

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”  
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Криптографія

КОМП'ЮТЕРНИЙ ПРАКТИКУМ 4

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

ФБ-32 Дорошенко Ілля

**Мета:** 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 і рекомендації щодо 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  $1 < p, q$  довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб  $pq \leq p_1q_1$ ;  $p$  і  $q$  – прості числа для побудови ключів абонента А,  $1 < p < q_1$  – 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  $(e, n)$ ,  $(d_1, n_1)$  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()`.

Кожну операцію рекомендується перевіряти шляхом взаємодії із тестовим середовищем, розташованим за адресою <http://asymcryptwebservice.appspot.com/?section=rsa>

Наприклад, для перевірки коректності операції шифрування необхідно а) зашифрувати власною реалізацією повідомлення для серверу та розшифрувати його на сервері, б) зашифрувати на сервері повідомлення для вашої реалізації та розшифрувати його локально.

## Хід роботи:

### Завдання 1:

**Схема Горнера (Modular Exponentiation):** Функція для швидкого піднесення до степеня за модулем  $x^a \pmod{m}$ . Ця функція є необхідною основою для всіх імовірнісних тестів простоти.

**Пробні Ділення (Trial Division):** Функція для швидкої перевірки числа на подільність малими простими числами (наприклад, до 1000). Це дозволить швидко відкинути більшість складених чисел.

**Тест Міллера-Рабіна (Miller-Rabin Primality Test):** Імовірнісний тест Міллера-Рабіна, який базується на властивостях сильних псевдопростих чисел.

- Розкласти  $p-1 = d \cdot 2^s$ , де  $d$  непарне.
- Вибрати випадкову основу  $x$ .
- Перевірити умови сильного псевдопростого числа:  $x^d \equiv 1 \pmod{p}$  або  $x^{d \cdot 2^r} \equiv -1 \pmod{p}$  для деякого  $0 \leq r < s$ .
- Повторити тест  $k$  разів для зменшення ймовірності помилки до  $4^{-k}$ .

### Функція Генерації Простого Числа:

- Генерувати випадкове непарне число  $x$  у заданому інтервалі  $p_0, p_1$  або заданої довжини.
- Починаючи з  $x$  (або найменшого непарного  $\geq x$ ), перевіряти послідовні непарні числа  $p$ :  $p = x, x + 2, x + 4, \dots$ .
- Для кожного  $p$ :
  - Виконати пробні ділення. Якщо складене, перейти до наступного непарного числа.
  - Виконати тест Міллера-Рабіна  $k$  разів (наприклад,  $k=40$ ).
- Перше число, яке пройде обидва тести, є шуканим простим числом  $p$ .

### Завдання 2:

**Вибір Довжини:** Визначити необхідну бітову довжину для  $p, q, p_1, q_1$ . Оскільки модуль  $n = pq$  має бути довжиною щонайменше  $2 \cdot 256 = 512$  біт, кожне з  $p, q$  повинно мати довжину близько 256 біт.

**Генерація  $p$  та  $q$  (Абонент А):** Використовуючи функцію `find_prime` з Завдання 1, згенерувати два випадкові прості числа  $p$  та  $q$  довжиною, наприклад, 256 біт.

### Генерація $p_1$ та $q_1$ (Абонент В):

- Згенерувати  $p_1$  та  $q_1$  довжиною, наприклад, 256 біт.
- Перевірити умову:  $pq \leq p_1q_1$ .
- Якщо умова не виконується, повторно згенерувати  $p_1$  та  $q_1$  або збільшити їхню бітову довжину (наприклад, до 257 біт), щоб забезпечити  $p_1q_1 \geq n$ . (Умова  $p_1q_1 \geq n$  (де  $n = pq$  і  $p_1 = p_1q_1$ ) необхідна для протоколу розсилання ключів.

Результат виконання коду:

```
PS D:\KPI\crypto25-26> python -u "d:\KPI\crypto25-26\lab4\Doroshenko_fb-32_cp4\lab4.py"
Генерація простого числа p (довжина 256 bit)...
Знайдено p: 10351158134228777607606091987013007694347113339832159957389053508751478764271
Генерація простого числа q (довжина 256 bit)...
Знайдено q: 87037458041674432872089264489973261834958291926971828627012118287963472064781

Генерація простих чисел для Абонента В...
Знайдено p1: 94252667932259699408077004579879739733027057561279751374590333750886255854221
Знайдено q1: 115738636053252341675606059943160134565131313664843414723256530537462036985689

Перевірка умови pq <= p1q1: 9009384917906742515568335263192871163937162496938842303918965800871156781080586990559560337023634456141949010027
063507074857250379060921422176254740239651 <= 1090867523085985328382697192732246237461010632672372522518081682853633163787127578339635974268
2846445246112768633746430060050080649481673533860500647243269 -> True

--- Результати генерації ---
Абонент А: p=10351158134228777607606091987013007694347113339832159957389053508751478764271, q=870374580416744328720892644899732618349582919
26971828627012118287963472064781, n=90093849179067425155683352631928711639371624969388423039189658008711567810805869905595603370236344561419
49010027063507074857250379060921422176254740239651
Абонент В: p1=94252667932259699408077004579879739733027057561279751374590333750886255854221, q1=11573863605325234167560605994316013456513131
3664843414723256530537462036985689, n1=10908675230859853283826971927322462374610106326723725225180816828536331637871275783396359742682846445
246112768633746430060050080649481673533860500647243269
PS D:\KPI\crypto25-26>
```

```
PS D:\KPI\crypto25-26> python -u "d:\KPI\crypto25-26\lab4\Doroshenko_fb-32_cp4\lab4.py"
Генерація простого числа p (довжина 256 bit)...
Знайдено p: 10826301856113729111133702639045536118376805710037581063891637801418190472429
Генерація простого числа q (довжина 256 bit)...
Знайдено q: 69981349445496668637469710527994895126706752551686035187140244797008229006159

Генерація простих чисел для Абонента В...
n1 (653820025134926485465712217569655317693075663751438683159299861224369214222906286438274445116426957045272913674955918308404505048193576561197030545175591) < n
(7576392133951240711800349867368899755928551652486169987646772375904949324247065836908144969815423588957597417808026460478487093854112606710245253360690211). Повт
орна генерація p1, q1.

Генерація простих чисел для Абонента В...
n1 (7172713327951543789681086704675944091695190498018064403394135893177380813435196802537951345545644050132611408888848974268605923007545293253762496886448369) < n
(7576392133951240711800349867368899755928551652486169987646772375904949324247065836908144969815423588957597417808026460478487093854112606710245253360690211). Повт
орна генерація p1, q1.

Генерація простих чисел для Абонента В...
n1 (6394103944711528694376793353755795611202552546413492367921693257130683734579509072973369573597077987527793421641419918547613362475634374707410020051671) < n
(7576392133951240711800349867368899755928551652486169987646772375904949324247065836908144969815423588957597417808026460478487093854112606710245253360690211). Повт
орна генерація p1, q1.

Генерація простих чисел для Абонента В...
Знайдено p1: 98812676163440074225371273954013496401021185580037953525050622482165273582349
Знайдено q1: 102712585806416004658946932582334552799089264906773713553871782648264989576149

Перевірка умови pq <= p1q1: 757639213395124071180034986736889975592855165248616998764677237590494932424706583690814496981542358895759741780802646047848709385411260
6710245253360690211 <= 10149305479198936037659461540997598559800706836116162195224784213705862246571654500419924546778672535577325427611744485951472567664764584438
505288357794001 -> True
```

### Завдання 3:

**Обчислення n та  $\phi(n)$ :** Обчислити модуль  $n = pq$  та функцію Ейлера  $\phi(n) = (p - 1)(q - 1)$ .

**Вибір e:** Обрати відкриту експоненту e. Рекомендовано використовувати  $e = 2^{16} + 1$ .

**Перевірка  $\gcd(e, \phi(n))$ :** Перевірити, чи e і  $\phi(n)$  є взаємно простими:  $\gcd(e, \phi(n)) = 1$ . Якщо ні, обрати інше e (або використовувати фіксоване e та повторно генерувати p, q).

**Знаходження d:** Знайти секретну експоненту d як обернений елемент до e за модулем  $\phi(n)$ . Для цього використовується Розширений алгоритм Евкліда.

Результат виконання коду:

```
--- Результати генерації 3 завдання ---
Ключі А: PK=(50043733738311760564287759685901621351356021272421148893257845945955426190206639146101866259054226973955352789918528777766569999441813752008334228739
9901, 65537), SK=(1666088204275789712303270867733780118505938673028910459285583625577614849604199722668092900996276752122945901999721491602062918965657826770481724
708416681, 64843256759065544097899528384882984755106699364004917020004595241246149263619, 7717646558724904582044368586318855690539768243582036946848477153255893590
4189)
Ключі В: PK=(967436871419580663204802571219897663089214428605570340655098695183006665695832621837170806009277544095391276549455050940299553421624100413693145496064
9587, 65537), SK=(5214124108867117534479765082415921579510388337459138436702839023948938988007918858214133052729740411962600822565756816917613511851970389343558884
649712849, 8517748535592261882965525188621611706862480404218805452608386339663857871823, 1135789425316797455916040513114801465884940340335687870812612052993777828
00669)
PS D:\KPI\crypto25-26>
```

### Завдання 4:

**Шифрування (Encrypt):**  $C = M^e \pmod n$ .

**Розшифрування (Decrypt):**  $M = C^d \pmod n$ .

**Створення Підпису (Sign):**  $S = M^d \pmod{n}$ .

**Перевірка Підпису (Verify):** Перевірити, чи  $M = S^e \pmod{n}$ .

Для практичної реалізації підпису повідомлення  $M$  спочатку піддається стисненню за допомогою геш-функції  $H(M)=m$ . У цілях цього практикуму ми будемо використовувати просту вбудовану геш-функцію Python (наприклад, `hashlib.sha256`) для отримання  $m$ .

Результат виконання коду:

```
--- Результати 4 завдання ---

--- Шифрування та Розшифрування ---
ВТ (M): 123456789012345
ШТ (C): 9214580055627771326754994864827563226684029747783400676709146667478028414511206275754076792825061012010843409244056820842648997489311594125331112704012370
Розшифр. M': 123456789012345
Перевірка розшифр.: True

--- Цифровий Підпис та Перевірка (Абонент А) ---
Повідомлення (M): 'Абонент А підписує це повідомлення.'
Геш H(M) mod n: 75011192470180708724842653574387097259144788499887219778146045778773312832135
Підпис (S): 6844997565565415439065404551844956557067141071658630033157367451299377947921393370912816048255069435469352320101912908548544142020592568135657161876063751
Відновлений Геш (S^e mod n): 75011192470180708724842653574387097259144788499887219778146045778773312832135
Перевірка підпису: True
PS: D:\KPI\crypto\25_26\
```

## Завдання 5:

### Ролі та Ключі

- **Абонент А (Відправник):** має ключі  $(e, n)$  та  $d$  (власні), а також  $(e_1, n_1)$  (відкритий ключ В).
- **Абонент В (Отримувач):** має ключі  $(e_1, n_1)$  та  $d_1$  (власні), а також  $(e, n)$  (відкритий ключ А).
- **Умова:**  $n_1 \geq n$ .

### Відправник (SendKey):

- Обчислює підпис ключа  $k$ :  $S = k^d \pmod{n}$ .
- Шифрує ключ  $k$ :  $k_1 = k^e \pmod{n_1}$ .
- Шифрує підпис  $S$ :  $S_1 = S^e \pmod{n_1}$ .
- Відправляє  $(k_1, S_1)$ .

### Отримувач (ReceiveKey):

- Розшифровує ключ:  $k = k_1^{d_1} \pmod{n_1}$ .
- Розшифровує підпис:  $S = S_1^{d_1} \pmod{n_1}$ .
- Перевіряє підпис А:  $k_{\text{check}} = S^e \pmod{n}$ .
- Автентифікація: Якщо  $k_{\text{check}} = k$ , підпис вірний, відправник автентифікований.

Результат виконання коду:

```
--- Результати 5 завдання ---
А (Відправник) генерує: k=12203022939108352833514010025446813044173709722064209868821034958737087719926580405384748748570343383336930314732163753066955176781238213
07210222652917427, S=3337997597016304600951057523688047912816978271184163618806035706230504871666814814334600866239829819726210785116348180031963836504550233525749
242404621985
А відправляє: k1=565364958508595497592821426099253660545573361541936968713452088388107116255682947543541012829348419684845915374479284701272202389059934875180652
44843239, S1=39624061023976353130480390123031082744249379948596267954506297443653118981975256647070637709611967282248344024509158401886518835251652522785972562906
7685
В (Отримувач) розшифровує: k=12203022939108352833514010025446813044173709722064209868821034958737087719926580405384748748570343383336930314732163753066955176781238
21307210222652917427, S=3337997597016304600951057523688047912816978271184163618806035706230504871666814814334600866239829819726210785116348180031963836504550233525
749242404621985
В перевіряє підпис (S^e mod n): k_check=122030229391083528335140100254468130441737097220642098688210349587370877199265804053847487485703433833369303147321637530669
5517678123821307210222652917427
Автентифікація успішна: True. Ключ відновлено: True
PS: D:\KPI\crypto\25_26\
```

Кожну операцію рекомендується перевіряти шляхом взаємодії із тестовим середовищем, розташованим за адресою

<http://asymcryptwebsevice.appspot.com/?section=rsa>

```
--- Шифрування та Розшифрування ---  
ВТ (М): 123456789012345  
ШТ (С): 4165023424417738322799370800385865722378061614335415608379464679725005877399393367743025970601491780935772076488458612059547852785477127167625913988889724  
Розшифр. М': 123456789012345  
Перевірка розшифр.: True
```

## Encryption

Clear

Modulus

7f55351bea2e14f3bc9d48b43fc7993b398d22fc86bdc07e6cf14bce7732e6a9e90f93d13a0fdf53bdf6e576e41d8d8

Public exponent

10001

Message

7048860DDF79

Bytes

Encrypt

Ciphertext

4F8635DE403DB2149CB6D120C93460AE7DB46C59136B60A6A75104C5ADBA0DB1A4A2679ED144B45B2942

## Decimal to Hexadecimal converter

From

Decimal

To

Hexadecimal

Enter decimal number

41650234244177383227993708003858657

10

Convert

Reset

Swap

Hex result

(4165023424417738322799370800385865722378061614335415608379464679725005877399393367743025970601491780935772076488458612059547852785477127167625913988889724)<sub>10</sub> = (4F8635DE403DB2149CB6D120C93460A

Hex number (128 digits)

4F8635DE403DB2149CB6D120C93460AE7DB46C59136B60A6A75104C5ADBA0DB1A4A2679ED144B45B2942FC35D261462C38D9E263797CDB06B21F9D24DE313C7C

16

Hex signed 2's complement

## Verify

✕ Clear

Message

a5d6d494ced4065e4fe5c6f0c9e97f6c02962e7f4c1f0f09fc1fce8c9dd88287

Bytes ▼

Signature

c0418155bc89e345ea124c1392fa077607d39a717e62c27efd0336d82fc84197860ee36063b318c20cacf10e3964z

Modulus

73f84f1cd8e40adf994f173a5d0f69383efa5cbc78efae0644ebb9387f4743e46d8ab71c3b56862806ac9166069d0cd4

Public exponent

10001

Verify

Verification

true ✓

### Висновок:

У ході виконання практикуму було досліджено принципи роботи асиметричної криптосистеми RSA, методи генерації параметрів для неї та протоколи захищеного обміну даними.

Отримано практичні навички розробки програмного забезпечення для асиметричної криптографії, засвоєно особливості роботи алгоритму RSA та важливість коректного вибору параметрів криптосистеми (зокрема, простих чисел) для забезпечення стійкості захисту інформації.

