Міністерство освіти і науки України Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського» Фізико-технічний інститут

ЛАБОРАТОРНА РОБОТА №5

з дисципліни

«Криптографія»

на тему: «Вивчення криптосистеми RSA та алгоритму електронного підпису; ознайомлення з методами генерації параметрів для асиметричних криптосистем»

Виконали:

студенти 3 курсу ФТІ

Ракович Дарина ФБ-73

Пекарчук Данило ФБ-74

Перевірили:

Чорний О.

Савчук М. М.

Завадська Л. О.

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 $p\ q \le p_1\ q_1$; 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 *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* і *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 $0 < k < n$.

Кожна з наведених операцій повинна бути реалізована у вигляді окремої процедури, інтерфейс якої повинен приймати лише ті дані, які необхідні для її роботи; наприклад, функція Encrypt(), яка шифрує повідомлення для абонента, повинна приймати на вхід повідомлення та відкритий ключ адресата (і тільки його), повертаючи в якості результату шифротекст. Відповідно, програмний код повинен містити сім високорівневих процедур: GenerateKeyPair(), Encrypt(), Decrypt(), Sign(), Verify(), SendKey(), ReceiveKey().

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.

Результати:

Згенеровані прості числа

p: 961087839029001566074045889263112155445324357165840450489116810720461596550 59

q: 612631087442508306668646851693663881690695657632547330394248809695143763412 27

p1: 106973336206024768195998646167073579418703766488697218919387940617484603089 27

q1: 110021291856065471927816456240494473722565720231803836940802625863131059625803

Отримані відкриті та закриті ключі
e:552480061199816828183298352829155968247078279265554195001014997087358959875098619
141512648694385937760184949699388050008725510815593169047292481129789871 8
n:588792287952107606157197581269723861713524289776259515064112080079235678165099107
283123264864327485985160546502532041681044223668937218626199654880081739 3
d:315779127988077210025512125539036764009979755275763449133097592940453049208561940
813017196267482142317612826568908278660839631023081576682970296969115314 9

e1:986589396845282085220574013689258253568665420334231385719778100562479104519853443401358004472236887119881478429468822615929497288742029685633786110131681
n1:117693446435400665178030800198239207150629954334587832206912722846500670036007250844041481259693604041569768192965946938762064315966561166300037040044338 1
d1:124496608880161829264458062095309878845294189600422215345611206477218672141518405511781317770616292042554144252927681221987825439560201992234142226024945

Зашифроване повідомлення та підписи:
BT: 300
ШТ:482681026706326201748025418914682444012487584109579163470713384527066416881508095760435109895586531304996312760132560095802276958450830858664440977162857 4

Підпис для A:
(300,294497583521763911149153564084106398971930263653171450493052833273197232941022112298362482535697842500984992282580362462454357486296011046618794826325752 7)

Підпис для B:
(300,107821766472431292723629574794747639374270951726170564407293164623089933830928198055408835536351379719935588428946060549168612217121062442172693184323170)

Код программи:

```
import binascii
from random import randrange, getrandbits



def gcd(a, b):
```

```
    def miller_rabin(self, in_number,
test_number=128):
        if in_number == 2:  # 2 is prime
        return True
        if in_number <= 1 or in_number % 2 ==
```

```
        while b != 0:
        a, b = b, a % b
        return a


def inv(a, m):
        input_m = m
        y = 0
        x = 1

        while (a > 1) :
        q = a // m
        t = m

        m = a % m
        a = t
        t = y

        y = x - q * y
        x = t

        if (x < 0):
        x = x + input_m

        return x


class RSAClass:
        def __init__(self, ):
        self.e = 0
        self.n = 0
        self.p =
self.generate_prime_number(length=256)
        self.q =
self.generate_prime_number(length=256)
        self.keypair =
self.generate_keypair(self.p, self.q)

        def generate_keypair(self, p, q):
        if not self.miller_rabin(p) or not
self.miller_rabin(q):
        return 'Numbers not prime'

        n = p * q
        phi = (p-1) * (q-1)
        e = randrange(1, phi)  # same as phi - 1

        # saving data to model for future
        self.e = e
        self.n = n
```

```
0:
        return False

        s = 0
        d = in_number - 1
        while d & 1 == 0:  # d must be odd
        d //= 2
        s += 1

        for i in range(test_number):  # making n
tests
        x = pow(randrange(2, in_number - 1), d,
in_number)

        if x != in_number - 1 and x != 1:
                for j in range(s):
                if pow(x, 2, in_number) == 1:
                print('Not approved: ', pow(x, 2,
in_number))
                return False

                if x != in_number - 1:
                return False

        return True

        def encrypt(self, pk, plaintext):
        key, n = pk

        cipher = pow(plaintext, key, n)
        return cipher

        def decrypt(self, pk, ciphertext):
        key, n = pk
        decrypted = pow(ciphertext, key, n)

        return decrypted

        def sign(self, message, n, d):
        return (message, pow(message, n, d))

        def verify(self, message, S, e, n):
        return message == pow(S, e, n)

        def send_key(self, message, e, n, d, n_2):
        sign = self.sign(message, d, n_2)

        return (self.encrypt((e, n), sign[0]),
self.encrypt((e, n), sign[1]))

        def recieve_key(self, message, e, n, d,
```

```
        # verify e and phi mutually prime
        g = gcd(e, phi)
        while g != 1:
        print('e and phi wasn\'t mutually prime')
        e = randrange(1, phi)
        g = gcd(e, phi)

        # generating secret key with eucl. alg.
        d = inv(e, phi)

        return ((e, n), (d, n))

  def generate_prime_number(self,
length=1024):
        p = 4
        while not self.miller_rabin(p, 128):
        p = (getrandbits(length) | 1)  # | 1 so it's
not even

        return p
```

```
n_2):
        s = pow(message, d, n)
        verify = self.verify(message, s, e, n)
        return (verify, self.decrypt((d, n_2),
message))


def main():
        rsa_a = RSAClass()
        rsa_b = RSAClass()
        print(rsa_a.p, '\n\n', rsa_a.q, '\n\n')
        print(rsa_b.p, '\n\n', rsa_b.q, '\n\n')

        print(rsa_a.e, '\n', rsa_a.n, '\n',
rsa_a.keypair[1][0])
        print(rsa_b.e, '\n', rsa_b.n, '\n',
rsa_b.keypair[1][0])
        message = int(input('Message: '))
        encrypted =
rsa_a.encrypt(rsa_a.keypair[0], message)
        decrypted =
rsa_a.decrypt(rsa_a.keypair[1], encrypted)
        sign_a = rsa_a.sign(message,
rsa_a.keypair[1][1], rsa_a.keypair[1][0])
        sign_b = rsa_b.sign(message,
rsa_b.keypair[1][1], rsa_b.keypair[1][0])
        print(encrypted)
        print(decrypted)
        print(sign_a)
        print(sign_b)
        # s = pow(message, rsa_a.keypair[1][0],
rsa_a.keypair[1][1])
        # verify = rsa_a.verify(message, s,
rsa_a.e, rsa_a.n)
        # key = rsa_a.send_key(message, rsa_a.e,
rsa_a.n, rsa_a.keypair[1][0], rsa_b.n)
        # recieved = rsa_b.recieve_key(message,
rsa_b.e, rsa_b.n, rsa_b.keypair[1][0], rsa_a.n)


if __name__ == '__main__':
        main()
```

Висновки:

Під час данного комп’ютерного практикуму, ми ознайомились з тестами перевірки чисел на простоту і методами генерації ключів для асиметричної криптосистеми типу RSA. Також, використовуючи криптосистему типу RSA, організували канал засекреченого зв’язку й електронний підпис, ознайомились із протоколом розсилання ключів.