КРИПТОГРАФІЯ КОМП’ЮТЕРНИЙ ПРАКТИКУМ №5

Вивчення криптосистеми RSA та алгоритму електронного підпису;

ознайомлення з методами генерації параметрів для асиметричних криптосистем

Виконали: ФБ-12 Карабінський Василь та Мосейко Олег

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.

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.

```
def find_p():
    while True:
        p = secrets.randbits(256)
        if miller_rabin(p, k=20):
            return p

def find_q():
    while True:
        q = secrets.randbits(256)
        if miller_rabin(q, k=20):
            return q
```

Написана дві функції `def find_q() та def find_p():` відповідно для генерування простих чисел p та q, для того, щоб перевіряти простоту чисел використали тест Міллера-Рабіна із попередніми пробними діленнями, код цього тесту реалізоований в функції `def miller_rabin(p, k=20):`

2. За допомогою цієї функції згенерувати дві пари простих чисел p, q і 1 1 p , q довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб pq <= p1q1 ; p і q – прості числа для побудови ключів абонента A, 1 p і q1 – абонента B

```
def gen_pp_qq1():
    p = find_p()
    q = find_q()
    p1 = find_p()
    q1 = find_q()
    while p*q >= p1*q1:
        return(gen_pp_qq1())
    return p,q,p1,q1
```

За допомогою цієї функції ми в подальшому згенерували дві пари чисел p, q і p1 q1, також як можемо бачити перевіряється умова `pq <= p1q1`

Згенеровані числа:

```
Згенеровані числа p і q для користувача A:
p=69484180286085756662357980635987651746378420086802761811361930247679791845301
q=32719116355383950405404217610558744743902793085465731676497639007202372462709


Згенеровані числа p1 і q1 для користувача B:
p1=10637382132383856736653237496524069317114266350899409966050401501905200679931 3
q1=40986111285997201939553024530377876613112288107365586429648252965411227551123
```

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 A і B – тобто, створити та зберегти для подальшого використання відкриті ключі (e,n) , (,) 1 n1 e та секретні d і d1 .

```
def GenerateKey(p, q):
    n = p * q
    fn = (p-1)*(q-1)
    e = 2**16 + 1
    d = congurence(e, 1, fn)
    return p, q, n, e, d
```

функція для генерації ключових пар

```
p, q, n, e, d = GenerateKey(qqq[0], qqq[1])
p1, q1, n1, e1, d1 = GenerateKey(qqq[2], qqq[3])
open_key_A = (n, e)
open_key_B = (n1, e1)
secret_key_A = (p,q,d)
secret_key_B = (p1,q1,d1)
```

Можемо бачити результат виконання, так як ключі занадто довгі додам їх в протокол не скріншотом.

Відкритий ключ A

(22734609796389155389339295885538352940854885031350971981662493026297472204434000914315404798571867003658520697291496378547948546438891250650930589193804096553 7)

Відкритий ключ B

(43598492786956297264014640037637072459840195474827807663241515008668540189728951672493554717609050862941488170547910546204858260003257811581052303087784996553 7)

Таємний ключ A
(69484180286085756662357980635987651746378420086802761811361930247679791845301,
32719116355383950405404217610558744743902793085465731676497639007202372462709,
16801971794377355406604647698491693618723871347383681400305519778578788565569240766561458955493997401490451733008431258516103874177134335457973677637354 73)
Таємний ключ B
(10637382132383856736653237496524069317114266350899409966050401501905200679931 3,
40986111285997201939553024530377876613112288107365586429648252965411227551123,
42293272181069420305472187320029554017338914312711565561294830655600690336175357713987839511751515400747299555012330025626497447385069519002918621047158 73)

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

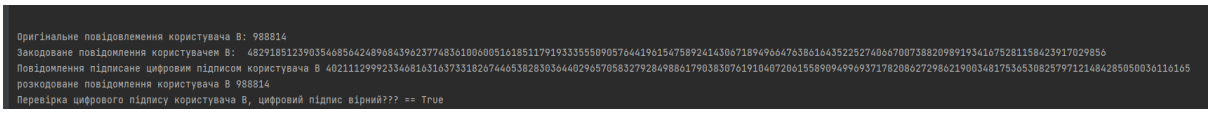
Використані функції `encrypt_rsa(message, public_key)`
`decrypt_rsa(message, secret_key)`
`signature(message, secret_key)`
`verify_sign(sign_mes, message, public_key)`
Ключі для А та В згенеровані за допомогою процедури з минулого пункту
Результат зашифрування, розшифрування, створення цифрового підпису, перевірка цифрового підпису для користувачів А та В:
А:

```
Оригінальне повідовлемення користувача А: 123456789
Закодоване повідомлення користувачем А:  29933210321340077084670899625382150720307995100481188192989414658586604311793308060076265954541937423034263275085091644810945994861599924969790757291397 22
Повідомлення підписане цифровим підписом користувача А 560957070611378456333379811624014243757004590008292465934724716648057723236973126143917507172126697628059809893908156691217803237691446763581439454107708 7
розкодоване повідомлення користувача А 123456789
Перевірка цифрового підпису користувача А, цифровий підпис вірний?? == True
```

додам ще і текстом
Оригінальне повідовлемення користувача А: 123456789

Закодоване повідомлення користувачем A:
29933210321340077084670899625382150720307995100481188192989414658586604311793308060076265954541937423034263275085091644810945994861599924969790757291397 22
Повідомлення підписане цифровим підписом користувача A:56095707061137845633337981162401424375700459000829246593472471664805772323697312614391750717212669762805980989390815669121780323769144676358143945410770 87
розкодоване повідомлення користувача A: 123456789
Перевірка цифрового підпису користувача A, цифровий підпис вірний??? == True

B:



Оригінальне повідовлемення користувача B: 988814
Закодоване повідомлення користувачем B:
48291851239035468564248968439623774836100600516185117919333555090576441961547589241430671894966476386164352252740667007388209891934167528115842391702985 6
Повідомлення підписане цифровим підписом користувача B
40211129992334681631637331826744653828303644029657058327928498861790383076191040720615589094996937178208627298621900348175365308257971214842850500361161 65
розкодоване повідомлення користувача B 988814
Перевірка цифрового підпису користувача B, цифровий підпис вірний??? == True

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().

До минулого коду були додані ще дві функції:

```
def send_key(prvt_key_A, public_key_b, msg):
    k1 = encrypt_rsa(msg, public_key_b)
    S = signature(msg, prvt_key_A)
    S1 = encrypt_rsa(S, public_key_b)
    return k1,S1

def recieve_key(prvt_key_B, public_A, k1, S1):
    k = decrypt_rsa(k1, prvt_key_B)
    S = decrypt_rsa(S1, prvt_key_B)
    if  verify_sign(S, k, public_A) == False:
        raise TypeError("cant verify signature")
    else:
        return k,S
```

Вони нам потрібні для конфіденційного розсилання ключів з підтвердженням справжності по відкритому каналу за допомогою алгоритму RSA

результат роботи коду для обміну ключами:

закодований ключ:

```
24213571686185868820973655486831592743963187336861094724062028840671504575247991167947078895611633226321510263669245025418449036964386203308426132192064
```

закодований підпис:

```
38934664213577030224551519813278467490281090891508789499432621542846639252475584168555633593597358381100910077135123158020140383490314704406662947249963
```

декодований ключ:

```
44648860808276966654688289023262154370692109805213542883914008829980054837592253788458779013356125124577468658504293924762710580101762388282196365152208 4
```

декодований підпис:

```
36506710371656197544164220201955307452177668696513010337415354676888267103822813318172181308629677959865312160020306526474603586523635655525070715114052 1
```

Висновок

В ході виконання лабораторної роботи було успішно реалізовано криптоалгоритм RSA, вивчено тести простоти чисел і методи генерації ключів. Практично застосовано систему захисту інформації на основі RSA, включаючи засекречений зв'язок та електронний підпис. Вивчено протокол розсилання ключів, надаючи поглиблене розуміння безпеки комунікації. Лабораторна робота надала практичні навички в галузі асиметричних криптосистем.