Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Фізико-технічний інститут

КОМП’ЮТЕРНИЙ ПРАКТИКУМ №4

з дисципліни «Криптографія»

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

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.

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 реалізовувати власноруч, використання готових реалізацій тестів не дозволяється

```python
def is_prime(num, k=10):
    if num <= 1:
        return False
    if num <= 3:
        return True
    if num % 2 == 0:
        return False

    r, d = 0, num - 1
    while d % 2 == 0:
        r += 1
        d //= 2

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

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

    return True
```

```python
def generate_random_prime(bit_len):
    candidates = []
    while True:
        candidate = random.getrandbits(bit_len) | (1 << (bit_len - 1)) | 1
        if is_prime(candidate):
            return candidate, candidates
        candidates.append(candidate)
```

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, p1і q1 – абонента B.

```python
def generate_prime_pairs(bit_len=256):
    while True:
        prime1, prime1_candidates = generate_random_prime(bit_len)
        prime2, prime2_candidates = generate_random_prime(bit_len)
        if prime1 * prime2 < generate_random_prime(bit_len * 2)[0]:
            return prime1, prime2, prime1_candidates, prime2_candidates
```

Перша пара:

```
Prime p: 93381638940247244751821631111215604513157271171049908331858171895624722333341
Prime q: 68553478681897190592511530027202352568039153360124580853619305304920081793297
```

Друга пара:

```
Prime p: 626241749568677716808365904048600505999580170089127212455064668934744738231 59
Prime q: 784977646083309593289153003165521015250443951774816340786449055144121505917 97
```

Приклад підбору кандидатів(які не підійшли):

```
Candidates for p in A that failed primality test: [810259999664965963957462571629356611334719531445211502490969649116295815937 21, 682514809795167298648001104171327979929312896
Candidates for q in A that failed primality test: [699494524171631877256149345823941103264733115121566835883936623238661591263 07, 626272621021226667226721550703287347001963864
Candidates for p in B that failed primality test: [719995048128846384954980112099950760977236099269169478095118596387559713086 69, 971338712604287499643612530384874458646489916
Candidates for q in B that failed primality test: [113358048268938313955766674580360076254258168938081738480200182555711096662245, 860404529838320705662756887847348021580770293
```

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.

```python
def generate_rsa_keys(bit_len=256):
    prime1, prime2, prime1_candidates, prime2_candidates = generate_prime_pairs(bit_len)
    modulus = prime1 * prime2
    phi = (prime1 - 1) * (prime2 - 1)

    print(f"Prime p: {prime1}")
    print(f"Prime q: {prime2}")
    print(f"Modulus n: {modulus}")
    print(f"Euler's totient phi(n): {phi}")

    e = 65537
    if gcd(e, phi) != 1:
        raise ValueError("e and phi(n) are not coprime!")

    d = pow(e, -1, phi)

    return (e, modulus), (d, prime1, prime2), prime1_candidates, prime2_candidates
```

```
Public key A: (65537, 640163619437086005393349151474992596522417048000791411019316403777102471046714017832122656284715196028984464684697097197090342564842881777325207349341527
Private key A: (830570709076329753247729959410998679864826305621363408135442174850344433115655522864339966274861532469832141603227684536533610809268750820716940788333633, 9338
Public key B: (65537, 491585774455514094930139849197619803990469662380082284846782540676828340941879911889977734071842361041713803285142662151027612644056341992542454166974026723
Private key B: (229459701029473910584981127100788675463821313790211593080976009000180902051552815858791894650883237418523730986547327331575300066533159557552048098287949 7, 6262
```

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.

```
def encrypt(message, public_key):
    e, n = public_key
    return pow(message, e, n)


2 usages
def decrypt(ciphertext, private_key):
    d, p, q = private_key
    n = p * q
    return pow(ciphertext, d, n)


1 usage
def sign(message, private_key):
    d, p, q = private_key
    n = p * q
    return pow(message, d, n)


1 usage
def verify(signature, message, public_key):
    e, n = public_key
    return pow(signature, e, n) == message
```

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 $0 < k < n$.

```
if __name__ == "__main__":
    public_key_A, private_key_A, candidates_p_A, candidates_q_A =
generate_rsa_keys()
    public_key_B, private_key_B, candidates_p_B, candidates_q_B =
generate_rsa_keys()
```

```
    print("Public key A:", public_key_A)
    print("Private key A:", private_key_A)
    print("Public key B:", public_key_B)
    print("Private key B:", private_key_B)

    print("Candidates for p in A that failed primality test:",
candidates_p_A)
    print("Candidates for q in A that failed primality test:",
candidates_q_A)
    print("Candidates for p in B that failed primality test:",
candidates_p_B)
    print("Candidates for q in B that failed primality test:",
candidates_q_B)

    random_message = random.randint(1, public_key_A[1] - 1)
    print("Random message M:", random_message)

    digital_signature = sign(random_message, private_key_A)
    print("Digital signature by A:", digital_signature)

    encrypted_message = encrypt(random_message, public_key_B)
    encrypted_signature = encrypt(digital_signature, public_key_B)
    print("Encrypted message for B:", encrypted_message)
    print("Encrypted signature for B:", encrypted_signature)

    decrypted_message = decrypt(encrypted_message, private_key_B)
    decrypted_signature = decrypt(encrypted_signature, private_key_B)
    print("Decrypted message by B:", decrypted_message)
    print("Decrypted signature by B:", decrypted_signature)

    is_valid_signature = verify(decrypted_signature, decrypted_message,
public_key_A)
    print("Signature validity verified by B:", is_valid_signature)
```

```
Random message M: 14587651256241234126325575774086186668415764730482283806276242633115408599644555137430876169431586121762434014508910142606814423455487110782157944266542
Digital signature by A: 20651725774585752136490987413971670645310114068679629605868971774347367488775294246651610469934595652320641373288261629263345111606570612397229438684352
Encrypted message for B: 34963302833087747432818632398613197991748986103048140111860555306379845516757524505420786597645405508351298714155294238864920892092579075429625097043121
Encrypted signature for B: 34743413143814476482336054854477478427877688122843260597960665700998178591507328670134410984820975986208980500654307243361473369275266269941599073812121
Decrypted message by B: 14587651256241234126325575774086186668415764730482283806276242633115408599644555137430876169431586121762434014508910142606814423455487110782157944266542
Decrypted signature by B: 20651725774585752136490987413971670645310114068679629605868971774347367488775294246651610469934595652320641373288261629263345111606570612397229438684352
Signature validity verified by B: True
```

Висновок:

Під час виконання роботи ми дослідили принципи функціонування криптосистеми RSA та алгоритму цифрового підпису. Написали програми: 1.Для перевірки чисел на простоту за допомогою тесту Міллера–Рабіна та 2.Для створення ключів до системи типу RSA. Також перевірили роботу протоколу конфіденційного розсилання ключів з підтвердженням валідності.