

МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ  
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»  
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Лабораторна робота №2  
З дисципліни «МЕТОДИ РЕАЛІЗАЦІЇ КРИПТОГРАФІЧНИХ  
МЕХАНІЗМІВ»

Роботу виконали  
студенти групи ФІ-21мн, ФТІ  
Татенко Вадим  
Хмелевський Святослав  
Кірсенко Єгор

2023

**Мета роботи:** Аналіз стійкості реалізацій ПВЧ та генераторів ключів для бібліотеки PyCryptoDome під Linux платформу.

**Оформлення результатів работ:** Опис функції генерації ПВП та ключів бібліотеки PyCryptoDome з описом алгоритму, вхідних та вихідних даних, кодів повернення. Контрольний приклад роботи з функціями.

**Хід роботи :**

\*Посилання га [GitHub](#) бібліотеки\*

Опис функції генерації ПВП:

```
pycryptodome / lib / Crypto / Random / __init__.py
Code Blame 57 lines (43 loc) · 1.77 KB Code 55% faster with GitHub Copilot
22
23 __all__ = ['new', 'get_random_bytes']
24
25 from os import urandom
26
27 class _UrandomRNG(object):
28
29     def read(self, n):
30         """Return a random byte string of the desired size."""
31         return urandom(n)
32
33     def flush(self):
34         """Method provided for backward compatibility only."""
35         pass
36
37     def reinit(self):
38         """Method provided for backward compatibility only."""
39         pass
40
41     def close(self):
42         """Method provided for backward compatibility only."""
43         pass
44
45
46 def new(*args, **kwargs):
47     """Return a file-like object that outputs cryptographically random bytes."""
48     return _UrandomRNG()
49
50
51 def atfork():
52     pass
53
54
55 #: Function that returns a random byte string of the desired size.
56 get_random_bytes = urandom
```

Бібліотека PyCryptoDome для генерації псевдовипадкових чисел/послідовностей (ПВП) використовує /dev/urandom, як генератор псевдовипадкових чисел.

В бібліотеці використовується `urandom` з бібліотеки `os`. Таким чином, що можна викликавши функцію `get_random_bytes`, задавши як параметр функції кількість байт, отримати псевдорандомну послідовність.

Контрольний приклад роботи функції:

```
!cat /etc/*release

DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=22.04
DISTRIB_CODENAME=jammy
DISTRIB_DESCRIPTION="Ubuntu 22.04.3 LTS"
PRETTY_NAME="Ubuntu 22.04.3 LTS"
NAME="Ubuntu"
VERSION_ID="22.04"
VERSION="22.04.3 LTS (Jammy Jellyfish)"
VERSION_CODENAME=jammy
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
UBUNTU_CODENAME=jammy

[ ] !pip install pycryptodome

[1] from Crypto.Random import get_random_bytes

[4] randBytes = get_random_bytes(16)

print(randBytes)

b'9\xa7w\xfa\x9bF\xb4C\x11\x1f\xbft\x08)\x85\x1c'
```

### Опис функції генерації ключів:

Вхідні дані:

- біти (ціле число): довжина ключа RSA або розмір модуля RSA в бітах. Він має бути не менше 1024, рекомендовано 2048 або більше.
- randfunc (необов'язковий параметр, if randfunc is None: get\_random\_bytes): функція, яка повертає випадкові байти. Дефолтним є Crypto.Random.get\_random\_bytes.
- e (ціле, необов'язкове): Публічна показник (exponent) RSA, непарне додатне ціле число, яке за замовчуванням зазвичай дорівнює 65537.

Алгоритм:

- Як пишуть самі розробники в себе в коментарях, алгоритм генерації впливає з стандарту NIST FIPS 186-4, який визначає набір алгоритмів, які можна використовувати під час створення цифрового підпису один з яких і є RSA.
- Модуль RSA (n) є добутком двох простих чисел, кожне з яких проходить тест Міллера-Рабіна який визначає чи є число простим, та тест Лукаса, який також перевіряє число на простоту. (добавить скрин кода тестов)
- Функція генерує два прості числа, p і q, добуток яких n має потрібну кількість бітів.
- Використовується публічний показник RSA (e), який зазвичай є невеликим числом, наприклад 65537 (по дефолту).
- Компонент приватного ключа (d) обчислюється як мультиплікативне обернення e за модулем найменшого спільного кратного (p-1) і (q-1).

```
n = p * q
lcm = (p - 1).lcm(q - 1)
d = e.inverse(lcm)
```

Вихідні дані:

- Об'єкт ключа RSA (класу RsaKey), який містить як приватний, так і публічний ключі.

```
class RsaKey(object):
    r"""Class defining an actual RSA key.
    Do not instantiate directly.
    Use :func:`generate`, :func:`construct` or :func:`import_key` instead.

    :ivar n: RSA modulus
    :vartype n: integer

    :ivar e: RSA public exponent
    :vartype e: integer

    :ivar d: RSA private exponent
    :vartype d: integer

    :ivar p: First factor of the RSA modulus
    :vartype p: integer
```

```
:ivar q: Second factor of the RSA modulus
:vartype q: integer

:ivar invp: Chinese remainder component (:math:`p^{-1} \text{mod } q` )
:vartype invp: integer

:ivar invq: Chinese remainder component (:math:`q^{-1} \text{mod } p` )
:vartype invq: integer

:ivar u: Same as ``invp``
:vartype u: integer

:undocumented: exportKey, publickey
"""
```

```
428 def generate(bits, randfunc=None, e=65537):
429     """Create a new RSA key pair.
430
431     The algorithm closely follows NIST `FIPS 186-4`_ in its
432     sections B.3.1 and B.3.3. The modulus is the product of
433     two non-strong probable primes.
434     Each prime passes a suitable number of Miller-Rabin tests
435     with random bases and a single Lucas test.
436
437     Args:
438         bits (integer):
439             Key length, or size (in bits) of the RSA modulus.
440             It must be at least 1024, but **2048 is recommended.**
441             The FIPS standard only defines 1024, 2048 and 3072.
442         randfunc (callable):
443             Function that returns random bytes.
444             The default is :func:`Crypto.Random.get_random_bytes`.
445         e (integer):
446             Public RSA exponent. It must be an odd positive integer.
447             It is typically a small number with very few ones in its
448             binary representation.
449             The FIPS standard requires the public exponent to be
450             at least 65537 (the default).
451
452     Returns: an RSA key object (:class:`RsaKey`, with private key).
453
454     .. _FIPS 186-4: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf
455     """
456
457     if bits < 1024:
458         raise ValueError("RSA modulus length must be >= 1024")
459     if e % 2 == 0 or e < 3:
460         raise ValueError("RSA public exponent must be a positive, odd integer larger than 2.")
461
462     if randfunc is None:
463         randfunc = Random.get_random_bytes
464
465     d = n = Integer(1)
466     e = Integer(e)
```

```

428     def generate(bits, randfunc=None, e=65537):
467
468         while n.size_in_bits() != bits and d < (1 << (bits // 2)):
469             # Generate the prime factors of n: p and q.
470             # By construction, their product is always
471             # 2^{bits-1} < p*q < 2^bits.
472             size_q = bits // 2
473             size_p = bits - size_q
474
475             min_p = min_q = (Integer(1) << (2 * size_q - 1)).sqrt()
476             if size_q != size_p:
477                 min_p = (Integer(1) << (2 * size_p - 1)).sqrt()
478
479             def filter_p(candidate):
480                 return candidate > min_p and (candidate - 1).gcd(e) == 1
481
482             p = generate_probable_prime(exact_bits=size_p,
483                                         randfunc=randfunc,
484                                         prime_filter=filter_p)
485
486             min_distance = Integer(1) << (bits // 2 - 100)
487
488             def filter_q(candidate):
489                 return (candidate > min_q and
490                         (candidate - 1).gcd(e) == 1 and
491                         abs(candidate - p) > min_distance)
492
493             q = generate_probable_prime(exact_bits=size_q,
494                                         randfunc=randfunc,
495                                         prime_filter=filter_q)
496
497             n = p * q
498             lcm = (p - 1).lcm(q - 1)
499             d = e.inverse(lcm)
500
501             if p > q:
502                 p, q = q, p
503
504             u = p.inverse(q)
505
506             return RsaKey(n=n, e=e, d=d, p=p, q=q, u=u)

```

$p * u = 1 \bmod q \sim u = p.\text{inverse}(q)$

Використання:

```
[6] from Crypto.PublicKey import RSA

key = RSA.generate(2048)
private_key = key.export_key()
public_key = key.publickey().export_key()
print(f'Public key: {public_key} \n Private key: {private_key}')
```

```
Public key: b'-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAO\nPrivate key: b'-----BEGIN RSA PRIVATE KEY-----\nMIIEogIBAAKCAQEA...
```

### Висновки:

Під час виконання роботи було проаналізовано алгоритм функції генерації ПВП та алгоритм функції генерації приватних та публічних ключів (RSA) у бібліотеці PyCryptoDome. Під час роботи було виявлено, що PyCryptoDome використовує /dev/urandom для генерації псевдовипадкової послідовності послідовностей. А алгоритм генерації RSA ключів впливає з стандарту NIST FIPS 186-4, який задає правила реалізації та вимоги для забезпечення сек'юрності.