МІНІСТЕРСТВО ОСВІТИ, НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Методи реалізації криптографічних механізмів

Лабораторна робота:

"Реалізація алгоритмів генерації ключів гібридних криптосистем"

(Підгрупа 2В)

Підготували:
студент 5 курсу
групи ФІ-22 (мн)
Толмачов Є.Ю.
Ковальчук О.М.
Коломієць А. Ю.

Київ – 2023

Лабораторна робота №2

Тема: Реалізація алгоритмів генерації ключів гібридних криптосистем.

Мета роботи: Дослідження алгоритмів генерації псевдовипадкових послідовностей, тестування простоти чисел та генерації простих чисел з точки зору їх ефективності за часом та можливості використання для генерації ключів асиметричних криптосистем.

Завдання на лабораторну роботу

Підгрупа 2В. Бібліотека PyCryptodome. Для тестування було обрано 3 критерії із курсу Асиметричної криптографії [1] та два статистичних критерії із стандарту NIST [2].

Хід роботи

Імпортуємо необхідні для роботи бібліотеки.

```
import numpy as np
import math
import Cryptodome as Crypto
from Cryptodome.PublicKey import RSA
from Cryptodome.PublicKey import DSA
from Cryptodome.PublicKey import ECC
```

Обчислюємо квантилі χ^2, які є необхідними для перших трьох критеріїв.

```
r = 12
alpha = [ 0.01, 0.05, 0.1 ]
quantil = [ 2.327, 1.645 , 1.285 ]
Xi_1 = [0]*3
Xi_2 = [0]*3
Xi_3 = [0]*3
print(Xi_1)
Xi_1 = [i*np.sqrt(255 * 2) + 255 for i in quantil]

Xi_2 = [i*np.sqrt(255 * 255 * 2) + 255*255 for i in quantil]

Xi_3 = [i*np.sqrt(255 * 255 * 2) + 255*(r-1) for i in quantil]
```

Генеруємо випадкові послідовності:

Звичайний генератор в PyCryptodome – генерує випадкові байти на основі urandom.

```
get_random_bytes = bytearray()
for i in range(int(Bytes/N)):
    get_random_bytes += Crypto.Random.get_random_bytes(N)
```

Рандомізатор по бітам – генерує n-бітне ціле число на основі urandom.

```
getrandbits = bytearray()
for i in range(int(Bytes)):
    getrandbits += Crypto.Random.random.getrandbits(8).to_bytes(1,'big')
```

Генерація ключів для RSA. RSA генератор, як і DSA, має ряд недоліків пов'язаних із процесом генерації через необхідність генерації випадкових величезних простих чисел. А також згенерована послідовність є ключем – а отже може

```
n = 2048
RSApk = bytearray()
for i in range(int(Bytes/(n/8))):
    a = Crypto.PublicKey.RSA.generate(n)
    RSApk += a.p.to_bytes(128, 'big') + a.q.to_bytes(128, 'big')
```

Генерація ключів для еліптичних кривих.

```
ECCpk = bytearray()
while (len(ECCpk)<Bytes):
    ECCpk += ECC.generate(curve='P-256').d.to_bytes()
```

Генерація ключів для цифрового підпису DSA.

```
DSApk = bytearray()
while (len(DSApk)<Bytes):
    DSApk += DSA.generate(2048).p.to_bytes(256, 'big')
```

Критерій перевірки рівноімовірності знаків.

```
def criterion_1(data):
    Xi = 0.0
    H = [True, True, True]
    n = int(len(data)/256)
    byte_count = [0]*256
    for i in range(256):
        byte_count[i] = data.count(i)
        Xi += ((byte_count[i] - n)**2 / n)
    for i in range(3):
        if Xi > Xi_1[i]:
            H[i] = False

    return Xi, H
```

Критерій перевірки незалежності знаків.

```
def criterion_2(data):
    Xi = 0.0
    H = [True, True, True]
    n = int(len(data)/2)
    byte_count = [[0]*256 for i in range(256)]
    first = [0]*256
    second = [0]*256
    for i in range(n):
        byte_count[data[2*i]][data[2*i+1]] += 1
        first[data[2*i]] += 1
        second[data[2*i+1]] += 1

    for i in range(256):
        for j in range(256):
            Xi += ((byte_count[i][j]**2)/(first[i]*second[j]))
    Xi = (Xi-1)*n
```

```
    for i in range(3):
        if Xi > Xi_2[i]:
            H[i] = False

    return Xi, H
```

Критерій перевірки однорідності двійкової послідовності.

```
def criterion_3(data):
    Xi = 0.0
    H = [True, True, True]
    n = int(len(data))
    m_ = int((n/r))
    n = m_*r
    byte_count_r = [[0]*12 for i in range(256)]
    byte_count = [0]*256
    for j in range(r):
        for i in range(m_):
            byte_count_r[data[i + j*m_]][j] +=1
            byte_count[data[i + j*m_]] += 1
    for i in range(256):
        for j in range(12):
            Xi += ((byte_count_r[i][j]**2)/(byte_count[i]*m_))
    Xi = (Xi-1)*n
    for i in range(3):
        if Xi > Xi_3[i]:
            H[i] = False

    return Xi, H
```

Універсальний статистичний тест Мюрера (Maurer's "Universal Statistical" Test)

```
def criterion_4(data):
    Xi = 0.0
    sum_ = 0
    H = [True, True, True]
    data_bit = ""
    for i in data:
        data_bit += '{0:08b}'.format(i)
    n = int(len(data_bit))
    Q = 10*(2**L)
    T = [0]*(2**L)
    for i in range(Q):
        T[int(data_bit[i*L: i*L+L],2)] = i
    K = int(n/L - Q)
    for i in range(K):
        sum_ += math.log2(i+Q - T[int(data_bit[Q*L + i*L: Q*L + i*L+L],2)])
        T[int(data_bit[Q*L + i*L: Q*L + i*L+L],2)] = i+Q
    f_n = sum_/K
    c = 0.7 - (0.8/L) + (4 + 32/L)*(pow(K, (-3/L))/15)
    gamma = c*math.sqrt(var/K)
    P = math.erfc(abs(f_n - expV)/(math.sqrt(2)*gamma))
    #1% rule states to check P-value < 0.01 => False
    for i in range(3):
        if P < alpha[i]:
            H[i] = False
```

```
    return P, H


L = 8
expV = 7.1836656
var = 3.238
```

Random Excursions Variant Test

```
def criterion_5(data):
    Xi = 0.0
    sum_ = 0
    H = [True, True, True]
    data_bit = ""
    for i in data:
        data_bit += '{0:08b}'.format(i)
    n = int(len(data_bit))
    xi = [0]*19
    S = 0
    xi[9] += 1
    for i in range(n):
        if(data_bit[i] == '1'):
            S += 1
        else:
            S -= 1
        if (abs(S)<=9):
            xi[S+9] += 1
    J = xi[9]
    P = [0]*19

    for i in range(19):
        if(i != 9):
            P[i] = math.erfc(abs(xi[i]-J)/math.sqrt(2*J*(4*abs(i-9)-2)))
    #1% rule states to check P-value < 0.01 => False
    P.pop(9)
    for i in range(3):
        for j in range(18):
            if P[j] < alpha[i]:
                H[i] = False
                break
    return P, H
```

Результати

Звичайний генератор в PyCrypto.

```
(231.9208984375, [True, True, True])
(64733.273770514585, [True, True, True])
(2713.2082519033584, [True, True, True])
(0.11150358912339814, [True, True, True])
([0.9397309575096909, 0.7292594547341302, 0.6972207829947468, 0.569588085563856, 0.5160423810100728, 0.3864762307712326, 0.3256
7103678286924, 0.2606338425315887, 0.3577648175618734, 0.9875638831765086, 0.7391579082705373, 0.944426752288836, 0.42642188541
29438, 0.5261650033087717, 0.9700088087582979, 0.7457646413634914, 0.702215354184505, 0.5106732390107467], [True, True, True])
```

Рандомізатор по бітам.

```
(244.1982421875, [True, True, True])
(64698.98133374803, [True, True, True])
(2899.372616706713, [True, True, True])
(0.36201293506084126, [True, True, True])
([0.5918420734686674, 0.7198380560481024, 0.8326671994984294, 0.7355318762955201, 0.8149745964518661, 0.7433286647292777, 0.593
0625981066426, 0.36064704767104244, 0.16478920460634883, 0.6326610502373972, 0.9518631313415185, 0.7233073123857566, 0.59563192
01472444, 0.6940671765173658, 0.8712014685757845, 0.5959202492164613, 0.4026437860695563, 0.3670210957124318], [True, True, Tru
e])
```

Генерація ключів для RSA

```
(408.1591796875, [False, False, False])
(65215.779879146896, [True, True, True])
(2787.129886081236, [True, True, True])
(0.11066727041051792, [True, True, True])
([0.10927093392001368, 0.10598585916278155, 0.05577945284478564, 0.03495507264363056, 0.027734470235048183, 0.0252923243963986
4, 0.03770719138706869, 0.1498121351670915 5, 0.6597890810315775, 0.6955808273182371, 0.21404421720997704, 0.18937309460325985,
0.18932919450771196, 0.17083529853378324, 0.11799977503975212, 0.05753961187264973, 0.059875515962479545, 0.06595420164423037],
[True, False, False])
```

Генерація ключів для еліптичних кривих

```
(238.53857421875, [True, True, True])
(64878.67444660347, [True, True, True])
(2851.986795278689, [True, True, True])
(0.06229353442169084, [True, True, False])
([0.9043132010267174, 0.5664130791782349, 0.4953312122622224, 0.4880392736138859, 0.5431833514113554, 0.8221278693532301, 0.852
2892477910072, 0.8907458009320663, 0.634194817329482, 0.08103475417128944, 0.20798030941952225, 0.5764381972359925, 0.952196201
9705574, 0.7560957014958838, 0.7559159758919114, 0.9473890666567608, 0.6523621476238477, 0.427535517515224], [True, True, Fals
e])
```

Генерація ключів для цифрового підпису DSA

```
(264.0636942675158, [True, True, True])
(64817.988896884526, [True, True, True])
(2817.9734278418814, [True, True, True])
(0.5684911190508302, [True, True, True])
([0.2938128087536618, 0.5545805612380654, 0.68903734077464, 0.6635466389640836, 0.7028835190625773, 0.7068509494219011, 0.65632
39748867087, 0.4551658905375948, 0.6906192201976447, 0.028590893764117786, 0.04748223864823018, 0.15445188236025, 0.21457149348
404794, 0.2594429134031647, 0.24806677651344947, 0.12560788237384796, 0.06072428097795445, 0.042643136577263874], [True, False,
False])
```

Висновок

Використовуючи бібліотеку PyCryptodome, було побудовано 5 критеріїв на випадковість для деяких генераторів випадкових послідовностей. Всі вони є стійкими відносно цих 5 критеріїв, хоча генератор ключів має ряд недоліків. То не є дивним оскільки в основі всіх них є криптографічно стійкий генератор urandom.

Список використаної літератури

[1] «Асиметричні криптосистеми та протоколи» Комп'ютерний практикум №1 «Побудова тестів для перевірки якості випадкових та псевдовипадкових послідовностей»

https://www.dropbox.com/s/wflzr4df6c0tidv/AsCrypto_CP1.2016.pdf?dl=0

[2] *Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, Elaine Barker ,Stefan Leigh, Mark Levenson, Mark Vangel, David Banks, Alan Heckert, James Dray, San Vo* A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications – NIST Special Publication 800-22 Revision 1a – 2010