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:
ФБ-31 Острун Катерина
ФБ-31 Острун Михайло

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 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.

Хід роботи:

1) Написали функцію пошуку випадкового простого числа заданої довжини. В якості тесту
перевірки на простоту використовували тест Міллера-Рабіна із попередніми пробними діленнями.

```python
def check_small_divisors(n):
    primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53]
    for p in primes:
        if n % p == 0:
            return False
    return True

def probabilistic_primality_test(p, k=40):
    if p % 2 == 0: return False

    s = 0
    d = p - 1
    while d % 2 == 0:
        d //= 2
        s += 1

    for _ in range(k):
        x = random.randint(2, p - 1)
        if find_gcd(x, p) > 1: return False

        x_r = horner_pow(x, d, p)

        if x_r == 1 or x_r == p - 1:
            continue

        is_strong = False
        for _ in range(1, s):
            x_r = horner_pow(x_r, 2, p)

            if x_r == p - 1:
                is_strong = True
                break
            if x_r == 1:
                return False

        if not is_strong:
            return False

    return True

def generate_candidate_prime(bits):
    while True:
        p = random.getrandbits(bits)
        if p % 2 == 0: p += 1
        if check_small_divisors(p) and probabilistic_primality_test(p):
            return p
```

Для непарного числа p (кандидата) ми знаходимо його розклад:

$$p-1=d\cdot 2^s$$

де d - непарне число, а s - найбільша кількість послідовних ділень на 2.

2) Викликається generate_candidate_prime(256) для p, q. Цикл while True, генерує p1, q1 для абонента B до тих пір, поки не виконається умова n_B ≥ n_A.

```
Генерація p для A:
Знайдено просте p_A = 21488094790643165662888876040276345952169456144486790415969188537717672426923 (254 біт)
Відкинуті кандидати (приклади):
   32243583089735341868038470340870340324521870297019923761621729663867480161089 — відкинуто тестом Міллера-Рабіна
   46915618820746625732604808659864514259016898494297111860726677432710503555141 — відкинуто тестом Міллера-Рабіна
   19205009347768632470203876483036025765342380605631181824941051706545431845305 — відкинуто пробними діленнями
   70649857166156589058426624621293552491252084837260765077802485498484494659939 — відкинуто пробними діленнями
   37025068008151399060486660399929021427415488199156257461145004621392240584947 — відкинуто тестом Міллера-Рабіна

Генерація q для A:
Знайдено просте q_A = 91986736909287612650443242602784717890569631171522715056034105479932526842567 (256 біт)
Відкинуті кандидати (приклади):
   45128023520294887086970379186927021444565293527744039272835187245612469684341 — відкинуто тестом Міллера-Рабіна
   68329537187495844921419666755743113687951895782058497960083071767671639788787 — відкинуто пробними діленнями
   65167224787148673087231962577562672249919304307747325846664045243695460412079 — відкинуто тестом Міллера-Рабіна
   78318868197629562889850940321699163754788695955793710524691827426099505667821 — відкинуто пробними діленнями
   18487356206286248780425667104485599668473684723371236473760746671123901267 33 — відкинуто пробними діленнями
```

```
Спроба 2 генерації для B:
   Генерація p1 для B:
   Знайдено просте p1_B = 44521666228706021166747536396304098843783988386929309562345499337383744519611 (255 біт)
   Відкинуті кандидати (приклади):
     53562979412088615824672534652397834529317614475699327057279390275650915750377 — відкинуто тестом Міллера-Рабіна
     27952957950099593000415309376352433497984301303604094601601654047585622141817 — відкинуто пробними діленнями
     86740829642014948245937606420538162177266479546102447829677350273820748762503 — відкинуто пробними діленнями
     97279035392139750726815426698219184861909429057328347824673278816256223227779 — відкинуто тестом Міллера-Рабіна
     68076081434116587433561399942316160971847024061793614460687407180717413223419 — відкинуто пробними діленнями
   Генерація q1 для B:
   Знайдено просте q1_B = 91335651057679067000727910931384238663791929895373084622613859667730631285379 (256 біт)
   Відкинуті кандидати (приклади):
     90207786554840753130730458881038589035461935780367299853550372691269290545323 — відкинуто тестом Міллера-Рабіна
     63192993994806781852934854538956937041358060427272594813521284965879503538 69 — відкинуто тестом Міллера-Рабіна
     24052592716732765193361255234739492933700099624946264057533427562914323445561 — відкинуто пробними діленнями
     68860903799690036386578256156748873380110843982047299900753808537256047823371 — відкинуто пробними діленнями
     44205292608451873695986434233422773735407097533082111340406951325972199881937 — відкинуто тестом Міллера-Рабіна
Успіх після 2 спроб. Умова n_B (511 біт) >= n_A (510 біт) виконана
```

3) Написана функція генерації ключових пар для RSA. Після генерування функція повертає та виводить секретний ключ (d, p,q) та відкритий ключ (n,e). За допомогою цієї функції побудувані схеми RSA для абонентів.

```
def GenerateKeyPair(p, q, e=65537, label=""):
    n = p * q
    phi_n = (p - 1) * (q - 1)
    d = modular_inverse(e, phi_n)
    public_key = (e, n)
    private_key = (d, p, q)
    print(f"--- ЗГЕНЕРОВАНІ КЛЮЧІ {label} ---")
    print(f"Секретний ключ (d, p, q):\n   d = {d}\n   p = {p} ({p.bit_length()} біт)\n   q = {q} ({q.bit_length()} біт)")
    print(f"\nВідкритий ключ (n, e):\n   n = {n} ({n.bit_length()} біт)\n   e = {e}")
    print("-----------------------------------")
    return public_key, private_key
```

```
=== Завдання 3: Генерація ключів абонента A ===
--- ЗГЕНЕРОВАНІ КЛЮЧІ A ---
Секретний ключ (d, p, q):
   d = 36178764638483782516422281688131669633227692522681118723273967348723306531970063137532973194014894181084135468875223512260266907239499680105814612244358 17
   p = 83900876452386528819059760795153643608049865910792748098582234421310514688579 (256 біт)
   q = 93480554496944348857669505711954739490997390756467983586154792868531606463597 (256 біт)

Відкритий ключ (n, e):
   n = 78431004535487137533616720419274427996190773869834027249088882211414751089470267342293961709297891964217814500313261099505943961216985987423994059551586631 (512 біт)
   e = 65537
-----------------------------------

=== Завдання 3: Генерація ключів абонента B ===
--- ЗГЕНЕРОВАНІ КЛЮЧІ B ---
Секретний ключ (d, p, q):
   d = 41789410832184815340980293790522639928258405189817576874434728240904657601919796121251688676296615144441548675267377729082566600796167777244830860810782 73
   p = 92167924585281688139695903489568969186597036469725179327913028015627037414291 (256 біт)
   q = 11054617826840753780989789494217776255293248270594951152571472454397724728453 9 (256 біт)

Відкритий ключ (n, e):
   n = 10188811821833691380215124680615633381615591893322449909299958276503603218962322053382883156407687937255521685463131397937643559052380590614518235701946849 (512 біт)
   e = 65537
```

4) Шифрування, розшифрування і створення повідомлення з

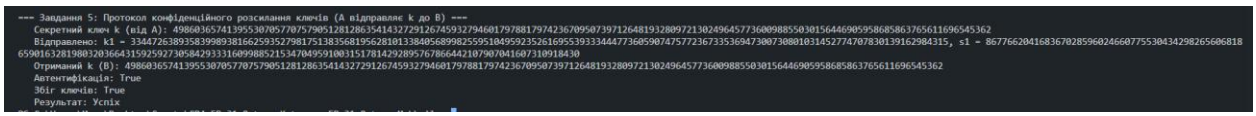
цифровим підписом для абонентів A і B. За допомогою датчика випадкових чисел вибрано відкрите повідомлення M і знайдено криптограму для абонентів A и B.

```python
def Encrypt(message, public_key):
    e, n = public_key
    if message >= n:
        raise ValueError("Повідомлення завелике")
    return horner_pow(message, e, n)

def Decrypt(ciphertext, private_key):
    d, p, q = private_key
    n = p * q
    return horner_pow(ciphertext, d, n)

def Sign(message, private_key):
    d, p, q = private_key
    n = p * q
    return horner_pow(message, d, n)

def Verify(message, signature, public_key):
    e, n = public_key
    return horner_pow(signature, e, n) == message
```

```
=== Завдання 4: Шифрування та розшифрування (для B як приклад) ===
   Відкрите повідомлення M = 655279304116794393472143101868597012818062078941177274142724458707495712141409299960954613344186287664893194195491740947647414026357536774105710121460 07
   Шифротекст C = 396364878704406044905108444071303391929739852439850301393380256320408069198850815169793081858243722540822023023779402891054685879107344986481363864741883 0
   Розшифровано: 655279304116794393472143101868597012818062078941177274142724458707495712141409299960954613344186287664893194195491740947647414026357536774105710121460 07 [OK]

=== Завдання 4: Цифровий підпис та перевірка (для A як приклад) ===
   Відкрите повідомлення M = 655279304116794393472143101868597012818062078941177274142724458707495712141409299960954613344186287664893194195491740947647414026357536774105710121460 07
   Підпис S = 917102416268505532166778405881781953228172298002516773332063575516637354838419082643155365228908648652687938856519755412075803725712825593877733042704499
   Підпис вірний: True [OK]
```

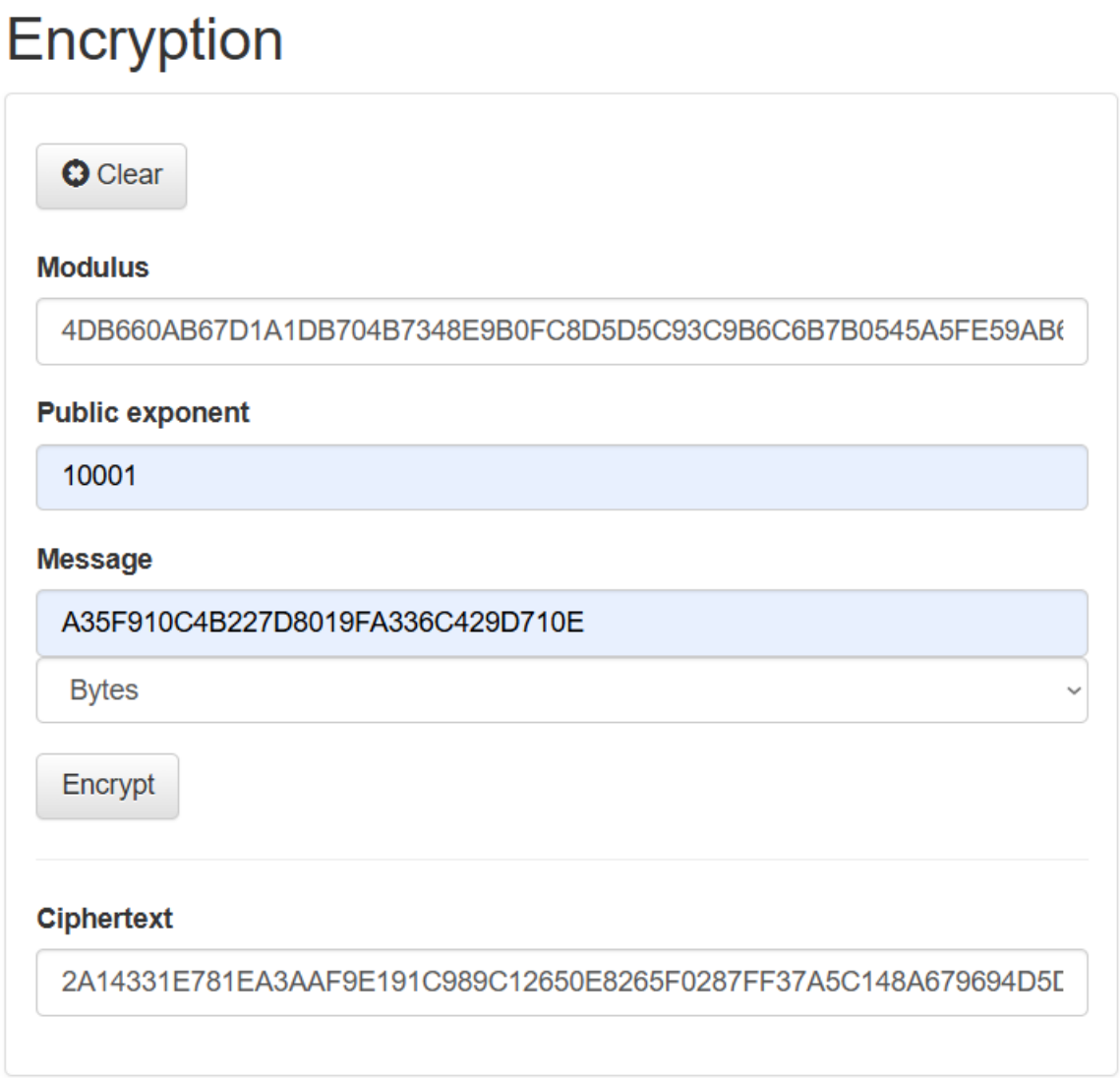
5) Організовуємо роботу протоколу конфіденційного розсилання ключів з підтвердженням справжності по відкритому каналу за допомогою алгоритму RSA

```python
def SendKey(k, sender_priv, recipient_pub):
    d, p, q = sender_priv
    n = p * q
    e1, n1 = recipient_pub
    if n1 < n:
        raise ValueError("n1 має бути >= n")
    s = Sign(k, sender_priv)
    k1 = horner_pow(k, e1, n1)
    s1 = horner_pow(s, e1, n1)
    return k1, s1

def ReceiveKey(payload, recipient_priv, sender_pub):
    k1, s1 = payload
    d1, p1, q1 = recipient_priv
    n1 = p1 * q1
    k = horner_pow(k1, d1, n1)
    s = horner_pow(s1, d1, n1)
    is_valid = Verify(k, s, sender_pub)
    return k, is_valid
```



Перевіряємо шляхом взаємодії із тестовим середовищем:

```
--- ЗГЕНЕРОВАНІ КЛЮЧІ A ---
Секретний ключ (d, p, q):
 d = 88560E02A880FE951CCD41D4092897EAEA80204722823E80FB7C12019F0C5A745B9D5532077FD3721C961A925130190DD048B827
95EECA871FFF3E594E6BB998462ADBB4CADC90B1231
 p = EF688E1DD9C5319C74F390273AE8A6F808386D4991AEEFB47A621669A4E055DC47451D9BDD (296 біт)
 q = 53191B242A59F9E1076214119DF760C59C8A034334F248387CA609ACF20090F39250CDF79B (295 біт)

Відкритий ключ (n, e):
 n = 4DB660AB67D1A1DB704B7348E9B0FC8D5D5C93C9B6C6B7B0545A5FE59AB6CFE8772B36F7BB8BCFA5F1E0E79D28D8B26765492EF0
9DB3CB52CA7E05CE6EEF7F437FB08856674B4A4899CF (591 біт)
 e = 65537
```




```
Виберіть операцію: 2
Введіть повідомлення M: A35F910C4B227D8019FA336C429D710E
Введіть e (відкритий експонент): 10001
Введіть n (модуль): 4DB660AB67D1A1DB704B7348E9B0FC8D5D5C93C9B6C6B7B0545A5FE59AB6CFE8772B36F7BB8BCFA5F1E0E79D2
8D8B26765492EF09DB3CB52CA7E05CE6EEF7F437FB08856674B4A4899CF
Шифротекст C = 2A14331E781EA3AAF9E191C989C12650E8265F0287FF37A5C148A679694D5DCF0D88A665A73F2376684A5C3677848D
393CD329A7CD5AAA4373EE6AB044D5FB62EE6EA9713F5200127 73A
```

```
Виберіть операцію: 4
Введіть повідомлення M: A35F910C4B227D8019FA336C429D710E
Введіть d (секретний експонент): 88560E02A880FE951CCD41D4092897EAEA80204722823E80FB7C12019F0C5A745B9D5532077F
D3721C961A925130190DD048B82795EECA871FFF3E594E6BB998462ADBB4CADC90B1231
Введіть n (модуль): 4DB660AB67D1A1DB704B7348E9B0FC8D5D5C93C9B6C6B7B0545A5FE59AB6CFE8772B36F7BB8BCFA5F1E0E79D2
8D8B26765492EF09DB3CB52CA7E05CE6EEF7F437FB08856674B4A4899CF
Підпис S = 2E5D5A5A813940EF94BE3588454D0BD56847E8CA6F9FC857AF8D6985CCB0C141C7B34BE64A90EFDFBD272D1D863ECD2374
D9195B097FFBC4C9C6FF22EF4D3FD82267A8C50DC544C052A3
```

Verify

Clear

Message

A35F910C4B227D8019FA336C429D710E

Bytes

Signature

2E5D5A5A813940EF94BE3588454D0BD56847E8CA6F9FC857AF8D6985CCB0(

Modulus

4DB660AB67D1A1DB704B7348E9B0FC8D5D5C93C9B6C6B7B0545A5FE59AB(

Public exponent

10001

Verify

Verification

true ✔

Висновки:

Під час виконання практичної роботи було глибоко теоретичні засади RSA, методи перевірки чисел на простоту та принципи формування параметрів асиметричних криптосистем.

У прикладній частині досліджено низку алгоритмів тестування простоти - зокрема, ймовірнісний тест Міллера-Рабіна та перевірку на основі ознаки подільності Паскаля. Це дало змогу порівняти їхню швидкодію та ресурсомісткість під час генерування великих простих чисел.

Також було засвоєно використання схеми Горнера для ефективного обчислення степенів за модулем - операції, що лежить в основі багатьох криптографічних процедур.

У підсумку було поглиблено розуміння принципів створення криптографічних систем, механізмів захищеного обміну даними та застосування електронного підпису. Додатково відпрацьовано процес безпечної передачі ключів в асиметричних схемах і підтверджено коректність кожного етапу в тестовому середовищі.