



VIT[®]

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

B.Tech. Winter Semester 2024-25
School Of Computer Science and Engineering
(SCOPE)

Digital Assignment - VI

Cryptography and Network Security Lab

Apurva Mishra: 22BCE2791

Date: 28 March, 2025

Contents

1 Diffie Hellman Key Exchange	3
1.1 Code	3
1.2 Output	5
2 RSA	5
2.1 Code	5
2.2 Output	8
3 Point Doubling in ECC	8
3.1 Code	8
3.2 Output	11
4 Negative Point in ECC	11
4.1 Code	11
4.2 Output	13

5 Elgamal Cryptography 13

5.1 Code 13

5.2 Output 16

6 RC4 16

1 Diffie Hellman Key Exchange

1.1 Code

Code 0: main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4  #include <math.h>
5
6  // Function for modular exponentiation ( $a^b \bmod p$ )
7  long long power(long long base, long long exp, long long mod) {
8      long long res = 1;
9      base = base % mod;
10     while (exp > 0) {
11         if (exp % 2 == 1) res = (res * base) % mod;
12         base = (base * base) % mod;
13         exp = exp / 2;
14     }
15     return res;
16 }
17
18 // Function to find a prime number (for simplicity, not cryptographically
19 // strong)
20 long long findPrime() {
21     long long primes[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41,
22     43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}; //small list of primes
23     return primes[rand() % (sizeof(primes) / sizeof(primes[0]))];
24 }
25
26 // Function to find a primitive root modulo p (not fully robust, for
27 // simplicity)
28 long long findPrimitiveRoot(long long p) {
29     // For simplicity, using 2. In real implementation, more robust
30     // algorithms are needed.
31     return 2;
32 }
33
34 int main() {
35     srand(time(0));
36
37     // Publicly known parameters
38     long long p = findPrime(); // Large prime number
39     long long g = findPrimitiveRoot(p); // Primitive root modulo p
40
41     printf("Publicly known parameters:\n");
42     printf("Prime (p): %lld\n", p);
43     printf("Primitive root (g): %lld\n\n", g);
44
45     // Alice's private key (a)
46     long long a = rand() % (p - 2) + 2; // Random number between 2 and p-1
47 }
```

```

44 // Bob's private key (b)
45 long long b = rand() % (p - 2) + 2; // Random number between 2 and p-1
46
47 // Alice computes A = g^a mod p
48 long long A = power(g, a, p);
49
50 // Bob computes B = g^b mod p
51 long long B = power(g, b, p);
52
53 printf("Alice's private key (a): %lld\n", a);
54 printf("Bob's private key (b): %lld\n\n", b);
55
56 printf("Alice computes A: %lld\n", A);
57 printf("Bob computes B: %lld\n\n", B);
58
59 // Alice computes shared key K_A = B^a mod p
60 long long KA = power(B, a, p);
61
62 // Bob computes shared key K_B = A^b mod p
63 long long KB = power(A, b, p);
64
65 printf("Alice's shared key (KA): %lld\n", KA);
66 printf("Bob's shared key (KB): %lld\n\n", KB);
67
68 if (KA == KB) {
69     printf("Shared keys match! Diffie-Hellman key exchange successful.
\n");
70 } else {
71     printf("Shared keys do not match! Diffie-Hellman key exchange
failed.\n");
72 }
73
74 return 0;
75 }

```

1.2 Output

```
da/ass6/diffi via C v16.0.0-clang took 9s
> just run
cc main.c -o main
./main
Publicly known parameters:
Prime (p): 19
Primitive root (g): 2

Alice's private key (a): 18
Bob's private key (b): 17

Alice computes A: 1
Bob computes B: 10

Alice's shared key (KA): 1
Bob's shared key (KB): 1

Shared keys match! Diffie-Hellman key exchange successful.

da/ass6/diffi via C v16.0.0-clang
> 
```

2 RSA

2.1 Code

Code 0: main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4  #include <stdbool.h>
5
6  // Function to check if a number is prime (using a basic method)
7  bool is_prime(long long n) {
8      if (n <= 1) return false;
9      if (n <= 3) return true;
10     if (n % 2 == 0 || n % 3 == 0) return false;
11
12     for (long long i = 5; i * i <= n; i += 6) {
13         if (n % i == 0 || n % (i + 2) == 0) return false;
14     }
15     return true;
16 }
```

```

17
18 // Function to generate a random prime number within a specified range.
19 long long chose_prim() {
20     long long min = 1000; // Example minimum range
21     long long max = 2000; // Example maximum range
22
23     if (min < 2) {
24         min = 2;
25     }
26
27     long long randomNum;
28     srand(time(NULL));
29
30     while (1) {
31         randomNum = (rand() % (max - min + 1)) + min;
32         if (is_prime(randomNum)) {
33             return randomNum;
34         }
35     }
36 }
37
38 // Function to calculate the greatest common divisor (GCD)
39 long long gcd(long long a, long long b) {
40     while (b != 0) {
41         long long temp = b;
42         b = a % b;
43         a = temp;
44     }
45     return a;
46 }
47
48 // Function to choose a valid 'e' (public exponent)
49 long long chose_E(long long phi) {
50     long long e = rand() % (phi - 2) + 2; // Ensure e is between 2
and phi-1
51
52     while (gcd(e, phi) != 1) {
53         e = rand() % (phi - 2) + 2;
54     }
55     return e;
56 }
57
58 // Function to calculate the modular inverse
59 long long mod_inverse(long long e, long long phi) {
60     long long d = 1;
61     while ((e * d) % phi != 1) {
62         d++;
63     }
64     return d;
65 }
66
67 // Function to encrypt the message
68 long long encrypt(long long msg, long long e, long long n) {

```

```

69     long long result = 1;
70     msg = msg % n;
71     while (e > 0) {
72         if (e % 2 == 1) {
73             result = (result * msg) % n;
74         }
75         msg = (msg * msg) % n;
76         e = e / 2;
77     }
78     return result;
79 }
80
81 // Function to decrypt the message
82 long long decrypt(long long ciphertext, long long d, long long n) {
83     long long result = 1;
84     ciphertext = ciphertext % n;
85     while (d > 0) {
86         if (d % 2 == 1) {
87             result = (result * ciphertext) % n;
88         }
89         ciphertext = (ciphertext * ciphertext) % n;
90         d = d / 2;
91     }
92     return result;
93 }
94
95 int main(void) {
96     srand(time(0));
97
98     long long p = chose_prim();
99     long long q = chose_prim();
100
101     while (q == p) {
102         q = chose_prim();
103     }
104
105     long long n = p * q;
106     long long phi = (p - 1) * (q - 1);
107
108     long long e = chose_E(phi);
109     long long d = mod_inverse(e, phi);
110
111     long long msg = 0;
112     printf("Enter msg to encrypt: ");
113     scanf("%lld", &msg);
114
115     printf("p: %lld, q: %lld, n: %lld, phi: %lld, e: %lld, d: %lld\n", p,
q, n, phi, e, d);
116
117     long long ciphertext = encrypt(msg, e, n);
118     printf("Cypher Text: %lld\n", ciphertext);
119
120     long long decrypted_msg = decrypt(ciphertext, d, n);

```

```

121     printf("Decrypted Text: %lld\n", decrypted_msg);
122
123     return 0;
124 }
125

```

2.2 Output

```

da/ass6/rsa via C v16.0.0-clang took 12s
> just run
zig cc main.c -o main
./main
Enter msg to encrypt: 23
p: 1811, q: 1193, n: 2160523, phi: 2157520, e: 1165913, d: 1027417
Cypher Text: 1704127
Decrypted Text: 23

da/ass6/rsa via C v16.0.0-clang took 3s
> |

```

3 Point Doubling in ECC

3.1 Code

Code 0: main.c

```

1  #include <stdint.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5
6  #define N 256
7
8  struct Point {
9     int x;
10    int y;
11 };
12
13 void print_point(struct Point p) {
14     printf("Point:\n");
15     printf("  X: %d\n  Y: %d\n", p.x, p.y);
16 }
17
18 struct Curve {
19     int a;
20     int b;
21     int p;
22 };
23

```



```

24 void print_curve(struct Curve c) {
25     printf("Curve:\n");
26     printf("  a: %d\n  b: %d\n  p: %d\n", c.a, c.b, c.p);
27 }
28
29 int gcd(int a, int b)
30 {
31     // Find Minimum of a and b
32     int result = ((a < b) ? a : b);
33     while (result > 0) {
34         // Check if both a and b are divisible by result
35         if (a % result == 0 && b % result == 0) {
36             break;
37         }
38         result--;
39     }
40     // return gcd of a and b
41     return result;
42 }
43
44 int modInverse(int a, int p) {
45     if (p < 0) {
46         perror("The prime can not be negative");
47         exit(0);
48     }
49
50     int flag = 0;
51     if (a < 0) {
52         a = -a;
53         flag = 1;
54     }
55
56     int i = 0;
57     while (i < 100) {
58         if ((a*i) % p == 1) {
59             break;
60         }
61
62         i += 1;
63     }
64
65     if (flag) {
66         return p - i;
67     } else {
68         return i;
69     }
70 }
71
72 int mod(int num, int p) {
73     if (p < 0) {
74         perror("The prime can not be negative");
75         exit(0);
76     }

```

```

77     if (num < 0) {
78         int tmp = (-num) % p;
79         return p - tmp;
80     } else {
81         return num % p;
82     }
83 }
84
85 int mod_frac(int num, int den, int p) {
86     int a = mod(num, p);
87     int b = modInverse(den, p);
88
89     return (a*b) % p;
90 }
91
92 struct Point ecc_double(struct Point P, struct Point Q, struct Curve E) {
93     int lambda = 0;
94     if (P.x == Q.x && P.y == Q.y) {
95         lambda = mod_frac((3*P.x*P.x) + E.a, (2*P.y), E.p);
96     } else {
97         lambda = mod_frac((Q.y - P.y), (Q.x - P.x), E.p);
98     }
99
100
101     struct Point R = {0};
102
103     int tmp_x = (lambda*lambda) - P.x - Q.x;
104     R.x = mod(tmp_x, E.p);
105
106     int tmp_y = (lambda*(P.x - R.x)) - P.y;
107     R.y = mod(tmp_y, E.p);
108
109     return R;
110 }
111
112 int main(void) {
113     struct Curve E = {0};
114     struct Point P = {0};
115     struct Point Q = {0};
116
117     printf("Inputs: \n");
118     printf("Curve: a b p: ");
119     scanf("%d %d %d", &E.a, &E.b, &E.p);
120
121     printf("Point P: x y: ");
122     scanf("%d %d", &P.x, &P.y);
123     printf("Point Q: x y: ");
124     scanf("%d %d", &Q.x, &Q.y);
125
126     print_curve(E);
127     print_point(P);
128     print_point(Q);
129

```

```

130     struct Point R = ecc_double(P, Q, E);
131
132     printf("P + Q = \n");
133     print_point(R);
134     return 0;
135 }
136

```

3.2 Output

```

da/ass6/doub_ecc via C v16.0.0-clang took 18s
> just run
cc main.c -o main
./main
Inputs:
Curve: a b p: 1 1 23
Point P: x y: 3 10
Point Q: x y: 3 10
Curve:
  a: 1
  b: 1
  p: 23
Point:
  X: 3
  Y: 10
Point:
  X: 3
  Y: 10
P + Q =
Point:
  X: 7
  Y: 12

```

4 Negative Point in ECC

4.1 Code

Code 0: main.c

```

1  #include <stdint.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5
6  #define N 256
7
8  struct Point {
9      int x;

```

```

10     int y;
11 };
12
13 void print_point(struct Point p) {
14     printf("Point:\n");
15     printf("  X: %d\n  Y: %d\n", p.x, p.y);
16 }
17
18 struct Curve {
19     int a;
20     int b;
21     int p;
22 };
23
24 void print_curve(struct Curve c) {
25     printf("Curve:\n");
26     printf("  a: %d\n  b: %d\n  p: %d\n", c.a, c.b, c.p);
27 }
28
29 int mod(int num, int p) {
30     if (p < 0) {
31         perror("The prime can not be negative");
32         exit(0);
33     }
34     if (num < 0) {
35         int tmp = (-num) % p;
36         return p - tmp;
37     } else {
38         return num % p;
39     }
40 }
41
42 struct Point ecc_neg(struct Point P, struct Curve E) {
43     struct Point R = {0};
44
45     R.x = P.x;
46     R.y = mod(-P.y, E.p);
47
48     return R;
49 }
50
51 int main(void) {
52     struct Curve E = {0};
53     struct Point P = {0};
54
55     printf("Inputs: \n");
56     printf("Curve: a b p: ");
57     scanf("%d %d %d", &E.a, &E.b, &E.p);
58
59     printf("Point P: x y: ");
60     scanf("%d %d", &P.x, &P.y);
61
62     print_curve(E);

```

```

63     print_point(P);
64
65     struct Point R = ecc_neg(P, E);
66
67     printf("\n-P = \n");
68     print_point(R);
69     return 0;
70 }
71

```

4.2 Output

```

da/ass6/neg_ecc via C v16.0.0-clang took 2m8s
> just run
cc main.c -o main
./main
Inputs:
Curve: a b p: 1 1 23
Point P: x y: 3 10
Curve:
  a: 1
  b: 1
  p: 23
Point:
  X: 3
  Y: 10
-10
23 10

-P =
Point:
  X: 3
  Y: 13

```

5 Elgamal Cryptography

5.1 Code

Code 0: main.c

```

1  #include <math.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>

```

```

5
6 struct pub_key {
7     int q;
8     int alpha;
9     int Y;
10 };
11
12 struct priv_key {
13     int x;
14     int q;
15 };
16
17 struct enc_msg {
18     int c1;
19     int c2;
20 };
21
22 void print_enc_msg(struct enc_msg E) {
23     printf("Encrypted Message:\n");
24     printf("  C1: %d, C2: %d\n", E.c1, E.c2);
25 }
26
27 int gcd(int a, int b) {
28     if (a < b)
29         return gcd(b, a);
30     else if (a % b == 0)
31         return b;
32     else
33         return gcd(b, a % b);
34 }
35
36 int gen_key(int q) {
37     int key = rand() % q;
38     while (gcd(q, key) != 1) {
39         key = rand() % q + 1;
40     }
41     return key;
42 }
43
44 int power(int a, int b, int c) {
45     int x = 1;
46     int y = a;
47     while (b > 0) {
48         if (b % 2 != 0) {
49             x = (x * y) % c;
50         }
51         y = (y * y) % c;
52         b = b / 2;
53     }
54     return x % c;
55 }
56
57 struct enc_msg encrypt(int msg, struct pub_key P) {

```

```

58     if (msg > P.q) {
59         perror("Msg cant not be more than q");
60         exit(0);
61     }
62
63     int k = rand() % P.q;
64
65     int K = power(P.Y, k, P.q);
66
67     int c1 = power(P.alpha, k, P.q);
68     int c2 = (K * msg) % P.q;
69
70     struct enc_msg E = {c1, c2};
71
72     return E;
73 }
74
75 int modInverse(int a, int p) {
76     if (p < 0) {
77         perror("The prime cna not be negative");
78         exit(0);
79     }
80
81     int flag = 0;
82     if (a < 0) {
83         a = -a;
84         flag = 1;
85     }
86
87     int i = 0;
88     while (i < 100) {
89         if ((a*i) % p == 1) {
90             break;
91         }
92
93         i += 1;
94     }
95
96     if (flag) {
97         return p - i;
98     } else {
99         return i;
100     }
101 }
102
103 int decrypt(struct enc_msg E, struct priv_key P) {
104     int K = power(E.c1, P.x, P.q);
105     int tmp = modInverse(K, P.q);
106
107     int msg = (E.c2 * tmp) % P.q;
108
109     return msg;
110 }

```

```

111
112 int main() {
113     int msg;
114     printf("Enter the message: ");
115     scanf("%d", &msg);
116     int q = rand() % 91;
117     int alpha = rand() % (q - 2) + 2;
118     int X = gen_key(q);
119
120     struct priv_key priv_key = {X, q};
121
122     int Y = power(alpha, X, q);
123     struct pub_key pub_key= {q, alpha, Y};
124
125     struct enc_msg E = encrypt(msg, pub_key);
126
127     int dec_msg = decrypt(E, priv_key);
128
129     printf("Decrypted Message: %d\n", dec_msg);
130     return 0;
131 }
132

```

5.2 Output

```

da/ass6/elgama1 via C v16.0.0-clang took 25s
> just run
cc main.c -o main
./main
Enter the message: 23
Encrypted Message:
  C1: 43, C2: 2
Decrypted Message: 23

da/ass6/elgama1 via C v16.0.0-clang
> |

```

6 RC4

Done in Assessemnt 5