Cryptography and Network Security
Lab Assignment - II

Program 1: Fast Exponentiation using Successive Square

Code:

```python
print("Siddhanth Monnapp\t22BCE3061")
a=int(input("Enter base: "))
i=int(input("Enter exponent: "))
n=int(input("Enter modulo: "))
i_bin = bin(i)[2:]
print(f"Binary representation of x: {i_bin}")
y=1
print("y=a^x mod n\ta=a^2 mod n")
for bit in i_bin[::-1]:
    if bit=='1':
        y=(a*y)%n
    a=(a*a)%n
    print(f"y={y}\ta={a}")
print(f"Fast Exponentiation Result: {y}")
```

Input/Output:

```
Siddhanth Monnappa      22BCE3061
Enter base: 23
Enter exponent: 67
Enter modulo: 67
Binary representation of x: 1000011
y=a^x mod n     a=a^2 mod n
y=23    a=60
y=40    a=49
y=40    a=56
y=40    a=54
y=40    a=35
y=40    a=19
y=23    a=26
Fast Exponentiation Result: 23
```

```
Siddhanth Monnappa      22BCE3061
Enter base: 23
Enter exponent: 66
Enter modulo: 67
Binary representation of x: 1000010
y=a^x mod n     a=a^2 mod n
y=1     a=60
y=60    a=49
y=60    a=56
y=60    a=54
y=60    a=35
y=60    a=19
y=1     a=26
Fast Exponentiation Result: 1
```

```
Siddhanth Monnappa      22BCE3061
Enter base: 63
Enter exponent: 65
Enter modulo: 128
Binary representation of x: 1000001
y=a^x mod n     a=a^2 mod n
y=63    a=1
y=63    a=1
y=63    a=1
y=63    a=1
y=63    a=1
y=63    a=1
y=63    a=1
Fast Exponentiation Result: 63
```

```
Siddhanth Monnappa      22BCE3061
Enter base: 11
Enter exponent: 23
Enter modulo: 21
Binary representation of x: 10111
y=a^x mod n     a=a^2 mod n
y=11    a=16
y=8     a=4
y=11    a=16
y=11    a=4
y=2     a=16
Fast Exponentiation Result: 2
```

Program 2: Extended Euclidean s, t, d

Code:

```python
print("Siddhanth Monnappa\t22BCE3061")
r1=int(input("Enter a value: "))
r2=int(input("Enter b value: "))
s1=1
s2=0
t1=0
t2=1
print(f"r1: {r1}\tr2: {r2}\ts1: {s1}\ts2: {s2}\tt1: {t1}\tt2: {t2}")
while(r2>0):
    q=r1//r2

    r=r1-q*r2
    r1=r2
    r2=r

    s=s1-q*s2
    s1=s2
    s2=s

    t=t1-q*t2
    t1=t2
    t2=t
    print(f"r1: {r1}\tr2: {r2}\ts1: {s1}\ts2: {s2}\tt1: {t1}\tt2: {t2}")

d=r1
s=s1
t=t1

print(f"gcd = d value: {d}")
print(f"s value: {s}")
print(f"t value: {t}")
```

Input/Output:

```
Siddhanth Monnappa      22BCE3061
Enter a value: 428
Enter b value: 23
r1: 428 r2: 23  s1: 1   s2: 0   t1: 0   t2: 1
r1: 23  r2: 14  s1: 0   s2: 1   t1: 1   t2: -18
r1: 14  r2: 9   s1: 1   s2: -1  t1: -18 t2: 19
r1: 9   r2: 5   s1: -1  s2: 2   t1: 19  t2: -37
r1: 5   r2: 4   s1: 2   s2: -3  t1: -37 t2: 56
r1: 4   r2: 1   s1: -3  s2: 5   t1: 56  t2: -93
r1: 1   r2: 0   s1: 5   s2: -23 t1: -93 t2: 428
gcd = d value: 1
s value: 5
t value: -93
```

```
Siddhanth Monnappa      22BCE3061
Enter a value: 125
Enter b value: 23
r1: 125 r2: 23  s1: 1   s2: 0   t1: 0   t2: 1
r1: 23  r2: 10  s1: 0   s2: 1   t1: 1   t2: -5
r1: 10  r2: 3   s1: 1   s2: -2  t1: -5  t2: 11
r1: 3   r2: 1   s1: -2  s2: 7   t1: 11  t2: -38
r1: 1   r2: 0   s1: 7   s2: -23 t1: -38 t2: 125
gcd = d value: 1
s value: 7
t value: -38
```

```
Siddhanth Monnappa      22BCE3061
Enter a value: 125634
Enter b value: 2356
r1: 125634      r2: 2356        s1: 1   s2: 0   t1: 0   t2: 1
r1: 2356        r2: 766 s1: 0   s2: 1   t1: 1   t2: -53
r1: 766 r2: 58  s1: 1   s2: -3  t1: -53 t2: 160
r1: 58  r2: 12  s1: -3  s2: 40  t1: 160 t2: -2133
r1: 12  r2: 10  s1: 40  s2: -163        t1: -2133       t2: 8692
r1: 10  r2: 2   s1: -163        s2: 203 t1: 8692        t2: -10825
r1: 2   r2: 0   s1: 203 s2: -1178       t1: -10825      t2: 62817
gcd = d value: 2
s value: 203
t value: -10825
```

Program 3: Multiplicative Inverse using Extended Euclidean

Code:

```python
def gcd(a,b):
    if(b==0):
        return a
    else:
        return gcd(b,a%b)
print("Siddhanth Monnappa\t22BCE3061")
a=int(input("Enter a value: "))
n=int(input("Enter n value: "))
if(gcd(a,n)!=1):
    print("Extended Euclidean's Multiplicative Inverse not possible")
    exit()
r1=n
r2=a
t1=0
t2=1
print(f"r1: {r1}\tr2: {r2}\tt1: {t1}\tt2: {t2}")
while(r2>0):
    q=r1//r2
    r=r1-q*r2
    r1=r2
    r2=r

    t=t1-q*t2
    t1=t2
    t2=t
    print(f"r1: {r1} \tr2: {r2}\tt1: {t1}\tt2: {t2}")

if(r1==1):
    if(t1<0):
        t=t1+t2
    else:
        t=t1

print(f"Multiplicative inverse of {a} mod {n} = {t}")
```

Input/Output:

```
Siddhanth Monnappa      22BCE3061
Enter a value: 23
Enter n value: 428
r1: 428 r2: 23  t1: 0   t2: 1
r1: 23  r2: 14  t1: 1   t2: -18
r1: 14  r2: 9   t1: -18 t2: 19
r1: 9   r2: 5   t1: 19  t2: -37
r1: 5   r2: 4   t1: -37 t2: 56
r1: 4   r2: 1   t1: 56  t2: -93
r1: 1   r2: 0   t1: -93 t2: 428
Multiplicative inverse of 23 mod 428 = 335
```

```
Siddhanth Monnappa      22BCE3061
Enter a value: 24
Enter n value: 428
Extended Euclidean's Multiplicative Inverse not possible
```

```
Siddhanth Monnappa      22BCE3061
Enter a value: 2356
Enter n value: 12563
r1: 12563       r2: 2356        t1: 0   t2: 1
r1: 2356        r2: 783 t1: 1   t2: -5
r1: 783         r2: 7   t1: -5  t2: 16
r1: 7   r2: 6   t1: 16  t2: -1781
r1: 6   r2: 1   t1: -1781       t2: 1797
r1: 1   r2: 0   t1: 1797        t2: -12563
Multiplicative inverse of 2356 mod 12563 = 1797
```

Program 4: Multiplicative Inverse using Fermat's Theorem

Code:

```python
def gcd(a,b):
    if(b==0):
        return a
    else:
        return gcd(b,a%b)
print("Siddhanth Monnappa\t22BCE3061")
a=int(input("Enter a value: "))
n=int(input("Enter n value: "))
for i in range(2,n//2):
    if(n%i==0):
        print("Not possible using Fermat's Inverse")
        exit()
if(gcd(a,n)!=1):
    print("Not possible using Fermat's Inverse")
    exit()
a_inv=a**(n-2)%n
print(f"Multiplicative Inverse using Fermat's Theorm: {a_inv}")

```

Input/Output:

```
Siddhanth Monnappa      22BCE3061
Enter a value: 23
Enter n value: 428
Not possible using Fermat's Inverse
```

```
Siddhanth Monnappa      22BCE3061
Enter a value: 23
Enter n value: 1949
Multiplicative Inverse using Fermat's Theorm: 339
```

```
Siddhanth Monnappa      22BCE3061
Enter a value: 7
Enter n value: 23
Multiplicative Inverse using Fermat's Theorm: 10
```

Program 5: Multiplicative Inverse using Euler's Theorem

Code:

```python
def gcd(a,b):
        if(b==0):
            return a
        else:
            return gcd(b,a%b)
def phi(n):
    res=1
    for i in range(2,n):
        if(gcd(n,i)==1):
            res+=1
    return res

print("Siddhanth Monnappa\t22BCE3061")
a=int(input("Enter a value: "))
m=int(input("Enter m value: "))
if(gcd(a,m)!=1):
    print("Euler's Multiplicative Inverse not possible")
    exit()
a_inv=a**(phi(m)-1)%m
print(f"Multiplicative Inverse using Euler's Theorm: {a_inv}")
```

Input/Output:

```
Siddhanth Monnappa      22BCE3061
Enter a value: 23
Enter m value: 428
Multiplicative Inverse using Euler's Theorm: 335
```

```
Siddhanth Monnappa      22BCE3061
Enter a value: 23
Enter m value: 1949
Multiplicative Inverse using Euler's Theorm: 339
```

```
Siddhanth Monnappa      22BCE3061
Enter a value: 7
Enter m value: 23
Multiplicative Inverse using Euler's Theorm: 10
```

Program 6: Chinese Remainder Theorem

Code:

```python
print("Siddhanth Monnappa\t22BCE3061")
n = int(input("Enter the number of equations: "))

a = []
m = []
for i in range(n):
    a.append(int(input(f"Enter a{i+1}: ")))
    m.append(int(input(f"Enter m{i+1}: ")))

M = 1
for mi in m:
    M *= mi

x = 0
for i in range(n):
    Mi = M // m[i]
    try:
        Mi_inv = pow(Mi, -1, m[i])
    except:
        print("Inverse does not exist")
        exit()
    x += a[i] * Mi * Mi_inv

x = x % M
print(f"The solution x is: {x}")
```

Input/Output:

```
Siddhanth Monnappa      22BCE3061
Enter the number of equations: 3
Enter a1: 2
Enter m1: 3
Enter a2: 3
Enter m2: 5
Enter a3: 2
Enter m3: 10
Inverse does not exist
```

```
Siddhanth Monnappa      22BCE3061
Enter the number of equations: 3
Enter a1: 2
Enter m1: 3
Enter a2: 3
Enter m2: 5
Enter a3: 2
Enter m3: 7
The solution x is: 23
```

```
Siddhanth Monnappa      22BCE3061
Enter the number of equations: 3
Enter a1: 3
Enter m1: 13
Enter a2: 5
Enter m2: 10
Enter a3: 7
Enter m3: 23
The solution x is: 835
```

Program 7: Miller Rabin Primality Testing

Code:

```python
print("Siddhanth Monnappa\t22BCE3061")
n = int(input("Enter n value: "))
a = int(input("Enter Base value:"))

n_minus=n-1
k=0
while n_minus%2==0:
    n_minus//=2
    k+=1

m=(n-1)//(2**k)
T=pow(a,m,n)

def miller_rabin_test(T):
    if T==1 or T==n-1:
        return "Prime"

    for i in range(k - 1):
        T = pow(T, 2, n)
        if T == n - 1:
            return "Prime"
        if T == 1:
            return "Composite"

    return "Composite"

print(f"{n} is a {miller_rabin_test(T)} number")


```

Input/Output:

```
Siddhanth Monnappa      22BCE3061
Enter n value: 4033
Enter Base value:2
4033 is a Prime number
```

```
Siddhanth Monnappa      22BCE3061
Enter n value: 561
Enter Base value:2
561 is a Composite number
```

```
Siddhanth Monnappa      22BCE3061
Enter n value: 61
Enter Base value:2
61 is a Prime number
```

```
Siddhanth Monnappa      22BCE3061
Enter n value: 4033
Enter Base value:3
4033 is a Composite number
```

Program 8: P-Box and S-Box Testing for Simplified-DES

Code:

```python
def p8_permutation(bits, table):
    return [bits[i - 1] for i in table]

def s_box_substitution(bits, s_boxes):
    output = []
    for i in range(2):
        block = bits[i*4:(i+1)*4]
        row = int(f"{block[0]}{block[3]}", 2)
        col = int(f"{block[1]}{block[2]}", 2)
        output.extend(f"{s_boxes[i][row][col]:02b}")
    return list(map(int, output))

print("Siddhanth Monnappa\t22BCE3061")
bits = list(map(int, input("Enter 8 bits: ").split()))

p8_table = list(map(int, input("Enter 8 values for P8 Table: ").split()))

s_boxes = []
print("Enter values for S-boxes:")
for i in range(2):
    s_box = []
    for j in range(4):
        s_box.append(list(map(int, input(f"Enter 4 values for S-box {i+1} row {j}: ").split())))
    s_boxes.append(s_box)

permuted_bits = p8_permutation(bits, p8_table)
substituted_bits = s_box_substitution(permuted_bits, s_boxes)
print(f"Permutation: {permuted_bits}")
print(f"S-box substitution: {substituted_bits}")
```

Input/Output:

```
Siddhanth Monnappa      22BCE3061
Enter 8 bits: 1 0 1 1 0 0 1 0
Enter 8 values for P8 Table: 3 5 6 1 8 7 2 4
Enter values for S-boxes:
Enter 4 values for S-box 1 row 0: 1 0 3 2
Enter 4 values for S-box 1 row 1: 3 2 1 0
Enter 4 values for S-box 1 row 2: 0 2 1 3
Enter 4 values for S-box 1 row 3: 3 1 3 2
Enter 4 values for S-box 2 row 0: 0 1 2 3
Enter 4 values for S-box 2 row 1: 2 0 1 3
Enter 4 values for S-box 2 row 2: 3 0 1 0
Enter 4 values for S-box 2 row 3: 2 1 0 3
Permutation: [1, 0, 0, 1, 0, 1, 0, 1]
S-box substitution: [1, 1, 0, 1]
```

```
Siddhanth Monnappa      22BCE3061
Enter 8 bits: 1 1 1 0 0 1 1 1
Enter 8 values for P8 Table: 2 1 3 6 7 8 4 5
Enter values for S-boxes:
Enter 4 values for S-box 1 row 0: 1 0 3 2
Enter 4 values for S-box 1 row 1: 3 2 1 0
Enter 4 values for S-box 1 row 2: 0 2 1 3
Enter 4 values for S-box 1 row 3: 3 1 3 2
Enter 4 values for S-box 2 row 0: 0 1 2 3
Enter 4 values for S-box 2 row 1: 2 0 1 3
Enter 4 values for S-box 2 row 2: 3 0 1 0
Enter 4 values for S-box 2 row 3: 2 1 0 3
Permutation: [1, 1, 1, 1, 1, 1, 0, 0]
S-box substitution: [1, 0, 0, 1]
```