



Міністерство освіти і науки України

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

Виконали:

студенти групи ФБ-32

Кошеленко Н. Е.

Кухарук І. А.

Київ – 2025

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.

Порядок і рекомендації щодо 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, p_1, q_1 довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб $p \neq p_1, q \neq q_1$; $p \neq q$ – прості числа для побудови ключів абонента А, p_1, q_1 – абонента В.
3. Написати функцію генерації ключових пар для RSA. Після генерування функція повинна повернати 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 и 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 k $n < 0$.

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: GenerateKeyValuePair(), 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.

Отже, постановка задачі: у роботі розглядається програмна реалізація крипtosистеми RSA мовою Python. Нам потрібно відтворити основні механізми алгоритму: генерацію простих чисел, побудову ключів, шифрування та розшифрування повідомлень, а також цифровий підпис. Для цього використовуються базові математичні процедури - тест Міллера–Рабіна, модульне піднесення до степеня та розширений алгоритм Евкліда. Крім основних операцій RSA, демонструється протокол передавання секретного ключа між двома абонентами. Також все перевірити за допомогою тестового середовища.

Код:

```
import random
from math import gcd
```

random використовується для генерації випадкових чисел, простих чисел, основи Miller–Rabin та сесійного ключа в протоколі.

gcd – знаходження найбільшого спільного дільника, потрібне при виборі відкритого показника e.

```
# Евклід
def egcd(a, b):
    if b == 0:
        return a, 1, 0
    g, x1, y1 = egcd(b, a % b)
    return g, y1, x1 - (a // b) * y1
```

Алгоритм знаходить НСД чисел a та b і коефіцієнти (x, y), що задовольняють рівняння:

$$a*x + b*y = \text{gcd}(a, b)$$

Використовується для пошуку $d = e^{-1} \pmod{\varphi(n)}$.

```
def mod_inverse(a, mod):
    g, x, _ = egcd(a, mod)
    if g != 1:
        return None
    return x % mod
```

Обчислює приватну експоненту d у RSA. Якщо НСД(a, mod) ≠ 1 - зворотний елемент не існує.

```
# modpow
```

```

def modpow(base, exp, mod):
    r = 1
    base %= mod
    while exp > 0:
        if exp & 1:
            r = (r * base) % mod
        base = (base * base) % mod
        exp >>= 1
    return r

```

Попередня перевірка на малі дільники:

```

# Miller-Rabin

small_primes = [2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,
                 53,59,61,67,71,73,79,83,89,97]

def trial_division(n):
    for p in small_primes:
        if n == p:
            return True
        if n % p == 0:
            return False
    return True

```

Відсіює складені числа, що діляться на маленькі прості.

```

def miller_rabin(n, k=20):
    if n < 2:
        return False
    if not trial_division(n):
        return False
    d = n - 1
    s = 0
    while d % 2 == 0:
        d //=
        s += 1
    for _ in range(k):
        a = random.randrange(2, n - 1)
        x = modpow(a, d, n)
        if x == 1 or x == n - 1:
            continue
        ok = False
        for _ in range(s - 1):
            x = (x * x) % n
            if x == n - 1:

```

```

        ok = True
        break
    if not ok:
        return False
return True

```

Це ймовірнісний тест на простоту, який підходить для великих чисел (256 біт). Повертає True - число дуже ймовірно просте.

```

def random_prime(bits=256):
    while True:
        x = random.getrandbits(bits)
        x |= (1 << (bits - 1)) | 1
        if miller_rabin(x):
            return x

```

Генерує 256-бітні прості числа: встановлює старший біт (гарантія розміру), робить число непарним, перевіряє Miller–Rabin.

```

# RSA
def GenerateKeyPair(bits=256):
    p = random_prime(bits)
    q = random_prime(bits)
    while q == p:
        q = random_prime(bits)
    n = p * q
    phi = (p - 1)*(q - 1)
    while True:
        e = random.randrange(2**16, phi - 1)
        if gcd(e, phi) == 1:
            break
    d = mod_inverse(e, phi)
    return (e, n), (d, p, q)

```

Генерує p та q, формує n = p · q, знаходить φ(n), вибирає e, взаємно просте з φ(n), знаходить d як обернене, повертає ключі.

Операції RSA

Тут реалізовано основні формули RSA. Всі вони базуються на modpow(). Функція modpow()- це саме $M^e \text{mod } n$.

$$C = M^e \text{mod } n, M = C^d \text{mod } n,$$

M — повідомлення у числовій формі,

C — шифротекст,

e — відкритий ключ,

d — секретний ключ.

$$S = M^d \text{mod} n, M = S^e \text{mod} n.$$

```
def Encrypt(M, pub):
    e, n = pub
    return modpow(M, e, n)

def Decrypt(C, priv):
    d, p, q = priv
    return modpow(C, d, p*q)

def Sign(M, priv):
    d, p, q = priv
    return modpow(M, d, p*q)

def Verify(M, S, pub):
    e, n = pub
    return M == modpow(S, e, n)
```

Перетворення значень:

```
def text_to_int(t):
    return int.from_bytes(t.encode("utf-8"), "big")

def int_to_text(v):
    return v.to_bytes((v.bit_length() + 7) // 8, "big").decode("utf-8",
errors="ignore")

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

Код дозволяє шифрувати текст, який попередньо переводиться в число:

```
# умова n_A ≤ n_B
while True:
    public_A, private_A = GenerateKeyPair()
    public_B, private_B = GenerateKeyPair()
    eA, nA = public_A
    eB, nB = public_B
    if nA <= nB:
        break
```

```
dA, pA, qA = private_A  
dB, pB, qB = private_B
```

Ключ А не повинен бути сильнішим за ключ В.

Виведення ключів у числовій та HEX формі (прінти пропустимо).

Шифрування

```
# шифрування  
print("\n===== ШИФРУВАННЯ =====\n")  
  
msg = "NikitaIrynaCP4"  
M = text_to_int(msg)  
cipher = Encrypt(M, public_A)  
cipher_hex = to_hex(cipher)  
  
print("M як число =", M)  
print("Шифртекст =", cipher)  
print("Шифртекст HEX =", cipher_hex)  
  
dec = Decrypt(cipher, private_A)  
print("Розшифровано =", int_to_text(dec))
```

Переводимо текст → число, шифруємо → отримуємо ciphertext, отримуємо HEX для сайту.

Підпис RSA

```
# підпис  
print("\n===== ПІДПИС RSA =====\n")  
  
sig = Sign(M, private_A)  
sig_hex = to_hex(sig)  
  
print("Підпис =", sig)  
print("Підпис HEX =", sig_hex)  
print("Перевірка =", Verify(M, sig, public_A))
```

Показано підпис у числі та у HEX.

```
# протокол  
print("\n===== ПРОТОКОЛ А → В =====\n")  
  
k = random.randint(2, nA - 1)  
C1 = Encrypt(k, public_B)  
S1 = Sign(k, private_A)
```

```

k_recv = Decrypt(C1, private_B)
valid = Verify(k_recv, S1, public_A)

print("k =", k)
print("C1 =", C1)
print("S1 =", S1)
print("Отриманий k' =", k_recv)
print("Перевірка підпису =", valid)

```

A → B передає сесійний ключ з підтвердженням автентичності. В отримує k, перевіряє підпис
→ valid = True.

Результати:

```

===== ГЕНЕРАЦІЯ КЛЮЧІВ =====

[ Абонент А ]
e_A = 273505311754115299639918981627838339733012629559528435312949976748119114262622972793000151300873409995375296452879179502653894809158379347965620830
2238717
n_A = 408720531261691607728719713478909028126469504372515853551739140563421307812695756380800943170046761684300392045280868097722550693475302093887398544
4063769
d_A = 124389763402190498224705720665778219156685780569576408329878224121084381003995802904811579671052062334122847567970280743482220377291260717332940711
4657153
p_A = 65000836799320580421954476882068786101145682592073374552971936253192150904819
q_A = 62879272235148170983192805333413069785467391856504690517913480760245440122051

HEX:
n_A HEX = E4E9D7EA11BB2B74770F99E1E175BCB3D4327FC32FA7DA82D77A36079885BFEE49634AEF5600DA666620E36A4F13727EC77243419B1C41A884B8DF75AFAA19
e_A HEX = 3438A941C4E4D9BA042BE99FB1B8DBFAB5B2DA3B7B9E374C88E240CECE85745D6AE1AEAF42E65C9DF2163B9E373004D0EAE1B1D4F28E1B886914E68FD
d_A HEX = 17C00B8408BED4EBD38008122F86EC7FE0A3C2AC27BADCF38193E01BFBEA563787B62DDF3479861BA80EAD048BC91F8ED6A0D52D93FA64C1B8083B83684EBD81

[ Абонент В ]
e_B = 25773195147413929272948896452237976152752360502345596764658100461976174595834662883947089003131674736933404321142090998507194908429109914633090374
1698581
n_B = 52895997022305169963902261849624838219339162851667341537249467302061365573770208514173505063083283887084292567758312953577296993970383248925089721
415357
d_B = 70610871394605718348613353419430451048634726907935038376308063881091225969759
q_B = 113495050198680404171144169439133779800585944527551259534372052627351160100179

HEX:
n_B HEX = 99038A2CAF45735013F18D076C59390D03F7A85CC5DF1520C0DAF49D1B9785C6DF6A548E52FCC408FC6F3E1D64FAC903CD0869A471406A73B624D78389E8CD
e_B HEX = 3135ACE50410C563A9F1E7BA5A5ADEFCA2AA88645801B2498176F010B9A24A36044E1D7971ADA3955F6068CF66C8CE861B4C5D4E381DA235B27E7596E8B9215
d_B HEX = A1980A24E521E2DASF489E00D9CE23485DA9857AA961C868A8F8A8A2302285078D8F2D31E3CB9721250EAC4E65AA24E1B9A948CEE000AEE28531C6FD662F6BD

```

```

===== ШИФРОВАННЯ =====

М як число = 1590380148531679130587680381030452
Шифртекст = 69595869193112695901117544825480603261012059610338684040508138190843461658471350066065941179458848352431284514551542552177977915004245733028
440564993978
Шифртекст HEX = D49C675213881777C897076FDAB71EAB3DF56A82939274B5C04B85802EEE6CC8A85BBD071DDACD62E9612E1C25B997B7237CB4C6313F257B892B306A568FBA
Розшифровано = NikitaIrynaCP4

```

```

===== ПІДПІС RSA =====

Підпіс = 403061854877079844684610770660553706240370204036770582181569069016532192341035479254282584753450173094251299526045691234010034581367730795832930
1746931614
Підпіс HEX = 4CF540C98A423FA34184FA47CADFFEE2BC89542636C605F41260FDF955E7A727901731CADEBB8E90DEC1DB89522EDF6E8184441E3CE9CACBB4FF7A99785579E
Перевірка = True

```

```

===== ПРОТОКОЛ А → В =====

k = 23864052051107215508454442070565644245736385925316770572420164673206938927507899429994656484950473621061659248689607614798680116438031899225885153051
7803
C1 = 6624704728758518648589528594442876391801001375033415538568547043496977666792903063149443121303674296297655432024138382038183646977799592305561330047
136261
S1 = 209288792375390049629988348318473370770798192412208051961930603646602777844879212449103382291300893848730526888662430090607400708278214929013670835
981372
Отриманий k' = 238640520511072155084544420705656442457363859253167705724201646732069389275078994299946564849504736210616592486896076147986801164380318992
258851530517803
Перевірка підпису = True

```

Перевірка:

Server Key

RSA Testing Environment

Server Key

Encryption

Decryption

Signature

Verification

Send Key

Receive Key

Get server key

Key size: 256

Modulus: A2DAA1482AFF2D717F132D1D6FEF3E33BA6B6464E1FC91F499EA719E298D43D1

Public exponent: 10001

Signature

RSA Testing Environment

Server Key

Encryption

Decryption

Signature

Verification

Send Key

Receive Key

Sign

Message: NikitalrynaCP4

Type: Text

Signature: 02AB94621923EBEB101162DD9084DEA4DCB21D8B5DD57E0142A97F638053E1B4

Verification Перевірка підпису сервера

RSA Testing Environment

Server Key

Encryption

Decryption

Signature

Verification

Send Key

Receive Key

Verify

Message: NikitalrynaCP4

Type: Text

Signature: 02AB94621923EBEB101162DD9084DEA4DCB21D8B5DD57E0142A97F638053E1B4

Modulus: A2DAA1482AFF2D717F132D1D6FEF3E33BA6B6464E1FC91F499EA719E298D43D1

Public exponent: 10001

Verification: true

TRUE - серверний підпис валідний.

Verification Перевірка нашого підпису на сайті

RSA Testing Environment

Server Key

Encryption

Decryption

Signature

Verification

Send Key

Receive Key

Verify

Message	NikitalrynaCP4	<input type="button" value="Text"/>
Signature	4CF540C98A423FA34184FA47CADFFEE2BC89542636C605F41260FDF955E7A727901731CADEBBAE90DEC1	
Modulus	4E09D7EA11BB2B74770F99E1E175BCCB3D4327FC32FA7DA822D77A360798B5BFEE49634AEF5600DA66667	
Public exponent	3438A941C4E4D9BAD42BE99FB01B8DBFAB5B2DA3B7B9E374C88E240CECE885745D6AED1AEA49F42E65C	
<input type="button" value="Verify"/>		
Verification	true	<input checked="" type="checkbox"/>

TRUE - наш підпис правильний.

Encryption

Server Key

Encryption

Decryption

Signature

Verification

Send Key

Receive Key

Encryption

Modulus	A2DAA1482AFF2D717F132D1D6FEF3E33BA6B6464E1FC91F499EA719E298D43D1	
Public exponent	10001	
Message	NikitalrynaCP4	<input type="button" value="Text"/>
<input type="button" value="Encrypt"/>		
Ciphertext	5B8FD2D4EBF1C5C7AF598C058805F3340CEF42CBE89B5CD48A312966677ECE1E	

Decryption

RSA Testing Environment

Server Key

Encryption

Decryption

Signature

Verification

Send Key

Receive Key

Decryption

Ciphertext	5B8FD2D4EBF1C5C7AF598C058805F3340CEF42CBE89B5CD48A312966677ECE1E	<input type="button" value="Text"/>
<input type="button" value="Decrypt"/>		
Message	NikitalrynaCP4	

Висновки:

У ході лабораторної роботи була реалізована повноцінна крипtosистема RSA: генерація простих чисел, побудова ключів, шифрування, розшифрування, цифровий підпис та протокол безпечної розсылки ключів між двома абонентами.

Під час виконання завдання ми використали частину коду з попередньої лабораторної роботи - зокрема алгоритм Евкліда та модульне піднесення до степеня, що дозволило швидко та правильно реалізувати ключові математичні операції RSA.

Вдалося самостійно реалізувати тест Міллера–Рабіна для генерації великих простих чисел, а також усі необхідні функції RSA. Робота програми була успішно перевірена в онлайн-сервісі, що підтвердило коректність реалізації.

У результаті виконання роботи ми на практиці розібралися, як працює RSA, як формуються ключі, як створюється цифровий підпис і як організовується передача секретних даних по відкритому каналу. Також лабораторна робота показала зв'язок між теорією та реальною реалізацією криптографічних алгоритмів, ну і сама теорія ще раз закріпилася.