



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Фізико-технічний інститут

КРИПТОГРАФІЯ

КОМП'ЮТЕРНИЙ ПРАКТИКУМ №4

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

Виконав:
студент III курсу ФТІ
групи ФБ-95
Чорний Анатолій
Перевірила:
Селюх П. В.

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 $1 < p, q$ 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, $1 < p$ 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) , (d, n) і 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()`.

Хід роботи

Bob	
n	7770052468583481671689698392413576634030051817481798680000654104698353851632093077449366704171555232627229352556163391174817001566205078749246021004126449
e	5988456901140636526672819621542038726849245941871117423603981867180831401718867506992699475841905397814846965902992666055463675797356600393329493734945699
d	6863087819966519702431142049016970136662323358815544445618087620802218775353692473741302198081944013953663127051908604331546904221180497474740545769189579

Alice	
n	6105119720500251414804179223380853773135641482250770203927580735903660834244020847515138586890079898987930934847923054905826711830438444198610777332316967
e	4957455505211771531190247584846922929303140139917096593292892906325617435967861717994897201776912778122110732822317621643233554174677541693894190911195159
d	974562406590431407973164164977916648151995289657106787202833308660279093784266770700103710675694353652979011062050661532248959882770742892871407844692799

```
Alice = abonent()
Bob = abonent()

msg = 'hello, my name is tolya its pleasure for me to meet you'

Bob.GenerateKeyPairReceiver()
Alice.GenerateKeyPairSender(Bob.n)

packet = Alice.SendKey(msg, Bob.e, Bob.n)
output = Bob.ReceiveKey(packet, Alice.e, Alice.n)
```

```
C:\Users\morning star\Desktop\backup\third\crypto\4>python mr.py
hello, my name is tolya its pleasure for me to meet you
```

```
def testserver():
    Alice = abonent()
    servermod = int('0xA53346F4729270537DF889F6CBD514B8AD5E7101F7D13DFFE77CE2636014A713D64E4764F952407115714B8D4A187EDFA91F7063EB381ACEE99BE642BD8F6DAF', 16)
    serversign = int('0x830CCAF347763CDC6AE2BBD08D1E449FA829B6930F62916A79D4B352F2A2EEA37F45B2F01439FDDC26377DD3DD2FC9149F5394781676FC69CC207516C8CE3B', 16)
    e = 0x10001
    msg = 0x24082001
    Alice.GenerateKeyPairSender(servermod)
    packet = Alice.SendKey(msg, e, servermod)
    print(f'encrypted message is {hex(encrypt(msg, e, servermod))[2:]}')
    print(f'verify is ok {verify(msg, serversign, e, servermod)}')
    print(f'message is {hex(packet[0])[2:]}')
    print(f'signature is {hex(packet[1])[2:]}')

testserver()
```

Get server key

Clear

Key size

512

Get key

Modulus

A53346F4729270537DF889F6CBD514B8AD5E7101F7D13DFFE77CE2636014A713D64E4764F952407115714B

Public exponent

10001

Verify

Clear

Message

24082001

Bytes

Signature

830CCAF347763CDC6AE2BBD08D1E449FA829B6930F62916A79D4B352F2A2EEA37F45B2F01439FDDC263

Modulus

A53346F4729270537DF889F6CBD514B8AD5E7101F7D13DFFE77CE2636014A713D64E4764F952407115714B

Public exponent

10001

Verify

Verification

true

✓

Decryption

Clear

Ciphertext

2ACD790619EAE39AE7DD0A7CF1B0C3A02303E7A354BBDAC9E1DEBF067D00

Bytes

Decrypt

Message

24082001

Receive key

Clear

Key

2acd790619eae39ae7dd0a7cf1b0c3a02303e7a354bbdac9e1debff067d00d9de5eed06c7767864270945c215f6ccc2ba23f32dc318b762874615ce7af13efb26

Signature

6820937a8a679d95334004d2b0804f6ae1993b5a7974d5b74fd4a94228df12cbe45b81015edcecd838014940282b3ee3d93fab2c4a10b6a6d3964f532e0155b

Modulus

7cc186b3dd28055cc53b66e9a810c4b7142058086d111ac4bd6ce437ba28297da48eff0d920ee06533ec0f99038a0ab154ada19e542432e66ab0ff7a246c3289

Public exponent

10001

Receive

Key

24082001

Verification

true

✓

```
C:\Users\morning_star\Desktop\backup\third\crypto\4>python mr.py
Alice n = 7cc186b3dd28055cc53b66e9a810c4b7142058086d111ac4bd6ce437ba28297da48eff0d920ee06533ec0f99038a0ab154ada19e542432e66ab0ff7a246c3289
Alice e = 3f28c1c9fbaf7d11a9a48f7e588b3c7bfe3e1759a8755edc597409c1e5120024b6ffff8c72c5f27a907f3c86a6be82dc26ffea5677b9d3ee3a1bf22e767335315
Alice d = 3c59189d61edc3af49d00455ae0bacade190344933c8828063422eb8037efd76eb1b917f04b826f3c19785e7245ac8f559b386d52b89d7528744ddf4d21ce875
encrypted message is 2acd790619eae39ae7dd0a7cf1b0c3a02303e7a354bbdac9e1debff067d00d9de5eed06c7767864270945c215f6ccc2ba23f32dc318b762874615ce7af13efb26
verify is ok True
message is 2acd790619eae39ae7dd0a7cf1b0c3a02303e7a354bbdac9e1debff067d00d9de5eed06c7767864270945c215f6ccc2ba23f32dc318b762874615ce7af13efb26
signature is 6820937a8a679d95334004d2b0804f6ae1993b5a7974d5b74fd4a94228df12cbe45b81015edcecd838014940282b3ee3d93fab2c4a10b6a6d3964f532e0155b
```

1. В якості теста на простоту числа було обрано ймовірнісний тест Міллера-Рабіна. Також мали місце спроби реалізувати тести Соловея-Штрассена та Ферма, але за низької швидкодії їх було відкинуто.
2. Була написана функція вибору випадкового простого числа `getran()`.
 - 2.1. За допомогою генератора псевдовипадкових чисел вбудованого у мову Python, було обрано випадкове число з проміжку від найменшого числа довжиною X біт до найбільшого числа довжиною X біт.
 - 2.2. Потім базуючись на постулаті Бертана було побудовано алгоритм що на основі випадкового числа перевіряв методом Міллера-Рабіна, кожне наступне непарне число.
3. Для створення ключів була створена функція `genkey()` що створює пару різних ймовірно простих чисел, та на їх основі вираховує значення e , n , d (відкритий та таємний ключ)
4. Щоб привести умови задачі до умов реального світу, були написані функції `encode()` та `decode()` що перетворюють строкове значення у числове, для того щоб повідомлення могло бути оброблено у нашій криптосистемі.
5. На основі формул представлених у методичних матеріалах були написані функції: `encrypt()`, `decrypt()`, `sign()` та `verify()`. Для демонстрування коректного обміну ключами було створено клас `Abonent` що має методи реалізовані на основі функцій що були написані раніше (`Encrypt()`, `Decrypt()`, `Sign()`, `Verify()`). Для демонстрації було створено 2 об'єкти класу `Abonent`: `Alice` і `Bob` – класичні герої криптографічних задач.
6. Через вимогу для числа n (n відправника повинен бути менше за n отримувача), були створені 2 різних функції для створення ключа для об'єкту класу- `GenerateKeyPairSender(n)` та `GenerateKeyPairReceiver()`.
7. На основі вже написаних функцій, були написані функції
 - 7.1. `SendKey()` що приймає на вхід повідомлення та відкритий ключ отримувача, та віддає на вихід пакет що містить у собі зашифровані підпис та повідомлення.
 - 7.2. `ReceiveKey()` приймає на вхід пакет що був відправлений раніше, та відкритий ключ відправника, розшифровує підпис та повідомлення. Якщо перевірка розшифрованого підпису не дає позитивного результату повідомлення не розшифровується. Інакше ми розшифровуємо і декодуємо повідомлення.

Висновки: в ході виконання комп'ютерного практикуму мною був досліджений метод побудови криптосистеми RSA. Для досягнення результату були реалізовані ймовірнісні тести на простоту чисел, реалізовані методи вибору ймовірно простих чисел у проміжку. Були вдосконаленні знання щодо організації засекреченого зв'язку й електронного підпису за допомогою RSA, були проаналізовані протоколи розсилання ключів.