

КРИПТОГРАФІЯ

КОМП'ЮТЕРНИЙ ПРАКТИКУМ №4

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

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_with_len(bits):  
    while True:  
        num = random.getrandbits(bits)  
        if miller_rabin_test(num) is True:  
            return num
```

Для реалізації цієї функції використаємо тест Міллера Рабіна:

```
def miller_rabin_test(p, k=30):  
    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, 101, 103,  
              107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223,  
              227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293]  
  
    if p % 2 == 0:  
        return False  
  
    # попереднє ділення  
    for num in primes:  
        if p % num == 0:  
            return False  
  
    # знаходимо  
    s = 0  
    d = p - 1  
    while d % 2 == 0:  
        d //= 2  
        s += 1  
  
    # print(d)  
    for i in range(k):  
        x = random.randrange(2, p - 1)  
        gcd = gcd_euclid(x, p)  
        # print(gcd)  
        if gcd != 1:  
            return False  
        # тест Міллера-Рабіна  
        result = 1  
        binary_exp = bin(exp)[2:]  
        for bit in binary_exp:  
            result *= 2  
            result %= modulo  
            if bit == '1':  
                result *= a  
                result %= modulo  
        if result != 1 and result != -1:  
            return False  
    return True  
  
def horner_scheme(a, exp, modulo):  
    binary_exp = bin(exp)[2:]  
    result = 1  
    for bit in binary_exp:  
        result *= 2  
        result %= modulo  
        if bit == '1':  
            result *= a  
            result %= modulo  
    return result  
  
def gcd_euclid(x, y):  
    while y:  
        x, y = y, x % y  
    return abs(x)
```

А також
функцію
схеми
Горнера
та
алгоритм
Евкліда:

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

Тепер можемо згенерувати числа p, q :

```
def gen_pq(self):  
    self.p = generate_prime_with_len(256)  
    self.q = generate_prime_with_len(256)
```

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 А і В – тобто, створити та зберегти для подальшого використання відкриті ключі (e, n) , (e_1, n_1) та секретні d і d_1 :

- 4.
- ```
def generate_key_pair(self):
 n = self.p*self.q
 phi = (self.p-1) * (self.q-1)
 e = pow(2, 16) + 1
 d = pow(e, -1, phi)
 # print(f"n == {n}\nphi == {phi}\ne == {e}\nd == {d}")
 return [d, (n, e)]
```

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.

```

class Abonent:
 def __init__(self, uuid):
 self.uuid = uuid
 self.gen_pq()
 self.private_key, self.public_key = self.generate_key_pair() # [d, (n, e)] return from gen_key
 self.public_key_other = None
 self.message = None

 # генерація ключових пар
 def generate_key_pair(self):
 n = self.p*self.q
 phi = (self.p-1) * (self.q-1)
 e = pow(2, 16) + 1
 d = pow(e, -1, phi)
 # print(f"n == {n}\nphi == {phi}\ne == {e}\nd == {d}")
 return [d, (n, e)]

 def gen_pq(self):
 self.p = generate_prime_with_len(256)
 self.q = generate_prime_with_len(256)

 def generate_message(self):
 self.message = random.getrandbits(128)

 def recieve_public_key(self, key):
 self.public_key_other = key

 def send_public_key(self):
 key = self.public_key
 return key

 def send_message(self):
 # шифруємо відкритим ключем абонента В
 cypher = rsa_cypher(message=self.message, e=self.public_key_other[1], n=self.public_key_other[0])
 signature = sign(message=self.message, d=self.private_key, n=self.public_key[0])
 return cypher, signature

 def recieve_message(self, var):
 message = var[0]
 signature = var[1]
 # розшифровуємо своїм d та n
 decypher_text = rsa_decypher(message=message, d=self.private_key, n=self.public_key[0])
 # верифікація за допомогою відкритого ключа абонента В
 verif = verify(message=decypher_text, signature=signature, e=self.public_key_other[1], n=self.public_key_other[0])

 return verif, decypher_text

```

реалізуємо клас який буде в собі містити всі необхідні методи для шифрування та розшифрування повідомлень

також реалізуємо інтерфейс для взаємодії з цими функціями:

```

def main():
 is_run = True
 abonents = []

 def get_abonent():
 current_abonent = None
 id = int(input('abonent`s uuid: '))

 for i in abonents:
 if id == i.uuid:
 current_abonent = i

 if current_abonent == None:
 raise Exception('no abonent with such id')

 return current_abonent

 print('input h or help to see all commands')
 while is_run:
 command = input('\nyour command: ')
 if command == 'h' or command == 'help':
 print('C, Create abonent with p,q')
 print('S, Show info')
 print('SM, Send msg')
 print('GM, Generate msg')
 print('RM, Receive msg')
 print('Q, Quit program')

 elif command == 'C':
 uuid = len(abonents)+1
 abonent = Abonent(uuid=uuid)
 abonents.append(abonent)
 print('abonent created with uuid ', uuid)

```

```

 elif command == 'S':
 if (len(abonents) == 0):
elif command == 'RM':
 print('who should receive?')
 receiver = get_abonent()

 cypher = int(input('cypher: '))
 signature = int(input('signature: '))

 verif, decypher_text = receiver.recieve_message(var=[cypher, signature])

 if verif is True:
 print(f"Message : {hex(decypher_text)}")
 else:
 print("Signature does not match")

elif command == 'Q':
 is_run = False
else:
 print('unknown command, write h to see all commands')

 receiver = get_abonent()

 current_abonent.recieve_public_key(receiver.send_public_key())
 receiver.recieve_public_key(current_abonent.send_public_key())

 res = current_abonent.send_message()
 print('message was sent: ', res)

```

Запустимо нашу програму та перевіримо її функціональність:

```

input h or help to see all commands

your command: h
C, Create abonent with p,q
S, Show info
SM, Send msg
GM, Generate msg
RM, Receive msg
Q, Quit program

your command: █

```

створимо 2-ох користувачів та переглянемо інформацію про них:

```

your command: C
abonent created with uuid 1

your command: C
abonent created with uuid 2

your command: S
uuid: 1
public key: (787950258425116914492196214438155195408767755298661443595852803243578496289936252081937567947550161833134898259159947962076012942251868155832661917200729, 65537)

uuid: 2
public key: (431963404051692840014956706033669500317971750452071880313136345989256459029222123865341175169187933116471434131085510020058442812763737087181226873455249, 65537)

```

Згенеруємо повідомлення для 1:

```
your command: GM
abonent`s uuid: 1
message was generated: 0xdd511dcd4ef670d24fd0d59d93b7ee2c
```

Надішлемо повідомлення від 1 користувача до 2:

```
your command: SM
who should send?
abonent`s uuid: 1
who should receive?
abonent`s uuid: 2
message was sent: (332136028898514996522780937403271724026855445317486082534592589304899712418490327608084328948212144164269196646095589388965373637644029177046180692913046, 688900963994661132385766236790717527495486144374605753939377939364492134866661271856950560976802548597206132013883433286941504784994205484798915223080242)
```

Приймемо та розшифруємо повідомлення 1 користувача яке він надіслав для 2:

```
your command: RM
who should receive?
abonent`s uuid: 2
cypher: 332136028898514996522780937403271724026855445317486082534592589304899712418490327608084328948212144164269196646095589388965373637644029177046180692913046
signature: 688900963994661132385766236790717527495486144374605753939377939364492134866661271856950560976802548597206132013883433286941504784994205484798915223080242
Message : 0xdd511dcd4ef670d24fd0d59d93b7ee2c
```

Як бачимо ми отримали те саме повідомлення, яке і надіслали

Також перевіримо роботу наших алгоритмів за допомогою стороннього сервісу для цього напишемо функцію для взаємодії з цим сайтом:

```
def site_vefify():
 p = generate_prime_with_len(256)
 q = generate_prime_with_len(256)
```

```
E:\PyCharm\Projects\Scripts\python.exe C:\Users\nazar\PycharmProjects\pythonProject\Lab4crypto.py
d = 0x4bf2576c9823b4020a4e9166906339393af9c68dfb0410228f68c1803fe08d25ffad5fb9515e7455704818b039f67512e39e0c79856f04e333a2924be3f93591
n = 0x5d6b07600462273c101f9efd6ee713b9126a12438051b0ef7eeab49cda19fe4bd026d38ec37b35238ef7d0cca67d298ea9518f784ac4220079bdbc5893dc23
e = 0x10001
```

```
print(f"d = {hex(d_test)}")
print(f"n = {hex(n)}")
print(f"e = {hex(e)}")
server_cypher = int(input("enter server cypher: "), 16)

print(type(server_cypher), type(d_test), type(n))
decrypted_server_cypher = rsa_decypher(server_cypher, d_test, n)
print(hex(decrypted_server_cypher))

message = 0x200320036969 # random msg
ds = sign(message=message, d=d_test, n=n)

print(hex(ds))
```

згенеруємо приватний та публічний ключі:

шифруємо на сайті відкритим ключем повідомлення:

## Encryption

Modulus

5d6b07600462273c1c101f9efd6ee713b9126a12438051b0ef7eeab49cda19fe4bd026d38ec37b35238ef7d0cca67d:

Public exponent

10001

Message

69696969

Bytes

Ciphertext

4328835CDBF6FCCD7A4820555299F7707EF89CAF61CA03BA4943E99F8F9877405CD04D478454BDD523B97

розшифровуємо секретним ключем:

```
enter server cypher: 4328835CDBF6FCCD7A4820555299F7707EF89CAF61CA03BA4943E99F8F9877405CD04D478454BDD523B97A1A79F8A48B8D133F1A80007ECC31A8A9E980D48
decrypted message 0x69696969
```

перевіримо цифровий підпис та повідомлення

```
message = 0x200320036969
```

```
digital sign 0x42ed5a7455cb4678b8ed5fc61a35f26292214d0e388b1e4aabb91b583b164bfa593e4f0ae0bab52adaef91a1eef5f0ba49341a5be6dde720cda1552d5c64b2f4
```

перевіримо сам підпис через сайт:

## Verify

Message

200320036969

Bytes

Signature

42ed5a7455cb4678b8ed5fc61a35f26292214d0e388b1e4aabb91b583b164bfa593e4f0ae0bab52adaef91a1eef5f0t

Modulus

5d6b07600462273c1c101f9efd6ee713b9126a12438051b0ef7eeab49cda19fe4bd026d38ec37b35238ef7d0cca67d

Public exponent

10001

Verify

Verification

true

✓

отримаємо ключ на сайті:

## Get server key

Key size

256

Get key

Modulus

A71F1E8A2304024FF99162FCA5F05848D5600A2E9BBD0966DF7EEFD520AB5A4

Public exponent

10001



локально зашифруємо на сервері:

```
m = 0xA71F1E8A2304024FF99162FCA5F05848D5600A2E9BBD0966DF7EEFD520AB5A4F
mes = rsa_cypher(0x6969, 0x10001, m)
print(hex(mes))
```


lab4crypto ×

E:\PyCharm\Projects\Scripts\python.exe C:\Users\nazar\PycharmProjects\pythonProject\lab4crypto.py  
0x5874113a75711fea0dc15d0c14e20b5cf238a6ded588aee11d0588325a2c3808


Process finished with exit code 0

розшифруємо на сайті:

## Decryption

 Clear

**Ciphertext**  
  

Bytes 

Decrypt

**Message**





