Тарасов Микита ФБ-12
Гранік Микита ФБ-12

Лабораторна робота №4

Мета: 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.

У якості випадкових чисел заданої довжини (256 біт) використали secrets.randbits(256), які потім будуть проходити тести на перевірки простоти через тест Міллера-Рабіна.

```python
def pick_random():
    random = secrets.randbits(256)
    if (miller_rabin_test(random) == False):
        return pick_random()
    else:
        return random
```

```python
def miller_rabin_test(n, k=5):
    if n == 2 or n == 3:
        return True
    if n % 2 == 0:
        return False

    # Знаходження чисел s і d, таких, що n - 1 = 2^s * d
    s, d = 0, n - 1
    while d % 2 == 0:
        s += 1
        d //= 2

    # Проведення k ітерацій тесту Міллера-Рабіна
    for _ in range(k):
        a = random.randint(2, n - 2)
        x = pow(a, d, n)

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

        for _ in range(s - 1):
            x = pow(x, 2, n)
            if x == n - 1:
                break
        else:
            return False  # Число не є простим

    return True  # Число, можливо, просте
```

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 А, 1 p і q1 – абонента В.

Для цього використали ф-цію generate_pq(). У якості аргументів до ф-ції передаємо обєкти класів абонентів (у нашому випадку Абонент А і Б). Генеруються 4 простих числа довжини 256 бітів, що поміщаються у список. Цей список потім сортується, і перші два числа ми додаємо як p and q для абонента А, і p1 and q1 для абонента Б. Таким чином, умова **pq ≤ p1q1** буде виконана.

```
def generate_pq(a, b):
    pq = []
    for i in range (0, 4): pq.append((pick_random()))
    pq = sorted(pq, reverse=True)
    greatest = pq[:2]
    least = pq[2:]
    print("\nGenerating p, q for abonent A:\np: {}\nq: {}".format(least[0], least[1]))
    print("\n\nGenerating p1, q1 for abonent B:\np: {}\nq: {}\n".format(greatest[0], greatest[1]))
    a.p = least[0]
    a.q = least[1]
    b.p = greatest[0]
    b.q = greatest[1]
    return True
```

```
Generating p, q for abonent A:
p: 867572564817946538241565164204432018490364866341854651328470266535575936578 59
q: 39589556949409057315351140956474458749641621718630666469021064408392251571487


Generating p1, q1 for abonent B:
p: 111161466643896406085505734663680529057014926484476432448055479736629366488589
q: 935650610636239996845180727356980224257391603687740488574671507614415092607 21
```

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) , (,) 1 n1 e та секретні d і d1 .

У якості абонентів ми використовували обєкти класу Abonent(). На коді вище вже можна бачити призначені атрибути p and q.

```
def generate_key_pair(abonent):
    global e
    abonent.e = e
    abonent.phi_n= ((abonent.q) - 1) * ((abonent.p)-1)
    abonent.n = abonent.q * abonent.p
    abonent.d = mod_inverse(e, abonent.phi_n)
```

На коді вище призначаємо додаткові атрибути, що потрібні для подальшої роботи. Також додамо більш узагальнені атрибути для ключів, наприклад:

```
print("Public key of Abonent A: (e: {}, n: {})\n".format(e, hex(Abonent_A.n)))
Abonent_A.public_key = (Abonent_A.e, Abonent_A.n)
Abonent_A.private_key = (Abonent_A.d, Abonent_A.p, Abonent_A.q)
print("Public key of Abonent B: (e: {}, n: {})\n".format(e, hex(Abonent_B.n)))
Abonent_B.public_key = (Abonent_B.e, Abonent_B.n)
Abonent_B.private_key = (Abonent_B.d, Abonent_B.p, Abonent_B.q)
```

На коді вище кожному обєкту додається атрибути public_key and private_key.

```
Public key of Abonent A: (e: 65537, n: 0×41945f69b5d0ff64fcc7d9ab7072bc54a35bb6fc82bcb0f1855
b5416b8f49fc0f1dd8491eb2daf8e38d76352ddb918825a5f98fee3dbf7dc68174b544945e55d)

Public key of Abonent B: (e: 65537, n: 0×c69631f77380493138d70e4f5990c097c6af872e8cd202d9445
fb00df81971509b220b07583357c103eea20b278ac990d4e42f48f9a8dd53e38cfaa1dc2dbffd)
```

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

Реалізовані функції: receive_key(), send_key(), sign_message(), verify_message(), encrypt(), decrypt()

```
    def receive_key(self, public_key):
        #print("Received public key:", public_key)
        self.received_e = public_key[0]
        self.received_n = public_key[1]
        self.public_key_received = (public_key)

    def send_key(self):
        return self.e, self.n
```

```
def sign_message(self, cleartext):
    signed = pow(cleartext, self.d, self.n)
    return str(hex(signed))

#VERIFY DIGITAL SIGNATURE
def verify_message(self, cleartext, signature):
    if pow(signature, self.received_e, self.received_n) == cleartext:
        return True
    else:
        return False
```

```python
def encrypt(self, cleartext):
    print("[+]"+"-"*100+"[+]\n")
    #JUST SOME STUFF TO CONVERT CLEARTEXT TO INT
    print("\nCleartext message to encode:", cleartext)
    cleartext_bytes = cleartext.encode('utf-8')
    print("Cleartext message to encode in bytes:", cleartext_bytes.hex())
    cleartext_int = int.from_bytes(cleartext_bytes, byteorder='big')
    #ENCRYPTING USING RECEIVED PUBLIC KEY OF ANOTHER ABONENT
    print("Encrypting with received public key...")
    encrypted = pow(cleartext_int, self.received_e, self.received_n)
    #[2:] to remove 0x
    result = hex(encrypted)+"."+self.sign_message(cleartext_int)[2:]
    print("Encrypted using recieved public key:", result[2:])
    return result
```

На коді вище для спілкування між абонентами (без серверу), ми реалізували додавання сигнатури через «.» до основного зашифрованого повідомлення. Таким чином, і на ф-ції нижче для дешифрування буде дві реалізації дешифрування: як для зашифрованого тексту із сигнатурою, так і без:

```python
def decrypt(self, encrypted):
    #CUSTOM REALIZATION
    if "." in encrypted:
        print("[+]"+"-"*100+"[+]\n")
        print("\nEncrypted message to decode (hex):", encrypted[2:])
        #DECRYPTING USING OWN PRIVATE KEY VALUES
        cleartext = pow(int(encrypted.split('.')[0], 16), self.d, self.n)
        #VERYFYING DIGITAL SIGNATURE
        verified = self.verify_message(cleartext, int(encrypted.split('.')[1], 16))
        if verified:
            print("\nSignature verified")
            print("Decoded hex values:", hex(cleartext))
            byte_string = bytes.fromhex(hex(cleartext)[2:]).decode('utf-8')
            print("Cleartext plaintext: {}".format(byte_string))
            return hex(cleartext)
        else:
            print("Corrupted signature")
            return None
    #INTERACTION WITH THE SERVER
    else:
        print("[+]"+"-"*100+"[+]\n")
        print("\nEncrypted message to decode (hex):", encrypted)
        #DECRYPTING USING OWN PRIVATE KEY VALUES
        cleartext = pow(int(encrypted, 16), self.d, self.n)
        print("Decoded hex values:", hex(cleartext))
        byte_string = bytes.fromhex(hex(cleartext)[2:]).decode('utf-8')
        print("Cleartext plaintext: {}".format(byte_string))
        return hex(cleartext)
```

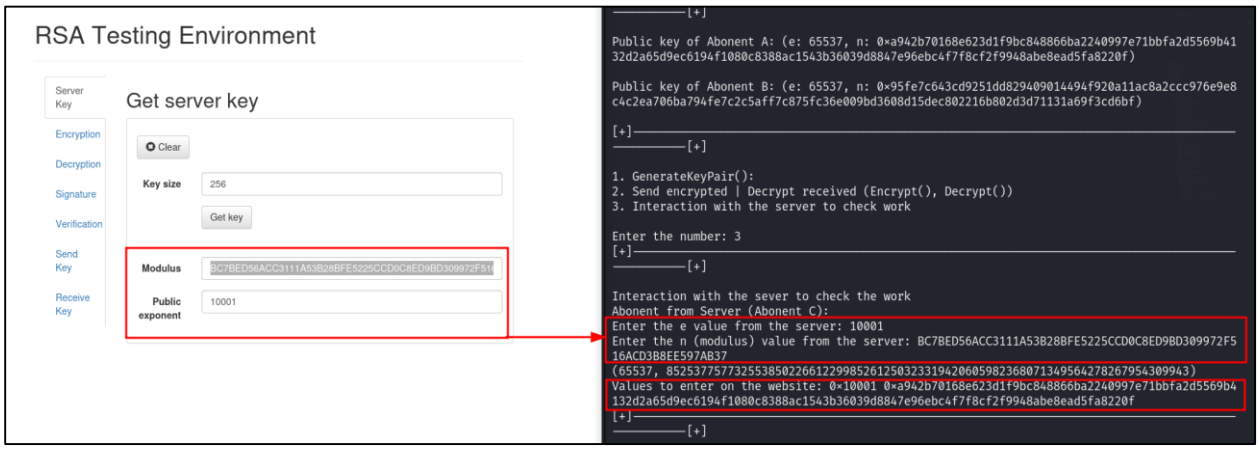
```
2. Send encrypted | Decrypt received (Encrypt(), Decrypt())
3. Interaction with the server to check work

Enter the number: 2
Enter the message to encrypt: ABOBA
1. A → B
2. B → A
1

Encrypting message from Abonent A with public key of Abonent B
[+]————————————————————————————————————————————————————————————————————————————————
——————————[+]


Cleartext message to encode: ABOBA
Cleartext message to encode in bytes: 41424f4241
Encrypting with received public key...
Encrypted using recieved public key: bc0ed7abe8dafb41a73881daf66801919524553c8d72e2ca893b9e6
daa13bc908138e553036edcf208481ff3b239c529e18a230999abf89e9f2be4eda8f7079f.1214bb668d32e80f48
8d608a93adbdce76f699c3ccc502109256bd78ddd5fe409fc84ec7cb9d056aa38e1fef0a3c22db9fb376320033f5
dd429ad31e2f013c3

Decrypting on the Abonent B with private ley B:
[+]————————————————————————————————————————————————————————————————————————————————
——————————[+]


Encrypted message to decode (hex): bc0ed7abe8dafb41a73881daf66801919524553c8d72e2ca893b9e6da
a13bc908138e553036edcf208481ff3b239c529e18a230999abf89e9f2be4eda8f7079f.1214bb668d32e80f488d
608a93adbdce76f699c3ccc502109256bd78ddd5fe409fc84ec7cb9d056aa38e1fef0a3c22db9fb376320033f5dd
429ad31e2f013c3

Signature verified
Decoded hex values: 0×41424f4241
Cleartext plaintext: ABOBA
1. GenerateKeyPair():
2. Send encrypted | Decrypt received (Encrypt(), Decrypt())
3. Interaction with the server to check work

Enter the number:
```

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

Перевіримо ф-ції шифрування, дешифрування, перевірка підпису та створення підпису через роботу із сервером. Для цього, на сервері генеруємо публічний ключ, створюємо абонента_С у якості абонента серверу, додаємо йому необхідні параметри e and n, і потім із абонента_С ділимось публічним ключем із абонентом_А. Таким чином, подальше спілкування із сервером буде можливим:

```
print("Interaction with the sever to check the work")
print("Abonent from Server (Abonent_C):")
Abonent_C = Abonent("Abonent C")
var1 = str(input("Enter the e value from the server: "))
var2 = str(input("Enter the n (modulus) value from the server: "))
Abonent_C.e = int(var1, 16)
Abonent_C.n = int(var2, 16)
Abonent_A.receive_key(Abonent_C.send_key())
print(Abonent_A.public_key_received)
print("Values to enter on the website:", hex(Abonent_A.public_key[0]), hex(Abonent_A.public_key[1]))
```

Також, виведемо наш публічний ключ для абоненту_А для відправки його на сервер:

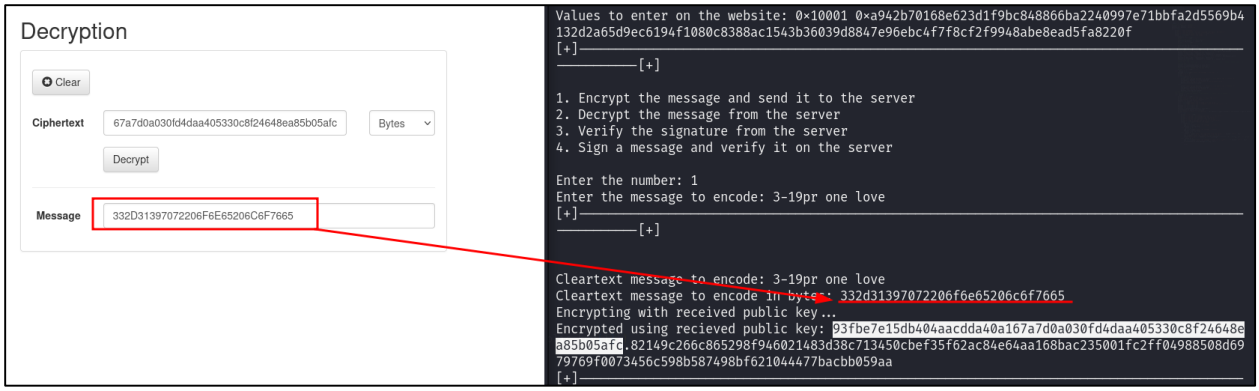


Після цього, можемо виконати наступні дії:

```
1. Encrypt the message and send it to the server
2. Decrypt the message from the server
3. Verify the signature from the server
4. Sign a message and verify it on the server
```

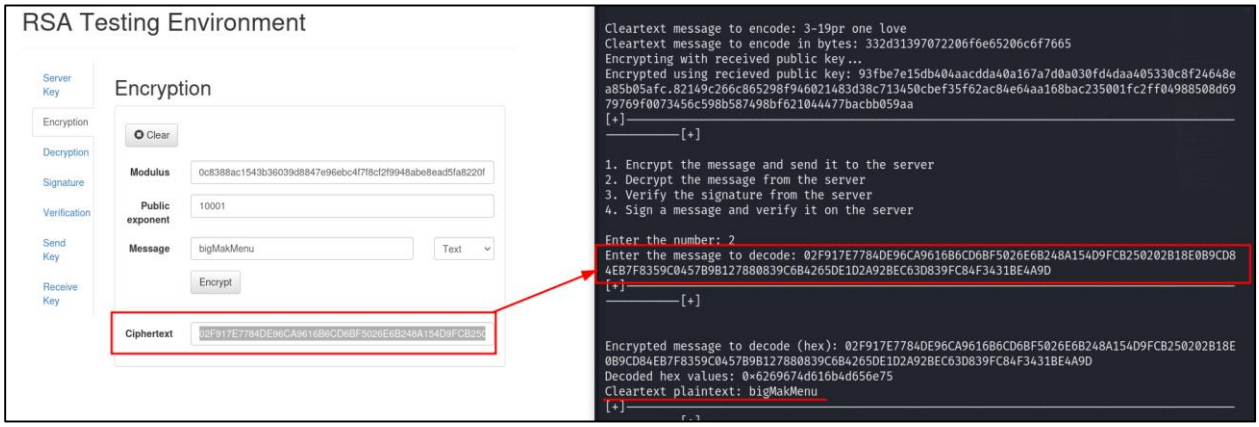
Наприклад, зашифруємо повідомлення публічним ключем сервера, та спробуємо його на сервері розшифрувати:

```
1. Encrypt the message and send it to the server
2. Decrypt the message from the server
3. Verify the signature from the server
4. Sign a message and verify it on the server

Enter the number: 1
Enter the message to encode: 3-19pr one love
[+]————————————————————————————————————————————————————————————————————————————————
——————————[+]


Cleartext message to encode: 3-19pr one love
Cleartext message to encode in bytes: 332d31397072206f6e65206c6f7665
Encrypting with received public key...
Encrypted using recieved public key: 93fbe7e15db404aacdda40a167a7d0a030fd4daa405330c8f24648e
a85b05afc.82149c266c865298f946021483d38c713450cbef35f62ac84e64aa168bac235001fc2ff04988508d69
79769f0073456c598b587498bf621044477bacbb059aa
[+]————————————————————————————————————————————————————————————————————————————————
——————————[+]
```

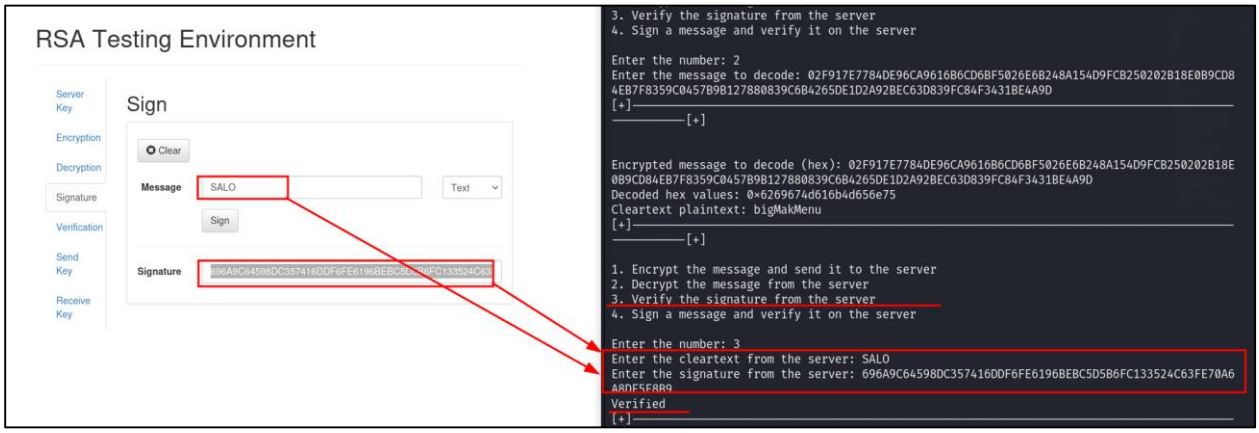


Оскільки сервер працює без сигнатур, надсилаємо лише першу частину зашифрованого повідомлення (до крапки, бо після вже йде сигнатура).

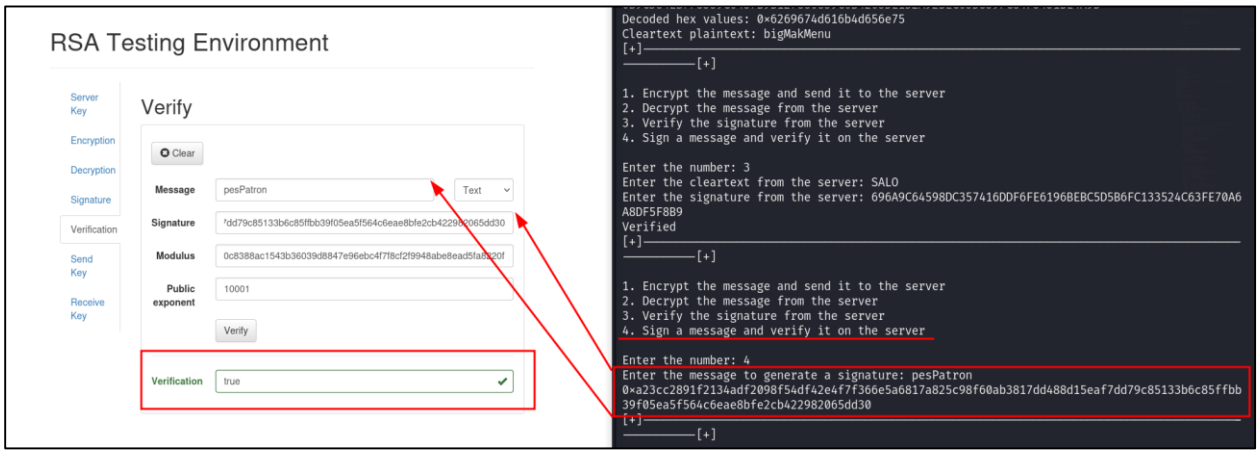
Спробуємо зашифрувати повідомлення на сервері, та дешифрувати на абоненті_А.



Тепер перевіримо цифровий підпис від сервера:



Ну і на останок створимо власний цифровий підпис, та перевіримо його на сервері:



Висновок: Ми були ознайомлені із різними тестами перевірок великих чисел на простоту, та реалізували тест Міллера-Рабіна. Завдяки цьому навчилися генерувати приватні та публічні ключи для роботи асиметричної криптосистеми RSA. Ознайомилися із поняттями цифрового підпису, та його підтвердження. І завдяки набутим знанням, реалізували модель роботи RSA на практиці.