# Зразок коду для операцій з точками на еліптичній кривій

class EllipticCurve:

def \_\_init\_\_(self, a, b, p):

self.a = a

self.b = b

self.p = p

def inverse\_mod(self, x):

return pow(x, self.p - 2, self.p)

def point\_add(self, P, Q):

if P == Q:

return self.point\_double(P)

x1, y1 = P

x2, y2 = Q

if x1 == x2 and y1 != y2:

return None

lamb = ((y2 - y1) \* self.inverse\_mod(x2 - x1)) % self.p

x3 = (lamb\*\*2 - x1 - x2) % self.p

y3 = (lamb \* (x1 - x3) - y1) % self.p

return (x3, y3)

def point\_double(self, P):

x, y = P

if y == 0:

return None

lamb = ((3 \* x\*\*2 + self.a) \* self.inverse\_mod(2 \* y)) % self.p

x3 = (lamb\*\*2 - 2 \* x) % self.p

y3 = (lamb \* (x - x3) - y) % self.p

return (x3, y3)

def scalar\_mult(self, k, P):

R = None

Q = P

while k:

if k & 1:

R = self.point\_add(R, Q) if R else Q

Q = self.point\_double(Q)

k >>= 1

return R

def point\_inverse(self, P):

x, y = P

return (x, -y % self.p)

# Приклад використання

curve = EllipticCurve(a=2, b=3, p=97)

P = (3, 6)

Q = (10, 7)

print("Сума P + Q:", curve.point\_add(P, Q))

print("Подвоєння точки P:", curve.point\_double(P))

print("Множення на скаляр 3P:", curve.scalar\_mult(3, P))

print("Інверсія точки P:", curve.point\_inverse(P))

# Зразок коду виконання лабораторної роботи: Еліптичні криві в криптографії

**# Крок 1: Генерація ключової пари ECDSA**

# Використання бібліотеки cryptography:

from cryptography.hazmat.primitives.asymmetric import ec

from cryptography.hazmat.primitives import serialization

# Генерація приватного ключа

private\_key = ec.generate\_private\_key(ec.SECP256K1())

# Збереження приватного ключа у PEM форматі

pem\_private = private\_key.private\_bytes(

encoding=serialization.Encoding.PEM,

format=serialization.PrivateFormat.TraditionalOpenSSL,

encryption\_algorithm=serialization.NoEncryption()

)

print("Приватний ключ:", pem\_private.decode())

# Генерація публічного ключа

public\_key = private\_key.public\_key()

pem\_public = public\_key.public\_bytes(

encoding=serialization.Encoding.PEM,

format=serialization.PublicFormat.SubjectPublicKeyInfo

)

print("Публічний ключ:", pem\_public.decode())

**# Крок 2: Підпис повідомлення за допомогою ECDSA**

from cryptography.hazmat.primitives import hashes

from cryptography.hazmat.primitives.asymmetric import ec

# Підпис повідомлення

message = b"……….."

signature = private\_key.sign(message, ec.ECDSA(hashes.SHA256()))

print("Підпис:", signature)

# Перевірка підпису

try:

public\_key.verify(signature, message, ec.ECDSA(hashes.SHA256()))

print("Підпис вірний!")

except Exception:

print("Підпис некоректний!")

**# Крок 3: Реалізація протоколу Діффі-Геллмана на еліптичних кривих (ECDH)**

# Генерація приватних ключів для двох учасників

private\_key\_A = ec.generate\_private\_key(ec.SECP256K1())

private\_key\_B = ec.generate\_private\_key(ec.SECP256K1())

# Отримання публічних ключів

public\_key\_A = private\_key\_A.public\_key()

public\_key\_B = private\_key\_B.public\_key()

# Обчислення спільного секрету

shared\_key\_A = private\_key\_A.exchange(ec.ECDH(), public\_key\_B)

shared\_key\_B = private\_key\_B.exchange(ec.ECDH(), public\_key\_A)

# Перевірка, що спільний секрет однаковий

assert shared\_key\_A == shared\_key\_B

print("Спільний секрет:", shared\_key\_A.hex())