**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

Виконали:
Студенти групи ФБ-32
Коптєва Ганна, Чупріна Вікторія

Київ – 2025

**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 $pq \leq p_1 q_1$; 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 М і знайти криптограму для абонентів А и 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 $0 < k < n$.

**Хід роботи:**

**Генерація простих чисел**

Для пошуку випадкових простих чисел використано функцію get_random_prime(bits) у нашому коді, яка базується на вбудованому генераторі псевдовипадкових чисел та

самостійно реалізованому тесті Міллера-Рабіна з попередніми пробними діленнями. Було згенеровано дві пари простих чисел $p, q$ і $p_1, q_1$ довжиною 256 біт.

У результаті отримали:

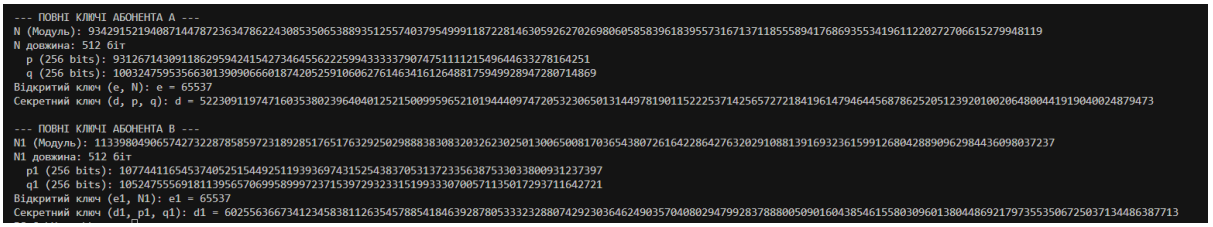
| **Абонент** | **Просте число** | **Значення** | **Довжина** |
|---|---|---|---|
| **A** | $p$ | 59587187144443188946839569655482188923784326132697431635425371206789246913959 | 256 bits |
| **A** | $q$ | 11230202485562384902759176828412814920357747916233746977157639958192679458521 1 | 256 bits |
| **B** | $p_1$ | 79421660228844666877043142081648579773783489975204460728975133827740866702727 | 256 bits |
| **B** | $q_1$ | 10785768039252557855197766728127207359889833770834428833607240997627438733690 1 | 256 bits |

Перевірка умови $pq \leq p_1 q_1$:

Статус: N_A (512 bits) <= N_B (512 bits) - OK

**Генерація ключових пар RSA**

За допомогою функції GenerateKeyPair($p, q$) було побудовано схеми RSA для абонентів A і B. Функція обчислила модуль $N = pq$, функцію Ейлера $\phi(n)$, обрала відкриту експоненту $e = 65537$ та знайшла секретну експоненту $d = e^{-1} mod \phi(n)$.




--- ПОВНІ КЛЮЧІ АБОНЕНТА A ---

N (Модуль):

93429152194087144787236347862243085350653889351255740379549991187228146305 9

26270269806058583961839557316713711855589417686935534196112202727066152799448119
N довжина: 512 біт
p (256 bits):
93126714309118629594241542734645562225994333337907475111121549644633278164251
q (256 bits):
10032475953566301390906660187420525910606276146341612648817594992894728071486 9
Відкритий ключ (e, N): e = 65537
Секретний ключ (d, p, q): d =
52230911974716035380239640401252150099596521019444097472053230650131449781901152225371425657272184196147946445687862520512392010020648004419190400248794 73

--- ПОВНІ КЛЮЧІ АБОНЕНТА B ---
N1 (Модуль):
11339804906574273228785859723189285176517632925029888383083203262302501300650081703654380726164228642763202910881391693236159912680428890962984436098037237
N1 довжина: 512 біт
p1 (256 bits):
10774411654537405251544925119393697431525438370531372335638753303380093123739 7
q1 (256 bits):
10524755569181139565706995899972371539729323315199333070057113501729371164272 1
Відкритий ключ (e1, N1): e1 = 65537
Секретний ключ (d1, p1, q1): d1 =
60255636673412345838112635457885418463928780533323288074292303646249035704080294799283788800509016043854615580309601380448692179735535067250371344863877713

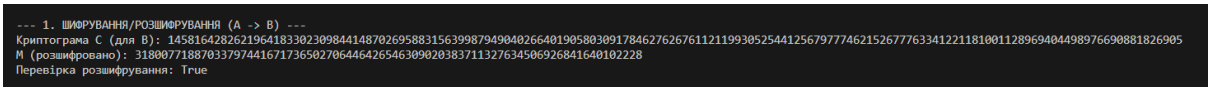
**Шифрування, розшифрування, створення цифрового підпису, перевірка цифрового підпису**

Для демонстрації були використані згенеровані ключові пари абонентів A та B (Завдання 3). Відкрите повідомлення M (обране випадковим чином) та його числове значення:

```
--- Шифрування, розшифрування, створення цифрового підпису, перевірка цифрового підпису (Завдання 4) ---
Оригінальне повідомлення M: 31800771887033797441671736502706446426546309020383711327634506926841640102228
```

Спочатку була перевірена пара Encrypt/Decrypt. Абонент A зашифрував повідомлення M для абонента B, використовуючи його відкритий ключ $(e_B, N_B)$. Потім абонент B розшифрував отриману криптограму $C$ за допомогою свого секретного ключа $(d_B, N_B)$.

Перевірка підтвердила, що розшифроване повідомлення повністю збігається з оригінальним, доводячи коректність реалізації шифрування.



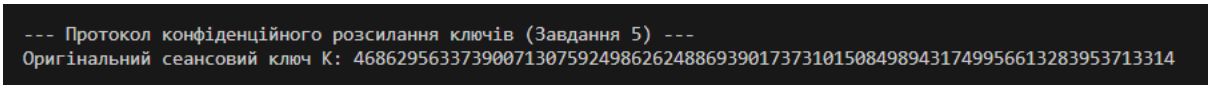

--- 1. ШИФРУВАННЯ/РОЗШИФРУВАННЯ (A -> B) ---
Криптограма C (для B):
145816428262196418330230984414870269588315639987949040266401905803091784627626761121199305254412567977746215267776334122118100112896940449897669088182690 5
M (розшифровано):
318007718870337974416717365027064464265463090203837113276345069268416401022 28
Перевірка розшифрування: True

Далі була перевірена пара Sign/Verify. Абонент A створив підпис $S$ повідомлення M, використовуючи свій секретний ключ $(d_A,\ N_A)$. Потім абонент B перевірив цей підпис, використовуючи відкритий ключ абонента A $(e_A,\ N_A)$. Перевірка пройшла успішно, що підтвердило коректність реалізації механізму цифрового підпису.


--- 2. ЦИФРОВИЙ ПІДПИС (A підписує) ---
Підпис S:
420280153938447135369342577544047578267004513032511571823202792586483216730801681025974021583808564305328414212240848834741740256624511069736448475959 6743
Перевірка підпису: True

**Протокол конфіденційного розсилання ключів**

Для перевірки роботи протоколу було обрано випадковий сеансовий ключ K (згідно з вимогою $0 < k < n$), використовуючи датчик випадкових чисел.




Абонент A (Відправник) активував процедуру SendKey(). Ця процедура виконала дві операції: 1) зашифрувала ключ K за допомогою відкритого ключа Абонента B $(e_B,\ N_B)$ для забезпечення конфіденційності, створивши криптограму C; 2) підписала ключ K за допомогою власного секретного ключа $(d_A,\ N_A)$ для забезпечення справжності, створивши підпис S. Отриманий пакет (C, S) був надісланий Абоненту B.

```python
def SendKey(K: int, d_A: int, e_B: int, n_A: int, n_B: int) -> Tuple[int, int]:
    C = Encrypt(K, e_B, n_B)
    S = Sign(K, d_A, n_A)

    return C, S
```

[АБОНЕНТ A (Відправник)]
Надсилає C (шифрований K):
795027573760376170551900117198332139005695311705409632771852523867037996026084533332697516642629912409831866239748905723723266208776005791579361851943 0773
Надсилає S (підписаний K):
510689354140268333200600616209882041979971574147956725844828289918357088275099273910894109635452503025821173145497167712879588912685155089646190809320 2967

Абонент B (Отримувач) активував процедуру ReceiveKey(). Спочатку він розшифрував криптограму C за допомогою власного секретного ключа $(d_B, N_B)$, щоб отримати ключ $K_{received}$. Далі він перевірив підпис S на отриманому $K_{received}$, використовуючи відкритий ключ Абонента A $(e_A, N_A)$. Перевірка справжності повернула True, що підтвердило цілісність даних та особу відправника.

```
def ReceiveKey(C: int, S: int, d_B: int, e_A: int, n_A: int, n_B: int) -> Union[Tuple[bool, int], Tuple[bool, None]]:
    K_decrypted = Decrypt(C, d_B, n_B)
    is_verified = Verify(K_decrypted, S, e_A, n_A)

    if is_verified:
        return True, K_decrypted
    else:
        return False, None
```

[АБОНЕНТ B (Отримувач)]
Результат перевірки справжності: True
Отриманий ключ K:
468629563373900713075924986262488693901737310150849894317499566132839537133 14

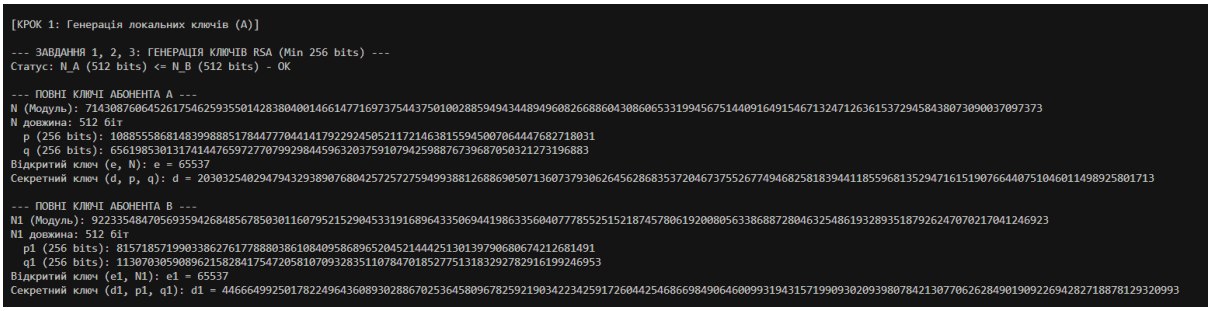
Фінальна перевірка підтвердила, що отриманий ключ $K_{received}$ повністю збігається з оригінальним ключем K. Це доводить, що протокол розсилання ключів був реалізований коректно, успішно забезпечивши і конфіденційність, і підтвердження справжності.

```
if is_protocol_ok and K_received is not None:
    print(f"  Отриманий ключ K: {K_received}")
    print(f"  ПЕРЕВІРКА КЛЮЧІВ: {K_session == K_received}")
else:
    print("  ПОМИЛКА: Протокол завершено невдачею. Підпис недійсний.")
```

ПЕРЕВІРКА КЛЮЧІВ: True

**Двостороння перевірка коректності операцій**

Спочатку були згенеровані ключові пари для Абонента А (Локально), використовуючи самостійно реалізований тест Міллера-Рабіна (Завдання 1-3). Було отримано модуль $N_A$ довжиною 512 біт.




--- ЗАВДАННЯ 1, 2, 3: ГЕНЕРАЦІЯ КЛЮЧІВ RSA (Min 256 bits) ---
Статус: N_A (512 bits) <= N_B (512 bits) - OK

--- ПОВНІ КЛЮЧІ АБОНЕНТА A ---
N (Модуль):
7143087606452617546259355014283804001466147716973754437501002885949434489496082668860430860653319945675144091649154671324712636153729458438073090037097373
N довжина: 512 біт
p (256 bits):
1088555868148399888517844777044141792292450521172146381559450070644476827180 31
q (256 bits):
6561985301317414476597277079929844596320375910794259887673968705032127319683
Відкритий ключ (e, N): e = 65537
Секретний ключ (d, p, q): d =
2030325402947943293890768042572572759499388126886905071360737930626456286835372046737552677494682581839441185596813529471615190766440751046011498925801713

--- ПОВНІ КЛЮЧІ АБОНЕНТА B ---
N1 (Модуль):
9223354847056935942684856785030116079521529045331916896433506944198633560407778552515218745780619200805633868872804632548619328935187926247070217041246923
N1 довжина: 512 біт
p1 (256 bits):
8157185719903386276177888038610840958689652045214442513013979068067421268149 1

q1 (256 bits):
113070305908962158284175472058107093283511078470185277513183292782916199246953
Відкритий ключ (e1, N1): e1 = 65537
Секретний ключ (d1, p1, q1): d1 = 44666499250178224964360893028867025364580967825921903422342591726044254686698490646009931943157199093020939807842130770626284901909226942827188781293209993

Далі, було здійснено запит до веб-сервера (serverGetKey) для отримання відкритого ключа Абонента B (Сервер). Сервер повернув ключ довжиною 256 біт ($N_B$= 256 bits, $e_B$= 65537).

```
[КРОК 2: Отримання ключів Сервера (B)]

[*] Отримання відкритого ключа Сервера (size=256)...
[*] Ключ Сервера (B) отримано: N_B=256 bits, e_B=65537
```

Було проведено чотири тести для перевірки коректності всіх процедур RSA для тестового повідомлення M='Hello World!':
Тест A (Перевірка Decrypt):Серверу було надіслано запит на шифрування повідомлення M відкритим ключем Абонента A. Сервер повернув криптограму (C_server: 089B...). Ця криптограма була успішно розшифрована локальною функцією Decrypt, відновивши оригінальне повідомлення 'Hello World!'.

```
[КРОК 3: Запуск 4 тестів сумісності]

--- [A. ТЕСТ: Сервер Encrypt -> Локальний Decrypt] ---
[*] Надсилаємо M='Hello World!' та (e_A, n_A) на Сервер для шифрування...
[*] Сервер повернув C_server: 16E6CC68AC5FF80304C6...
[*] Локально розшифровано (Decrypt): 'Hello World!'
```

Тест B (Перевірка Encrypt):Повідомлення M було зашифровано локальною функцією Encrypt, використовуючи відкритий ключ Сервера (B). Отримана криптограма (C_local: 5b12...) була надіслана Серверу, який успішно її розшифрував, повернувши {'message': 'Hello World!'}.

```
--- [B. ТЕСТ: Локальний Encrypt -> Сервер Decrypt] ---
[*] Локально зашифровано (Encrypt): C_local=0f6a6daf103a0bcd4d83...
[*] Надсилаємо C_local на Сервер для розшифрування...
[*] Сервер розшифрував: {'message': 'Hello World!'}
```

Тест C (Перевірка Sign):Повідомлення M було підписано локальною функцією Sign, використовуючи секретний ключ Абонента A. Отриманий підпис (S_local: 1bbd...) та відкритий ключ A були надіслані Серверу, який успішно перевірив підпис, повернувши {'verified': True}.

```
--- [C. ТЕСТ: Локальний Sign -> Сервер Verify] ---
[*] Локально створено підпис (Sign): S_local=7a7ed3fb3c10f8da8825...
[*] Надсилаємо M, S_local та (e_A, n_A) на Сервер для перевірки...
[*] Сервер перевірив: {'verified': True}
```

Тест D (Перевірка Verify):Серверу було надіслано запит на підписання повідомлення M секретним ключем Сервера. Отриманий підпис (S_server: 7B0C...) був успішно перевірений локальною функцією Verify.

```
--- [D. ТЕСТ: Сервер Sign -> Локальний Verify] ---
[*] Просимо Сервер підписати M='Hello World!'...
[*] Сервер повернув S_server: 6CE9EFF7BF8066D7E097...
[*] Локальна перевірка (Verify): True
```

Всі чотири тести завершилися з результатом "Успіх"

```
--- ЗАГАЛЬНИЙ РЕЗУЛЬТАТ ПЕРЕВІРКИ ---
  Перевірка Decrypt (A): Успіх
  Перевірка Encrypt (B): Успіх
  Перевірка Sign (C):    Успіх
  Перевірка Verify (D):  Успіх
PS C:\Users\kante>
```