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

Виконали:
студенти ФІ-31мн
Карабан А.
Шевченко Ю.
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:

У криптографії важливу роль відіграє генерація випадкових чисел, оскільки вони використовуються в різноманітних криптографічних алгоритмах, таких як генерація ключів, симетричне шифрування, підписування повідомлень та інші. Випадкові числа використовуються для забезпечення непередбачуваності в процесах шифрування і захисту інформації. Вони також використовуються для створення ключів для асиметричного шифрування, що є основою багатьох сучасних криптографічних протоколів.
Генерація випадкових байтів — це важливий процес у криптографії для створення ключів та інших випадкових даних. В Python для цього можна використовувати різні бібліотеки, однією з яких є os.urandom, що генерує криптографічно безпечні випадкові байти. Однак для аналізу цієї генерації важливо виміряти час, необхідний для її виконання, а також оцінити, наскільки добре ці байти є випадковими, вимірявши їх ентропію.
Ентропія — це міра випадковості або непередбачуваності даних. Вона оцінюється за допомогою формули Шеннона, де чим вище значення ентропії, тим більш випадкові дані. Відсутність ентропії означає, що дані можуть бути передбачуваними, що робить їх непридатними для використання в криптографії. У випадку випадкових байтів, які генеруються для шифрування, нам потрібно мати високу ентропію, щоб уникнути вразливостей.
RSA є одним з найбільш поширених алгоритмів для асиметричного шифрування. Він використовує пару ключів: публічний (для шифрування) та приватний (для дешифрування). Генерація RSA ключів також є важливою задачею в криптографії, і ми виміряємо час, необхідний для їх створення, а також оцінимо довжину публічного ключа та його ентропію. Для цього в Python ми використовуємо бібліотеку cryptography, яка дозволяє створювати ключі RSA різних розмірів.

Опис реалізації:

Функція generate_random_bytes генерує випадкові байти розміру `size` за допомогою бібліотеки os.urandom, що гарантує криптографічну безпеку. Вимірюється час, необхідний для генерації, а також виводиться перші 128 символів у вигляді шістнадцяткових значень для зручності перегляду.

Функція calculate_entropy розраховує ентропію даних, використовуючи формулу Шеннона. Для цього вона спочатку обчислює ймовірність кожного байта в даних, а потім використовує цю ймовірність для розрахунку ентропії.

Функція generate_rsa_keypair генерує пару RSA ключів за допомогою бібліотеки cryptography. Генерується приватний ключ за заданим розміром (key_size), а також вимірюється час генерації ключів.

Функція compare_random_and_rsa порівнює генерування випадкових байтів та генерування RSA ключів. Вона:

- Викликає функцію `generate_random_bytes` для генерації випадкових байтів
- Викликає функцію `generate_rsa_keypair` для генерації RSA ключів
- Обчислює ентропію для кожного з типів даних
- Порівнює результати — час виконання, ентропія для випадкових байтів і публічного ключа, а також довжина публічного ключа RSA

```
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


def generate_random_bytes(size):
    """Generate random bytes using a cryptography library and measure the time taken.

    Args:
        size (int): The number of random bytes to generate.

    Returns:
        tuple: Random bytes and time taken to generate them.
    """
    start_time = time.time()
    random_data = os.urandom(size)
    elapsed_time = time.time() - start_time
    print(f'Generated Random Bytes: {random_data.hex()[:128]}... (first 128 hex
characters)')

    return random_data, elapsed_time


def calculate_entropy(data):
    """Calculate the Shannon entropy of the given byte data.

    Args:
        data (bytes): The byte data to analyze.

    Returns:
        float: The entropy of the byte data in bits per byte.
    """
    probabilities = [data.count(byte) / len(data) for byte in set(data)]

    # Shannon's entropy formula: -sum(p * log2(p) for each probability p)
```

```
    return -sum(p * math.log2(p) for p in probabilities)


def generate_rsa_keypair(key_size=2048):
    """Generate an RSA key pair using a cryptography library and measure the time
taken.

    Args:
        key_size (int): The size of the RSA key in bits.

    Returns:
        tuple: The RSA key pair and time taken to generate it.
    """
    start_time = time.time()
    key = rsa.generate_private_key(public_exponent=65537, key_size=key_size,
backend=default_backend())
    elapsed_time = time.time() - start_time
    print(f'Generated RSA Key Pair ({key_size}-bit)')

    return key, elapsed_time


def compare_random_and_rsa(size, key_size):
    """Compare random byte generation and RSA key generation in terms of time and
entropy.

    Args:
        size (int): The number of bytes for random data generation.
        key_size (int): The size of the RSA key to generate (in bits).
    """
    # Random byte generation test
    print(f"\n{'=' * 40}\nTesting Random Byte Generation ({size} bytes):")
    random_bytes, random_time = generate_random_bytes(size)
    random_entropy = calculate_entropy(random_bytes)
    print(f'• Generation Time: {random time:.6f} seconds')
    print(f'• Entropy: {random_entropy:.6f} bits per byte')
    print(str('-' * 40))

    # RSA key generation test
    print(f"\n{'=' * 40}\nTesting RSA Key Generation ({key_size}-bit):")
    rsa_key, rsa_time = generate_rsa_keypair(key_size)
    rsa_public_key = rsa_key.public_key().public_bytes(
        encoding=serialization.Encoding.PEM,
format=serialization.PublicFormat.SubjectPublicKeyInfo
    )
    rsa_key_entropy = calculate_entropy(rsa_public_key)

    print(f'• Generation Time: {rsa_time:.6f} seconds')
    print(f'• Public Key Length: {len(rsa_public_key)} bytes')
    print(f'• Public Key Entropy: {rsa_key_entropy:.6f} bits per byte')
    print(str('-' * 40))

    # Final summary and comparison
    print(f"\n{'=' * 40}\nSummary Comparison:")
    print('Random Byte Generation:')
    print(f'  - Time: {random_time:.6f} seconds')
    print(f'  - Entropy: {random_entropy:.6f} bits per byte')
    print(f'\nRSA Key Generation ({key_size}-bit):')
    print(f'  - Time: {rsa_time:.6f} seconds')
    print(f'  - Key Length: {len(rsa_public_key)} bytes')
    print(f'  - Entropy: {rsa_key_entropy:.6f} bits per byte')
    print(str('=' * 40))


# Example usage of the comparison function
compare_random_and_rsa(size=1024, key_size=2048)
```

Приклад виконання програми:

```
D:\Study\Sem_11\mrkm24-25\venv\Scripts\python.exe D:\Study\Sem_11\mrkm24-25\lab2\FI-31mn-Karaban-Shevchenko\Lab_2.py

========================================
Testing Random Byte Generation (1024 bytes):
Generated Random Bytes: 2379ea75c28b6854f0efff0b512ec82297b3953ba8a940d7bdaf9083d2c4197c80a51243838d26f39a2b19526d289bfb5d05a09ac429c203d2c1dde5a84c1305... (first 128 hex characters)
• Generation Time: 0.000000 seconds
• Entropy: 7.810975 bits per byte
----------------------------------------

========================================
Testing RSA Key Generation (2048-bit):
Generated RSA Key Pair (2048-bit)
• Generation Time: 0.020986 seconds
• Public Key Length: 451 bytes
• Public Key Entropy: 5.863671 bits per byte
----------------------------------------

========================================
Summary Comparison:
Random Byte Generation:
  - Time: 0.000000 seconds
  - Entropy: 7.810975 bits per byte

RSA Key Generation (2048-bit):
  - Time: 0.020986 seconds
  - Key Length: 451 bytes
  - Entropy: 5.863671 bits per byte
========================================

Process finished with exit code 0
```

Висновки:

У ході виконання цієї лабораторної роботи ми дослідили два ключових аспекти в галузі криптографії: генерацію випадкових байтів та створення пари RSA-ключів. Ми розробили програму, яка порівнює ці два процеси зі кількома параметрами, включаючи час генерації та ентропію, що дозволяє оцінити якість генерованих даних. Ось які результати ми отримали:

- Генерація випадкових байтів:
    - За допомогою бібліотеки os.urandom ми генерували 1024 байти випадкових даних. Програма продемонструвала, що час генерації виявився мінімальним (0.000000 секунд, що є результатом округлення для дуже швидкого процесу)
    - Ентропія згенерованих байтів становила 7.81 біт на байт, що є дуже хорошим показником для випадкових даних
- Генерація пари RSA-ключів:
    - Генерація пари RSA-ключів із довжиною 2048 біт зайняла 0.0209 секунд, що вказує на значно більший час обробки порівняно з генерацією випадкових байтів
    - Довжина публічного ключа виявилася рівною 451 байту
    - Ентропія публічного ключа становила 5.86 біт на байт, що менше, ніж у випадкових байтів, що є очікуваним результатом, оскільки криптографічний ключ повинен мати певну структуру, що впливає на його ентропію

Як показали результати експерименту, генерація випадкових байтів є значно швидшою за генерацію RSA-ключів, але публічні ключі мають нижчу ентропію, що є наслідком необхідності забезпечення певної структури та впорядкованості в криптографії. Ентропія випадкових байтів досягає досить високого значення, що свідчить про їх високу випадковість та непередбачуваність. RSA-ключі хоч і є більш безпечними завдяки своїй

складності, мають меншу ентропію, що є нормальним через обмеження, пов’язані з їх структурою.

Ця лабораторна робота допомогла нам глибше зрозуміти процеси генерації випадкових чисел і створення криптографічних ключів. Крім того, ми отримали практичний досвід у вимірюванні ефективності та аналізі ентропії криптографічних даних, що є важливою складовою у забезпеченні безпеки інформаційних систем.