МІНІСТЕРСТВО ОСВІТИ, НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Методи реалізації криптографічних механізмів

Лабораторна робота:

"Реалізація основних асиметричних криптосистем"

(Підгрупа 2В)

Підготували:

студент 5 курсу

групи ФІ-22 (мн)

Толмачов Є.Ю.

Ковалючук О.М.

Коломієць А. Ю.

Київ – 2023

Лабораторна роботи № 3.

Тема: Реалізація основних асиметричних криптосистем.

Мета роботи: Дослідження можливостей побудови загальних та спеціальних криптографічних протоколів за допомогою асиметричних криптосистем.

Завдання на лабораторну роботу

Підгрупа 2В. Бібліотека PyCrypto під Linux платформу. Стандарт ДСТУ 4145-2002 [3]. Всі нижчеописані функції було переписано з псевдокоду документації[1].

Хід роботи

Імпортуємо необхідні модулі, для реалізації стандарту.

```
import galois

import numpy as np

import math

import Cryptodome as Crypto

from Cryptodome.Hash import SHA512
```

Перевірка точки.

```
def check_point(GF, A, B, m, P):

    if ((P[1]*P[1] + P[0]*P[1])==(P[0]**3 + A*(P[0]**2) + B)):

        return True

    return False
```

Обчислення геш-функції.

```
def SHA3_512(input_byte_string):
    hash_f = SHA512.new(truncate="256")
    hash_f.update(input_byte_string)
    output_byte_string = hash_f.digest()
    return output_byte_string
```

В документації є помилка в формулах, що описують додавання точок [4] еліптичної кривої. Згідно таких зауважень запозичена була ідея, як правильно додавати точки, вже з існуючої реалізації на C++ [2].

Додавання точок еліптичної кривої.

```
def eleptic_curve_addition(P, Q, A, B):
    if (P==Q):
        if(P[0] == GF(0)):
            return ((GF(0),GF(0)))
        t = P[1] * (P[0]**(-1)) + P[0]
        Rx = t*t + t + A
        Ry = P[0]*P[0] + t*Rx + Rx
    elif ((P[0]==Q[0]) and (P[1] == (Q[1] + Q[0]))):
        return ((GF(0),GF(0)))

    else:
        Rx = ((P[1] + Q[1])*((P[0] + Q[0])**(-1)))**2 + (P[1] + Q[1])*((P[0] + Q[0])**(-1)) + P[0] + Q[0] + A
        Ry = ((P[1] + Q[1])*((P[0] + Q[0])**(-1)))*(P[0] + Rx) + Rx + P[1]
    R = (Rx, Ry)
    if (check_point(GF, A, B, m, R)==False):
        return ((GF(0),GF(0)))
     return R
```

Добуток точок еліптичної кривої.

```
def eleptic_curve_multiplication(P, n, A, B):
    k = bin(n)[2:]
    Q = P
    if (Q == (GF(0), GF(0))):
        return Q
    #k = k[::-1]
    k = k[1:]
    for i in k:
        Q = eleptic_curve_addition(Q, Q, A, B)
        #if (Q == (GF(0), GF(0))):
            #return Q
        if(i == '1'):
            Q = eleptic_curve_addition(Q, P, A, B)
            #if (Q == (GF(0), GF(0))):
                #return Q
    return Q
```

Обчислення випадкового елемента основного поля.

```
def generate_random_element(GF):
    return GF.Random()
```

Обчислення напівсліду.

```
def htr(x):
    t = x
    for i in range(int((m-1)/2)):
```

```
        t = (t**4) + x
    return t
```

Обчислення сліду.

```
def tr(x):
    t = x
    for i in range((m-1)):
        t = (t**2) + x
    return t
```

Перевірка кореня.

```
def check_square(GF, m, u, w):
    if (u == 0):
        return 1, w ** (2**(m-1))
    if (w == 0):
        return 2, GF(0)

    v = w*(u ** (-2))
    if (tr(v) == 1):
        return 0, GF(0)
    t = htr(v)
    return 2, t*u
```

Обчислення випадкової точки еліптичної кривої.

```
def random_point(GF, A, B, m):
    while True:
        u = generate_random_element(GF)
```

```
        w = u**3 + A*(u**2) + B
        temp, z = check_square(GF, m, u, w)
        if(temp != 0):
            break
        return((u, z))
```

Перетворення геш-коду на елемент основного поля.

```
def hash_to_GF(GF, m, hash_string):
    bitstring = byte_to_bit(hash_string)[:m]
    return(GF(int(bitstring, 2)))
```

Перетворення пари цілих чисел на цифровий підпис.

```
def pair_to_signature(Ld, r, s):
    R = "",    S = ""
    O = "0"*Ld
    l = int(Ld/2)
    R += bin(r)[2:]
    R += O[:(l - len(R))]
    S += bin(s)[2:]
    S += O[:(l - len(S))]
    D = S + R
    return D
```

Перетворення двійкового рядка на пару цілих чисел.

```
def signature_to_pair(Ld, D):
    r_bit = ""
    s_bit = ""
```

```
    O = "0"*Ld
    l = int(Ld/2)
    r_bit += D[l:]
    s_bit += D[:l]
    r = int(r_bit[:r_bit.rfind('1')+1], 2)
    s = int(s_bit[:s_bit.rfind('1')+1], 2)
    return r, s
```

Перетворення рядка байтів в бітовий формат.

```
def byte_to_bit(data):
    data_bit = ""
    for i in data:
        data_bit += '{0:08b}'.format(i)
    return (data_bit)
```

Весь попередній функціонал реалізовано в вигляді класу, параметри цифрового підпису згідно стандарту, містяться в даній структурі і підрозумівається, що у всіх користувачів використовуються однакові параметри криптосистеми.

```
class curve():


    #ініціалізація параметр кривої
    def __init__(self, param):
        if (param == 163):
            self.curve_163()
        elif (param == 179):
            self.curve_179()
        else:
            print("error: curve was not created")
```

```
        GF = self.GF


        self.sk, self.Q = self.keygen()
        while True:
            Fe, e = self.pre_signature()
            self.D, self.L = self.signature(self.iH, 512, e, Fe, 1337)
            if (self.check_signature(self.D, self.L, self.Q)==True):
                break
            self.sk, self.Q = self.keygen()
    #обчислення напівсліду
    def htr(self, x):
        t = x
        for i in range(int((self.m-1)/2)):
            t = (t**4) + x
        return t


    #обчислення сліду
    def tr(self, x):
        t = x
        for i in range((self.m-1)):
            t = (t**2) + x
        return t


    #генерація ключової пари
    def keygen(self):
        sk = Crypto.Random.random.randint(0, self.n)
        npk = eleptic_curve_multiplication(self.P, sk, self.A, self.B)
        pk = (npk[0], npk[1]+npk[0])
        return sk, pk


    #перевірка правильності приватного ключа
    def check_private_key(self):
```

```
        Q_ = eleptic_curve_multiplication(self.P, self.sk, self.A, self.B)
        pk = (Q_[0], Q_[1]+Q_[0])
        if (pk == self.Q):
            return True
        return False
    #перевірка правильності публічного ключа
    def check_public_key(self, Q):    #check
        if (Q == (self.GF(0), self.GF(0))):
            return False
        if (check_point(self.GF, self.A, self.B, self.m, Q) == False):
            return False
        check = eleptic_curve_multiplication(Q, self.n, self.A, self.B)
        if (check == (self.GF(0), self.GF(0))):
            return True
        return False

    #обчислення цифрового передпідпису
    def pre_signature(self):
        while True:
            e = Crypto.Random.random.randint(0, self.n)
            R = eleptic_curve_multiplication(self.P, e, self.A, self.B)
            if (R[0] != self.GF(0)):
                break
        return R[0], e

    #обчислення базової точки кривої
    def base_point(self):
        while True:
            P = random_point(self.GF, self.A, self.B, self.m)
            R = eleptic_curve_multiplication(self.P, self.n, self.A, self.B)
            if (R == (self.GF(0), self.GF(0))):
                break
```

```
        return P


    #підписування
    def signature(self, iH, Ld, e, Fe, T):
        if (Ld%16 != 0):
            return "Error"
        if (Ld < 2*len(bin(self.n)[2:])):
            return "Error"
        H_t = self.hash_f(iH, bytes(T))
        h = hash_to_GF(self.GF, self.m, H_t)
        y = h*Fe
        r = int(y) % self.n
        s = (e+ self.sk*r)%self.n
        D = pair_to_signature(Ld, r, s)
        return ((iH, T, D), (len(bin(iH)[2:])+len(bin(T)[2:])+Ld))
    #перевірка цифрового підпису
    def check_signature(self, signature, L, sk, Q):
        if (signature[0] != self.iH):
            return "Error"
        if (self.check_public_key(Q) == False):
            return "Error"
        if ((L - len(signature[2]) - len(bin(self.iH)[2:])) <= 0):
            return "Error"
        r, s = signature_to_pair(len(signature[2]), signature[2])
        R = eleptic_curve_addition(eleptic_curve_multiplication(self.P, s, self.A, self.B), eleptic_curve_multiplication(Q, r, self.A, self.B), self.A, self.B)
        H_t = self.hash_f(signature[0], bytes(signature[1]))
        h = hash_to_GF(self.GF, self.m, H_t)
        r_ = int(h*R[0])%(self.n)
        if(r == r_):
            return True
        else:
            return "Error"
```

```
#гешування
def hash_f(self, iH, input_byte_string):
    if (iH == 1):
        return SHA3_512(input_byte_string)
    else:
        print("Error: wrong hash ID")
        return "Error"

# параметри кривої   curve_163
def curve_163(self):
    self.GF = galois.GF(2**163, repr = "poly")
    self.A = self.GF(1)
    self.B = self.GF(0x5FF6108462A2DC8210AB403925E638A19C1455D21)
    self.n = 0x400000000000000000002BEC12BE2262D39BCF14D
    self.m = 163
    self.P = (self.GF(0x2E2F85F5DD74CE983A5C4237229DAF8A3F35823BE), self.GF(0x3826F008A8C51D7B95284D9D03FF0E00CE2CD723A))
    self.iH = 1

# параметри кривої curve_179
def curve_179(self):
    self.GF = galois.GF(2**179, repr = "poly")
    self.A = self.GF(1)
    self.B = self.GF(0x4A6E0856526436F2F88DD07A341E32D04184572BEB710)
    self.n = 0x3FFFFFFFFFFFFFFFFFFFFFFB981960435FE5AB64236EF
    self.m = 179
    self.P = base_point(self.GF, self.A, self.B, self.n, self.m)
    self.iH = 1

#створення підпису
  def create_signature(self, T):
    while True:
```

```
        Fe, e = self.pre_signature()
        D, L = self.signature(self.iH, 512, e, Fe, 1337)
        if (self.check_signature(self.D, self.L, self.Q)==True):
            break
    return D, L
```

Контрольний приклад роботи з асиметричною криптосистемою

```
param = 163
m = param
GF = galois.GF(2**param, repr = "poly")
c = curve(param)
print(c.check_public_key(pk))
print(c.check_private_key(pk, sk))
```

```
Result:True
Result:True
```

```
c.check_signature(c.D, c.L, c.Q)
Result: True
```

Висновок

Нами було реалізовано функціонал стандарту ДСТУ 4145-2002 України за допомогою бібліотеки PyCrypto під Linux платформу, перевірено коректність роботи окремих частин стандарту, та проведено тестування асиметричної криптосистеми.

Список використаних літературних джерел

[1] *iTender - системы электронных торгов, электронные торговые площадки, автоматизация закупок - Ай Тендер*.
URL: https://itender-online.ru/wp-content/uploads/2017/09/dstu-4145-2002-1.pdf (дата звернення: 11.01.2023).

[2] GitHub - shamray/dstu4145: Modern C++ implementation of DSTU 4145-2002 - Ukrainian standard for eleptic *curve* cryptography digital signature algorithm. *GitHub*. URL: https://github.com/shamray/dstu4145 (дата звернення: 11.01.2023).

[3] Учасники проектів Вікімедіа. ДСТУ 4145-2002 – Вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/ДСТУ_4145-2002 (дата звернення: 11.01.2023).

[4] ЭЦП стран СНГ на Python. *PVSM.RU - новости информационных технологий*. URL: https://www.pvsm.ru/python/121075 (дата звернення: 11.01.2023).