

Міністерство освіти і науки України

Національний технічний університет України
"Київський політехнічний інститут імені Ігоря
Сікорського"

Фізико-технічний інститут

Криптографія

Лабораторна робота №4

Виконали студенти групи ФБ-13

Дмитрів Анастасія

Лагно Костянтин

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

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 А, p_1 і 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) , (e_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 $0 < k < n$.

Хід роботи:

1. Написали функцію пошуку випадкового простого числа з заданого інтервалу або заданої довжини, використовуючи датчик випадкових чисел та тест перевірки на простоту. В якості теста перевірки на простоту, вибрали тест Міллера-Рабіна. Оскільки даний тест використовує пошук НСД, з попередньої лабораторної взяли функції `gcd` та `inverse_mod`.

Код функцій:

```
def gcd(a, b):  
    while b:  
        a, b = b, a % b  
    return a
```

```

#a^-1
def inverse_mod(a, mod, v = None):
    if v is None:
        v = [0, 1]
    if a == 0 or gcd(a, mod) != 1:
        return None
    else:
        d = mod % a
        q = mod // a
        v.append((v[len(v) - 2] - q * v[len(v) - 1]))
        if d != 0:
            mod, a = a, d
            return inverse_mod(a, mod, v)
        else:
            return v[len(v) - 2]

def miller_rabin(n):
    k = 100
    if n <= 1:
        return False
    if n <= 3:
        return True
    # Розклад  $n - 1 = 2^s * d$ , де d непарне
    s, d = 0, n - 1
    while d % 2 == 0:
        s += 1
        d //= 2

    for _ in range(k):
        x = random.randint(1, n)
        if gcd(n, x) != 1:
            return False
        xd = pow(x, d, n)
        if xd == 1 or xd == n - 1:
            return True
        for i in range(0, s-1):
            w = (2**i)*d
            if pow(x, w, n) == n-1:
                return True
    return False

```

Перевірка роботи буде в наступному пункті, оскільки вони пов'язані.

2. З допомогою даних функцій, а також функцій `generate_random_prime` і `generate_prime_pair` згенерували дві пари чисел довжиною 256 біт. Зразу також додали перевірку, якщо пара для абонента А більше, ніж для В, то вони міняються місцями, замість того щоб заново генеруватись.

Код функцій:

```

def generate_random_prime(bits):
    while True:
        number = random.randint(2**((bits-1)), 2**bits - 1)
        if miller_rabin(number):
            return number

def generate_prime_pair(bits):
    p = generate_random_prime(bits)
    q = generate_random_prime(bits)
    p1 = generate_random_prime(bits)
    q1 = generate_random_prime(bits)

```

```

if p*q > p1*q1:
    p, p1 = p1, p
    q, q1 = q1, q

return p, q, p1, q1

```

Перевірка роботи:

Якщо число є простим, функція тесту виводить значення True, в іншому випадку – False. Функції для генерації числа та пари чисел мають виводити прості числа розміром 256 байт, якщо ж числа не прості – функція циклічно генерує нове число, поки всі не будуть простими.

```

76
77 print(miller_rabin(7))
78 print(miller_rabin(10))
79 print(generate_random_prime(256))
80 a = generate_prime_pair(256)
81 for i in a:
82     print(i)

```

Run: test

C:\Users\ACER\AppData\Local\Microsoft\WindowsApps\python3.10.exe C:\Users\ACER\Desktop\Крипта\crypto-23-24\cp4\cp4_fb-13_lahno\test.py

```

True
False
99687886005100883374575105800540547357764552758094431586412741395255177329667
89618052828621466655394939198948370595683158538781185354871518220140567870141
76674717934150772868485959508426645545773874519022897922132230813147016266713
10107156154061070760480928890804739978480586097446545345428349281225250425523
102559284284832618173039772525515807579113112782317324661599987196914922867607

```

Process finished with exit code 0

Бачимо, що коли в функцію тесту передається число 7, яке є простим, результатом є True, а при 10, яке не є простим, False.

Перевірю на окремому сайті, чи є отримані числа простими:

Miller-Rabin primality test

Integer number: 99687886005100883

Bases: ☒ Random ☐ List

Number of rounds: 100

☐ Details

CALCULATE

Can be prime
yes

[LINK](#) [SAVE](#) [WIDGET](#)

Miller–Rabin primality test

Integer number
89618052828621466

Bases
☒ Random ☐ List

Number of rounds
100

☐ Details

CALCULATE

Can be prime
yes

[LINK](#)
[SAVE](#)
[WIDGET](#)

Miller–Rabin primality test

Integer number
76674717934150776

Bases
☒ Random ☐ List

Number of rounds
100

☐ Details

CALCULATE

Can be prime
yes

[LINK](#)
[SAVE](#)
[WIDGET](#)

Miller–Rabin primality test

Integer number
10107156154061076

Bases
☒ Random ☐ List

Number of rounds
100

☐ Details

CALCULATE

Can be prime
yes

[LINK](#)
[SAVE](#)
[WIDGET](#)

Miller–Rabin primality test

Integer number
10255928428483261

Bases
☒ Random ☐ List

Number of rounds
100

☐ Details

CALCULATE

Can be prime
yes

[LINK](#)
[SAVE](#)
[WIDGET](#)

І окремим скриптом перевірю числа на розмір:

```

lp x
C:\Users\ACER\AppData\Local\Microsoft\WindowsApps\python3.10.exe C:\Users\ACER\Downloads\lp.py
Розмір числа 99687886005100883374575105800540547357764552758094431586412741395255177329667 в бітах: 256 bit
Розмір числа 89618052828621466655394939198948370595683158538781185354871518220140567870141 в бітах: 256 bit
Розмір числа 76674717934150772868485959508426645545773874519022897922132230813147016266713 в бітах: 256 bit
Розмір числа 101071561540610707604809288908047399784805860974465453454528349281225250425523 в бітах: 256 bit
Розмір числа 102559284284832618173039772525515807579113112782317324661599987196914922867607 в бітах: 256 bit

Process finished with exit code 0

```

Як бачимо, функції працюють коректно.

3 – 4 – 5.

Тепер переходимо до основного завдання лабораторної.

Код основних функцій:

```

def GenerateKeyPair(p, q):
    n = p*q
    f = (p-1)*(q-1) #ойлера
    e = (2**16) + 1
    if gcd(e, f) != 1:
        return false
    else:
        m = inverse_mod(e, f)

```

```

        if m < 0:
            d = m + f
        else:
            d = m
        public_k = (e, n)
        secret_k = (d, n)
        return public_k, secret_k

def Encrypt(key_p, M): #tuple
    e, n = key_p[0], key_p[1]
    C = pow(M, e, n)
    return C

def Decrypt(key_s, C):
    d, n = key_s[0], key_s[1]
    M = pow(C, d, n)
    return M

def Sign(key_s, k):
    d, n = key_s[0], key_s[1]
    S = pow(k, d, n)
    return S

def Verify(k, S, key_p):
    e, n = key_p[0], key_p[1]
    return k == pow(S, e, n)

def SendKey(key_s, key_p):
    e1, n1 = key_p[0], key_p[1]
    k = random.randint(1, 1000)
    k1 = pow(k, e1, n1)
    S = Sign(key_s, k)
    S1 = pow(S, e1, n1)
    return k1, S1, k

def ReceiveKey(key_s, k1, S1, key_p):
    d1, n1 = key_s[0], key_s[1]
    k = pow(k1, d1, n1)
    S = pow(S1, d1, n1)
    check = Verify(k, S, key_p)
    return check

```

Генеруємо дві пари ключів для кожного з абонентів:

```

Keys of A:
Public_A (e, n): (65537, 6352943233631662715237733602569637418734037551070177287042676046503044776037999363898632242269640290051729156726158461122139422194712058092282668201778689)
Secret_A (d, n): (64947616863744358441937859739103972878706763495689744454561437831408822496381434510082595693096940128648402975473604278928145812596950844761184781709273, 6352943233631662715237733602569637418734037551070177287042676046503044776037999363898632242269640290051729156726158461122139422194712058092282668201778689)

Keys of B:
Public_B (e, n): (65537, 8619914583352562105855999422665737238849494146747222836926682906749544822086328868390788657956356660290799482125604558806763046881487598429769005117990551)
Secret_B (d, n): (8105247665967595208985307664713877528828924535745698888073898323929070624717468808941437690500637136009941839919681985986608118334923313169492344741876737, 8619914583352562105855999422665737238849494146747222836926682906749544822086328868390788657956356660290799482125604558806763046881487598429769005117990551)

```

Шифрування повідомлень:

```

Encryption
Message for B:
776
Encrypted text by A for B:
936581553507222591531980607525346914036704409349240904891857587052026872569453955952779184687247567117895878050080428343897805486616384830134509733048151

Message for A:
400
Encrypted text by B for A:
3266625840892765578109633059707590961128250100494717811730465124161578580075867574949480189967431021399852126028636460489263311588414196336039842534959299

```

Розшифрування повідомлень:

```
Decryption
Decrypted text by A from B:
400
Origin B's message :
400

Decrypted text by B from A:
776
Origin A's message :
776
```

Обмін ключами (тут одразу і використовується створення цифрового підпису):

```
-----
Keys trading
A generate k1 and S1
k1: 8002012342447716377838432616740040251747738553528504307050055820462038963731831168660036380246503019400708396987225765337112772334423448573188044936928107
S1: 912032943509747625381949307163979650124102137334572277749322704806377530704787614996872134391143959993647142510158588278773940060458528076856486981885646
A sends message (k1, S1) to B -----> B recieved message
B verifies the signature
Has it been verified? True

B generate k2 and S2
k2: 1902082132203651102207208370526465834311350451537912620087314442351710848355488400570230527184686479122772942912444016813954011472320531974423596513339672
S2: 5788673539922657056451184178556240933757105090149702936331763447444491653797010063353846301515541162840014678040693082589637645511691959900088580150200084
B sends message (k2, S2) to A -----> A recieved message
A verifies the signature
Has it been verified? True
```

Тепер перевіримо роботу нашої програми з сервісом з методички. Основний момент, над яким ми довго ламали голову – в нас працювали всі функції і завдання, окрім отримання ключа сайтом. Як виявилось, на сайті потрібно при генерації ключів задавати вдвічі більше значення розміру, ніж яке ми задаємо при створенні ключів у нас, так як якщо задати на сайті 256 біт, і у нас 256, то виходить, що у сайту значення n буде 256 біт, а у нас, так як ми перемножуємо числа розміром 256 байтів, набагато більше. На роботу шифрування, дешифрування і перевірки цифрового підпису це ніяк не впливає, оскільки там допускається різниця між розмірами ключів в абонентів в обидві сторони, а от при розсиланні ключів важливо, щоб розмір n_1 у абонента, який отримує ключ, був більшим, ніж розмір n у того, хто надсилає. Додатково ще була додана перевірка, що якщо публічний ключ нашого абонента більше, ніж на сайті, то пари ключів генеруються заново, поки публічний ключ сайту не буде більшим:

```
#надсилаємо свій ключ та сигнатуру серверу
print('Sending key and S for server')

while Public_A[1] > public_server[1]:
    a = generate_prime_pair(256)
    Public_A, Secret_A = GenerateKeyPair(a[0], a[1])[0], GenerateKeyPair(a[0], a[1])[1]

print('New keys of A: ')
print('Public_A (e, n): ', '\n e: ', decimal_to_hex(Public_A[0]), '\n n: ', decimal_to_hex(Public_A[1]))
print('')
```

```

Checking using the server

Public_A:
e: 10001
n: 794c8a285eacca938af66a2434576e47be9952317a284407ffa0e797cbd98afb26ebdf3c8e5d5529ffadabe51da0de78d09f29e90958646ffb9b9b0c3d1ce01

Secret_A:
d: 6494761686374435844193785973910397287870676349568974445456143783140822496381434510082595693096940128648402975473604278928145812596950844761184781709273
Please enter server exponent: 10001
Please enter server modulus: 8000a29593c28447e7965895ce79503417a7bf844f10ee1430f995047be9a15b990e3212b285a3a3e93aa874ce1cf4e0dc3c9b9fc4aef2fb681c1a0f
Encrypted message from server: 4870e884d9c710900f208415b2fc907fb2c2824a20f0945b9930a0f0e4ef777af2b1410f4220944f72e497411bf0b24121994a00efcc000f9f7a3f8a58bc
Decrypted message by A: 123456

```


Шифрування повідомлення для сервера:

```
Message for server:
3CF
Encrypted text by A for server:
61825DF6B807003355B72E1DF88CC7DD31E05218E68751E55175F88B912547793084000D5DF2FED59BC58E0D3ED4B5A8B1221853EF5EF3FBE4FEC096B9B79E0F
```

Server Key

Encryption

Decryption

Signature

Verification

Send Key

Receive Key

Decryption

Clear

Ciphertext

61825DF6B807003355B72E1DF88CC7DD31EC

Bytes

Decrypt

Message

03CF

Створення цифрового підпису сайтом:

Server Key

Encryption

Decryption

Signature

Verification

Send Key

Receive Key

Sign

Clear

Message

12808

Bytes

Sign

Signature

76393C715A0FEECAF5A0DB817F51D9556A17611B5361ECD17F

```
Use of verify function
Message from server: 12808
Signature from server: 76393C715A0FEECAF5A0DB817F51D9556A17611B5361ECD17F
Verify? True
```

Створення цифрового підпису нами та перевірка на сайті:

```
Use of sign function
k: 1AB
SA: 749572E5ECB460C831CFA617C629C4D8E617FDA1C056F0BC1A45E187F10FA5E8AAD892CB82F4B7F53C9C651258D8B3F856D45ED899941EA02ACB58B98D2CD5CA
Modulus: 794C8A285EACCA938AF66A2434576E47BE9952317A284407FFA0E797C8D98AFB26E8DF03C8E5D5529FFADABE51DA6DE78D09F29E90958646FFB9B9B0C3D1CE01
Exponent: 10001
```

- Server Key
- Encryption
- Decryption
- Signature
- Verification
- Send Key
- Receive Key

Verify

✖ Clear

Message

1AB

Bytes ▾

Signature

749572E5ECB460C831CFA617C629C4D8E617FDA1C056F0BC1,

Modulus

794C8A285EACCA938AF66A2434576E47BE9952317A284407FF/

Public exponent

10001

Verify

Verification

true

✓

Отримання ключа від сайту:

- Server Key
- Encryption
- Decryption
- Signature
- Verification
- Send Key
- Receive Key

Send key

✖ Clear

Modulus

794C8A285EACCA938AF66A2434576E47BE9952317A284407FF/

Public exponent

10001

Send

Key

2817CA3AFB1F2714535C2A1CDBA7429C80D80FDDC97D7CFD!

Signature

2992553B58EEF1F37779009940F8B3B30B00BD9B9101F24C7F3

```
Server send key and we try to verify it
k from server: 30D230D1C70CC736389D72613D937EA3E51A0E0F51CAC54A93972620B0FCD569DAA6AB6D1126A986E00385604EDE8451E0C9A33E92766F10D61A04EE527B44DF
S from server: 76A2C8D0D182A46748AB1016CA0603AEFACB42975AB8699A4A85597895CD4E3F01D01F034BB2297AD839B3E404EF01459D2D68D9E093C1BC650BC276B4526919
A verifies the server signature
Has it been verified? True
```

Надсилання ключа на сайт:

```
Sending key and S for server
New keys of A:
Public_A (e, n):
e: 10001
n: 794C8A285EACCA938AF66A2434576E47BE9952317A284407FFA0E797CBD98AFB26EBDFD3C8E5D5529FFADABE51DA6DE78D09F29E90958646FFB9B9B0C3D1CE01

A generate k1 and S1
k1: 30D230D1C70CC736389D72613D937EA3E51A0E0F51CAC54A93972620B0FCD569DAA6AB6D1126A986E00385604EDE8451E0C9A33E92766F10D61A04EE527B44DF
S1: 76A2C8D0D182A46748AB1016CA0603AEFACB42975AB8699A4A85597895CD4E3F01D01F034BB2297AD839B3E404EF01459D2D68D9E093C1BC650BC276B4526919
Modulus: 794C8A285EACCA938AF66A2434576E47BE9952317A284407FFA0E797CBD98AFB26EBDFD3C8E5D5529FFADABE51DA6DE78D09F29E90958646FFB9B9B0C3D1CE01
Exponent: 10001
Key (for checking): F9

Process finished with exit code 0
```

Бачимо, що модуль (відкритий ключ) не змінився, отже виконується умова розсилання ключів, в іншому випадку модуль би змінився на новий.

Server
Key

Encryption

Decryption


Signature

Verification

Send
Key

Receive
Key

Receive key

 Clear

Key

30D230D1C70CC736389D72613D937EA3E51A0E0F51CAC54A93972620B0FCD569DAA6AB6D1126A986E00385604EDE8451E0C9A33E92766F10D61A04EE527B44DF

Signature

76A2C8D0D182A46748AB1016CA0603AEFACB42975AB8699A4A85597895CD4E3F01D01F034BB2297AD839B3E404EF01459D2D68D9E093C1BC650BC276B4526919

Modulus

794C8A285EACCA938AF66A2434576E47BE9952317A284407FFA0E797CBD98AFB26EBDFD3C8E5D5529FFADABE51DA6DE78D09F29E90958646FFB9B9B0C3D1CE01

Public exponent

10001


Receive

Key

F9

Verification

true



Верифікація пройдена, отже все працює правильно.

Висновки:

В ході виконання даної лабораторної роботи, ми ознайомились з криптосистемою RSA, алгоритмом цифрового підпису та з методами генерації параметрів для асиметричних криптосистем. Під час виконання роботи набули практичних навичок роботи з даними

алгоритмами, розібрали принципи та умови даної криптосистеми, а також отримали додаткові знання по алгоритму пошуку простих чисел.