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-23:
Кушнарьов Данііл та Присєвок Оксана

Київ - 2024

Мета: Ознайомлення з тестами перевірки чисел на простоту і методами генерації ключів для асиметричної криптосистеми типу RSA; практичне ознайомлення з системою захисту інформації на основі криптосистеми RSA, організація з використанням цієї системи засекреченого зв'язку й електронного підпису, вивчення протоколу розсилання ключів.

Хід 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 реалізувати власноруч, використання готових реалізацій тестів не дозволяється.

Виконання завдання:

```
#lab4, v_6
import random
from math import gcd

# Завдання 1: Перевірка числа на простоту
def is_probable_prime(n, iterations=5):
    """Перевіряємо, чи є число n імовірно простим за допомогою
тесту Міллера-Рабіна."""
    if n < 2:
        return False
    if n in (2, 3):
        return True
    if n % 2 == 0:
        return False

    # Розкладання n-1 у вигляді 2^s * d
    s, d = 0, n - 1
    while d % 2 == 0:
        s += 1
        d //= 2

    # Локальна функція перевірки свідків складеності
    def is_composite(base):
        result = pow(base, d, n)
```

```
        if result in (1, n - 1):
            return False
        for _ in range(s - 1):
            result = pow(result, 2, n)
            if result == n - 1:
                return False
        return True

    for _ in range(iterations):
        candidate = random.randint(2, n - 2)
        if is_composite(candidate):
            return False
    return True

def find_prime(bit_size):
    """Генеруємо випадкове просте число заданого розміру в
бітах."""
    while True:
        num = random.getrandbits(bit_size) | (1 << (bit_size -
1)) | 1
        if is_probable_prime(num):
            return num

# Генерація випадкового простого числа для завдання 1
bit_length = 256
random_prime_number = find_prime(bit_length)
print(f"№1: Згенеровано випадкове просте число ({bit_length}
біт): {random_prime_number}")
```

Результат:

```
PS D:\kpi\crypto 1> & C:/Users/Home/AppData/Local/Programs/Python/Python312/python.exe "d:/kpi/crypto 1/lab4_1.py"
№1: Згенеровано випадкове просте число (256 біт): 81391934231376981079556524360698638819751300463066206681592178426634505468591
```

2. За допомогою цієї функції згенерувати дві пари простих чисел p, q і pl, ql довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб pq ≤ plql; p, q – прості числа для побудови ключів абонента A, pl,ql – абонента B.

Виконання завдання:

```
# Завдання 2: Генерація пар простих чисел
def generate_key_pairs(size=256):
```

```
    """Створюємо дві пари простих чисел (p, q) і (p1, q1),
дотримуючись умови p*q <= p1*q1."""
    while True:
        p, q = find_prime(size), find_prime(size)
        p1, q1 = find_prime(size), find_prime(size)
        if p * q <= p1 * q1:
            return (p, q), (p1, q1)

key_pair_a, key_pair_b = generate_key_pairs()
print("№2: Пара (p, q) A:", key_pair_a)
print("№2: Пара (p1, q1) B:", key_pair_b)
```

Результат:

```
№2: Пара (p, q) A: (73351678178776623360959843497181365522470095200042761386897153916192450781117, 739767481905884627949806401959083740796181940926876889525058816850877438356 79)
№2: Пара (p1, q1) B: (979888331151036745079857073361704652005202167826666719276441258299317125869 07, 106777460838702105330703674420392133018451621113417180426839077120174574865329)
```

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.

Виконання завдання:

```
# Завдання 3: Генерація ключів RSA
def rsa_key_generation(prime1, prime2):
    """Генеруємо публічний та приватний ключі RSA на основі двох
простих чисел."""
    modulus = prime1 * prime2
    phi = (prime1 - 1) * (prime2 - 1)

    # Пошук відкритої експоненти e
    public_exponent = 65537  # Стандартне значення
    if gcd(public_exponent, phi) != 1:
        for candidate in range(3, phi, 2):
            if gcd(candidate, phi) == 1:
                public_exponent = candidate
                break

    # Обчислення секретної експоненти d
    private_exponent = pow(public_exponent, -1, phi)
```

```
    # Повернення пари ключів
    return (public_exponent, modulus), (private_exponent, prime1,
prime2)

# Генерація RSA-ключів для абонентів A та B
public_key_a, private_key_a = rsa_key_generation(key_pair_a[0],
key_pair_a[1])
public_key_b, private_key_b = rsa_key_generation(key_pair_b[0],
key_pair_b[1])

print("№3: Відкритий ключ A:", public_key_a)
print("№3: Секретний ключ A:", private_key_a)
print("№3: Відкритий ключ B:", public_key_b)
print("№3: Секретний ключ B:", private_key_b)
```

Результат:

```
№3: Відкритий ключ A: (65537, 542631862598844080218980114643711446799050149994431442406343268003361747163349259613389435631760853711678368820041990470247502692171019588082362524407344 3)
№3: Секретний ключ A: (161439126858908126342519100284161401798798843776666980390143131289126347893662603258637445702898556985397569140928418823324662657716827424144304496304465, 733516781787766233609598434971813655224700952000427613868971539161924507811 17, 739767481905884627949806401959083740796181940926876889525058816850877438356 79)
№3: Відкритий ключ B: (65537, 104629987905780986331658898120734850698285880493039946813934921281045478037425309595614105200059265794882987703645579653749682335754699343499656237336474 03)
№3: Секретний ключ B: (610167311254870304806395078700026741968321416385170777924192251161981342617495406839630938494296127488261496007784472884009583121393639360037514779858908 9, 979888331151036745079857073361704652005202167826666719276441258299317125869 07, 106777460838702105330703674420392133018451621113417180426839077120174574865329)
```

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.

```
# Функція для перетворення рядка в число
def string_to_number(message):
    return int.from_bytes(message.encode('utf-8'), 'big')

# Функція для перетворення числа в рядок
def number_to_string(number):
    byte_length = (number.bit_length() + 7) // 8
```

```
    return number.to_bytes(byte_length, 'big').decode('utf-8',
errors='ignore')

# Функція шифрування
def encrypt(message, public_key):
    e, n = public_key
    m = string_to_number(message)
    cipher = pow(m, e, n)
    return cipher

# Функція дешифрування
def decrypt(cipher, private_key):
    d, p, q = private_key
    n = p * q
    decrypted = pow(cipher, d, n)
    return number_to_string(decrypted)

# Функція хешування повідомлення (SHA-256)
def hash_message(message):
    return int(hashlib.sha256(message.encode('utf-8')).hexdigest(), 16)

# Функція підпису повідомлення
def sign_message(message, private_key):
    d, p, q = private_key
    n = p * q
    message_hash = hash_message(message)
    signature = pow(message_hash, d, n)
    return signature

# Функція перевірки підпису
def verify_signature(message, signature, public_key):
    e, n = public_key
    message_hash = hash_message(message)
    verified_hash = pow(signature, e, n)
    return message_hash == verified_hash
```

```
# Приклад шифрування та дешифрування
    print('\n№4')
    message = "Hello, RSA!"
    print(f"Повідомлення для шифрування: {message}")
    encrypted_message = rsa.encrypt(message, public_key_a)
    print(f"Зашифроване повідомлення: {encrypted_message}")
    decrypted_message = rsa.decrypt(encrypted_message, private_key_a)
    print(f"Дешифроване повідомлення: {decrypted_message}")

     # Приклад підпису та перевірки підпису
    signature = rsa.sign_message(message, private_key_a)
    print(f"Підпис повідомлення: {signature}")
```

```
    is_valid = rsa.verify_signature(decrypted_message, signature,
public_key_a)
    print(f"Перевірка підпису: {'Дійсний' if is_valid else 'Недійсний'}")

     # Приклад перевірки підпису зміненого повідомлення
    is_valid = rsa.verify_signature(decrypted_message + 'a', signature,
public_key_a)
    print(f"Перевірка підпису пошкодженого повідомлення: {'Дійсний' if
is_valid else 'Недійсний'}")
```

```
№4
Повідомлення для шифрування: Hello, RSA!
Зашифроване повідомлення: 3140442310866980554634895658224169236341263976597800357060167
Дешифроване повідомлення: Hello, RSA!
Підпис повідомлення: 18236847750520949899907330987429959223599485382345521189528085022
Перевірка підпису: Дійсний
Перевірка підпису пошкодженого повідомлення: Недійсний
```

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 $0 < k < n$.

```
import rsa

class Person:
    def __init__(self,name, bit_size=256):
        # Generate key pairs
        prime1, prime2 = rsa.generate_key_pairs(size=bit_size)[0]
        self.name = name
        self.public_key, self.private_key = rsa.rsa_key_generation(prime1,
prime2)
        self.friends_public_key = None

    def send_message(self, message):
        encrypted_message = rsa.encrypt(message, self.friends_public_key)  #
Encrypt message
        signature = rsa.sign_message(message, self.private_key)  # Create
signature
        return encrypted_message, signature

    def receive_message(self, encrypted_message, signature):
        # Decrypt the message
        decrypted_message = rsa.decrypt(encrypted_message, self.private_key)
```

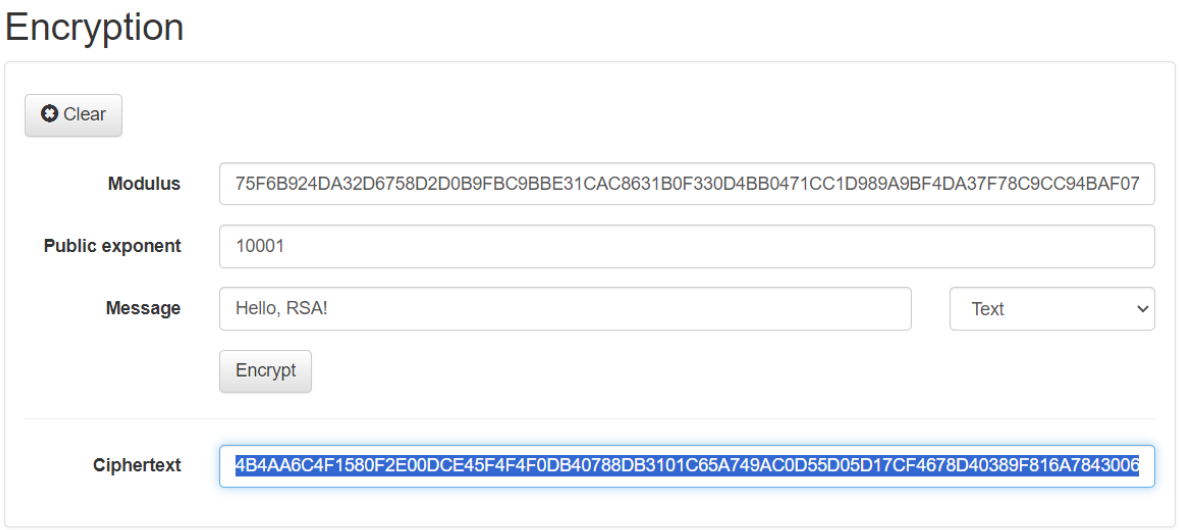
```
        # Verify the signature
        is_valid = rsa.verify_signature(decrypted_message, signature,
self.friends_public_key)
        if not is_valid:
            return None

        # Return the decrypted message as a string
        return decrypted_message

    def take_friends_key(self, public_key):
        self.friends_public_key = public_key
```

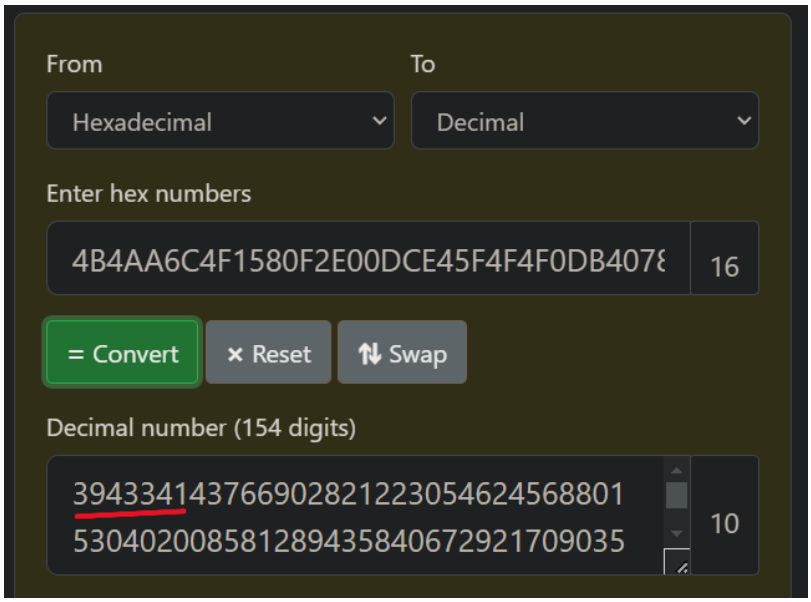
```
    print('\n№5')
    # Create two people (sender and receiver)
    alice = Person('Alice')
    bob = Person('Bob')

    #Exchange public keys
    alice.take_friends_key(bob.public_key)
    bob.take_friends_key(alice.public_key)

    # Alice sends a message to Bob
    message = "Hello, Bob!"
    print(f"Аліса пише '{message}' Бобу")
    encrypted_message, encrypted_message_signature =
alice.send_message(message)
    print(f"Зашифроване повідомлення: {encrypted_message}")
    print(f"Підпис повідомлення Аліси: {encrypted_message_signature}")

    decrypted_message= bob.receive_message(encrypted_message,
encrypted_message_signature)

    if decrypted_message:
        print(f"Боб отримав: {decrypted_message}")
    else:
        print('Повідомлення пошкоджене')

    # Bob sends a message to Alice
    message = "Hello, Alice!"
    print(f"\nБоб пише '{message}' Алісі")
    encrypted_message, encrypted_message_signature = bob.send_message(message)
    print(f"Зашифроване повідомлення: {encrypted_message}")
    print(f"Підпис повідомлення Боб: {encrypted_message_signature}")

    decrypted_message= alice.receive_message(encrypted_message,
encrypted_message_signature)
```

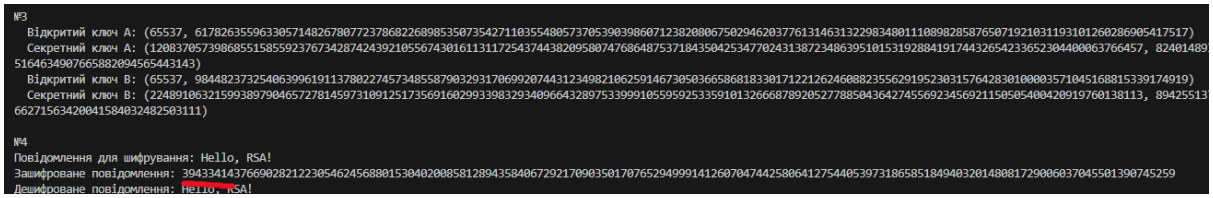
```python
    if decrypted_message:
        print(f"Аліса отримала: {decrypted_message}")
    else:
        print('Повідомлення пошкоджене')
```

```
№5
Аліса пише 'Hello, Bob!' Бобу
Зашифроване повідомлення: 181076421831635533840696732891168178400113851132498505156432978800845
Підпис повідомлення Аліси: 230397131193390023684921177452139350258437325468775679209881696165959
Боб отримав: Hello, Bob!

Боб пише 'Hello, Alice!' Алісі
Зашифроване повідомлення: 110162265204475842223262071840177790879213108743138501065742956099285
Підпис повідомлення Боб: 319099715275646013481889719807633560702384609714502483877407850803443
Аліса отримала: Hello, Alice!
```

Підставив відкритий ключ А конвертований у HEX

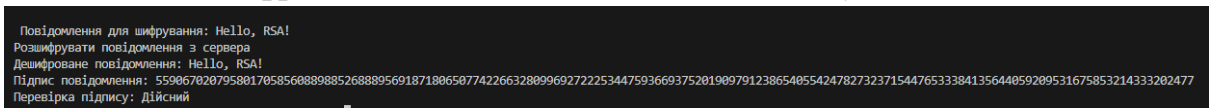







```python
if __name__ == "__main__":
    message = "Hello, RSA!"
    print(f"\n Повідомлення для шифрування: {message}")

    # Приклад шифрування та дешифрування
    print('Розшифрувати повідомлення з сервера')

    public_key_a = (
        65537,
        617826355963305714826780772378682268985350735427110355480573705390398607123820806750294620377613146313229834801110898285876507192103119310126028690541751 7
        )

    private_key_a = (
        120837057398685515855923767342874243921055674301611311725437443820958074768648753718435042534770243138723486395101531928841917443265423365230440006376645 7,
        824014893380795459324578106728276457197870704250334849919703416760963047846 19,
        749775715131151794598692765942144863370051029657651646349076658820945654431 43
        )

    encrypted_message = 39433414376690282122305462456880153040200858128943584067292170903501707652949991412607047442580641275440539731865851849403201480817290060370455013907452 59

    decrypted_message = decrypt(encrypted_message, private_key_a)
    print(f"Дешифроване повідомлення: {decrypted_message}")

     # Приклад підпису та перевірки підпису
    signature = sign_message(message, private_key_a)
    print(f"Підпис повідомлення: {signature}")
    is_valid = verify_signature(decrypted_message, signature, public_key_a)
    print(f"Перевірка підпису: {'Дійсний' if is_valid else 'Недійсний'}")
```

Підставив зашифроване повідомлення і ключі з сайту



Підпис дійсний