Національний технічний університет України
«Київський політехнічний інститут ім. Ігоря Сікорського»
Навчально-науковий фізико-технічний інститут

Комп'ютерний практикум №2 з курсу «Методи реалізації криптографічних механізмів»

Тема: «Реалізація алгоритмів генерації ключів гібридних криптосистем»

Виконав студент
групи ФІ-22мн
Максименко М.С.

Перевірила:
Байденко П. В.

Київ — 2023

Мета

Дослідження алгоритмів генерації псевдовипадкових послідовностей, тестування простоти чисел та генерації простих чисел з точки зору їх ефективності за часом та можливості використання для гененерації ключів асиметричних криптосистем.

Завдання

Аналіз стійкості реалізацій ПВЧ та генераторів ключів для обраної бібліотеки.

Підгрупа 2В. Бібліотека PyCrypto під Linux платформу.

Хід роботи

Тестування ГВЧ

Для генерування випадкової інформації в бібліотеці PyCryptodome міститься пакет **Crypto.Random**. Цей пакет включає в себе наступні функції:

1) Crypto.Random.get_random_bytes(N) – повертає випадковий байтовий рядок довжиною N.

2) Crypto.Random.random.getrandbits(N) – повертає випадкове ціле число довжиною не більше N біт.

3) Crypto.Random.random.randrange([start,]stop[, step]) – повертає випадкове число в діапазоні (початок, кінець, крок). За замовчуванням початок встановлений в 0, крок в 1.

4) Crypto.Random.random.randint(a, b) – повертає випадкове ціле число не менше за a і не більше від b.

5) Crypto.Random.random.choice(seq) – повертає випадковий елемент, вибраний з послідовності seq.

6) Crypto.Random.random.shuffle(seq) – випадковим чином перемішує послідовність seq на місці.

7) Crypto.Random.random.sample(population, k) – випадковим чином вибирає k різних елементів зі списку.

Безпосередньо для генерування випадкової інформації (чисел або байтів) використовуються перші 4 функції. Більше того, функції 2-4 у своїй роботі використовують функцію 1. А тому тестувати будемо саме послідовності цієї функції.

Якщо звернутися до вихідного кода цієї функції, то побачимо, що в якості джерела випадковості ця функція використовує os.urandom(). os.urandom() – це функція, яка повертає випадкові байти зі специфічного для ОС джерела випадковості. Дані, що повертаються, мають бути достатньо непередбачуваними для криптографічних застосувань, хоча їхня

точна якість залежить від реалізації ОС. У Unix-подібних системах випадкові байти зчитуються з пристрою /dev/urandom.

Тести **Diehard** – це набір статистичних тестів для визначення якості генераторів випадкових чисел (ГВЧ). Тести призначені для вимірювання різноманітних властивостей, таких як однорідність вхідних даних, незалежність між послідовними вибірками, кореляція між вибірками, взятими з різних частин послідовності ГВЧ тощо. Тести можуть бути використані щоб переконатися, що ГВЧ придатний для використання в криптографічних застосунках, а також для інших цілей, таких як наукові та інженерні симуляції.

Результати

Оскільки джерелом випадковості є пристрій операційної системи, то тестувався безпосередньо він. Для цього була виконана наступна команда:

```
cat /dev/urandom | dieharder -a -m 0.2 -g 200 > results.txt
```

Отримані такі результати:

```
#=============================================================================#
#            dieharder version 3.31.1 Copyright 2003 Robert G. Brown          #
#=============================================================================#
   rng_name    |rands/second|   Seed   |
stdin_input_raw|  1.30e+07  |2510848606|
#=============================================================================#
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
#=============================================================================#
   diehard_birthdays|   0|       100|      20|0.94547764|  PASSED
      diehard_operm5|   0|   1000000|      20|0.40855273|  PASSED
  diehard_rank_32x32|   0|     40000|      20|0.98290153|  PASSED
    diehard_rank_6x8|   0|    100000|      20|0.92195997|  PASSED
   diehard_bitstream|   0|   2097152|      20|0.61854748|  PASSED
        diehard_opso|   0|   2097152|      20|0.88088772|  PASSED
        diehard_oqso|   0|   2097152|      20|0.96551663|  PASSED
         diehard_dna|   0|   2097152|      20|0.52904181|  PASSED
diehard_count_1s_str|   0|    256000|      20|0.99999636|   WEAK
diehard_count_1s_byt|   0|    256000|      20|0.69162646|  PASSED
 diehard_parking_lot|   0|     12000|      20|0.08262389|  PASSED
    diehard_2dsphere|   2|      8000|      20|0.86386430|  PASSED
```

```
    diehard_3dsphere|   3|       4000|      20|0.46577415|  PASSED
     diehard_squeeze|   0|     100000|      20|0.42253667|  PASSED
        diehard_sums|   0|        100|      20|0.07390926|  PASSED
        diehard_runs|   0|     100000|      20|0.97295933|  PASSED
        diehard_runs|   0|     100000|      20|0.82496142|  PASSED
       diehard_craps|   0|     200000|      20|0.89749897|  PASSED
       diehard_craps|   0|     200000|      20|0.26187203|  PASSED
marsaglia_tsang_gcd|   0|   10000000|      20|0.98478971|  PASSED
marsaglia_tsang_gcd|   0|   10000000|      20|0.98053270|  PASSED
         sts_monobit|   1|     100000|      20|0.89186591|  PASSED
            sts_runs|   2|     100000|      20|0.55689149|  PASSED
          sts_serial|   1|     100000|      20|0.94551593|  PASSED
          sts_serial|   2|     100000|      20|0.36859925|  PASSED
          sts_serial|   3|     100000|      20|0.96019721|  PASSED
          sts_serial|   3|     100000|      20|0.40599948|  PASSED
          sts_serial|   4|     100000|      20|0.87782791|  PASSED
          sts_serial|   4|     100000|      20|0.93558385|  PASSED
          sts_serial|   5|     100000|      20|0.54763030|  PASSED
          sts_serial|   5|     100000|      20|0.51190174|  PASSED
          sts_serial|   6|     100000|      20|0.99287234|  PASSED
          sts_serial|   6|     100000|      20|0.07682141|  PASSED
          sts_serial|   7|     100000|      20|0.88384036|  PASSED
          sts_serial|   7|     100000|      20|0.69495122|  PASSED
          sts_serial|   8|     100000|      20|0.67412872|  PASSED
          sts_serial|   8|     100000|      20|0.84387304|  PASSED
          sts_serial|   9|     100000|      20|0.97313361|  PASSED
          sts_serial|   9|     100000|      20|0.39131879|  PASSED
          sts_serial|  10|     100000|      20|0.94268437|  PASSED
          sts_serial|  10|     100000|      20|0.55056892|  PASSED
          sts_serial|  11|     100000|      20|0.46828560|  PASSED
          sts_serial|  11|     100000|      20|0.70660718|  PASSED
          sts_serial|  12|     100000|      20|0.15741173|  PASSED
          sts_serial|  12|     100000|      20|0.00016214|   WEAK
          sts_serial|  13|     100000|      20|0.50781954|  PASSED
          sts_serial|  13|     100000|      20|0.70321146|  PASSED
          sts_serial|  14|     100000|      20|0.99286822|  PASSED
          sts_serial|  14|     100000|      20|0.58557402|  PASSED
          sts_serial|  15|     100000|      20|0.95120646|  PASSED
          sts_serial|  15|     100000|      20|0.51441905|  PASSED
```

```
          sts_serial|  16|    100000|      20|0.05319528|  PASSED
          sts_serial|  16|    100000|      20|0.00327531|   WEAK
         rgb_bitdist|   1|    100000|      20|0.86567633|  PASSED
         rgb_bitdist|   2|    100000|      20|0.02908610|  PASSED
         rgb_bitdist|   3|    100000|      20|0.46150391|  PASSED
         rgb_bitdist|   4|    100000|      20|0.97492105|  PASSED
         rgb_bitdist|   5|    100000|      20|0.95678438|  PASSED
         rgb_bitdist|   6|    100000|      20|0.55544081|  PASSED
         rgb_bitdist|   7|    100000|      20|0.70109599|  PASSED
         rgb_bitdist|   8|    100000|      20|0.49256449|  PASSED
         rgb_bitdist|   9|    100000|      20|0.83424421|  PASSED
         rgb_bitdist|  10|    100000|      20|0.94674244|  PASSED
         rgb_bitdist|  11|    100000|      20|0.05196150|  PASSED
         rgb_bitdist|  12|    100000|      20|0.89038708|  PASSED
rgb_minimum_distance|   2|     10000|     200|0.44837526|  PASSED
rgb_minimum_distance|   3|     10000|     200|0.16289735|  PASSED
rgb_minimum_distance|   4|     10000|     200|0.78603470|  PASSED
rgb_minimum_distance|   5|     10000|     200|0.33947956|  PASSED
    rgb_permutations|   2|    100000|      20|0.07607428|  PASSED
    rgb_permutations|   3|    100000|      20|0.90133530|  PASSED
    rgb_permutations|   4|    100000|      20|0.30203286|  PASSED
    rgb_permutations|   5|    100000|      20|0.78059570|  PASSED
      rgb_lagged_sum|   0|   1000000|      20|0.77902377|  PASSED
      rgb_lagged_sum|   1|   1000000|      20|0.46804827|  PASSED
      rgb_lagged_sum|   2|   1000000|      20|0.99998552|   WEAK
      rgb_lagged_sum|   3|   1000000|      20|0.77325669|  PASSED
      rgb_lagged_sum|   4|   1000000|      20|0.67846493|  PASSED
      rgb_lagged_sum|   5|   1000000|      20|0.96875515|  PASSED
      rgb_lagged_sum|   6|   1000000|      20|0.60308927|  PASSED
      rgb_lagged_sum|   7|   1000000|      20|0.95388700|  PASSED
      rgb_lagged_sum|   8|   1000000|      20|0.89243534|  PASSED
      rgb_lagged_sum|   9|   1000000|      20|0.89309572|  PASSED
      rgb_lagged_sum|  10|   1000000|      20|0.54062887|  PASSED
      rgb_lagged_sum|  11|   1000000|      20|0.27859612|  PASSED
      rgb_lagged_sum|  12|   1000000|      20|0.88577381|  PASSED
      rgb_lagged_sum|  13|   1000000|      20|0.11980101|  PASSED
      rgb_lagged_sum|  14|   1000000|      20|0.96880760|  PASSED
      rgb_lagged_sum|  15|   1000000|      20|0.54453363|  PASSED
      rgb_lagged_sum|  16|   1000000|      20|0.55441452|  PASSED
```

```
      rgb_lagged_sum|  17|   1000000|      20|0.96830270|  PASSED
      rgb_lagged_sum|  18|   1000000|      20|0.10512755|  PASSED
      rgb_lagged_sum|  19|   1000000|      20|0.80333327|  PASSED
      rgb_lagged_sum|  20|   1000000|      20|0.82820399|  PASSED
      rgb_lagged_sum|  21|   1000000|      20|0.36201582|  PASSED
      rgb_lagged_sum|  22|   1000000|      20|0.99963426|   WEAK
      rgb_lagged_sum|  23|   1000000|      20|0.24923721|  PASSED
      rgb_lagged_sum|  24|   1000000|      20|0.72923708|  PASSED
      rgb_lagged_sum|  25|   1000000|      20|0.24156449|  PASSED
      rgb_lagged_sum|  26|   1000000|      20|0.98736563|  PASSED
      rgb_lagged_sum|  27|   1000000|      20|0.50735212|  PASSED
      rgb_lagged_sum|  28|   1000000|      20|0.99968514|   WEAK
      rgb_lagged_sum|  29|   1000000|      20|0.38172117|  PASSED
      rgb_lagged_sum|  30|   1000000|      20|0.98852094|  PASSED
      rgb_lagged_sum|  31|   1000000|      20|0.09745837|  PASSED
      rgb_lagged_sum|  32|   1000000|      20|0.93114116|  PASSED
     rgb_kstest_test|   0|     10000|     200|0.52223697|  PASSED
     dab_bytedistrib|   0|  51200000|       1|0.85126207|  PASSED
             dab_dct| 256|     50000|       1|0.19322700|  PASSED
Preparing to run test 207.  ntuple = 0
        dab_filltree|  32|  15000000|       1|0.47609325|  PASSED
        dab_filltree|  32|  15000000|       1|0.25949518|  PASSED
Preparing to run test 208.  ntuple = 0
       dab_filltree2|   0|   5000000|       1|0.32246541|  PASSED
       dab_filltree2|   1|   5000000|       1|0.83143484|  PASSED
Preparing to run test 209.  ntuple = 0
        dab_monobit2|  12|  65000000|       1|0.57328923|  PASSED
```

Генерація ключів

Бібліотека PyCryptodome наступні види ключів криптосистем:

– RSA

– DSA

– ECC

– ElGamal (застарілий)

Модулі Crypto.PublicKey.RSA та Crypto.PublicKey.DSA аналогічні в

застосування і надають засоби для генерації нових ключів, їхньої реконструкції з відомих компонентів, експорту та імпорту.

Алгоритм генерування ключа RSA повністю відповідає NIST FIPS 186-4 в його розділах B.3.1 та B.3.3. Модуль є добутком двох несильних ймовірних простих чисел. Кожне просте число проходить відповідну кількість тестів Міллера-Рабіна з випадковими основами і один тест Лукаса.

Алгоритм генерування ключа DSA відповідає Додатку A.1/A.2 та B.1 FIPS 186-4 відповідно для генерації домену та генерації пари ключів.

Приклад генерування ключа RSA 1024 біта:

```
RSA_key = RSA.generate(1024)
print(RSA_key.export_key())
print(RSA_key.public_key().export_key())
```

Результат:

```
b'-----BEGIN RSA PRIVATE KEY-----\nMIICXQIBAAKBgQC+KJuXii/8hRooYKTzzcBoU+
    CxSqGMnOe3pxPYtL4/jSfCeowA\
    nlqqm1UTwU9eHp5S26m6dqOhZmUWOUacPeoO19SXFUjcX4g00sqPHD9kVAfIiFYt9\
    njGMOpQMPPwx3vU8T78ZjGzZLj2a4XbYY59pMfkVVcPO9ZnaSsarQLV+ohwIDAQAB\
    nAoGACGt4NuBdrz2I2CMuBUTl0UNu2QV4HBJHKPjDEQmGZ6j34DgYjTQepUQfM9Gl\nOEf2y+
    ybGOxa3BDEetXpuEPmo1WOfUq862fYU2UuLhFhhblxZ4BDV8Fzj6foloxd\nFzzppE8zeoPp+
    XbN5ri0BZAdq/BSzMkJu6DcyFH/jIsCvwECQQDCLddZxxkr7Vme\
    nc9lHYo5Kb3rEop7FPuqoHf52gNiBfhc1YwjfSljrDJoOW1MRydFM+qvMLg6gSHOX\nrZm9QNWHAkEA+
    rMWuQheKCwvdfj5qHIiph6hswYv4pB65uvRdddsaygz3WLVXpBM\nQ4NQnZlbLj/DKnaDXtOmooDEgQ5w
    +z9VAQJAEKC0TC/MbAKLJhJtjLtXy+/351a1\
    nmdZlabgHzkYCRDRz8FJmtBg9vQL7Pjd7yY4qqhjdFULsZbVOkLKA87/ViQJBAIBC\
    nSXjGwC07Vdi8VTYJlpJ4j2+g1bTLCCXGNfZdDTnRap9z5geoOWw87WyDvfNuabH6\
    nygEktVbkHCSOt9zCrQECQQCkF7QsZ9UrSJ9rN8xe17xvTVqYk+T8OFDj7L+rveCh\ndjYv7oUz/
    xA7HthxIKds3Ml5lxjIO+nac0nW3LxSDgMM\n-----END RSA PRIVATE KEY-----'
```

Модуль Crypto.PublicKey.ECC надає аналогічний функціонал, що й попередні, але довжина ключа тут встановлюється не безпосередньо, а вибором еліптичної кривої. У порівнянні з традиційними алгоритмами, такими як RSA, ключ ECC значно менший за розміром при тому ж рівні безпеки.

В залежності від кривої, алгоритм керується стандартами FIPS 186-4, розділ D.1.2 або RFC8032.

Приклад генерування ключа ECC для кривої NIST P-256:

```
ECC_key = ECC.generate(curve='P-256')
print(ECC_key.export_key(format='PEM'))
print(ECC_key.public_key().export_key(format='PEM'))
```

Отримали:

```
-----BEGIN PRIVATE KEY-----
MIGHAgEAMBMGByqGSM49AgEGCCqGSM49AwEHBG0wawIBAQQgjkYdcYY+VsRUfJmU
UZcCXswBkzE8SnQ7giYu1hNHRdehRANCAATUK0LUrQ7GGvyiDZi+F51uDcdiKztC
hec+F4Nu8TWEtMYW8u93vEAtQvHQgbcrIo5o3LtU0GBfL+w+zjVO6Lkw
-----END PRIVATE KEY-----
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE1CtC1K00xhr8og2YvhedbG3HYis7
QoXnPheDbvE1hLTGFvLvd7xALULx0IG3KyKOaNy7VNBgXy/sPs41Tui5MA==
-----END PUBLIC KEY-----
```

Модуль Crypto.PublicKey.ElGamal вважається застарілим. Тут присутній функціонал генерування та конструювання нових ключів, але відсутня можливість їхнього імпорту та експорту. Дана криптосистема для високих рівнів безпеки потребує довгих ключів, а час їх генерування в порівнянні з іншими довший.

Приклад генерування ключа довжини 512 біт криптосистеми ElGamal:

```
ElGamal_key = ElGamal.generate(512, Crypto.Random.get_random_bytes)
print(f"Public key:\n{ElGamal_key.g = }\n{ElGamal_key.p = }\n{ElGamal_key.y = }\n")
print(f"Private key:\n{ElGamal_key.x = }\n")
```

Результат виконання зображено на рис. 1.

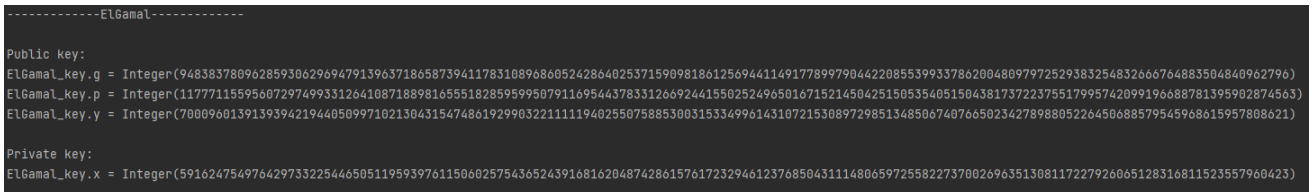

Рисунок 1 – Згенеровані ключі для криптосистеми Ель-Гамаля, довжина 512 біт

ВИСНОВКИ

Генератор випадкових чисел, що використовується в бібліотеці PyCryptodome є придатним для криптографічного застосування. Він проходить всі тести з набору diehard, видаючи лише на одиницях із них результат «WEAK».

Бібліотека PyCryptodome містить широкий функціонал для роботи із відкритими та закритими ключами: генерування, конструювання, імпорт та експорт. Алгоритми генерування ключів відповідають визначеним стандартам, в залежності від криптосистеми підтримуються наступні формати ключів: PEM, DER, OpenSSH, SEC1, PKCS#1, PKCS#8, X.509.