

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий фізико-технічний інститут
Кафедра інформаційної безпеки

Дисципліна «Криптографія»

Комп'ютерний практикум

Робота №4

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

Виконали: студенти гр. ФБ-12 Головка М. С. і Марчук І. С.

Київ – 2023

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 $pq \leq p_1q_1$; p і q – прості числа для побудови ключів абонента A, p_1 і q_1 – абонента B.
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 A і B – 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 A і B. 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.
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:

1. Написали функцію генерації випадкового простого числа `generate_random_number(bits)`, як аргумент вона приймає кількість бітів потрібного числа. В ній на початку перевіряється, чи число парне чи непарне (очевидно, що якщо число парне, воно ділиться на 2 і тому воно складене, не підходить), а потім число перевіряється на простоту за допомогою імовірнісного тесту Міллера-Рабіна, який реалізован у функції `miller_rabin(p, iteration=500)`. Також ми використовували вже написану нами в минулій лабі функцію пошуку НСД `gcd(a, b)`. Тест Міллера-Рабіна реалізован за допомогою теоретичних відомостей, наданих в методичних вказівках. Для

генерації випадкового числа використовували функцію з бібліотеки `random` – `random.randint()`.

```
PS C:\Users\Igorm> & C:/Python311/python.exe "d:/uni year 3/crypto labs/crypto-23-24/cp4/fb-12_Marchuk_Holovko/c
Random prime: 99793130836274818984114343279722358433646308519748858818280672990649889162107
Random prime: 111279295687536249468694572062661825425404843225550211379827899464704995667871
Random prime: 113574272658613684647363775534590996767126517702612801589785738183624520712849
Random prime: 78178374225192832808254403414253929254636435230298336345672208431767410557199
Random prime: 101305533590413818949521262555888678637009444404777849229539523503918153439593
Random prime: 7169540904103356985450950914396789105455565737390400369144364984854577307340804
```

2. Реалізували генерацію пар чисел p, q і p_1, q_1 у функції `generate_pairs()`, в якій використовується функція `generate_random_number(256)`. При цьому була збережена умова, що $pq \leq p_1q_1$. Таким чином отримали p, q – прості числа для побудови ключів абонента А, і p_1, q_1 – для абонента В.

```
PS C:\Users\User> & C:/Users/User/AppData/Local/Microsoft/WindowsApps/python3.10.exe "c:/Users/User/Desktop/university/3 year/crypto labs done/lab4/lab4.py"
A pair: 6167560015302584365358639183430273206002707291265056135349963004051504431139 957146294252567375060807116962283639343630374125429697198644574781396
00910919
B pair: 7894574975292380591030632779034924985864479333955764860550620128580414216447 842713275465276749333522502540465453895230456053630032240167281509303
32148561
PS C:\Users\User> & C:/Users/User/AppData/Local/Microsoft/WindowsApps/python3.10.exe "c:/Users/User/Desktop/university/3 year/crypto labs done/lab4/lab4.py"
A pair: 67051468070797828695563450257000444633028819243574745920878109561896939109779 879128921390103127061279104812087720492150372964572227390690661212946
25747319
B pair: 70150219341642018304838369849258537820845482216895685369463887932787500794129 988620377442146841724705853507330827621792728196162602276684950066001
64220197
PS C:\Users\User>
```

Також додали умову, що якщо число не виявилось простим, треба його просто вивести у консоль з приміткою ‘this value didn’t work out’:

```
this value didnt work out: 94741087424504768834294499423680258822209375046722303090649538107874394222653
this value didnt work out: 68563080099845297215804586043934194146184705020262851247620305448122110127109
this value didnt work out: 106653961520710734312880972348942849034236656576115301088306906910275574942605
this value didnt work out: 110907807170583270323694637523114330578254466927734170471790553009025449941243
this value didnt work out: 96557153900436189667950847776457149871190155021240139855240832113543889219141
this value didnt work out: 112728742431750506146027008890329277861517239839171929326944443147536599514819
this value didnt work out: 111747641638497171903692040396891200513962832382728429370961028839903127617967
this value didnt work out: 111082701374959542410662939174595396532440452383663000595244355741962117358453
this value didnt work out: 65497165385841689925802300565183095753184569153351023219785749125706131527759
this value didnt work out: 93153387899154705215478940899533274224215998719582972613999970972586195173683
this value didnt work out: 91375783749395959855200988599316079280634547449866474437143811783756948431627
this value didnt work out: 73659485975294719939385568550915231521611676482100546269416588165271883929727
this value didnt work out: 100770751397906434708285355197015881130555363077469081817333963190341835255147
this value didnt work out: 92480686060126621061435239133513500191274140088079425756520920790311164802727
this worked out: 79324213214389734014767016796745846404609784138764442487997458764227366586979
```

3. Генерація ключів RSA виконується за наступним алгоритмом:

Абонент А генерує два великих випадкових простих числа p і q , обчислює добуток $n = pq$ і функцію Ойлера $\phi(n) = (p-1)(q-1)$. Потім А обирає випадкове число e , $2 \leq e \leq \phi(n)-1$ таке, що $\gcd(e, \phi(n)) = 1$, і знаходить для e обернений за $\text{mod } \phi(n)$ елемент d : $ed \equiv 1 \pmod{\phi(n)}$.

Зауваження: в сучасних криптографічних системах майже завжди обирається $e = 2^{16} + 1$.

Пара чисел n і e – це відкритий ключ А, а d – секретний (чи особистий) ключ А. При цьому числа p і q абонент А також тримає в таємниці. Не знаючи p і q , не можна обчислити $\phi(n)$ і знайти секретний ключ d по відкритому e і n .

За цим алгоритмом ми створили функцію `generate_rsa_keys(p, q, p1, q1)`, яка приймає як аргументи згенеровані раніше числа p, q для абонента А і p_1, q_1 для абонента В. У функції

шукається число p , функція Ойлера від цього числа p , генерується число e , і знаходиться обернений до e елемент d . Ці операції проводяться і над числами для абонента А, і над числами для абонента В.

```
PS C:\Users\User> & C:\Users\User\AppData\Local\Microsoft\WindowsApps\python3.10.exe "c:\Users\User\Desktop\university\3 year\crypto labs done\lab4\lab4.py"

Public Key A: (755687225110629396229516579122603283583428025479921004197655481319258243157738704365724121683693988929315555542623881273445062754867152709565
8274737452853, 368597658147519866863978586494482687377713711508800151269426489605001064411779583807616376904709840870233029870306855199536880699748114353425
9621743358349)
Private Key A: (27309047972074534318939140584203373110892451001577000743783022388835286693173198629890354987638869541119501829332015514677095068437028218299
62054890346949, 68758360946344910424790636964447132729955959602408897916828186774725114171333, 1099047759006827483568087080416607469618821442104821180907537
48952837746315441)
Public Key B: (939583039299238229081679599534982512716297423669626032260303376218835946804334278553172886483609020451720809207329757821875698904368965280619
5067360658187, 307031999986917377623756845590820417217503425696433555274215232253405851944847252170514474178326920273943422798844632614748838781079984170798
9777059058221)
Private Key B: (48923244481209295915689258446297430030089498035528449614074271230986320516823487937606180878556380522853276912807768937806588142374073547180
43336086228261, 105606916433658546473217083181628030697280767461579182841252061728023887860631, 889698393844760229849846293320329053017086843358097710620772
49588984000981677)
PS C:\Users\User>
```

4. Шифрування і розшифрування повідомлення реалізовано відповідно у функціях `encrypt(message, public key)` і `decrypt(ciphertext, private key)`. Для зашифрування повідомлення використовується відкритий ключ e :

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

Для розшифрування використовується секретний ключ d :

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

У функції шифрування, якщо повідомлення $M \geq n$, то виводиться помилка «Message is too large for the given public key».

```
Public Key A: (5910564277889683538187118688550561429043036821543985842236711224220799497836593218896918128044963135515879141017737527049231207550624741989128843
94541211, 185460676760871515095703307379152721887514148987006195927766128095002494919356755600091154064640359970443246273313563333878574802784230741272320590640
809)
Private Key A: (174572638361443802415680444457451068921790134141134922549188149396791387245774156701716843237979295693632178844555964809467254075539472663432183
585906681, 93352986957072568358093947075466243266182548916719040354246889951105460633003, 63314142059620243956939361793680780647347446789908717914373233881795370
006737)
Public Key B: (62455574603490576769140319512410888034280346256905860486836024477448126462776423117584985540497087420691875421133046345703996222077064283041358238
66345431, 328243099307094163764889873075753132988731826853681002017748966059205758005300272235320156702266496233072943080454452038078508698251939716613876672741
377)
Private Key B: (5300457047023178161195415176717552536723906276253041437748268469802614232077005623267399465677302618254019035369730649838870294090602731758160887
011640425, 66305316323948850853330658639777342131434969234724733518804200458049698195867, 94193916967910182613193249607917499530282067928291086213525532890385934
949493)
Message to encrypt: 123123123123
Ciphertext A: 24682799755559618241507855403101899924254582754205121452150258314264978771659925862788909454289659097591896525933256797288581114826873882877259730
1519079
d: 174572638361443802415680444457451068921790134141134922549188149396791387245774156701716843237979295693632178844555964809467254075539472663432183585906681
Decrypted Message A: 123123123123
```

```
Public Key B: (10508101063015903600322318656474678261586955809516338160128576302766640769967176385062531301267847233034390721002792546772750248309735800149201066
469972753, 55657774437204071214590465364733411835295359635206769900631102467058348267511882456488767911997941532117509994718522623092353429076588400768321050733
4127)
Private Key B: (3594371869505856170669356175108205526825126063841744409907060577368932537336220394082664499325876741016964841259114254590987751051596187434072546
263102063, 96595699550283713669793348210741830334903360804989993304309479730113898247301, 10878435698419288471508588648321826109416487099721628699509616741347783
2525853)
Message to encrypt: 31993999716875508947060859574031369639174112355942593959750532598040069986915766513862323484717296667498928862597311772537533267606068244582
842493885485094
Traceback (most recent call last):
  File "c:\Users\User\Desktop\university\3 year\crypto labs done\lab4\lab4.py", line 132, in <module>
    ciphertext_A = encrypt(message, public_key_A)
  File "c:\Users\User\Desktop\university\3 year\crypto labs done\lab4\lab4.py", line 118, in encrypt
    raise ValueError("Message is too large for the given public key")
ValueError: Message is too large for the given public key
```

Також треба реалізувати функцію електронного підпису – в нас це функція `sign(m, private_key)`, яка повертає підпис S :

Цифровий підпис у системі RSA. Щоб підписати відкрите повідомлення M , абонент A зашифрує його на своєму секретному ключі $S = M^d \bmod n$ і додає S до повідомлення M . Підписане повідомлення (M, S) може передаватися як у відкритому, так і в зашифрованому виді. Перевіряється підпис за допомогою відкритого ключа: якщо виконується рівність $M = S^e \bmod n$, то повідомлення M не спотворене і підпис вірний. На

Перевіряється підпис за допомогою функції `verify(s, m, public_key)`: якщо підпис вірний, функція повертає `True`:

```
print("verification result:", verify(sign(message_to_encrypt, private_key_A), message_to_encrypt, public_key_A))
```

```
Decrypted Message A: 123123123123
verification result: True
PS C:\Users\Igor> □
```

Також перевірили цю функцію на ключах абонента B:

```
138 print("Decrypted Message A:", decrypted_message_A)
139 sign_val=sign(message_to_encrypt,private_key_A)
140 print("verification result A:", verify(sign_val,message_to_encrypt,public_key_A))
141 sign_val_b=sign(message_to_encrypt,private_key_B)
142 print("verification result B:", verify(sign_val_b,message_to_encrypt,public_key_B))
```

```
Decrypted Message A: 123123123123
verification result A: True
verification result B: True
PS C:\Users\Igor> □
```

5. Протокол конфіденційного розсилання ключів по відкритих каналах зв'язку з підтвердженням справжності відправника реалізований у функціях `send_key()`, яка приймає як аргументи відкриті ключі A і B і секретний ключ A , а також `recv_key()`, яка приймає значення k_1, S_1 (які були створені у попередній функції), відкриті ключі A і B і секретний ключ B . Створення протоколу здійснюється за алгоритмом, наданому у теоретичних відомостях:

Нехай абонент A має відкритий ключ (e, n) , секретний d і відкритий ключ (e_1, n_1) абонента B , якому він має намір передати секретне значення $0 < k < n$ для створення спільного ключа симетричної чи асиметричної криптосистеми. Обов'язковою умовою роботи протоколу є $n_1 \geq n$; якщо $n_1 < n$, абонент A повинен згенерувати собі іншу пару ключів.

Абонент A формує повідомлення (k_1, S_1) і відправляє його B , де

$$k_1 = k^{e_1} \bmod n_1, \quad S_1 = S^{e_1} \bmod n_1, \quad S = k^d \bmod n.$$

Абонент B за допомогою свого секретного ключа d_1 знаходить (конфіденційність):

$$k = k_1^{d_1} \bmod n_1, \quad S = S_1^{d_1} \bmod n_1,$$

і за допомогою відкритого ключа e абонента A перевіряє підпис A (автентифікація):

$$k = S^e \bmod n.$$

Перевірили це на практиці (абонент A – відправник, абонент B – одержувач):

```

155
156 k_1,S_1=send_key(public_key_B,private_key_A,public_key_A)
157
158 print("perevirka pidpeesu:",recv_key(k_1,S_1,public_key_B,private_key_B,public_key_A))

```

```

verification result B: true
k value: 2746729352221021358409839168197218316727727548300783150981439147378376637841469511276948120485215447815611169783565334284670978559328796743215868532824774
perevirka pidpeesu: True
B5-C\Users\Tanya>

```

```

verification result B: true
k value: 456308174183661615640651211857772376961502006340302887602929577413539669560280198954754822896585038854296910624817036263784759712727038625728543288926606
perevirka pidpeesu: True
B5-C\Users\Tanya>

```

Висновки: у ході лабораторної роботи було розглянуто методи генерації параметрів для асиметричних криптосистем, було побудовано криптосистему RSA та алгоритм електронного підпису. Ми навчилися генерувати RSA ключі і використовувати їх для шифрування і розшифрування тексту, а також розробили протокол конфіденційного розсилання ключів по відкритих каналах зв'язку з підтвердженням справжності відправника. RSA є основним компонентом цифрових сертифікатів, які використовуються для автентифікації особи, веб-сайтів і організацій в Інтернеті, тому важливо розуміти, як цей алгоритм працює на практиці.