Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Фізико-технічний інститут

ЛАБОРАТОРНА РОБОТА

з

Методи реалізації криптографічних механізмів

Виконав:

студент 6 курсу ФТІ

групи ФБ-21мн

Мельник Дмитро

Скидан Данилов

київ-2023

Для генерації простих чисел я використовую бібліотеку PyCrypto
А саме функцію getPrime

```python
from Crypto.Util.number import getPrime, isPrime


if __name__ == "__main__":
    prime_number = getPrime(256)

```

Сама функція генерує випадкове ціле число з проміжку у бітах який ми вказали. першим параметром

```python
def getPrime(N, randfunc=None):
    """getPrime(N:int, randfunc:callable):long
    Return a random N-bit prime number.

    If randfunc is omitted, then Random.new().read is used.
    """
    if randfunc is None:
        _import_Random()
        randfunc = Random.new().read

    number=getRandomNBitInteger(N, randfunc) | 1
    while (not isPrime(number, randfunc=randfunc)):
        number=number+2
    return number

```

Генерація випадкового числа Може виконуватись генератором випадкових чисел які ми самі вкажемо. Чи використовувати генератор самої бібліотеки

```python
def getRandomInteger(N, randfunc=None):
    """getRandomInteger(N:int, randfunc:callable):long
    Return a random number with at most N bits.

    If randfunc is omitted, then Random.new().read is used.

    This function is for internal use only and may be renamed or
    the future.
    """
    if randfunc is None:
        _import_Random()
        randfunc = Random.new().read

    S = randfunc(N>>3)
    odd_bits = N % 8
    if odd_bits != 0:
        char = ord(randfunc(1)) >> (8-odd_bits)
        S = bchr(char) + S
    value = bytes_to_long(S)
    return value
```

Після генерації дане число перевіряється за на простоту isPrime

```python
def isPrime(N, false_positive_prob=1e-6, randfunc=None):
    """isPrime(N:long, false_positive_prob:float, randfunc:callable):bool
    Return true if N is prime.

    The optional false_positive_prob is the statistical probability
    that true is returned even though it is not (pseudo-prime).
    It defaults to 1e-6 (less than 1:1000000).
    Note that the real probability of a false-positive is far less. This is
    just the mathematically provable limit.

    If randfunc is omitted, then Random.new().read is used.
    """
    if _fastmath is not None:
        return _fastmath.isPrime(long(N), false_positive_prob, randfunc)

    if N < 3 or N & 1 == 0:
        return N == 2
    for p in sieve_base:
        if N == p:
            return 1
        if N % p == 0:
            return 0

    rounds = int(math.ceil(-math.log(false_positive_prob)/math.log(4)))
    return _rabinMillerTest(N, rounds, randfunc)
```

У самій перевірці є дата сет вже простих чисел

sieve_base це кортеж усіх чисел простих саме перша перевірка перевіряє чи наше число є там. Якщо так тоді вже повретається відповідь.
Якщо наше число не є у данному тада сеті.
Тоді виконується тест Міллера на перевірку простоти числа.

```
def _rabinMillerTest(n, rounds, randfunc=None):
    """_rabinMillerTest(n:long, rounds:int, randfunc:callable):int
    Tests if n is prime.
    Returns 0 when n is definitly composite.
    Returns 1 when n is probably prime.
    Returns 2 when n is definitly prime.

    If randfunc is omitted, then Random.new().read is used.

    This function is for internal use only and may be renamed or removed in
    the future.
    """
    # check special cases (n==2, n even, n < 2)
    if n < 3 or (n & 1) == 0:
        return n == 2
    # n might be very large so it might be beneficial to precalculate n-1
    n_1 = n - 1
    # determine m and b so that 2**b * m = n - 1 and b maximal
    b = 0
    m = n_1
    while (m & 1) == 0:
        b += 1
        m >>= 1

    tested = []
    # we need to do at most n-2 rounds.
    for i in xrange (min (rounds, n-2)):
        # randomly choose a < n and make sure it hasn't been tested yet
        a = getRandomRange (2, n, randfunc)
        while a in tested:
            a = getRandomRange (2, n, randfunc)
        tested.append (a)
        # do the rabin-miller test
        z = pow (a, m, n) # (a**m) % n
        if z == 1 or z == n_1:
            continue
        composite = 1
        for r in xrange (b):
            z = (z * z) % n
            if z == 1:
                return 0
```

Висновок
Даний механізм підходить для швидкої перевірки простих числе. Також цей алгоритм має певні недоліки.
Вразливий до чисел Кармайкла
Потребує багато ресурсів.
Якість результатів залежить від кількості ітерацій.

Дана бібліотека застаріла її не слід використовувати.