

КРИПТОГРАФІЯ
КОМП'ЮТЕРНИЙ ПРАКТИКУМ №4

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

Виконали: ФБ-13 Летюка, Мірошніков

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.

Хід 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.

Реалізуємо функцію пошуку випадкового числа за заданою довжиною (256 біт) з використанням стандартної для python бібліотеки random:

```
def generate_number():
    while True:
        number = random.getrandbits(256)
        if division(number):
            if test_miller_rabin(number, 15):
                return number
            else:
                print(f'Не пройшло - {number}')
        else:
            print(f'Не пройшло - {number}')
```

Далі число перевіряється на простоту у 2 кроки:

Спочатку виконуємо просте відсорткування шляхом пробного ділення на числа від 2 до 49:

```
def division(n):
    for i in range(2,50):
        if n % i == 0:
            return False
    return True
```

Якщо число проходить цю перевірку, то далі використовуємо тест Міллера-Рабіна:

```
def test_miller_rabin(n, k):
    if n == 2 or n == 3:
        return True
    if n % 2 == 0 or n < 2:
        return False

    r, d = 0, n - 1

    while d % 2 == 0:
        r += 1
        d //= 2
```

Приклад чисел що не пройшли перевірку:

```
Не пройшло - 36002714150077618082825562541380441583417394495694262125579946543902568170435
Не пройшло - 30182438100064094353583378384715326226524197116641757749206770683570530454699
Не пройшло - 18100642768662873549572607037825532844388443785439292076957253975776116824037
Не пройшло - 79735107075918589303870279607521323741310784387640624969189546266391170792609
Не пройшло - 12531679585250300240415751478269056766198407107045892679990178685144336806691
```

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

```
def gen_keys():
    p = generate_number()
    q = generate_number()
    return p, q
```

2. 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 .

```
class RSA():
    def __init__(self, name : str):
        self.name = name
        self.p = None
        self.q = None
        self.d = None
        self.e = 2**16 + 1
        self.n = None

    def generate_key(self):
        self.p, self.q = gen_keys()
        self.n = self.p * self.q
        fn = (self.p-1)*(self.q-1)
        self.d = pow(self.e, -1, fn)
```

В результаті виконання цієї частини коду отримаємо:

```
Користувач: а
Відкритий ключ: (n = 87876A842A564293AC87267B35CFE8C484E2CE41D0A1DCAF8166B27D15F3D639BAFFBBAC699EEF1264DAA4C1D486B0B9005FFA18AA7C20D8093D4BF4403F1749, e = 10001)
Закритий ключ: (p = EDF13DFA01D6F065AFBF86BA0EC6007DEC17211642C7AF714C85367A9CE2E449, q = 91D07B15D5AB2BC32A880BF4258195421DD29C4B1D38CA728EB79FA722049B01, d = 65ECA25FA9135C5880ECD734DC207316B0357E16A317C7C739D30932E73A80597102898A58E05D45B4205DB3AAD9AD4574E4E952DEFD67C56C843D3BF302801)
```

Секретний ключ що складається з (d, p, q) зберігається в об'єкті класу RSA.

Значення n та e виводяться в hex форматі, за допомогою такої функції:

```
def to_hex(n:int):
    return hex(n)[2:].upper()
```

3. 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 М і знайти криптограму для абонентів А и В, перевірити правильність розшифрування. Скласти для А і В повідомлення з цифровим підписом і перевірити його.

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

Наприклад, для перевірки коректності операції шифрування необхідно а) зашифрувати власною реалізацією повідомлення для серверу та розшифрувати його на сервері, б) зашифрувати на сервері повідомлення для вашої реалізації та розшифрувати його локально.

Розглянемо реалізовані функції та приклад результату їх роботи:

- Шифрування

```
def Encrypt(M, received):  
    return pow(M, received.e, received.n)
```

```
Введіть користувача: a  
Повідомлення - 54269E51DA5D7EF0FC8A2852A161A464FDE3A87965EC7912B0C052373EFBDD4DE1A6DC0D03DDDA02CADA00BD0AFFBFED05235AA6756FA92E348C8D90ECB2430  
Зашифроване повідомлення - 148AEE2BD71E5AFCEFF94F46F77FE4B20DB80D38205FD923676DA9ABCFE372D298A1B2907633F254FF0F398C0921EBDB2479A72092D8EAB551C66DB27CF72B7
```

- Розшифрування:

```
def Decrypt(C, received):  
    return pow(C, received.d, received.n)
```

```
Розшифроване повідомлення - 54269E51DA5D7EF0FC8A2852A161A464FDE3A87965EC7912B0C052373EFBDD4DE1A6DC0D03DDDA02CADA00BD0AFFBFED05235AA6756FA92E348C8D90ECB2430
```

- Як зазначено в методичці, перевіримо коректність роботи наших функції на сайті <http://asymcryptwebservice.appspot.com/?section=rsa>.

Encryption

✖ Clear

Modulus

692B842ABB6864D7D136C39A344932922BEC56151B357583D25CFFA0779A8A0ADDC49EF37D3A0E5FD5ED

Public exponent

10001

Message

54269E51DA5D7EF0FC8A2852A161A464FDE3A87965EC7912B0C052373EFBD

Bytes ▾

Encrypt

Ciphertext

0148AEE2BD71E5AFCEFF94F46F77FE4B20DB80D38205FD923676DA9ABCFE372D298A1B2907633F254FF0F

Як бачимо, значення шифртексту та відкритого тексту співпадають з тими що отримали в нашій програмі.

- Створення та перевірка цифрового підпису

```
def sign(sender):
    M = generate_msg(sender.n - 1)
    signature = pow(M, sender.d, sender.n)
    return [M, signature]

def verify(M, sign, sender):
    if M == pow(sign, sender.e, sender.n):
        print('Повідомлення не спотворене')
    else:
        print('Усе пропало')
```

```
Хто підписує?
Ім'я користувача:a
Повідомлення - 1661B416D99E97F9816A0540F228EA377BE6BDAB731569E935DAF6E3471CDBC545BA546235AD5D5A616B302FB8D103BB0983F0ECA8DFC51
12811670B45069DF5
Підпис - A563FEFF5C31B28C4C5A0C3798180DDA7EE4E5160C66078DE502086783CA37B9422BE50B0467C510C471DF74FF0223E9C9C9FF2209B9FC053E20E
77EE1F4364
Повідомлення не спотворене
```

4. 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 $0 < k < n$.

```
def send_key(user1, user2):
    while user1.n > user2.n:
        user1.generate_key()
    k = random.randint(1, user1.n)
    k1 = pow(k, user2.e, user2.n)
    S = pow(k, user1.d, user1.n)
    S1 = pow(S, user2.e, user2.n)
    return [k1, S1]

def receive_key(user2, signed):
    k1, S1 = signed
    k = pow(k1, user2.d, user2.n)
    S = pow(S1, user2.d, user2.n)
    return [k, S]
```

Приклад роботи:

```
Перший користувач:a
Другий користувач:b
k1 - 24A7412699C7AA48F89CFE44963FD791976941B381090A67A45F80A995580E5F58CFC47F04324AEADC02F1FDDBD321300ACA5AFCBFE76227EEC8C26B8B4D8B62
S1 - 1E80C02E7C77C6D43CC4F4423D9F4A3C6BE2063DE2AD3BD9B8F4CE684E5D9E2E833EAA8747E1AC5844213D8DEF223DEFFAFBC5597FE300D20860A2B9DC0155C
k - 46E9C0A70B8DB9A07B4030BE853E61985C39A1ADC5C672429BEA042D9F9E16A58BB62D02A1357438A9DDA414B53C70AFD8A4B984B118FC2D665153F9B4C8C0F
S - 2985664C30F44C780365A98293A328537182346E8F3F7B5EB77CFEBF0052DCFFBC803B2E775C91BC9CD2EA9CD5C0B2845A9B162396D8B1732755616F24FF0C
Автентифікація успішна
```

* Тут була помилка з використанням перевірки для K, тому змінили функцію:

```
def send_key(user1, user2):
    while user1.n > user2.n:
        user1.generate_key()
        k = random.randint(1, user1.n)
        k1 = pow(k, user2.e, user2.n)
        S = pow(k, user1.d, user1.n)
        print(f'k - {k}\nS - {S}')
        S1 = pow(S, user2.e, user2.n)
        return [k1, S1]

def receive_key(user1, user2, signed):
    k1, S1 = signed
    k = pow(k1, user2.d, user2.n)
    S = pow(S1, user2.d, user2.n)
    if k == Encrypt(S, user1):
        print('Автентифікація пройдена успішно')
    else:
        print('Не пощастило')
```

Тепер як тільки запускається функція, відбувається перевірка k (тобто робота продовжиться лише після підтвердження k, якщо воно неправильне то програма зупиняється)

```
k - 71B06657E73760FB2FB8E261F847FA7FEDA4AB564568704717A9E9102D32F58445B8B30A387F809D18D0D84F27A338CA469A1C48EB008B7BEA2CC9A38659F19
S - A5BC34C0A28225BA887652D47F31A447367788357FF702CEE1EB5F7998DE1B5BC6BC2ACF0C65DE0888DDE062886347D10A9F55EF224A80B049A9EEB9AF1F58
k1 - AF970754B882B9AE7D19B6580AF97089255DF35FED7CC19BF8F940F66F2CAA82319EE78B243AA83F933828CF113B6DEF3847BD1EEDB3E5A3C2E1D88FFF9376AA
S1 - 1F8028667F1C0A78CB0475BB5546F5CEDD6D3700F71AF25A2E04173EFB33AC4F886A7EA26C184F9B18B4803F214C16B955A68255DC7A6FA4EEC4D54F1F07E21
Автентифікація пройдена успішно
```

Труднощі

Під час реалізації функцій шифрування та розшифрування та перевірки їх роботи на відповідному сайті виникла проблема з тим що ми використовували дані в десятковому форматі, а на сайті бачимо hex-значення, що призводило до неможливості перевірки коректності роботи програми. Також були невеликі труднощі у виборі методу зберігання ключів для певного користувача. У початковій версії програми також іноді пара ключів k, S некоректно розшифровувалась через те що число n у першого користувача генерувалось більше ніж у другого, що призводило до неправильної генерації пари k,S

Висновок:

В результаті виконання даної лабораторної роботи ми ближче ознайомилися з шифром RSA. Отримали навички використання тесту Міллера-Рабіна та схеми Горнера для роботи даного алгоритму. Також ознайомилися та реалізували на практиці технології цифрового підпису та протокол конфіденційного розсилання ключів по відкритих каналах зв'язку з підтвердженням справжності відправника.