Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Фізико-технічний інститут

Методи реалізації криптографічних механізмів

Лабораторна робота №2

Виконала:
Студентка ФІ-31мн
Двоєглазова (Панкєєва) С.Д.

Перевірила:
Селюх П. В.

Київ – 2024

Лабораторна робота №2. Реалізація алгоритмів генерації ключів гібридних криптосистем

Мета:

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:

Для другого типу лабораторних робіт — аналіз стійкості реалізації ПВЧ та генераторів ключів для обраної бібліотеки.

Бібліотека OpenSSL під Windows платформу.

Оформлення результатів роботи. Опис функції генерації ПСП та ключів бібліотеки з описом алгоритму, вхідних та вихідних даних, кодів повернення. Контрольний приклад роботи з функціями.

Хід роботи:

Генерація випадкових байтів є важливою в криптографії для створення ключів і випадкових даних. В Python для цього часто використовують os.urandom, який генерує криптографічно безпечні байти. Оцінка якості таких байтів включає вимір часу їх генерації та аналіз ентропії, що визначає їх випадковість. Чим вища ентропія, тим більш випадкові дані, що критично для криптографії, оскільки передбачувані байти можуть створити вразливості.

RSA — це алгоритм асиметричного шифрування, який використовує пару ключів: публічний для шифрування і приватний для дешифрування. Важливо виміряти час генерації RSA ключів, а також оцінити довжину публічного ключа та його ентропію. Для цього в Python використовують бібліотеку cryptography, яка підтримує створення RSA ключів різних розмірів.

Ентропія — це міра непередбачуваності або випадковості даних, яка показує, скільки інформації міститься в певному наборі даних. У криптографії ентропія важлива для оцінки якості випадкових чисел або байтів. Чим вища ентропія, тим більш випадкові і менш передбачувані дані, що робить їх придатними для криптографічних цілей.Ентропія вимірюється за допомогою формули Шеннона.

Функції :

1. generate_random_data(byte_count)

- Функція генерує випадкові байти заданої довжини та вимірює час, необхідний для їх створення.
  Використовується для аналізу генерації псевдовипадкових послідовностей у бібліотеці.

  Вхідні дані:

  - byte_count (int): Кількість байтів, які потрібно згенерувати.

  Вихідні дані:

  - random_bytes (bytes): Згенеровані випадкові дані.
  - duration (float): Час, витрачений на генерацію (у секундах).

2. compute_shannon_entropy(byte_data)

- Функція розраховує ентропію Шеннона (рівень невпорядкованості) для заданих даних у байтах.
  Використовується для оцінки якості генерації випадкових даних.

  Вхідні дані:

  - byte_data (bytes): Масив байтів, для якого обчислюється ентропія.

  Вихідні дані:

  - entropy (float): Значення ентропії у бітах на байт.

3. rsa_key_generation(bits=2048)

- Створює пару RSA-ключів (приватний і публічний) заданого розміру та вимірює час генерації.
  Аналізує продуктивність алгоритму RSA у бібліотеці OpenSSL.

  Вхідні дані:

  - bits (int): Розмір ключа у бітах (типово 2048).

  Вихідні дані:

  - private_key (RSA): Згенерований приватний ключ.
  - duration (float): Час генерації ключа (у секундах).

4. evaluate_generation(byte_size, rsa_size)

- Порівнює два процеси: генерацію випадкових даних та генерацію RSA-ключів. Аналізує час виконання, якість випадкових даних (ентропія) та довжину публічного ключа.

  Вхідні дані:

  - byte_size (int): Кількість байтів для генерації випадкових даних.
  - rsa_size (int): Розмір ключа RSA (у бітах).

  Вихідні дані:

  - Немає явного повернення даних. Усі результати виводяться в консоль.

Висновки:

1. Бібліотека OpenSSL забезпечує генерацію випадкових чисел із високим рівнем ентропії (понад 7.99 біт/байт), що наближається до максимальної теоретичної межі (8 біт/байт).
2. Генерація ПВЧ відбувається за лічені мілісекунди, що робить її надзвичайно ефективною навіть для великих обсягів даних (1024 байти та більше).
3. Процес створення RSA-ключів займає значно більше часу через обчислювальну складність алгоритму, особливо для ключів розміру 2048 біт.
4. Отриманий публічний ключ має структуровану ентропію (близько 5.5–6 біт/байт), що є нормальною характеристикою RSA-ключів, оскільки вони містять регулярні компоненти.
5. Генерація випадкових даних є значно швидшою, ніж RSA, через відсутність необхідності виконання обчислювально складних операцій, таких як піднесення до степеня чи факторизація.
6. Випадкові дані забезпечують максимальний рівень ентропії, що робить їх придатними для створення ключів, паролів та інших криптографічних компонентів.
7. OpenSSL показує високу продуктивність у криптографічних операціях, що робить її ідеальним вибором для безпечних додатків під Windows.

Код програми :

[11]:
```python
import math
import os
import time

from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.asymmetric import rsa
```

[12]:
```python
def generate_random_data(byte_count):
    """Генеруе випадкові дані заданого розміру та вимірює тривалість генерації.

    Аргументи:
        byte_count (int): Кількість байтів для генерації.

    Повертає:
        tuple: Згенеровані дані у вигляді байтів та час, витрачений на їх створення.
    """
    start = time.time()  # Початок вимірювання часу
    random_bytes = os.urandom(byte_count)  # Створення випадкових байтів
    duration = time.time() - start  # Обчислення часу виконання

    # Виведення перших 64 символів у вигляді hex
    print(f'Згенеровані дані: {random_bytes.hex()[:64]}... (64 символи в hex)')

    return random_bytes, duration
```

[13]:
```python
def compute_shannon_entropy(byte_data):
    """Розраховує ентропію Шеннона для заданих байтових даних.

    Аргументи:
        byte_data (bytes): Масив байтів для аналізу.

    Повертає:
        float: Ентропія даних у бітах на байт.
    """
    # Ймовірності появи кожного байта
    probabilities = [byte_data.count(byte) / len(byte_data) for byte in set(byte_data)]
    # Формула Шеннона
    return -sum(p * math.log2(p) for p in probabilities)
```

[14]:
```python
def rsa_key_generation(bits=2048):
    """Створює пару RSA-ключів заданого розміру та вимірює час генерації.

    Аргументи:
        bits (int): Розмір ключа в бітах (типово 2048).

    Повертає:
        tuple: Приватний ключ RSA та час, витрачений на генерацію.
    """
    start = time.time()  # Початок вимірювання часу
    private_key = rsa.generate_private_key(public_exponent=65537, key_size=bits, backend=default_backend())
    duration = time.time() - start  # Час генерації

    print(f'RSA-ключ ({bits}-бітний) успішно створений')

    return private_key, duration
```

[15]:
```python
def evaluate_generation(byte_size, rsa_size):
    """Порівнює генерацію випадкових даних і ключів RSA за часом та ентропією.

    Аргументи:
        byte_size (int): Кількість байтів для генерації випадкових даних.
        rsa_size (int): Розмір RSA-ключа в бітах.
    """
    # Тест генерації випадкових байтів
    print(f"\n{'=' * 40}\nТест: Генерація випадкових даних ({byte_size} байтів):")
    random_data, gen_time = generate_random_data(byte_size)
    entropy_random = compute_shannon_entropy(random_data)
    print(f'• Час генерації: {gen_time:.6f} секунд')
    print(f'• Ентропія даних: {entropy_random:.6f} біт на байт')
    print('-' * 40)

    # Тест генерації RSA ключів
    print(f"\n{'=' * 40}\nТест: Генерація RSA-ключів ({rsa_size} біт):")
    rsa_key, rsa_time = rsa_key_generation(rsa_size)
    public_key_bytes = rsa_key.public_key().public_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PublicFormat.SubjectPublicKeyInfo
    )  # Публічний ключ у форматі PEM
    entropy_public_key = compute_shannon_entropy(public_key_bytes)

    print(f'• Час генерації: {rsa_time:.6f} секунд')
    print(f'• Довжина публічного ключа: {len(public_key_bytes)} байтів')
    print(f'• Ентропія публічного ключа: {entropy_public_key:.6f} біт на байт')
    print('-' * 40)
```

```python
    # Підсумки
    print(f"\n{'=' * 40}\nРезультати порівняння:")
    print('Випадкові дані:')
    print(f'  - Час генерації: {gen_time:.6f} секунд')
    print(f'  - Ентропія: {entropy_random:.6f} біт на байт')
    print(f'\nRSA-ключі ({rsa_size} біт):')
    print(f'  - Час генерації: {rsa_time:.6f} секунд')
    print(f'  - Довжина ключа: {len(public_key_bytes)} байтів')
    print(f'  - Ентропія: {entropy_public_key:.6f} біт на байт')
    print('=' * 40)


# Виконання тесту
evaluate_generation(byte_size=1024, rsa_size=2048)
```

Скрін виконання програми :

```
========================================
Тест: Генерація випадкових даних (1024 байтів):
Згенеровані дані: 3267276c1635b47c4e78954ca72b948907d12e5aad2891974ac5ca9e378cc7b6... (64 символи в hex)
• Час генерації: 0.000000 секунд
• Ентропія даних: 7.795693 біт на байт
----------------------------------------

========================================
Тест: Генерація RSA-ключів (2048 біт):
RSA-ключ (2048-бітний) успішно створений
• Час генерації: 0.185000 секунд
• Довжина публічного ключа: 451 байтів
• Ентропія публічного ключа: 5.849175 біт на байт
----------------------------------------

========================================
Результати порівняння:
Випадкові дані:
  - Час генерації: 0.000000 секунд
  - Ентропія: 7.795693 біт на байт

RSA-ключі (2048 біт):
  - Час генерації: 0.185000 секунд
  - Довжина ключа: 451 байтів
  - Ентропія: 5.849175 біт на байт
========================================
```

Використана література :

1. Maurer U. M. Fast generation of secure RSA-moduli with almost maximal diversity Proceedings of Eurocrypt '89. - P. 636-647.
2. Б. Шнайер Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си. – М.: Изд-во ТРИУМФ,2002. – 816 с.