Чалий Олексій, ФБ-21 мн
Реалізація алгоритмів генерації ключів гібридних криптосистем.

Лаб 2

Варіант 2В Бібліотека PyCrypto під Linux платформу.

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

Зміст

Для Pycrypto, список функцій для реалізації ПВЧ можна подивитися тут
https://www.dlitz.net/software/pycrypto/api/2.6/Crypto.Random.random.StrongRandom-class.html

Ось їх список, їх 6-ть штук

Home Trees Indices Help

Package Crypto :: Package Random :: Module random :: Class StrongRandom

Class StrongRandom

```
object --+
         |
        StrongRandom
```

Instance Methods	
	__init__(self, rng=None, randfunc=None) x.__init__(...) initializes x; see help(type(x)) for signature
	getrandbits(self, k) Return a python long integer with k random bits.
	randrange(self, *args) randrange([start,] stop[, step]): Return a randomly-selected element from range(start, stop, step).
	randint(self, a, b) Return a random integer N such that a <= N <= b.
	choice(self, seq) Return a random element from a (non-empty) sequence.
	shuffle(self, x) Shuffle the sequence in place.
	sample(self, population, k) Return a k-length list of unique elements chosen from the population sequence.

Функції генерації ПВЧ

getranbits(self, k)

Згідно опису, повертає long integer з k випадковими бітами

Алгоритм:

Вхідні дані: Функція приймає один аргумент k, який являє собою довжину (у бітах) бажаної послідовності випадкових бітів. Значення k має бути додатним цілим числом.

Генерація випадкових бітів: Функція генерує послідовність випадкових бітів, довжиною k, з використанням криптографічно безпечних джерел випадковості, доступних в операційній системі.

Вихідні дані: Функція повертає ціле число, що являє собою випадкову послідовність бітів заданої довжини k.

Код повернення: не повертає коди помилок, оскільки вона не обробляє винятки, пов'язані з нестачею ресурсів або іншими проблемами.

Код та результат

```
┌──(kali㉿kali)-[~/lab2]
└─$ cat getrandbits.py
from Crypto.Random import random

k = 32
random_bits = random.getrandbits(k)
print(f'{k}-bit random number: {random_bits}')

┌──(kali㉿kali)-[~/lab2]
└─$ python3 getrandbits.py
32-bit random number: 3210373818

┌──(kali㉿kali)-[~/lab2]
└─$ python3 getrandbits.py
32-bit random number: 1700718054
```

randrange(self, *args)

Замість *args, можна підставити параметри start, stop, step, де start – задає мінімальне значення ПВЧ, stop – задає макс. Значення ПВЧ, step задає шаг, з яким в проміжку між мін макс може вибиратися ПВЧ

Алгоритм:

Вхідні дані: Функція приймає змінну кількість аргументів *args, що дозволяє передавати різну кількість параметрів залежно від потреб:

- Якщо передано один аргумент stop, функція генерує випадкове число від 0 (включно) до stop (виключно).
- Якщо передано два аргументи start і stop, функція генерує випадкове число від start (включно) до stop (виключно).
- Якщо передано три аргументи start, stop, і step, функція генерує випадкове число від start (включно) до stop (виключно) із заданим кроком step.

Генерація випадкового числа: Функція використовує криптографічно безпечні джерела випадковості для генерації випадкових чисел у заданому діапазоні.

Вихідні дані: Функція повертає випадкове ціле число із зазначеного діапазону, що задовольняє заданим параметрам.

Код повернення: не повертає коди помилок, оскільки вона не обробляє винятки, пов'язані з нестачею ресурсів або іншими проблемами.

Код та результат

```
┌──(kali㉿kali)-[~/lab2]
└─$ cat randrange.py
from Crypto.Random.random import randrange

result1 = randrange(10)
result2 = randrange(5, 15)
result3 = randrange(0, 100, 10)

print(result1)
print(result2)
print(result3)

┌──(kali㉿kali)-[~/lab2]
└─$ python3 randrange.py
3
9
0

┌──(kali㉿kali)-[~/lab2]
└─$ python3 randrange.py
1
14
60
```

randint(self, a, b)

Алгоритм:

Вхідні дані: Функція приймає два аргументи a і b, що представляють нижню і верхню межі діапазону (включно). Функція генерує випадкове ціле число, яке перебуває включно в зазначеному діапазоні.

Генерація випадкового числа: Функція використовує криптографічно безпечні джерела випадковості для генерації випадкового цілого числа. Гарантується, що згенероване число буде розподілено рівномірно в заданому діапазоні.

Вихідні дані: Функція повертає випадкове ціле число, яке знаходиться включно в заданому діапазоні [a, b].

Код повернення: не повертає коди помилок, оскільки вона не обробляє винятки, пов'язані з нестачею ресурсів або іншими проблемами.

Код та результат

choice(self, seq)

Алгоритм:

Вхідні дані: Функція приймає один аргумент seq, що являє собою послідовність елементів, з якої потрібно вибрати випадковий елемент.

Генерація випадкового елемента: Функція використовує криптографічно безпечні джерела випадковості для генерації випадкового індексу (позиції) в заданій послідовності seq.

Вихідні дані: Функція повертає елемент послідовності seq, обраний випадковим чином.

Код повернення: не повертає коди помилок, оскільки вона не обробляє винятки, пов'язані з нестачею ресурсів або іншими проблемами.

Код та результат

```
┌──(kali㉿kali)-[~/lab2]
└─$ cat choice.py
from Crypto.Random.random import choice

sequence = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
random_element = choice(sequence)

print(f'random element from sequence: {random_element}')

┌──(kali㉿kali)-[~/lab2]
└─$ python3 choice.py
random element from sequence: 6

┌──(kali㉿kali)-[~/lab2]
└─$ python3 choice.py
random element from sequence: 2

┌──(kali㉿kali)-[~/lab2]
└─$ python3 choice.py
random element from sequence: 1
```

shuffle(self, x)

Алгоритм:

Вхідні дані: Функція приймає один аргумент x, який являє собою послідовність елементів, яку потрібно перемішати. Ця послідовність може бути списком, кортежем, рядком тощо.

Перемішування елементів: Функція змінює порядок елементів у послідовності x, переставляючи їх випадковим чином. Це робиться з використанням криптографічно безпечних джерел випадковості.

Вихідні дані: Функція не повертає нову перемішану послідовність. Вона змінює послідовність x на місці.

Код повернення: не повертає коди помилок, оскільки вона не обробляє винятки, пов'язані з нестачею ресурсів або іншими проблемами.

Код та результат

```
┌──(kali㉿kali)-[~/lab2]
└─$ cat shuffle.py
from Crypto.Random.random import shuffle

sequence = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
shuffled_sequence = list(sequence)
shuffle(shuffled_sequence)

print(f'Shuffled sequence: {shuffled_sequence}')

┌──(kali㉿kali)-[~/lab2]
└─$ python3 shuffle.py
Shuffled sequence: [9, 1, 5, 3, 8, 7, 10, 4, 2, 6]

┌──(kali㉿kali)-[~/lab2]
└─$ python3 shuffle.py
Shuffled sequence: [9, 7, 3, 4, 6, 1, 8, 5, 10, 2]

┌──(kali㉿kali)-[~/lab2]
└─$ python3 shuffle.py
Shuffled sequence: [8, 3, 6, 5, 9, 7, 2, 1, 4, 10]
```

sample(self, population, k)

Алгоритм:

Вхідні дані: Функція приймає два аргументи:

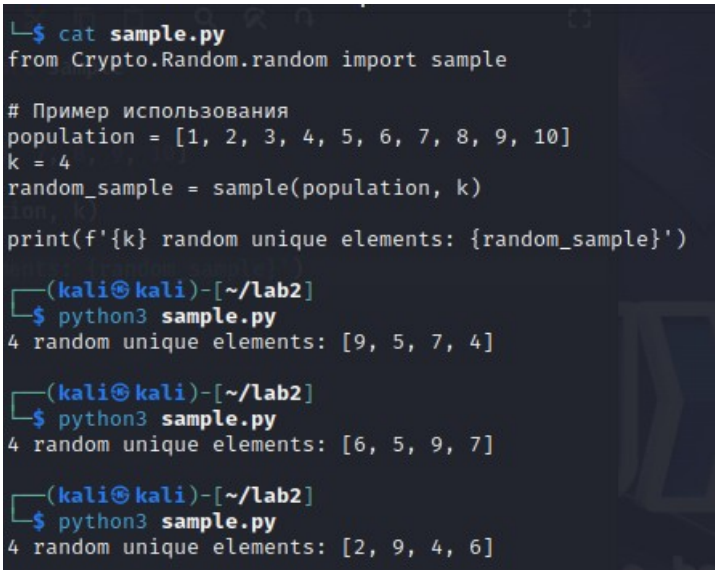
- population: "Популяція" або послідовність елементів, з якої потрібно вибрати елементи.
- k: Кількість унікальних елементів, які потрібно вибрати випадковим чином. k має бути меншим або дорівнювати довжині population.

Вибір випадкових елементів: Функція вибирає k унікальних елементів з population, використовуючи криптографічно безпечні джерела випадковості. Це означає, що кожен елемент буде обрано лише один раз у результаті вибору.

Вихідні дані: Функція повертає список, що містить вибрані випадкові елементи.

Код повернення: не повертає коди помилок, оскільки вона не обробляє винятки, пов'язані з нестачею ресурсів або іншими проблемами.

Код та результат



Генератор ключів

Взагалі, для цієї задачі раніше в 2012 році був спеціальний пакет призначений для різних ОС

Package OSRNG

Provides a platform-independent interface to the random number generators supplied by various operating systems.

Submodules

- Crypto.Random.OSRNG.fallback
- Crypto.Random.OSRNG.nt
- Crypto.Random.OSRNG.posix
- Crypto.Random.OSRNG.rng_base

Variables

	`__revision__` = 'Id'
	`__package__` = 'Crypto.Random.OSRNG'

Home Trees Indices Help

Generated by Epydoc 3.0.1 on Thu May 24 09:02:36 2012

Для Linux потрібно б було використати пакет ...posix и в неї функцію нижче

Class DevURandomRNG

object --+
|
rng_base.BaseRNG --+
|
DevURandomRNG

Instance Methods

	`__init__`(self, devname=None) x.__init__(...) initializes x; see help(type(x)) for signature

Inherited from `object`: `__delattr__`, `__format__`, `__getattribute__`, `__hash__`, `__new__`, `__reduce__`, `__reduce_ex__`, `__repr__`, `__setattr__`, `__sizeof__`, `__str__`, `__subclasshook__`

Inherited from rng_base.BaseRNG

	`__del__`(self)
	`__enter__`(self)
	`__exit__`(self) PEP 343 support
	`close`(self)
	`flush`(self)
	`read`(self, N=-1) Return N bytes from the RNG.

Properties

Inherited from `object`: `__class__`

Method Details

__init__(self, devname=None)
(Constructor)
x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__
(inherited documentation)

Він використовує системні джерела ентропії, такі як /dev/urandom, щоб гарантувати надійну генерацію ключів.

Якби, ця функція ще працювала б і її не вилучили з пакету, то код реалізації був приблизно такий:

from Crypto.PublicKey import RSA

from Crypto.Random.OSRNG import DevURandomRNG

Створюємо об'єкт генератора випадкових чисел з використанням DevURandomRNG

rng = DevURandomRNG()

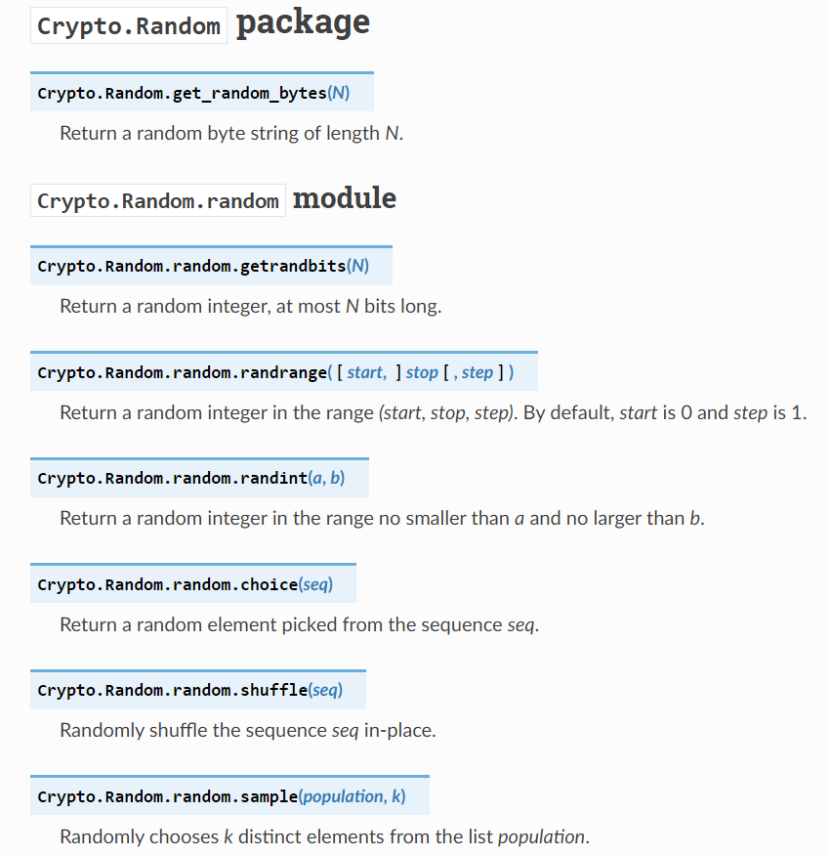
Генеруємо пару ключів RSA з використанням цього генератора

key = RSA.generate(2048, rng)

Виводимо відкритий ключ

print(key.publickey().export_key())

Доказом, що її прибрали є поточний список пакетів на сайті
https://pycryptodome.readthedocs.io/en/latest/src/random/random.html



`Crypto.Random` package

`Crypto.Random.get_random_bytes`(*N*)

Return a random byte string of length *N*.

`Crypto.Random.random` module

`Crypto.Random.random.getrandbits`(*N*)

Return a random integer, at most *N* bits long.

`Crypto.Random.random.randrange`([*start*,] *stop* [, *step*])

Return a random integer in the range *(start, stop, step)*. By default, *start* is 0 and *step* is 1.

`Crypto.Random.random.randint`(*a*, *b*)

Return a random integer in the range no smaller than *a* and no larger than *b*.

`Crypto.Random.random.choice`(*seq*)

Return a random element picked from the sequence *seq*.

`Crypto.Random.random.shuffle`(*seq*)

Randomly shuffle the sequence *seq* in-place.

`Crypto.Random.random.sample`(*population*, *k*)

Randomly chooses *k* distinct elements from the list *population*.

Висновки

Бібліотека PyCryptodome є добре відомою та широко використовується бібліотекою для розробки криптографічних програм. Сам по собі генератор ПВЧ від PyCryptodome може бути використаний у криптографічних системах, і він має високий рівень стійкості. Проте, при використанні в критичних криптографічних системах, які вимагають максимальної стійкості та безпеки, рекомендується враховувати наступні аспекти:

Джерело ентропії: Важливо використовувати системні джерела ентропії, такі як /dev/urandom, для підвищення стійкості.

Аудит та перевірка коду: Перед використанням генератора ПВЧ в критичних системах слід провести аудит та перевірку коду бібліотеки, щоб переконатися в його відповідності криптографічним стандартам та відсутності вразливостей.

Оновлення та моніторинг: Система повинна бути постійно оновлювана та моніторинг за можливими загрозами безпеці повинен бути постійним.

Загальна рекомендація полягає в тому, що важливо вибирати та налаштовувати генератор ПВЧ з урахуванням конкретних потреб і загроз вашої криптографічної системи.