



МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. ІГОРЯ
СІКОРСЬКОГО”

КРИПТОГРАФІЯ
КОМП'ЮТЕРНИЙ ПРАКТИКУМ №4

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

Виконали:
Студенти групи ФБ-22
Орлов Антон, Ялбуган Федір
(бригада 7)

КИЇВ 2024

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 $1 < p, q$ 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, p, q) і (e, n) 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 $0 < k < n$.

Хід роботи

1. , 2. Функція пошуку випадкового простого числа з заданого інтервалу та генерація пар p , q :

```
import random

def is_prime(n, k=10):
    r, d = 0, n - 1
    while d % 2 == 0:
        r += 1
        d //= 2
    for _ in range(k):
        a = random.randint(2, n - 2)
        x = pow(a, d, n)
        if x == 1 or x == n - 1:
            continue
        for _ in range(r - 1):
            x = pow(x, 2, n)
            if x == n - 1:
                break
        else:
            return False
    return True

def find_p_q(print_result=True):
    lower_bound = 2**255
    upper_bound = 2**256 - 1
    primes = []
    while len(primes) < 2:
        candidate = random.randint(lower_bound, upper_bound)
        small_primes = [2, 3, 5, 7]
        if any(candidate % p == 0 for p in small_primes):
            if print_result:
                print(f"{candidate} - test failed")
            continue

        if is_prime(candidate):
            if print_result:
                print(f"{candidate} - test passed")
            primes.append(candidate)
        else:
            if print_result:
                print(f"{candidate} - test failed")
    num1, num2 = primes
    if print_result:
        print("\nNumbers, that passed the test:")
        print(f"num1 = {num1}")
        print(f"num2 = {num2}\n\n")
    return num1, num2

p, q = find_p_q()
p1, q1 = find_p_q()

pq = p * q
p1q1 = p1 * q1
if pq > p1q1:
    p, q, p1, q1 = p1, q1, p, q
```

Випадково генеруються за допомогою `random.randint` числа з усіх можливих довжиною 256 біт, потім кожне перевіряється на простоту тестом Міллера-Рабіна, до тих пір, поки тест не пройдуть 2 числа – p , q . Далі p , q та $p1$, $q1$ міняються місцями між собою, якщо pq більше, ніж $p1q1$.

```
99800309073320367197659480060019523366714460781617948038901835886269529020421 - test failed
114502234762760531822474429819360868262381612807155061780908165904528845777115 - test failed
94902089832782686669829303016481572534294588635617399328611688246474127740967 - test failed
68656667112681987441863941780666454780890750856575808587980377027032680713384 - test failed
101041083568336137794916905120268090139475368780809176160976843823263837399771 - test failed
114385468156576160794849508131159213887850041436772956075857947681549891342467 - test passed

Numbers, that passed the test:
num1 = 67845620259981189366585723195734144770483020503469685846483122172578848654491
num2 = 114385468156576160794849508131159213887850041436772956075857947681549891342467
```

```

99190670506797102802518681477847222368752664286071213739545123485735411189665 - test failed
76836052010635465275568984069773084302713344388242797615523607243057835868570 - test failed
68889201123552499350840167661576357288484198987162776483914173960320670940978 - test failed
99800309073320367197659480060019523366714460781617948038901835886269529020421 - test failed
114502234762760531822474429819360868262381612807155061780908165904528845777115 - test failed
94902089832782686669829303016481572534294588635617399328611688246474127740967 - test failed
68656667112681987441863941780666454780890750856575808587980377027032680713384 - test failed
101041083568336137794916905120268090139475368780809176160976843823263837399771 - test failed
114385468156576160794849508131159213887850041436772956075857947681549891342467 - test passed

Numbers, that passed the test:
num1 = 67845620259981189366585723195734144770483020503469685846483122172578848654491
num2 = 114385468156576160794849508131159213887850041436772956075857947681549891342467

```

Final primes:

```

p = 67845620259981189366585723195734144770483020503469685846483122172578848654491
q = 114385468156576160794849508131159213887850041436772956075857947681549891342467
p1 = 91063306518095577083619788945055598688778957653014223597442823931269532793079
q1 = 107146066373338155100823897361216599784275260701096629187947543255092172013131

```

3. Функція генерації ключових пар для RSA:

```

from math import gcd
from sympy import mod_inverse

def generate_keys(p, q):
    n = p * q
    phi_n = (p - 1) * (q - 1)
    e = random.randint(2, phi_n - 1)
    while gcd(e, phi_n) != 1:
        e = random.randint(2, phi_n - 1)
    d = mod_inverse(e, phi_n)
    return (n, e), d

public_key, private_key = generate_keys(p, q)
public_key1, private_key1 = generate_keys(p1, q1)

```

```

Abonent A (p, q): 67845620259981189366585723195734144770483020503469685846483122172578848654491, 1143854
public_key (n, e): (776055303581123676394664570561478927828065232859597471210903579041227563931973272539
private_key (d): 108117401226827187427846157248338473618849163203904993137589334517257715330785714107709

Abonent B (p1, q1): 91063306518095577083619788945055598688778957653014223597442823931269532793079, 10714
public_key (n1, e1): (9757075084363505749356247757025365322628034001619006131463005627162989720236463520
private_key (d1): 85892171509089630503539683180315825638698402429611019190220386052711188225870401882180

```

4. Функції шифрування, розшифрування, створення цифрового підпису, перевірки цифрового підпису:

```

def rsa_encrypt(message, n, e):
    message_encrypted = pow(message, e, n)
    return message_encrypted

def rsa_decrypt(message_encrypted, n_self, d_self):
    message_decrypted = pow(message_encrypted, d_self, n_self)
    return message_decrypted

def rsa_sign(message, n, e, n_self, d_self):
    auth = rsa_encrypt(message, n_self, d_self)
    auth_encrypted = rsa_encrypt(auth, n, e)
    return auth_encrypted

def rsa_verify(auth_encrypted, n, e, n_self, d_self):
    auth_decrypted = rsa_decrypt(auth_encrypted, n_self, d_self)
    auth = rsa_decrypt(auth_decrypted, n, e)
    return auth

```

5. Функції роботи протоколу конфіденційного розсилання ключів з підтвердженням справжності по відкритому каналу за допомогою алгоритму RSA.

```
def send_key(n, e, n_self, d_self):
    message, _ = find_p_q(print_result=False)
    print(f"Message to send: {message}")
    k = rsa_encrypt(message, n, e)
    print(f"Encrypted Message: {k}")
    s = rsa_sign(message, n, e, n_self, d_self)
    print(f"Encrypted Authenticator: {s}\n")
    return k, s

def receive_key(k, s, n, e, n_self, d_self):
    message = rsa_decrypt(k, n_self, d_self)
    auth = rsa_verify(s, n, e, n_self, d_self)
    if message != auth:
        print('Verification failed')
    else:
        print('Verification complete', f'\nReceived message: {message}')
```

– опис кроків протоколу конфіденційного розсилання ключів з підтвердженням справжності, чисельні значення характеристик на кожному кроці;

1) Для абонентів А та В генеруються значення p , q , p_1 , q_1 відповідно:

Abonent A (p , q):

67845620259981189366585723195734144770483020503469685846483122172578848654491,
114385468156576160794849508131159213887850041436772956075857947681549891342467

Abonent B (p_1 , q_1):

91063306518095577083619788945055598688778957653014223597442823931269532793079,
107146066373338155100823897361216599784275260701096629187947543255092172013131

2) Для абонентів А та В генеруються відкриті ключі (n, e) , (n_1, e_1) , а також секретні ключі d , d_1 :

public_key (n, e):

(77605530358112367639466457056147892782806523285959747121090357904122756393197327
25395241279998067013415608548284739965428235452797732480427398272638569297,
288976268753249784722168377786300411425816860014700936549357176820728993513147548
401024822637191612978317565358257499021103424642680778601983413649225497)

private_key (d):

108117401226827187427846157248338473618849163203904993137589334517257715330785714
1077093916564451286205744507542764930426365333471345641940971297802559313

public_key (n_1, e_1):

(97570750843635057493562477570253653226280340016190061314630056271629897202364635
20094746751276999263651590229322933035343424362407271013550248912693920349,
597017611955147196327013933634884076639787326787028169447917036925025989637024664
2860219899961384450905972793313180893450545551142040600778482153146671079)

private_key (d_1):

858921715090896305035396831803158256386984024296110191902203860527111882258704018
8218001825979027658665744072033202680061070812065368870266023325313884219

3) Абонент А шифрує своє повідомлення відкритим ключем Абонента В (n_1, e_1):

On abonent A side

Message to send:

77633844615474466032948802040045145852619664267906824550427988761543202335487

Encrypted Message:

704730667200664429335114823255449471209002946142622145494445208415971725035788426

5012630739104590003230293657913663769661073293451875152865992992161114416

4) Абонент А підписує повідомлення своїм секретним ключем (d), а потім, зашифровує відкритим ключем Абонента В (n_1, e_1), щоб зробити підпис:

Encrypted Authenticator:

450336391072730911228408486453072624475794707115289805446426317024799501301989400

3158984196729195557158377142875835908051964547783489308813704135950811631

5) Абонент А передає зашифроване повідомлення (k) та цифровий підпис (s) Абоненту В

6) Абонент В розшифровує повідомлення своїм секретним ключем (d)

7) Абонент В розшифровує підпис спочатку своїм секретним ключем (d_1), а потім відкритим ключем Абонента А (n, e)

8) Абонент В верифікує підпис (порівнює підпис з розшифрованим повідомленням):

On abonent B side

Verification complete

Received message:

77633844615474466032948802040045145852619664267906824550427988761543202335487

- Перевірка шляхом взаємодії із тестовим середовищем:

Encryption

Clear

Modulus

BA4B96ED92E629AF87ABAA351CF256B00FC085B5E8277A8FD62D216C72DF62D2D52E702807127C81696E4

Public exponent

1B4705BECE4C8334D000A6CFDD726E563A280CBF813A2A681C8EC9BE1762CDE4672BD639A05D0A339

Message

911

Bytes

Encrypt

Ciphertext

7D2A873C75FDCBFC7944737C7D3902E0F31839E1B9ABC8691D2BE9CBD6001E9F101F3687926C2FE69901


```
text = rsa_decrypt(65554819425116490780520725577155513449661578222622466698356055161201127
97570750843635057493562477570253653226280340016190061314630056271629897
42264119732637390676390064275577295303501843811547316483527347523174365

text = hex(text)[2:]
print(text)
```

✓ 0.0s

911

Get server key



Key size

256

Get key

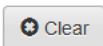
Modulus

AAC6F74617D16CC663E828060AAA0198366400E4D265FCC26590B783EB0A8BFD

Public exponent

10001

Send key



Modulus

BA4B96ED92E629AF87ABAA351CF256B00FC085B5E8277A8FD62D216C72DF62D2D52E702807127C81696E4

Public exponent

1B4705BECEF4C8334D000A6CFDD726E563A280CBF813A2A681C8EC9BE1762CDDE4672BD639A05D0A339

Send

Key

28ED2914D8304E2F11D7D7E9DF735224D5B32E2104FF4F117204565FB396012D27FA0E78DFC1A25A47A02A

Signature

25C3163E46214629C4D234EADA137A41DC9A34EE48D822375EBF1B73FA9C66A639EC14DCFE554CF82DFC

```
k = 21434899183770238452156636638909541531576989927380236122442265172512487997725273479895539400901457
s = 19777594639996838152715023867753957247772504037662095860877261871446341379746220677799474842362076
n = 77244726599033665206644363132815992345328773232384729189197344850445857360893
e = 65537
n_self = 975707508436350574935624775702536532262803400161900613146300562716298972023646352009474675127
d_self = 422641197326373906763900642755772953035018438115473164835273475231743653570234422753560756599
```

```
text = receive_key(k, s, n, e, n_self, d_self)
```

✓ 0.0s

Verification complete

Received message: 15874756307178248791

- **Висновки:** ми ознайомились з тестами перевірки чисел на простоту і методами генерації ключів для асиметричної криптосистеми типу RSA. Практично ознайомились з системою захисту інформації на основі криптосхеми RSA, організація з використанням цієї системи засекреченого зв'язку й електронного підпису, вивчення протоколу розсилання ключів.