

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Фізико-технічний інститут

Криптографія
Комп'ютерний практикум №4

Виконав:
студент групи ФБ-05
Ємельянов Р.

Київ - 2022

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.

Постановка задачі:

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. 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 $p \neq q$ і $p_1 \neq q_1$ – прості числа для побудови ключів абонента А, $p_1 \neq q_1$ – 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 $0 < k < n$. 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()`. 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

Хід роботи:

У нашої лабораторній роботі ми реалізували усі потрібні функції а також тести на них

```
[derek_smits@Cachy yemelianov_fb-05_cp4]$ python Lab4.py

Ключі А
e: 0084936689957741999103726873524495313337491475647133990950775342839402814986725465339015869783889436704253023802713055033657626323141473233620477162787701
n: 7800932883261880933363876383323668125749355663085283955554581729107003949551434690185076817081751792277757332161083498468268142086121706255235808084938557
d: 446786117273437868172955172911775110931036979909433088119978783565920014538351335802534109672838109179801931395654121871731269658315250728423914473249961
p: 107553024556391748448861660538317897882970633100657412834694211879830063878171
q: 72531041460128614873218630643317329509721950536872410794715755803590462604167

Ключі В
e_1: 31691271771880560924623965125710905495900855848459414732208614876682007807013005463105892753648889902258789512763586128836670062973207774207695292992837
n_1: 94065572037235248378679130673759218400001332589225918570037906044841918063639577961053470198939749606180744074808706261335961184049602144222956621413197
d_1: 7234822748145284311680778263102681489126953601932175047143317739493403721860151798789483356626128416408405540829476950314784163317131199301641857965126273
p_1: 108721137135473024509413255647934458856498224106530662579881708856938051163639
q_1: 87347846554364819848074784902299940713062094013892884898095469217599947970523

Start k: 5221085049960268674179942239419328555575467030732805715255136240954972190809049474216952796114332804717699034073796347790061870353086689032917195522590460
Message: 7695658576258002589814394247976603724260017761180481852553143993169166511719173129046589240394484920460298264721899739275841384586840608802909080439709543

The key has been received: 5221085049960268674179942239419328555575467030732805715255136240954972190809049474216952796114332804717699034073796347790061870353086689032917195522590460

Encrypted message: 3225967906422039823147963237968658500974113435402979991056161047251716821391307379891540892492774908065998454525519264151163727228900761501734117404622399
Decrypted message: 7695658576258002589814394247976603724260017761180481852553143993169166511719173129046589240394484920460298264721899739275841384586840608802909080439709543
```

Get server key

Clear

Key size

256

Get key

Modulus

801073D97D3D40CA083CAB93E2AF4F7FC6FDE8C3F4F7886D34F5D526FF12C06D

Public exponent

10001

Encryption

Clear

Modulus

801073D97D3D40CA083CAB93E2AF4F7FC6FDE8C3F4F7886D34F5D526FF12C06D

Public exponent

10001

Message

C32

Bytes

Encrypt

Ciphertext

246482D622612E41367803FF140C73CA8C55D1174EC5FFA58344CF550C18B061

Sign

Message

C32

Bytes

Signature

59FEFB74834333B39449082D48CFA5F930F463E26AE86079B8BDD124AADC1855

Verify

Message

C32

Bytes

Signature

59FEFB74834333B39449082D48CFA5F930F463E26AE86079B8BDD124AADC1855

Modulus

801073D97D3D40CA083CAB93E2AF4F7FC6FDE8C3F4F7886D34F5D526FF12C06D

Public exponent

10001

Verification

true

✓

Внесемо певні зміни у нашу програму аби перевірити отримані дані

```
def set_test_custom(self, m, s, e, n):
    self.m = int(m, 16)
    self.s = int(s, 16)
    self.e = int(e, 16)
    self.n = int(n, 16)

def test_case(self):
    print(f'n: {self.n}')
    print(f'e: {self.e}')
    print(f'mes: {self.m}')
    print(f'Ciphertext: {hex(self.encrypting(self.m, self.e, self.n))}')
    print(f'sign: {self.s}')
    if self.m == pow(self.s, self.e, self.n):
        print('Passed!')
    else:
        print('Failed!')
    return self.m == pow(self.s, self.e, self.n)
```

```
rsa_test.set_test_custom(  
    "C32",  
    "59FEFB74834333B39449082D48CFA5F930F463E26AE86079B8BDD124AADC1855",  
    "10001",  
    "801073D97D3D40CA083CAB93E2AF4F7FC6FDE8C3F4F7886D34F5D526FF12C06D"  
)  
rsa_test.test_case()
```

```
n: 57925113736022094062863366039231680897275770885516528154985558985582962720877  
e: 65537  
mes: 3122  
Ciphertext: 0x246482d622612e41367803ff140c73ca8c55d1174ec5ffa58344cf550c18b061  
sign: 407063581578748314969184041395780355385075815922134917196839236750567368356  
Passed!
```

Висновки:

В результаті виконання практикуму я ознайомився з системою захисту інформації на основі крипто схеми RSA. Було засвоєно і практично використано тест Міллера-Рабіна для перевірки чисел на простоту і генерації простих чисел. З допомогою Asym Crypto Lab Environment перевінив створені функції крипто схеми RSA.