НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»
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-22
Мартинюк Артем
Шеїна Еліна

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.
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 $pq \leq p1q1$; p і q – прості числа для побудови ключів абонента A, p1 і q1 – абонента B.
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 .
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 і B, перевірити правильність розшифрування. Скласти для A і B повідомлення з цифровим підписом і перевірити його.

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 $0 < k < n$.

Хід роботи

Для початку реалізуємо тест Міллера-Рабіна

```python
def miller_rabin(n, k=40):
    if n <= 1 or (n > 3 and n % 2 == 0):
      return False
    if n <= 3:
      return True

    # Представляємо n-1 у вигляді (2^r)*d
    r, d = 0, n - 1
    while d % 2 == 0:
        r += 1
        d //= 2

    # Тестування
    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
```

Після цього генеруємо рандомне число необхідної нам довжини та тестуємо його.

```python
# Генеруємо випадкове просте число
def generating_prime(bit_length):
    while True:
        number = random.getrandbits(bit_length) | (1 << (bit_length - 1)) | 1
        if miller_rabin(number):
            return number
```

Приклад результату роботи функцій:

```
Просте число: 100574064494694283857249577410038087635246173863423551143772504817695638085723
```

Тепер генеруємо дві пари простих чисел p, q і p1, q1.

```python
# Генеруємо дві пари простих чисел p, q і p1, q1
def generating_pairs(bit_length):
    while True:
        p = generating_prime(bit_length)
        q = generating_prime(bit_length)
        p1 = generating_prime(bit_length)
        q1 = generating_prime(bit_length)
        if p*q <= p1*q1:
            return (p, q), (p1, q1)

bit_length = 256
(pair1, pair2) = generating_pairs(bit_length)
(p, q) = pair1
(p1, q1) = pair2
```

Результат:

```
Пара для A: p = 75696055949374935531539360269919943641097864649664761825099781654484847217323 , q = 81059111580080410830968372851627995147020976835150522596362127618916968815207
Пара для B: p1 = 81734248121107113957517993519244515354238157209037091213785191711352798812171 , q1 = 83622390664251948001183617437572689071602655261494339700910807651229859167739
```

Наступним етапом генеруємо відкритий і секретний ключі для абонентів A і B. Обираємо за значення $e = 2^{16} + 1 = 65537$.

```python
def inverse(e, phi):
    # Обчислюємо e (mod phi) за розширеним алгоритмом Евкліда
    a, b = e, phi
    x0, x1 = 1, 0
    while b != 0:
        q, a, b = a // b, b, a % b
        x0, x1 = x1, x0 - q*x1
    return x0 % phi

# Генеруємо пару ключів RSA: (n, e) та (d, p, q)
def generating_rsa_keys(p, q):
    n = p*q
    phi = (p - 1)*(q - 1)
    e = 65537
    d = inverse(e, phi)
    return (n, e), (d, p, q)

# Генеруємо ключі для A і B
pub_a, priv_a = generating_rsa_keys(p, q)
pub_b, priv_b = generating_rsa_keys(p1, q1)
```

Результат:

```
Відкритий ключ A:
(61358550453723925132662732953852356741814903426463853020887319739628441838764108805798203405142884274651133983540266892818781206086784709174170821562308 61, 65537)
```

```
Секретний ключ A:
(560472445238831355439175098882984800035400976352569097951141875412838097904341176684439906540874189021218487470067048813408768832768240211275330393343657
7,
756960559493749355315393602699199436410978646496647618250997816544848472173
23,
810591115800804108309683728516279951470209768351505225963621276189169688152
07)

Відкритий ключ B:
(683481322703211984798749553034608898702211842501293911573344297959982262880410055237725720751984060124045205725216977125778397469778317417309493384375
1369, 65537)

Секретний ключ B:
(114791322748893820539234879995299451424618852715439249350562288289752731548964576803169750882670714358639934856959454113571822131068589253792929169021
7533,
817342481211071139575179935192445153542381572090370912137851917113527988121
71,
836223906642519480011836174375726890716026552614943397009108076512298591677
39)
```

Шифруємо повідомлення за допомогою формули

$$C = M^e \bmod n$$

```python
# Шифруємо повідомлення
def encrypting_message(message, pub_key):
    n, e = pub_key
    return pow(message, e, n)
```

Розшифровуємо повідомлення через формулу

$$M = C^d \bmod n$$

```python
# Розшифровуємо повідомлення
def decrypting_message(cipher, priv_key):
    d, p, q = priv_key
    n = p*q
    return pow(cipher, d, n)
```

Підписуємо повідомлення, використовуючи формулу

$$S = M^d \bmod n$$

```python
# Підписуємо повідомлення
def signing(message, priv_key):
    d, p, q = priv_key
    n = p*q
    return pow(message, d, n)
```

Перевіряємо підпис за допомогою формули

$$M = S^e \bmod n$$

```python
# Перевіряємо підпис
def verifying(signature, pub_key, message):
    n, e = pub_key
    return pow(signature, e, n) == message
```

Результат шифрування та підпису повідомлення:

```
Повідомлення: 7836246787960157983440498352639089471307492828368290437920622607633266288259025183558037988088992489610128675179790488167957081343575604379323515570038 25
Шифротекст: 3327221206454051302028899961144073105520745337996975207329412543186601821945636019426313885537888097546447276340454160514554298715296416472664733196118604 3
Розшифроване повідомлення: 7836246787960157983440498352639089471307492828368290437920622607633266288259025183558037988088992489610128675179790488167957081343575604379323515570038 25
Цифровий підпис: 1971970979307441879881762820833906737940990456313254774580049699852399710185842517443505673574484289589831630945182163497065677836341919153727664062132 50
Перевірка підпису: True
```

Відправляємо ключ. Формуємо підпис за допомогою формули

$$S = k^d \bmod n$$

Шифруємо ключ через формулу

$$k_1 = k^{e_1} \bmod n_1$$

Шифруємо підпис за допомогою формули

$$S_1 = S^{e_1} \bmod n_1$$

Функція відправки ключа:

```python
def send_key(secret_key, recipient_pub_key, k):
    signature = signing(k, secret_key)  # Підписуємо ключ
    encrypted_key = encrypting_message(k, recipient_pub_key)  # Шифруємо ключ
    encrypted_signature = encrypting_message(signature, recipient_pub_key)  # Шифруємо підпис
    return encrypted_key, encrypted_signature
```

Отримуємо ключ. Розшифровуємо його через формулу

$$k = k_1^{d_1} \bmod n_1$$

Далі розшифровуємо підпис, використовуючи формулу

$$S = S_1^{d_1} \bmod n_1$$

Перевіряємо підпис із формулою

$$k = S^e \bmod n$$

Функція отримання ключа:

```python
def receive_key(encrypted_key, encrypted_signature, recipient_priv_key, sender_pub_key):
    decrypted_key = decrypting_message(encrypted_key, recipient_priv_key)  # Розшифровуємо ключ
    decrypted_signature = decrypting_message(encrypted_signature, recipient_priv_key)  # Розшифровуємо підпис
    # Перевіряємо справжність підпису
    if verifying(decrypted_signature, sender_pub_key, decrypted_key):
        return decrypted_key
    else:
        print("Невірний підпис! Ключ не є автентичним.")
```

Результат відправки та отримання ключа:

```
Переданий ключ: 59305227958101909336544094501462503092123828646511231488928723665862795670057332294598055953645521792788341678551872339760498777360176938659411454185717698
Отриманий ключ: 59305227958101909336544094501462503092123828646511231488928723665862795670057332294598055953645521792788341678551872339760498777360176938659411454185717698
Перевірка пройшла успішно!
```

Висновок:

У ході виконання лабораторної роботи було розглянуто принципи роботи алгоритму RSA. Було реалізовано функцію пошуку простих чисел за допомогою тесту Міллера-Рабіна, згенеровані пари ключів для двох абонентів, реалізовано функції шифрування та розшифрування, був створений цифровий підпис, реалізовано конфіденційну передачу ключа, а також перевірена робота всіх цих функцій. У підсумку ми можемо стверджувати, що RSA є ефективним криптографічним алгоритмом, що забезпечує захист даних у відкритих каналах зв’язку.