КРИПТОГРАФІЯ
КОМП’ЮТЕРНИЙ ПРАКТИКУМ №5
Вивчення криптосистеми RSA та алгоритму електронного підпису; ознайомлення з методами генерації параметрів для асиметричних криптосистем

ФБ-33
Стогнійчук Інна
Грабченко Олександр

Мета та основні завдання 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 і рекомендації щодо 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 для генерації простого числа з інтервалу, перевірку на подільність на прості числа, перервірки тест Міллера-Рабіна :

```python
from mod import gcd, Horner
import random

BASE_PRIMES = [
    2, 3, 5, 7, 11, 13, 17, 19, 23, 29,
    31, 37, 41, 43, 47, 53, 59, 61, 67,
    71, 73, 79, 83, 89, 97
]

def trial_division(n):
    if n < 2:
        return False

    for p in BASE_PRIMES:
        if n == p:
            return True
        if n % p == 0:
            return False
    return True

def miller_rabin(n, rounds):
    if n < 2:
        return False

    d = n - 1
    s = 0
    while d % 2 == 0:
        d >>= 1
        s += 1

```

```python
    for _ in range(rounds):
        a = random.randrange(2, n - 1)
        if gcd(a, n) > 1:
            return False

        x = Horner(a, d, n)
        if x == 1 or x == n - 1:
            continue

        composite = True
        for _ in range(s - 1):
            x = Horner(x, 2, n)
            if x == n - 1:
                composite = False
                break

        if composite:
            return False
    return True

def generate_prime(low, high, accuracy=10):
    """Генерує випадкове просте число і повертає список відхилених кандидатів."""
    rejected = []

    while True:
        candidate = random.randint(low, high)

        if not trial_division(candidate):
            rejected.append((candidate, "trial_division"))
            continue
        if not miller_rabin(candidate, accuracy):
            rejected.append((candidate, "miller_rabin"))
            continue

        return candidate, rejected
```

2) За допомогою цієї функції згенерували 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.

```python
def generate_prime_pairs(bits):
    left = 2 ** (bits - 1)
    right = 2 ** bits - 1

    p, rejected_p = generate_prime(left, right)
    q, rejected_q = generate_prime(left, right)

    while True:
        p1, rejected_p1 = generate_prime(left, right)
        q1, rejected_q1 = generate_prime(left, right)

        if p * q <= p1 * q1:
            # повертаємо пари та всі відхилені кандидати
            return (p, q, rejected_p, rejected_q), (p1, q1, rejected_p1, rejected_q1)

if __name__ == "__main__":

    bit_len = 256

    (p, q, rejected_p, rejected_q), (p1, q1, rejected_p1, rejected_q1) = generate_prime_pairs(bit_len)
```

Результат :

```
PS C:\Users\u1208> & C:/Users/u1208/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/u1208/OneDrive/Робочий стіл/crypto4/random_prime.py"
================================================================================================================================================================
Перша пара простих:
  p  = 99384097374660429424795750679055848764211243444875979665575906253208654020577
  q  = 93188667195897935338845148805047349900840483655644738575137806070475226722729
Відхилені кандидати для p:
   * 69400214248236088378447163620628119389121586552113857868393213695602748028201 — miller_rabin
   * 10245266973563724895775969164431124835453171848937235597270211840443066060699 4 — trial_division
   * 96590251168301223843764639104316049672022150560256714705114765872559879332315 — trial_division
   * 11287846224128450484465794211185359629600681645833651638606593667107511526248 0 — trial_division
   * 11576198625544272076732159054444957203507393771079925377145778318718137081104 4 — trial_division
   * 58667679004895883934454095745074211407400680732167622590595418606236009681855 — trial_division
   * 61091261356326786379332649862201505492068234235568446142176395086326481968400 — trial_division
   * 97527174746097774261987456805375156966358952520227447412054364769448700017054 — trial_division
   * 81573583247528186405984594725029000983493724765996156440680316831255357121490 — trial_division
   * 66359181713597477626262769893995744534600614713518225005963344646186429544670 — trial_division
Відхилені кандидати для q:
   * 67466010799062515720448313182966005327162721185915241901247862327029968255043 — trial_division
   * 84624491739924364242035655494389511753170310529325652492164008129304667828941 — miller_rabin
   * 74475386922760308971411524192265993097200919897522144288199683350086665244310 — trial_division
   * 10946335938154162070784792330441462204859501263190102909666648905750654926705 9 — trial_division
   * 95118692894487819224481566495494180574288233674931832353569114481044384069814 — trial_division
   * 96409027693612945093517786446772164484083812195791404509012036568961474662748 — trial_division
--------------------------------------------------------------------------------------------------------------------------------------------------------------
Друга пара простих:
  p1 = 11445577666442378313193446272521959564075842204791808062034446350808525503650 9
  q1 = 81245424302726018775951214032471026023291066354551435754540825096594550241341
Відхилені кандидати для p1:
   * 71728430635432043056887232187500083246433357118772045610061742432203184752658 — trial_division
   * 85176839476118325182292688657490358231284111734884170094628829569782743379614 — trial_division
   * 58891430386044503012459419682822730761317319946766819693520316948336547621975 — trial_division
```

```
Відхилені кандидати для q1:
  * 756571479904898406214445707708525763419185882639870230632789092440020200146206 – trial_division
  * 779788485593553938638817975124782607294387062058227922670104435195853162038 78 – trial_division
  * 680395132363543714876653451566713111872188081064247631270479888221650147616 69 – miller_rabin
  * 102399156475414344228065037823472851236981716050868786663574311954456081091774 – trial_division
  * 878732945288590331053683984879439054623855678163998589825632970724830703976 1 – trial_division
```

```
n1 = p*q   = 92614715748119444769550114673484135916382594445397522911569450271706339562347606721991284649174536423735676645352476648515978285054369017932100370395946 33
n2 = p1*q1 = 92990081389991575717684963521377517130008895908711215389460507127433787068384863555425683617374375455258815796691518535547776385473757193166239617161185 69
✓ Перевірка підтверджена: n1 <= n2
```

3. Написана функція генерації ключових пар для RSA. Після генерування функція повертатє та/або зберігати секретний ключ (d, p,q)та відкритий ключ (n,e). За допомогою цієї функції побудована схеми RSA для абонентів A і B– тобто, створені та збережені для подальшого використання відкриті ключі (e,n), (e1,n1) та секретні d і d1.

```python
from random_prime import generate_prime_pairs
from mod import gcd, mod_inverse

def generate_rsa_keys(p, q):
    """
    Генерує пару ключів RSA для заданих простих чисел p і q.
    Повертає:
      - Відкритий ключ (n, e)
      - Секретний ключ (d, p, q)
    """
    n = p * q
    phi = (p - 1) * (q - 1)

    e = 65537
    if gcd(e, phi) != 1:
        raise ValueError("Не вдалося обрати e взаємно просте з phi(n)")

    d = mod_inverse(e, phi)
    return (n, e), (d, p, q)
```

```python
if __name__ == "__main__":
    bit_length = 256

    (p, q, _, _), (p1, q1, _, _) = generate_prime_pairs(bit_length)

    # Генеруємо ключі для абонента A
    public_a, private_a = generate_rsa_keys(p, q)
    n_a, e_a = public_a
    d_a, p_a, q_a = private_a

    # Генеруємо ключі для абонента B
    public_b, private_b = generate_rsa_keys(p1, q1)
    n_b, e_b = public_b
    d_b, p_b, q_b = private_b

    print("=" * 160)
    print("Ключі абонента A:")
    print("-" * 160)
    print(f"Відкритий ключ:\n  e = {e_a}\n  n = {n_a}")
    print("-" * 160)
    print(f"Секретний ключ:\n  d = {d_a}\n  p = {p_a}\n  q = {q_a}")
    print("=" * 160)

    print("Ключі абонента B:")
    print("-" * 160)
    print(f"Відкритий ключ:\n  e = {e_b}\n  n = {n_b}")
    print("-" * 160)
    print(f"Секретний ключ:\n  d = {d_b}\n  p = {p_b}\n  q = {q_b}")
    print("=" * 160)
```

Результат :

```
PS C:\Users\u1208> & C:/Users/u1208/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/u1208/OneDrive/Робочий стіл/crypto4/RSA_key.py"
================================================================================================================================================================
Ключі абонента A:
----------------------------------------------------------------------------------------------------------------------------------------------------------------
Відкритий ключ:
  e = 65537
  n = 49336079988730302072666471272366531219611766236582055643917921004102807928294811178214701594995344075044830038953269083979140840863134339965589982132726 01
----------------------------------------------------------------------------------------------------------------------------------------------------------------
Секретний ключ:
  d = 49246497103968123365269927181526065265797374363561422038143683053029548027115929712676736344190088667749165954440366307421455923348864737160672066204934 73
  p = 83477710182100791460750977730256352072376952225762752964108473165814930758901
  q = 59100902361968351924215977856283028658721898652403316114698541065742512583701
================================================================================================================================================================
Ключі абонента B:
----------------------------------------------------------------------------------------------------------------------------------------------------------------
Відкритий ключ:
  e = 65537
  n = 66360836837752356823759038587864346738343772227830653081199991605199989738882418500958972321369190725823246873652775907601973639330196882896322319317883 23
----------------------------------------------------------------------------------------------------------------------------------------------------------------
Секретний ключ:
  d = 39851739558164388170234897564773747571310799145074248337836459856353754309064026923909433005734229734185034113943356242400636230492586796202835570265322 41
  p = 63148495007120377977724235782898736780925946958782224976464327642851461791817
  q = 10508696498668696334154630552092645529815359958868727397694925870685527601419
================================================================================================================================================================
```

4. Створюємо функції для шифрування, розшифрування і підписання ЦП та його перевірки для абонентів A і B:

```python
def RSA_Encrypt(plaintext, public_key):
    n, e = public_key
    return Horner(plaintext, e, n)

def RSA_Decrypt(ciphertext, private_key):
    d, p, q = private_key
    n = p * q
    return Horner(ciphertext, d, n)

def RSA_Sign(plaintext, private_key):
    d, p, q = private_key
    n = p * q
    return Horner(plaintext, d, n)

def RSA_Verify(signature, public_key, plaintext):
    n, e = public_key
    verified_value = Horner(signature, e, n)
    print(f"Підпис відновлюється в {verified_value}, очікуване: {plaintext}")
    return verified_value == plaintext
```

Результат :

```
Шифрування
================================================================================================================================================================
Оригінальне повідомлення (plaintext): 4496571097796133980318483438601769810125281410532192189955271650793827103187686450926643771733800068731156846048208686078420974614800817607341055413994456
----------------------------------------------------------------------------------------------------------------------------------------------------------------
Шифротекст (ciphertext): 7446956208001418761231363092580059618023923100007726892821361103993678744885952914239791832487904984247064835107674050602339930283551850121368572083154 67

Розшифрування
================================================================================================================================================================
Розшифроване повідомлення (plaintext): 4496571097796133980318483438601769810125281410532192189955271650793827103187686450926643771733800068731156846048208686078420974614800817607341055413994456

Цифровий підпис
================================================================================================================================================================
Розшифрування
================================================================================================================================================================
Розшифроване повідомлення (plaintext): 4496571097796133980318483438601769810125281410532192189955271650793827103187686450926643771733800068731156846048208686078420974614800817607341055413994456

Цифровий підпис
================================================================================================================================================================
Цифровий підпис (signature): 6184948757208089510967606433506725725567985672144003262863842359085008113874896150068380577248933601244336799304312081901644562160239073648742906145076109

Перевірка підпису
================================================================================================================================================================
Підпис відновлюється в 4496571097796133980318483438601769810125281410532192189955271650793827103187686450926643771733800068731156846048208686078420974614800817607341055413994456, очікуване: 4496571097796133980318483438601769810125281410532192189955271650793827103187686450926643771733800068731156846048208686078420974614800817607341055413994456
Підпис валідний: True
```

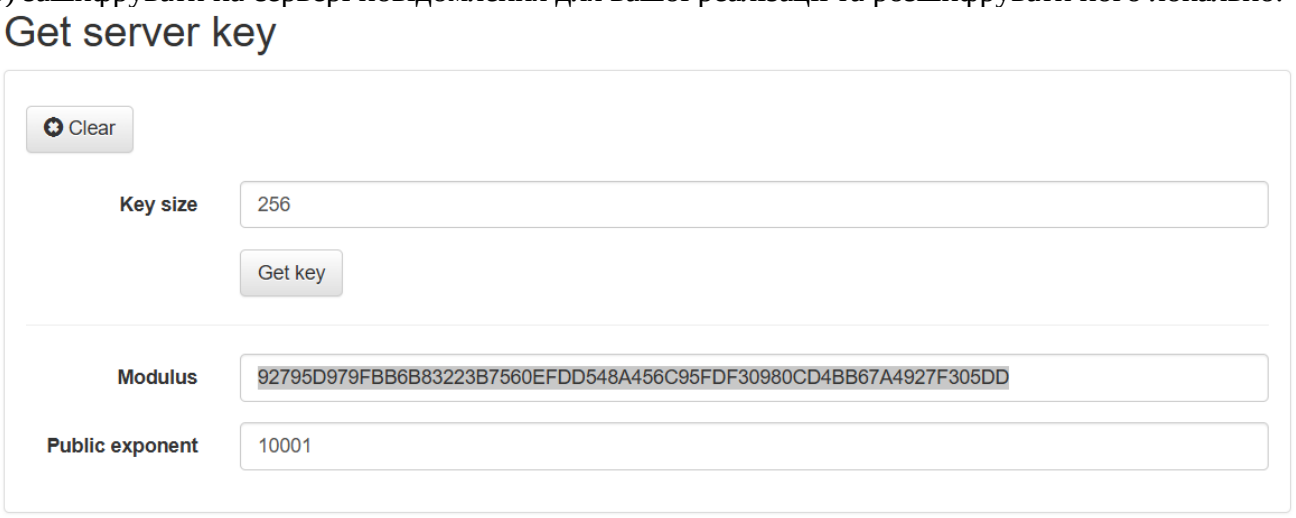
5.Пишемо функції для організації роботи протоколу конфіденційного розсилання ключів:

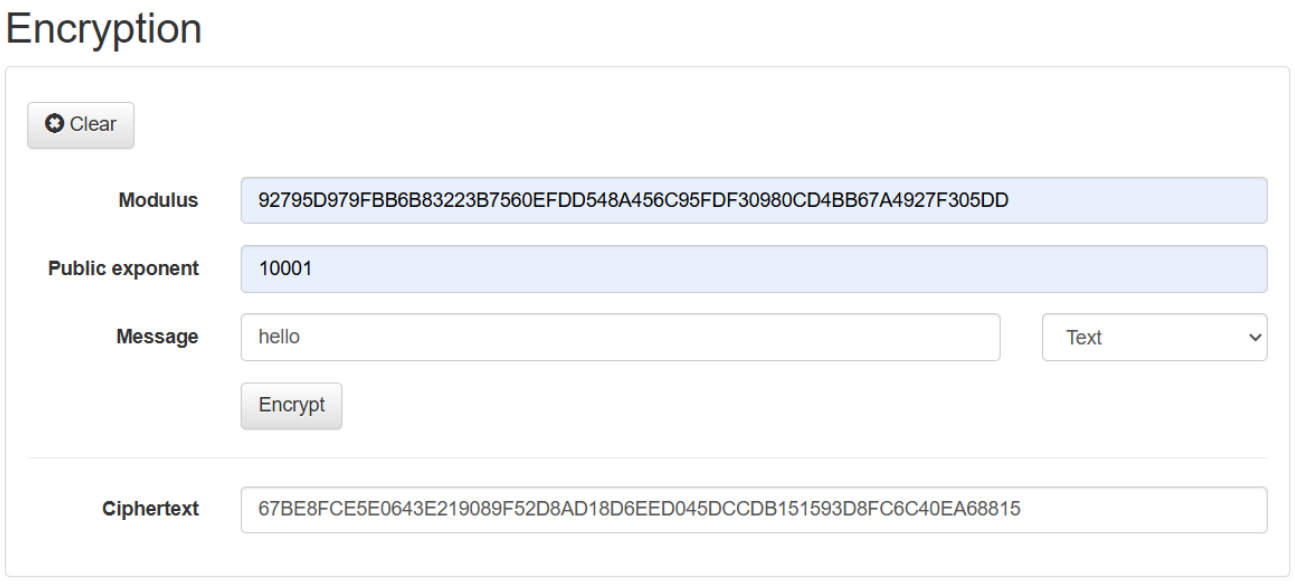
```python
def RSA_SendKey(k, public_key_b, private_key_a):
    """
    Реалізація роботи відправника A.
    public_key_b: відкритий ключ абонента B (n1, e1)
    private_key_a: секретний ключ абонента A (d, p, q)
    """
    k1 = RSA_Encrypt(k, public_key_b)   # k1 = k^e1 mod n1
    S = RSA_Sign(k, private_key_a)      # S = k^d mod n
    S1 = RSA_Encrypt(S, public_key_b)   # S1 = S^e1 mod n1

    return k1, S1

def RSA_ReceiveKey(k1, S1, private_key_b, public_key_a):
    """
    Реалізація роботи приймача B.
    k1: зашифроване значення k
    S1: зашифрований підпис S
    private_key_b: секретний ключ абонента B (d1, p1, q1)
    public_key_a: відкритий ключ абонента A (n, e)
    """
    k_dec = RSA_Decrypt(k1, private_key_b)  # k = k1^d1 mod n1
    S_dec = RSA_Decrypt(S1, private_key_b)  # S = S1^d1 mod n1

    k_verify = RSA_Verify(S_dec, public_key_a, k_dec)  # Перевірка підпису

    return k_dec, k_verify
```

Результат :

```
Підпис відновлюється в 43702, очікуване: 43702
Підпис відновлюється в 43702, очікуване: 43702
Оригінальний k: 43702
Розшифрований абонентом B k: 43702
Перевірка підпису A пройшла успішно: True
Перевірка відповідності ключів: Успішно
```

Перейдемо до демонстрації протоколу :
92795D979FBB6B83223B7560EFDD548A456C95FDF30980CD4BB67A4927F305DD — ключ серверу.
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.






```
=== Шифрування ===
Оригінальне повідомлення (plaintext): hello
Шифротекст власний: 469249168597628682664947548787892553436463513135694356660143310794663452037 33
Шифротекст сервер: 46924916859762868266494754878789255343646351313569435666014331079466345203733
```

Decryption

Clear

Ciphertext 67BE8FCE5E0643E219089F52D8AD18D6EED045DCCDB151593D8FC6C40EA6 Text

Decrypt

Message hello

Зашифруємо на основі свогго :

Encryption

Clear

Modulus 8fb13d3419b7f9373b5766e9ccd6a460966952906d9c76064887592d30bdac63

Public exponent 10001

Message hello Text

Encrypt

Ciphertext 3B4589EE11B64D448A3FA7AC0BAA879692873C841F947198268A6626C0535078

```
=== Розшифрування ===
Оригінальне повідомлення (plaintext): hello
Шифротекст: 26809322471464546102044282757390116209023462083585674052464340294001264644216
Розшифрований: hello
```

Sign

Clear

Message hello Text

Sign

Signature 171724600AA21AAF590E9E66645B393FCA87DE673B5F9B9C6EA3955855946B6C

Verify

Clear

Message hello Text

Signature 171724600AA21AAF590E9E66645B393FCA87DE673B5F9B9C6EA3955855946B6C

Modulus 92795D979FBB6B83223B7560EFDD548A456C95FDF30980CD4BB67A4927F305DD

Public exponent 10001

Verify

Verification true

```
=== Перевірка підпису ===
Підпис відновлюється в 448378203247, очікуване: 448378203247
Підпис коректний: True
```

```
=== Цифровий підпис ===
Цифровий підпис (signature): 3101913473957510849582103267562088573094775729884036366194435228161636996037 0
Цифровий підпис у hex форматі: 0x44943545a67c96baefac571d6670ea58483e0c5ecebe1a700ff93de19b81c5b2
public_key у hex форматі: (n: 0x8fb13d3419b7f9373b5766e9ccd6a460966952906d9c76064887592d30bdac63, e: 0x10001)
```

Verify

Clear

Message hello Text

Signature 44943545a67c96baefac571d6670ea58483e0c5ecebe1a700ff93de19b81c5b2

Modulus 8fb13d3419b7f9373b5766e9ccd6a460966952906d9c76064887592d30bdac63

Public exponent 10001

Verify

Verification true ✔

Send key

Clear

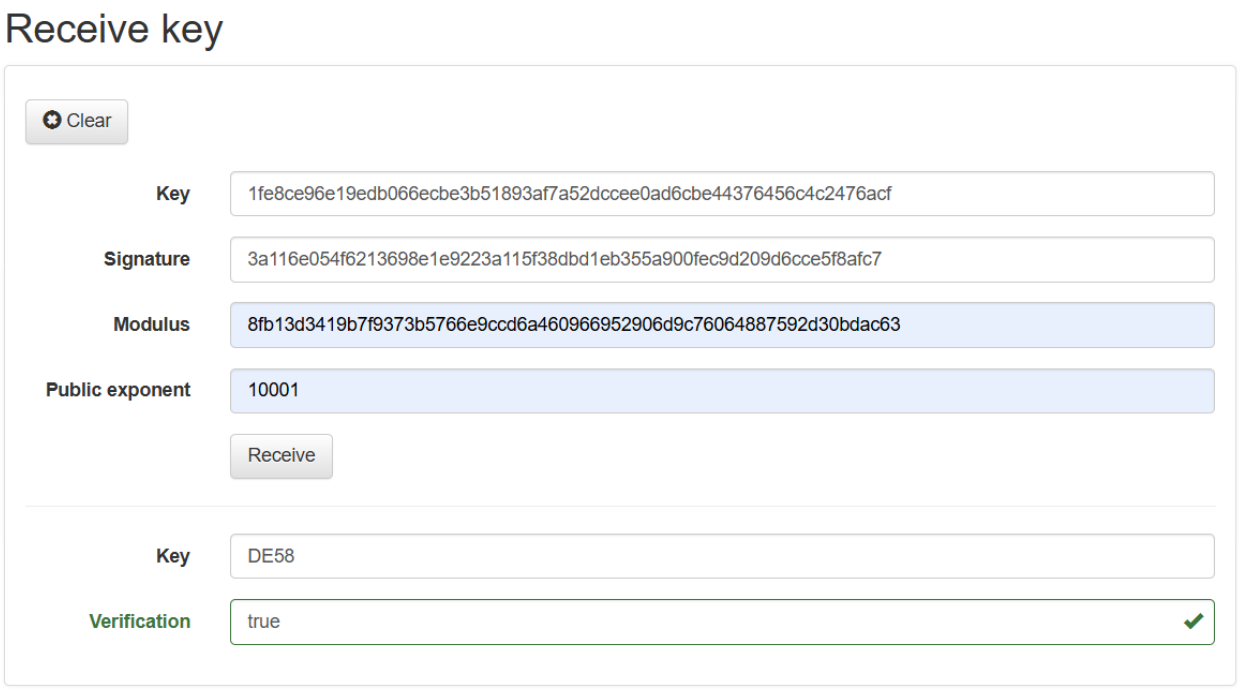
Modulus 8fb13d3419b7f9373b5766e9ccd6a460966952906d9c76064887592d30bdac63

Public exponent 10001

Send

Key 5EBB49656302C83D4BFFFD08DB229EC17E9CA397385FDBAB2126AE29268C7CAE

Signature 03BA3C5565470A54B10692AD458141BEC740652C3E3B00171963CCC30A34E846



Висновки:
Під час виконання практикуму було детально вивчено теоретичні основи криптосистеми RSA, методи перевірки чисел на простоту та алгоритми генерації параметрів асиметричних криптографічних систем.

У практичній частині були розглянуті різні тести простоти, зокрема тест Міллера–Рабіна та перевірка за ознакою подільності Паскаля. Це дозволило оцінити їх ефективність та обчислювальні витрати при генерації великих простих чисел.

Опановано також алгоритм Горнера для швидкого піднесення чисел до степеня за модулем, що є ключовим елементом багатьох криптографічних операцій.

Результатом роботи стало закріплення розуміння принципів побудови криптографічних систем, організації захищеного обміну інформацією та використання електронного підпису. Крім того, було відпрацьовано процедуру конфіденційної розсилки ключів у системах з асиметричними алгоритмами, а кожен етап перевірено за допомогою тестового середовища.