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):
    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(100000000000000000000000000000000000000000000000
000000000000000000000000000000,
55555555555555555555555555555555555555555555555555555555555555555555555555555
5)
q1 =
generate_random_prime_number(100000000000000000000000000000000000000000000000
000000000000000000000000000000,
55555555555555555555555555555555555555555555555555555555555555555555555555555
5)

p2 =
generate_random_prime_number(555555555555555555555555555555555555555555555555
555555555555555555555555555555,
99999999999999999999999999999999999999999999999999999999999999999999999999999
9)
q2 =
generate_random_prime_number(555555555555555555555555555555555555555555555555
555555555555555555555555555555,
99999999999999999999999999999999999999999999999999999999999999999999999999999
9)

print(f"\nПерша 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
------------------------------------------------------------------------


Перша пара простих чисел: p = 127161297882589062150910945520919936121243405271010561426960564029248375546703
q = 5220391214158581056203349121532227466743813883520416962505669813742527192822333
Довжина p = 257 bit, q = 259 bit

Дурга пара простих чисел: p1= 901791489289234296562874903826641057218267344666185136587527012100656612276373
q1 = 7429070553506422784462486536617646190535565708934119375932249253374394222642339
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

```python
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
```

```python
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: (391848193593977842392447968125600297257578513117492530497760390309957367324947956332865270806401501341935034193641980960565135757364827355
Приватний ключ: (595508097934489780137456746438359590310120493013956238328776947920892226102090316801101266889295330489150008315822331510610509901571329349

Публічний ключ1: (268180583743291096357345264145420367346265635023600064224157392489868028581641293218729793172776121709554969293815069032912768632163279 65
Приватний ключ1: (161563373933844948693221760638194403115013277881429878035334174614341621722768648868565363867997392134440027910452710860974397942790095 47
--------------------------------------------------------------------
```

```
135757364827355372523821691200 93, 768598379946459442840830931893374065477770625486958687688978109738790836318765994871517443975847135735113404867981725 9
509901571329349906449870400730 45, 768598379946459442840830931893374065477770625486958687688978109738790836318765994871517443975847135735113404867981725 9

276863216327965567407446892547 9089, 853036013574626648185791779291329283946078342409422626693753876048132802664770242577703020647420810868818889450402990
439794279009547136036417693394 8249, 853036013574626648185791779291329283946078342409422626693753876048132802664770242577703020647420810868818889450402990
```

4. Написати програму шифрування, розшифрування і створення повідомлення з цифровим підписом для абонентів A і B. 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 A и B, перевірити правильність розшифрування. Скласти для A і B повідомлення з цифровим підписом і перевірити його.

Код

```
def decimal_to_hexadecimal(decimal_num):
    decimal_num = int(decimal_num)
    hexadecimal_num = hex(decimal_num)
    return hexadecimal_num[2:]


def hex_to_decimal(hex_string):
    decimal_result = int(hex_string, 16)
    return decimal_result


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 Encrypt(input_text, key, server = False):
    e, n = key
    if server:
        number_decrypted = pow(input_text, e, n)
        number_decrypted = decimal_to_hexadecimal(number_decrypted)
        return number_decrypted
    input_text = int(input_text)
    text_encrypted = pow(input_text, e, n)
    text_encrypted = str(text_encrypted)
    return text_encrypted


def Decrypt(encrypted_text, key, server = False):
    d, n = key
    if server:
        encrypted_text = str(encrypted_text)
        encrypted_text = encrypted_text.lower()
        number_encrypted = hex_to_decimal(encrypted_text)
        number_decrypted = pow(number_encrypted, d, n)
        return number_decrypted
    encrypted_text = int(encrypted_text)
    text_decrypted = pow(encrypted_text, d, n)
    text_decrypted = str(text_decrypted)
    return text_decrypted


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

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


def Verify(input_text, sign, public_key):
    hash_decrypted_with_pbl = Decrypt(sign, public_key)
    hash_decrypted_with_pbl = decimal_to_hexadecimal(hash_decrypted_with_pbl)
    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

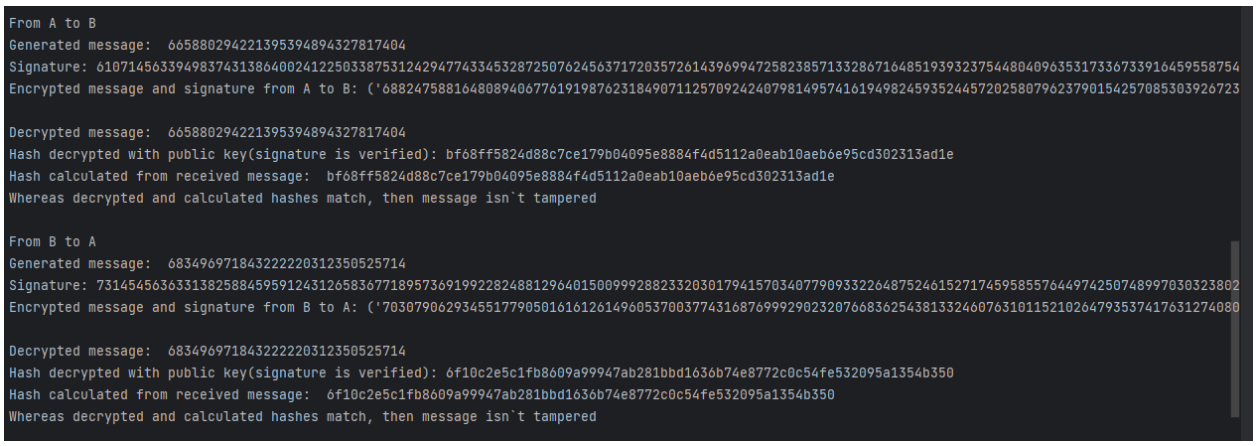
```
def create_message_for_abonent(public_key, length, my_private_key):
    characters = string.digits
    random_message = ''.join(random.choice(characters) for i in
range(length))
    print("Generated message: ", random_message)
    encrypted_message = Encrypt(random_message, public_key)
    signed_message = Sign(random_message, my_private_key)
    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)
    if ver_mes:
        print("Whereas decrypted and calculated hashes match, then message
isn`t tampered\n")
    else:
        print("Message is tampered by someone\n")

def Send_key(public_key, my_private_key):
    d, n = my_private_key
    k = random.randint(0, n)
    print("Generated secret k: ", k)
    encrypted_message = Encrypt(k, public_key)
```

```
    signed_message = Sign(k, my_private_key)
    signed_message_enc = Encrypt(signed_message, public_key)
    message = (encrypted_message, signed_message_enc)
    return message


def Receive_key(message, my_private_key, public_key):
    enc_mes, signed_mes_enc = message
    decrypted_message = Decrypt(enc_mes, my_private_key)
    print(f"\nDecrypted secret key: {decrypted_message}", )
    signed_mes_dec = Decrypt(signed_mes_enc, my_private_key)
    ver_mes = Verify(decrypted_message, signed_mes_dec, public_key)
    if ver_mes:
        print("Whereas decrypted and calculated hashes match, then message
isn`t tampered\n")
        return decrypted_message
    else:
        print("Message is tampered by someone\n")
```

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

Результати ось цього коду:

```
def create_message_for_abonent(public_key, length, my_private_key):
    characters = 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)
    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)
    if ver_mes:
        print("Whereas decrypted and calculated hashes match, then message
isn`t tampered\n")
    else:
        print("Message is tampered by someone\n")
```



Так як зашифровані повідомлення та підпис не влізли 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 signature from A to B:
('688247588164808940677619198762318490711257092424079814957416194982459352445720258079623790154257085303926723003271251950828562935943541440645177158970999 44',
'610714563394983743138640024122503387531242947743345328725076245637172035726143969947258238571332867164851939323754480409635317336733916459558754031244684 00')

Encrypted message and signature from B to A:
('703079062934551779050161612614960537003774316876999290232076683625438133246076310115210264793537417631274080089930338700828348324656438578077492674450368 65',
'731454563633138258845959124312658367718957369199228248812964015009992882332030179415703407790933226487524615271745958557644974250748997030323802278314269976')

Ось результати виконання коду самого протоколу конф. розсилання ключів:

```
def Send_key(public_key, my_private_key):
    d, n = my_private_key
    k = random.randint(0, n)
    print("Generated secret k: ", k)
    encrypted_message = Encrypt(k, public_key)
    signed_message = Sign(k, my_private_key)
    signed_message_enc = Encrypt(signed_message, public_key)
    message = (encrypted_message, signed_message_enc)
    return message


def Receive_key(message, my_private_key, public_key):
    enc_mes, signed_mes_enc = message
    decrypted_message = Decrypt(enc_mes, my_private_key)
    print(f"\nDecrypted secret key: {decrypted_message}", )
    signed_mes_dec = Decrypt(signed_mes_enc, my_private_key)
```

```
    ver_mes = Verify(decrypted_message, signed_mes_dec, public_key)
    if ver_mes:
        print("Whereas decrypted and calculated hashes match, then message
isn`t tampered\n")
        return decrypted_message
    else:
        print("Message is tampered by someone\n")
```

```
Confidential key distribution protocol
Generated secret k:  2189481821777746461448875632793734319091835345627076765657480781086254245597027793161574775710831373859639840265664708756829606288228
Signature: 4001786759030784978541522459874260732014034886277690020573192388718473687485732438898525265148759455779876570113247111559717499479630035338501650
Encrypted key and encrypted signature: ('4093343518434366576025749997009174715899548614319508578303095021052753126856829220306906692830444053334523619085750

Decrypted secret key: 2189481821777746461448875632793734319091835345627076765657480781086254245597027793161574775710831373859639840265664708756829606288228
Hash decrypted with public key(signature is verified): 1fc15e35b16c63e96910cb2b5437b8f3e8e2cb152095d7bf4fbc74c1bd96f959
Hash calculated from received message:  1fc15e35b16c63e96910cb2b5437b8f3e8e2cb152095d7bf4fbc74c1bd96f959
Whereas decrypted and calculated hashes match, then message isn`t tampered

------------------------------------------------------------------
```

Та перевірку на сайті:

Тут ми спочатку генеруємо ключі потім перетворюємо публічний ключ у шістнадцяткове число потім на сервері вводимо шістнадцятковий ключ тобто модуль і сам ключ і шістнадцяткове число далі на сервері ми зашифровуємо його і потім у себе локально за допомогою нашого приватного ключа його розшифровуємо

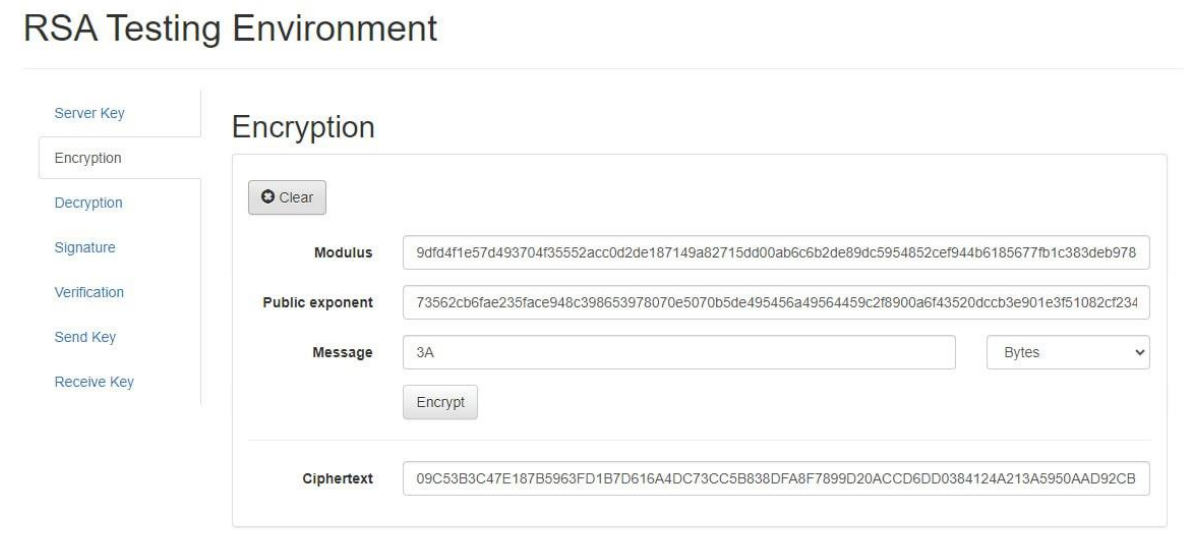
Ось код що генерує ключі та виводить спочатку обидва в десяткових значеннях , а потім виводить публічний ключ в шістнадцятковому виді для сервера(деякі рядки, які не треба закоментовані):

```
print("TESTING ENCRYPTION AND DECRYPTION WITH REMOTE SERVER ")
print("------------------------------------------------------------------------
------------------------------------------------------------------------")

p =
generate_random_prime_number(10000000000000000000000000000000000000000000000
000000000000000000000000000000,
55555555555555555555555555555555555555555555555555555555555555555555555555
5)
q =
generate_random_prime_number(10000000000000000000000000000000000000000000000
000000000000000000000000000000,
55555555555555555555555555555555555555555555555555555555555555555555555555
5)

key = GenerateKeyPair(p, q)


e, d = key
print(f"Публічний ключ: {e}\nПриватний ключ: {d}\n")
# d =
(110585553162052249205688721910964264929692967903476586088856811226853347466
333143035782163053632133095308165839150541176809746726117551443408493781893312
1,
132393294799818189097283724580025322476457628221172300060096806533776543743259
98787038470048434448099371719041938240034215825141298681741956369195410928307
87)
```

```
# print(d)
e1, n1 = e
e1 = decimal_to_hexadecimal(e1)
n1 = decimal_to_hexadecimal(n1)
print(e1, n1)

# num_enc_hex =
"09C53B3C47E187B5963FD1B7D616A4DC73CC5B838DFA8F7899D20ACCD6DD0384124A213A5950
AAD92CBD94D018F45A6977BC124DD9DC67CEFD9B977F87FDF1D6DD"
# print("Num encrypted in hex: ", num_enc_hex)
# num_dec_hex = Decrypt(num_enc_hex, d, True)

# print("Num decrypted: ", decimal_to_hexadecimal(num_dec_hex))
print("-------------------------------------------------------------------------
-------------------------------------------------------------------------")
```

```
D:\Криптографія\Lab4\venv\Scripts\python.exe D:\Криптографія\Lab4\test3.py
TESTING ENCRYPTION AND DECRYPTION WITH REMOTE SERVER
-------------------------------------------------------------------------------------------------------------------------------
Публічний ключ: (96650702842233722242084129278777814972070497878366083743817123590176392315120666841974822489589200148152645933443244568029960526343365486
Приватний ключ: (11058555316205224920568872191096426492969296790347658608885681122685334734663331430357821630536321330953081658391505411768097467261175514

73562cb6fae235face948c398653978070e5070b5de495456a49564459c2f8900a6f43520dccb3e901e3f51082cf234e79bbf81100a63cec2831230b5a8c21de9 9dfd4f1e57d493704f35552ac

Process finished with exit code 0
```

```
-----------------
64593344324456802996052634336548685793235343468900l, 13239329479981818909728372458002532247645762822117230006009680653377654374325987870384700484
08165839150541176809746726117551443408493781893312l, 13239329479981818909728372458002532247645762822117230006009680653377654374325987870384700484

a8c21de9 9dfd4f1e57d493704f35552acc0d2de187149a82715dd00ab6c6b2de89dc5954852cef944b6185677fb1c383deb9780c632496a120e78b1c6a36d488fd343ce43
```



Ось код який розшифровує число, яке сайт зашифрував моїм публічним ключем(непотрібні рядки закоментовані)**:**

```
print("TESTING ENCRYPTION AND DECRYPTION WITH REMOTE SERVER ")
print("-------------------------------------------------------------------------
-------------------------------------------------------------------------")
```

```
# p =
generate_random_prime_number(100000000000000000000000000000000000000000000000
0000000000000000000000000000,
55555555555555555555555555555555555555555555555555555555555555555555555555555
5)
# q =
generate_random_prime_number(100000000000000000000000000000000000000000000000
0000000000000000000000000000,
55555555555555555555555555555555555555555555555555555555555555555555555555555
5)
#
# key = GenerateKeyPair(p, q)


# e, d = key
# print(f"Публічний ключ: {e}\nПриватний ключ: {d}\n")
d =
(1105855531620522492056887219109642649296929679034765860888568112268533473466
3331430357821630536321330953081658391505411768097467261175514434084937818933l
21,
1323932947998181890972837245800253224764576282211723000600968065337765437432 5
9878703847004843444809937171904193824003421582514129868174195636919541092830 7
87)
print(d)
# e1, n1 = e
# e1 = decimal_to_hexadecimal(e1)
# n1 = decimal_to_hexadecimal(n1)
# print(e1, n1)

num_enc_hex =
"09C53B3C47E187B5963FD1B7D616A4DC73CC5B838DFA8F7899D20ACCD6DD0384124A213A5950
AAD92CBD94D018F45A6977BC124DD9DC67CEFD9B977F87FDF1D6DD"
print("Num encrypted in hex: ", num_enc_hex)
num_dec_hex = Decrypt(num_enc_hex, d, True)

print("Num decrypted: ", decimal_to_hexadecimal(num_dec_hex))
print("----------------------------------------------------------------------
---------------------------------------------------------------------")
```

```
TESTING ENCRYPTION AND DECRYPTION WITH REMOTE SERVER
------------------------------------------------------------------------------------------------------------------------------------
(110585553162052249205688721910964264929692967903476586088856811226853347346633314303578216305363213309530816583915054117680974672611755144340849378189331 2
Num encrypted in hex:  09C53B3C47E187B5963FD1B7D616A4DC73CC5B838DFA8F7899D20ACCD6DD0384124A213A5950AAD92CBD94D018F45A6977BC124DD9DC67CEFD9B977F87FDF1D6DD
Num decrypted:  3a
------------------------------------------------------------------------------------------------------------------------------------
```

Як бачимо цифри збігаються ,що означає правильність роботи нашої реалізацій алгоритму з довільно вибраним корректним джерелом,у нашому випадку – навчальним сайтом.

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.

ПРИМІТКА ПО КОДОВІ:

На гітхаб ви просили також прикріпити код який шифрує і текст і також великий текст який розбиває на блоки, тому ми його також прикріпили, основний код це той який шифрує тільки числа