Міністерство освіти і науки України Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського"

Фізико-технічний інститут

КОМП'ЮТЕРНИЙ ПРАКТИКУМ №4

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

Варіант №6

Виконали:
ФБ-21 Захожий М.
ФБ-21 Хав'юк А.

Київ 2024

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.
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, 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 и 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.

Кожна з наведених операцій повинна бути реалізована у вигляді окремої процедури, інтерфейс якої повинен приймати лише ті дані, які необхідні для її роботи; наприклад, функція Encrypt(), яка шифрує повідомлення для абонента, повинна приймати на вхід повідомлення та відкритий ключ адресата (і тільки його), повертаючи в якості результату шифротекст. Відповідно, програмний код повинен містити сім високорівневих процедур: GenerateKeyPair(), Encrypt(), Decrypt(), Sign(), Verify(), SendKey(), ReceiveKey().

Кожну операцію рекомендується перевіряти шляхом взаємодії із тестовим середовищем, розташованим за адресою http://asymcryptwebservice.appspot.com/?section=rsa. Наприклад, для перевірки коректності операції шифрування необхідно а) зашифрувати власною реалізацією повідомлення для серверу та розшифрувати його на сервері, б) зашифрувати на сервері повідомлення для вашої реалізації та розшифрувати його локально.

Порядок виконання роботи:

1. Код:

```
def is_prime_trial_division(n):
    if n < 2:
        return False
    for p in [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31]:
        if n % p == 0 and n != p:
            return False
    return True

def miller_rabin_test(n, k=5):
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0:
        return False

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

    def check_composite(a):
        x = pow(a, d, n)
        if x == 1 or x == n - 1:
            return False
        for _ in range(s - 1):
            x = pow(x, 2, n)
```

```
            if x == n - 1:
                return False
        return True

    for _ in range(k):
        a = random.randint(2, n - 2)
        if check_composite(a):
            return False
    return True

def generate_random_prime(bit_length):
    while True:
        candidate = random.getrandbits(bit_length) | (1 << (bit_length - 1)) | 1
        if is_prime_trial_division(candidate) and miller_rabin_test(candidate):
            return candidate
```

Пояснення:

Функція generate_random_prime реалізує пошук випадкового простого числа заданої довжини в бітах. Вона використовує вбудований генератор псевдовипадкових чисел та два методи перевірки на простоту: пробне ділення (is_prime_trial_division) і тест Міллера-Рабіна (miller_rabin_test).

Використовується вбудований генератор псевдовипадкових чисел (random.getrandbits) для створення випадкового числа заданої довжини в бітах. Кандидат генерується так, щоб:

- мати задану довжину в бітах (завдяки | (1 << (bit_length - 1))).
- бути непарним (завдяки | 1).

Кандидат перевіряється на простоту:

- спершу методом пробних ділень (is_prime_trial_division).
- потім тестом Міллера-Рабіна (miller_rabin_test).
- якщо обидві перевірки пройдені, число повертається як результат.

Результат:

```
Випадкове просте число (256 біт): 100415482435426262717278513742904038217685693247178427925596747857006
726880139
```

2. Код:

```
def generate_prime_pairs(bit_length):
    # Генерація першої пари для абонента A
    p = generate_random_prime(bit_length)
    q = generate_random_prime(bit_length)

    # Генерація другої пари для абонента B
    p1 = generate_random_prime(bit_length)
```

```
    q1 = generate_random_prime(bit_length)

    # Перевірка умови pq <= p1q1
    if p * q <= p1 * q1:
        return (p, q), (p1, q1)
    return (p1, q1), (p, q)
```

Пояснення:

Використання функції generate_random_prime гарантує, що всі згенеровані числа мають довжину щонайменше bit_length (наприклад, 256 біт). Логіка функції забезпечує, що добуток чисел абонента A завжди буде меншим або рівним добутку чисел абонента B, як цього вимагає завдання.

Результат:

```
--------------------------------------------------------------------------------------------------------
Пара для абонента A:
       p = 790946522158869339643390401888132129460260804365544874792128748292517580142 69,
       q = 824838894103685681715267599248918109804873597574538882940905414156744332810 97

Пара для абонента B:
       p1 = 10589751383873480520809912159564580834963092491050359016947691178585206068044 1,
       q1 = 65452731117854063036415156025178312413168302825876700842194232148473293761741

--------------------------------------------------------------------------------------------------------
```

3. Код:

```
def generate_rsa_keypair(p, q):
    # Обчислення n та функції Ейлера φ(n)
    n = p * q
    phi_n = (p - 1) * (q - 1)

    # Генерація відкритої експоненти e (взаємно простої з φ(n))
    e = 65537  # Стандартний вибір для RSA
    if gcd(e, phi_n) != 1:
        # Якщо e не взаємно просте з φ(n), вибираємо інше
        while True:
            e = random.randint(2, phi_n - 1)
            if gcd(e, phi_n) == 1:
                break

    # Обчислення секретного ключа d (обернений за модулем φ(n))
    d = pow(e, -1, phi_n)

    # Повернення секретного (d, p, q) та відкритого (e, n) ключів
    return (d, p, q), (e, n)

def rsa_setup(p, q, p1, q1):
    # Генерація ключових пар для абонента A
    private_key_a, public_key_a = generate_rsa_keypair(p, q)

    # Генерація ключових пар для абонента B
    private_key_b, public_key_b = generate_rsa_keypair(p1, q1)

    return private_key_a, public_key_a, private_key_b, public_key_b
```

Пояснення:

Функція генерації ключових пар для RSA generate_rsa_keypair повертає секретний ключ (d, p, q); відкритий ключ (e, n).
Функція rsa_setup дозволяє побудувати ключі для двох абонентів, забезпечуючи їх подальше використання в протоколі.

Результат:

```
Відкритий ключ A:
        e = 65537,
        n = 652403454632678107752326871213423330732468699844853420348640252419325316509959820105618615439083985467630832438748916601571346538295373224165390701397309 3

Секретний ключ A:
        d = 2949488374095796215053129204446727782210402537147437629355915290436271083948475024619434750509992988486088322277843017427713596772292734267177971112426849,
        p = 790946522158869339643390401888132129460260804365544874792128748292517580144269,
        q = 824838894103685681715267599248918109804873597574538882940905414156744332810 97

Відкритий ключ B:
        e = 65537,
        n = 693128149933593885697300888954014531345586915375872590796277604416711280926090425866993743915673869069206427220114540039837279468488393703963051699280778 1

Секретний ключ B:
        d = 2486449304200496246813791278103801155367921690111961885594471287949842411854674926790535114367189497345065618877424889047738866631062407055675686488822 73,
        p = 10589751383873480520809912159564580834963092491050359016947691178585206068044 1,
        q = 654527311178540630364151560251783124131683028258767008421942321484732937617 41
```

4. Код:

```python
# Функція шифрування повідомлення
def encrypt(message, public_key):
    e, n = public_key
    return pow(message, e, n)

# Функція розшифрування повідомлення
def decrypt(ciphertext, private_key):
    d, p, q = private_key
    n = p * q
    return pow(ciphertext, d, n)

# Функція створення цифрового підпису
def sign(message, private_key):
    d, p, q = private_key
    n = p * q
    return pow(message, d, n)

# Функція перевірки цифрового підпису
def verify(message, signature, public_key):
    e, n = public_key
    return pow(signature, e, n) == message
```

Пояснення:

Шифрування (encrypt), розшифрування (decrypt), створення підпису (sign) та перевірка підпису (verify) реалізовані як окремі функції. Використовується випадкове повідомлення M, яке шифрується, розшифровується, підписується і перевіряється.

Результат:

```
Повідомлення для шифрування та підпису: 37462138794623786428372

Зашифроване повідомлення: 641587706712425020222540614918809601613197377597140716305413620425475377169674496829823602648218038216800590878713700817045157868981473338031815931433052 6

Розшифроване повідомлення: 37462138794623786428372

Цифровий підпис: 308414490528877524537235693561167491336305640639090404361055016090043382240814039916725865931120559496937247207558961041798568386878895586924900928638 4393

Цифровий підпис вірний: True
```

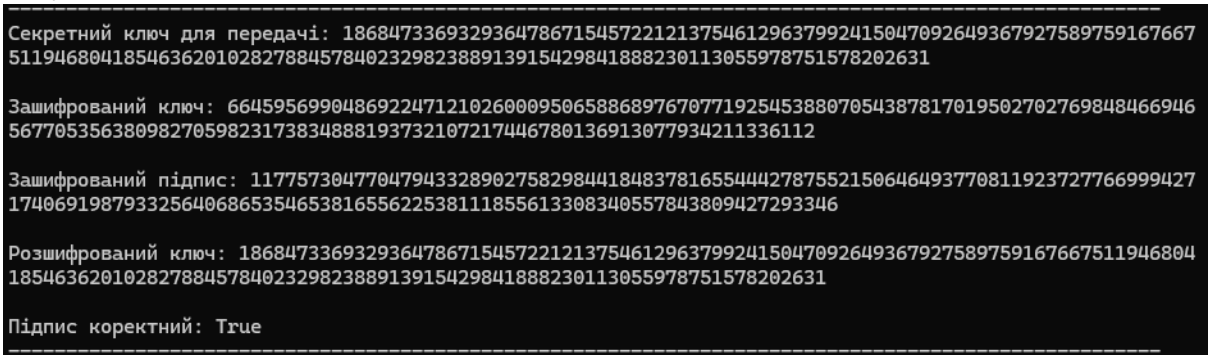
5. Код:

```
# Функція для генерації пар ключів (GenerateKeyPair)
def generate_keypair(bit_length):
    p, q = generate_random_prime(bit_length), generate_random_prime(bit_length)
    private_key, public_key = generate_rsa_keypair(p, q)
    return private_key, public_key

# Функція шифрування повідомлення (Encrypt)
def encrypt(message, public_key):
    e, n = public_key
    return pow(message, e, n)

# Функція розшифрування повідомлення (Decrypt)
def decrypt(ciphertext, private_key):
    d, p, q = private_key
    n = p * q
    return pow(ciphertext, d, n)

# Функція створення цифрового підпису (Sign)
def sign(message, private_key):
    d, p, q = private_key
    n = p * q
    return pow(message, d, n)

# Функція перевірки цифрового підпису (Verify)
def verify(message, signature, public_key):
    e, n = public_key
    return pow(signature, e, n) == message

# Функція відправки ключа (SendKey)
def send_key(k, sender_private_key, recipient_public_key):
    encrypted_key = encrypt(k, recipient_public_key)  # Шифруємо ключ для
отримувача
    signature = sign(k, sender_private_key)  # Підписуємо ключ
    encrypted_signature = encrypt(signature, recipient_public_key)  # Шифруємо
підпис
    return encrypted_key, encrypted_signature

# Функція отримання ключа (ReceiveKey)
def receive_key(encrypted_key, encrypted_signature, recipient_private_key,
sender_public_key):
    decrypted_key = decrypt(encrypted_key, recipient_private_key)  # Розшифровуємо
ключ
    decrypted_signature = decrypt(encrypted_signature, recipient_private_key)  #
Розшифровуємо підпис
    is_valid_signature = verify(decrypted_key, decrypted_signature,
sender_public_key)  # Перевіряємо підпис
    return decrypted_key, is_valid_signature
```

Пояснення:

Реалізовано всі 7 процедур: GenerateKeyPair, Encrypt, Decrypt, Sign, Verify, SendKey, ReceiveKey. Передача ключа k захищена за допомогою RSA та підтверджена підписом.

Результат:



Висновки

У ході роботи ми детально вивчили основні принципи роботи алгоритму RSA. Під час виконання завдань ми навчилися генерувати випадкові прості числа заданої довжини з використанням методу пробних ділень та імовірнісного тесту Міллера-Рабіна. Ці знання дозволили нам отримувати надійні прості числа для створення ключів.

Ми реалізували функції генерації відкритих та закритих ключів RSA на основі згенерованих простих чисел, зокрема обчислення модуля n, функції Ейлера $\phi(n)$, відкритої експоненти e, яка є взаємно простою з $\phi(n)$, та секретного ключа d як оберненого до e за модулем $\phi(n)$.

Далі ми розробили процедури для виконання основних операцій шифрування та розшифрування повідомлень за допомогою RSA. Окрім цього, було реалізовано функції створення та перевірки цифрового підпису, які дозволяють гарантувати автентичність переданих даних.

Протокол було побудовано на основі таких процедур:

- Шифрування секретного ключа для передачі одержувачу.
- Створення цифрового підпису для підтвердження справжності даних.
- Розшифрування секретного ключа на стороні отримувача.
- Перевірка автентичності переданого ключа за допомогою підпису.

Київ 2024

Таким чином, ми не лише здобули теоретичні знання про алгоритм RSA, але й закріпили їх на практиці, реалізувавши повноцінний механізм для забезпечення конфіденційності та автентичності даних за допомогою криптографічних методів.

Київ 2024