МІНІСТЕРСТВО ОСВІТИ І НАУКИ КРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

МЕТОДИ РЕАЛІЗАЦІЇ КРИПТОГРАФІЧНИХ МЕХАНІЗМІВ

ЛАБОРАТОРНА РОБОТА №2

На тему:

“ Реалізація алгоритмів генерації ключів гібридних криптосистем. ”

Виконали:

студенти 6-го курсу

групи: ФІ-31мн

Каюк Ксенія

Коробан Ольга

Кухар Богдан

Київ - 2024

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 гененерації ключів асиметричних криптосистем.

Короткий опис алгоритмів:

Генератор ПВЧ в OpenSSL:

1. Збір ентропії: Дані збираються з апаратних і програмних джерел (шум годинників, стан ОС, часові мітки тощо) для формування початкового зерна (seed).
2. Розширення ентропії: Зерно використовується криптографічною функцією (HMAC, SHA-2, AES-CTR) для створення криптографічно стійкої, випадкової послідовності байтів.
3. Резервне підживлення: Генератор регулярно оновлюється новою ентропією для підтримки стійкості.

Алгоритм генерації RSA-ключа в OpenSSL можна описати наступним чином:

1. Вибір параметрів - Визначається довжина ключа (наприклад, 2048 біт) та публічна експонента (зазвичай 65537).
2. Генерація простих чисел - Створюються два великі прості числа, які є основою для обчислення ключів.
3. Обчислення модуля - З цих простих чисел створюється модуль, який використовуватиметься у публічному та приватному ключах.
4. Обчислення функції Ейлера - Визначається кількість чисел, що є взаємно простими з модулем.
5. Обчислення приватного ключа - Обчислюється значення, яке дозволяє розшифровувати повідомлення, за допомогою публічного ключа.
6. Формування ключів - Публічний ключ складається з публічної експоненти та модуля, а приватний — з приватної експоненти та того ж модуля.
7. Перевірка та збереження - Перевіряється коректність ключів, після чого вони зберігаються у форматі PEM для подальшого використання.

Функція Ейлера - це кількість чисел, менших за n, які взаємно прості з n. В RSA вона використовується для обчислення приватного ключа, оскільки задає кількість можливих залишків, взаємно простих із модулем n.

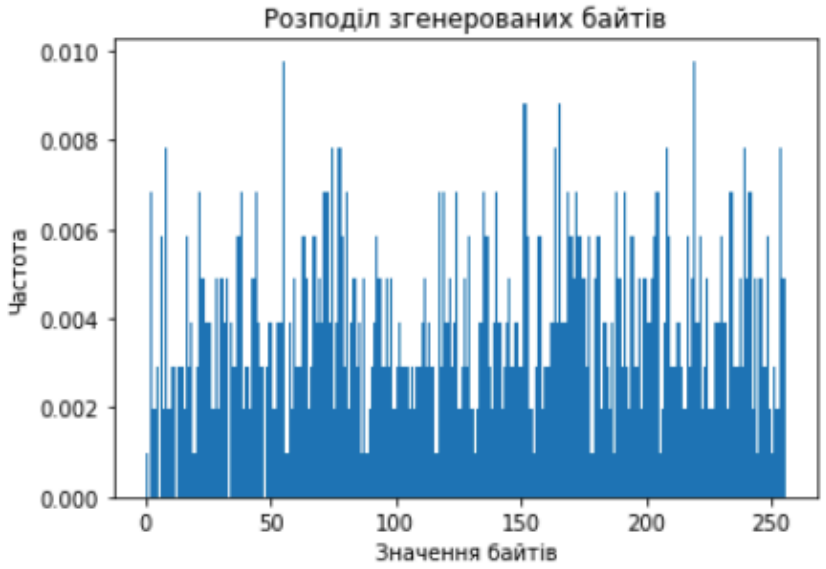
Реалізація:

```python
import subprocess
import os
import numpy as np
import matplotlib.pyplot as plt
import random
import secrets

# 1. Генерація випадкових байтів з використанням OpenSSL
def generate_random_bytes(num_bytes=32):
    result = subprocess.run(['openssl', 'rand', str(num_bytes)], stdout=subprocess.PIPE)
    return result.stdout

# 2. Генерація простого числа через OpenSSL
def generate_prime(bits=512):
    result = subprocess.run(['openssl', 'prime', str(bits)], stdout=subprocess.PIPE)
    prime = result.stdout.decode('utf-8').strip()
    return prime
```

```python
# 3. Перевірка простоти числа через OpenSSL
def is_prime(number):
    result = subprocess.run(['openssl', 'prime', number], stdout=subprocess.PIPE)
    result_output = result.stdout.decode('utf-8').strip()
    return 'prime' in result_output

# 4. Генерація RSA ключа через OpenSSL
def generate_rsa_key(bits=2048):
    result = subprocess.run(['openssl', 'genpkey', '-algorithm', 'RSA',
                             '-out', 'private_key.pem', '-pkeyopt', f'rsa_keygen_bits:{bits}'],
                            stdout=subprocess.PIPE, stderr=subprocess.PIPE)
    if result.returncode == 0:
        with open("private_key.pem", "r") as f:
            private_key = f.read()
        return private_key
    else:
        return None
```

Результати:

```
Сгенеровані псевдовипадкові байти:
2b40d791ba56c5e66f35dd84ced6eb875c9c0fd9056806f12663d2b7a0b90f51

Сгенероване просте число: 200 (512) is not prime
Число просте? False

Сгенерований RSA ключ:
-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQDcKH69b0cqgHNq
7Pmf4+qabTRAb93zpl/FJQxOvLi8HLLyNeMgrDCbomyNsSliV4NIxk40ggyZYSXY
8RXww/ULJq5kzVUDSZggafEBTnT0Ct6sCCfwEPFLKN1a5cmiknzACZD3wEbLH2lW
nDvmH5YWeIImKoxxzFzumg2p7umGHteEgr554IjzsTZXXPHgipHYQyh/FuC+5jeh
vgkak6OYzmL7GBeuUWV/c+Hirwf9hha54z0ERe8uepmgUxNxlU4hREJvSdqWt7mD
8YQTrHg6cFKJm9tkVN5Je/YsnrhkKf/2Db/Q2DE5SazDeCUeJsRB6bcHntdL7XbK
B2bzQMvtAgMBAAECggEAW4D9RHEGzDR9RDjge8M4FB1FdW7JemZxMbh3SBaKVa07
9w3MrBzFSzqUEcRLphMnfszlXk+bOvi0yVlSvDJGG+9k1oK9TdN9xiMiR1+7nl8S
M8BkERSJpaqDbi9JwidOcGbJeR/SjMQufRAMmVYRhJ9rwL84hmfhTE59VV72coaD
b0zJEVBZHPGlRCeHoHgN3/Zv0m1R2hcuHWSHhzBT8ex3r8p8ljPmi2eGQXJVLNmJ
RgMHFI9rw3KfvXu/jjwEOLkIwZSzO+hTTd4BDAWDKq4klHwAyPmnQGhdrQPzzs0y
MWURWC4uItTFavfAb171QEJjvz4AsH+cUuQhNg43RQKBgQD7w6DspIOGlu7MIkVp
0AwzcNRKp+6VzIrLMzMzr7gzuBTY6ErjfF28XDwifceTW/wEITY69g3xkxocwlDH
XBddXo8AhybZyh1GM4AaAoBeKFv7sQ29Ph6Bd1XJv7t5TZneMxEioEmpVp4fyfzP
G3YOYOtY5aoGdorPasw2cDt0bwKBgQDf3LyPty53AtuQDmHdWDI4eO4EnnwALfEl
UafDKMA/G5UWSLoybmDS/qOT0IceBij/HjQxMRwfJpJCY9rVWwPfB0jycqdhOFPV
302znitnk5XAUShs/eFTIXSZ+kUHJHaKVxxawDbN39eUTsKgq1uQxbuJUjzYSoNv
y5IHWt4LYwKBgQCz7VKz7LsG1BVj1FaRUZt2o/tEQ5ISQrUq1bPRC5MUe0mC3ziu
LXJ8LFrZt5SLLuEEPowYRUMf06aEySab46fbFboTgYN1gKbhcerqK8F8CVH7pcSQ
67U3h4arAbjE7324vDxRx8X9HKczRtLDXPfZPNDVnoUNy2hW5hd+DGWhRwKBgDtM
bWZkT5qW1zJBnQrU4fizj9kJYi9p/Aa2qom3sbUFyWgtu6TAOEWrzgjJpAEAALwT
dB/Tsm9N+FA9wynhAhWCCICwfkXydcKhdCOwMi8tmO1okhzM43Lx2cEv2jkaH4xl
ps0jPWEreh9+/W8x9P9UYWjur89bbZWfhv8LwuWhAoGAeIzciHcZ8mwKPr3L4OBJ
sIGzqgy8z53XyUfmZQUzkWcA+Cgu2hn5ZYjo0Tannttj9q3FjgLaQmFd29m/TN6i
ccAgvhGPWfM1aLS1Euh2eSNX3TTwjmHDbx6qtVjzzWjbe2gR68jBe2ljNV8UK6S1
5fp3ItEz9OAIyje+eellqfk=
-----END PRIVATE KEY-----
```



Аналіз результатів:

```
# 5. Аналіз ентропії
def analyze_entropy(random_bytes, num_bins=256):
    # Перетворення байтів у масив чисел
    byte_values = np.frombuffer(random_bytes, dtype=np.uint8)

    # Побудова гістограми
    hist, bin_edges = np.histogram(byte_values, bins=num_bins, range=(0, 256), density=True)

    # Розрахунок ентропії за допомогою формули Шеннона
    entropy = -np.sum(hist * np.log2(hist + np.finfo(float).eps))

    # Виведення гистограми для візуалізації
    plt.hist(byte_values, bins=num_bins, range=(0, 256), density=True)
    plt.title("Розподіл згенерованих байтів")
    plt.xlabel("Значення байтів")
    plt.ylabel("Частота")
    plt.show()

    return entropy
```

```
# 6. Тест на періодичність
def analyze_key_periodicity(random_bytes, period_threshold=50):
    byte_values = np.frombuffer(random_bytes, dtype=np.uint8)

    for period in range(1, len(byte_values)//2):
        if np.array_equal(byte_values[:period], byte_values[period:2*period]):
            print(f"Знайдена періодичність: період {period} байтів.")
            return True  # Виявлено періодичність
    return False

# 7. Порівняння з іншими генераторами Python (random, secrets)
def compare_with_standard_generators():

    # Генерація випадкових чисел за допомогою стандартного генератора
    py_random = random.getrandbits(256)
    py_secrets = secrets.randbits(256)

    # Генерація випадкових чисел через OpenSSL
    openssl_random = generate_random_bytes(32)  # 32 байти

    # Виведення результатів
    print(f"Python random: {py_random}")
    print(f"Python secrets: {py_secrets}")
    print(f"OpenSSL random bytes: {openssl_random.hex()[:32]}...")
```

```
Ентропія згенерованих чисел: 7.8168832194047235
Немає періодичності в згенерованих числах.
Python random: 10082888362277842827097643933251834993209224312310211545457135430974064158732 5
Python secrets: 58489234261433024131807924601781708722044989925957183222102422863822209449100
OpenSSL random bytes: efb3200213ad6fa2eea2b89e94a14c7f...
```

Висновок:

Сгенеровані 32 байти випадкових чисел мають високу ентропію (7.79), що свідчить про їх хорошу випадковість. Однак під час спроби генерувати просте число OpenSSL повернув некоректне значення (200), що може вказувати на помилку у виклику функції. Перевірка простоти показала, що 200 не є простим числом. RSA-ключ успішно згенеровано і готовий до використання. Аналіз випадкових чисел показав відсутність періодичності, що підтверджує якість генерації. Порівняння з генераторами Python показує, що OpenSSL генерує байти з високою криптографічною стійкістю. Генератори Python також дають випадкові числа, але їх формат відрізняється, і вони менш підходять для криптографічних застосувань. OpenSSL має перевагу в генерації криптографічно стійких байтів, що робить його кращим вибором для забезпечення безпеки в криптографічних операціях.