); // K_ab (from other side)
29
30     // Output
31     char *ya_str = BN_bn2dec(ya);
32     char *yb_str = BN_bn2dec(yb);
33     char *kab1_str = BN_bn2dec(kab1);
34     char *kab2_str = BN_bn2dec(kab2);
35
36     printf("Alice's public key (y_a): %s\n", ya_str);
37     printf("Bob's public key (y_b): %s\n", yb_str);
38     printf("Shared secret key (K_ab): %s\n", kab1_str);
39     printf("Verification (K_ab from other side): %s\n", kab2_str);
40
41     // Cleanup
42     OPENSSL_free(ya_str);
43     OPENSSL_free(yb_str);
44     OPENSSL_free(kab1_str);
45     OPENSSL_free(kab2_str);
46     BN_free(p); BN_free(g); BN_free(xa); BN_free(xb);
47     BN_free(ya); BN_free(yb); BN_free(kab1); BN_free(kab2);
48     BN_CTX_free(ctx);
49
50     return 0;
51 }

```

Output:

```

>>> Compiling dh_q2.c
>>> Running dh_q2
-----
Alice's public key (y_a): 1143
Bob's public key (y_b): 124
Shared secret key (K_ab): 245
Verification (K_ab from other side): 245
-----

```

# El-Gamal

Q1)

```
Computer Security CW2 > El-Gamal > C eg_q1.c
1  #include <stdio.h>
2  #include <openssl/bn.h>
3
4  int main() {
5      BN_CTX *ctx = BN_CTX_new();
6
7      BIGNUM *p = BN_new();
8      BIGNUM *g = BN_new();
9      BIGNUM *xa = BN_new(); // Alice's ephemeral message key
10     BIGNUM *xb = BN_new(); // Bob's secret key
11     BIGNUM *yb = BN_new(); // Bob's public key
12     BIGNUM *K = BN_new(); // Shared secret
13     BIGNUM *C1 = BN_new();
14     BIGNUM *C2 = BN_new();
15     BIGNUM *M = BN_new();
16     BIGNUM *temp = BN_new();
17
18     // Set values
19     BN_dec2bn(&p, "773");
20     BN_dec2bn(&g, "200");
21     BN_dec2bn(&xa, "333"); // Ephemeral key generated by Alice (sender)
22     BN_dec2bn(&xb, "603"); // Bob's private key (receiver)
23     BN_dec2bn(&M, "321");
24
25     // Compute Bob's public key: y_b = g^x_b mod p
26     BN_mod_exp(yb, g, xb, p, ctx);
27
28     // C1 = g^x_a mod p
29     BN_mod_exp(C1, g, xa, p, ctx);
30
31     // K = y_b^x_a mod p
32     BN_mod_exp(K, yb, xa, p, ctx);
33
34     // C2 = (M * K) mod p
35     BN_mul(temp, M, K, ctx);
36     BN_mod(C2, temp, p, ctx);
37
38     // Output
39     char *yb_str = BN_bn2dec(yb);
40     char *C1_str = BN_bn2dec(C1);
41     char *C2_str = BN_bn2dec(C2);
42     char *K_str = BN_bn2dec(K);
43
44     printf("Bob's public key (y_b): %s\n", yb_str);
45     printf("Ephemeral key (x_a): 333\n");
46     printf("Ephemeral message key (K): %s\n", K_str);
47     printf("Ciphertext (C1, C2): (%s, %s)\n", C1_str, C2_str);
48
49     // Cleanup
50     OPENSSL_free(yb_str);
51     OPENSSL_free(C1_str);
52     OPENSSL_free(C2_str);
53     OPENSSL_free(K_str);
54     BN_free(p); BN_free(g); BN_free(xa); BN_free(xb);
55     BN_free(yb); BN_free(K); BN_free(C1); BN_free(C2); BN_free(M); BN_free(temp);
56     BN_CTX_free(ctx);
57
58     return 0;
59 }
```

Output:

```
>>> Compiling eg_q1.c
>>> Running eg_q1
-----
Bob's public key (y_b): 405
Ephemeral key (x_a): 333
Ephemeral message key (K): 476
Ciphertext (C1, C2): (164, 515)
-----
```

Q2)

```

Computer Security CW2 > El-Gamal > C eg_q2.c
1  #include <stdio.h>
2  #include <openssl/bn.h>
3
4  int main() {
5      BN_CTX *ctx = BN_CTX_new();
6
7      BIGNUM *p = BN_new();
8      BIGNUM *xb = BN_new();    // Bob's private key
9      BIGNUM *C1 = BN_new();
10     BIGNUM *C2 = BN_new();
11     BIGNUM *K = BN_new();     // Shared key
12     BIGNUM *Kinv = BN_new();  // Inverse of K
13     BIGNUM *M = BN_new();     // Decrypted message
14     BIGNUM *temp = BN_new();
15
16     // Set values from question
17     BN_dec2bn(&p, "6469");
18     BN_dec2bn(&xb, "4127");
19     BN_dec2bn(&C1, "3533");
20     BN_dec2bn(&C2, "3719");
21
22     // K = C1^xb mod p
23     BN_mod_exp(K, C1, xb, p, ctx);
24
25     // K^-1 mod p
26     BN_mod_inverse(Kinv, K, p, ctx);
27
28     // M = (C2 * K^-1) mod p
29     BN_mul(temp, C2, Kinv, ctx);
30     BN_mod(M, temp, p, ctx);
31
32     // Output
33     char *K_str = BN_bn2dec(K);
34     char *Kinv_str = BN_bn2dec(Kinv);
35     char *M_str = BN_bn2dec(M);
36
37     printf("Recovered shared key (K): %s\n", K_str);
38     printf("Inverse of shared key (K^-1): %s\n", Kinv_str);
39     printf("Decrypted Message (M): %s\n", M_str);
40
41     // Cleanup
42     OPENSSL_free(K_str);
43     OPENSSL_free(Kinv_str);
44     OPENSSL_free(M_str);
45     BN_free(p); BN_free(xb); BN_free(C1); BN_free(C2);
46     BN_free(K); BN_free(Kinv); BN_free(M); BN_free(temp);
47     BN_CTX_free(ctx);
48
49     return 0;
50 }

```

Output:

```

>>> Compiling eg_q2.c
>>> Running eg_q2
-----
Recovered shared key (K): 4125
Inverse of shared key (K^-1): 1758
Decrypted Message (M): 4312
-----

```