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

```python
import random
import hashlib
import string

alphabet = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890"


def ext_gcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, x, y = ext_gcd(b % a, a)
        return (g, y - (b // a) * x, x)


def mod_inverse(a, m):
    g, x, y = ext_gcd(a, m)
    if g != 1:
        return None  # Оберненого елемента не існує
    else:
        return (x % m + m) % m


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(1, n - 1)
        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
        else:
            print(f"Кандидат що не пройшов: {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: 165785309102121456199988722832507394102261763166396824895741249124278485899697
Кандидат що не пройшов: 206358183712648068546664466883861975782868745896340518992449028690998749183183
Кандидат що не пройшов: 378934863580049544045848515051097105280658820721157922352797219088367274753044
Кандидат що не пройшов: 118190006063865614050455512277422180169266742460290341552811347834343694808238
Кандидат що не пройшов: 205492995456483138653792852311695744834021720942083966754633495035634745210744
Кандидат що не пройшов: 500602070543372274482185465755617730084933262702427920171619696860140371592385
Кандидат що не пройшов: 199308985088869367580535885527292202895681227214239898107910498995748550863810
Кандидат що не пройшов: 174201997183596298658057722847633225464072779752027601536962203475968080874283
Кандидат що не пройшов: 134082669568580684321982834361875841997340419335557911864896325668162080433202
Кандидат що не пройшов: 507246178024744053214936196475196177052409811745677457317042748682720183750845
Кандидат що не пройшов: 425442987575062089724281392422897153883533229255977515910143965620272750256565
Кандидат що не пройшов: 133189426924921057669855107091877378499108821462464902818486764117630234028880
Кандидат що не пройшов: 151162655648329216315422387195493732874303990667194265834290325654621232580305
Кандидат що не пройшов: 361566930513573769002453047308080643403070476167608382410594285404883710851311
Кандидат що не пройшов: 104966738673139222095317810368744431741606614026983473283759729238975986309235
Кандидат що не пройшов: 121067607631575739178183778119715782050970988915093478897805731873664535826158
Кандидат що не пройшов: 425083177321468591028742240655319260522423826558984768172910649457156213560664
Кандидат що не пройшов: 350836366834450230450975280034093747629328485477795229616102572879776855042375
Кандидат що не пройшов: 218319987168740751173403362045706791665555403118706926184865169576052960333660
Кандидат що не пройшов: 548275263922482980057573045924059167085451616836663682969699717780367434734629
Кандидат що не пройшов: 549399649022128383153670875155890482149784296092997081574077882713895688633515
Кандидат що не пройшов: 526447087206374457659137305731007786602685743111866975027189662586766803775130
```

```
Кандидат що не пройшов: 846746490605561332525940144508597510206806285721989687597700966802957077626851
Кандидат що не пройшов: 649655134077855094438907238056377664438314069920268626897899494529897259453175
Кандидат що не пройшов: 860264620926951639060545753633337967106377333803102256781739523528038198707828
Кандидат що не пройшов: 622341670908836874703470173539453475101790850332897338341758887878219804687430
Кандидат що не пройшов: 763062714665165718678824124378152732355861389338613528110120742304370274077258

Перша пара простих чисел: p = 110373357500526303233555796388783146933465210631878464547759889511657661144503
q = 313016789656339641576464310305017841826316374044420469262678615500007590287699
Довжина p = 256 bit, q = 258 bit

Дурга пара простих чисел: p1= 903544538287603794826950705744316035926568636235768112123875271241679781140753
q1 = 778100665571379841807350286627543277110433723354696269394118676432123348719061
Довжина p1 = 259 bit, q1 = 259 bit
----------------------------------------------------------------------
```

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: (24692976311073757404927843301467559979817124239654178826232240699111048788643037753145652591857879752399940452104375876690076068641457022
Приватний ключ: (86861714354812330719975022601376127429825823285409692522160970359754601895736240484374888839182744449765226377640169317955794031109567320
1

Публічний ключ1: (38851803744250369451914367304933014513375052147022724150427921369997917349768918510793656616770723156805910464293432085842129557676629736
Приватний ключ1: (25076233941013078075789790511274926926208529143720961459456999351083690690366241823477249389850802792812398734328459934513888868954035704
----------------------------------------------------------------------
```

```
60686414570224190612955122218l7, 345487140284062191496943450209139101560290169090743758611245706570416015398029196172678960704924978670003133779965282620858
40311095673201063730635384952l, 345487140284062191496943450209139101560290169090743758611245706570416015398029196172678960704924978670003133779965282620858

29557676629736256392890916472563l, 703048606614969609364040314588339518868112644538483036931027005796570028652112001425018665984917409731691997948236230470
88868954035704777449467060749ll37, 70304860661496960936404031458833951886811264453848303693102700579657002865211200142501866598491740973169199794823623047O
```

```
28262085825440800559409858984231382368597)
8262085825440800559409858984231382368597)

362304706283917249890025877909507078949929333)
36230470628391724989002587790950707894992933)
```

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 ConvertToInt_en(text, hash=False):
    num = 0
    if not hash:
        for char in text:
            num = num * 50000 + ord(char)
    else:
        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:
        if char in alphabet:
            num = num * len(alphabet) + alphabet.index(char)
    return num


def ConvertToText_dec(num, hash=False):
    text = ""
    if not hash:
        while num > 0:
            char_code = num % 50000
            text = chr(char_code) + text
            num //= 50000
    else:
        while num > 0:
            char_code = num % 256
            text = chr(char_code) + text
            num //= 256
    return text


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


def split_blocks(text, max_block_size):
    blocks = []

    for i in range(0, len(text), 32):
        block = text[i:i + 32]
        blocks.append(block)

    for i in range(len(blocks)):
        while len(decimal_to_binary(ConvertToInt_en(blocks[i]))) >
```

```
max_block_size:
            half_size = len(blocks[i]) // 2
            block1 = blocks[i][:half_size]
            block2 = blocks[i][half_size:]
            blocks[i] = block1
            blocks.insert(i + 1, block2)

    return blocks


def Encrypt(input_text, key, hash=False):
    e, n = key
    if hash:
        text_in_num = ConvertToInt_en(input_text, hash)
        number_encrypted = pow(text_in_num, e, n)
        text_encrypted = ConvertToText_en(number_encrypted)
        return text_encrypted

    mes_len_in_num = len(decimal_to_binary(ConvertToInt_en(input_text)))

    if mes_len_in_num >= len(decimal_to_binary(n)):
        blocks = split_blocks(input_text, len(decimal_to_binary(n)) - 1)
        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, hash=False):
    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, hash)
            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, hash)
    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()
    print(sha256_hash_value)
    hash_encrypted_with_prv = Encrypt(sha256_hash_value, my_private_key,
True)
    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,
True)
    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 + ' ' +
'абвгдеёжзийклмнопрстуфхцчшщъыьэюя' + 'ґєіїҐЄІЇ' + '你好' + '中文' + ' ')
    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")
```

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 ми їх не переписували

Encrypted message and encrypted signature from A to B:
(['Lgebm3ZDSPKssPSPwQzrpHDeVxgyBGYGrwZqow2AmVEVnXuHvwfrtzAVXO350gb6y9Z0wHHrNIieYTwExpem8on',
'rB81d7sd7tXvLluIald4GrkEeVUX71IQuONLTYQe6MuvE710YsLY8GY4CbMCnJLbBzlOXpgmHsxwhuw8squOwMn',
'MkSVrHUsD6eoVWxqa1YBqW26vHTsyX0IlDn8oK3o1SZqfr5w3YlJglL6nkbhEcAN21utoRSENQpp39gZF2GmmoZ',
'jeGcSikkV3J9ZlKzzjts1O0yiFzoyP0vbntxjG5TF5rrVjZd7nIZFW8Ai7JusEPuYYT2AnBU9xvUTKVVZ5UaVmA',
'JpZOscHDWy56ZCXJNPH5LLFrNRPgqn9QCKP2b3QikrHeHfaNqJMQNNnLUoDNpMMUfSbELHTlS2SI8hnJZIW0yMT',
'Mx5BpCJS3nEdKdM93jdNfiesF4n6DHVWNaoG2K0WoaFzRepR6zyV2FyAeZoLcqAtK98f1yDMfC8vr3kzzDCqmsY',
'pBxLAlSPtgmAG6aGRqwTXgeaxa4uFRPlBzu4F1e3KxpK5FNkmXyXw5jLGG9PYa5AFyJsm6GJ5QtRxyUZOkeCC3b',
'w3kdkBbfmJBJQlGFnkwSMciTQt5XPGgG9ugX1XL3YajlDibn6kc76iv0NHQltidR9sVKYVHpo56ExwAQXnQgKeZ',
'nloFHwaG9aJIyw4RRKE03eL0o06Daz11kG6Gy0ySpG9SPuTGyxNLY2ISkgVH0ETu

gj6pyyuZhHkJBeuEwqLtsyq',
'uKIzGMAyiMzmtBsK4fA93dgdssjf0hpyfAGd9LG0mXscoB8O6DO4nlvfxCbTA6l4EY
9AmtJPTXwoJ1m1oSvHgjp'],
['SLVWF0ggC3YVdi00MTRU9zKZWqni2WMy4xnwClCZ93DAD1MBzUyMAUZQg
YVRIv2HYtxfLkQoRCidjicwWPEle8d',
'rhVbbIxNCYdL9TSyH12pZWshaZOjUb3BWzPySIS0Az091zznd8OnosMNPNBvMbg
CZJxTx4sffRLNJieon4HzoMV',
'LqscU7qGwVd8lARAIlFKlHnlKlS40BnjgT0P9iwlGfUUddWqMwu5tkpy6TpVFKkD1
2wZxqVGQcIKX2ZhFTlnXZa'])

Encrypted message and encrypted signature from B to A:
(['sAcylMJoyFefC6AwKpJSSrnKJ7gCjSzAPexCKFS1eXI6KuP23jcvdfFSWncXh8thrL4
A3IfLVlr6k1mvPr2KvN',
'JsiUxa03vwgsxSfPilULMm8Ws4RirUs7YnQrjmD3xm7fcWOV9T4BnoQb13CAd4ayd5
8vaol7aiCI92DHzEFUuH',
'btMpVpMw9LqPu3Cxm6yn0IwGE58S9SHwHsB72fwAMHCdzaAZvjgx5tMVnLzsC93
Dmp6v1AJ8fdBkKqzqr3QgOGS',
'u7pphiczy05VQvFe7Ay7BD6RFUFRdoUM2ZNuIpsgFjrkcMKyaDG3UOkez4y02AGiz
Em11lBepgU75r2agnsQkB',
'bPwgXHZJehfscovtkZMzyeSj81yL6OBMVW4ogxdeDPe6U2492G5rpBKWnFbP1Z3Rt
AaUUP2QcsLEpMKCmYQXgIc',
'DiyYRieSUQ68JFK3ht1JfG2xgu7Z4uBiCduFhz9Dqe96WOb263tYJGPwr8Yf8eKcNZJ
WXODGRrI1udgnOpqwcY',
'3LrsXarD73DXSJ653xH0xZ37uNhDRFbTerxH13FiXCK5jIjOMYSXtw7qShEuQzYpX
0J2nVHy2KIT6T6MHIHxY8',
'bHl24eWHRhf6N2cZIe5th0eDEaGQqQvRnDudhgrmpzQxmnSFN8bKTqkWo8Cn2urph
NefgeD5kQ7RHyRYm2sPD53',
'cqSzF4HcEpgc2Qlj2n76LR1lEMSjYd4r5Pdd0t0zI1iDmkySCeljoSWI2AoUVtvX3YWK
1ASAlz2WsE22s7z7T8X',
'3ZXVup6tbwBO3th44QsbMnJBHTPqK7U2YPSzGfBiwtlslly0vRvqm2dif3oqDnkxqvS
MwB7Ss9LNm2pewvsnwt'],
['cmm4ccLUXkLgaXWe83YcLbwRp2E907RtUPrPfXOpvuxapxgEFJ1kln0vBW0ZeGEE
UB6cdKexRU93jvUYIkewqBq',
'UZXV7BItcVveh6MlqyaXlpRZAjqSQQWLFHM2rt2WBsPKjV5zKejGKO2Nhxh6zjMb

921tmxY3nySmSpDBuTnoPS',
'Prl3bnth6swNm0NPI83AeFwO32y0WklO2lpkwR1m6Rba9vUjwn84E7R8A9KVDIei9i5mtW281g5vZOxPSa2259'])

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.