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 роботу:
Студенти 3 курсу
групи ФБ-35
Кохта А.
Церман М.
Ворона С.

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 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 1 1 p , q довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб pq p1q1 ; p і q – прості числа для побудови ключів абонента A, 1 p 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) , (,) 1 n1 e 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

Написали функцію для генерації випадкового простого числа заданої довжини у бітах. Для перевірки числа на простоту було використано алгоритм Міллера-Рабіна якщо число просте то вона повертає True, якщо ні то відповідно False.Алгоритм був реалізований за описом у розділі 4.3 "Імовірнісний тест Міллера-Рабіна" в методичці

Також додали додаткові попередні ділення на малі прості числа.

```
def power(a, b, m):
    res = 1
    a %= m
    while b > 0:
        if b % 2 == 1:
            res = (res * a) % m
        a = (a * a) % m
        b //= 2

    return res
```

```python
def miller_rabin_test(n, k=40):
    if n == 2 or n == 3:
        return True
    if n % 2 == 0 or n < 2:
        return False

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

    for _ in range(k):
        a = random.randrange(2, n - 1)
        x = power(a, d, n)

        if x == 1 or x == n - 1:
            continue

        for r in range(s - 1):
            x = power(x, 2, n)
            if x == n - 1:
                break
        else:
            return False

    return True
```

```python
def get_random_prime(bits):
    while True:
        candidate = random.getrandbits(bits)
        candidate |= (1 << (bits - 1)) | 1

        small_primes = [3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53]
        is_composite = False
        for p in small_primes:
            if candidate % p == 0 and candidate != p:
                is_composite = True
                break

        if is_composite:
            continue

        if miller_rabin_test(candidate):
            return candidate
```

[2]

```python
p, q, p1, q1 = get_random_prime(256), get_random_prime(256), get_random_prime(256), get_random_prime(256)
```

[3]

Пункт 2, викликали попередню функцію для генерації 4х ключів p, q, p1, q1, та написали ф-ю check_keys() аби перевірити чи задовільняють вони умові pq ≤ p1q1. Якщо ні то повертаємо їх же але поміняні місцями, тобто p = p1 q = q1 і навпаки. Якщо умова виконується то залишаємо як є та виводимо готові прості числа для подальшої роботи.

```python
def check_keys(p, q, p1, q1):
    n = p * q
    n1 = p1 * q1

    if n > n1:
        return (p1, q1), (p, q), n1, n
    else:
        return (p, q), (p1, q1), n, n1
```

[4]

```python
pair_A, pair_B, mult_A, mult_B = check_keys(p, q, p1, q1)
print("Пара для абонента A:")
print(f"p: {pair_A[0]}\n"
      f"q: {pair_A[1]}\n"
      f"pq: {mult_A}\n")

print("Пара для абонента B:")
print(f"p1: {pair_B[0]}\n"
      f"q1: {pair_B[1]}\n"
      f"pq1: {mult_B}\n")

print("pq < pq1:", ["Ні", "Так"][mult_A <= mult_B])
```

[5]

```
Пара для абонента A:
p: 960785320150623519185405923308862592988537820819097645207216241517833438040 07
q: 769414141583740526948555889550988067207984363887272684473809948013139868046 29
pq: 739241812349951314697598064559265887563957357478956180247122551833194340788292718880967901087489215012624003966037823141340394855949279458053208357634840 3

Пара для абонента B:
p1: 8903533682378518358541079203804889381492307872517008751280344063281226850622 1
q1: 1003372306165103379797128784712959328666760669320096241167279824742665378506 01
pq1: 8933559123906809033370966558344309483453674012609457799523448164981173307290627110312430019898726527933942786938119854773687517564414337292345030337088821

pq < pq1: Так
```

Пункт 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.

Для перевірки чи є число *e* взаємопростим з phi, а також для обчислення оберненого число до *e*, використали розширений алгоритм Евкліда.

```python
def euclid_extended(a, n):
    r0, r1 = a, n
    u0, v0 = 1, 0
    u1, v1 = 0, 1

    while r1 != 0:
        q = r0 // r1
        (r0, r1) = (r1, r0 - q * r1)

        (u0, u1) = (u1, u0 - q * u1)
        (v0, v1) = (v1, v0 - q * v1)

    gcd = r0
    u = u0
    v = v0

    return gcd, u, v

def generate_key_pair(p, q):
    n = p * q
    phi = (p - 1) * (q - 1)
    e = 65537
    gcd, u, _ = euclid_extended(e, phi)

    while gcd != 1:
        e = random.randrange(3, phi, 2)
        gcd, u, _ = euclid_extended(e, phi)

    d = u % phi

    return (e, n), (d, p, q)
[6]
```

```python
public_A, private_A = generate_key_pair(pair_A[0], pair_A[1])
public_B, private_B = generate_key_pair(pair_B[0], pair_B[1])

for pub, priv, name in [(public_A, private_A, "A"), (public_B, private_B, "B")]:
    e, n = pub
    d, p, q = priv

    print(f"Абонент {name}:")
    print("Відкритий ключ")
    print(f"e: {e}\n"
          f"n: {n}\n")

    print("Приватний ключ")
    print(f"d: {d}\n"
          f"p: {p}\n"
          f"q: {q}\n")
```

[7]

```
Абонент A:
Відкритий ключ
e: 65537
n: 75846596053820650659028737813261741190935801482874309191754132961012401207065507050432073496973087338315992407163359690303281061439908844218664707295564 39

Приватний ключ
d: 46074807850446430932713613833736308349383342671104749951052314592426045086830660945285705069514226247587980875982462121015701218632724855243552168813265 93
p: 703030082525137811000901056582822553788073162929213602128071206180758567510 41
q: 107885278225073179461317321405886951707136533170091933549782219120698480976279
```

```
Абонент B:
Відкритий ключ
e: 65537
n: 883378928289701923792557555592531800427059362977250257492425605955684044317447942332893904814209317360575540143404110996352203015817523250570601275228363 1

Приватний ключ
d: 307107519449357899075173281453533798326595976716567280569867784703210480579341520427746745091512162103291401267941701547767854488807847203800062121224192 1
p: 899731777812625077526405348836297419306782761828600306077109472831958232916 09
q: 981824750524340444385984837015799360117245524805502043758597802255419822443 59
```

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 М і знайти криптограму для абонентів А и В, перевірити правильність розшифрування. Скласти для А і В повідомлення з цифровим підписом і перевірити його.

```python
def Encrypt(message, public_key):
    e, n = public_key
    if message >= n:
        raise ValueError("Помилка: Повідомлення M має бути меншим за модуль n")

    return power(message, e, n)

def Decrypt(ciphertext, private_key):
    if len(private_key) == 3:
        d, p, q = private_key
        n = p * q
    else:
        d, n = private_key

    return power(ciphertext, d, n)

def Sign(message, private_key):
    if len(private_key) == 3:
        d, p, q = private_key
        n = p * q
    else:
        d, n = private_key

    return power(message, d, n)

def Verify(message, signature, public_key):
    e, n = public_key
    check_val = power(signature, e, n)

    return check_val == message
```

[8]

```python
subscribers = [("A", public_A, private_A), ("B", public_B, private_B)]

for name, pub_key, priv_key in subscribers:
    # Генерація повідомлення
    n_module = pub_key[1]
    message = random.randint(100, n_module // 2)
    print(f"Відкрите повідомлення {name}: {message}")

    # Шифрування
    crypto = Encrypt(message, pub_key)
    print(f"Криптограма {name}: {crypto}")

    # Розшифрування
    decrypted = Decrypt(crypto, priv_key)
    print(f"Розшифроване повідомлення {name}: {decrypted}")
    print(["Не співпадають\n", "Співпадають\n"][decrypted == message])

    # Цифровий підпис
    signature = Sign(message, priv_key)
    print(f"Цифровий підпис повідомлення {name}: {signature}")

    # Перевірка підпису
    is_valid = Verify(message, signature, pub_key)
    status = (["!Невалідний!", "!Валідний!"][is_valid])
    print(f"Підпис {name}: {status}\n")
[9]
```

```
Відкрите повідомлення A:
 19576983815931931881534395280291618492980639921556396784633143518185513747240072141539171888273567960722822492035915412583763863494139901052301984956960 54
Криптограма A:
 44038377267454932187390841649743341646508683698782242076323622529657353628854533881794161556456879209060338436664565514129263235785677337613036062816335 85
Розшифроване повідомлення A:
 19576983815931931881534395280291618492980639921556396784633143518185513747240072141539171888273567960722822492035915412583763863494139901052301984956960 54
Співпадають

Цифровий підпис повідомлення A:
 70477030152363389774036401634661832562807112436813378587969747540754122033934173210339312595367688542489563126406087462245953234579470155465140806527093 90
Підпис A: !Валідний!
```

```
Відкрите повідомлення B:
 23670409578073578348377653408022153177060665612164607224221706226606068029020855268429249748306592650908073912124773522443786768770609210175558802705825 69
Криптограма B:
 25343567133879296209917540350752629937450274840176366318570703608475360027488019348976313720371450915114123833275151467377763362554170322948119255045893 11
Розшифроване повідомлення B:
 23670409578073578348377653408022153177060665612164607224221706226606068029020855268429249748306592650908073912124773522443786768770609210175558802705825 69
Співпадають

Цифровий підпис повідомлення B:
 88757209034508012637393852863947632111833292031092387980053275241524976646697506562156396228926988189004133829395314161278254197489215718224491183976822 2
Підпис B: !Валідний!
```

Перевірка для абонента А:

Перевірка для абонента В:

Пункт 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.
Відправник створює цифровий підпис, та шифрує ключ і свій підпис публічним ключем отримувача.
Отримувач розшифровує ключ та підпис, а також перевіряє підпис.

```python
def SendKey(k, private_key_A, public_key_B):
    S = Sign(k, private_key_A)
    k1 = Encrypt(k, public_key_B)
    S1 = Encrypt(S, public_key_B)

    return k1, S1

def ReceiveKey(package, private_key_B, public_key_A):
    k1, S1 = package
    k = Decrypt(k1, private_key_B)
    S = Decrypt(S1, private_key_B)
    is_valid = Verify(k, S, public_key_A)

    if is_valid:
        return k, "Підпис вірний, ключ отримано!"
    else:
        return None, "Підпис невірний!"
```

Перевірити роботу програм для випадково обраного ключа 0 < k < n.

```python
# 0<k<n
n_a = public_A[1]
secret_k = random.randint(100, n_a // 2)
print(f"Секретний ключ k: {secret_k}\n")

print("Відправляє A:")
package = SendKey(secret_k, private_A, public_B)
print(f"Шифрований ключ: {package[0]}\n"
      f"Цифровий підпис: {package[1]}\n")

print("Отримує B:")
received_k, status_msg = ReceiveKey(package, private_B, public_A)
print(f"Відкритий ключ k: {received_k}")
print(status_msg)
```

```
Секретний ключ k:
 1166021370358548910534139437832066676205599666972718386930672141764087995839678136982221827392933074009773661105691080649391187667003361219811711852658008

Відправляє A:
Шифрований ключ:
 2011732000277930813712303628419389389825320163119108455577868423749997247877138821645034320125523299831517960107127413262180389873341618925522717387557638
Цифровий підпис:
 9723585410829702512806086808507417444898682069484727294657704116586590933996382278454653855855673465838261009542375612778047838668666626241357065901697 4
```

```
Отримує B:
Відкритий ключ k:
 1166021370358548910534139437832066676205599666972718386930672141764087995839678136982221827392933074009773661105691080649391187667003361219811711852658008
Підпис вірний, ключ отримано!
```

Висновки: В процесі виконання лабораторної роботи ми на практиці ознайомилися з побудовою асиметричної криптосистеми на базі алгоритму RSA, було розроблено функції для генерації великих та простих чисел, створення пар ключів, а також процедури шифрування та розшифрування даних, перевірили роботу наших алгоритмів за допомоги сайту dCode, а також зрозуміли, як працюють цифрові підписи.