КРИПТОГРАФІЯ КОМП'ЮТЕРНИЙ ПРАКТИКУМ №4

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.

Виконали Шанідзе Давид та Тивонюк Володимир

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.

Використали тест міллера-рабіна для перевірки числа на простоту

```
def generate_prime(size):
    while True:
        number = gen_int_size(size)
        # print(f"{num} is {millet_rabin(num)}")
        is_prime = miller_rabin(number)
        if is_prime:
            return number
```

Довільні числа розміру/ в межах чисел

```
def gen_int_size(size): #generate int number sized ""size"" bits
    return getrandbits(size)


def gen_int_limits(start, finish): #generate int number between the limits start/finish
    return randint(start, finish)
```

gcd з 3 лабки

```python
def linear_gcd(a: int, b: int): #returns gcd of nums
    rest = a % b
    while rest != 0:
        a = b
        b = rest
        rest = a - (b * int(a / b))
    return b
```

```python
def miller_rabin(number):
    #first check basic division
    k = 8 # 2^k = 256
    if number == 2 or number == 3 or number == 5 or number == 7 or number == 11 or number == 13:
        return True
    elif number <= 1 or number % 2 == 0 or number % 3 == 0 or number % 5 == 0 or number % 7 == 0 or number % 11 == 0 or number % 13 == 0:
        return False
    else: #miler-rabin
        #step 0: number - 1 = d * 2 ^ s
        s = 0
        d = number - 1
        while d % 2 == 0:
            s += 1
            d //= 2
        #step 1
        for iterations in range(k):
            random_x = gen_int_limits(2, number - 1) #important rand
            if linear_gcd(random_x, d) != 1:
                return False
            else:
                #step 2
                xd = pow(random_x, d, number) #xd = x ^ d mod number
                # step 2.1
                if xd == 1 or xd == number - 1:
                    continue
                # step 2.2
                xr = random_x #iteration parameter
                for r in range(1, s):
                    xr = pow(xr, 2, number) #xr = x(r - 1) ^ 2 mod number
                    if xr == number - 1:
                        break
                    elif xr == 1:
                        return False
                return False
        return True
```

повертає True коли просте, False коли складене число

2. За допомогою цієї функції згенерувати дві пари простих чисел p, q і p_1, q_1 довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб $pq \le p_1 q_1$; p і q – прості числа для побудови ключів абонента A, p_1 і q_1 – абонента B.

```python
def generate_primes_for_keys():
    p = generate_prime(256)
    q = generate_prime(256)
    p1 = generate_prime(256)
    q1 = generate_prime(256)
    if p * q > p1 * q1:
        p, q, p1, q1 = p1, q1, p, q
    return [p, q], [p1, q1]
```

Генерується 4 простих числа розміру 256 біт

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), (e_1, n_1) та секретні d і d_1.

```
def private_public_keys(A, B):
    p, q, p1, q1 = A[0], A[1], B[0], B[1]
    e = 65537 #ferma prime 2^2^n + 1
    n, n1 = p * q, p1 * q1
    f, f1 = (p - 1) * (q - 1), (p1 - 1) * (q1 - 1)
    d, d1 = pow(e, -1, f), pow(e, -1, f1)
    private_A, public_A, private_B, public_B = (d, p, q), (n, e), (d1, p1, q1), (n1, e)
    return private_A, public_A, private_B, public_B
```

Обраховуються відкриті/закриті ключі

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.

Шифрування:

```
def Encrypt(M, public_B):
    n, e = public_B
    return pow(M, e, n) #C


def Decrypt(C, private_B):
    d, p, q = private_B
    return pow(C, d, p * q) #M
```

Цифровий підпис:

Абонент A формує повідомлення (k_1, S_1) і відправляє його B, де

$$k_1 = k^{e_1} \bmod n_1, \quad S_1 = S^{e_1} \bmod n_1, \quad S = k^d \bmod n.$$

Абонент B за допомогою свого секретного ключа d_1 знаходить (конфіденційність):

$$k = k_1^{d_1} \bmod n_1, \quad S = S_1^{d_1} \bmod n_1,$$

і за допомогою відкритого ключа e абонента A перевіряє підпис A (автентифікація):

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

```python
def Generate_Sign(k, private_A, public_B):#генерація
    d, p, q = private_A
    n1, e1 = public_B
    n = p * q
    k1 = pow(k, e1, n1)
    S = pow(k, d, n)
    S1 = pow(S, e1, n1)
    return k1, S1


def Sign(k1, S1, private_B):#конфіденційність
    d1, p1, q1 = private_B
    n1 = p1 * q1
    k = pow(k1, d1, n1)
    S = pow(S1, d1, n1)
    return k, S


def Verify(k, S, public_A):#автентифікація
    n, e = public_A
    k_ = pow(S, e, n)
    if k_ == k:
        return True
    else:
        return False
```

```
===RSA Message Decryption===
Message M: 14538117088140153831928031308602063470550822041353148021301228598617592776602471734129754103530873080873019524545741859220878846762360441150471571164094
Encrypted C: 24876987956591660474092248992084969879413270002267345900575903704846276405828591727631562273541227535530459166662429596686437962402919054997922117575596
Decrypted C: 14538117088140153831928031308602063470550822041353148021301228598617592776602471734129754103530873080873019524545741859220878846762360441150471571164094
Correct decryption!
```

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 $0 < k < n$.

Використовуючи вище описані функції:

```
===Keys Generation===
private_A: (184650114183497510768376591863597441235025060281448068757432355170623859996306428189948238518224229462362266999589585179074179373970646564997276906338033, 527442509
public_A: (190118370722740469478211159130342887988120363471144065891972544760285891900621482464685245467270172805424884961481524698448390710558886407799017579106082 9, 65537)
private_B: (849744578037569879992887096566521268242391999663982047268884895989916953441613771736891042703606372371772709021665290697045571326300076789570462697076049, 1010205251
public_A: (513553212936630553494083516016239796827898490776317911458955129886682716497898047962223880751938348560946500560451369785307380610412320652917195245865917 , 65537)
===RSA Message Decryption===
Message M: 16346424390456336208949977935007871784250986734764483141810427655455310440580223284257242868242386887894396214298799831001097959707975213334827169936548 9
Encrypted C: 31922259949414867936910199838986850976813962842831384316073844235983605909451844248429611229024471420682133769711033691169739306627036621921185639764042 5
Decrypted C: 16346424390456336208949977935007871784250986734764483141810427655455310440580223284257242868242386887894396214298799831001097959707975213334827169936548 9
Correct decryption!
===RSA Signature Verification Protocol===
Message k: 13145878173475027780464177760463463132810293940876405347114983722496943711664665965290050564540036685357897024308125605110820872701753428681292843021171 7
Signed A k1, S1: 16808111988016269840315096080804066359370720349783497862844225638655785546875569027529490126798489888263460001981335055915020256214573128111988911182490595 51264
Designed B k, S: 13145878173475027780464177760463463132810293940876405347114983722496943711664665965290050564540036685357897024308125605110820872701753428681292843021171 87 90382
Verified B: True
```

Правильне розшифрування месседжів та цифровий підпис між абонентами

Висновок:

В результаті роботи було освоєно знання про криптосистему RSA та алгоритм електронного підпису. Також було ознайомлено з різними методами генерації параметрів для асиметричних криптосистем. Це дасть можливість ефективно використовувати ці знання у практичному застосуванні, наприклад, для створення безпечного засекреченого зв'язку та підписання електронних документів.