



**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»  
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ  
Кафедра Інформаційної Безпеки**

**Лабораторна робота №2  
з дисципліни  
«МЕТОДИ РЕАЛІЗАЦІЇ КРИПТОГРАФІЧНИХ МЕХАНІЗМІВ»**

**Виконали:**  
**Студени 6 курсу ФТІ**  
**групи ФБ-41мн**  
**Бондаренко О.Ю., Кригін Д.О.**

**Перевірила:**  
**асистент**  
**Байденко П.В.**

## Лабораторна робота №2

### Реалізація алгоритмів генерації ключів гібридних криптосистем

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

**Завдання:** Другий тип лабораторної роботи. Аналіз стійкості реалізацій ПВЧ та генераторів ключів для обраної бібліотеки. Підгрупа 2В. Бібліотека PyCrypto під Linux платформу.

### Хід Роботи

#### 1. Опис функції генерації ПСП (Псевдовипадкових чисел)

У бібліотеці **PyCryptodome** за генерацію криптографічно стійких псевдовипадкових даних відповідає модуль **Crypto.Random**.

Основна функція: **Crypto.Random.get\_random\_bytes(n)**  
На платформі Linux **PyCryptodome** (як і його попередник PyCrypto) покладається на системний генератор випадкових чисел, *getrandom()*.

**Джерело ентропії:** Модуль **Crypto.Random** читає дані безпосередньо з системного пристрою **/dev/urandom**. Це спеціальний файл пристрою в Linux, який надає доступ до криптографічно стійкого генератора псевдовипадкових чисел (CSPRNG) ядра операційної системи.

**Блокування:** Використання **/dev/urandom** є неблокуючим. Він надасть запитувану кількість байтів, використовуючи зібрану ентропію, і не буде блокувати процес, навіть якщо ентропійний пул ядра тимчасово вичерпаний (на відміну від **/dev/random**). Це вважається безпечним для переважної більшості криптографічних застосувань.

**Внутрішній стан:** **PyCryptodome** не підтримує власний PRNG в просторі користувача, який потребує ручного "засіювання" (seeding). Замість цього він завжди звертається безпосередньо до ОС, що вважається найкращою практикою.

##### Вхідні дані:

**n (int):** Кількість випадкових байтів, яку необхідно згенерувати.

##### Вихідні дані:

**(bytes):** Об'єкт типу **bytes** довжиною **n**, заповнений криптографічно стійкими випадковими даними.

##### Коди повернення (Обробка помилок)

У Python для сповіщення про помилки частіше використовуються винятки (exceptions), а не коди повернення.

**Успішне виконання:** Повертає об'єкт bytes.

**Помилка:** Якщо з якоїсь причини операційна система не може надати випадкові дані (наприклад, збій при читанні з /dev/urandom), функція згенерує виняток, зазвичай OSError або RuntimeError.

## 2. Опис функцій генерації ключів

Генерація ключів у **PyCryptodome** залежить від типу алгоритму (симетричний чи асиметричний).

### 2.1. Генерація симетричних ключів (наприклад, AES)

Для симетричних алгоритмів (як-от AES, ChaCha20) "генерація ключа" – це, по суті, генерація випадкової послідовності байтів потрібної довжини.

**Опис функції:** Використовується функція `Crypto.Random.get_random_bytes(n)`, де `n` - необхідна довжина ключа, наприклад 32.

**Опис алгоритму:** Алгоритм ідентичний генерації ПСП.

**Вхідні дані:** `n` (int) – довжина ключа в байтах (наприклад, 16, 24 або 32 для AES-128, AES-192, AES-256 відповідно).

**Вихідні дані:** (bytes) – ключ відповідної довжини.

#### Коди повернення (Обробка помилок)

У Python для сповіщення про помилки частіше використовуються винятки (exceptions), а не коди повернення.

**Успішне виконання:** Повертає об'єкт bytes.

**Помилка:** Якщо з якоїсь причини операційна система не може надати випадкові дані (наприклад, збій при читанні з /dev/urandom), функція згенерує виняток, зазвичай OSError або RuntimeError.

### 2.2. Генерація асиметричних ключів (наприклад, RSA)

Для асиметричних алгоритмів існують спеціалізовані модулі та функції.

**Основний модуль:** `Crypto.PublicKey.RSA` Основна функція:  
`RSA.generate(bits, randfunc=None, e=65537)`

#### Опис алгоритму

Функція реалізує стандартний алгоритм генерації пари ключів RSA:

**Вибір експоненти (e):** Використовується публічна експонента `e`. За замовчуванням це 65537 (або  $2^{16}+1$ ), оскільки це поширене просте число, що забезпечує ефективність при перевірці підпису та шифруванні.

**Генерація простих чисел (p, q):** Використовуючи надану `randfunc` (яка за замовчуванням є `Crypto.Random.get_random_bytes`), генеруються два великих випадкових простих числа `p` та `q` такого розміру, щоб їхній добуток  $n = p * q$  мав довжину `bits`. Процес включає генерацію випадкового числа та його перевірку на простоту (наприклад, за допомогою тесту Міллера-Рабіна).

**Обчислення n та  $\phi(n)$ :** Обчислюється модуль  $n = p * q$  та функція Ейлера  $\phi(n) = (p-1)(q-1)$ .

**Обчислення d:** Обчислюється секретна експонента `d` як  $d \equiv e^{-1} \pmod{\phi(n)}$  (мультіплікативне обернене `e` за модулем  $\phi(n)$ ).

**Формування ключа:** Створюється об'єкт `RsaKey`, що містить публічну частину ( $n$ ,  $e$ ) та приватну частину ( $d$ ,  $p$ ,  $q$  та інші оптимізаційні компоненти).

**Вхідні дані:**

`bits` (int): Довжина модуля  $n$  у бітах (наприклад, 2048, 3072, 4096). Має бути не менше 1024 і кратною 256.

`randfunc` (callable, опціонально): Функція, що повертає випадкові байти. Якщо `None`, використовується `Crypto.Random.get_random_bytes`.

`e` (int, опціонально): Значення публічної експоненти. За замовчуванням 65537.

**Вихідні дані:**

(`Crypto.PublicKey.RSA.RsaKey`): Об'єкт, що представляє згенеровану пару ключів RSA (публічний та приватний).

**Коди повернення (Обробка помилок):**

**Успішне виконання:** Повертає об'єкт `RsaKey`.

**Помилка:**

`ValueError`: Генерується, якщо `bits` має некоректне значення (наприклад, менше 1024 або не кратне 256).

`OSError` / `RuntimeError`: Може бути згенеровано внутрішньою `randfunc`, якщо ОС не може надати випадкові дані.

## Приклад роботи з функціями

### 1. Скрипт для шифрування/розшифрування

Даний скрипт (`aes-example.py`) приймає першим аргументом шлях до файлу та режим роботи. Режимів роботи 2: шифрування (`enc`), розшифрування (`dec`).

В режимі шифрування (`enc`), скрипт читає файл, генерує випадковий 32-байтний ключ та шифрує файл за допомогою AES-256-GCM та зберігає у файл з розширенням `.enc`

В режимі розшифрування (`dec`), скрипт читає файл, читає ключ шифрування від користувача, та намагається розшифрувати файл та зберегти його з розширенням `.dec`.

```

*[main][kali: ~/mrkm-2025/lab2]$ ./aes-example.py ./passwd enc
AES-256-GCM KEY: 782b89b32145c7d5f20a296fc2715049509fbc43a0413ac6b8b3df99daba788d
File encrypted successfully: ./passwd.enc
*[main][kali: ~/mrkm-2025/lab2]$ xxd ./passwd.enc | head
00000000: 7bfd c6c8 1992 b260 6a67 345e 0c36 a8b9  {.....`jg4^.6..
00000010: f342 3485 1c7e 90a9 c8f0 5327 6e20 bfea  .B4...~....S'n ..
00000020: 17ad 59c8 c866 c965 0094 3c02 dec1 aa26  ..Y...f.e...<...&
00000030: 41d3 fd11 5325 efc6 1e02 e6d0 b020 a149  A...S%..... .I
00000040: 5b92 ed00 8639 0a4a 95a2 b469 8783 172f  [....9.J...i.../
00000050: 4622 6898 cbe7 d52f 1103 194e bd22 d5f5  F"h..../...N"..
00000060: 4ce5 d7c3 1ff2 dfc9 0e5b 4652 e228 9c43  L.....[FR.(C
00000070: a933 28ff 10ff 8f6c e686 5f51 2f22 c5ce  .3(...L...Q/"..
00000080: d330 1b34 7ce5 5800 51f9 b845 61d2 7635  .0.4|.X.Q..Ea.v5
00000090: 8b07 d028 15b7 9351 9fd9 6471 8d38 6829  ...(...Q...dq.8h)
*[main][kali: ~/mrkm-2025/lab2]$ ./aes-example.py ./passwd dec
Enter 64-char hex key: 782b89b32145c7d5f20a296fc2715049509fbc43a0413ac6b8b3df99daba788d
File decrypted successfully: ./passwd.dec
*[main][kali: ~/mrkm-2025/lab2]$ xxd ./passwd.dec | head
00000000: 726f 6f74 3a78 3a30 3a30 3a72 6f6f 743a  root:x:0:0:root:
00000010: 2f72 6f6f 743a 2f75 7372 2f62 696e 2f7a  /root:/usr/bin/z
00000020: 7368 0a64 6165 6d6f 6e3a 783a 313a 313a  sh.daemon:x:1:1:
00000030: 6461 656d 6f6e 3a2f 7573 722f 7362 696e  daemon:/usr/sbin
00000040: 3a2f 7573 722f 7362 696e 2f6e 6f6c 6f67  :/usr/sbin/nolog
00000050: 696e 0a62 696e 3a78 3a32 3a32 3a62 696e  in.bin:x:2:2:bin
00000060: 3a2f 6269 6e3a 2f75 7372 2f73 6269 6e2f  :/bin:/usr/sbin/
00000070: 6e6f 6c6f 6769 6e0a 7379 733a 783a 333a  nologin.sys:x:3:
00000080: 333a 7379 733a 2f64 6576 3a2f 7573 722f  3:sys:/dev:/usr/
00000090: 7362 696e 2f6e 6f6c 6f67 696e 0a73 796e  sbin/nologin.syn
*[main][kali: ~/mrkm-2025/lab2]$ █

```

## 2. Скрипти для шифрування/розшифрування сесійного ключа за допомогою RSA

Створено три скрипти: `rsa-keygen.py`, `rsa-enc-session-key.py`, `rsa-dec-session-key.py`.

`rsa-keygen.py` - генерує пару (RSA.generate(2048)) з публічного та приватного ключа та зберігає їх у відповідні файли: `public.pem`, `private.pem`

`rsa-enc-session-key.py` - генерує випадковий сесійний ключ

(`get_random_bytes(32)`) та шифрує його за допомогою публічного ключа RSA та зберігає у файл `encrypted_session.bin`.

`rsa-dec-session-key.py` - читає зашифрований сесійний ключ з файлу `encrypted_session.bin` та розшифровує його приватним ключем RSA.

```

*[main][kali: ~/mrkm-2025/lab2]$ ./rsa-keygen.py
Generating RSA 2048-bit key pair...
Saved private.pem and public.pem
*[main][kali: ~/mrkm-2025/lab2]$ cat private.pem
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEA mGaEyAggc5Qh3HZoFjfpdaBX0epi1BBvaQq6a0fb/9WXwJ5C
v0ynVhnp+nnNMfjPEGLXq/nrw1xomFTBx2zudx095WLAUWPiI+i0HK62SL6q6Sm3
MVSwer6iasykXxkoF2d3tGgTANnhh2AqH/gq+Z8FG5yQ9I6uAmYcInnu/yJthbSB
CV0DmpmCqyis2g3Y397K8fNgugx2gQPAu5ymr923KYYoDM0qj70eH8wuFjceef4D
e/+uxt0eE0QWVexUBbEqCJhL83GrJhKdLBmSZEQxLf4ZwV4jY7QhAj2yQe4o1r5T
tLyLjppjX5vfuuJyFBEjwV5KVJF460Lo6VrpC7QIDAQABAOIBACGzdgr4C5kTdmN2
ybC0H4vT7+erMbycf86WQUAtDWDnPL4pgS+fE6g0JfguTC03UV7bNLiClVjHzBd3
kryDgb5UeJYLvuSgsKcEK7z7A7NVep0CDRE7HaWfF3KPm3+uI/f8+PMrmpyNoFEb
+mTEb9hwimQfPMzf6mn8Y5daKHHBsLMbqmGRrHvrayADG+Fc0d59cgNajjBEIQIh
s3ZVjhDtV3Lr2MMaFCMMspQ65cHn3DoVcuMfjuWBc8HoqItAbi0h3D07Ypy2zyGn
0gcFmDi5qlH1lZYT5Z80vrwmC8IYRhOmHpPW2y3Zzzy5WijL6pDFN3Cnk1k2fvij
FawuDHECgYEAu00mYv+E/9S1ipYYGwidFi7Fw/h/DbdKchlTtEU4T8wGP4sXEzDyt
SZVorJy9fQxhQWeV53KJQwghzXBrm0Cz3itbLDNEJhpXeUbnB4sbjiVVhsVPPzaM
V0092RgLaGATG46hXhUnQ5frjL4xssvL/z8RjyC7Z42vH1muG5uIMzUCgYEA0vkn
oR01VrkTpzCXJ0R0a/jjh+RhMWzcnWP/wn7xHgv9AkAaf5NGF2G0Tiq8e2m934K0
VN7f+T5t/l3Z5nmClwQ5fhK8ia9Ap5QZCQqY+PGgXiiiQYo1S/d47DroLF7KC9Ii
q3yruSlh68D9S7rAubB//DQBVXp6Es3b5hhDz9kCgYEArmGzGReq6zRCHPV8tDka
KJjilPH3nY+B/CZN20utNDH85PNpqM5u+jySAH0tnAXYkDyF90LifPtmJLwmPLi+
5HouWYoecW/uIVbi9RIQYfKcVbVdpvBhVtHXyKgL22FG02GSMY3xAKR51D6uzt0v
Ans0q9FJ51SMFxs0qZnwSECgYEAwWZik8LTXTLAj219wyN/zLMU0chMSPDQtP+s
v/F6u96li2CUagrQIBkTHu0SZ9ghLTn0clb13xiZ4KrMS/lr+2CGC8rFRzp+/KBI
6v7qCJwINQju5UH7Za5xDa1a9FiIw5LY6veYvF7xQxthmACVGCRyzxILfuSN75sD
UMVBtNECgYBdRx04iWJ+wMRXkJ00wqKDij094uhTVwrg6DsX0ZMb4qUiocTrVeWu
cZ+UfbP1xa+GhAQ6Pmd0SFqzJc+W0CE49+Dh82wMt5TVRm9Aa1JoPDdtf/vaDsik
mqE8l8quMwnEZPSuKLYrf27VcoQbd/+h+T5jbJzZJK9uP2fDI3dFjQ==
-----END RSA PRIVATE KEY-----%
*[main][kali: ~/mrkm-2025/lab2]$ cat public.pem
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAA0CAQ8AMIIBCgKCAQEAmGaEyAggc5Qh3HZoFjfp
daBX0epi1BBvaQq6a0fb/9WXwJ5Cv0ynVhnp+nnNMfjPEGLXq/nrw1xomFTBx2zu
dx095WLAUWPiI+i0HK62SL6q6Sm3MVSwer6iasykXxkoF2d3tGgTANnhh2AqH/gq
+Z8FG5yQ9I6uAmYcInnu/yJthbSB CV0DmpmCqyis2g3Y397K8fNgugx2gQPAu5ym
r923KYYoDM0qj70eH8wuFjceef4De/+uxt0eE0QWVexUBbEqCJhL83GrJhKdLBmS
ZEQxLf4ZwV4jY7QhAj2yQe4o1r5TtLyLjppjX5vfuuJyFBEjwV5KVJF460Lo6VrpC
7QIDAQAB
-----END PUBLIC KEY-----%
*[main][kali: ~/mrkm-2025/lab2]$ █

```

```

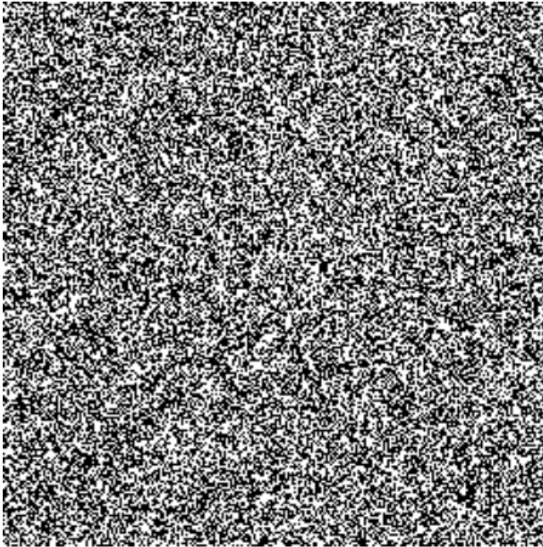
*[main][kali: ~/mrkm-2025/lab2]$ ./rsa-enc-session-key.py
New Session Key: 630b631b129420575df7999a4bc77e4cb0d76fcfe728c89b3a7502541061617b
Successfully encrypted session key and saved to encrypted_session.bin
*[main][kali: ~/mrkm-2025/lab2]$ xxd encrypted_session.bin
00000000: 2aa2 65c8 fb64 90d9 a2a5 59e2 ee31 b1fb *e..d....Y..1..
00000010: f27c f44b a3fe 50b2 6810 1575 0f1b 5d45 .|.K..P.h..u..]E
00000020: 65fe 1155 2db7 0c12 2f08 eb57 df00 07e7 e..U-.../.W....
00000030: f0f3 4030 4419 e55c ce26 b953 4a7c ac3e ..@0D..\.&.SJ|. >
00000040: 5730 be45 d8cf dd05 1758 3977 af8e 44df W0.E.....X9w..D.
00000050: 370f 9121 fffe 58c5 2424 8838 5d91 4a26 7...!.X.$$.8].J&
00000060: 38bb 4ba3 d3c9 4603 795e 8d0a b8e9 6061 8.K...F.y^....`a
00000070: 77c0 8eee cf96 c129 a496 db5d a105 8882 w.....)....]....
00000080: 4d93 22d4 ea1e 550d 03b7 1042 5093 4ccf M."...U....BP.L.
00000090: 8518 4a49 4560 d3c3 2954 8bd3 0b47 1a11 ..JIE`..)T...G..
000000a0: aa1d 5ad4 eb4f 9b8b 9807 abc7 1ba2 81ba ..Z..0.....
000000b0: ef7d 13c3 185b 4e8a ea7a 9063 ef1d 76bd .}....[N..z.c..v.
000000c0: 7000 6023 6125 c6ff 4689 106b b44d 288f p.`#a%..F..k.M(.
000000d0: 1130 9750 c654 a412 d826 83f7 e4b6 f759 .0.P.T...&.....Y
000000e0: 8e3e 3a3c 6c7d bb68 c171 f963 ceef 3013 .>:<l}.h.q.c..0.
000000f0: 5470 204d 73cf 8715 37dd 2f4c 9c96 5c97 Tp Ms...7./L..\..
*[main][kali: ~/mrkm-2025/lab2]$ ./rsa-dec-session-key.py
Recovered Session Key: 630b631b129420575df7999a4bc77e4cb0d76fcfe728c89b3a7502541061617b
*[main][kali: ~/mrkm-2025/lab2]$ █

```

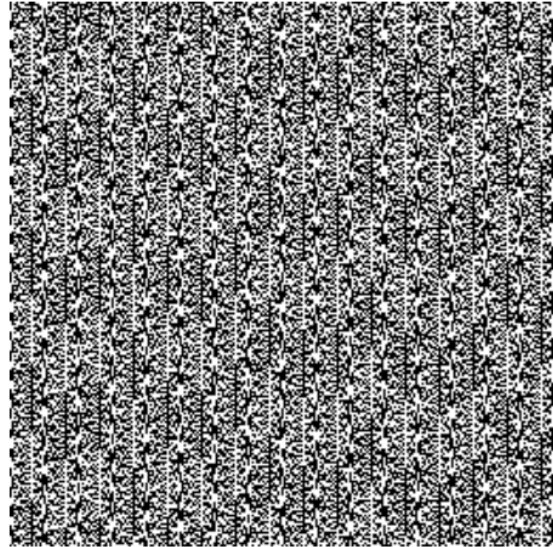
### 3. Аналіз якості генератора псевдовипадкових чисел

Джерело [RANDOM.ORG - Statistical Analysis](https://random.org/statistical-analysis)

Доцільно провести представлену у джерелі оцінку якості криптографічно захищених генераторів псевдовипадкових чисел (CSPRNG). Основна техніка полягає в перетворенні потоку отриманих випадкових байтів в растрове зображення у відтинках сірого. Кількість отриманих байтів точно дорівнює кількості пікселів зображення (512x512); потім ці байти перетворюються NumPy у двовимірну матрицю, де значення кожного байта (0-255) визначає інтенсивність сірого кольору пікселя. Matplotlib візуалізує цю матрицю. Високоякісний CSPRNG повинен створювати зображення з рівномірним білим шумом, що підтверджує основні криптографічні властивості статистичної незалежності та рівномірного розподілу по всіх можливих вихідних значеннях, тоді як будь-які видимі закономірності або відхилення означатимуть несправність генератора випадкових чисел.



**RANDOM.ORG**

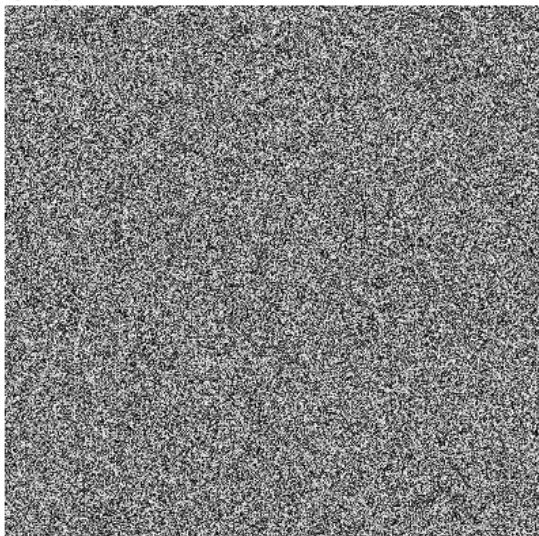


**PHP rand() on Microsoft Windows**

Створимо зображення за цією ж методологією для генератора випадкових чисел з `pycryptodome`

```
$ python3 plot_random.py
```

PyCryptodome CSPRNG Randomness (512x512)



Як бачимо, отримане зображення демонструє наступні характеристики генератора:

- Рівномірний розподіл: випадкові дані містять рівну ймовірність генерації будь-якого значення байта від 0 (чорний) до 255 (білий). Це призводить до збалансованого поєднання всіх відтінків сірого, створюючи текстуру білого шуму.
- Статистична незалежність: між одним пікселем і його сусідами немає помітних візерунків, ліній або кореляцій. Значення одного байта не

залежить від попереднього байта, що має вирішальне значення для криптографічної безпеки.

- Висока якість: CSPRNG (Crypto.Random PyCryptodome) працює правильно і видає дані, які статистично не відрізняються від справжньої випадковості.

Отже, зображення успішно демонструє придатність і надійність джерела випадкових даних для задач безпеки.

## **Висновки**

В ході лабораторної роботи було проаналізовано механізми генерації ПБЧ та ключів у бібліотеці PyCryptodome на платформі Linux. Встановлено, що бібліотека покладається на криптографічно стійкий генератор псевдовипадкових чисел (CSPRNG) операційної системи (наприклад, `getrandom()` або `/dev/urandom`) як єдине джерело ентропії.

Це джерело, доступне через `Crypto.Random.get_random_bytes()`, використовується як для прямої генерації симетричних ключів (наприклад, AES), так і в якості основи (`randfunc`) для генерації асиметричних пар ключів (наприклад, RSA). Статистичний аналіз ПБЧ шляхом візуалізації підтвердив його високу якість (рівномірний "білий шум").

Практична реалізація гібридної системи, що включала шифрування сесійного ключа за допомогою RSA-OAEP та шифрування файлів за допомогою AES-GCM, пройшла успішно, підтвердивши надійність та коректність використання досліджених функцій.