

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО»

Навчально-науковий фізико-технічний інститут
КАФЕДРА ІНФОРМАЦІЙНОЇ БЕЗПЕКИ

«Криптографія»

Комп'ютерний практикум №4

Студенти: Маврикін Едуард

Слобода Ірина

Група: ФБ-25

Варіант 2

Київ – 2025

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

Мета та основні завдання 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.

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.

Будемо реалізовувати функцію пошуку випадкового простого числа з заданого інтервалу, у якості датчика випадкових чисел будемо використовувати метод `random.randint()`. Для тесту перевірки на простоту використовуємо тест Міллера-Рабіна із попереднім пробним діленням. Для цього спочатку треба реалізувати тест пробних ділень, який використовує ознаку Паскаля. Знаємо, що натуральне число N (число яке ми тестуємо на простоту) записане у системі числення з основою B (у нашому випадку десятична система числення \Rightarrow основа 10):

$$N = \overline{a_i a_{i-1} \dots a_1 a_0} = a_i B^i + a_{i-1} B^{i-1} + \dots + a_1 B + a_0,$$

Це реалізовано у частині кода, яка виділена блакитним. Ми записуємо залишки від ділення у `num_parts = []`. Після чого обчислюємо послідовність

де $r_i = B^i \bmod m$.

- виділено зеленим та наприкінці розраховуємо залишок за формулою:

$$N \equiv \sum_{i=0}^i a_i r_i \pmod{m},$$

```

9 def pascal(N, m, B=10):
10     num_parts = []
11     while N > 0:
12         num_parts.append(N % B)
13         N //= B
14
15     r = [1]
16     for i in range(1, len(num_parts)):
17         r.append((r[i-1] * B) % m)
18
19     lishok = sum(d * r[i] for i, d in enumerate(num_parts)) % m
20     return lishok == 0

```

Далі виконуються пробні ділення на малих числах з використанням функції pascal():

```

22 def trial(N):
23     small = [2, 3, 5, 7, 11, 13, 17, 19, 23]
24     for prime in small:
25         if pascal(N, prime):
26             return False
27     return True

```

Тест Міллера-Рабіна реалізовано відповідно до трьох кроків з методички:

4.3. Імовірнісний тест Міллера-Рабіна

Оберемо деяке число $k \in N$.

Крок 0. Знаходимо розклад $p-1 = d \cdot 2^s$ та встановлюємо лічильник у 0.

Крок 1. Вибираємо випадкове число $x \in N$ з інтервалу $1 < x < p$ (незалежне від раніше обраних x). За допомогою алгоритму Евкліда знаходимо $\gcd(x, p)$. Якщо $\gcd(x, p) = 1$, то переходимо до кроку 2. Якщо $\gcd(x, p) > 1$, то p – складене число, алгоритм завершує свою роботу.

Крок 2. Перевіряємо, чи є p сильно псевдопростим за основою x :

2.1 Якщо $x^d \bmod p = \pm 1$, то p є сильно псевдопростим за основою x , інакше

2.2 Для всіх послідовних r від 1 до $s-1$ виконати такі дії:

Обчислити $x_r = x^{d \cdot 2^r} \bmod p$ (тобто $x_r = x_{r-1}^2 \bmod p$).

Якщо $x_r = -1$, то p є сильно псевдопростим за основою x ,

інакше якщо $x_r = 1$, то p не є сильно псевдопростим за основою x ,

інакше перейти до наступного значення r .

2.3 Якщо за кроки 2.1 та 2.2 не було встановлено, що p є сильно псевдопростим за основою x , то p не є сильно псевдопростим за основою x .

Якщо p виявилось не псевдопростим за основою x , то p – складене число, алгоритм завершує свою роботу; інакше лічильник збільшується на 1.

Крок 3. Якщо лічильник менший за k , то переходимо до кроку 1, інакше алгоритм завершує роботу.

```

29 ∨ def miller_rabin(p, k=10):
30 ∨     if p < 2 or p % 2 == 0:
31         return p == 2
32     s = 0
33     d = p - 1
34 ∨     while d % 2 == 0:
35         d //= 2
36         s += 1
37 ∨     def check(x):
38         x = pow(x, d, p)
39 ∨         if x == 1 or x == p - 1:
40             return False
41 ∨         for _ in range(s - 1):
42             x = pow(x, 2, p)
43 ∨             if x == p - 1:
44                 return False
45         return True
46     counter = 0
47 ∨     for _ in range(k):
48         x = random.randint(2, p - 2)
49
50 ∨         if gcd(x, p) != 1:
51             return False
52 ∨         if check(x):
53             return False
54         counter += 1
55 ∨     if counter < k:
56         return False
57     return True
58

```

Далі вже спробуємо знайти просте число на заданому інтервалі. Почнемо з малих значень, нехай це буде інтервалі 18-169:

```

lab4/Mavrykin_FB_25_Sloboda_FB-25_cp4/lab4.py5-26>
~~~~~
Початок діапазона: 18
Кінець діапазона: 169
Випадкове просте число у заданому діапазоні: 37
~~~~~

```

Тепер перевіримо на більшому інтервалі:

```

lab4/Mavrykin_FB_25_Sloboda_FB-25_cp4/lab4.py
~~~~~
Початок діапазона: 2
Кінець діапазона: 2567
Випадкове просте число у заданому діапазоні: 157
~~~~~

```

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 $pq \leq p_1q_1$; p, q – прості числа для побудови ключів абонента А, p_1, q_1 – абонента В.

Генеруємо пари на основі попередніх функцій та встановлюємо зазначені початок та кінець інтервалу на якому будуть обиратись прості числа (для того щоб вони були не менше 256 біт).

```
70
71 def gen_pairs(bit_length=256):
72     start = 2** (bit_length - 1)
73     finish = 2**bit_length - 1
74
75     p = find(start, finish)
76     q = find(start, finish)
77     p1 = find(start, finish)
78     q1 = find(start, finish)
79
80     while p * q > p1 * q1:
81         p1 = find(start, finish)
82         q1 = find(start, finish)
83
84     """print("\033[36m" + "~~~" * 50 + "\033[0m")
85     print("Перша пара (абонент А):")
86     print(f"p = {p}, q = {q}")
87     print("\n")
88     print("\nДруга пара (абонент В):")
89     print(f"p1 = {p1}, q1 = {q1}")"""
90     return (p, q), (p1, q1)
```

Перевіримо виконання:

```
Перша пара (абонент А):
p = 97165259655248766406792723409104283462189676731967204573733029582683175757063, q = 64111559864683813937874436690427287208031911906344541646610124768866461244643

Друга пара (абонент В):
p1 = 109065886204146938785345579311481385468433347626290125070149666341918550930653, q1 = 85012485552653786719392371365117320498992389862449793417411891400375421118027
```

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) , $(,)$ і n_1 та секретні d і d_1 .

Для реалізації генерації ключових пар, згідно до теоретичних вимог, нам необхідні деякі математичні функції – пошук оберненого елемента за модулем, який в свою чергу потребує розширеного алгоритму Евкліда та функція Ойлера. Реалізацію пошуку оберненого елемента та розширеного алгоритму Евкліда візьмемо з комп'ютерного практикуму 4. Функцію Ойлера реалізовано відповідно до формули

$$\varphi(n) = (p-1)(q-1).$$

```

91 def gcd_evc(a, b):
92     if b == 0:
93         return a, 1, 0
94     gcd, x1, y1 = gcd_evc(b, a % b)
95     x = y1
96     y = x1 - (a // b) * y1
97     return gcd, x, y
98
99 def obratn(a, m):
100     gcd, x, _ = gcd_evc(a, m)
101     if gcd != 1:
102         return None
103     return x % m
104
105 def oiler(p, q):
106     return (p - 1) * (q - 1)
107

```

Далі реалізовано саму функцію генерації RSA відповідно до логіки, яка розписана у теоретичних відомостях:

```

108 def rsa_pairs(bit_length=256):
109     (p, q), _ = gen_pairs(bit_length=256)
110     n = p * q
111     phi = oiler(p, q)
112     e = 2 ** 16 + 1
113     if gcd(e, phi) != 1:
114         raise ValueError("e повинно бути взаємнопростим з функцією Ойлера")
115     d = obratn(e, phi)
116     return (d, p, q), (n, e)
117

```

Перевіримо як воно працює:

```

Абонент А:
Секретний ключ: d = 416679013418302597484329661084808871947799871264027166369518719209208717572473851855558827303207867354991984689715545738419667371635438634668493464
2798877
Відкритий ключ: n = 62294163611550282481306916528308494524814563412255739221112640693509847214790988932298032240853139859647872840651127521128096146489909058238473077
8163509, e = 65537
p = 97165259655248766406792723409104283462189676731967204573733029582683175757063, q = 64111559864683813937874436690427287208031911906344541646610124768866461244643

Абонент В:
Секретний ключ: d1 = 14662093594704092094923334215035501600453466864109826408144535843881593835158888527424539121223069111065311177399114142267295312803443132542953918
04655357
Відкритий ключ: n1 = 79829660871988209987953024379063028029319503021779986151912307518523553640843199324567555882707217711851799819617325971629089186594195129158307276
88311941, e1 = 65537
p1 = 909368553091955652448452096141360818686978263394897423109122760524950121170059, q1 = 87785816433346368380636768227650040321292858445868933730245755109536763397999

```

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 A и B , перевірити правильність розшифрування. Скласти для A і B повідомлення з цифровим підписом і перевірити його.

Операцію шифрування реалізовано за логікою відповідною до формули:

```
133
134 def encrypt(message, public):
135     n, e = public
136     return pow(message, e, n)
137
```

$$C = M^e \bmod n ,$$

Операція розшифрування також реалізована відповідно до формули:

```
138 ✓ def decrypt(cipher, secret):
139     d, p, q = secret
140     n = p*q
141     return pow(cipher, d, n)
142
```

$$M = C^d \bmod n .$$

Створення цифрового підпису

```
142
143 ✓ def sign(message, secret):
144     d, p, q = secret
145     n = p*q
146     return pow(message, d, n)
147
```

$$S = M^d \bmod n$$

Перевірка цифрового підпису також відбувається за формулою:

```
147
148 ✓ def verify(message, sign, public):
149     n, e = public
150     return pow(sign, e, n) == message
151
```

$$M = S^e \bmod n,$$

Тобто, якщо ця рівність виконується, то повідомлення не було змінено і відповідає початковому стану.

Переглянемо, чи працює:

```
Повідомлення від А до В: 5204
Зашифроване: 468311228634306488942696607939198761193419332346763946954844709790878320780524284981402558445864104801145806152492587211951046617897080889832415017885487#
6

Повідомлення від В до А: 5934
Зашифроване: 229831865152759922767764058647988452593170697801179144929142245395255377867247737175164628943364776661620507716400515591077563692546316917909569989936219#
Розшифрування повідомлення від В: 5934

Розшифрування повідомлення від А: 5204

Підпис повідомлення від А: 4805567400003938644564224467055223651648439457221417370072433650578103450274726678751589267554447084450708002366369297612315885728434057264#
814759506750453

Підпис повідомлення від В: 2675757864011084982013192842735067037467856795307108819384416699246607357434135249324639798951554710037365705429985585196341830395295398729#
369951802748186

Перевірка цифрового підпису для А: True

Перевірка цифрового підпису для В: True
```

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 $0 < k < n$.

```
178
179 v def send_key(k, sender_se, sender_pub, receiver_pub):
180     en_k = encrypt(k, receiver_pub)
181     signa = sign(k, sender_se)
182     en_s = encrypt(signa, receiver_pub)
183     return en_k, en_s
184
185 v def receive_key(en_key, en_signa, receiver_se, sender_pub):
186     de_k = decrypt(en_key, receiver_se)
187     de_s = decrypt(en_signa, receiver_se)
188     valid_s = verify(de_k, de_s, sender_pub)
189     return de_k, valid_s
190
191 k = random.randint(1, public_b[1] - 1)
192 en_k, en_s = send_key(k, secret_a, public_a, public_b)
193 de_k, valid_s = receive_key(en_k, en_s, secret_b, public_a)
194
```

Тут ми генеруємо значення k відповідно до вимог – $0 < k < n$ (тобто менше за відкритий ключ, тому у коді граничний розмір генерації – `public_b[1]-1`). Також створюємо функції `send_key` та `receive_key`, які будуть відповідати за отримання та надсилання ключа.

Тут для відправки ключа необхідно спочатку зашифрувати k , створити підпис та зашифрувати підпис.

А для отримання ключа необхідно спочатку розшифрувати ключ, потім розшифрувати підпис та перевірити його.

Переглянемо як працює:

```
А відправляє ключ: 37510

Секретний ключ (A): d = 41667901341830259748432966108480087194779987126402716636951871920920871757247385185555882730320786735499198468971554573841966737163543863466849
34642798877, p = 971652596552487664067927234091042834621896767319672045737302952683175757063, q = 6411155986468381393787443669042728720803191190634454164661012476886
6461244643
Відкритий ключ (A): n = 62294163611550282481306916528308494524814563412255739221112640693509847214790988932298032240853139859647872840651127521128096146489900905823847
30778163509, e = 65537

Зашифрований ключ: 4639572048252274792159600681945801234404514558396730871585289029108127154143222598227816999749090724973736572029015058349940784027471253722472702227
465841

Зашифрований підпис: 4062726317658117099948773212676673611463369629031167517749230723475262800356868194512880970712712365156020688415775260619791352972351674897113230
32858253

Абонент В отримує ключ:
Секретний ключ (B): d = 14662093594704092094923334215035501600453466864109826408144535843881593835158888527424539121223069111065311177399114142267295312803443132542953
91804655357, p = 9093685309195565244845209614136081868697826339489742310912276052495012170059, q = 8778581643334636838063676822765004032129285844586893373024575510953
6763397999
Відкритий ключ (B): n = 7982966087198820998795302437906302802931950302177998615191230751852355364084319932456755588270721711851799819617325971629089186594195129158307
27688311941, e = 65537

Розшифрований ключ: 37510
Чи вірний підпис?: True
```

Тепер перевіримо це з використання сервісу <https://www.dcode.fr/rsa-cipher>

Results

Decryption using C,D,N

36002

RSA Cipher - dCode

Tag(s) : Modern Cryptography, Arithmetics

Share

RSA DECODER

Indicate known numbers, leave remaining cells empty.

VALUE OF THE CIPHER MESSAGE (INTEGER) C=1213305563070521001434280778373297439723521690181...

PUBLIC KEY E (USUALLY E=65537) E=65537

PUBLIC KEY VALUE (INTEGER) N=8976143044012417746661979585212454108045269130432...

PRIVATE KEY VALUE (INTEGER) D=1929260584371559826960047674402286167598847383482...

FACTOR 1 (PRIME NUMBER) P=9399211874141697317575941410821586272784477505460...

FACTOR 2 (PRIME NUMBER) Q=9549889037725396826907002306544062426546603879696...

INTERMEDIATE VALUE PHI (INTEGER) Φ=

DISPLAY

PLAINTEXT AS CHARACTER STRING

COMPUTED VALUES (C,D,E,N,P,Q,...)

PLAINTEXT AS INTEGER NUMBER

PLAINTEXT AS HEXADECIMAL FORMAT

CALCULATE/DECRYPT

```
Секретний ключ (A): d = 690259980763736590146663406102301580625927949276689911174071260297395525975214912478222921097317276881428047039048480285106717509521343515096542355832321, p = 798326860046750765938
74812880691458619502912940526309855765618188017812515633, q = 96409201806619232347135046869751982040354673123114740141763295482864691767069
Відкритий ключ (A): n = 7696605478309652376964098945925659090266119504759534958292757387132680185549340818574808995070389260420832346229687900536333120861851011456696642957089677, e = 65537

Зашифрований ключ: 121330556307052100143428077837329743972352169018191811584613026060128507300878335560263739624616611994784252982114416513257868679604017210677247813719472

Зашифрований підпис: 223945051490808377063637717074611725184578842137535215425489139291737877269059770556638704635247819788307270228410644418214167234493797068330520616165530

Абонент В отримує ключ:
Секретний ключ (B): d = 1929260584371559826960047674402286167598847381482093883596191731884739996081851933549257859134912023415871842768621647760191459568946367893045927477548345, p = 939921187414169731757
59414108215862727844775054606827722002566972239540240659, q = 95498890377253968269070023065440624265466038796960836489077358910955878130379
Відкритий ключ (B): n = 8976143044012417746661979585212454108045269130432059268013887372748133261632122022141992438123379323161864339401032564630383420977432584441469660638879761, e = 65537

Розшифрований ключ: 36002
Чи вірний підпис?: True
```

Бачимо, що все коректно працює.

Висновок: під час комп'ютерного практикуму ознайомились з механікою шифрування RSA. На основі цього ми реалізували відповідні функції (шифрування, дешифрування, створення цифрового підпису, його перевірку, надсилання та отримання ключа). Також набуті навички були протестовані на сторонньому ресурсі