Лабораторна робота № 2.
Реалізація алгоритмів генерації ключів гібридних криптосистем.

Виконали: Журибіда Ю.Б., Швець М.К., Шостак А.А.

Мета роботи: Дослідження алгоритмів генерації псевдовипадкових послідовностей, тестування простоти чисел і генерації простих чисел із точки зору їхньої ефективності за часом і можливості використання для гененерації ключів асиметричних криптосистем.
Завдання:
Підгрупа 2с. Бібліотека PyCrypto під Linux платформу.

Порівняння:

	Random Byte Generation	RSA Key Generation
Опис функції	Генерує послідовність криптографічно захищених випадкових байтів.	Генерує 2048-бітну пару ключів RSA.
Алгоритм	Використовує криптографічний генератор псевдовипадкових чисел (CSPRNG) для створення безпечних і непередбачуваних байтів.	Використовує ймовірнісні алгоритми для генерування двох великих простих чисел і обчислення їх модуля (n), а також загального (e) і приватного (d) показника степеня.
Вхід	`size` (integer) із зазначенням кількості байт, які потрібно згенерувати.	`key_size` (integer), зазвичай 2048 або 4096 біт.
Вихід функції	Випадкова послідовність байт заданого розміру.	Об'єкт ключа RSA з модулем n, відкритим ключем e та закритим ключем d.
Код поернення	Успіх: Згенеровано випадкові байти.Failure: Згенерує виключення, якщо генератор вийшов з ладу.	Успіх: Пара ключів згенерована.Failure: Згенерує виключення через невірні параметри або проблеми з пам'яттю.
Приклад	Input: `size = 256` Output: `a422e95254b3...` (truncated, 256 bytes).	Input: `key_size = 2048` Output: Public key length = 450 bytes, private key stored securely.

Код:

```
from Crypto.Random import get_random_bytes
import time
```

```
import math

def generate_random_bytes(size):
    """Generate random bytes and measure the time taken."""
    start_time = time.time()
    random_data = get_random_bytes(size)
    elapsed_time = time.time() - start_time
    print(f"Generated Random Bytes: {random_data.hex()[:64]}...
(truncated)")
    return random_data, elapsed_time

def calculate_entropy(data):
    """Calculate the Shannon entropy of the given byte data."""
    probabilities = [data.count(byte) / len(data) for byte in set(data)]
    return -sum(p * math.log2(p) for p in probabilities)

from Crypto.PublicKey import RSA

def generate_rsa_keypair(key_size=2048):
    """Generate an RSA key pair and measure the time taken."""
    start_time = time.time()
    key = RSA.generate(key_size)
    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."""
    print("\n--- Testing Random Byte Generation ---")
    random_bytes, random_time = generate_random_bytes(size)
    random_entropy = calculate_entropy(random_bytes)

    print("\n--- Testing RSA Key Generation ---")
    rsa_key, rsa_time = generate_rsa_keypair(key_size)

    print("\n--- Comparison Results ---")
    print(f"Random Bytes Generation (Size: {size} bytes):")
    print(f"  Time Taken: {random_time:.6f} seconds")
    print(f"  Entropy: {random_entropy:.6f} bits per byte")

    print(f"\nRSA Key Generation ({key_size}-bit):")
    print(f"  Time Taken: {rsa_time:.6f} seconds")
```

```
    print(f"  Public Key Length: {len(rsa_key.publickey().export_key())}
bytes")
    print(f"  Entropy:
{calculate_entropy(rsa_key.publickey().export_key())} bits per byte")
```

Вихід:

```
--- Testing Random Byte Generation ---
Generated Random Bytes:
a72a20ea89f5f09bd4f5cdfbdda7d9fc92cd671225b9b152562a65d00161299f...
(truncated)

--- Testing RSA Key Generation ---
Generated RSA Key Pair (2048-bit)

--- Comparison Results ---
Random Bytes Generation (Size: 256 bytes):
  Time Taken: 0.000428 seconds
  Entropy: 7.155945 bits per byte

RSA Key Generation (2048-bit):
  Time Taken: 0.311241 seconds
  Public Key Length: 450 bytes
  Entropy: 5.8780156621228645 bits per byte
```

Оцінка результатів:

Метрика	Random Byte Generation	RSA Key Generation
Process Description	Generates 256 cryptographically secure random bytes.	Generates a 2048-bit RSA key pair.
Execution Time	0.000428 seconds	0.311241 seconds
Output Size	256 bytes	Public Key: 450 bytes
Entropy (Randomness)	7.155945 bits per byte	5.878016 bits per byte
Algorithm Complexity	Низька	Висока
Primary Use	Використовується для безпечної генерації випадкових чисел, заповнення ключів або створення nonce.	Асиметричне шифрування та цифрові підписи.

Observations	Дуже висока ентропія, що свідчить про високу якість випадковості.	Трохи менша ентропія через структурні обмеження ключів RSA.

Висновки з таблиці

- Ефективність: Генерація випадкових байт відбувається набагато швидше, ніж генерація ключа RSA, як і очікувалося, завдяки меншій алгоритмічній складності.
- Ентропія: Генерація випадкових байт забезпечує вищу ентропію, що свідчить про більшу випадковість, в той час як RSA-ключі мають нижчу ентропію через вимоги до формату та математичної структури.
- Вихідний розмір: Випадкові байти створюють фіксований розмір на основі вхідних даних, в той час як довжина ключа RSA залежить від криптографічних стандартів.

Можливість використання рандомної послідовності для асиметричних систем:

Розподіл ключів:

- Випадкові байти можуть бути використані в якості насіння для генерації великих простих чисел, необхідних для асиметричних ключів (наприклад, ключів RSA).
- Випадкові байти з високою ентропією мають вирішальне значення для забезпечення непередбачуваності згенерованих ключів.

Генерація nonce або salt:

- У криптографічних протоколах випадкові байти часто використовуються для генерації нонсів (чисел, використаних один раз) або значень солі, які додають випадковості в схеми шифрування і запобігають атакам повторного відтворення.