

Міністерство освіти і науки України  
Національний технічний університет України  
"Київський політехнічний інститут імені Ігоря Сікорського"  
Фізико-технічний інститут

## **КРИПТОГРАФІЯ КОМП'ЮТЕРНИЙ ПРАКТИКУМ №4**

**Вивчення криптосистеми RSA та алгоритму електронного  
підпису; ознайомлення з методами генерації параметрів для  
асиметричних криптосистем**

Виконав:  
Андрєєв Д.Ю.  
Група: ФБ-06

Київ - 2022

# Мета роботи

Ознайомлення з тестами перевірки чисел на простоту і методами генерації ключів для асиметричної криптосистеми типу RSA; практичне ознайомлення з системою захисту інформації на основі криптосхеми RSA, організація з використанням цієї системи засекреченого зв'язку й електронного підпису, вивчення протоколу розсилання ключів.

## Порядок виконання роботи

1. Написати функцію пошуку випадкового простого числа з заданого інтервалу або заданої довжини, використовуючи датчик випадкових чисел та тести перевірки на простоту. В якості датчика випадкових чисел використовуйте вбудований генератор псевдовипадкових чисел вашої мови програмування. В якості тесту перевірки на простоту рекомендовано використовувати тест Міллера-Рабіна із попередніми пробними діленнями. Тести необхідно реалізовувати власноруч, використання готових реалізацій тестів не дозволяється.
2. За допомогою цієї функції згенерувати дві пари простих чисел  $p, q$  і  $p_1, q_1$  довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб  $pq < p_1q_1$ ;  $p$  і  $q$  – прості числа для побудови ключів абонента А,  $p_1$  і  $q_1$  – абонента В.
3. Написати функцію генерації ключових пар для RSA. Після генерування функція повинна повертати та/або зберігати секретний ключ  $(d, p, q)$  та відкритий ключ  $(n, e)$ . За допомогою цієї функції побудувати схеми RSA для абонентів А і В – тобто, створити та зберегти для подальшого використання відкриті ключі  $(e, n)$ ,  $(e_1, n_1)$  та секретні  $d$  і  $d_1$ .
4. Написати програму шифрування, розшифрування і створення повідомлення з цифровим підписом для абонентів А і В. Кожна з операцій (шифрування, розшифрування, створення цифрового підпису, перевірка цифрового підпису) повинна бути реалізована окремою процедурою, на вхід до якої повинні подаватись лише ті ключові дані, які необхідні для її виконання. За допомогою датчика випадкових чисел вибрати відкрите повідомлення  $M$  і знайти криптограму для абонентів А і В, перевірити правильність розшифрування. Скласти для А і В повідомлення з цифровим підписом і перевірити його.
5. За допомогою раніше написаних на попередніх етапах програм організувати роботу протоколу конфіденційного розсилання ключів з підтвердженням справжності по відкритому каналу за допомогою алгоритму RSA. Протоколи роботи кожного учасника (відправника та приймаючого) повинні бути реалізовані у вигляді окремих процедур, на вхід до яких повинні подаватись лише ті ключові дані, які необхідні для виконання. Перевірити роботу програм для випадково обраного ключа  $0 < k < n$ .  
Кожна з наведених операцій повинна бути реалізована у вигляді окремої процедури, інтерфейс якої повинен приймати лише ті дані, які необхідні для її роботи; наприклад, функція `Encrypt()`, яка шифрує повідомлення для абонента, повинна приймати на вхід повідомлення та відкритий ключ адресата (і тільки

його), повертаючи в якості результату шифротекст. Відповідно, програмний код повинен містити сім високорівневих процедур: `GenerateKeyPair()`, `Encrypt()`, `Decrypt()`, `Sign()`, `Verify()`, `SendKey()`, `ReceiveKey()`.

## Хід роботи

1. Написати функцію пошуку випадкового простого числа з заданого інтервалу або заданої довжини, використовуючи датчик випадкових чисел та тести перевірки на простоту. В якості датчика випадкових чисел використовуйте вбудований генератор псевдовипадкових чисел вашої мови програмування. В якості тесту перевірки на простоту рекомендовано використовувати тест Міллера-Рабіна із попередніми пробними діленнями. Тести необхідно реалізовувати власноруч, використання готових реалізацій тестів не дозволяється.

Для виконання цього завдання було реалізовано функцію `miller_rabin_test`, яка приймає ціле число та перевіряє його на простоту. Для реалізації було використано методичні вказівки щодо цього алгоритму. Далі на основі цієї функції було реалізовано пошук випадкового простого числа `get_rand_prime`. Для генерації псевдовипадкових чисел було використано модуль `random` мови програмування Python.

2. За допомогою цієї функції згенерувати дві пари простих чисел  $p$ ,  $q$  і  $p_1$ ,  $q_1$  довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб  $pq < p_1q_1$ ;  $p$  і  $q$  – прості числа для побудови ключів абонента А,  $p_1$  і  $q_1$  – абонента В.

Для цього було реалізовано функцію `get_keys`, яка приймає на вхід інтервал, а на вихід повертає 4 простих числа, перевічених за допомогою теста Міллера-Рабіна. Також перевірено, що  $pq < p_1q_1$ .

3. Написати функцію генерації ключових пар для RSA. Після генерування функція повинна повертати та/або зберігати секретний ключ  $(d, p, q)$  та відкритий ключ  $(n, e)$ . За допомогою цієї функції побудувати схеми RSA для абонентів А і В – тобто, створити та зберегти для подальшого використання відкриті ключі  $(e, n)$ ,  $(e_1, n_1)$  та секретні  $d$  і  $d_1$ .

Для досягнення такого результату було розроблено функцію `generate_keys`, яка приймає на вхід  $p$  та  $q$  а повертає секретний ключ  $(d, p, q)$  та відкритий ключ  $(n, e)$ . Алгоритм роботи функції було взято з методичних вказівок до лабораторної роботи.

4. Написати програму шифрування, розшифрування і створення повідомлення з цифровим підписом для абонентів А і В. Кожна з операцій (шифрування, розшифрування, створення цифрового підпису, перевірка цифрового підпису) повинна бути реалізована окремою процедурою, на вхід до якої повинні подаватись лише ті ключові дані, які необхідні для її виконання. За допомогою датчика випадкових чисел вибрати відкрите повідомлення М і знайти криптограму для абонентів А і В, перевірити правильність розшифрування. Скласти для А і В повідомлення з цифровим підписом і перевірити його.

Було реалізовано функції `encrypt_rsa`, `decrypt_rsa`, `create_sign_rsa`, `verify_sign_rsa`. Всі формули для роботи функцій було взято з методичних вказівок до лабораторної роботи.

5. За допомогою раніше написаних на попередніх етапах програм організувати роботу протоколу конфіденційного розсилання ключів з підтвердженням справжності по відкритому каналу за допомогою алгоритму RSA. Протоколи роботи кожного учасника (відправника та приймаючого) повинні бути реалізовані у вигляді окремих процедур, на вхід до яких повинні подаватись лише ті ключові дані, які необхідні для виконання. Перевірити роботу програм для випадково обраного ключа  $0 < k < n$ .

Для реалізації завдання було написано функції `send_message` та `receive_message`. Алгоритми роботи знову ж таки було взято з методичних вказівок.

Тепер можемо перевірити роботу функцій. А також виконати перевірку за допомогою запропонованого у методичці онлайн інструмента.

Для початку просто виконаємо наші операції та переконаємося, що все працює правильно.

```
For A:
p = 59006266589550610188633398537400837086051515902412333939026797195552851837421
q = 71226222163457968750014058549345924140810658457433425846309581115813744070583
n = 420279345314355912170026508107240402345186040214430512411413018515873300875322692607832039350925012716513698429272064271164771850978223712255464606443
e = 33421414364347530172924803112640702419828179143313358442762221854854065628935501486002185659610789972393602351722776514970178026375128269194059175415501
d = 313506531050795127823001893181470660759879619482921215381312518678023281459827097382061842355252737367489914057976898844955132635625967857443425923998541

For B:
p1 = 59006266589550610188633398537400837086051515902412333939026797195552851837421
q1 = 71226222163457968750014058549345924140810658457433425846309581115813744070583
n1 = 7840981339291022364776489706852412050130536655095016541264219235252503101536557496601875864129259700407413295118220566012438003272457484395337562215373613
e1 = 2643049675849061091087143065988384018262458591119613766818400607464823558627184333279060057211440880842656149548466828460689171101002039292461566462977693
d1 = 7772917361714646069286733103372094010885167387779337197311070733122065173883159117504725220953855176416665778572767095000912819302341624985196805432857225

Msg: 153550221024820063304316897556362482670854577143187512232764431911709041458412557657580281893655209625818715886130623253473226367422972808237595347227719
Encrypted msg: 244845338235356068313873571680938610060650229400758801831231308738616001630567506215260710449485619504648729690295302068734032178892396807371977813940115
Decrypted msg: 153550221024820063304316897556362482670854577143187512232764431911709041458412557657580281893655209625818715886130623253473226367422972808237595347227719
k = 332649155784232096853280549859790136536571686161881598279763263119904870881070703193726222415879592319206345502788351925884064519635848380416513694675228
(676700509496993391161909692280043400928791776559482288351515716551030129585426761402857066102441522646370171983324831202612508181493935040028086813342068, 71037163098398667202197039377812381090
576534350830214652638147526804418786183670524152479728536782296849713654911473863877612564101836744461221644048485)
Authentication success
Sign: 286750510022035010692829417012459033015109171342221520570422318656802672616053123083233620528168077685121102070560303800510318492228018722524226739525455
Verification success
```

Тепер перевіримо роботу функцій за допомогою сайту. Спочатку отримуємо ключ.

## RSA Testing Environment

Server Key

Encryption

Decryption

Signature

Verification

Send Key

Receive Key

### Get server key

Clear

Key size256Get key

Modulus92B9DBFF312ADB3B632556A769324CC6CA7A5920D2FEA6A9BE32BB8F01CBD9C7

Public exponent10001

Тепер перевіримо шифрування. В якості повідомлення виберемо “1525”.

## RSA Testing Environment

Server Key

Encryption

Decryption

Signature

Verification

Send Key

Receive Key

### Encryption

Clear

Modulus92B9DBFF312ADB3B632556A769324CC6CA7A5920D2FEA6A9BE32BB8F01CBD9C7

Public exponent10001

Message1525Bytes

Encrypt

Ciphertext5E736EBB1FF64E5A6B689D3B5F2949169A652A6C97D5EDBA5FEFE7F38FF9EF40

Ciphertext: 0x5e736ebb1ff64e5a6b689d3b5f2949169a652a6c97d5edba5fefe7f38ff9ef40

Далі перевіряємо розшифрування.

## RSA Testing Environment

[Server Key](#)  
[Encryption](#)  
[Decryption](#)  
[Signature](#)  
[Verification](#)  
[Send Key](#)  
[Receive Key](#)

### Decryption

✖ Clear

Ciphertext

5E736EBB1FF64E5A6B689D3B5F2949169A652A6C97D5EDBA5FEFE7F38FF9EF4

Bytes

Decrypt

Message

1525

Бачимо, що ми отримали початкове повідомлення.

Тепер перевіряємо цифровий підпис. Спочатку створення.

## RSA Testing Environment

[Server Key](#)  
[Encryption](#)  
[Decryption](#)  
[Signature](#)  
[Verification](#)  
[Send Key](#)  
[Receive Key](#)

### Sign

✖ Clear

Message

1525

Bytes

Sign

Signature

8C8AD159E021DBE3A081A894740466DCE47281820440CB56DADAC6CFA1509CDC

А тепер перевіряємо його.

## RSA Testing Environment

[Server Key](#)  
[Encryption](#)  
[Decryption](#)  
[Signature](#)  
[Verification](#)  
[Send Key](#)  
[Receive Key](#)

### Verify

Clear

Message

1525

Bytes

Signature

8C8AD159E021DBE3A081A894740466DCE47281820440CB56DADAC6CFA1509CDC

Modulus

92B9DBFF312ADB3B632556A769324CC6CA7A5920D2FEA6A9BE32BB8F01CBD9C7

Public exponent

10001

Verify

Verification

true

Sign is: 63569068584622326108610496896038483234608129002822606241375539641871542295772

Verification success

Код, який використовувався для тестування на сайті наведено в файлі main.py.

## Висновки

В ході виконання комп'ютерного практикуму відбулося ознайомлення з криптосхемою RSA, використано її для створення секретного зв'язку. Також було реалізовано тест Міллера-Рабіна, за допомогою якого було згенеровано ключі для криптосистеми RSA. Було також перевірено коректність реалізації функцій за допомогою тестового онлайн середовища.