КРИПТОГРАФІЯ

КОМП’ЮТЕРНИЙ ПРАКТИКУМ №4

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

ФБ-13 Владислав Садохін та Данило Розумовський

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.

Код

```
def trial_division(n):
    if n < 2:
        return False
    for i in range(2, 100):
        if n % i == 0:
            return False
    return True


def is_prime_miller_rabin(n, k=10):
    if n <= 1:
        return False
    if n <= 3:
        return True

    def find_s_d(n):
        s = 0
        while n % 2 == 0:
            s += 1
            n //= 2
        return s, n

    s, d = find_s_d(n - 1)

    for i in range(k):
        a = random.randint(2, n - 2)
        x = pow(a, d, n)
```

```
        if x == 1 or x == n - 1:
            continue
        else:
            for j in range(s - 1):
                x = pow(x, 2, n)
                if x == n - 1:
                    break
                if x == 1:
                    return False
            if x == n - 1:
                continue
            return False

    return True


def generate_random_prime_number(min_value, max_value):
    while True:
        n = random.randint(min_value, max_value)
        if trial_division(n):
            if is_prime_miller_rabin(n):
                return n


def decimal_to_binary(decimal_number):
    binary_representation = bin(decimal_number)[2:]
    return binary_representation
```

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.

Код

```
print("Task1-2")
print("-------------------------------------------------------------------------
-")
p1 =
generate_random_prime_number(10000000000000000000000000000000000000000000000000
00000000000000000000000000000,
55555555555555555555555555555555555555555555555555555555555555555555555555555
5)
q1 =
generate_random_prime_number(10000000000000000000000000000000000000000000000000
00000000000000000000000000000,
55555555555555555555555555555555555555555555555555555555555555555555555555555
5)

p2 =
generate_random_prime_number(55555555555555555555555555555555555555555555555555
55555555555555555555555555555,
99999999999999999999999999999999999999999999999999999999999999999999999999999
9)
q2 =
generate_random_prime_number(55555555555555555555555555555555555555555555555555
55555555555555555555555555555,
99999999999999999999999999999999999999999999999999999999999999999999999999999
9)

print(f"Перша пара простих чисел: p = {p1}\nq = {q1}")
```

```
print(f"Довжина p = {len(decimal_to_binary(p1))} bit, q =
{len(decimal_to_binary(q1))} bit\n")
print(f"Дурга пара простих чисел: p1= {p2}\nq1 = {q2}")
print(f"Довжина p1 = {len(decimal_to_binary(p2))} bit, q1 =
{len(decimal_to_binary(q2))} bit")
print("----------------------------------------------------------------------
-\n")
```

Результати:

Так як чисел що не пройшли перевірку дуже багато, то всі не влізли в скріншоти тому ми тільки початок та кінець цих чисел вставили в протокол, також в кінці числа які є прості та їхня довжина, яка має бути мінімум 256 біт

```
Task1-2
----------------------------------------------------------------------
Кандидат що не пройшов: 291272150448228616460115310527399266148817969758530995058177374115600635 7403
Кандидат що не пройшов: 492002264710862720704694538331153358619315367940613058563524909690713444077912
Кандидат що не пройшов: 476412681796191739142038251244808192326615789810509015717345762600297085494999
Кандидат що не пройшов: 201091382984612304974364236873083619217672682721947903799895707768622485792104
Кандидат що не пройшов: 529964299354323718797101099433221322210871398681231345489994751866452174464996
Кандидат що не пройшов: 202536005389036520613148507150157574118312423468929063983529285813197388683620
Кандидат що не пройшов: 483041215955138008872756664816547928048923828396097294031679434714527143726794
Кандидат що не пройшов: 198987460719278497180642729819523899385890178488425520497771875811096839816328
Кандидат що не пройшов: 186461122015675797628039441620389279107567651376301048163051447951711864986866
Кандидат що не пройшов: 130145091237367538700507857475938718767553777160863698257033730056529288660128
Кандидат що не пройшов: 371219112377623204802250721876136801460903300767254599472280445559023749527340
Кандидат що не пройшов: 172643397540313866589751367483422843410150046065448787673726179050527465254399
Кандидат що не пройшов: 119916965580435466732434994786350493643818994542667309176378054610735406455799
Кандидат що не пройшов: 413226168662533184189814701006221780091927601432234586711684486466053360379395
Кандидат що не пройшов: 401546466598387113120341087107819639230363973222804797096923957492586351513039
Кандидат що не пройшов: 550450288410131473716972119529563574883942559308914683482730465663167703546524
Кандидат що не пройшов: 295811169620815367198683528477973648555909130318858403587921197420287063130303
Кандидат що не пройшов: 460944458834312181210591372099750374890498333422123539819791205852367752678160
Кандидат що не пройшов: 189331352942821331552675903718503229178785905699030336868681968316918087254150
Кандидат що не пройшов: 214514306752936693106080615515569115356800017803718455422654834215931883724352
Кандидат що не пройшов: 227203299978013213037729469474778857976407536222262222276363810254397784792672
Кандидат що не пройшов: 144919502397199934069142526005253603871300884739586356971418172273120933670398
```

```
Кандидат що не пройшов: 644304372920532633553355239323940266055534568801680487794198994674887783867405
Кандидат що не пройшов: 645647113263466713854027888809891170289769209208763080149263927362607240005694
Кандидат що не пройшов: 963702478217421103948156976452327935947537743846615884637096288307817608333457
Кандидат що не пройшов: 904329715586639379272098322549532924018225402019335089460107030275069344339969
Кандидат що не пройшов: 924865097519442552989562555142458182545136733779851561784341315464231039375 6
Кандидат що не пройшов: 883556869974679907124555588215788258274503903686806830451090312053461969569 86
Кандидат що не пройшов: 855918204072265870620909000012219934018915591390316181343421918348996656585139
Кандидат що не пройшов: 838465384166018862708092930014862548674922138759642696476053169040291233728546
Кандидат що не пройшов: 944550847540824257281352063965700973911520977811141572298663227519622385375984
Кандидат що не пройшов: 843300732605797448990099841276286945070205028392154372617581452544958084578444
Кандидат що не пройшов: 854492712382791168046240121966560786626261216407715037118808948776669472934 16
Кандидат що не пройшов: 755753754698126496201717966390337863419171508082127163642575423289342886218498
Кандидат що не пройшов: 841181530846583684862848561992481757318081060588116114264653177117238629143046

Перша пара простих чисел: p = 151881298001966348849900039073006382312290477723132549283051647385558815627469
q = 214457080595769982265577573706421080422726558522815175417053909571471370245673
Довжина p = 257 bit, q = 257 bit

Дурга пара простих чисел: p1= 822466581516203966732500288747426039689364099995560118614941399866038047258727
q1 = 900651489474756688425635506913329031891789223185175158563262967596593535725301
Довжина p1 = 259 bit, q1 = 259 bit
----------------------------------------------------------------------

Task3
```

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 .

Код

```
def GenerateKeyPair(p, q):
    n = p*q
    totient = (p-1)*(q-1)
    e = random.randint(2, totient - 1)
    (g, x, y) = ext_gcd(e, totient)
    while g > 1:
        e = random.randint(2, totient-1)
        (g, x, y) = ext_gcd(e, totient)
    d = mod_inverse(e, totient)
    public_key = (e, n)
    private_key = (d, n)
    return public_key, private_key
```

```
print("Task3")
print("-------------------------------------------------------------------------
-")
key1 = GenerateKeyPair(p1, q1)
key2 = GenerateKeyPair(p2, q2)

e1, d1 = key1
e2, d2 = key2

print(f"Публічний ключ: {e1}\nПриватний ключ: {d1}\n")
print(f"Публічний ключ1: {e2}\nПриватний ключ1: {d2}")
print("-------------------------------------------------------------------------
-\n")
```

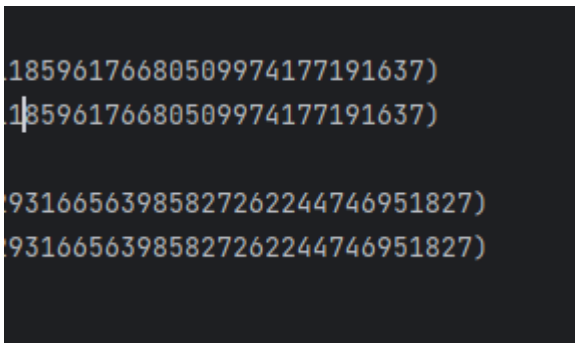
Результати:

```
Task3
------------------------------------------------------------------------
Публічний ключ: (218478427075996219147234107208433446265778786763525404449596237004093112544817627196142555089936716288758387518464398728046843106618361451
Приватний ключ: (290107615334134254909519280630844683201764914522321338932653140832874792386364315159838062561214221890570103638048715113074623738444077547

Публічний ключ1: (396709100204794514050932339316754475996933061590507773704816823458523361943640053523914771646417347882379895891469309694217304753604888088
Приватний ключ1: (624829075646543942044866919661190088727367310382147753421284985238641169345463948780652101242829449066837316654315685913187334967630892217
------------------------------------------------------------------------
```

```
614516683463483062374 5, 325720197665978556497153149051376453381647353965531789246984085441144209040279512094635777167794421247972307530280194418600597411187
077547533533406321494 5, 325720197665978556497153149051376453381647353965531789246984085441144209040279512094635777167794421247972307530280194418600597411187

888082423558197849964527, 740755751685780490846718843674605555847309369858434753736364124052210424053897774516610671359328106917802989906873195514332175293
892212121128413621120663, 740755751685780490846718843674605555847309369858434753736364124052210424053897774516610671359328106917802989906873195514332175293
```



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.

Код

```
def decimal_to_binary(decimal_number):
    binary_representation = bin(decimal_number)[2:]
    return binary_representation


def ConvertToInt_en(text):
    num = 0
    for char in text:
        num = num * 256 + ord(char)
    return num

def ConvertToText_en(number):
    text = ""
    while number > 0:
        char_code = number % len(alphabet)
        text = alphabet[char_code] + text
        number //= len(alphabet)
    return text

def ConvertToInt_dec(text):
    num = 0
    for char in text:
        num = num * len(alphabet) + alphabet.index(char)
    return num


def ConvertToText_dec(num):
    text = ""
    while num > 0:
        char_code = num % 256
        text = chr(char_code) + text
        num //= 256
    return text
```

```
def Encrypt(input_text, key):
    e, n = key
    if len(input_text)*8 >= len(decimal_to_binary(n)):
        blocks = []
        for i in range(0, len(input_text), 64):
            block = input_text[i:i + 64]
            blocks.append(block)
        enc_text = []
        for block in blocks:
            text_in_num = ConvertToInt_en(block)
            number_encrypted = pow(text_in_num, e, n)
            text_encrypted = ConvertToText_en(number_encrypted)
            enc_text.append(text_encrypted)
        return enc_text
    text_in_num = ConvertToInt_en(input_text)
    number_encrypted = pow(text_in_num, e, n)
    text_encrypted = ConvertToText_en(number_encrypted)
    return text_encrypted


def Decrypt(encrypted_text, key):
    d, n = key
    if isinstance(encrypted_text, list):
        dec_text = ""
        for block in encrypted_text:
            number_encrypted = ConvertToInt_dec(block)
            number_decrypted = pow(number_encrypted, d, n)
            text_decrypted = ConvertToText_dec(number_decrypted)
            dec_text += text_decrypted
        return dec_text
    number_encrypted = ConvertToInt_dec(encrypted_text)
    number_decrypted = pow(number_encrypted, d, n)
    text_decrypted = ConvertToText_dec(number_decrypted)
    return text_decrypted


def Sign(input_text, my_private_key, public_key):
    sha256_hash = hashlib.sha256()
    sha256_hash.update(input_text.encode('utf-8'))
    sha256_hash_value = sha256_hash.hexdigest()

    hash_encrypted_with_prv = Encrypt(sha256_hash_value, my_private_key)
    print(f"Signature: {hash_encrypted_with_prv}")

    hash_encrypted_with_pbl = Encrypt(hash_encrypted_with_prv, public_key)
    print(f"Signature encrypted with public key: {hash_encrypted_with_pbl}")
    return hash_encrypted_with_pbl


def Verify(input_text, sign, public_key, my_private_key):
    hash_decrypted_with_prv = Decrypt(sign, my_private_key)
    print(f"Signature decrypted with private key: {hash_decrypted_with_prv}")
    hash_decrypted_with_pbl = Decrypt(hash_decrypted_with_prv, public_key)
    print(f"Hash decrypted with public key(signature is verified):
{hash_decrypted_with_pbl}")

    sha256_hash_cal = hashlib.sha256()
    sha256_hash_cal.update(input_text.encode('utf-8'))
    sha256_hash_value = sha256_hash_cal.hexdigest()
    print("Hash calculated from received message: ", sha256_hash_value)

    if hash_decrypted_with_pbl == sha256_hash_value:
        return True
```

```
    else:
        return False
```

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.

Код

```
def create_message_for_abonent(public_key, length, my_private_key):
    characters = string.ascii_letters + string.digits + ' '
    random_message = ''.join(random.choice(characters) for i in
range(length))
    print("Generated message: ", random_message)
    encrypted_message = Encrypt(random_message, public_key)
    signed_message = Sign(random_message, my_private_key, public_key)
    message = (encrypted_message, signed_message)
    return message


def receive_message_from_abonent(message, my_private_key, public_key):
    enc_mes, signed_mes = message
    decrypted_message = Decrypt(enc_mes, my_private_key)
    print("Decrypted message: ", decrypted_message)
    ver_mes = Verify(decrypted_message, signed_mes, public_key,
my_private_key)
    if ver_mes:
        print("Whereas decrypted and calculated hashes match, then message
isn`t tampered\n")
    else:
        print("Message is tampered by someone\n")
```

```
print("Task4-5")
print("-----------------------------------------------------------------------
-")
print("From A to B")
message_from_A = create_message_for_abonent(e2, 200, d1)
print(f"Encrypted message and encrypted signature from A to B:
{message_from_A}\n")
receive_message_from_abonent(message_from_A, d2, e1)

print("From B to A")
message_from_B = create_message_for_abonent(e1, 201, d2)
print(f"Encrypted message and encrypted signature from B to A:
{message_from_B}\n")
receive_message_from_abonent(message_from_B, d1, e2)
print("-----------------------------------------------------------------------
-\n")
```

Опис кроків протоколу конфіденційного розсилання ключів з підтвердженням справжності:
Спочатку генерується повідомлення, потім воно зашифровується з публічним ключем отримувача, потім повідомлення(не зашифроване) шифрується з моїм приватним ключем(підписується) потім зашифроване моїм приватним ключем повідомлення, шифрується публічним ключем отримувача, щоб тільки він міг подивитися цей підпис. Далі зашифроване повідомлення та зашифрований підпис повертаються функцією як кортеж (зашифроване повідомлення, ашифрований підпис).

Результати:

Так як зашифровані повідомлення та підпис не влязли в скріншот, то ми написали їх нижче, спочатку від A до B, потім від B до A, інші дані видно на скріншоті, тому я їх не переписував

Encrypted message and encrypted signature from A to B:
(['1RodZKkgocChvfXK4IUH29JDPb8hcOg8b9rOx7lkRoZ7LvGKXGATd7XRGejhl5j6TacGoBMCL8Z8Z18UOJcNRb7',
'xmeUTuk0h7KZ16EigxKxaIEFRmZKGKwhly63I5rIRfNvKfSxGhK7ct7j0H7TmZXzzViNzerUVumqzkU0KowXt1v',
'my2uHIZwBiH3Al110GJEXUAv6yGuOW80tuRDNmsXG7x293XSMLoHhanrohlHjx3PuQdpPjdY7xLox5F1q7RJ8or',
'vNdM46F5czFzIH7nigX9Z8vWVtLkt9Uw74YXUSFCSolniWsOZpxIeUnuIJNKjXsO05DHGMbYRxsXswGZKZhkgIJ'],
['b0cbufWmfqljTWUmN8dFdMn197zp9Mqcvh2Hx6g9p7abJhPcaahn8f5Xey0Kl9AJ0Y4fAMsnM9lY59D3yJDj49W',
'r9IotDbocrIAH10VN3OeRd8ccRtZRofEZXYJULO9cG81ldOJhEaEH1oowVC0mKA8XExAL5NBAzni8ybD0spwmq4'])

Encrypted message and encrypted signature from B to A:
(['r6Ef4JOhrAD2skvqgVtiD59ZJHqjbMNPjFEe4xey5w3d6hFoibfAuuqw2ecjCP6PoBKr7UikVpPsTPY9pK4Uvq',
'b6Yqv6DLvtskYpWlGqGtRPTpFTguV37oVabcwLNeQeN6igv5mUCUoZWWKjqj9K1z63ThGgKBYKQnaxrcAUT6sTb',
'becWciIwAvdfWZygBLFyQ3tj95DPGrpIR0lJ1OHYgjJo2kZb4vtURv9zpPgLrK071ad8sgWgfa2YhjFuI9TGX63',
'bwwd3Wfv5FtChUCRsZLwsRA0XoAjlzbEsCuZMmd9anrd2K8IOLDFVWfbQth7Fc7vZRpVeArp2tbVB6reAiCRke6'],
['bWOMZJhl2znurrYCjCehms4VEqok1c8Sy8yT5CDY8PeHhRYT1MaW52Dlvp0KeoNk5xSDPtRBdKy1xWnkwnCTBvj',
'7ax2BXgaCaa4v1QqhTVrIZkvpkFYNtWYJIlWpk0vLuBH7mgReigTONr1xdp7zMpW1khAgV3Aoij6V4YdQS3JRG'])

Висновки:

У ході виконання даної роботи було досягнуто поставленої мети, яка передбачала ознайомлення з тестами перевірки чисел на простоту і методами генерації ключів для асиметричної криптосистеми типу RSA. Практичне засвоєння цих тестів та методів генерації ключів дозволило глибше зрозуміти принципи роботи криптосистеми RSA, що є однією з ключових технологій у сфері інформаційної безпеки. Також, робота надає можливість вивчити систему захисту інформації на основі криптосхеми RSA. Встановлення концепції захисту інформації з використанням даної системи, організація засекреченого зв'язку та електронного підпису, є важливим етапом для забезпечення конфіденційності та цілісності обмінюваної інформації.