Lab 6 - wstęp do kryptografii

Arkadiusz Kurnik, Jan Cichoń

Zadanie 1:

```python
from math import gcd

print("ZAD a)\n")

# Funkcja obliczająca największy wspólny dzielnik (NWD) dwóch liczb
def extended_gcd(a, b):

    if b == 0:
        return (a, 1, 0)
    else:
        gcd, x1, y1 = extended_gcd(b, a % b)
        x = y1
        y = x1 - (a // b) * y1
        return (gcd, x, y)

# Funkcja obliczająca odwrotność modulo
def mod_inverse(e, phi):

    gcd, x, _ = extended_gcd(e, phi)
    if gcd != 1:
        raise ValueError("e i phi nie są względnie pierwsze, odwrotność nie istnieje.")
    else:
        return x % phi

# Funkcja szyfrowania za pomocą klucza publicznego
def encrypt(message, public_key):
    e, n = public_key
    return pow(message, e, n)  # c = m^e mod n
```

```python
# Funkcja odszyfrowania za pomocą klucza prywatnego
def decrypt(ciphertext, private_key):
    d, n = private_key
    return pow(ciphertext, d, n)  # m = c^d mod n

# Parametry wejściowe
p = 7
q = 11
e = 13
m = 2  # Wiadomość do zaszyfrowania

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

if gcd(e, phi) != 1:
    raise ValueError("e musi być względnie pierwsze z Ø(n)")

if m >= n:
    raise ValueError("Wiadomość m musi być mniejsza od n")

d = mod_inverse(e, phi)


public_key = (e, n)
private_key = (d, n)


ciphertext = encrypt(m, public_key)
```

```python
decrypted_message = decrypt(ciphertext, private_key)


print("1) n =", n, ", Ø(n) =", phi)
print("Warunek dla e:")
print("e musi być względnie pierwsze z Ø(n), czyli gcd(e, Ø(n)) = 1, oraz spełniać 1 < e < Ø(n).")
print("2) d =", d)
print("3) Klucz publiczny:", public_key)
print("   Klucz prywatny:", private_key)
print("4) Zaszyfrowana wiadomość:", ciphertext)
print("5) Odszyfrowana wiadomość:", decrypted_message)
```

```
1) n = 77 , Ø(n) = 60
Warunek dla e:
e musi być względnie pierwsze z Ø(n), czyli gcd(e, Ø(n)) = 1, oraz spełniać 1 < e < Ø(n).
2) d = 37
3) Klucz publiczny: (13, 77)
   Klucz prywatny: (37, 77)
4) Zaszyfrowana wiadomość: 30
5) Odszyfrowana wiadomość: 2
```

Zadanie 2:

```python
print("ZAD b)\n")

import secrets

# Krzywa eliptyczna: y^2 = x^3 + x + 6 (mod p)
p = 11

# Dane wejściowe:
a = 4  # klucz prywatny
alfa = (2, 7)  # generator
m = (3, 6)  # wiadomość

multiplication_table = {
    1: (2, 7),
    2: (5, 2),
    3: (8, 3),
    4: (10, 2),
    5: (3, 6),
    6: (7, 9),
    7: (7, 2),
    8: (3, 5),
    9: (10, 9),
    10: (8, 8),
    11: (5, 9),
    12: (2, 4),
    13: 'O'  # punkt w nieskończoności
}
```

```python
def inverse_mod(k, p):
    if k == 0:
        raise ZeroDivisionError('Nie istnieje odwrotność 0 modulo p')
    return pow(k, -1, p)

def add_points(P, Q):
    if P == 'O':
        return Q
    if Q == 'O':
        return P

    x1, y1 = P
    x2, y2 = Q

    if x1 == x2 and (y1 != y2 or y1 == 0):
        return 'O'

    if P != Q:
        m = ((y2 - y1) * inverse_mod(x2 - x1, p)) % p
    else:
        m = ((3 * x1 * x1 + 1) * inverse_mod(2 * y1, p)) % p

    x3 = (m * m - x1 - x2) % p
    y3 = (m * (x1 - x3) - y1) % p

    return (x3, y3)
```

```python
def mul_point(k, P):
    result = 'O'
    addend = P

    while k:
        if k & 1:
            result = add_points(result, addend)
        addend = add_points(addend, addend)
        k >>= 1

    return result

# --- Rozwiązanie ---

# 1. Klucz publiczny: alfa * a
key_public = mul_point(a, alfa)
print(f"Klucz publiczny (a * G): {key_public}")

# 2. Wybieramy losowe k
k = secrets.randbits(8)
print(f"Losowe k: {k}")

# 3. gamma = k * alfa
gamma = mul_point(k, alfa)
print(f"γ (k * G): {gamma}")

# 4. k*a*alfa
k_a = (k * a) % 13
k_a_alfa = mul_point(k_a, alfa)
print(f"k * a * G: {k_a_alfa}")
```

```python
# 5. sigma = m + k*a*alfa
sigma = add_points(m, k_a_alfa)
print(f"σ (m + k*a*α): {sigma}")

# 6. Kryptogram:
print(f"Kryptogram: (γ, σ) = ({gamma}, {sigma})")

# --- Deszyfrowanie ---

# -a*gamma
minus_a_gamma = mul_point(-a % 13, gamma)  # -a mod 13
print(f"-a * γ: {minus_a_gamma}")

# -a*gamma + sigma
decrypted_m = add_points(minus_a_gamma, sigma)
print(f"Odszyfrowana wiadomość: {decrypted_m}\n")

# --- TABELKA Z ODPOWIEDZIAMI ---

print("--- WYPEŁNIONA TABELKA ---\n")
print("1. wybieramy klucz prywatny a = 4")
print("2. generatorem grupy jest α = (2,7)")
print(f"3. α ·a = {key_public}, zatem klucz publiczny: {key_public}")
print(f"4. chcemy zaszyfrować wiadomość m = {m}")
print(f"5. γ = {k} * {alfa} = {gamma}, σ = {m} + {k_a_alfa} = {sigma}")
print(f"6. Kryptogram: ({gamma}, {sigma})")
print(f"7. Deszyfrowanie: -a ·γ = {minus_a_gamma}, -a ·γ + σ = {decrypted_m}\n")
```

```
Klucz publiczny (a * α): (10, 2)
Losowe k: 253
γ (k * α): (7, 9)
k * a * α: (5, 9)
σ (m + k*a*α): (8, 3)
Kryptogram: (γ, σ) = ((7, 9), (8, 3))
-a * γ: (5, 2)
Odszyfrowana wiadomość: (3, 6)

--- WYPEŁNIONA TABELKA ---

1. wybieramy klucz prywatny a = 4
2. generatorem grupy jest α = (2,7)
3. α ·a = (10, 2), zatem klucz publiczny: (10, 2)
4. chcemy zaszyfrować wiadomość m = (3, 6)
5. γ = 253 * (2, 7) = (7, 9), σ = (3, 6) + (5, 9) = (8, 3)
6. Kryptogram: ((7, 9), (8, 3))
7. Deszyfrowanie: -a ·γ = (5, 2), -a ·γ + σ = (3, 6)
```