

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
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-33 Охріменко Анастасія

ФБ-33 Телегіна Софія

Перевірила:
Селюх Поліна Валентинівна

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.

```
def generate_prime(bits: int, k: int = 8, max_failed_store: int = 30) -> int:
    if bits < 2:
        raise ValueError("bits must be >= 2")
    lower = 1 << (bits - 1)
    upper = (1 << bits) - 1
    failed: List[int] = []

    while True:
        candidate = secrets.randbits(bits) | (1 << (bits - 1)) | 1
        if candidate > upper:
            candidate = candidate >> 1
        if is_probable_prime(candidate, k=k):
            if failed:
                print(f"Candidates that failed primality test (last {len(failed[-max_failed_store:])}): {failed[-max_failed_store:]} (total {len(failed)})")
            return candidate
        else:
            failed.append(candidate)
```

В функції `generate_prime` генерує випадкове число потрібної бітової довжини (використовується вбудований генератор `secrets`), робить його непарним, перевіряє на простоту через тест Міллера-Рабіна, повторює генерацію, поки не знайде просте число

```
def is_probable_prime(n: int, k: int = 8) -> bool:
    if n < 2:
        return False
```

```

if n in (2, 3):
    return True
if n % 2 == 0:
    return False

d = n - 1
r = 0
while d % 2 == 0:
    d //= 2
    r += 1

for _ in range(k):
    a = secrets.randbelow(n - 3) + 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
        if x == 1:
            return False
    else:
        return False
return True

```

Функція `is_probable_prime` попередньо спочатку відсіює елементарні випадки, предсатвляє числа у вигляді $n - 1 = 2^r * d$, далі виконується цикл: вибирається випадкове число ($a \in [2, n - 2]$), обчислюємо $x = a^d \bmod n$ та якщо $x == 1$ або $x == n - 1$ - число може бути простим. Наступний етап $a^{2^t d} \bmod n$: якщо в якийсь момент $x == n - 1$ число може бути простим; якщо $x == 1$ раніше ніж перше піднесення - це ознака складеного числа; якщо цикл завершився без $n - 1$ - число складене.

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 $pq \leq p_1q_1$; p і q – прості числа для побудови ключів абонента А, p_1 і q_1 – абонента В.

```

pub_A, sec_A = generate_keypair(bits=256)
pub_B, sec_B = generate_keypair(bits=256)

```

```

def generate_keypair(bits: int = 256) -> Tuple[Tuple[int,int],
Tuple[int,int,int]]:
    p = generate_prime(bits)

```

```

q = generate_prime(bits)
while q == p:
    q = generate_prime(bits)

n = p * q
phi = (p - 1) * (q - 1)
e = 65537
if math.gcd(e, phi) != 1:
    while True:
        e = secrets.randbelow(phi - 2) + 2
        if math.gcd(e, phi) == 1:
            break

d = modinv(e, phi)
if d is None:
    raise RuntimeError("Failed to find modular inverse for phi")

return (n, e), (d, p, q)

```

Генерація двох пар простих чисел p, q та p_1, q_1 за допомогою тесту Міллера–Рабіна

```

n_A = sec_A[1] * sec_A[2]
n_B = sec_B[1] * sec_B[2]

if n_A >= n_B:
    pub_A, pub_B = pub_B, pub_A
    sec_A, sec_B = sec_B, sec_A
    n_A, n_B = n_B, n_A

```

Перевірка й перестановка пар так, щоб $p \cdot q \leq p_1 \cdot q_1$. Обчислюється $n_A = p \cdot q$ і $n_B = p_1 \cdot q_1$. Якщо випадково вийшло $n_A > n_B$ (тобто $p \cdot q > p_1 \cdot q_1$), то міняються місцями ключі абонентів A та 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) , (e_1, n_1) та секретні d і d_1 .

```

def generate_keypair(bits: int = 256) -> Tuple[Tuple[int, int],
Tuple[int, int, int]]:
    p = generate_prime(bits)
    q = generate_prime(bits)
    while q == p:
        q = generate_prime(bits)

```

```

n = p * q
phi = (p - 1) * (q - 1)
e = 65537
if math.gcd(e, phi) != 1:
    while True:
        e = secrets.randbelow(phi - 2) + 2
        if math.gcd(e, phi) == 1:
            break

d = modinv(e, phi)
if d is None:
    raise RuntimeError("Failed to find modular inverse for phi")

return (n, e), (d, p, q)

```

У функції `generate_keypair` виконується генерація простих чисел p і q , далі відбувається обчислення модуля n , обчислення значення $\phi(n)$. Тепер береться стандартне значення $e = 65537$, якщо раптом $\gcd(e, \phi) \neq 1$, тобто e не є взаємно простим з $\phi(n)$, тоді програма: випадково підбирає інше число e , поки не знайде таке, що $\gcd(e, \phi) = 1$. Далі знаходження $d = e^{-1} \bmod \phi(n)$

4. Написати програму шифрування, розшифрування і створення повідомлення з цифровим підписом для абонентів А і В. 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 А и В, перевірити правильність розшифрування. Скласти для А і В повідомлення з цифровим підписом і перевірити його.

```

def encrypt(m: int, e: int, n: int) -> int:
    return pow(m, e, n)

```

На вхід подається лише відкритий ключ (e, n) і саме повідомлення M . Повертається криптограма $C = M^e \bmod n$.

```

def decrypt(c: int, d: int, n: int) -> int:
    return pow(c, d, n)

```

На вхід подається лише секретний ключ (d) та n . Повертає $M = C^d \bmod n$

```

def sign(m: int, d: int, n: int) -> int:

```

```
return pow(m, d, n)
```

Створюємо підпис $S = M^d \bmod n$

```
def verify_signature(m: int, s: int, e: int, n: int) -> bool:  
    return m == pow(s, e, n)
```

Декриптує підпис: $S^e \bmod n$

```
message = secrets.randbelow(min(n_A, n_B) - 1) + 1
```

Вибераємо випадкове повідомлення М датчиком випадкових чисел

```
encrypted_message = encrypt(message, pub_B[1], pub_B[0])  
encrypted_message = encrypt(message, e_receiver, n_receiver)
```

Шифрування повідомлення для А та В

```
decrypted_message = decrypt(encrypted_message, sec_B[0], pub_B[0])
```

Перевірка правильності розшифрування

```
signature = sign(message, sec_A[0], pub_A[0])  
  
signature = sign(message, d_sender, n_sender)  
encrypted_signature = encrypt(signature, e_receiver, n_receiver)
```

Створення повідомлення з цифровим підписом

```
signature_valid = verify_signature(decrypted_message, decrypted_signature,  
pub_A[1], pub_A[0])
```

Перевірка цифрового підпису

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 $0 < k < n$.

```
def sender_protocol(k: int, sender_public: Tuple[int,int], sender_secret:  
Tuple[int,int,int],  
                    receiver_public: Tuple[int,int]) -> Tuple[int,int]:  
    encrypted_message, encrypted_signature = send_key(k, sender_public,  
sender_secret, receiver_public)  
    return encrypted_message, encrypted_signature
```

Спочатку виконується підпис k, далі шифрування k, шифрування підпису та повернення encrypted_message, encrypted_signature

```
def receiver_protocol(encrypted_message: int, encrypted_signature: int,
                      sender_public: Tuple[int,int],
                      receiver_secret: Tuple[int,int,int],
                      receiver_public: Tuple[int,int]) -> Tuple[int,bool]:
    k, is_valid = receive_key(encrypted_message, encrypted_signature,
                              sender_public, receiver_secret, receiver_public)
    return k, is_valid
```

Відбувається розшифрування k, розшифрування підпису, перевірка підпису та повернення значень (k, is_valid)

- Кожну операцію рекомендується перевіряти шляхом взаємодії із тестовим середовищем, розташованим за адресою <http://asymcryptwebservice.appspot.com/?section=rsa>

Результат:

```
0633098671081962882308767528689336000203, 936826777004226648924461823215233476917746077247911641104588161890666934789, 106195175531971702958105420751888471664832063665087995388342343645514443202139, 9602655084548415081
447063710265680894826455945966772371789080389823138724286637, 70982684817829767915401224432088281011958176315301582748915119665456178155981, 1875294557165354707319337568964751889867870469638935577811220626714808189617999,
86507810537869113222568548556961888078278918407392758263283306339387499808261, 1141287291553749999998412798889065650479079076992915083956132688086633973223993, 11129752900768888674582357674283581997252778533076531762346
7418062096827673641] (total 144)
=== Subscriber A ===
n_A = 9786768224717501563241865396650978547879388186972017997624488143294518559702801590572933746511135874053072126081177895420408902193555644771797166461682991
e_A = 65537
d_A = 3233788885747249589575393978431678965874973697131071757089725671651778784051144141461814824922116988137138476829462929478927512390549131495128328434228753

=== Subscriber B ===
n_B = 1226201171398719231255956989980613129689799019659060597515999502201745073308575274889903277273879147350516732409666774219244995458743166577585390909366293
e_B = 65537
d_B = 1091681003134851032381193889107174438680320344544255264111686465594940892643894713642829371873748787097351040441655820381477671818347305414503404857382133

=== Original transmitted key (message) ===
message (k) = 758976369265495586229985528478968317261393425638571341158667147477679864222892145743359116987969714723253287670318809171172668462054694566520491633902830

=== Sender A forms ===
signature = 3621659829645676475719587086389943639456796264747448455307636812228496588773828429427716397975056008432505103183148404890131663618342783571564885672543151
encrypted_message = 61472866366981213371350970745088489945278338869548250155697257922592191403919019972969557871217597547468458438936038843375912416625961423568093470172387
encrypted_signature = 2382882534507819929681549786814542481468235910688617329270683654412716803326949803843113218173885125986619548335226683856507332385174474929022901952303415

=== Receiver B gets ===
received_key = 758976369265495586229985528478968317261393425638571341158667147477679864222892145743359116987969714723253287670318809171172668462054694566520491633902830
signature valid = True

Protocol completed successfully
```

Перевірки:

Encryption

Clear

Modulus

EA1F76FA74C44C9D8CA0E7CF937569250A8831F992D713F955836B708AD6E0663DD9D0BE61D45D9A0459C

Public exponent

10001

Message

90E9FDFAE0987337972A23EFA4CA6DF23B85C5A310ADBA55A8098F8C45C9

Bytes

Encrypt

Ciphertext

0BBCBB2A81583C76F3DE9CD072C04366DAE1C584AA7AC6E6823DB598D725427CC469E4D4869F8F43645C

Decimal to Hexadecimal converter

From

Decimal

To

Hexadecimal

Enter decimal number

61472866366981213371350970745088489

10

= Convert

× Reset

↕ Swap

Hex result

(614728663669812133713509707450884899452783388695482501556972579225921914039190199729695578712175975474604584389360388433759124166225061423568093470172387)₁₀ =
(BBCBB2A81583C76F3DE9CD072C04366


16

Hex number (127 digits)

BBCBB2A81583C76F3DE9CD072C04366DAE1C584AA7AC6E6823DB598D725427CC469E4D4869F8F436450BEB78010D2D2E4A25964128EDF581A5C9DEBE8EEA4E3

16

Verify

 Clear

Message

90E9FDFDAE0987337972A23EFA4CA6DF23B85C5A310ADBA55A8098F8C45C9

Bytes

Signature

45264DF36E1292426A5610C1AE17D18B79FAEC1C719E773ABB80A3219BBBF78F18AD97189E234C575E289

Modulus

BADCB000671B8CF91BC7F5F10BC3E5770B5245014BEB721E8A1D9A1DB144B57B1E37B2FBFA55BAEB6F6

Public exponent

10001

Verify

Verification

true

✓

Висновки:

У ході роботи було реалізовано алгоритм RSA з генерацією простих чисел, обчисленням модуля n , публічного (e) та приватного (d) ключів, а також операцій шифрування, дешифрування, підпису та перевірки підпису. Було реалізовано роботу протоколу у вигляді окремих процедур для відправника (`sender_protocol`) та отримувача (`receiver_protocol`). Ща допомогою сайту було перевірено шифрування та перевірка підпису.