НТУУ "КПІ ім 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-14

Гавриленко Давид

Земляний Олександр

Перевірила:

Селюх П.В .

Київ 2023

Варіант-1

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:

Програма Lab4.ipynb на мові Python3

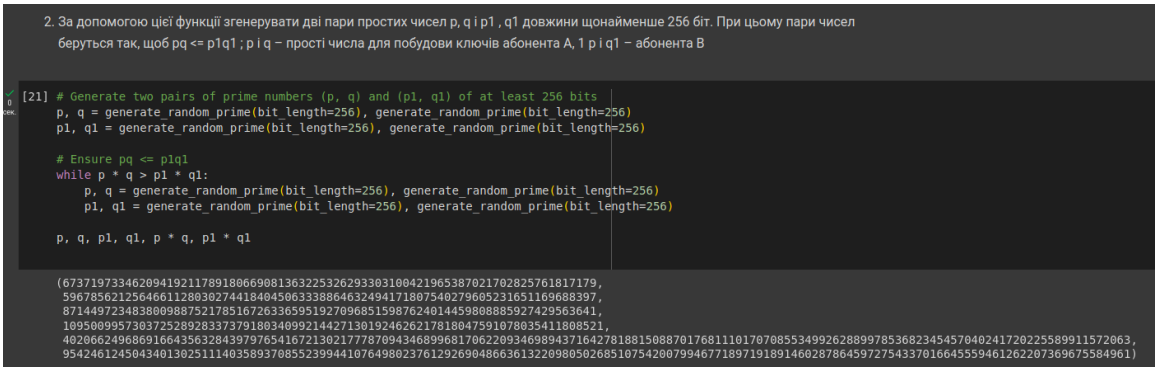
1. Написати функцію пошуку випадкового простого числа з заданого інтервалу або заданої довжини, використовуючи датчик випадкових чисел та тести перевірки на простоту. В якості датчика випадкових чисел використовуйте вбудований генератор псевдовипадкових чисел вашої мови програмування. В якості тесту перевірки на простоту рекомендовано використовувати тест Міллера-Рабіна із попередніми пробними діленнями. Тести необхідно реалізовувати власноруч, використання готових реалізацій тестів не дозволяється

```python
import random

# Function to check if a number is probably prime using Miller-Rabin test with preliminary divisions
def is_prime(n, k=10):
    # Handle small numbers directly
    if n in (2, 3):
        return True
    if n <= 1 or n % 2 == 0:
        return False

    # Preliminary divisions by first few primes
    for p in [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]:
        if n % p == 0 and n != p:
            return False

    # Write n as 2^r * d + 1 with d odd
    r, d = 0, n - 1
    while d % 2 == 0:
        r += 1
        d //= 2

    # Witness loop
    for _ in range(k):
        a = random.randint(2, n - 2)
        x = pow(a, d, n)
        if x == 1 or x == n - 1:
            continue

        for _ in range(r - 1):
            x = pow(x, 2, n)
            if x == n - 1:
                break
        else:
            return False

    return True

# Function to generate a random prime number within a given range or of a given bit length
def generate_random_prime(min_val=None, max_val=None, bit_length=None):
    if bit_length is not None:
        min_val = 2 ** (bit_length - 1)
        max_val = 2 ** bit_length - 1

    # Ensure min_val and max_val are set
    if min_val is None or max_val is None:
        raise ValueError("Either range or bit length must be specified")

    # Generate prime within the range
    while True:
        candidate = random.randint(min_val, max_val)
        if is_prime(candidate):
            return candidate

# Example usage
generate_random_prime(bit_length=16)  # Generate a random prime number with a bit length of 16
```

```
45691
```

45691

2. За допомогою цієї функції згенерувати дві пари простих чисел p, q і p1 , q1 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




```
(673719733462094192117891806690813632253262933031004219653870217028257618
17179,

596785621256466112803027441840450633388646324941718075402796052316511696
8397,
871449723483800988752178516726336595192709685159876240144598088859274295
3641,

109500995730372528928337379180340992144271301924626217818047591078035411
808521,
402066249686916643563284397976541672130217778709434689968170622093469894
371642781881508870176811101707085534992628899785368234545704024172022558
9911572063,

954246124504340130251114035893708552399441076498023761292690486636132209
805026851075420079946771897191891460287864597275433701664555946126220736
9675584961)
```

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) , (e1, n1) та секретні d і d1

```
[25] def generate_rsa_key_pair(p, q, e=65537):
        n = p * q
        phi = (p - 1) * (q - 1)

        # Ensure that e and phi are coprime
        if gcd(e, phi) != 1:
            raise ValueError("e and phi are not coprime.")

        # Calculate d
        d = pow(e, -1, phi)

        # The public key is (e, n) and the secret key is (d, p, q)
        return (e, n), (d, p, q)

    # Generate RSA key pairs for subscribers A and B
    public_key_A, secret_key_A = generate_rsa_key_pair(p, q)
    public_key_B, secret_key_B = generate_rsa_key_pair(p1, q1)

    (public_key_A, secret_key_A, public_key_B, secret_key_B)
```

```
((65537,
  4020662496869166435632843979765416721302177787094346899681706220934698943716427818815088701768111017070855349926288997853682345457040241720225589911572063),
 (17211603575028528060741174886296102372115384869147489551027704659707185081093247791947243634461281252570470479314704587997896719318626501707780669645507 93,
  6737197334620941921178918066908136322532629330310042196538702170282576181 79,
  5967856212564661128030274418404506333886463249417180754027960523165116968 8397),
 (65537,
  9542461245043401302511140358937085523994410764980237612926904866361322098050268510754200799467718971918914602878645972754337016645559461262207369675584961),
 (7549576812418944375470385085049801550133542604059587748944873479230747636281219187506359480697867518390526789991766333611552021815652617263121693753747 3,
  8714497234838009887521785167263365951927096851598762401445980888592742956364 1,
  1095009957303725289283373791803409921442713019246262178180475910780354118085 21))
```

```
((65537,
```

```
402066249686916643563284397976541672130217778709434689968170622093469894371642781881508870176811101707085534992628899785368234545704024172022558991157206 3),
```

```
(172116035750285280607411748862961023721153848691474895510277046597071850810932477919472436344612812525704704793147045879978967193186265017077806696455079 3,
```

```
673719733462094192117891806690813632253262933031004219653870217028257618171 79,
```

```
596785621256466112803027441840450633388646324941718075402796052316511696883 97),
```

```
 (65537,
```

```
954246124504340130251114035893708552399441076498023761292690486636132209805026851075420079946771897191891460287864597275433701664555946126220736967558496 1),

(754957681241894437547038508950498015501335426040595877489448734792307476362812191875063594806978675183905267899917663336115520218156526172631216937537473,

871449723483800988752178516726336595192709685159876240144598088859274295636641,

109500995730372528928337379180340992144271301924626217818047591078035411808521))
```

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

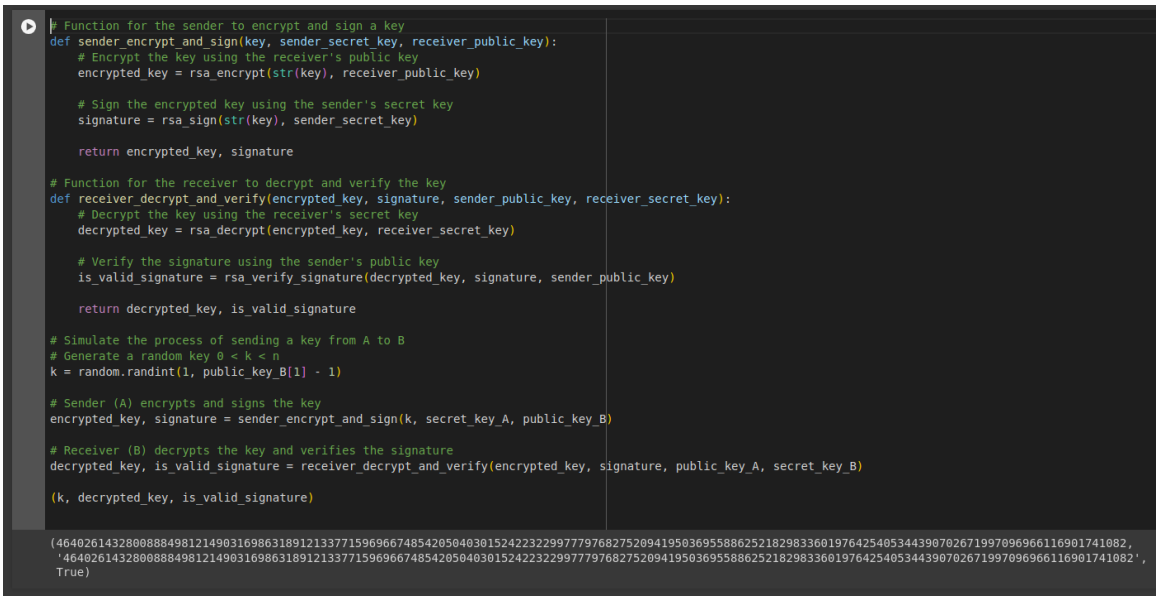
За допомогою датчика випадкових чисел вибрати відкрите повідомлення M і знайти криптограму для абонентів А и В, перевірити правильність розшифрування. Скласти для А і В повідомлення з цифровим підписом і перевірити його

```python
# Function to encrypt a message using RSA
def rsa_encrypt(message, public_key):
    e, n = public_key
    encrypted_message = [pow(ord(char), e, n) for char in message]
    return encrypted_message

# Function to decrypt a message using RSA
def rsa_decrypt(encrypted_message, secret_key):
    d, p, q = secret_key
    n = p * q  # Calculate n from p and q

    decrypted_message = ''.join([chr(pow(char, d, n)) for char in encrypted_message])
    return decrypted_message

# Function to create a digital signature using RSA
def rsa_sign(message, secret_key):
    d, p, q = secret_key
    hash_of_message = simple_hash(message)
    signature = pow(hash_of_message, d, p * q)
    return signature

# Function to verify a digital signature using RSA
def rsa_verify_signature(message, signature, public_key):
    e, n = public_key
    hash_of_message = simple_hash(message)
    hash_from_signature = pow(signature, e, n)
    return hash_of_message == hash_from_signature
```

```python
# Generating a random message
random_message = "Hello, this is a secure message"

# Encrypt the message for both A and B
encrypted_message_A = rsa_encrypt(random_message, public_key_A)
encrypted_message_B = rsa_encrypt(random_message, public_key_B)

# Decrypt the message for both A and B
decrypted_message_A = rsa_decrypt(encrypted_message_A, secret_key_A)
decrypted_message_B = rsa_decrypt(encrypted_message_B, secret_key_B)

# Create a digital signature for both A and B
signature_A = rsa_sign(random_message, secret_key_A)
signature_B = rsa_sign(random_message, secret_key_B)

# Verify the digital signature for both A and B
verification_A = rsa_verify_signature(random_message, signature_A, public_key_A)
verification_B = rsa_verify_signature(random_message, signature_B, public_key_B)

(decrypted_message_A, decrypted_message_B, verification_A, verification_B)
```

```
('Hello, this is a secure message',
 'Hello, this is a secure message',
 True,
 True)
```

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



```python
# Function for the sender to encrypt and sign a key
def sender_encrypt_and_sign(key, sender_secret_key, receiver_public_key):
    # Encrypt the key using the receiver's public key
    encrypted_key = rsa_encrypt(str(key), receiver_public_key)

    # Sign the encrypted key using the sender's secret key
    signature = rsa_sign(str(key), sender_secret_key)

    return encrypted_key, signature

# Function for the receiver to decrypt and verify the key
def receiver_decrypt_and_verify(encrypted_key, signature, sender_public_key, receiver_secret_key):
    # Decrypt the key using the receiver's secret key
    decrypted_key = rsa_decrypt(encrypted_key, receiver_secret_key)

    # Verify the signature using the sender's public key
    is_valid_signature = rsa_verify_signature(decrypted_key, signature, sender_public_key)

    return decrypted_key, is_valid_signature

# Simulate the process of sending a key from A to B
# Generate a random key 0 < k < n
k = random.randint(1, public_key_B[1] - 1)

# Sender (A) encrypts and signs the key
encrypted_key, signature = sender_encrypt_and_sign(k, secret_key_A, public_key_B)

# Receiver (B) decrypts the key and verifies the signature
decrypted_key, is_valid_signature = receiver_decrypt_and_verify(encrypted_key, signature, public_key_A, secret_key_B)

(k, decrypted_key, is_valid_signature)
```

```
(4640261432800888498121490316986318912133771596966748542050403015242232299777976827520941950369558862521829833601976425405344390702671997096966116901741082,
 '4640261432800888498121490316986318912133771596966748542050403015242232299777976827520941950369558862521829833601976425405344390702671997096966116901741082',
 True)
```