

Міністерство освіти і науки України

Національний Технічний Університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Фізико-технічний інститут

Криптографія

Комп’ютерний практикум №4

**Виконали:**

Студени групи ФБ-81

Аль Біні Ейман  
Кіндерись Роман

**Перевірив:**

Чорний О.М.

page1image58749888

Київ - 2020

**Мета комп’ютерного практикуму**

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.

**Постановка задачі**

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 p1 , q1 довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб pq ≤p1q1 ; p і q – прості числа для побудови ключів абонента А, p1 і q1 – 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 А і B – тобто, створити та зберегти для подальшого використання відкриті ключі (e, n) , (e , n ) та секретні d і d .
4. Написати програму шифрування, розшифрування і створення повідомлення з цифровим підписом для абонентів А 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 А и B, перевірити правильність розшифрування. Скласти для А 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 .

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

**Хід роботи**1. Використовуючи бібліотечну функцію random.randint() генеруемо випадкові числа з вказаного діапазону, діапазон визначається необхідною довжиною у бінарній системі, маємо наступне:

def miller\_rabin(p):

k = 100

s = 0

d = p - 1

while d % 2 == 0:

s += 1

d = d // 2

for i in range(k):

x = random.randint(2, p - 1)

if math.gcd(x, p) > 1:

return False

else:

x\_r = x2d\_mod\_p(x, d, p)

if x\_r == 1 or x\_r == p - 1:

continue

for r in range(1, s):

x\_r = (x\_r \*\* 2) % p

if x\_r == p - 1:

break

elif x\_r == 1:

return False

return False

return True

def prime(length):

n0 = 2 \*\* length

n1 = 2 \*\* (length + 1) - 1

# print("prime from ", n0, " to ", n1)

p = random.randint(n0, n1)

if p % 2 == 0:

p += 1

while not miller\_rabin(p):

p += 2

return p

2. Генеруємо 2 пари ключів і перевіряємо чи виконується умова pq ≤ p1q1 (або те саме що n ≤ n1)

def inverse\_a\_mod\_n(a, n):

if a == 0:

return 0

if n == 0:

return a, 1, 0

d, x, y = inverse\_a\_mod\_n(n, a % n)

return d, y, x - (a // n) \* y

def generate\_key\_pair(key\_len):

mIn = 2 \*\* key\_len

mAx = 2 \*\* (key\_len + 1) - 1

p = prime(key\_len // 2)

q = prime(key\_len // 2)

while mIn <= p \* q <= mAx:

q = prime(key\_len // 2)

n = p \* q

fi\_n = (p - 1) \* (q - 1)

e = 2 \*\* 16 + 1

d = inverse\_a\_mod\_n(e, fi\_n)[1] % fi\_n

open\_key = [n, e]

secret\_key = [d, p, q]

return [open\_key, secret\_key]  
  
[[n, e], [d, p, q]] = rsa.generate\_key\_pair(512) # keys for A

[[n1, e1], [d1, p1, q1]] = rsa.generate\_key\_pair(512) # keys for b

while n1 < n:

[[n1, e1], [d1, p1, q1]] = rsa.generate\_key\_pair(512)

3. Функція генерації ключових пар наведена в пункті 2

4 - 5. Реалізація протоколу RSA наведена в файлі rsa\_protocol.py приклад виводц файлу:

Keys of A

n is: 2a1208ce36358d6af83e2258ce9269158699cc9d937f1082dc89b7b7b581d3733887b435e3287c2c3709e145e40f90dc8ddbb2d6f51463174d2c1ca4ff8ff8b2d

e is: 10001

fi(n) is: 2a1208ce36358d6af83e2258ce9269158699cc9d937f1082dc89b7b7b581d3730499a54b0c8f628aded8700f86c5eab0a22c78a9ad2f2a1f1589b31af5df02b7c

d is: 963e51b41f64a0e6a425420465b5fa1325df9d1e22e265d4f245e215f1e0f3848fa926e61eb4f2f77f60f6d27c3dd5277a03d7e81ed59157f153128ff603b971

p is: 18f0793a22c3ccce5143f9f6d5ab73c7ad7bf1217710976d106fcf686a9d13567

q is: 1afd95b0b3d54cd306ed773f879e32643e33490bd0d4a18b27329a219f13e2a4b

Keys of B

n1 is: 2e8263250f00b19173355328902a6dc5952a2881914e7acce623c4f65c5622c925b541226e95026d4a3b421a8ddc36960edff2ac67067f65d9ec248f05aa1e80d

e1 is: 10001

fi(n1) is: 2e8263250f00b19173355328902a6dc5952a2881914e7acce623c4f65c5622c8eeb26ec819a0e09f600f1e7af2d5aab942192388caf80dbe77d91c67c34562f7c

d1 is: 24159794a60a34ba93b738b8ccf98e4770cecfb48f2763915bb13edca19156d5ac9495ecfc2d5c796acef0281605feba53752ba3b34e3763a29ab9362c43c6605

p is: 1f0705dfb7c27572bc1e6599ec78c68a831798e2287b5f299f4ff9bb4c6048d3b

q is: 17fbcc7a9d31ac5b2e0dbe05ae8dc55249af36417393127dc2c30e6bf60472b57

local test for encryption and decryption

msg is: 62078

Encrypted msg is: 26262186595389801518923930040399268292079147299959584256775186242769539759615059515502324925423526801047652938901375178560000681771780139750201440823971654

Decrypted msg is: 62078

--------------------------------------- local check for signing and verification ---------------------------------------

msg is: 39087

encrypted msg is:2d3a61c89dd8df8485763a50a218f884519699080a539981019d83dccda708300c99ece18a37b097434f28546c86196cbb503a45893e56ad32694a77a61ffdbd0

signature is: 11d515f0cc84d1f1be6484672c5ed7c9fe598c382bf52c8a310a865c6b051e9bd13b7c4d9fc1d3565f1e22ac85e91e2129983b5f388ff4d6c68e8345316b7f0d7

Signature is valid

encrypted msg is: 39087

--------------------------------------- user-server check for signing and verification ---------------------------------------

Server public key:{'modulus': '557CC26331C73767292F5442AE85D846E0445831682AAC51334DC8EA7424A47363EEC993DFC06E403BD4A53C62C2CCF5063FF2AA1A01921CCAF73ADE2B9042E35', 'publicExponent': '10001'}

msg is: 62159

msg server got after verification: 62159

verification result: True

Параметри системи:  
а) Локальний тест:

|  |  |  |
| --- | --- | --- |
|  | А | В |
| n | 2a1208ce36358d6af83e2258ce9269158699cc9d937f1082dc89b7b7b581d3733887b435e3287c2c3709e145e40f90dc8ddbb2d6f51463174d2c1ca4ff8ff8b2d | 2e8263250f00b19173355328902a6dc5952a2881914e7acce623c4f65c5622c925b541226e95026d4a3b421a8ddc36960edff2ac67067f65d9ec248f05aa1e80d |
| e | 10001 | 10001 |
| ϕ(n) | 2a1208ce36358d6af83e2258ce9269158699cc9d937f1082dc89b7b7b581d3730499a54b0c8f628aded8700f86c5eab0a22c78a9ad2f2a1f1589b31af5df02b7c | 2e8263250f00b19173355328902a6dc5952a2881914e7acce623c4f65c5622c8eeb26ec819a0e09f600f1e7af2d5aab942192388caf80dbe77d91c67c34562f7c |
| d | 963e51b41f64a0e6a425420465b5fa1325df9d1e22e265d4f245e215f1e0f3848fa926e61eb4f2f77f60f6d27c3dd5277a03d7e81ed59157f153128ff603b971 | 24159794a60a34ba93b738b8ccf98e4770cecfb48f2763915bb13edca19156d5ac9495ecfc2d5c796acef0281605feba53752ba3b34e3763a29ab9362c43c6605 |
| p | 18f0793a22c3ccce5143f9f6d5ab73c7ad7bf1217710976d106fcf686a9d13567 | 1f0705dfb7c27572bc1e6599ec78c68a831798e2287b5f299f4ff9bb4c6048d3b |
| q | 1afd95b0b3d54cd306ed773f879e32643e33490bd0d4a18b27329a219f13e2a4b | 17fbcc7a9d31ac5b2e0dbe05ae8dc55249af36417393127dc2c30e6bf60472b57 |
| ВТ | 39087 | 39087 |
| ШТ | 2d3a61c89dd8df8485763a50a218f884519699080a539981019d83dccda708300c99ece18a37b097434f28546c86196cbb503a45893e56ad32694a77a61ffdbd0 | 2d3a61c89dd8df8485763a50a218f884519699080a539981019d83dccda708300c99ece18a37b097434f28546c86196cbb503a45893e56ad32694a77a61ffdbd0 |
| Статус верифікації | valid | valid |

б) тест з сервером в якості В

|  |  |  |
| --- | --- | --- |
|  | А | В (сервер) |
| n | 2a1208ce36358d6af83e2258ce9269158699cc9d937f1082dc89b7b7b581d3733887b435e3287c2c3709e145e40f90dc8ddbb2d6f51463174d2c1ca4ff8ff8b2d | 557CC26331C73767292F5442AE85D846E0445831682AAC51334DC8EA7424A47363EEC993DFC06E403BD4A53C62C2CCF5063FF2AA1A01921CCAF73ADE2B9042E35 |
| e | 10001 | 10001 |
| ϕ(n) | 2a1208ce36358d6af83e2258ce9269158699cc9d937f1082dc89b7b7b581d3730499a54b0c8f628aded8700f86c5eab0a22c78a9ad2f2a1f1589b31af5df02b7c | невідомо |
| d | 963e51b41f64a0e6a425420465b5fa1325df9d1e22e265d4f245e215f1e0f3848fa926e61eb4f2f77f60f6d27c3dd5277a03d7e81ed59157f153128ff603b971 | Невідомо |
| p | 18f0793a22c3ccce5143f9f6d5ab73c7ad7bf1217710976d106fcf686a9d13567 | Невідомо |
| q | 1afd95b0b3d54cd306ed773f879e32643e33490bd0d4a18b27329a219f13e2a4b | невідомо |
| ВТ | 62159 | 62159 |
| Статус верифікації | valid | valid |

Висновок:

Вході виконання лабораторної роботи ми здобули більше глибоке розуміння криптосистеми rsa та техніки її використаня як системи шифрування та механізму цифрового підпису.