ASSIGNMENT

BIG DATA SECURITY

Submitted By -

Name - Aryan Juneja SAPID - 500108707

Name - Alankrit Tomar SAPID - 500109754

Name - Vansh Malik SAPID - 500101897

Name - Siddharth Singh SAPID - 500101726

Name - Dhruv Sharma SAPID - 500109639

Submitted To –

Dr. Priyabrata Dash Sir

1. **Euler's Totient Function without using existing library function (from Scratch).**

   Euler's Totient Function, denoted as ϕ(n), counts the number of integers from 1 to n that are **coprime** with n (i.e., numbers whose greatest common divisor with n is 1).

```
def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

def euler_totient(n):
    count = 0
    for i in range(1, n + 1):
        if gcd(n, i) == 1:
            count += 1
    return count

n = 10
print(f"Euler's Totient function ϕ({n}) = {euler_totient(n)}")
```

```
Euler's Totient function ϕ(10) = 4
```

2. **Design an RSA cryptosystem from scratch to encrypt a given message and decrypt to the original message. Steps: Generate two large prime numbers: p and q Compute n = p * q and φ(n) = (p-1)*(q-1) Choose e (public exponent) such that 1 < e < φ(n) and gcd(e, φ(n)) = 1 Compute d, the modular inverse of e mod φ(n) Encrypt and decrypt a message.**

```python
def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True

def mod_inverse(e, phi):
    original_phi = phi
    x0, x1 = 0, 1
    while e > 1:
        q = e // phi
        e, phi = phi, e % phi
        x0, x1 = x1 - q * x0, x0
    return x1 + original_phi if x1 < 0 else x1

def generate_keys(p, q):
    if not (is_prime(p) and is_prime(q)):
        raise ValueError("Both numbers must be prime.")
    elif p == q:
        raise ValueError("p and q cannot be equal.")

    n = p * q
    phi = (p - 1) * (q - 1)

    e = 2
    while e < phi:
        if gcd(e, phi) == 1:
            break
        e += 1

    d = mod_inverse(e, phi)

    return ((e, n), (d, n))

def encrypt(message, public_key):
    e, n = public_key
    encrypted = [pow(ord(char), e, n) for char in message]
    return encrypted

def decrypt(ciphertext, private_key):
    d, n = private_key
    decrypted = [chr(pow(char, d, n)) for char in ciphertext]
    return ''.join(decrypted)

p = 61
q = 53
public_key, private_key = generate_keys(p, q)
```

```python
message = "HELLO"
print("Original Message:", message)

cipher = encrypt(message, public_key)
print("Encrypted Message:", cipher)

original = decrypt(cipher, private_key)
print("Decrypted Message:", original)
```

```
Original Message: HELLO
Encrypted Message: [1087, 155, 83, 83, 913]
Decrypted Message: HELLO
```

3. Consider the following medical data.

ZIP Code	Age	Gender	Condition
13053	29	F	Diabetes
13053	29	F	Cancer
13068	45	M	Hypertension
13053	31	M	Flu
13068	47	M	Asthma
13053	33	M	Allergy

Write a program to fix the data set so that it fully satisfies k-anonymity (k=2 or 3).

For K=2

```python
import pandas as pd

data = [
    {'ZIP Code': '13053', 'Age': 29, 'Gender': 'F', 'Condition': 'Diabetes'},
    {'ZIP Code': '13053', 'Age': 29, 'Gender': 'F', 'Condition': 'Cancer'},
    {'ZIP Code': '13068', 'Age': 45, 'Gender': 'M', 'Condition': 'Hypertension'},
    {'ZIP Code': '13053', 'Age': 31, 'Gender': 'M', 'Condition': 'Flu'},
    {'ZIP Code': '13068', 'Age': 47, 'Gender': 'M', 'Condition': 'Asthma'},
    {'ZIP Code': '13053', 'Age': 33, 'Gender': 'M', 'Condition': 'Allergy'},
]

df = pd.DataFrame(data)

def generalize(df, k=2):
    df['ZIP Code'] = df['ZIP Code'].str[:3] + '**'

    def age_range(age):
        if age < 30:
            return '20-29'
        elif age < 40:
            return '30-39'
        elif age < 50:
            return '40-49'
        else:
            return '50+'
    df['Age'] = df['Age'].apply(age_range)

    groups = df.groupby(['ZIP Code', 'Age', 'Gender'])
```

```python
    counts = groups.transform('count')['Condition']

    df.loc[counts < k, ['ZIP Code', 'Age', 'Gender']] = '***'

    return df

k = 2
anon_df = generalize(df.copy(), k)

print("Anonymized Data (k={}):".format(k))
print(anon_df.to_string(index=False))
```

```
Anonymized Data (k=2):
ZIP Code    Age Gender    Condition
   130**  20-29      F     Diabetes
   130**  20-29      F       Cancer
   130**  40-49      M Hypertension
   130**  30-39      M          Flu
   130**  40-49      M       Asthma
   130**  30-39      M      Allergy
```

For K=3

```
import pandas as pd

data = {
    "ZIP Code": [13053, 13053, 13068, 13053, 13068, 13053],
    "Age": [29, 29, 45, 31, 47, 33],
    "Gender": ['F', 'F', 'M', 'M', 'M', 'M'],
    "Condition": ['Diabetes', 'Cancer', 'Hypertension', 'Flu', 'Asthma', 'Allergy']
}

df = pd.DataFrame(data)

def generalize_zip(zip_code):
    return str(zip_code)[:2] + "***"

def generalize_age(age):
    if 20 <= age <= 39:
        return "20-39"
    elif 40 <= age <= 59:
        return "40-59"
    else:
        return "Other"

def generalize_gender(gender):
    return "*"

df['ZIP Code'] = df['ZIP Code'].apply(generalize_zip)
```

```
df['Age'] = df['Age'].apply(generalize_age)
df['Gender'] = df['Gender'].apply(generalize_gender)

print("Generalized Data for k-anonymity (k=3):\n")
print(df)
```

```
Generalized Data for k-anonymity (k=3):

  ZIP Code    Age Gender     Condition
0    13***  20-39      *      Diabetes
1    13***  20-39      *        Cancer
2    13***  40-59      *  Hypertension
3    13***  20-39      *           Flu
4    13***  40-59      *        Asthma
5    13***  20-39      *       Allergy
```

4. Encrypt two numbers and calculate their sum without decrypting first.

```python
import random
import math

def gcd(a, b):
    while b:
        a, b = b, a % b
    return a

def lcm(a, b):
    return a * b // gcd(a, b)

def modinv(a, m):
    g, x, y = extended_gcd(a, m)
    if g != 1:
        raise Exception('No modular inverse exists')
    return x % m

def extended_gcd(a, b):
    if a == 0:
        return (b, 0, 1)
    g, y, x = extended_gcd(b % a, a)
    return (g, x - (b // a) * y, y)

def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(math.sqrt(n)) + 1):
        if n % i == 0:
```

```
            return False
    return True

def random_prime(bits=8):
    while True:
        p = random.randint(2**(bits-1), 2**bits - 1)
        if is_prime(p):
            return p

def generate_keys(bits=8):
    p, q = random_prime(bits), random_prime(bits)
    while p == q:
        q = random_prime(bits)
    n = p * q
    g = n + 1
    lam = lcm(p - 1, q - 1)
    n_sq = n * n
    mu = modinv((pow(g, lam, n_sq) - 1) // n, n)
    return (n, g), (lam, mu)

def encrypt(m, public_key):
    n, g = public_key
    n_sq = n * n
    r = random.randint(1, n - 1)
    while gcd(r, n) != 1:
        r = random.randint(1, n - 1)
    return (pow(g, m, n_sq) * pow(r, n, n_sq)) % n_sq

def decrypt(c, public_key, private_key):
```

```python
    n, g = public_key
    lam, mu = private_key
    n_sq = n * n
    u = pow(c, lam, n_sq)
    l = (u - 1) // n
    return (l * mu) % n

public_key, private_key = generate_keys(bits=8)
n, g = public_key
n_sq = n * n

a = 10
b = 7

enc_a = encrypt(a, public_key)
enc_b = encrypt(b, public_key)

enc_sum = (enc_a * enc_b) % n_sq

decrypted_sum = decrypt(enc_sum, public_key, private_key)

print("Original numbers:     ", a, b)
print("Encrypted values:     ", enc_a, enc_b)
print("Encrypted sum:        ", enc_sum)
print("Decrypted sum result: ", decrypted_sum)
```

```
Original numbers:      10 7
Encrypted values:      2897698417 146046747
Encrypted sum:         3244307927
Decrypted sum result:  17
```